
UNIT 1 DATABASE MANAGEMENT SYSTEM – AN INTRODUCTION

Structure

- 1.0 Introduction
- 1.1 Objectives
- 1.2 Need for a Database Management System
 - 1.2.1 The File Based System
 - 1.2.2 Limitations of File Based System
 - 1.2.3 The Database Approach
- 1.3 Logical DBMS Architecture
 - 1.3.1 Three Level Architecture of DBMS
 - 1.3.2 Mappings between levels and Data Independence
 - 1.3.3 The Need of Three Level Architecture
- 1.4 Physical DBMS Structure
 - 1.4.1 DML Precompiler
 - 1.4.2 DDL Compiler
 - 1.4.3 File Manager
 - 1.4.4 Database Manager
 - 1.4.5 Query Processor
 - 1.4.6 Database Administrator
 - 1.4.7 Data files, indices and Data Dictionary
- 1.5 Database System Architectures
- 1.6 Data Models and Trends
- 1.7 Summary
- 1.8 Solutions/Answers

1.0 INTRODUCTION

In the present time, most of your online activities require interaction with a database system running as the backend of an application, such as purchasing from supermarkets or e-commerce website, depositing and/or withdrawing from a bank, booking hotel, airline or railway reservation, accessing a computerised library, ordering a magazine subscription from a publisher, using your smart phone apps to purchase goods. In all the above cases a database is accessed. Most of these backend database systems may be called Traditional Database Applications. In these types of databases, the information stored and accessed is textual or numeric. However, with advances in technology in the past decades, different newer database models have been developed, which will be discussed in Block 4 of this course. In this unit, we will be introducing the architecture and structure of a traditional Database Management System.

1.1 OBJECTIVES

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

- describe the File Based system and its limitations;
- describe the structure of DBMS;
- define the functions of DBA;
- explain the three-tier architecture of DBMS and its need; and
- appreciate different database models.

1.2 NEED FOR A DATABASE MANAGEMENT SYSTEM

A Database is an organised, persistent collection of data of an organisation. The database management system manages the database of an enterprise. Prior to use of the database systems, file-based systems were popularly used. In order to appreciate the strengths of database management systems, you may first list the problems associated with the file base systems, which are discussed next.

1.2.1 File Based System

Computerised file-based systems were primarily designed to make electronic version of physical filing system used by the businesses. For example, a physical file can be set up to hold all the correspondence relating to a particular matter of a project, product, task, client or employee. In an organisation there could be many such files, which may be labeled and stored. Similarly, at homes you may have to maintain files relating to bank statements, loans, receipts, tax payments, etc. You can use electronic means to create these files. How do you search specific information from these files? For finding information, the entries could be searched sequentially. Alternatively, an indexing system could be used to locate the information directly, for example, you search for specific content in different files in your computer file search options.

The filing system works well when the number of items to be stored is small. It even works quite well when the number of items stored is quite large and they are only needed to be stored. However, a manual file system crashes when cross-referencing and processing of information in the files is carried out. For example, University has several students who can register for different programmes of the University. The University has several faculty members who teaches different courses to the enrolled students. The university may have to maintain separate files for the personal details of students, fees paid by them, the details of the courses undertaken by them. In addition, the University also needs files for keeping details of each faculty member in various departments, courses taught by them. How would be the following queries answered?

- Fee paid by the students of Computer Science Department per annum.
- The students who are willing to avail the pickup bus facility.
- Number of students taught by a faculty in a specific academic period.
- The number of students who have passed the programme this year in comparison to earlier years.
- How many students of a specific department have registered for courses of department other than their own department?

Please refer to *Figure 1*. The answer to all the questions cannot be computed by just simple statements but will require extensive file processing. Thus, each of these queries will require substantial amount of time in the file-based system.

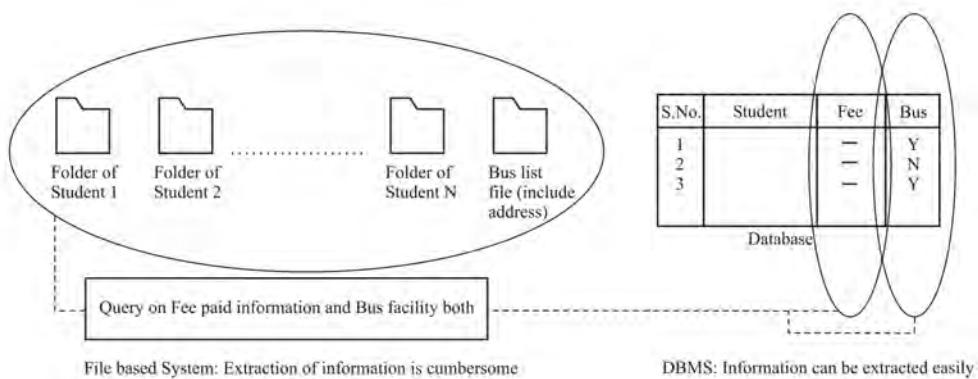


Figure 1: Information extraction using file-based system

1.2.2 Limitations of File Based System

File based systems required that several files should be opened for a particular system application. Several files may consists of duplicate data, which can result in several shortcomings. Some of these shortcomings are listed below

- Data isolation:** Since the file system store data in separate files, which may belong to different applications. These files are not accessible to other applications and are difficult to share, especially when an application needs to use more than one files. Also, as the number of files can be very large for such systems, therefore, it would be difficult to search the relevant data from these files
- Data Duplication:** As stated earlier, a file system has different files for different applications, which may have overlapping data requirements. This will result in duplication of data, which can result in inconsistent data when duplicate data is updated. In addition, data duplication can also result in waste of storage. An example of data duplication is shown in Figure 1, where the address of several students may be stored in two different files, viz. student folder and bus list file.
- Inconsistent Data:** The data in a file system can become inconsistent if more than one person modifies the data concurrently, for example, if any student changes the residence and the change is notified to only his/her file and not to the bus list. Entering wrong data is also another reason for inconsistencies.
- Data dependence:** In the file systems, you need to clearly define the storage organization of data files and the structure of the records in the application code. This means that it is extremely difficult to make changes to the existing structure, as any change in structure would require change in all the programs using that structure of the data. The programmer would have to identify all the affected programs, modify them and retest them. This characteristic of the File Based system is called **program data dependence**.
- Incompatible File Formats:** Since the structure of the files is embedded in application programs, the structure is dependent on application programming languages. Hence the structure of a file generated by COBOL programming language may be quite different from a file generated by 'C' programming language. This incompatibility makes them difficult to process jointly. The application developer may have to develop software to convert the files to some common format for processing. However, this may be time consuming and expensive.
- Fixed Queries:** File based systems are very much dependent on application programs. Any query or report needed by the organization would require the

application programmer to write a new program. As the type and number of queries or reports are expected to increase with time, producing different types of queries or reports would be difficult to implement in file-based systems. Since applications of file-based systems are designed to answer specific queries, therefore, new queries cannot be answered without generating an application. This entire process is time consuming and complex.

The file-based systems require large number of applications, therefore, are difficult to maintain. Further, each application would require separate provision from security. Besides the above, the maintenance of the File Based System is difficult and there is no provision for security. Further, data recovery from failures is inadequate or non-existent.

1.2.3 The Database Approach

As discussed in the previous section, file system has many weaknesses. Therefore, a new approach was proposed that eliminates the weaknesses of file system. This approach, called the database approach, separated data from application programs. The data in this approach is integrated from various application and securely shared using a management system. *A database stores the integrated data of an organisation in a persistent manner.* The following are some of the characteristics of database approach:

- The database can store data in a centralised database or a distributed database.
- The data is managed by a database management system (DBMS)
- It contains additional data about data, called meta data, which describes the structure and constraints on the stored data. The meta data is stored in DBMS in a data dictionary or system catalog.
- Database integrates the data of an organization. This data is shared under the control of DBMS in a secure manner.
- DBMS allows several basic operations related to data, such as creating a database structure, inserting and editing data in a database, enforcing security and constraints on data and allowing access to data to authorised users.

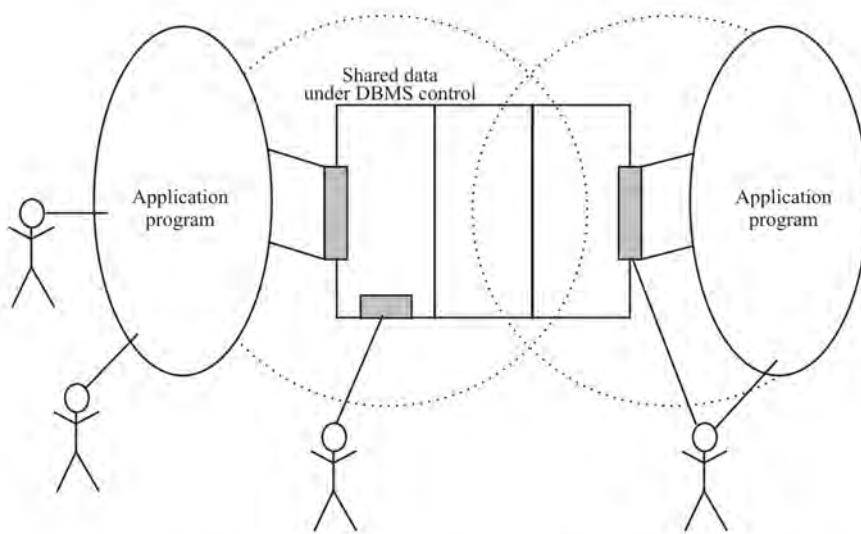
Let us discuss these advantages of database systems in more detail.

Reduction of Redundancies

The data of a database system is integrated, therefore, removes any duplicate data, as was the case of a file-based system, which maintains different files for different applications. The files that contain the copy of the data may become inconsistent as some of the files may be updated whereas others may not be. The reduction in duplicate data may also help in reducing such data inconsistencies among the duplicated data of files. Thus, in database approach data is stored at a single place or with controlled redundancy under the control of DBMS, which helps in deducing data inconsistency.

Sharing of Data

The data of a database is integrated data of the entire organisation, however, the authorised users may be permitted to share partial data (why?). This scheme can be best explained with the help of a logical diagram (*Figure 2*). New applications can be built and added to the current system and data can be shared with these data, as per the need of such applications. Please note that data sharing is controlled by the DBMS, such that only the authorised users or applications can access it.



A user can either access data window through DBMS or use an application program.

Figure 2: User interaction to data through DBMS

Data Independence

In the file-based system, the descriptions of data and logic for accessing the data are built into each application program making the program more dependent on data. A change in the structure of data may require alterations to programs. Database Management systems separates data descriptions from data. Hence it is not affected by changes. This is called Data Independence, where details of data are not exposed. DBMS provides an abstract view and hides details. For example, in Figure 2, you can observe that the interface or window to data provided by DBMS to a user may still be the same although the internal structure of the data is changed.

Improved Integrity

Data Integrity refers to validity and consistency of data. Data Integrity means that the data should be accurate and consistent. This is implemented by enforcing checks or constraints on the data, while it is being entered or manipulated in a database. Data of a database is not allowed to violate these constraints or rules. Constraints may apply to data items within a record or relationships between records. For example, a constraint on age of an employee can be between 18 and 70 years only. While entering or modifying the data of the age of an employee, the DBMS should enforce this constraint. However, please note there is still a possible error if you enter the age of a person as 55 instead of 25. Such errors would require different ways of checking. Database systems support many other types of constraints, which will be discussed in later units.

Efficient Data Access

DBMS utilises techniques to store and retrieve the data efficiently at least for unforeseen queries. An advanced DBMS allows its users to access data efficiently.

User Interfaces as per Users technical knowledge

A DBMS allows different types of interfaces, based on different levels of their technical knowledge. For example, the following types of interfaces may be provided by a DBMS:

- Menu driven forms and reports-based interfaces
- Application programming interface
- Query language interfaces
- Natural language interfaces

Representing relationship among data

Data of a database system is integrated data of an organization, which includes large number of related data objects. DBMS should maintain and preserve these relationships so that related data can be accessed easily.

Improved Security

The data of an organisation is vital and confidential. DBMS allows users to share only that information that they are authorised to access. For example, the critical information of an employee of an organisation should not be accessible to any other employee of that organization. Hence, data of the database should be protected from unauthorised users. This is implemented by Database Administrator (DBA), who provides the users with controlled privileges on the data and type of operations. To enforce security, DBMS has a security and authorisation subsystem. Only authorised users may use the specific data of a database. Further, the operations performed by these users on data can be restricted to data retrieval, insertion, update, deletion etc. or any combination of these of these operations. For example, the Branch Manager of any company may have access to all data and is allowed to modify the incentive data of employees, whereas the Sales Assistant may not have access to salary details.

Improved Backup and Recovery

A file-based system may fail to provide measures to protect data from system failures, as taking backups periodically is the responsibility of the user. However, most of the DBMSs are designed in such a manner that it can recover from different types of hardware failures. In addition, a DBMS also support data backup. In general, aa DBMS has a subsystem to support such provisions.

Support for concurrent transactions

A transaction, in the context of a database system, is an atomic operation that is either completed fully or not at all. A database should allow multiple transactions to be carried out at the same time. For example, in a bank several money withdrawal and money transfer operations may be carried out at the same time. Most of the commercial DBMSs ensure that all these transactions do not interfere with each other.

Check Your Progress 1

- 1) What is a DBMS?

.....
.....

- 2) What are the advantages of a DBMS?

.....
.....

- 3) Compare and contrast the traditional File based system with Database approach.

.....
.....

1.3 THE LOGICAL DBMS ARCHITECTURE

As discussed in the previous section that most of the advantages of database systems are because of creation of a DBMS software. A DBMS must support many services and therefore are complex in nature. In addition, DBMS are also required to store, manipulate and control a very large amount of data in a reliable manner. In this and sub-sequent section, we discuss the architecture DBMS, which will help you about the processes and features of a DBMS.

In this section two different architectures, which deals with two different aspects of database management. The first architecture is the logical architecture, which defines

the data organization and access at different logical levels of a database. The second architecture defines various components of a DBMS software. This architecture is referred to as physical database architecture

1.3.1 Three Level Database Architecture

The three-level database architecture of a database defines the three different levels of abstraction of data for different types of users of database. The proposed architecture was designed and standardized by the American National Standards Institute (ANSI) and is also known as ANSI/SPARC architecture. As per this architecture, a database schema can be visualized at three different levels. Figure 3 shows these three levels of this architecture. These levels are explained next.

The External or View Level

This level of abstraction provides a view of data for the users of a database system. Typically, this abstraction is created based on access rights of the users. Different types of users can be allowed different external view of data, as shown in Figure 3. Users can have different view of data. This level hides the overall structure of a database system from external users. From the database designer's point of view, this level also provides information on the mapping of external records of external views to the conceptual record of conceptual view of data.

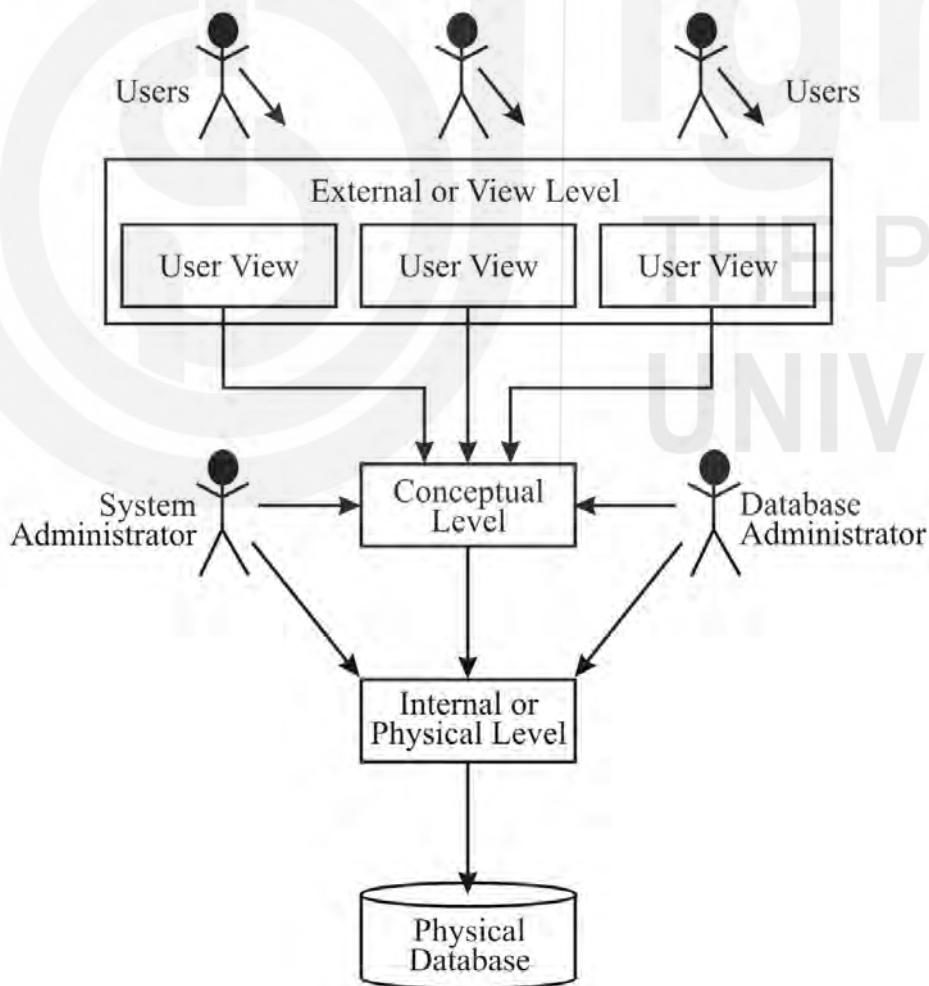


Figure 3: Logical DBMS Architecture

The Conceptual Level

The purpose of conceptual level is to define the structure, relationships and constraints of a database system. Therefore, the conceptual level includes the structure of the data objects, relationships among those objects and the constraints on the data objects and

access control of objects. A standard language called Data Definition Language (DDL) is often used in DBMSs to define the conceptual level or schema of a database system. The conceptual schema can be defined by the database administrator or the system administrator, as shown in Figure 3.

The Internal or Physical Level

A database system consists of many files, which include the data files, the meta data files, the access structure files, etc. The data files contains the actual data. The meta-data files contains structure related information of data files, relationship details among data objects, constraints on data items etc. The access structure files defines the control related information of the data objects. These files can be controlled by the DBMS software for access and updates. DDL can be used to describe the internal or physical schema too. In addition, this level can store additional files, which are used for enhancing the performance of database system. These files though are stored on an operating system, however, the access to these file is also under the control of database system administrator.

1.3.2 Mappings between the three Levels and its relationship with Data Independence

The three level architecture defines a single database system across all the three levels. Therefore, different levels must map with each other. This mapping leads to the concept of data independence, which was one of the major weakness in file systems. This concept is explained next.

The first mapping is between the conceptual level and the external level. the external level is derived from the conceptual level. it is a part of conceptual level, however, please note that these parts must be related else the database will lose database integrity. The advantage of this mapping is that an external user only need to see the external level any change in the conceptual level will be hidden from the user. For example, at conceptual level you may keep information about name of a person using the data items like *title*, *firstname* and *lastname*. At the external level you may just map it to a data item *name* field. Thus, there exists a mapping, which will map:

name = *title* || *firstname* || *lastname* (|| is a concatenation operation)

Suppose, at a later point you decide to add additional data item *middlename* in the conceptual level, then you just need to change your mapping and not the programs which are based on the external level. The mapping would be changed to:

name = *title* || *firstname* || *middlename* || *lastname*

Thus, the conceptual to external mapping may isolate the external users from the changes into conceptual level. These changes can be hidden in the conceptual to internal mapping. This is also referred to as the logical data independence.

The second mapping is between the physical level and the conceptual level. This mapping insulates conceptual level from changes in file organisation, indexes etc, without changing the conceptual level. In general, such changes may be performed to enhance the performance of a database management system. For example, instead of organising the student file on enrolment number, which is the primary key, you may organize the student file on Programme + Student name. This may enhance the access time of data for many important queries to the database. The conceptual level in this case still remains unchanged. This is also referred to as physical data independence.

1.3.3 Objective of the three-level architecture

The three level architecture separates the data that is presented to the user from the data that is stored in the database physically. The basic objectives of three level architecture are:

- It can support different view for different users. In case of change in any application, the views related to that application are required to change. Thus, three-level architecture ensures the independence of data and programs.
- As stated above, due to independence of data and application programs, the applications need not deal with physical file organization. Therefore, the application programs and users of a database are provided with higher level of abstraction of data. Thus, a user or application programs are not required to deal with conceptual schema and the physical schema of a database. In general, the Database Administrator (DBA) is responsible for creating and modifying the conceptual schema and physical schema using DDL.

1.4 PHYSICAL DBMS ARCHITECTURE

A Data Base Management System (DBMS) is a complex software, as it manages many aspects of database, such as query languages, data integrity, data security, access to files, performance of data access. Figure 4 shows some of the important software components of a DBMS, which should be present logically in most of the DBMS of today.

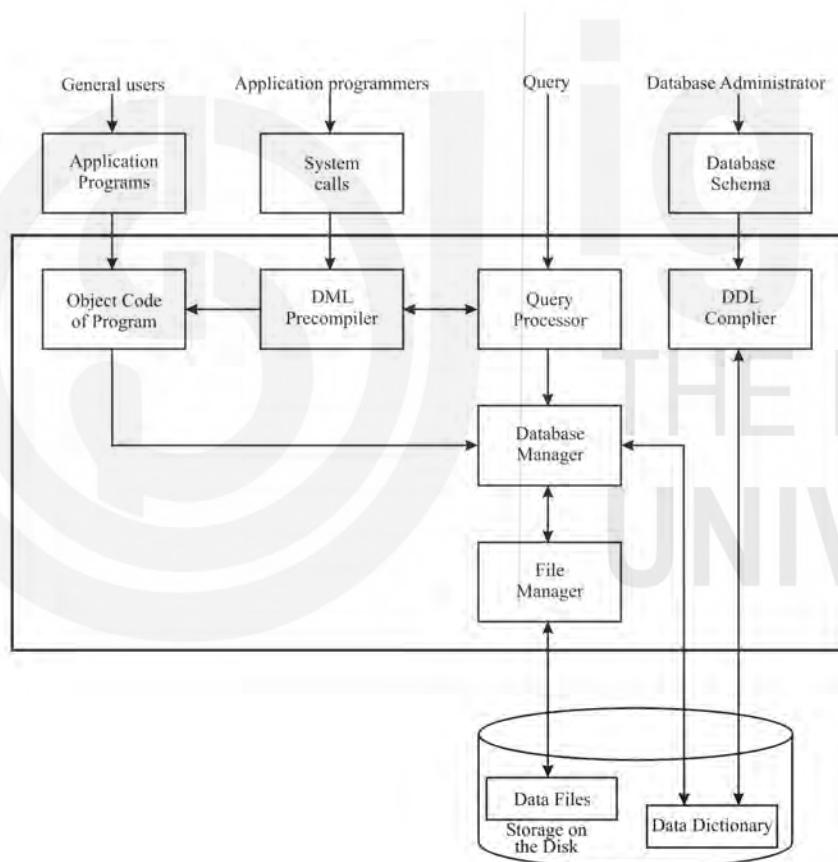


Figure 4: DBMS Structure

The software components of DBMS, as shown in Figure 4 are discussed next.

1.4.1 DML Precompiler

A DBMS consists of several languages, which are used for defining the data and for manipulation of data. For data definition, as stated earlier a language called Data Definition Language (DDL) is used. Using DDL, you can define the structure of the data objects, data type of different data elements, integrity constraints, storage constraints and access rights on the data. In addition, every DBMS consist of a Data Manipulation Language (DML), which is used for data insertion, modification, data access etc. The purpose of DML precompiler is to translate the DML commands, which are written as part of application programs into related procedure calls, so that

they can be executed to perform the desired operation. In addition, this component of the DBMS also coordinates the translation of queries, which are input to the query processor component.

1.4.2 DDL Compiler

A DDL compiler is used to compile a DDL statement into the related tables, which include the meta data, constraints, access rights etc. The meta data tables are stored in the data dictionary or system catalog. This meta data is used for providing information about the tables to other components of DBMS. In addition, the meta data is used for protecting data and enforcing data integrity.

1.4.3 File Manager

All the tables and metadata of the database are physically stored as files under the control of file sub-system of the operating system. The file sub-system of the operating system has generic features, which may not be sufficient for a DBMS file. The File Manager component of the DBMS keeps track of all the data files, their index files and all other related information files of a database system. The final input/output to the database files are performed by the operating system.

1.4.4 Database Manager

Database Manager is one of the most important components of the DBMS. The role of database manager is to accept the requests of data access or data manipulation by the user queries and application programs and performs the relevant action on the data taking the help of data dictionary and file manager. Database manager protects the data from unauthorised access and manipulation. It also enforces integrity constraints. You may please note that a database contains a large amount of data. Database manager is also responsible to decide, which data should be brought from secondary storage to main memory and back. Database manager allows simultaneous transactions on data and is also responsible for the recovery of database in case of failures. It is also responsible for enhancing the performance of a database system access. The role of database manager are as follows:

- **Collaborating with file manager for file handling:** As explained earlier, the file sub-system of operating system is responsible for storing or manipulating data and related files of a database system. The file manager component of the DBMS interacts with the operating system for this purpose. The database manager collaborates and controls the file manager for various file operations.
- **Integrity enforcement:** The data integrity relates to correctness of data in a database system. A DBMS forces several constraints on data, which allows only correct data to be stored in a database system. These constraints are stored in the data dictionary and the database manager uses data dictionary to enforce these integrity constraints. You will learn more about integrity constraints in the later units.
- **Security enforcement:** A DBMS does not allow unauthorized users to access the data of a database. This access control can relate to the entire data or a specific item of the data. For example, as an employee of an organization, you can read only your salary information. You do not have permission to change your salary. In addition, you cannot read or change the salary of any other employee. The security constraints can also be part of the data dictionary and implemented by the database manager.
- **Backup and recovery:** The purpose of backup and recovery component of data manager is to guard the database against the loss of data at the time of a failure of computer system. The failure of a computer system can occur due to many different reasons such as hardware failure communication failure software failure etc. The database manager ensures that data of various transactions is not corrupted even after a failure.
- **Concurrency control:** One of the major uses of database system is in transaction management systems, such as Online banking system. In such system multiple transaction access and modify database simultaneously. The database manager ensures that consistency of data is ensured even in the

presence of concurrent transactions. Transaction management is discussed in the block 3 in details.

To perform the roles or functions as discussed above, database manager has the following software components. These components are also shown in Figure 5.

- Authorisation control is used to verify the credentials of a user to ascertain if s/he is an authorised person to access or modify the data.
- Command Processor converts the DDL/DML or other programming language commands to executable sequence of code.
- Integrity checker checks if the data that is being inserted in a database or is being modified, follows all the specified constraints on the database.
- Query Optimizer tries to optimize the processing time of different queries.
- Transaction Manager is one of the important components, which checks if the user has right to perform a transaction.
- Scheduler component manages the consistency of data, while concurrent transactions are being processed by a database. The scheduler ensures an ordering of transaction execution, even though they may be conflicting with each other.
- Recovery Manager is responsible for recovery of a database system to a consistent state, in case of a failure.
- Buffer Manager optimizes the process of transfer of data between main memory and the disk, where database is stored. The buffer manager is also termed as the *cache manager*.

IGNOU
THE PEOPLE'S
UNIVERSITY

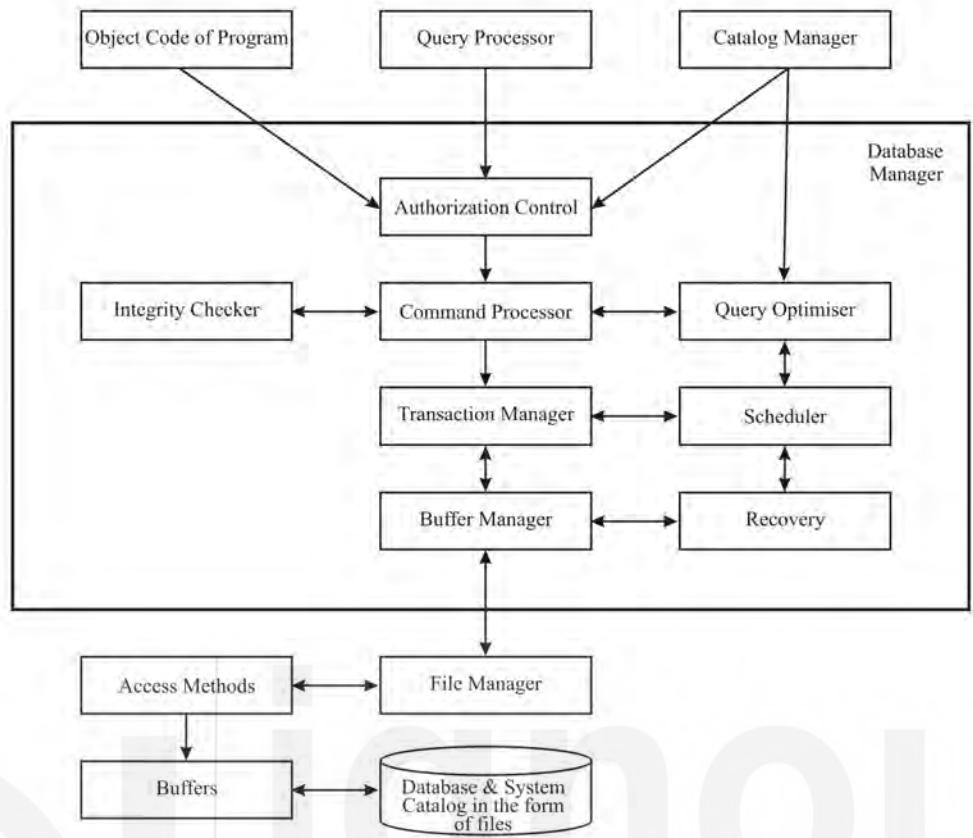


Figure 5: Components of Database Manager

1.4.5 Query Processor

Every DBMS consists of a query language which enables database creation, data manipulation, data retrieval and data control operations. This query language is usually a high level 4th generation languages and is translated and optimised, so that the required operation can be handled by the DBMS in a most optimal manner. In general, this query languages processes consist of two basic components - the first one parses the query language into the native database language and the second optimises the query and prepares the code for query execution. The parser checks the syntax of the query, converts the query to a sequence of tasks and sends these tasks to the query optimizer. The query optimizer is responsible for generating a set of alternative query evaluation plans. It then estimates the expected total query response time, which includes access time of data from the disks, the time to execute a query and the time of communication of data or results over the network, for each of these query evaluation plans. Finally, the best query evaluation plan is selected to execute the query efficiently.

1.4.6 Database Administrator

The role of database administrator (DBA) is to monitor control the database creation (at different levels of 3-level architecture and mapping between the levels), modification, access, query response time, security, recovery in case of failure etc. The functions of a DBA are given below:

- Defines the database Schema at different levels using DDL
- Specifies the organisation of file and any other database access methods including indexes using DDL
- Performs the changes in the schema definitions, file organization or indexes, whenever needed.
- Allows authorization of data access or modification to different users.
- Specifies the integrity constraint on the database.

1.4.7 Data files, Indices and Data Dictionary

Basic Concepts

The data is stored in the data files. Data files can be standard files or encrypted files. These files, in general, are not accessible directly to any system user. The indices are stored in the index files. Indices provide fast access to data items. For example, a book database may be organised in the order of accession number of books yet may be indexed on Author name and Book titles for allowing faster access for searches on these attributes.

Data Dictionary: The data of an organisation is a valuable corporate resource. Commercial databases have large number of entities, attributes, data interrelationships and other information about the data. A data dictionary/directory is designed to store meta data, which is data about the data in a database system. A good data dictionary allows efficient data access, as it can provide a road map to guide users to access information within a large database. An ideal data dictionary may include the following information about a database:

- Schema definitions of internal, conceptual and external schemas, which includes:
 - the definition of every object or table
 - the attributes and their data types including primary key and attribute constraints.
 - any relationships between different attributes including foreign key constraints etc.
 - authorisation of access or modification at the level of attributes, if any.
 - Authorisations at the level of objects/tables, if any.
- Users and application programs of external schema and their permissions
- Database statistics such as number of records in every object/table, different number of occurrences of a data in an attribute etc.

+ Check Your Progress 2

1) What are the major components of Database Manager?

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

2) Explain the functions of the person who has the control of both data and programs accessing that data.

.....
.....

1.5 DATABASE SYSTEM ARCHITECTURES

In the present time several database system architectures are popular.

First and earliest architecture of database system was having a centralized database system. These systems contained database in a single machine, which was either used by a single user or were shared by several users. A single user application in the present time may be an address book of your mobile device that is being used by a single user. However, today most of the centralized database system use multiple core processor having large memory and multiple disks, called the database servers. A large number of users are connected to these servers, mostly through remote connections. These systems fall under the category of client-server system. At its most basic level, this client server database architecture can be broken down into two parts: the *back end* and the *front end*.

The back end are the servers, which are responsible for managing the physical database and providing the necessary support and mappings for the internal, conceptual, and external levels. Other functions of a DBMS, such as security, integrity and access control, are also the responsibility of the back end.

The front end is just any application that runs on the DBMS. These may be applications provided by the DBMS vendor, the user, or a third party. The user interacts with the front end. A front-end may be user friendly interface provided by an application developed for a database system access and modifications. These applications may be developed by the vendors themselves or by software developers using an Application Programming Interface (API) or third-party applications.

In this context, many different types of interfaces and utilities are offered in support of commercial database management systems. Some of these are:

- **Interfaces:**

- **Command line Interface:** This interface existed since start of DBMS. You can use this interface to write DDL and DML and other permitted commands. These interfaces allow very rich set of commands and are useful for expert users like DBA.
- **Graphical User Interface:** Such interfaces are developed to allow users to interact through database applications using menu driven interfaces. Such interfaces have become more useful over the last decade.
- **Utilities for Backup/Restore:** The purpose of these utilities is to ensure that that the current state of a database is duly backed up on secondary storage, so that it can be recovered or restored in the case of any failure or even disaster. The backup is done periodically.
- **Utilities for Load/Unload of data:** Life of a database system is quite long. The size of a database system keeps increasing if the old data is not deleted. Also, hardware wears down over a period. Therefore, a database may be required to move from one machine to another machine. Sometimes the change in DBMS software or its versions also necessitates movement of data. The Load and Upload utilities are used for the purposes as above.
- **Utilities for Reporting or Analysis:** Reports forms the important component of a database systems, especially management information systems. The reports and analysis of data can be used at various levels of decision making in an organisation. These utilities analyse and report data in various visual forms for decision making.

1.6 DATA MODELS AND TRENDS

A database system required that the data should be structures in such a manner that it allows easier data access and manipulation operations. There structures are termed as database models. A database model has to address the following issues:

- It should define a logical structure of data, so that different attributes of data are easily identified
- It should be able to represent relationships between different data elements or objects
- It should support constraints on a data elements and data objects.

The following Table defines some of the Data Models:

Model Type	Examples
Conceptual Model: Uses entities, their attributes and relationships among entities	<i>Entity-Relationship Model:</i> This model identifies the physical or logical entities and their attributes. In addition, this model also identifies the relationships among these entities. It is mainly represented in graphical form using E-R diagrams. This is very useful in Database design. The entity relationship diagram is discussed in this Block.
Hierarchical Model: Uses data hierarchy	Hierarchical Model: It defines data as and relationships through hierarchy of data values. <i>Figure 7</i> shows an example of hierarchical model. These models are now not used in commercial DBMS products.
Record based Logical Models:	<p><i>Network Model:</i> Network model, as the name suggests, represent data about entities using a set of records. The relationships among these data records are represented using links. A simple network model example is explained in <i>Figure 6</i>. It shows a sample diagram for such a system. This model is a very good model as far as conceptual framework is concerned but is nowadays not used in database management systems.</p> <p><i>Relational Model:</i> It models data of both entities and relationships as tables. The relationship tables are related with entity tables using a set of constraints. This model is based on sound mathematical theory of relations. This is one of the widely used data model and will be discussed in more detail in the subsequent units of this course.</p>
Object-based Models: Use objects as key data representation components.	<i>Object-Oriented Model:</i> An object in object-oriented model contains the data of an entity. In addition, the object also allows a set of defined operations on the data stored in an object. Further, the relationships among the objects are implemented by creating links among these objects and by implementing message passing. Object-Models are useful for databases where data interrelationships are complex, for example, Computer Assisted Design based components. These are explained
Semi-structured Model: Uses user defined tags	<i>Semi-structured data Model:</i> Relational model is typically called a structured model, whereas, the semi-structured model uses user defined tags, which loosely force integrity contains. However, these models are very flexible in comparison to relational models. These models are popular in web-based data management, as they are very light database systems.
Graph-based Models	<i>Graph data Model:</i> Graph is an important data structure consisting of nodes and arcs. The node can represent an entity while the links can be used to represent a relationship. Some of these types of data models are used in NoSQL database management system discussed in Block 4.

Figure 6 shows the records of students and the result of the students in various courses taken by them. All the records of the students are shown as a single file. The student records can contain one or many pointers to the course file. Please notice in Figure 6 that the person named Ajay, whose enrolment number is 21026 has obtained 50 marks

in course having course code 101. Further, he has obtained 19 marks in course having course code 204. Thus, each record on course is linked to one record of the student.

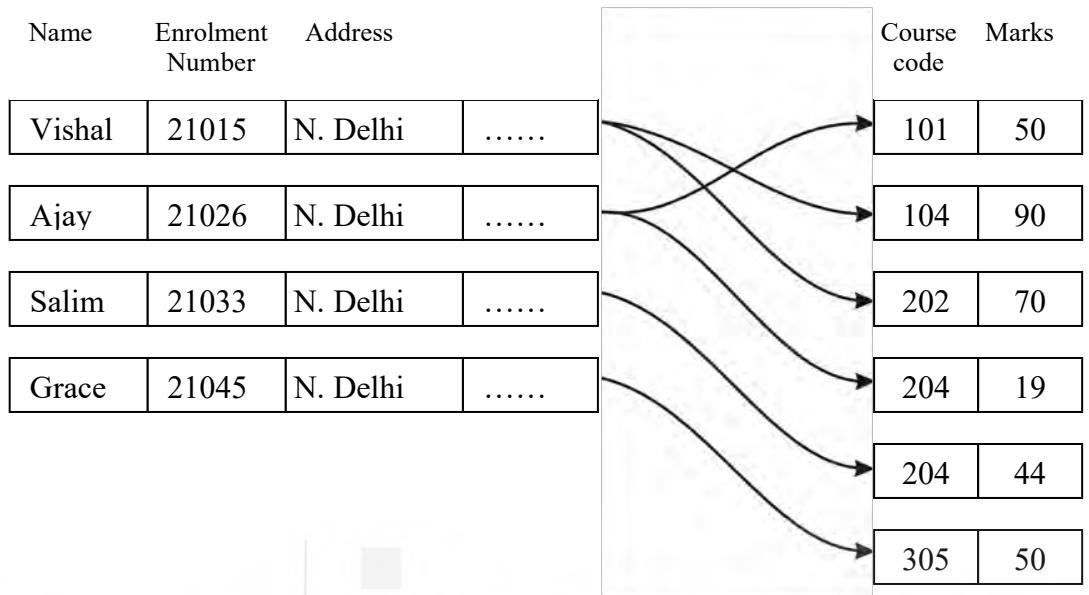


Figure 6: An example of Network Model

Figure 7 shows the data of Figure 6 in a hierarchical data model. Both these models are of historical nature. The relational model, object-oriented model and other model models are explained in the later blocks of this course.

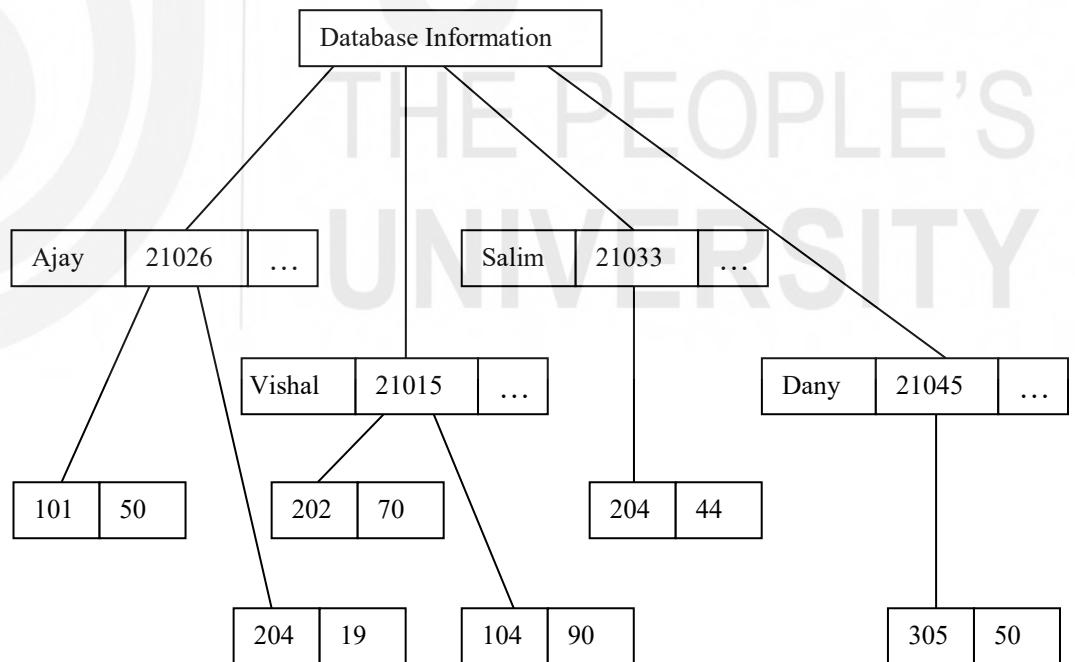


Figure 7: An example of Hierarchical Model

Relational Database management systems, which were supported by structured query language (SQL), became popular in the 1990s and later. These database systems were used to process transactions of a high-end web application. Such systems provided application users a graphical or menu driven interface using devices like computer, mobile devices and had features of user data input through keyboard, mouse, touch screens, forms, speech etc. With the growth of web applications lead to growth of semi-structured data. This led to use of eXtensible Markup Language (XML) and Java Script Object Notation (JSON) as the data exchange formats. Thus, several different database management systems, like geographical database management systems,

emerged that supported such data. With the rapid rate of growth of data newer database management systems like NoSQL databases started to grow. These databases lowered the time of application development but did not have strict control on consistency requirements. In the present time, the features of traditional and NoSQL database management systems are growing. Present day DBMSs are using cloud services for data storage and offer many database services at higher level of abstraction leading to lesser time for database application deployment.

+ Check your Progress 3

- 1) What is a database backend?
.....
- 2) What is the need of utilities in commercial database system? Explain any two such utilities.
.....
- 3) What is the difference between network and hierarchical models?
.....

1.7 SUMMARY

This unit provides you an introduction about the database system and database management system. It first defines the need of a database management system by comparing the database system approach to file system approach. The file system has several limitations due to data redundancy, however, database approach integrates and shares the data among several applications. The unit also discusses the basic advantages of database approach are - sharing of data, data independence, data integrity and security enforcement and transaction management for concurrent users.

The unit also explains the three-level architecture of database systems, which allows a database to be defined in terms of schemas at different levels for different types of users. Such a design allows access of relevant data to relevant users. In addition, the unit explains the physical architecture of DBMS and describes various components of DBMS. Next, the unit discusses the database system architecture for the commercial database and several data models. It also briefly presents the recent trends in DBMSs. You are advised to go through the further readings for more details these topics.

1.8 SOLUTIONS/ANSWERS

Check Your Progress 1

- 1) DBMS manages the data of an organisation. It allows facilities for defining, updating and retrieving data of an organisation in a sharable, secure and reliable way.
- 2) The advantages of DBMS are:
 - Reduces redundancies
 - Provides environment for data independence
 - Enforces data integrity
 - Data Security
 - Answers unforeseen queries
 - Provides support for transactions, recovery etc.

File Based System	Database Approach
Cheaper Data dependent Data redundancy Inconsistent Data Fixed Queries	Costly Data independent Controlled data redundancy Consistent Data Unforeseen queries can be answered

Check Your Progress 2

- 1) Integrity enforcement, control of file manager, security, backup, recovery, concurrency control, etc.
- 2) A database administrator is normally given such controls. His/her functions are: defining database, defining and optimising storage structures, and control of security, integrity and recovery.

Check Your Progress 3

1. Most of the database systems are deployed in a multi-user environment either as centralized or distributed system. A database server, which is at the backend, manages and controls the physical data. It also provides integrity, security and access control to multiple client's requests.
2. Utilities are special purpose software, which are provided to support additional features in support of a DBMS. Two such utilities are for backup/restore – to protect data against any failure and load/unload – to load data from one hardware/software combination to a different hardware/software combination.
3. A network model stores of basic entities as data records and points are used to represent the relationship between them. A hierarchical model uses almost similar model to represent data of entities but the relationship between them is implemented as a hierarchy.

UNIT 2 RELATIONAL DATABASE

Structure	Page Nos.
2.0 Introduction	
2.1 Objectives	
2.2 The Relational Model	
2.2.1 Domain, Attribute, Tuple and Relation	
2.2.2 Super keys Candidate keys and Primary keys for the Relations	
2.3 Relational Constraints	
2.3.1 Domain Constraint	
2.3.2 Key Constraint	
2.3.3 Integrity Constraint	
2.3.4 Update Operations and Dealing with Constraint Violations	
2.4 Relational Algebra	
2.4.1 Basic Set Operation	
2.4.2 Cartesian Product	
2.4.3 Relational Operations	
2.5 Summary	
2.6 Solution/Answers	

2.0 INTRODUCTION

In the first unit of this block, you have been provided with the details of the Database Management System, its advantages, structure, etc. This unit is an attempt to provide you information about relational model. The relational model is a widely used model for DBMS implementation. Most of the commercial DBMS products available in the industry are relational at core.

Relational model is based on theory of relations and was first proposed by E.F. Codd. In this unit we will discuss the terminology, operators and operations used in relational model.

2.1 OBJECTIVES

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

- describe relational model and its advantages;
- perform basic operations using relational algebra;
- identify the relational constraints in a database system;
- identify key of a relation.

2.2 THE RELATIONAL MODEL

A model of database system basically defines the *structure or organisation of data*, the *set of integrity constraints* on the data and a *set of operations* that can be performed on the data. The basic structure of data in a relational model in the form of two-dimensional tables. A table in this model is called a relation. Such organization of data simplifies enforcement of integrity constraints and database operations. The following table represents a simple relation:

PERSON_ID	NAME	AGE	ADDRESS
1	Sanjay Prasad	35	B-4,Modi Nagar, UP
2	Sharada Gupta	30	Pocket 2, Mayur Vihar, Delhi
3	Vibhu Datt	36	C-2, Saket, New Delhi

Figure 1: A Sample Person Relation

Following are some of the advantages of relational model:

- **Ease of use:** The simple tabular representation of database helps the user define and query the database conveniently. For example, you can easily find out the age of the person whose first name is “Vibhu”.
- **Flexibility:** Since the database is a collection of tables, new data can be added and deleted easily. Also, manipulation of data from various tables can be done easily using various basic operations. For example, you can add a telephone number field in the table at *Figure 1*.
- **Accuracy:** In relational databases the relational algebraic operations are used to manipulate database. These are mathematical operations and ensure accuracy (and less of ambiguity) as compared to other models. These operations are discussed in more detail in Section 2.4.

2.2.1 Tuple, Attribute, Domain and Relation

Before we discuss the relational model in more detail, let us first define some very basic terms used in this model.

Tuple: Each row in a table represents a record or information of a single object, which is also termed as a tuple. For example, Figure 1 has three records or tuples. A record/tuple consists of a number of attributes, which is defined next.

Attribute: A relation consists of large number of columns. Each of these columns, which defines a separate data value, is termed as an attribute. The column name in a relation is generally related to the meaning of data items of that column. For example, *Figure 2* represents a relation PERSON. The columns PERSON_ID, NAME, AGE, ADDRESS and TELEPHONE are the attributes of the relation PERSON and each row in the relation represents a separate tuple (record).

Relation Name: PERSON

PERSON_ID	NAME	AGE	ADDRESS	TELEPHONE
1	Sanjay Prasad	35	B-4,Modi Nagar, UP	011-25347527
2	Sharada Gupta	30	Pocket 2, Mayur Vihar, Delhi	023-12245678
3	Vibhu Datt	36	C-2, Saket, New Delhi	033-1601138

Figure 2: An extended PERSON relation

The relation of Figure 2 consists of 5 attributes, therefore, each tuple in this relation is called a 5-tuple. Thus, if a relation has n attributes, then each record in that relation would be termed as *n*-tuple.

Domain: A domain is a set of permissible values that can be accepted by a specific attribute. For example, in Figure 2, if you assume that there may be a maximum of 100 persons in the relation, then you may assign PERSON_ID to a domain of integer values, which should be in the range from 1 to 100 only. Once you have assigned this domain, then you will not be able to assign any value below 1 and above 100 to PERSON_ID. The domain of attribute AGE can be integer values between 0 and 150. The domain can be defined by assigning a type or a format or a range to an attribute. For example, a domain for a number 501 to 999 can be specified by having a 3-digit number format having a range of values between 501 and 999. Domains need not be contiguous numbers. For example, the enrolment number of IGNOU has the last digit as the check digit, thus the enrolment numbers are non-continuous.

Relation: Each table is a relation. A relation is defined using two basic aspects, viz. schema and an instance.

Relational and E-R Models

Relational Schema: A relational schema denoted as R, specifies the relation's name, the list of attributes of the relation and the domain of each attribute, which is denoted as R (A₁, A₂, ..., A_n). The relation R has n attributes, which is also called the degree of a relation.

For example, the relation schema of the relation PERSON (see Figure 1) can be defined as the follows:

PERSON (PERSON_ID: Integer, NAME: Character, AGE: Integer,
ADDRESS: Character)

Since the PERSON relation contains four attributes, so this relation is of degree 4.

Relation Instance or Relation State: A relation instance denoted as r is a collection of tuples for a given relational schema at a specific point of time.

A relation state r of the relation schema R (A₁, A₂, ..., A_n), also denoted by r(R) is a set of n-tuples

r = {t₁, t₂, ..., t_m} has m tuples
Where each n-tuple is an ordered list of n values
t = <v₁, v₂, ..., v_n>
where each v_i belongs to domain (A_i) or contains null values.

Let us elaborate the definitions above with the help of examples:

Example 1:

RELATION SCHEMA For STUDENT:

STUDENT (RollNo: string, name: string, login: string, age: integer)

RELATION INSTANCE

STUDENT				
	ROLLNO	NAME	LOGIN	AGE
t ₁	3467	Shikha	xyz@hotmail.com	21
t ₂	4677	Kanu	abc@gmail.com	20

Where t₁ = (3467, shikha, xyz@hotmail.com, 20) for this relation instance, the number of tuples m = 2 and each tuple contains n = 4 values.

Example 2:

RELATIONAL SCHEMA For PERSON:

PERSON (PERSON_ID: integer, NAME: string, AGE: integer, ADDRESS: string)

RELATION INSTANCE

In this instance, the number of tuples m = 3 and each tuple has n = 4 values.

PERSON_ID	NAME	AGE	ADDRESS
1	Sanjay Prasad	35	B-4, Modi Nagar, UP
2	Sharad Gupta	30	Pocket 2, Mayur Vihar, Delhi
3	Vibhu Datt	36	C-2, Saket, New Delhi

If a relation has proper integrity constraints, then each tuple of that relation would be valid tuple. It may be noted that ‘Null’ value can be assigned for some of the attributes where the values are unknown or missing. However, the Null’ value cannot be assigned to certain attributes, this will be explained in the later sections.

Ordering of tuples

In a relation, tuples are not inserted in any specific order. **Ordering of tuples is not defined as a part of a relation definition.** However, records may be organised later according to some attribute value in the storage system. For example, records in PERSON table may be organised according to PERSON_ID. Such data organisation depends on the requirement of the underlying database application. However, for the purpose of display, you may get these tuples displayed in the sorted order of age. The following table is sorted by age. Please note that this is sorted related is same as that or relation displayed in the order of PERSON_ID as each and every tuple is same. It is also worth mentioning that relational model **does not allow duplicate tuples.**

PERSON

PERSON_ID	NAME	AGE	ADDRESS
2	Sharad Gupta	30	Pocket 2, Mayur Vihar, Delhi
1	Sanjay Prasad	35	B-4, Modi Nagar, UP
3	Vibhu Datt	36	C-2, Saket, New Delhi

2.2.2 Super Keys, Candidate Keys and Primary Keys for the Relations

As discussed in the previous section, ordering of relations does not matter and all tuples in a relation are unique. However, can you uniquely identify a tuple in a relation? To answer this question, let us discuss the concept of keys in relations.

Super Keys

A **super key** is an attribute or set of attributes used to identify the records uniquely in a relation.

For Example, in the Relation PERSON described earlier PERSON_ID is a super key since PERSON_ID is unique for each tuple/record. Similarly (PERSON_ID, AGE) and (PERSON_ID, NAME) are also super keys of the relation PERSON since their combination is also unique for each tuple/record.

Candidate keys

Super keys of a relation may contain extra attributes. A candidate key is a minimal super key, which means that a candidate key does not contain any extraneous attribute. An attribute is called extraneous if even after removing it from the key, the remaining attributes still has the properties of a super key. A relation may have several candidate keys. A candidate key may contain one or many attributes of a relation.

The following properties must be satisfied by a candidate key:

- In any instance of a relation, the value of the candidate key attribute(s) should be unique.
- You cannot put a ‘Null’ value in any attribute that is a part of the candidate key. This rule is also termed as the entity integrity rule. Thus, a candidate key is unique and not null.
- A candidate key should have a minimal set of attributes.

- The value of a candidate key must be **stable**, which means it should not change frequently or its **value** change should not be outside the control of the system.

A relation can have more than one candidate keys and one of them can be chosen as a **primary key**.

For example, in the relation PERSON the two possible candidate keys are PERSON_ID and NAME (assuming unique names exists in the table). PERSON_ID may be chosen as the primary key.

Check Your Progress 1

Answer the following questions for the relational instance s of the following Supplier relation S:

S		
SNO	SNAME	CITY
S1	Smita	Delhi
S2	Jim	Pune

- What are the different attributes of relation S and how many tuples does S have?
.....
- What are the domains of the attributes of S?
.....
.....
- On sorting this relation on the field CITY, will the relation change? Will the order of tuples in the relation change?
.....
- List the super keys, all the possible candidate keys and the primary key of the relation
.....

2.3 RELATIONAL CONSTRAINTS

A relational database is collection of relations. Each relation consists of tuples, which consists of attributes. You can associate constraints, called the relational constraints, with the attributes. Such constraints can be of the following types::

- Domain Constraints
- Key Constraint
- Integrity Constraints

2.3.1 Domain Constraint

These constraints bind the possible values of attribute in a relation to a specific set of values, called the domain of the attribute. For example, an attribute COUNTRY may have domain of names of all the possible names of the countries. In commercial relational database management systems, the data type associated with an attribute defines the broad domain of an attribute. These domains include:

- Integral Data type like integer, long etc.
- Data types related to real numbers like float, double etc.
- Data types relating to alphabets like characters, strings etc.
- Boolean

Thus, domain constraint specifies the possible set of values that you want to put in an attribute of a relation. The values that appear in each attribute/column must be drawn from the domain associated with that column.

For example, consider the relation:

STUDENT

ROLLNO	NAME	LOGIN	AGE
3467	Shikha	xyz@hotmail.com	21
4677	Kanu	abc@gmail.com	20

In the relation above, AGE of the relation STUDENT always belongs to the integer domain within a specified range (say 0 representing just born to 150), and not to strings or any other domain. Within a domain non-atomic value should be avoided. This sometimes cannot be checked by domain constraints. For example, a database which has area code and phone numbers as two different fields will take phone numbers as-

Area_code	Phone
11	29534466

A non-atomic value in this case for a phone can be 1129534466, however, this value can be accepted by the Phone field only.

2.3.2 Key Constraint

This constraint states that the key attribute value in each tuple must be unique, i.e., no two tuples can contain the same value for the key attribute. This is because the value of the primary key is used to identify a unique tuple in a relation.

Example 3: If A is the key attribute in the following relation R, then A1, A2 and A3 must be unique.

R	
A	B
A1	B1
A3	B2
A2	B2

Example 4: In relation PERSON, PERSON_ID is the primary key so PERSON_ID cannot be assigned same for two different persons.

2.3.3 Integrity Constraint

There are two types of integrity constraints:

- Entity Integrity Constraint
- Referential Integrity Constraint

Entity Integrity Constraint:

It states that any attribute of a **primary key cannot take a Null value**. This is because the primary key is used to identify individual tuple in the relation. You will not be able to identify the records uniquely if they contain null values for the primary key attributes. This constraint is specified for each relation of a database.

Example 5: Let R be a relation; an instance r of R is given below. Is the instance r valid?

A#	B	C
Null	B1	C1
A1	B2	C2
Null	B2	C3
A2	B3	C3
Null	B1	C5

Note: A ‘#’ in the headings row is indicating that A is the Primary key of R.

In the instance r of relation R above, the primary key has Null values in the tuple t_1 , tuple t_3 and tuple t_5 . As per the entity integrity constraint, Null value in primary key is not permitted. Thus, relation instance r is an invalid instance.

Referential integrity constraint

For defining the referential integrity constraint, first we explain the concept of a **foreign key** and **foreign key constraint**.

Consider an attribute set A of a relation R, which references an attribute set B in a relation S, then A will be referred to foreign key in R provided it fulfills the following conditions:

1. B is the Primary key of S.
2. A and B are defined over the same domains. A is called the foreign key in relation R, which references B in relation S.
3. For every value x of attribute set A, in any instance r of R, there exist a tuple in any instance s of S, where the value of attribute set B is x . Please note that there will be only one such tuple having the value x , as B is the Primary key of S. This is called foreign key constraint.
4. Please note that the relation R is a referencing relation and S is called referenced relation.

A referential integrity constraint is more generic form of foreign key constraint, which requires that the attribute of the referencing relation must be present in the attribute of referenced relation in some tuple. Thus, foreign key constraint is a specific form of referential integrity constraint. Foreign key constraint is explained with the help of following example.

Example 6: Consider the two relational instance r of relation R and s of relation S.

Now answer the following questions:

- a) If C in R is foreign key of C in S, then is the following instances r and s valid?
- b) A new tuple having values (A6, B2, C4) is added in r. Does it violate foreign key constraint?

Instance r of R

A#	B	C [^]
A1	B1	C1
A2	B2	C2
A3	B3	C3
A4	B4	C3
A5	B1	C5

Instance s of S

E	C#
E1	C1
E2	C3
E3	C5
E2	C2

Notes:

- '#' identifies the Primary key of a relation.
 - '^' identifies the foreign key in a relation.
- a) In the example above, every value of C in r is matching with the values of C in s in some tuple. The r contains the values C1, C2, C3, C5 and s contains all these values. Thus, the instances r and s does not violate foreign key constraint.
 - b) If you try to insert the tuple (A6, B2, C4) in r, then it is violating the foreign key constraint for the present instances of relations, as there is no tuple in s, which contains value C4 in s.

2.3.4 Data Updating Operations and Constraint Violations

There are three basic data updating operations to be performed on relations:

- Insertion of tuples
- Deletion of tuples
- Update the value of attribute(s) in a relation

The INSERT Operation:

- To add new records in a database system, you use insert operation. For example, in the instance r of R of example 6, you may like to add a new record <'A5', 25, 'C6'>. Addition of a new record may cause constraint violation, which are explained below:
- Violation of Domain constraint: Such a violation occurs if the domain of a value, which you are trying to insert in an attribute, does not match the attribute's domain. e inserting a numeric value 25 into attribute B, whose domain is alphanumeric sting. This will cause a domain constraint violation.
- Violation of Key constraint: This violation occurs when you try to insert a key value, which already exists in some tuple of existing relational instance. For example, when you are trying to insert tuple <'A5', 25, 'C6'> in R, you are inserting a value A5 in the key attribute A. However, the value A5 already exists in attribute A of tuple t₅. Therefore, it is a violation of key constraint.
- Violation of Entity Integrity constraint: This violation will occur; in case you try to insert a Null value into the primary key attribute. For example, if you try to insert a tuple <'Null', 25, 'C6'> in R, you are violating this constraint.
- Violation of Referential Integrity constraint: Consider that while inserting a new tuple, the value that you are trying to insert in a foreign key attribute, which refers to an attribute in another relation where it is the primary key, does not match with any value of referred attribute of the referenced relation. (Please see example 6 par (b)). For example, when you are trying to insert tuple <'A5', 25, 'C6'> in R, you are inserting a value C6 in the foreign key attribute A of R. However, the value C6 is not a value in referred attribute C in referenced relation S. Thus, insertion of this tuple violates the referential integrity constraint.

Dealing with constraints violation during insertion:

If the *insertion* violates one or more constraints, then two options are available:

- Default option: - *Insertion can be rejected and the reason of rejection can also be explained to the user by DBMS.*
- Ask the user to correct the data, resubmit, also give *the reason for rejecting the insertion.*

Example 7:

Consider the Relation PERSON of Example 2:

PERSON

PERSON_ID	NAME	AGE	ADDRESS
1	Sanjay Prasad	35	B-4,Modi Nagar, UP
2	Sharad Gupta	30	Pocket 2, Mayur Vihar, Delhi
3	Vibhu Datt	36	C-2, Saket, New Delhi

(1) Insert <1, 'Vipin', 20, 'Mayur Vihar'> into PERSON

Violated constraint: - Key constraint

Reason: - Primary key 1 already exists in PERSON.

How to resolve: - DBMS could ask the user to provide valid PERSON_ID value and accept the insertion if valid PERSON_ID value is provided.

(2) Insert <'null', 'Anurag', 25, 'Patparganj'> into PERSON

Violated constraint: - Entity Integrity constraint

Reason: - Primary key is 'null'.

How to resolve: - DBMS could ask the user to provide valid PERSON_ID value and accept the insertion if valid PERSON_ID value is provided.

(3) Insert <'abc', 'Suman', 25, 'IP college'> into PERSON

Violated constraint: - Domain constraint

Reason: - value of PERSON_ID is given a string which is not valid.

How to resolve: - DBMS could ask the user to provide valid PERSON_ID value and accept the insertion if valid PERSON_ID value is provided.

(4) Insert <10, 'Anu', 25, 'Patpatganj'> into PERSON

Violated constraint: - None

Note: In the first 3 cases of constraint violations above DBMS will reject the insertion.

The Deletion Operation:

The delete operation is used to delete some existing records from a relation. To delete some specific records from the database a condition is specified based on which records are selected for deletion.

Constraints that can be violated during deletion

Only one type of constraints can be violated during deletion, it is referential integrity constraint. When you want to delete a record in a referenced relation, there is a possibility that you may delete a tuple whose attribute is being referenced by the referencing relation, where the attribute is the foreign key. Please go through the example 8 very carefully.

Dealing with Constraints Violation

If the *deletion* violates referential integrity constraint, then three options are available:

- Default option: - *Reject the deletion*. It is the job of the DBMS to explain to the user why the deletion was rejected.
- *Attempt to cascade (or propagate) the deletion* by deleting tuples whose foreign key references the tuple that is being deleted.
- Change the value of *referencing attribute* that causes the violation.

Example 8:

Let instance r of relation R be:

A#	B	C [^]
A1	B1	C1
A2	B3	C3
A3	B4	C3
A4	B1	C5

And instance q of relation Q be:

C#	D
C1	D1
C3	D2
C5	D3

Note:

- 1) '#' identifies the Primary key of a relation.
 - 2) '^' identifies the Foreign key of a relation.
- (1) Delete a tuple with C# = 'C1' in Q.
 Violated constraint: - Referential Integrity constraint is violated
 Reason: - If stated tuple is deleted from Q, the first tuple of R, which contains the Foreign key (C) value C1 will violate the foreign key constraint.
 How to resolve: - Options available are
- 1) Reject the deletion.
 - 2) DBMS may automatically delete all tuples from relation Q and S with C # = 'C1'. This is called cascade detection.
 - 3) The third option would result in putting NULL value in R where the value of attribute C is C1, which is the first tuple of R.

The Update Operations:

Update operations are used for modifying values of attributes in a database. The constraint violations faced by this operation are logically the same as the problem faced by Insertion Deletion Operations. You may define these actions yourself.

Check Your Progress 2

- 1 Consider the tables Suppliers, Parts, project and SPJ relation instances in the relations below.

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

P			
PNO	PNAME	COLOUR	CITY
P1	Nut	Red	Delhi
P2	Bolt	Blue	Pune
P3	Pen	White	Mumbai

J			SPJ			
<u>JNO</u>	<u>JNAME</u>	<u>CITY</u>	<u>SNO</u>	<u>PNO</u>	<u>JNO</u>	<u>QUANTITY</u>
J1	Sorter	Pune	S1	P1	J1	200
J2	Display	Mumbai	S2	P2	J2	700

Using the instance of relations as given above, answer the following questions:

- 1 List a domain constraint for S.

.....

- 2 List the Primary keys of all the relations and primary key constraint for the SPJ relation.

.....

- 3 List all the Foreign keys and Foreign key constraint.

.....

- 4 What would be the constraint violations if the following operations are performed:

- (a) Insert <Null, ‘Jack’, ‘Mumbai’> into S
- (b) Insert <‘S2’, Null, ‘J3’, 200> into SPJ
- (c) Insert <‘P2’, ‘Pencil’, ‘Grey’, ‘Kolkata’> into P
- (d) Insert <‘J3’, ‘Monitor’, ‘Jaipur’> into J

.....

.....

2.4 RELATIONAL ALGEBRA

Relational Algebra is a set of basic operations used to manipulate the data in relational model. These operations enable the user to specify basic retrieval requests. The result of retrieval is a new temporary relation, which is computed after performing these operations on one or two relations. Some of these operations can be classified in three categories:

- Relational Operations

These can be unary operations

 1. SELECTION
 2. PROJECTION

Or binary operations

 1. CARTESIAN PRODUCT
 2. JOIN
- Basic Set Operations
 1. UNION
 2. INTERSECTION
 3. SET DIFFERENCE
- Assignment Operation, rename and other operations

In this section, we will discuss the relational operation and basic set operations. You may refer to the other operations from the further readings.

2.4.1 Relational Operations

Relational algebra forms the basis of the query languages on a database system. Thus, by studying relational algebra you may be able to write better queries for a database system. Let us discuss these operations in detail.

Selection Operation:

Selection is a unary operator, that is used to choose only those tuples from a relation that fulfil a given criteria. It is represented using the mathematical symbol σ , as given below:

$$\sigma_{<\text{Selection condition}>} (\text{Relation})$$

You should specify a Boolean expression as a <Selection condition>. A selection condition uses comparison operators like $\{\leq, \geq, \neq, =, <, <\}$ and logical connectors like and (\wedge), or (\vee) and not (\neg).

Example 13:

Consider the relation PERSON. If you want to display details of persons having age less than or equal to 30, then the selection operation will be used as follows:

$$\sigma_{\text{AGE} \leq 30} (\text{PERSON})$$

The resultant relation will be as follows:

PERSON_ID	NAME	AGE	ADDRESS
2	Sharad Gupta	30	Pocket 2, Mayur Vihar, Delhi

Note:

- 1) Selection operation is commutative, i.e.,

$$\sigma_{<\text{condition1}>} (\sigma_{<\text{condition2}>} (R)) = \sigma_{<\text{condition2}>} (\sigma_{<\text{condition1}>} (R))$$

Hence, you can apply a sequence of Selection operations in any order

- 2) You can apply more than one conditions by using logical connectors.

Projection operation

The projection operation is used to select specific attributes from amongst all the attributes of records. This is denoted as Π .

$$\Pi_{\text{List of attributes for projection}} (\text{Relation})$$

Example 14:

Consider the relation PERSON. If you want to display only the names of persons, then the projection operation will be used as follows:

$$\Pi_{\text{NAME}} (\text{PERSON})$$

The resultant relation will be as follows:

NAME
Sanjay Prasad
Sharad Gupta
Vibhu Datt

Please note that for projection operation:

$$\Pi_{<\text{List1}>} (\Pi_{<\text{list2}>} (R)) = \Pi_{<\text{list1}>} (R) \text{ provided } <\text{list2}> \text{ contains attributes in } <\text{list1}>.$$

2.4.2 Cartesian Product

Given two relations R and S, the cartesian product of these two relations can be represented as $T = R \times S$. The cartesian product operation produces all possible combinations of the tuples of tuples of relations R and S. The cartesian product operation is commutative as well as associative. In addition, the degree of the resultant relation (T) is the sum of degree of relation R and relation S. The Cartesian product can be expressed using the following mathematical expression:

$T = \{t_1 \parallel t_2 \mid t_1 \in R1 \wedge t_2 \in R2\}$. Here, \parallel is a concatenation operator.

Example 12: Consider the following two relational instances of relation R1 and R2. What would be the cartesian product of the relations?

R1	R2
A	B
A1	B1
A1	B2
A2	B2
A2	B3

The Cartesian product of the relations is denoted by $R_3 = R_1 \times R_2$. The relation R_3 will be as shown below:

A	B	C
A1	B1	C1
A1	B1	C2
A1	B2	C1
A1	B2	C2
A2	B2	C1
A2	B2	C2
A2	B3	C1
A2	B3	C2

Please note that in cartesian product every tuple/record of R1 is concatenated with every record of R2. You can observe that the record $\langle A1, B1 \rangle$ is concatenated with both the records of R2, that is why in the output R3 you have tuples $\langle A1, B1, C1 \rangle$ and $\langle A1, B1, C2 \rangle$, likewise for all the tuples of R1. Please also note that R1 has 4 tuples and R2 has 2 tuples, therefore, R3 has $4 \times 2 = 8$ tuples.

The JOIN operation

JOIN is a binary operator. It combines two relations based on a given join condition. A JOIN operation is represented by mathematical symbol \bowtie . But how does JOIN operation will combine two relations? It would require that there should be at least one attribute in each of the relation, which have the same domain. Such attributes are called domain compatible attribute.

Syntax:

$R1 \bowtie_{\text{join condition}} R2$ is used to combine related tuples from two relations $R1$ and $R2$ into a single tuple.

<join condition> is of the form:

<condition>AND<condition>AND.....AND<condition>.

- Degree of Relation:

$\text{Degree}(\text{R1} \bowtie_{\text{join condition}} \text{R2}) \leq \text{Degree}(\text{R1}) + \text{Degree}(\text{R2})$.

- Three types of joins are there:

a) **Theta (θ) join**

When each condition is of the form $A \theta B$, where A is an attribute of R1 and B is an attribute of R2; both A and B have the same domain; and θ is one of the comparison operators $\{\leq, \geq, \neq, =, <, <\}$.

b) **Equijoin**

Equijoin is a restricted form of When each condition appears with equality operator (=) only.

c) **Natural join**

Natural join is defined as a specialised type of join, when the joining attributes in the two joining relations have the identical name, in addition to the identical domain and the condition for the join operation is equality (=) condition. In such cases only one joining attribute is kept in the result.

The following example explains the Natural join operation.

Example 15:

Consider the instance of the following relations. The primary key of STUDENT relation is ROLLNO and foreign key is COURSE_ID, which references the primary key COURSE_ID of COURSE relation.

STUDENT

ROLLNO	NAME	ADDRESS	COURSE_ID
100	Kanupriya	234, Saraswati Vihar.	CS1
101	Rajni Bala	120, Vasant Kunj	CS2
102	Arpita Gupta	75, SRM Apartments.	CS4

COURSE

COURSE_ID	COURSE_NAME	DURATION
CS1	MCA	3yrs
CS2	BCA	3yrs
CS3	M.Sc.	2yrs
CS4	B.Sc.	3yrs
CS5	MBA	2yrs

Display the name and other details of all the students along with their course details.

Solution: You may observe that the course details are existing in the COURSE relation and student names and other details are in the STUDENT relation. Therefore, you need to join these two relations to extract information. The join attributes in this case is COURSE_ID in STUDENT and COURSE_ID in the COURSE relation and equality operator is to be used; therefore, you use natural join operation, which can be represented as:

$$\text{STUDENTCOURSE} = \text{STUDENT} \bowtie \text{COURSE}$$

The output of this natural join will be the following relation. Please note that output has just one COURSE_ID attribute.

STUDENTCOURSE

ROLLNO	NAME	ADDRESS	COURSE_ID	COURSE_NAME	DURATION
100	Kanupriya	234, Saraswati Vihar.	CS1	MCA	3yrs

101	Rajni Bala	120, Vasant Kunj	CS2	BCA	3yrs
102	Arpita Gupta	75, SRM Apartments.	CS4	B.Sc.	3yrs

There are other types of joins like outer joins. You must refer to further reading for more details on those operations. They are also explained in later blocks of this course.

2.4.3 Basic Set Operation

The set operations are the binary operations, i.e., each is applied to two sets or relations, which should be union compatible. Two relations R (A_1, A_2, \dots, A_n) and S (B_1, B_2, \dots, B_n) are said to be union compatible if they have the *same degree n* and domains of the corresponding attributes are also the same i.e. **Domain (A_i) = Domain (B_i) for $1 \leq i \leq n$** .

Union

If R1 and R2 are two union compatible relations, then $R3 = R1 \cup R2$ is the relation containing tuples that are either in R1 or in R2 or in both. In other words, R3 will have tuples such that $R3 = \{t \mid t \in R1 \vee t \in R2\}$.

Example 9:

R1

A	B
A1	B1
A2	B2
A3	B3
A4	B4

R2

X	Y
A1	B1
A7	B7
A2	B2
A4	B4

$$R3 = R1 \cup R2 \text{ is}$$

R3

A	B
A1	B1
A2	B2
A3	B3
A4	B4
A7	B7

Note: 1) Union is a commutative operation, i.e,

$$R \cup S = S \cup R.$$

2) Union is an associative operation, i.e.

$$R \cup (S \cup T) = (R \cup S) \cup T.$$

Intersection

If R1 and R2 are two union compatible functions or relations, then the result of $R3 = R1 \cap R2$ is the relation that includes all tuples that are in both the relations

In other words, R3 will have tuples such that $R3 = \{t \mid t \in R1 \wedge t \in R2\}$.

Example 10:

R1

X	Y
A1	B1
A7	B7
A2	B2
A4	B4

R2

A	B
A1	B1
A2	B2
A3	B3
A4	B4

$$R3 = R1 \cap R2 \text{ is}$$

A	B
A1	B1
A2	B2
A4	B4

Note: 1) Intersection is a commutative operation, i.e.,

$$R1 \cap R2 = R2 \cap R1.$$

2) Intersection is an associative operation, i.e.,

$$R1 \cap (R2 \cap R3) = (R1 \cap R2) \cap R3$$

Set Difference

If R1 and R2 are two union compatible relations, then result of $R3 = R1 - R2$ is the relation that includes only those tuples that are in R1 but not in R2. In other words, R3 will have tuples such that $R3 = \{t \mid t \in R1 \wedge t \notin R2\}$.

Example 11:

R1

X	Y
A1	B1
A7	B7
A2	B2
A4	B4

R2

A	B
A1	B1
A2	B2
A3	B3
A4	B4

$$R1 - R2 =$$

A	B
A7	B7

$$R2 - R1 =$$

A	B
A3	B3

Note: -1) Difference operation is not commutative, i.e.,

$$R1 - R2 \neq R2 - R1$$

2) Difference operation is not associative, i. e.,

$$R1 - (R2 - R3) \neq (R1 - R2) - R3$$

Please note that a relational query language would be relationally complete, if it supports five relational algebraic operators – Selection, Projection, Union, Set Difference and Cartesian Product.

+ Check Your Progress 1

- 1) A database system is fully relational if it supports a language as powerful as _____.
- 2) Primitive operations are union, difference, product, selection and projection. The $A \cap B$ can be computed using as.....
- 3) Which of the following is not a traditional set operator in relational algebra?
 - a. Union
 - b. Intersection
 - c. Difference
 - d. Join
- 4) Consider the relational instances of the relations Suppliers, Parts, Project and SPJ relations given below. (Underline represents a key attribute. The SPJ relation has three Foreign keys: SNO, PNO and JNO.)

S		
<u>SNO</u>	SNAME	CITY
S1	Smita	Delhi
S2	Jim	Pune
S3	Ballav	Pune
S4	Seema	Delhi
S5	Salim	Agra

P			
<u>PNO</u>	PNAME	COLOUR	CITY
P1	Nut	Red	Delhi
P2	Bolt	Blue	Pune
P3	Part1	White	Mumbai
P4	Part2	Blue	Delhi
P5	Camera	Brown	Pune
P6	Part3	Grey	Delhi

J		
<u>JNO</u>	JNAME	CITY
J1	Sorter	Pune
J2	Display	Bombay
J3	OCR	Agra
J4	Console	Agra
J5	RAID	Delhi
J6	EDP	Udaipur
J7	Tape	Delhi

SPJ			
<u>SNO</u>	<u>PNO</u>	<u>JNO</u>	QUANTITY
S1	P1	J1	200
S1	P1	J4	700
S2	P3	J2	400
S2	P2	J7	200
S2	P3	J3	500
S3	P3	J5	400
S3	P4	J3	500
S3	P5	J3	600
S3	P6	J4	800
S4	P6	J2	900
S4	P6	J1	100
S4	P5	J7	200
S5	P5	J5	300
S5	P4	J6	400

Using the in the relations above, which of the following operations and constraints would be valid:

- i. UPDATE Project J7 in J setting CITY to Nagpur.
- ii. UPDATE part P5 in P, setting PNO to P4.
- iii. UPDATE supplier S5 in S, setting SNO to S8, if the relevant update rule is RESTRICT.
- iv. DELETE supplier S3 in S, if the relevant rule is CASCADE.
- v. DELETE part P2 in P, if the relevant delete rule is RESTRICT.
- vi. DELETE project J4 in J, if the relevant delete rule is CASCADE.
- vii. UPDATE shipment S1-P1-J1 in SPJ, setting SNO to S2. (shipment S1-P1-J1 means that in SPJ table the value for attributes SNO, PNO and JNO are S1, P1 and J1 respectively)

- viii. UPDATE shipment S5-P5-J5 in SPJ, setting JNO to J7.
 - ix. UPDATE shipment S5-P5-J5 in SPJ, setting JNO to J8
 - x. INSERT shipment S5-P6-J7 in SPJ.
 - xi. INSERT shipment S4-P7-J6 in SPJ
 - xii. INSERT shipment S1-P2-jjj (where jjj stands for a default project number).
-
.....
.....

- 5) Find the name of projects in the relations above, to which supplier S1 has supplied using relational algebra.
-
.....
.....

2.5 SUMMARY

This unit is an attempt to provide a detailed viewpoint of data base design. The topics covered in this unit include the relational model including the representation of relations, operations such as set type operators and relational operators on relations. The E-R model explained in this unit covers the basic aspects of E-R modeling. E-R modeling is quite common to database design, therefore, you must attempt as many problems as possible from the further reading. E-R diagram has also been extended. However, that is beyond the scope of this unit. You may refer to further readings for more details on E-R diagrams.

2.6 SOLUTIONS/ ANSWERS

Check Your Progress 1

1. Supplier Number (SNO), Supplier Name (SNAME) and City of the location of the supplier (CITY). The present instance s of the relation S has 2 tuples.
2. Domain of SNO is the codes that are being assigned to suppliers. The present coding use first character as S followed by the sequence number of supplier. The domain of SNAME is strings of characters of some defined length and the domain of CITY is the set of possible cities in India.
3. The relation will not change; the order of tuples will change.
4. The super key of the relation could be (SNO, SNAME, CITY) or (SNO, SNAME) or (SNO, CITY) or assuming every supplier has unique name (SNAME, CITY) or SNO or assuming every supplier has unique name SNAME.
The possible candidate keys are SNO or assuming every supplier has unique name SNAME
Primary key may be selected as SNO and alternate key would be SNAME

Check Your Progress 2

1. Domain constraint for S will be for CITY which should be checked to be in a selected list of cities of India. SNO may be checked by a Range constraint and SNAME may be data type with maximum allowable length of name.

2. The Primary key of relation S, P and J are SNO, PNO and JNO respectively. The primary key of SPJ is a composite key (SNO, PNO, JNO). As per the primary key constraint none of the attributes of primary key in SPJ, viz. SNO, PNO or JNO can be null.
3. There are three foreign keys, all of them are in SPJ relation. These are (i) SNO (ii) PNO (iii) JNO. SNO of SPJ references attribute SNO in S; PNO of SPJ references attribute PNO in P; and JNO of SPJ references attribute JNO in J. As per foreign key constraints the values of SNO, PNO and JNO in SPJ can be only as per the related referenced attributes.
4. The violations are as under:
 - a. Primary key of S is SNO, it cannot be Null, you may change it to <S4, Jack, Mumbai>, which will be inserted successfully in S
 - b. Primary key violation in SPJ as PNO cannot be Null as it is a part of primary key. There is another violation here, foreign key JNO has a value J3, which does not exist in JNO attribute of relation J for this instance, thus, the allowable values for JNO in SPJ is just J1 or J2. Thus, a correct record insertion in SPJ would be: <'S2', 'P3', 'J2', 200>
 - c. The primary key 'P2' already exists and cannot be duplicated, the correct insertion may be to Insert <'P4', 'Pencil', 'Grey', 'Kolkata'> into P
 - d. There is no violation.

Check Your Progress 3

1. relational algebra
2. A – (A – B)
3. (d) Join
4.
 - i. Accepted
 - ii. Rejected (candidate key uniqueness violation)
 - iii. Rejected (violates RESTRICTED update rule, as SPJ contains tuples having value S5 in SNO)
 - iv. Accepted (supplier S3 and all shipments for supplier S3 in the relation SPJ would be deleted, as the rule is CASCADE)
 - v. Rejected (violates RESTRICTED delete rule, as SPJ contains tuples having value P2 in PNO)
 - vi. Accepted (project J4 and all shipments for project J4 from the relation SPJ are deleted)
 - vii. Accepted
 - viii. Rejected (primary/candidate key uniqueness violation as tuple S5-P5-J7 already exists in relation SPJ)
 - ix. Rejected (referential integrity violation as there exists no tuple for J8 in relation J)
 - x. Accepted
 - xi. Rejected (referential integrity violation as there exists no tuple for P7 in relation P)
 - xii. Rejected (referential integrity violation – the default project number jjj does not exist in relation J).
- 5) The answer to this query will require the use of the relational algebraic operations. This can be found by selection supplies made by S1 in SPJ, then taking projection of resultant on JNO and joining the resultant to J relation. Let us show steps:

First find out the supplies made by supplier S1 by selecting those tuples from SPJ where SNO is S1. The relation operator being:

$$SPJT = \sigma_{SNO = 'S1'} (SPJ)$$

The resulting temporary relation SPJT will be:

<u>SNO</u>	<u>PNO</u>	<u>JNO</u>	<u>QUANTITY</u>
S1	P1	J1	200
S1	P1	J4	700

Next, you may take projection of SPJT on PNO

$$SPJT2 = \Pi_{JNO} (SPJT)$$

The resulting temporary relation SPJT will be:

<u>JNO</u>
J1
J4

Next, take natural JOIN this table with J:

$$RESULT = SPJT2 \bowtie J$$

The resulting relation RESULT will be:

<u>JNO</u>	<u>JNAME</u>	<u>CITY</u>
J1	Sorter	Pune
J4	Console	Agra

You can write it as a single relational algebraic query as:

$$RESULT = (\Pi_{JNO} (\sigma_{SNO = 'S1'} (SPJ))) \bowtie J$$

UNIT 3 ENTITY RELATIONSHIP MODEL

- 3.0 Introduction
- 3.1 Objective
- 3.2 Entity-Relationship (E-R) Model
 - 3.2.1 Entities
 - 3.2.2 Attributes
 - 3.2.3 Relationships
 - 3.2.4 E-R diagram Basics
 - 3.2.5 More about Relationships
 - 3.2.6 Extended E-R Features
 - 3.2.7 Defining Relationship for College Database
- 3.3 An Example
- 3.4 Conversion of E-R Diagram to Relational Database
- 3.5 Enhanced E-R Model
- 3.6 Converting E-R and EER diagram to Relations
- 3.7 Summary
- 3.8 Solution/Answers

3.0 INTRODUCTION

In the previous unit of this block, you have gone through the concept of relational database management systems and one of the important languages for relational database – relational algebra. This unit explains, you about of an analysis model of database system, known as Entity-Relationship (E-R) model. The E-R model is a widely used model for analysis of data requirements of an organization. E-R model is primarily a semantic model and is very useful in creating raw database design that can be further normalised. With the availability of object-oriented technologies, the E-R model has been extended to include the object-oriented features. This unit also discusses these E-R extensions. We will also discuss the conversion of E-R diagrams to tables, in this unit.

3.1 OBJECTIVES

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

- Define and explain various components of E-R model
- draw an E-R diagram for a given problem;
- convert an E-R diagram to a relational database;
- Explain the role of extended features of E-R model;
- Convert an EER diagram to relations.

3.2 ENTITY RELATIONSHIP (E-R) MODEL

Some of the important characteristics of E-R model are listed below:

- *Entity relationship model is a high-level conceptual data model.*
- *It allows you to describe the data involved in a real-world enterprise in terms of entities and their relationships.*
- *It is widely used to create an initial design of a database.*
- *It provides a set of useful concepts that make it convenient for a developer to move from a basic set of information to a detailed and precise description of information that can be easily implemented in a database system.*
- *It describes data as a collection of entities, relationships and attributes.*

In the following sections, we explain the basic terms used in E-R model.

3.2.1 Entities

First let us answer the question: **What are entities?**

- An entity is an object of concern, which is used to represent the things in the real world, e.g., car, table, book, etc.
- An entity need not be a physical entity, it can also represent a concept in real world, e.g., project, loan, etc.
- It represents a class of things, not any one instance, e.g., ‘STUDENT’ entity has instances of ‘Ramesh’ and ‘Mohan’.

Entity Set or Entity Type: A collection of a similar kind of entities is called an Entity Set or entity type.

For the COLLEGE database, the objects of concern are Student, Faculty, Course and Department. The collections of all the students’ entities form an entity set STUDENT. Similarly, collection of all the courses form an entity set COURSE.

You may please note that entity sets need not be disjoint. For example – an entity, say Mohan, may be part of the entity set STUDENT, the entity set FACULTY and the entity set PERSON.

Entity identifier - key attributes: An entity set usually has one or more attributes, which attains unique value for every distinct entity in a given entity set. Such an attribute or set of attributes is/are called key attribute(s) and its values can be used to identify each entity uniquely in the given entity set.

Strong entity set: An entity set which contains at least one key attribute is a Strong entity set. For example, a Student entity set would contain at least one key attribute *Enrolment number*, which is unique for every student, thus, the entity set Student is a strong entity set.

Weak entity set: Entity sets that do not contain any key attribute, and hence cannot be identified independently, are called weak entity sets. A weak entity cannot be identified uniquely by its attributes, therefore, are recognised in conjunction with the

primary key attributes of another strong entity on which its existence is dependent (called owner entity set).

Generally, a primary key of owner entity set is attached to a weak entity set, which has identifying attributes, called discriminator attributes. These two together forms the primary key of the weak entity set. The following restrictions must hold for the above:

- The owner entity set and the weak entity set must participate in one to many relationship set. This relationship set is called the identifying relationship set of the weak entity set.
- The weak entity set must have total participation in the identifying relationship.

One of the most common examples about the weak entity set is an entity set Dependent and the related Strong entity set Employee in an organisation. The Dependent entity set is used to list all the dependents of each employee of an organisation. The attributes of Dependent entity set are: Dependent name, birth date, gender and relationship with the employee. Each Employee entity is said to own the dependent entities that are related to it. However, please note that the ‘Dependent’ entity does not exist of its own, it is dependent on the Employee entity. In other words, you can say that in case an employee leaves the organization, all dependents related to him/her also leave along with this employee. Thus, a ‘Dependent’ entity has no significance without the entity ‘Employee’. Thus, it is a weak entity.

3.2.2 Attributes

Let us first define - **What is an attribute?**

An attribute is an element of an entity, which can contain a representative value. In other words, an entity is represented by a set of attributes.

For example, Student entity set may consist of attributes - Roll no, student’s name, age, address, course, etc. An entity will have a value for each of its attributes. For example, for a particular student the following values can be assigned:

Roll No:	1234
Name:	Mohan
Age:	18
Address:	Z-894, Maidan Garhi, Delhi.
Course:	B.Sc. (H)

Domains: Each simple attribute of an entity type contains a possible set of values that can be attached to it. This is called the domain of an attribute. An attribute cannot contain a value outside this domain.

EXAMPLE- for STUDENT entity Age has a specific domain, integer values say from 15 to 90.

Types of attributes

Attributes attached to an entity can be of various types. They are explained below:

Simple: An attribute that cannot be further divided into smaller parts and represents the basic meaning is called a simple attribute. For example: Each of the attributes - 'FirstName', 'LastName', age of PERSON entity set are simple attributes.

Composite: Attributes that can be further divided into smaller units and each smaller unit contains specific meaning. For example, the attribute NAME of an FACULTY entity can be sub-divided into First name, Last name and Middle name.

Single valued: Attributes having a single value for a particular entity. For Example, Age is a single valued attribute of a STUDENT entity.

Multivalued: Attributes that have more than one values for a particular entity is called a multivalued attribute. Different entities may have different number of values for these kinds of attributes. For multivalued attributes you must also specify the minimum and maximum number of values that can be attached. For example, phone number for a PERSON entity is a multivalued attribute.

Stored and derived: Attributes that are directly stored in the database are called stored attributes. For example, 'Birth date' attribute of a PERSON entity can be a stored attribute. However, there are certain attribute, whose value can be computed from the value of stored attribute. For example, in the PERSON entity the attribute 'Birth Date' can be used to compute the attribute Age of a person on a specific day. Thus, 'Birth Date' is a stored attribute, whereas Age may be a derived attribute for this entity.

3.2.3 Relationships

First, let us define the term relationships, i.e. **What Are Relationships?**

A relationship can be defined as:

- a connection or set of associations, or
- a rule for communication among entities:

Example: In COLLEGE database, the association between student and course entity set, i.e., in the statement "Student **opts** Course" **opts** is an example of a relationship between the two entities Student and Course.

Relationship sets

A relationship set is a set of relationships of the same type. For example, consider the relationship between two entity sets STUDENT and COURSE. Collection of all the instances of relationship **opts** forms a relationship set.

3.2.4 E-R Diagram Basics

The logical structure of a database is modelled using an E-R model, which is graphically represented with the help of an E-R diagram. The basic symbols using in an E-R diagrams are given in the following table:

Object	Represented by
Entity Set	Rectangle
Attribute	Ellipse
Relationship Set	Diamonds

Figure 3.1 shows different kinds of entities, attributes, relationships and participation constraints, which are explained in this Unit.

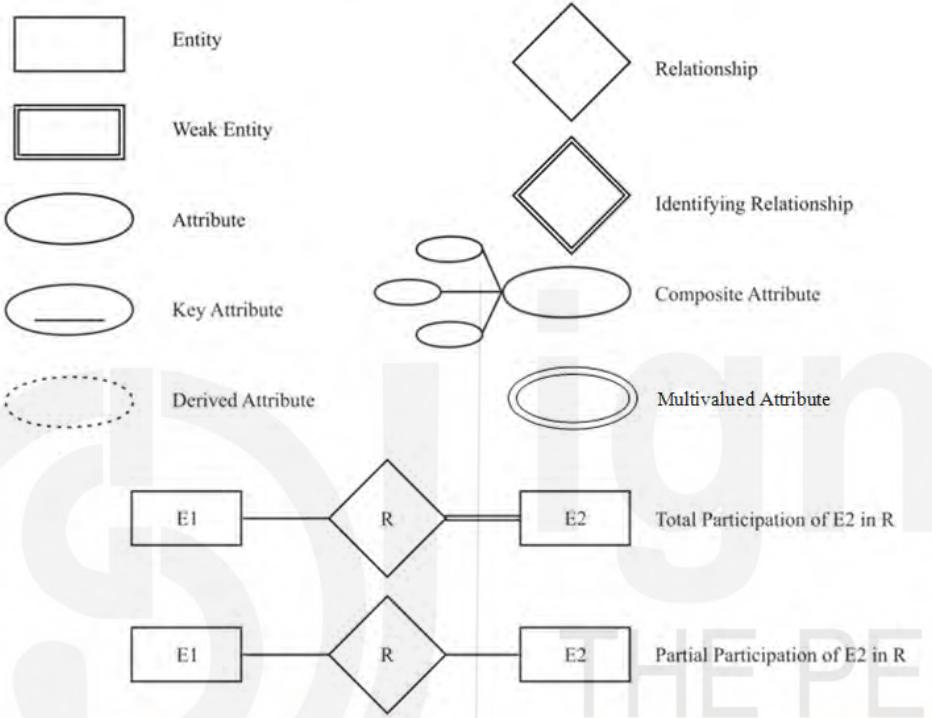


Figure 3.1: Symbols of E-R diagrams

3.2.5 More about Relationships

In this section, you will go through some of the important concepts, which are used for making good E-R models.

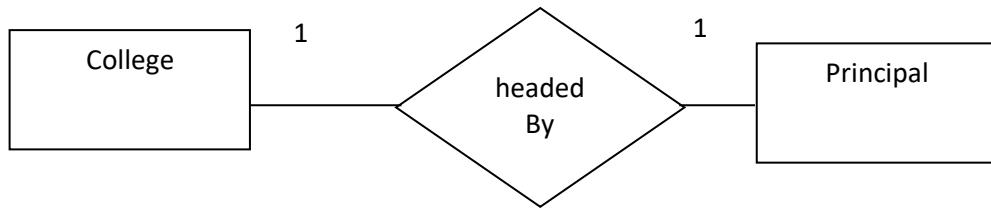
Degree of a relationship set: The degree of a relationship set is the number of participating entity sets. The relationship between two entities is called binary relationship. A relationship among three entities is called ternary relationship. Similarly, relationship among n entities is called n-ary relationship.

Cardinality of a relationship set: Cardinality specifies the number of instances of an entity associated with another entity participating in a relationship. Based on the cardinality binary relationship can be further classified into the following categories:

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

For example, relationship *headedBy* between college entity set and principal entity set would be one-to-one, as one college can have at the most one principal and one

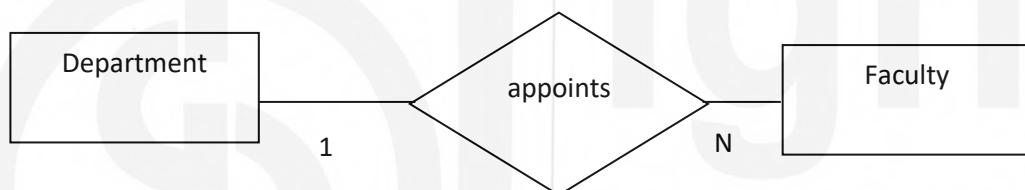
principal can be principal of only one college. (This example assumes that a principal can be head of only one college.)



Similarly, you can define the relationship between University and Vice-Chancellor.

- **One-to-many:** Consider entity sets A and entity set B has a relationship cardinality A : B, as 1:N. This suggests that one specific entity of A may be related with several entities of entity set B.

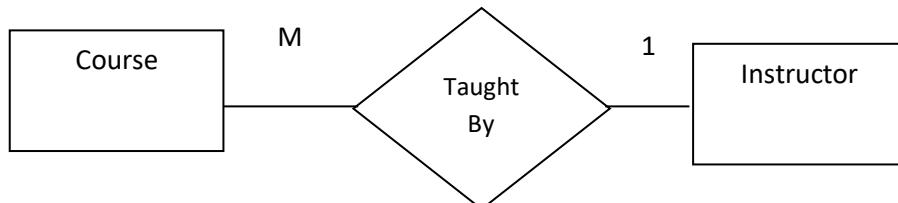
For example, relationship between Department entity set and Faculty entity set (assuming that a faculty member can work in only one department).



For example, in the diagram above, several faculty members may be appointed in one department, however, a specific faculty member will be appointed in a precisely one department.

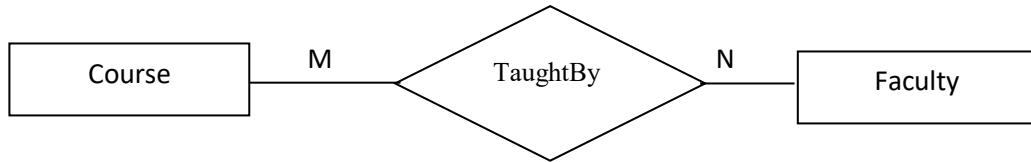
- **Many-to-one:** Consider entity sets A and entity set B has a relationship cardinality A : B, as N : 1. This suggests that several entity of A may be related with one specific entity of entity set B.

For example, relationship between entity set Course and entity set Instructor. An instructor can teach various courses, but a single course can be taught only by one instructor. Please note this is an assumption.



Many-to-many: Entities in entity set A and entity set B are associated with any number of entities from each other.

For example, consider that a course can be taught jointly by many faculty members and each faculty member can teach several courses, then many-to-many relationship holds, as shown below:



Another example is shown in diagram given below. The relationship cardinality M : N. This implies that an Author entity can be correlated to many Book entities, which are writer by him/her. Further, a Book entity can also be correlated with several Author entities, who have written the Book.



Recursive relationships: A recursive relationship is that relationships in which both the participating entity sets are the same entity set, however, the role of the entity set in each participation is different.

Participation constraints: The participation Constraints specify whether the existence of an entity depends on its being related to another entity via the relationship type. There are two types of participation constraints:

Total: When all the entities from an entity set participate in a relationship set, it is termed as the *total* participation. For example, the participation of the entity set Course in the relationship set 'TaughtBy' can be said to be *total* because every Course must be taught by a faculty.

Partial: When it is not necessary for all the entities from an entity set to participate in a relationship set, it is termed as *partial* participation. For example, the participation of the entity set Instructor in the relationship set 'TaughtBy' can be partial, since not every Instructor may be teaching a course.

3.2.6 Extended E-R Features

Although, the basic features of E-R diagrams are sufficient to design many database situations. However, with more complex relations and advanced database applications, it is required to use extended features of E-R models. The three such features are:

- Generalisation
- Specialisation, and
- Aggregation

We have explained them with the help of an example. More detail on them are available in the further readings.

Example 1: A bank has an Account entity set. Any accounts of the bank can be one of two types: (1) Savings account and (2) Current account

The statement as above represents a specialisation/ generalisation hierarchy. It can be shown as:

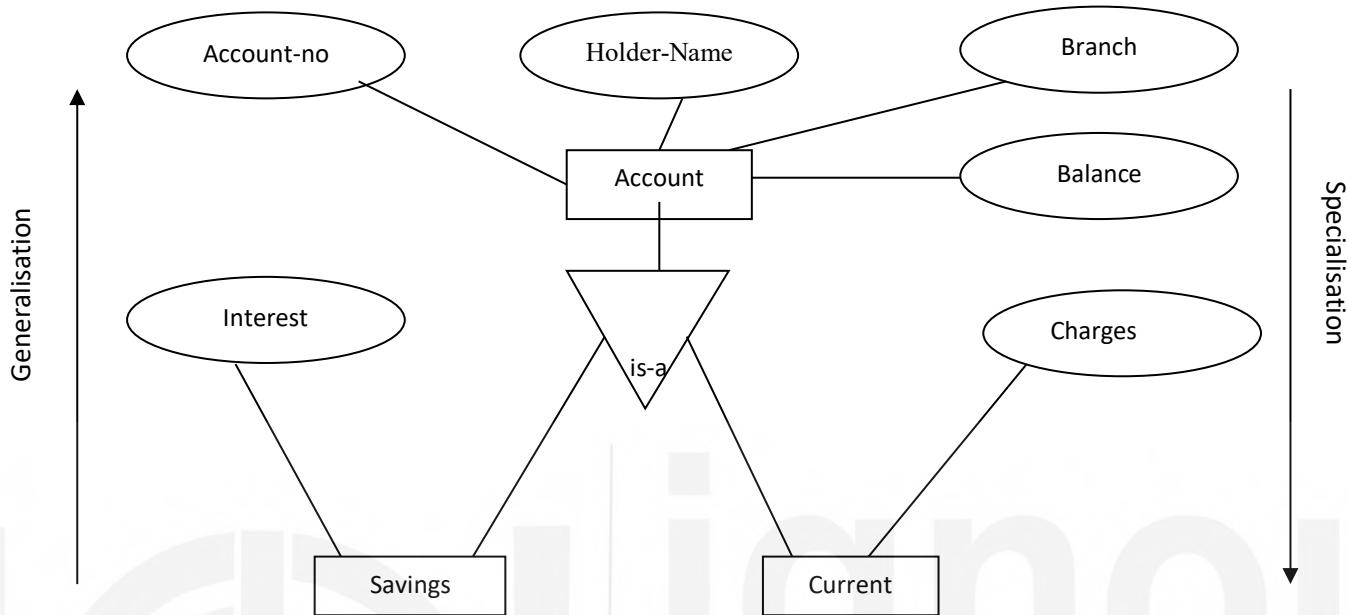


Figure 3.2: Generalisation and Specialisation hierarchy

Aggregation: One limitation of the E-R diagram is that they do not allow representation of relationships among relationships. In such a case the relationship along with its entities are promoted (aggregated to form an aggregate entity which can be used for expressing the required relationships). A detailed discussion on aggregation is beyond the scope of this unit you can refer to the further readings for more detail.

3.3 AN EXAMPLE

Let us explain it with the help of an example application. We will describe here an example database application of a COLLEGE database and use it for illustrating various E-R modeling concepts.

Problem Statement

A college database keeps track of Students, faculty, Departments and Courses. Following paragraphs gives the description of a COLLEGE database system.

A College contains various departments like Department of English, Department of Hindi, Department of Computer Science etc. Each department is assigned a unique id and name. Some faculty members are appointed to each department and one of them works as head of the department.

There are various courses conducted by each department. Each course is assigned a unique id, name and duration.

The information contained for various objects are stated below:

- Faculty information contains name, address, department, basic salary etc. A faculty member is assigned to only one department but can teach courses of another department.
- Student's information contains Roll no (unique), Name, Address etc. A student can opt only for one course and one department only.
- Student's Parent or Guardian information to be recorded is - name of parent or guardian, age of the parent or guardian, gender and address of parent or guardian.

A student is allowed to take only one course. A student is assisted by his/her guardian. A faculty works in a department, which is headed by a faculty member. A course is taught by a faculty, who is allowed to teach several courses.

Defining Entities and Attributes

One of the simplest ways to identify the entities is to look for the proper nouns. This gives us possible set of entities in the problem statement are:

Student, Faculty, Department, Course, Guardian

The attributes of these entities can be found from the statements. You may also note except of Guardian:

Student (Rollno – Primary Key, Name, Address)
Faculty (Id – Primary Key, Name, Address, Basic_Sal)
Department (D_No, - Primary Key, D_Name)
Course (Course_ID – primary key, Course_Name, Duration)
Guardian (Name, Address, Relationship)

Please note that Guardian is a weak entity. It is related to strong entity Student.

Defining Relationship

Using the concepts defined earlier, we have identified that strong entities in COLLEGE database are Student, Faculty, Course and Department. This database also has one weak entity called Guardian. You can specify the following relationships among these entities. Further, these relationships also show the relationship cardinality and participation constraints:

1. **Head_of** is a 1:1 relationship between Faculty and Department. Participation of the entity Faculty is *partial* since not all the faculty members participate in this relationship (as all cannot be the head), while the participation from Department side is *total* since every department has one head. Please also note that this relationship has an attribute Date_from, which stores the date from which the given appointment was applicable.
2. **Works_in** is a 1:N relationship between Department And Faculty, as one department can have many faculty, whereas a faculty is associated with only one department. Participation from both the side is total, as every department has at least one faculty and every faculty must belong to a department.
3. **Opts** is a 1:N relationship between Course and Student. Participation from Student side is *total* because we are assuming that each student opts for one

course. But the participation from the course side is *partial*, since there can be courses that no student.

4. **Taught_by** is a M: N relationship between Faculty and Course, as a Faculty can teach many courses and a Course can be taught by many faculty members.
5. **Enrolled** is a 1:N relationship between Student and Department as a student is allowed to enroll for only one department at a time. A student must enroll in a Department. However, a newly created Department may not have a student.
6. **Assisted_by** is a 1:N relationship between Student and Guardian as a student can have more than one local guardian and one local guardian is assumed to be related to one student only. The weak entity Guardian has *total* participation in the relation “Assisted_by”.

Now, you are ready to make an E-R diagram for the college database. The E-R diagram.

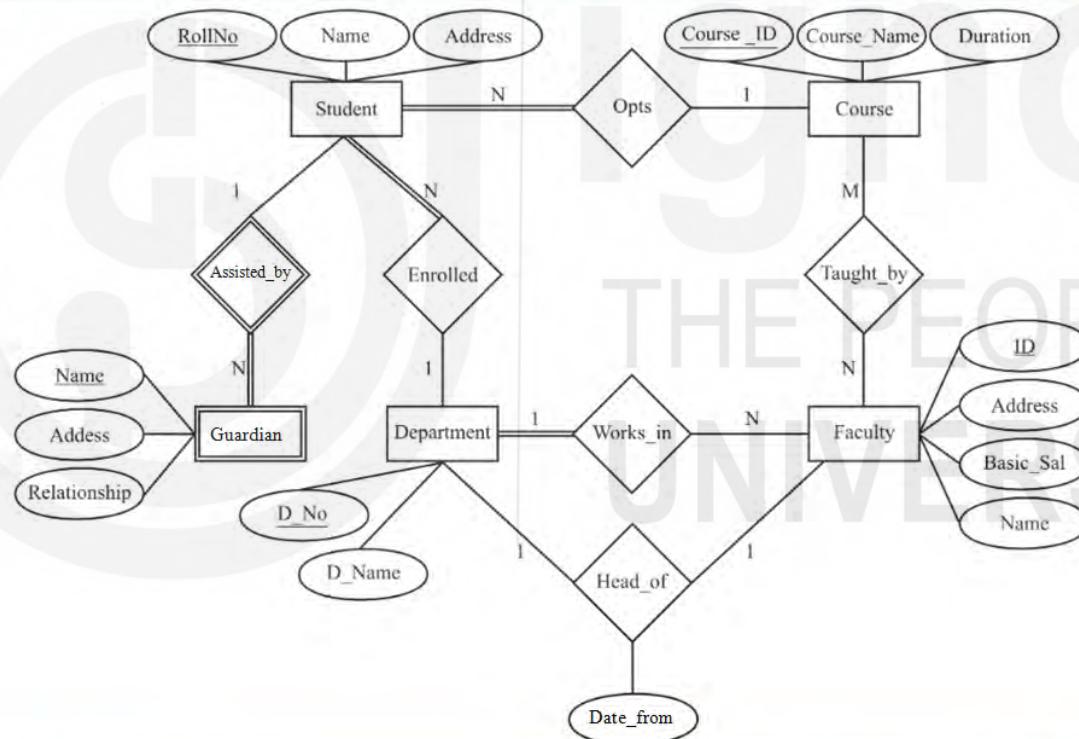


Figure 3.3: E-R diagram of COLLEGE database

3.4 CONVERSION OF E-R DIAGRAM TO RELATIONAL DATABASE

A relational database management system is designed from an E-R diagram, which represents various entities and relationships among entities for an application using the following methods.

Conversion of entity sets:

Strong entity set: For each strong entity set E in the E-R diagram, you create a relation R containing all the simple attributes of E. The primary key of the relation R will be one of the key attributes of R.

For example, the entities Student, Faculty, Course and Department, which are strong entities, relations as shown in Figure 3.4 would be created.

Student (Rollno, Name, Address)
Faculty (Id, Name, Address, Basic_Sal)
Department (D_No, D_Name)
Course (Course_ID, Course_Name, Duration)

Figure 3.4: Conversion of Strong Entities to relations

II) For each weak entity type W in the E-R Diagram, you create another relation R that contains all simple attributes of W. Further, you add the key attribute(s) of owner entity set (say KP) of W in R. The primary key to this relation R is – <KP + Discriminator attribute of W> and foreign key is KP, which references the owner entity of W.

For example, conversion of weak entity Guardian into relation is shown in Figure 3.5. Please note that the owner entity of the Guardian entity is the strong entity Student, whose key is RollNo. Therefore, the key to Guardian relation is RollNo+Name. The Foreign key in Guardian relation is RollNo, which refers to Student relation.

Guardian (RollNO, Name, Address, Relationship)
Foreign Key: RollNO refers to relation Student

Figure 3.5: Conversion of weak entity Guardian to relation.

Conversion of relationship sets

Binary Relationships

I) One-to-one relationships:

For each 1:1 relationship set in the E-R diagram involving two entities E1 and E2 you choose one of the two entities (say E1), preferably the one with total participation, and add primary key attribute of the other entity E2, in the relation of entity E1. Make this added attribute in E1 relation as a foreign key attribute to the relation created from other entity (E2). You should also include all the simple attributes of the relationship type, if any, in the relation E1.

For example, the Head_of relationship in Figure 3.3 is 1:1. The two entities participating in the relationship are - Department and Faculty. The participation of Department entity is *total* in the Head_of relationship. Therefore, the ID attribute, which is the primary key of Faculty entity is added to Department relation. Please note, you can rename this attribute in the Department relation, if needed. In addition, this attribute in Department relation will be the foreign key to the Faculty relation. Further, the attribute Date_from of the Head_of Relationship is also added to the

Department relation. This is shown in Figure 3.6. Please note that you will keep information in this the relation only stores the ID of the current head and Date from which s/he is the head. Please also note that you will not create a separate table of the relationship Head_of.

Department (D_No, D_Name, Head_ID, Date_from))

ID attribute of Faculty relations is added to Department relation and has been renamed as Head_ID.

Foreign Key: Head_ID references ID attribute in Faculty relation

Figure 3.6: Converting 1:1 relationship (No new relation is added)

II) One-to-many or many-to-one relationships:

Both relationship sets involve two entity sets, say E1 and E2. Further, assume that E1 is on the many side and E2 is on the one side of the relationship set. You just need to include the primary key of the relation of entity on the one side (E2 in the case as above) to the relation created for the entity set of many side (E1 in the present case). You should include all simple attributes (or simple components of a composite attributes of relationship set, if any) in the relation of E1. Please note that now relation of E1 has a foreign key reference to relation of E2.

For example, the Works_in relationship between the entities Department and Faculty. For this relationship, the entity at N side is Faculty, therefore add primary key attribute of entity Department, i.e., D_No as a foreign key attribute in relation created for Faculty entity set. This is shown in Figure 3.7

Faculty (Id, Name, Address, Basic_Sal, D_No))

The Works_in relationship has been included in relation Faculty.

D_No is a foreign key and references the relation Department.

Figure 3.7: Converting 1:N relationship to relation (No new table is added in this case)

III) Many-to-many (M : N) relationship:

Consider that a M : N relationship set is binary with two participating entities, say Entity1 and Entity2. For this type of relationship set a relation a created. This relationship set should contain the Primary key of Entity1 (say PKE1), as well as the Primary key of Entity2 (say PKE2). In addition, any attribute of the relationship set is added to the relation. The primary key of this newly form relation of the M : N relationship set is the composite primary key PKE1+PKE2. Please also note that for this new relation of the relationship set, there exists two foreign keys – PKE1, which refers to relation of Entity1 and PKE2, which refers to the relation of Entity set Entity2.

For example, the m : n relationship Taught_by between entitiy sets Course and Faculty should be represented as a new table. The structure of the table will include primary key of Course and primary key of Faculty entities. Please note that both Course and Faculty relations remain unchanged.

Add the following relation into already existing list of relations:

Taught_by (Course_ID, ID)

Primary Key: Course_ID + ID

Foreign Keys:
 Course_ID references Course relation
 ID references Faculty relation
 This relation has no other attribute, as the relationship has no attribute.

Figure 3.8: Converting m : n relationship Taught_by into relations

N-ary Relationships

There are several cases for creating relation for n-ary relationship. A very general case is presented here. For each n-ary relationship set R where $n > 2$, you create a new table S to represent R. You should include the primary key of all the participating entity sets as the foreign key attributes in S. You should include any simple attributes of the n-ary relationship set (or simple components of complete attributes) 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 sets. Figure 3.8 is a special case of n-ary relationship, i.e. a binary relationship.

Multivalued attributes:

For each multivalued attribute ‘A’ of an entity set E, you can create a new relation R that includes an attribute corresponding to the primary key attribute of the relation entity E that represents the entity set or relationship set that has as an attribute. The primary key of R is then combination of A and primary key of relation of E. For example, if a Student entity had RollNo, Name and PhoneNumber attributes, where phone number is a multivalued attribute, then you will create two tables for this entity as given below:

Student (RollNo, Name)
 Phone (RollNo, PhoneNumber)

Converting Generalisation / Specialisation hierarchy to tables:

A simple rule for conversion may be to decompose all the specialised entities into relations in case they are disjoint. For example, for the E-R diagram of Figure 3.1, you can create the two tables as:

Saving_account (account-no, holder-name, branch, balance, interest).
 Current_account (account-no, holder-name, branch, balance, charges).

The other way might be to create tables that are overlapping (not disjoint) for example, assuming that in the E-R diagram of Figure 3.2 contains overlapping sub-classes, then you would be creating the following three relations:

The first relation would be for the for higher level entity:

account (account-no, holder-name, branch, balance)

The specialisation entities will contain the Primary key of the generalized entity and all the attributes of entity itself, as shown below:

saving (account-no, interest)
 current (account-no, charges)

Thus, the information about a single account can be found in all the three relations.

Check Your Progress 1

- 1) A company wants to develop an application to store information about its clients. Find the possible entities and relationships among the entities. Show the step-by-step procedure to make an E-R diagram for the application.

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

- 2) An employee works for a department. If the employee is a manager, then s/he manages the department. Every employee works on at least one or more projects, which are controlled by various departments of a company. An employee can have many dependents. Draw an E-R diagram for the above company. Find all possible entities and their relationships.

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

- 3) A supplier, located in only one-city, supplies various parts for the projects of different companies located in various cities. You can name this database as “supplier-and-parts”. Draw the E-R diagram for the supplier-and-parts.

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

- 4) Convert the E-R diagram created for question 2 above into a relational database.

.....
.....

3.5 ENHANCED E-R MODEL

Enhanced E-R models can help in designing of relational and object-relational database system. In addition, to E-R modeling concepts the Enhanced ER model includes:

- 1) Subclass and Super class

- 2) Inheritance
- 3) Specialisation and Generalisation.

As discussed earlier, an entity may be an object with physical existence or it may be an object with a conceptual existence. An entity is depicted by a set of attributes. Sometimes, an entity can consist of explicit sub-groupings. For example, an entity *vehicle* consists of sub-groups – Truck (or Commercial Vehicles), Light Motor Vehicles (or Car), Two-Wheelers (or Scooter), etc. Every sub-grouping must belong to entity set *vehicle*, therefore, these sub-groupings are called a subclass of the *vehicle* entity set and the *vehicle* itself is called the super class for each of these sub-classes.

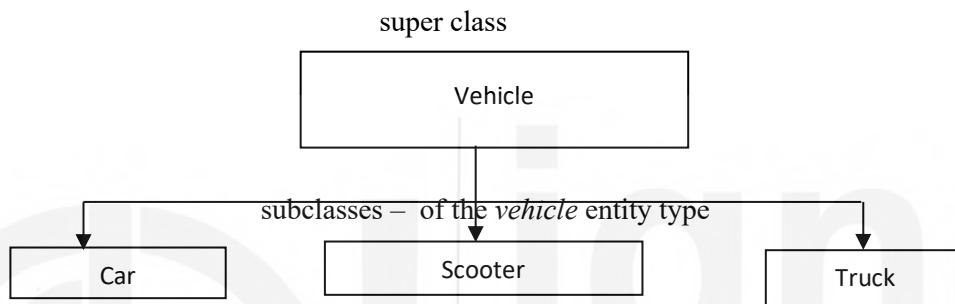


Figure 3.9: A class hierarchy

The relationship between a super class and any of its subclasses is called class/subclass relationship. It is often called an IS-A relationship because of the way we refer to the concept, as you would write, “Car *is-a* vehicle”. The member entity of the sub-class represents the same real world as the member entity of the super class. If an entity is a member of a sub-class, by default it must also become a member of the super class; whereas it is not necessary that every entity of the super class must be a member of its sub-class. An entity that is a member of a sub-class inherits all the attributes of its super class. An entity set is identified by its attributes and the relationship sets in which it participates; therefore, a sub-class entity also inherits all the relationships in which the super class participates. According to **inheritance** the sub-class inherits attributes and relationships of the super class. In addition, every subclass can contain its own attributes and relationships in which it has participated.

Specialisation is a process in which an entity set (super class) is modeled as a set of sub-entity sets (sub-classes) by using a specific distinguishing characteristic of the super class. For example, in Figure 3.9, the super class *vehicle* is modeled using sub classes - Truck (or Commercial vehicle), Car (or Light Motor Vehicle), Scooter (or Two-wheelers)) using the *Type* attribute of the *vehicle* class. Please note that several specialisations hierarchies can be modeled using a super class by using of different distinguishing characteristics. Figure 3.10 shows how you can represent a specialisation with the help of an EER diagram.

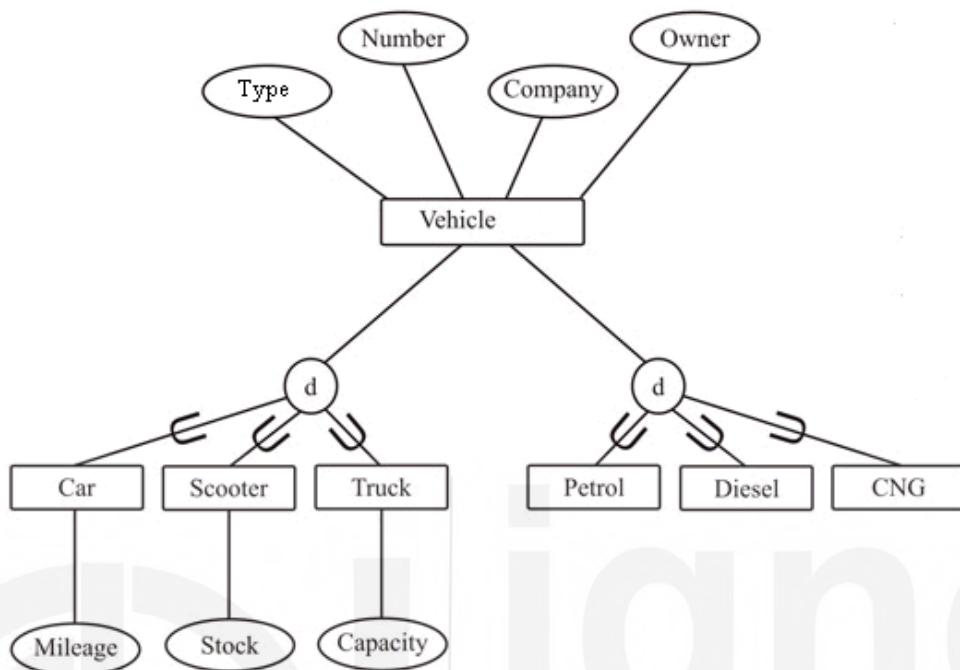


Figure 3.10: EER diagram showing more than one specialization from one super class

In Figure 3.10, letter 'd' in the circle indicates that all these subclasses are **disjoint** in nature, i.e. all the vehicle entities are disjoint, as they can be part of only one of the sub-class. Please also notice that in Figure 3.10, common attributes, like vehicle number, owner name etc., are attributes of the super class, whereas attributes like mileage of car, stock of scooter and capacity of truck are the attributes of the sub-classes. Please note that an entity will be appearing twice in the EER diagram – once in the sub-class and the other in the super class (Please refer to Figure 3.11).

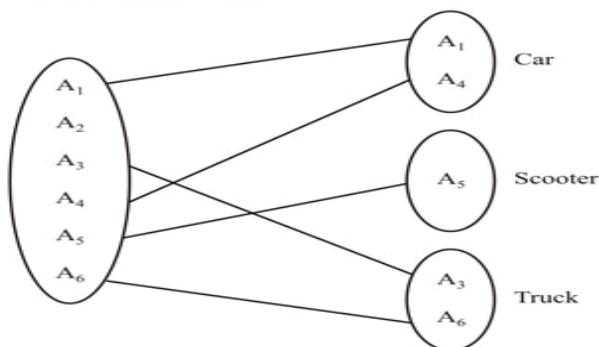


Figure 3.11: Sharing of members of the super class vehicle and its sub-classes

Hence, the specialisation is a set of sub-classes of an entity set, which establishes additional specific attributes with each sub-class and establishes additional specific relationship sets between a sub-class and other entity types or other sub-classes.

Generalisation is the reverse process of specialisation; in other words, it is a process of representing entity sets consisting of entities, which have almost similar attributes excepting few. The similar attributes of these entity set forms the super For example, the entity set Car and Truck can be generalised into entity set Vehicle. Therefore, Car and Truck can now be sub-classes of the super class generalised class Vehicle (Refer to Figure 3.12).

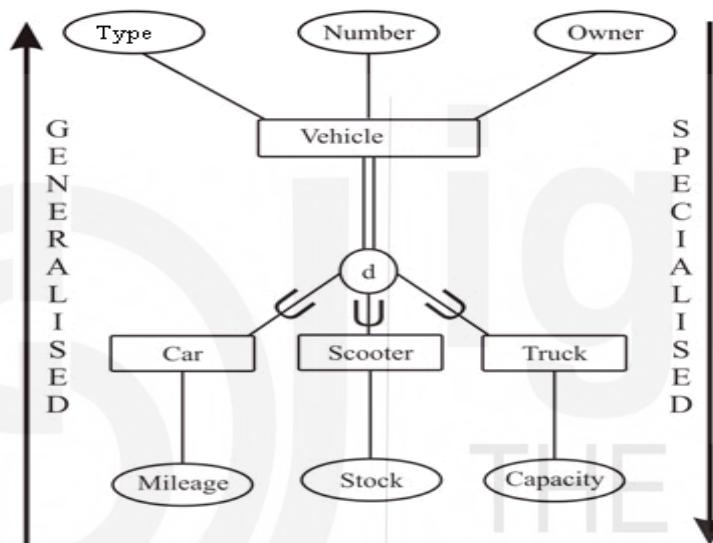


Figure 3.12: Generalisation and Specialisation

Constraints and Characteristics of Specialisation and Generalisation: A super class may either have a single sub-class or many sub-classes in specialisation. In case of only one sub-class you do not use circle notation to show the relationship of sub-class/super class. Sometimes in specialisation, the subclass becomes the member of the super class after satisfying a condition on the value of some attributes of the super class. Such sub-classes are called *condition-defined* sub-classes or predicate defined subclasses. For example, Vehicle entity set has an attribute vehicle “Type”, as shown in Figure 3.12.

You can specify the condition of membership for a subclass – car, truck, scooter – by the predicate – vehicle ‘Type’ of the super class vehicle. Therefore, a vehicle object can be a member of the sub-class, if it satisfies the membership condition for that sub-class. For example, to be a member of sub-class Car a vehicle entity must have the condition vehicle “type = car” as true. A specialisation in which an attribute or a set of attributes of the super class (called the *defining attribute* of specialisation) specify membership condition of the sub-classes, is termed as *attribute-defined* specialisation. In case no membership condition is stated for

specialisation, a database user determines the membership. Such specialisation is termed as user-defined specialisation.

Disjointness is also the constraints to a specialisation, which specifies that a given entity cannot be a member of more than one sub-classes for a specific specialisation hierarchy. For example, in Figure 3.12, the symbol ‘d’ in circle stands for disjointness, as an entity can be a member of a single sub-class only. But if the real-world entity is not disjoint entities sets of the sub-classes may overlap. This is represented by an (o) in the circle. For example, if you classify a class Book into sub-classes Textbooks and Reference Books, then you may like to define a specific book in both the sub-classes. This is a case of Overlapping constraint.

When every entity in the super class must be a member of some subclass in the specialisation it is called total specialisation. But if every entity does not necessarily needs to belong to any of the subclasses, it is called partial specialisation. The total is represented by a double line. This is to note that in specialisation and generalisation the deletion of an entity from a super class implies automatic deletion from sub-classes belonging to the same; similarly, insertion of an entity in a super class results in insertion of the entity in all the sub-classes for which the attributes of this entity fulfils the constraints of attribute-defined specialisation. In case of total specialisation, insertion of an entity in a super class implies compulsory insertion in at least one of the sub-classes of the specialisation.

Union: In some cases, a single class has a similar relationship with more than one classes. For example, the sub class ‘Car’ may be owned by two different types of owners: Individual or Organisation. Both these types of owners are different classes; thus, such a situation can be modeled with the help of a Union (Refer to Figure 3.13).

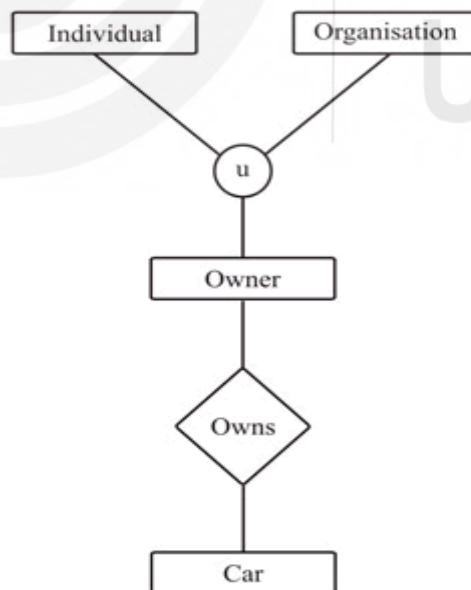


Figure 3.13: Union of classes

In the next section, we discuss how these extended features can be converted to relations.

3.6 CONVERTING EER DIGRAM TO RELATIONS

The rules for converting EER diagram, which primarily includes specialisation and generalisation hierarchy are the same as in E-R diagram. Let us recapitulate these rules:

- Create a relation for each strong entity set.
- Create a relation for a weak entity set. The primary key of this relation would be a composite key involving the attributes of the primary key of the strong entity set on which it depends and discriminator of the weak entity.
- Create a relation for each binary m : n relationship set having the primary keys of both the participating entities, which form the composite primary key to the relation. The individual primary key are the foreign keys to respective relations.
- For a binary m : 1 or 1 : m relationship, in general, the primary key on the m side is added to 1 side of entity. This also becomes the foreign key. For binary 1:1 relationship set the primary key of chosen participating relation is added to the other participating relation.
- Composite attribute may sometimes be converted to a separate relation.
- For generalisation or specialisation hierarchy a relation can be created for higher level and each of the lower-level entities. The higher-level entity would have the common attributes and each lower-level relation would have the primary key of higher-level entity and the attributes defined at the lower specialised level. However, for a complete disjoint hierarchy no relation may be made at the higher level, but the relations are made at the lower level including the attributes of higher level.
- For an aggregation, all the entities and relationships of the aggregation are transformed into the relation on the basis of rules stated above. A relation is also created for the relationship set that exists between the aggregated entity (say AgE) and other simple entity (SE). The primary key of this new relation is the composite key involving primary key of the AgE and SE.

So let us now discuss the process of converting EER diagram into a relation. In case of disjoint constraints with total participation. It is advisable to create separate relations for the sub-classes. But the only problem in such a case will be to implement the referential entity constraints suitably.

For example, assuming that this EER diagram can be converted into a relation as:

Car (Number, owner, mileage)
Scooter (Number, owner, stock)
Truck (Number, owner, capacity)

Please note that referential integrity constraint in this case would require relationship with three relations and therefore is more complex to implement.

In case, in the EER diagram of Figure 3.12 there is NO total participation of Vehicle super class in the sub-classes, then there will be some vehicles, which are not Car, Scooter and Truck, so how can you represent these? In addition, in case of overlapping constraint, some tuples may get represented in more than one relation. Thus, in such cases, it is ideal to create one relation for the super class and other relations for the sub-classes having the primary key and any other attributes of that sub-class. For example, with NO total participation the following relations would be created for the EER diagram of Figure 3.12:

Vehicle (Number, owner, type)

Car (Number, mileage)

Scooter (Number, stock)

Truck (Number, capacity)

Finally, in the case of union since it represents dissimilar classes, you may represent separate relations. For example, both individual and organisation will be modeled to separate relations.

Check Your Progress 2

- 1) What is the use of EER diagram?

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

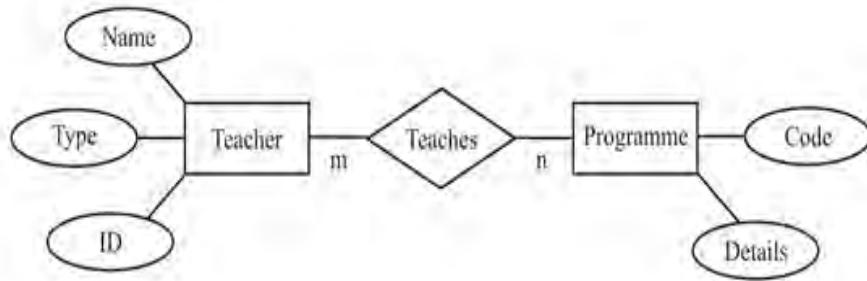
- 2) What are the constraints used in EER diagrams?

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

- 3) How is an EER diagram converted into a relation?

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

- 4) Consider the following E-R diagram.



‘Type’ can be regular or visiting faculty. A visiting faculty members can teach only one programme. Make a suitable EER diagram for this and convert the EER diagram to table.

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

3.7 SUMMARY

This unit presents the concept of E-R model and EER models. Both these models are represented with the help of E-R diagram and EER diagram. These diagrams are very powerful tools to represent the need of data in a database system and can be used for design of good database system. The E-R model explained in this unit covers the basic aspects of E-R modeling. The unit defines the concept of entities, attributes and relationships. Further, it defines different types of entities like strong and weak entities; different types of attributes such as simple, composite, derived etc.; the cardinality and participation constraints in a relationship. These concepts are very useful and you should attempt solving related problems from the further readings. The different concepts of EER diagram including generalisation, specialisation, union etc. have also been explained in this unit. You may refer to further readings for more details on E-R and EER diagrams.

3.8 SOLUTIONS/ ANSWERS

Check Your Progress 1

1.

Let us show the step-by-step process of development of Entity-Relationship Diagram for the Client Application system. The first two important entities of the system are **Client** and **Application**. Each of these terms represents a noun, thus, they are eligible to be the entities in the database. But are they the correct entity sets? Client and Application both are **independent** of anything else and company plans to keep track

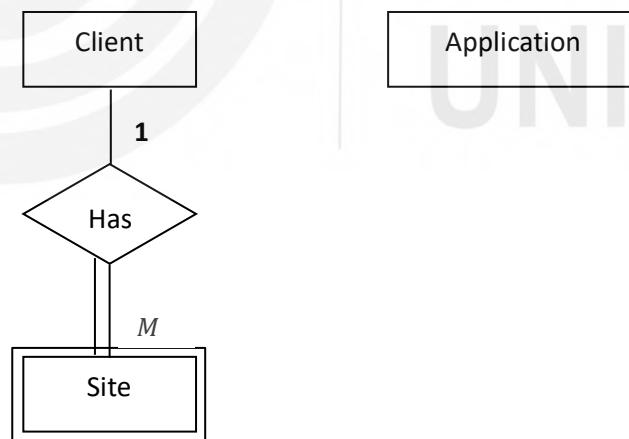
of its clients and the applications being developed for them. Therefore, each of the entities-Client and Application form an entity set.

But how are these entities related? Are there more entities in the system? Let us first consider the relationship between these two entities, if any. Obviously, the relationship among the entities depends on interpretation of written requirements. Thus, we need to define the terms in more detail.

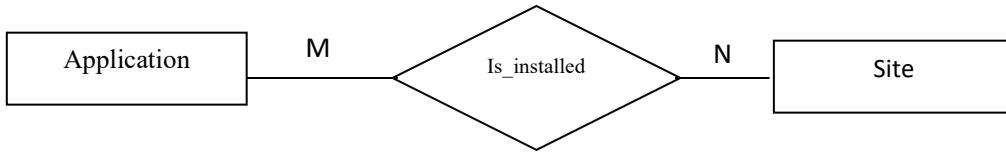
Let us first define the term **Application**. Some of the questions that are to be answered in this regard are: Is keeping track of **Accounts** an application? Is the accounting system installed at each client site regarded as a different application? Can the same application be installed more than once at a particular client site?

Before you answer these questions, do you notice that another entity is in the offering? The client **site** seems to be another candidate entity. This is the kind of thing you need to be sensitive to at this stage of the development of entity relationship modeling process. So, let us first deal with the relationship between Client and Site before coming back to Application. Just a word of advice: "It is often easier to tackle what seems likely to prove simple before trying to resolve the apparently complex."

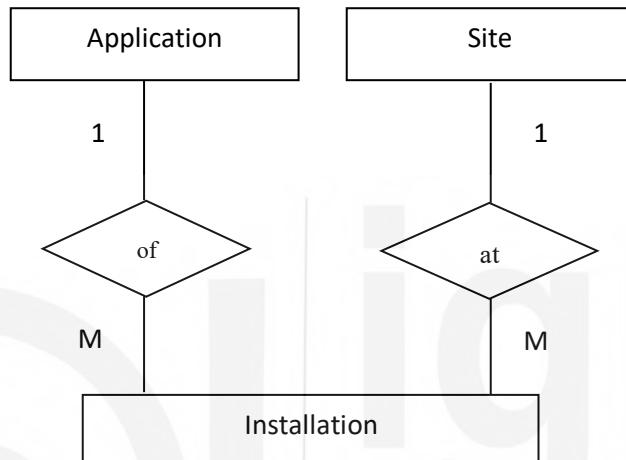
Each Client can have many sites, but each site belongs to one and only one client. Now the question arises what entity type is Site? You cannot have a site without a client. If any site exists without a client, then who would pay the company? This is a good example of an existential dependency and one-to-many relationship. Thus, site is a weak entity. This is illustrated in the part E-R diagram given below:



Let us now relate the entity Application to other entities. Please note that several applications developed by the company can be installed at several client sites. Thus, there exists a many-to-many relationship between the entities Site and Application:



However, the M: M relationship “is_installed” have many attributes that need to be stored with it, specifically relating to the authorised persons, setup constraints, dates, etc. Thus, it may be a good idea to promote this relationship as an Entity.



The Figure as given above consists of the following relationships:

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

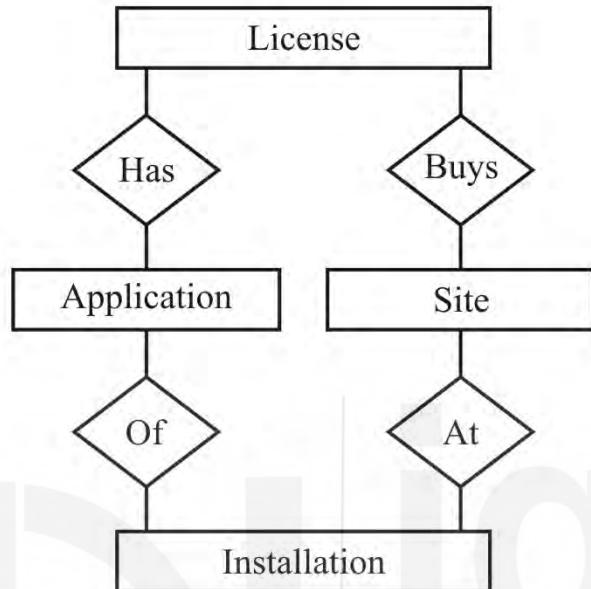
Please note that entities can be recognised in one of two ways – from the nouns of the requirement specification of the system or because of resolving a M:M relationship, as in this case. When you create a new entity in this way, you must find a suitable name for it. This can sometimes be based on the **verbs** used to describe the M:M. For example, from the statement **you can install the application at many sites**, you can choose the verb **install** and convert it to related noun **Installation**. But what how will you identify the attributes and relationships of Installation entity? To find these, you may like to answer the following questions:

- How do you desire to identify each Installation entity?
- What data would be stored in Installation entity?
- Is an Installation independent of any other entity, that is, can an entity Installation exist without being associated with the entities Client, Site and Application?

In the present design, there cannot be an Installation until you can specify the Client, Site and Application. But since Site is existentially dependent on Client or in other words, Site is subordinate to Client, the Installation can be identified by (Client) Site (it means Client or Site) and Application. You do not want to allow more than one record for the same site and application.

But what if we also sell multiple copies of packages for a site? In such a case, you need to keep track of each individual copy (license number) at each site. In that

case, you need another entity named Package (with license number). You may even need separate entities for Application and Package. This will depend on what attributes you want to keep in each of these entities. Thus, with these requirements, the E-R diagram may be extended to as shown below:



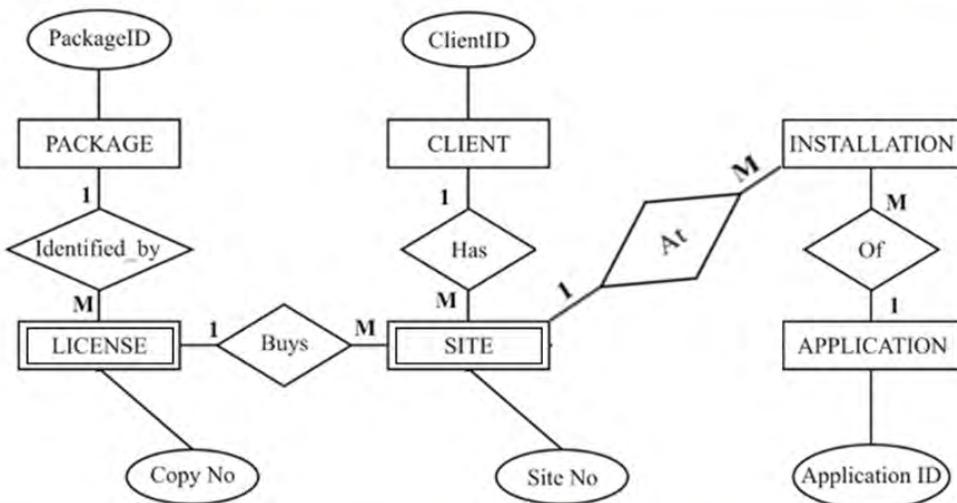
Let us define the additional relationships given above:

Has: describes that each application has one or more licenses

Buys: describes each site buys the licensed copies of application

You might decide that License should be sub-class to Package entity, therefore, the best unique identifier for the License entity could be Package ID and License serial number. The reason for this relationship is that the license numbers are issued by the companies to whom the Package belongs. The present company does not issue license numbers, and thus have no control over their duration, other service requirements, as well as the length of data types used for different attributes of the Package and License entity sets. Also, the company does not have control over their uniqueness and changeability of license numbers. Please note that **it is safer to base primary keys on internally assigned values**. What if you make License as a sub-class to Package entity set? It also seems that the client site is not an essential part of the identifier, as the client is free to move the copy of a package to another site. Thus, we should definitely not make client/site part of the primary key of License.

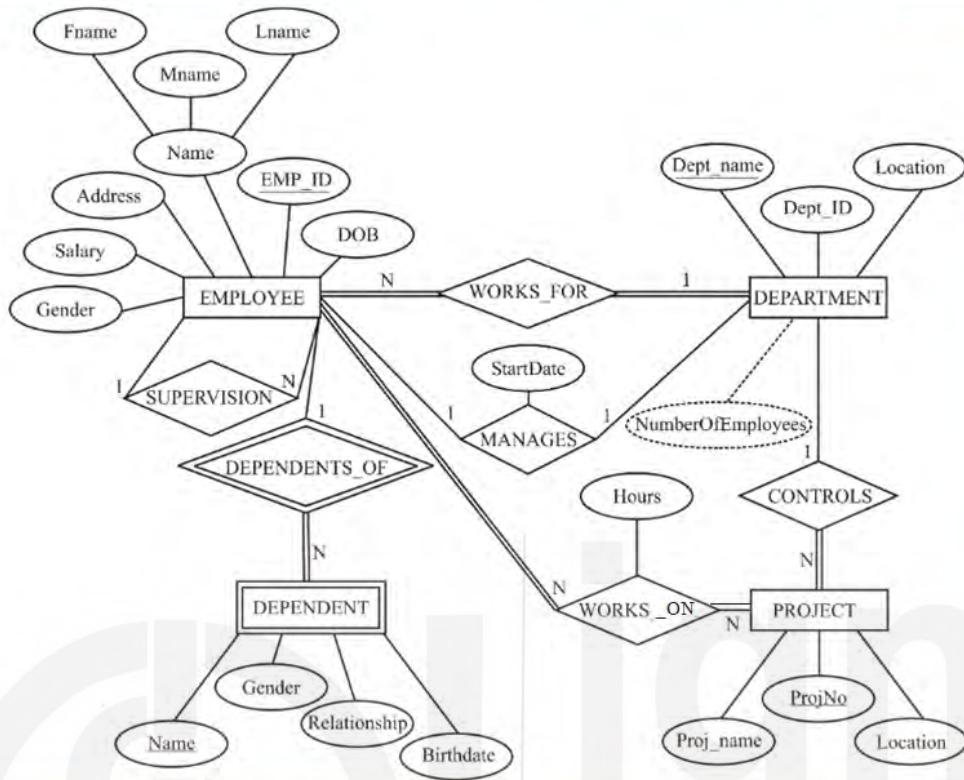
A final proposed E-R diagram for the problem is given below. Please keep thinking and refining your reasoning. Please note that knowing and thinking about a system is essential for making good E-R diagrams. (Please note that in this E-R Diagram, Site and License are modelled as weak entities, though you can decide to change it.)



The following table lists probable entities identified so far, together with its super class, if any, primary keys, and any foreign keys.

Entity	Super Class (if any)	Primary Key	Foreign Keys
Client	-	Client ID	
Site	Client	Client ID, Site No	Client ID
Application	-	Application ID	
Package	-	Package ID	
Installation	Site, Application	Client ID, Site No, Application ID	Client ID, Site No, Application ID
License	Package	Package ID, Copy No	Package ID

2) The E-R diagram is given below:



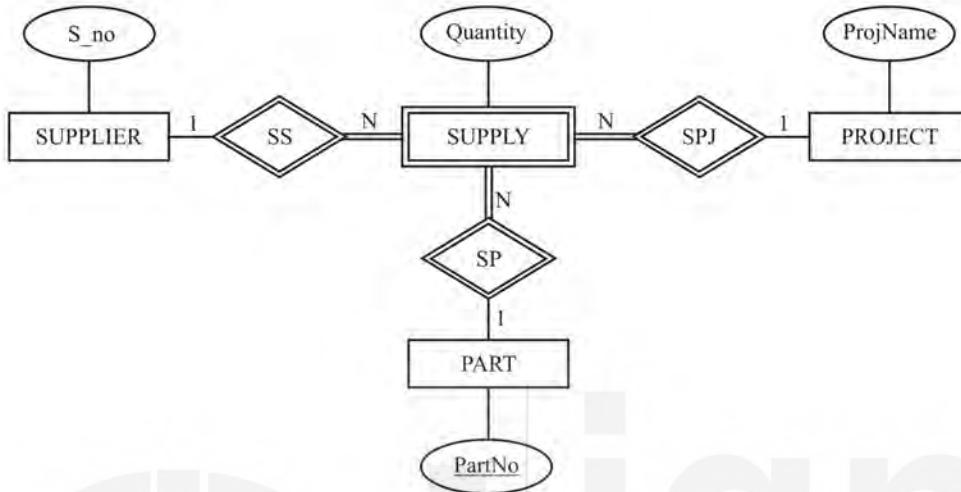
In the E-R diagram given above, EMPLOYEE is an entity, who works for a department, i.e., entity DEPARTMENT, thus, WORKS_FOR is many-to-one relationship, as many employees work for one department. Only one employee (i.e., Manager) manages the department, thus **manages** is the one-to-one relationship. The attribute Emp_Id is the primary key for the entity EMPLOYEE, thus Emp_Id is unique and NOT NULL. The candidate keys for the entity DEPARTMENT are **Dept_name** and **Dept_Id**. Along with other attributes, NumberOfEmployees is the derived attribute on the entity DEPARTMENT, which is the number of employees working for a department. Both the entities EMPLOYEE and DEPARTMENT participate totally in the relationship WORKS_FOR, as at least one employee work for the department, similarly an employee must work in a department.

The entity EMPLOYEES and the entity PROJECTS are related through the many-to many relationship WORKS_ON, as many employees can work for one or more than one projects simultaneously. The entity DEPARTMENT and entity PROJECT has a relationship CONTROLS. Since one department controls many projects, thus, CONTROLS in a 1:N relationship. The entity EMPLOYEE participates totally in the relationship WORKS_ON, as each employee works on at least one project. A project should also have at least one employee, therefore, its participation is also total in WORKS_ON.

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

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

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



4. Let us first make simple relation for the E-R diagram in the answer of question 2:
EMPLOYEE(EMP_ID, Fname, Mname, Lname, DOB, Address, Salary, Gender)
DEPARTMENT(Dept_ID, Dept_name, Location)
DEPENDENT(EMP_ID, Name, Gender, Relationship, Birthdate)

Foreign Key: EMP_ID references EMPLOYEE

PROJECT(ProjNO, Proj_name, Location)

WORKS_FOR: due to this relationship the Primary key of 1 side (Dept_ID) will be added to EMPLOYEE relation.

SUPERVISION: this relationship is on the same entity, therefore, an attribute

Supervisor_ID whose domain is EMP_ID will be added to the EMPLOYEE

MANAGES: It is a 1 : 1 relation, you can chose the Department side as there will be less records. It also has an attribute StartDate. Therefore,

MANAGER_ID whose domain is EMP_ID and StartDate attribute would be added to DEPARTMENT.

CONTROLS: It is a 1 : N relationship, so the Primary key of 1 side (Dept_ID) will be added to PROJECT relation.

DEPENDENT_OF: This relation is already included in DEPENDENT relation

WORKS_ON: It is a many to many (N : N) relation with total participation on both sides, therefore, a separate table will be created for WORKS_ON with primary key of both the participating entities and attributes of this relation (Hours)

Thus, the final relations would be:

EMPLOYEE (EMP_ID, Fname, Mname, Lname, DOB, Address, Salary, Gender, Dept_ID, Supervisor_ID)

Foreign Key: Dept_ID references DEPARTMENT.

Domain Constraint: Domain of Supervisor_ID is EMP_ID.

DEPARTMENT (Dept_ID, Dept_name, Location, MANAGER_ID, StartDate)

Foreign Key: MANAGER_ID references EMP_ID of EMPLOYEE.

DEPENDENT (EMP_ID, Name, Gender, Relationship, Birthdate)

Foreign Key: EMP_ID references EMPLOYEE

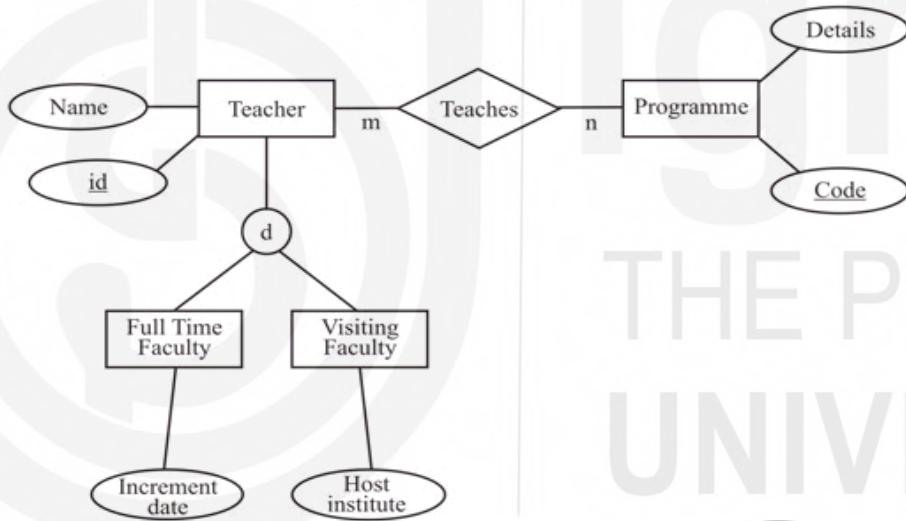
PROJECT (ProjNO, Proj_name, Location, Dept_ID)

Foreign Key: Dept_ID references DEPARTMENT.

WORKS_ON (EMP_ID, ProjNo, Hours)

Check Your Progress 2

- 1) The EER diagrams are used to model advanced data model requiring inheritance, specialisation and generalisation.
- 2) The basic constraints used in EER diagrams are disjointness, overlapping and unions.
- 3) For disjointness and union constraints the chances are that you create separate relations for the subclasses and no relation for super class. For overlapping constraint, it is advisable to create a relation of super class and the relations of sub-classes will have only those attributes that are not common to super class except for the primary key.
- 4) The modified EER diagram is:



FullTimeFaculty (id, Name, Increment-date)

VisitingFaculty (id, Name, Host-institute)

Teachers (id, code)

Programme (code, details)

UNIT 4 FILE ORGANISATION IN DBMS

Structure	Page Nos.
4.0 Introduction	
4.1 Objectives	
4.2 Physical Database Design	
4.3 Database Storage on HDD and SSD	
4.4 File Organisations	
4.4.1 Unordered Heap file Organisation	
4.4.2 Sequential File Organisation	
4.4.3 Indexed (Indexed Sequential) File Organisation	
4.4.4 Hashed File Organisation	
4.5 Indexes	
4.6 Implementing Index Using Tree Structure	
4.7 Multi-key File Organisations	
4.7.1 Multiple Access Paths	
4.7.2 Multi-list File Organisation	
4.7.3 Inverted File Organisation	
4.8 Importance of File Organisation in Databases	
4.9 Summary	
4.10 Solutions/Answers	

4.0 INTRODUCTION

In earlier units, you studied the basic concepts of database management systems, entity relationship diagram and relational algebra. Databases are used to store information. Normally, the principal operations you need to perform on database are those relating to:

- Creation of data
- Retrieving the data using conditions
- Modifying
- Deleting some information which we are sure is no longer useful or valid.

Database structures data as two-dimensional tables, which allows easy processing of these operations. However, as the size of databases are large, the databases are required to be stored on secondary memory of computer (such as hard disk or SSD). Therefore, the secondary storage system of a databases are mainly concerned with the following issues:

- Storing table or tables as files: A single table can be stored as a file or several tables can be put together as a cluster of related records called cluster file.
- Attributes of a table: In general, a table represents the data of one object, therefore, the attributes represent data of a specific object and may be required to be accessed by a specific operation. Thus, it may be good idea to store all the attributes of an object together. Does the order of attributes in a file matter?
- It seems logical to store all the records of a table contiguously. But, how such records should be ordered? The ordering of records in primary storage does not matter, however, for the secondary storage a specific sequence may be desired. Such decisions may be taken by the database demonstrator and may relate to the performance of the database.
- In the cases of analytical queries, particular attributes are stored together, this approach is called column-oriented approach, this approach is beyond the scope of this Unit. You may refer to further readings for this approach.

This unit focuses on the file Organisation in DBMS, the access methods available and the system parameters associated with them. File Organisation is the way the files are arranged on the disk and access method is how the data can be retrieved based on the file organisation.

4.1 OBJECTIVES

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

- define storage of databases on hard disks and SSD;
- discuss the implementation of various file Organisation techniques;
- discuss the advantages and the limitation of the various file Organisation techniques;
- describe various indexes used in database systems, and
- define the multi-key file organisation.

4.2 PHYSICAL DATABASE DESIGN

The database design involves the process of logical design with the help of E-R diagram, normalisation, etc., followed by the physical design.

The Key issues in the Physical Database Design are:

- The purpose of physical database design is to translate the logical description of data into the technical specifications for storing and retrieving data for the DBMS.
- The goal is to create a design for storing data that will provide adequate performance and ensure database integrity, security and recoverability.

Some of the basic inputs required for Physical Database Design are:

- Normalised relations
- Attribute definitions
- Data usage: entered, retrieved, deleted, updated
- Requirements for security, backup, recovery, retention, integrity
- DBMS characteristics.
- Performance criteria such as response time requirement with respect to volume estimates.

However, for such data some of the Physical Database Design Decisions that are to be taken are:

- Optimising attribute data types.
- Modifying the logical design.
- Specifying the file Organisation.
- Choosing indexes.

Designing the attributes in the data base

The following are the considerations one has to keep in mind while designing the attributes in the data base.

- Choosing data type
- Coding, compression, encryption
- Controlling data integrity
- Default value
 - Range control
 - Null value control
 - Referential integrity
- Handling missing data

- Substitute an estimate of the missing value
- Trigger a report listing missing values
- In programs, ignore missing data unless the value is significant.

Physical Records

These are the records that are stored in the secondary storage devices. For a database relation, physical records are the group of fields stored in adjacent memory locations and retrieved together as a unit. Considering the page memory system, data page is the amount of data read or written in one I/O operation to and from secondary storage device to the memory and vice-versa. In this context we define a term blocking factor that is defined as the number of physical records per page.

The issues relating to the Design of the Physical Database Files

Physical File is a file as stored on the disk or SSD. The main issues relating to physical files are:

- Constructs to link two pieces of data:
 - Sequential storage.
 - Pointers.
- File Organisation: How the files are arranged on the disk?
- Access Method: How the data can be retrieved based on the file Organisation?

Let us see in the next section how the data is stored on the hard disks (HDD) or SSDs

4.3 DATABASE STORAGE ON HDD OR SSD

At this point, it is worth while to note the difference between the terms file Organisation and the access method. A file organisation refers to the organisation of the data records, which are part of file, into a group of records that are placed in a block of secondary storage. The file organisation also entails the access structures and interlinking of records. An access method is the way - how the data can be retrieved based on the file Organisation.

Mostly the databases are stored persistently on HDD or SSD for the reasons given below:

- The databases being very large may not fit completely in the main memory.
- Data of database is to be stored permanently using the non-volatile storage.
- Primary storage is expensive, using secondary reduce the cost of the storage per unit.

Each hard drive is usually composed of a set of disk platters. Each disk platter has a layer of magnetic material deposited on its surface. The entire disk can contain a large amount of data, which is organised into smaller packages called BLOCKS (or pages). On most computers, one block is equivalent to 1 KB of data (= 1024 Bytes). A block is the smallest unit of data transfer between the hard disk and the processor of the computer. Each block therefore has a fixed, assigned, address. Typically, the computer processor will submit a read/write request, which includes the address of the block, and the address of RAM in the computer memory area called a buffer (or cache) where the data must be stored / taken from. The processor then reads and modifies the buffer data as required, and, if required, writes the block back to the disk. Let us see how the tables of the database are stored on the hard disk.

A Solid-State Disk (SSD) are made up by using flash memories. These SSD devices also provide similar block-oriented access to data, however, since they do not have physically moving component, they provide lower latency and lower access time than that of disks. Therefore, in the subsequent sections, we will provide details on disk oriented approach of data storage.

Fixed and Variable Length Records

There are two basic ways of storing a record on the disks – Fixed Length records and Variable Length Records. In the fixed length records all the attributes are assigned equal space in terms of bytes, just like fixed length structure in C programming. Thus, each record will be of same size. In such cases, only meta data can be used to identify different records and their attributes.

As far as variable length records are concerned, it may be noted that each record of a table may be of different length. This is because in some records, some attribute values may be 'null'; or some attributes may be of type *varchar*, which allows variable number of characters to be stored in an attribute, therefore each record may have a different length string as the value of this attribute. To store variable length records, it is necessary to use a character that marks the end of an attribute value. In addition, an end of record marker will also be needed, as one disk block may store several records. Therefore, each record is separated from the next, again by another special character, the record separator.

The next section discusses different types of file organisation that can be used to store the files on the disks.

4.4 FILE ORGANISATIONS

File organization is used to organize the content of a file on a secondary storage device. A good file organization should support efficient data access and update operations on the content of a file, which is stored on the secondary storage.

Data files are organized so as to facilitate access to records and to ensure their efficient storage. A tradeoff between these two requirements generally exists: if rapid access is required, more storage space is required to make it possible. Selection of File Organisations is dependant on two factors as shown below:

- Typical DBMS applications may need to access a small subset of the data of a database at any given time.
- Whenever a portion of the data is needed by the DBMS; it is located on disk, copied to memory for processing, and rewritten to disk if the data was modified.

A file of record is likely to be accessed and modified in a variety of ways, and different ways of arranging the records enable different operations over the file to be carried out efficiently. A DBMS supports several file Organisation techniques. The important task of the DBA is to choose a good Organisation for each file, based on its type of use.

For a database system, you select a file organisation a suitable file organization based on the parameters like number of keys used to access the files, the ratio of update and access operations, the type of storage technology etc. *Figure 4.1* uses access key as the basis for classifying different file organisations. This classification will be used in this unit to describe various file organisations.

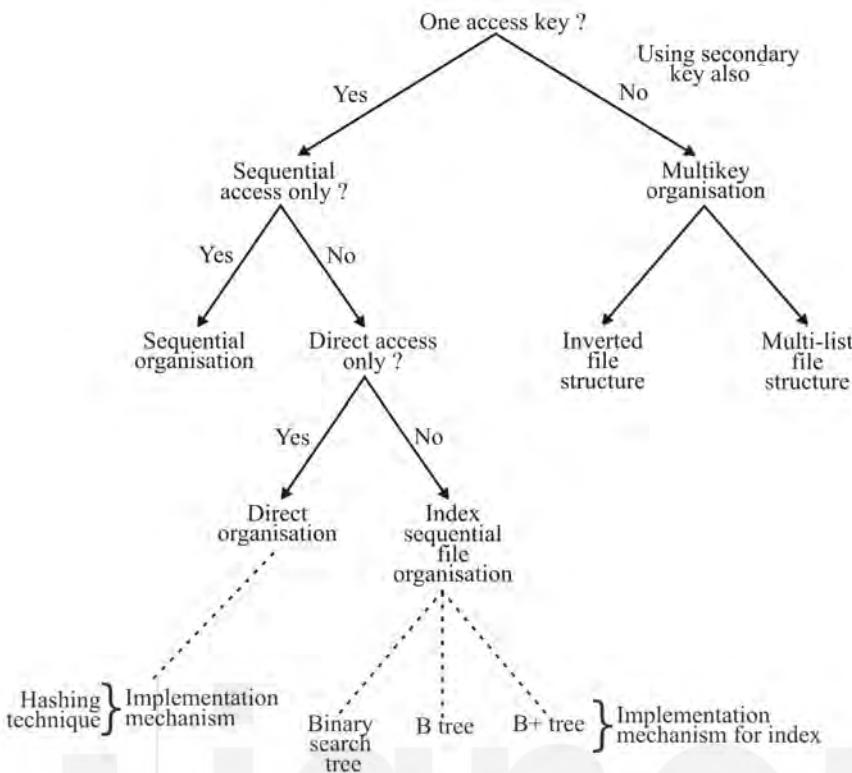


Figure 4.1: Different File Organisation Based on Access Keys

Let us discuss some of these techniques in more details:

4.4.1 Unordered Heap file Organisation

Basically, these files are unordered files. It is the simplest and most basic type. These files consist of randomly ordered records. The records will have no particular ordering. The operations that you can perform on the records of a heap file are insert, retrieve and delete. The features of the heap file Organisation are:

- New records can be inserted in any empty space that can accommodate them.
- When old records are deleted, the occupied space becomes empty and available for any new insertion.
- If updated records grow; they may need to be relocated (moved) to a new empty space. This needs to keep a list of empty space.

Advantages of heap files

1. This is a simple file Organisation.
2. Insertion is somehow efficient.
3. Good for bulk-loading data into a table.
4. Best if file scans are common or insertions are frequent.

Disadvantages of heap files

1. Retrieval requires a linear search and is inefficient.
2. Deletion can result in unused space/need for reorganisation.

4.4.2 Sequential File Organisation

In the sequential Organisation records of the file are stored in sequence (or consecutively) by the primary key field values. In this organisation, the records can be accessed in the order of its primary key. This kind of file Organisation works well for the tasks in which nearly every record of the file is required to be accessed, such as payroll system. Figure 4.2 shows this file organization.

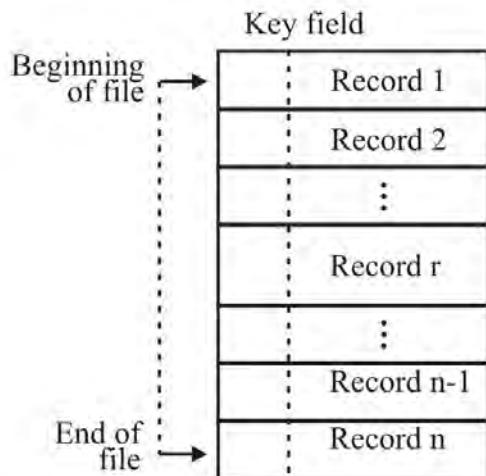


Figure 4.2: Structure of sequential file

A sequential file maintains the records in the logical sequence of its primary key values. Sequential files are inefficient for random access, however, are suitable for sequential access. A sequential file can be stored on devices like magnetic tape that allow sequential access.

On an average, to search a record in a sequential file would require looking into half of the records of the file. However, if a sequential file is stored on a disk (remember disks support direct access of its blocks) with keys stored separately from the rest of record, then only those disk blocks are needed to be read that contains the desired record or records. This type of storage allows binary search on sequential file blocks, thus, enhancing the speed of access.

Updating a sequential file usually creates a new file so that the record sequence on primary key is maintained. The update operation first copies the records till the record after which update is required into the new file and then the updated record is put followed by the remainder of records. Thus, method of updating a sequential file automatically creates a backup copy. However, such update operations are very time consuming.

Addition of records in the sequential files are also handled in a similar manner to update operation. If a record is to be inserted at the last record of the file, it can be performed very easily. However, if a record is required to be inserted in between two records, then such insertion would require shifting down all the subsequent records in the file by one record space. In case of deletion of a record, all the subsequent records need to be shifted up by one record space.

Sequential file organization is most suitable, if all the records of a file are to be processed in a sequence. For example, processing the monthly payroll of all the employees of an organization, will require processing of all the employees records sequentially. However, a single update is expensive as new file must be created, therefore, to reduce the cost per update, all update requests are stored in a single update file, which is sorted in the order of the sequential file ordering key. The file containing the updates is sometimes referred to as a transaction file and is used to update the sequential file in a single processing cycle.

This process is called the batch mode of updating. In this mode each record of master sequential file is checked for one or more possible updates by comparing with the update information of transaction file. The records are written to new master file in the sequential manner. A record that requires multiple updates is written only when all the updates have been performed on the record. A record that is to be deleted is not

written to new master file. Thus, a new updated master file will be created from the transaction file and old master file.

Thus, update, insertion and deletion of records in a sequential file require a new file creation. Can we reduce creation of this new file? Yes, it can be done easily, if the original sequential file is created with holes, which are empty record spaces, as shown in the *Figure 4.3*. Thus, reorganisation of file on addition and update operation can be restricted to only one block, which is read into/ written from the main memory as a single unit. Thus, holes increase the performance of sequential file insertion and deletion. This organisation also supports a concept of overflow area, which can store the spilled over records if a block is full. This technique is also used in index sequential file organisation. A detailed discussion on it can be found in the further readings.

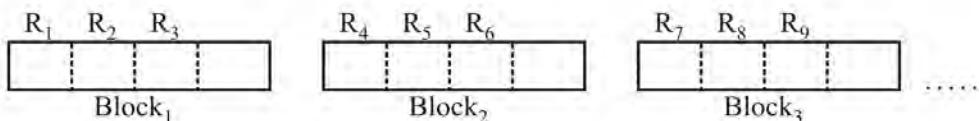


Figure 4.3: A sequential file with empty spaces for record insertions

Advantages of Sequential File Organisation

- It is fast and efficient when dealing with large volumes of data that needs to be processed periodically (batch system).

Disadvantages of sequential File Organisation

- Requires all the new transactions to be sorted into a proper sequence for sequential access processing.
- Locating, storing, modifying, deleting, or adding records in the file require rearranging the file.
- This method is too slow to handle applications requiring immediate updating or responses.

4.4.3 Indexed (Indexed Sequential) File Organisation

It organises the file like a large dictionary, i.e., records are stored in order of the key, but an index is kept which also permits a type of direct access. The records are stored sequentially by primary key values and there is an index built over the primary key field.

To locate a record in a sequential file, which is not indexed sequential, on average, you may read about half the number of records. Therefore, retrieval of a specific record in a sequential file is inefficient, especially if the file has very large number of disk storage blocks. The use of index in a sequential file improves the query response time by adding an index, which is defined as a set of *<index value, address>* pair. An index is a mechanism for faster search, as the size of index is much smaller than the original file. Thus, an index may be contained entirely in the main memory of the computer.

An indexed sequential file is a sequential file, which is stored in the order of its primary key, that contains an index on its primary key. Indexed sequential file allows advantages of indexing and sequential organization of records. The sequential organisation facilitates sequential processing of records, whereas the index helps in enhancing the performance of locating specific record based on the index value. In order, to make such files more efficient for insertion and deleting of records, such files are supported with overflow blocks. This reduces the need of moving all the records in a file. Figure 4.6 is an example of indexed sequential file. Please note that index is represented in Figure 4.6 as *<Primary Index Value, Block Pointer>*.

Hashing is the most common form of purely random access to a file or database. It is also used as an optimisation technique to access columns that do not have an index. Hashing involves the use of a hash function. Input to a hash function is the value of the attribute or set of attributes of a record that are to be used for file organization and the output is the block address or page address, where that record can be found. Figure 4.4 shows a file using hashing file organization. The hash function used for this file is $key \bmod 4$. Notice that records with different key values can be placed in a Block based on the hashing function. To search the location of a record, you can apply the hashing function on the key value and then search the hashed block. For example, if you are searching for the location of key 29, then the record can be found in $29 \bmod 4 =$ Block 1, read this Block in the main memory and do linear search on key value to locate the record in the main memory. The most popular form of hashing is division hashing with chained overflow. You can refer further readings for more details on this file organisation.

Advantages of Hashed file Organisation

1. Insertion or search on hash-key is fast.
2. Best if equality search is needed on hash-key.

Disadvantages of Hashed file Organisation

1. It is a complex file Organisation method.
2. Search is slow.
3. It suffers from disk space overhead.
4. Unbalanced buckets degrade performance.
5. Range search is slow.

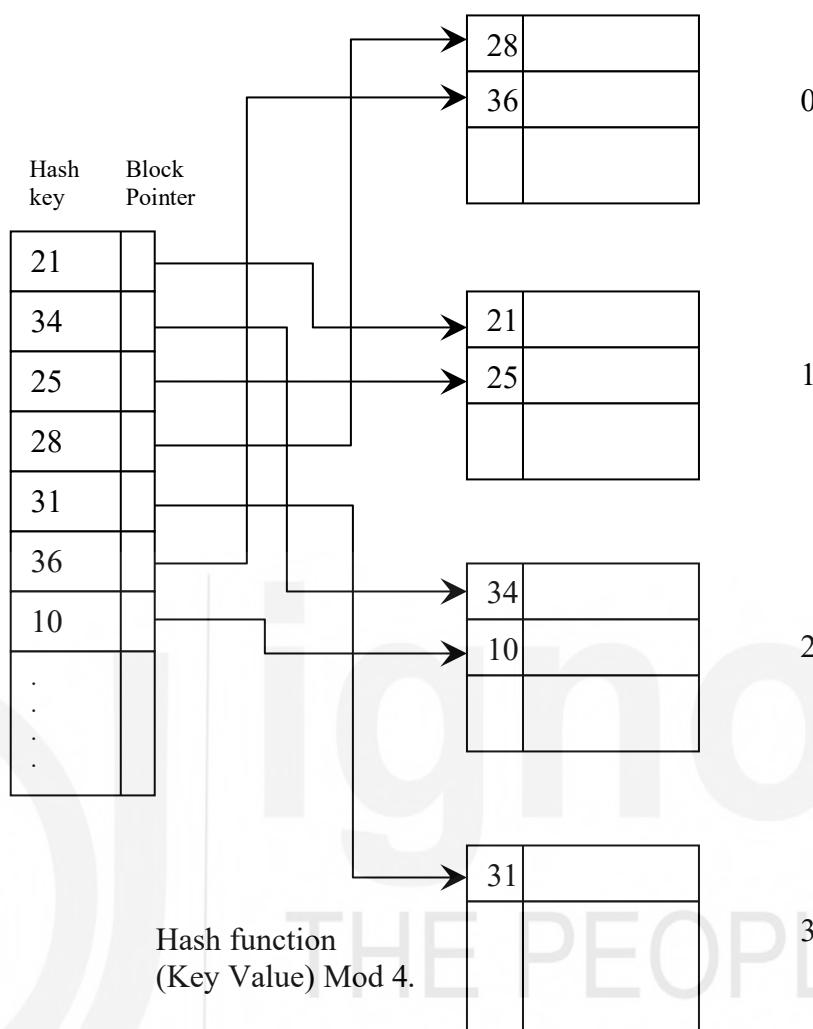


Figure 4.4: A Hashed file

+ Check Your Progress 1

- 1) List five operations on a sequential file Organisation. Comment on performance of each of these operations.
.....
.....
- 2) What are Direct-Access systems? What can be the various strategies to achieve this?
.....
.....
.....
- 3) What is file Organisation and what are the essential factors that are to be considered?
.....
.....
.....

4.5 INDEXES

One of the terms used during the file organisation is the term index. In this section, let us define this term in more detail.

Every printed book that you read, in general, have an index of keywords at the end. Notice that this index is a sorted list of keywords (index values) and page numbers (address) where the keyword can be found. In databases also an index is defined in a similar way, as the $\langle \text{index value}, \text{address} \rangle$ pair.

The basic advantage of having sorted index pages at the end of the book is that you can locate the description about a desired keyword in the book. You could have used the topic and sub-topic listed in the table of contents, but it is not necessary that the given keyword can be found there; also, they are not in any sorted sequence. If a keyword is not listed in index and table of contents, then you need to search each page of the book to find the required keyword, which is very cumbersome. Thus, an index at the back of the book helps in locating the required keyword references very easily in the book.

The same is true for the databases that have very large number of records. A database index allows fast search on the index value in database records. It will be difficult to locate an attribute value in a large database, if index on that attribute is not provided. In such a case the value is to be searched record-by-record in the entire database, which is cumbersome and time consuming. It is important to note that for a large database all the records cannot be kept in the main memory at a time, thus, data needs to be transferred from the secondary storage device, which is more time consuming.

An index entry consists of a pair consisting of index value and a list of pointers to disk blocks for the records that have that index value. An index contains such information for every stored value of index attribute. An index file is very small compared to a data file that stores a relation. Also index entries are ordered, so that an index can be searched using an efficient search method like binary search. In case an index file is very large, you can create a multi-level index, that is index on index. Multi-level indexes are defined later in this section.

There are many types of indexes those are categorised as:

Primary index	Single level index	Spare index
Secondary index	Multi-level index	Dense index
Clustering index		

A primary index is defined on the attributes *in the order of which the file is stored*. This field is called the ordering field. A primary index can be on the primary or candidate key of a file. If an index is on the ordering attributes, which are not candidate key attributes, then several records may be related to one ordering field value. This is called clustering index. It is to be noted that there can be only one physical ordering attribute or set of attributes for a file. Thus, a file can have either the primary index or clustering index, not both. Secondary indexes are defined on the non-ordering fields. Thus, there can be several secondary indexes in a file, but only one primary or clustering index.

Primary index

Primary index is a file that contains a sorted sequence of index records having two columns: the ordering key field; and a block address for that key field in the data file. The ordering key field for this index can be the primary key of the data file. Primary index contains one index entry of the ordering key field for each Block of data. An

entry in primary index file contains either the key value of the first record or the key value of last record, which are stored in that data block; and a pointer to that data block.

Let us discuss primary index with the help of an example. Let us assume a student database as (Assuming that one block stores only four student records). Figure 4.5 shows sample of this data file. The sample file is ordered on the attribute - enrolment number.

Block Number	Enrolment Number	Name	City	Programme
BLOCK 1	2109348	...	CHEENNAI	CIC
	2109349	...	CALCUTTA	MCA
	2109351	...	KOCHI	BCA
	2109352	...	KOCHI	CIC
BLOCK 2	2109353	...	VARANASI	MBA
	2238389	...	NEW DELHI	MBA
	2238390	...	VARANASI	MCA
	2238411	...	NEW DELHI	BCA
BLOCK 3	2238412	...	AJMER	MCA
	2238414	...	NEW DELHI	MCA
	2238422	...	MUMBAI	BSC
	2258014	...	MUMBAI	BCA
BLOCK 4	2258015	...	MUMBAI	BCA
	2258017	...	NEW DELHI	BSC
	2258018	...	MUMBAI	MCA
	2258019	...	LUCKNOW	MBA
...
BLOCK r	2258616	...	AJMER	BCA
	2258617	...	LUCKNOW	MCA
	2258618	...	NEW DELHI	BSC
	2318935	...	FARIDABAD	MBA
...
BLOCK N-1	2401407	...	BAREILLY	CIC
	2401408	...	BAREILLY	BSC
	2401409	...	AURANGABAD	BCA
	2401623	...	NEW DELHI	MCA
BLOCK N	2401666	...	MUMBAI	MCA
	2409216	...	LUCKNOW	MBA
	2409217	...	ALMORA	BCA
	2409422	...	MUMBAI	BSC

Figure 4.5: A Student file stored in the order of student enrolment numbers

The primary index on this file would be on the ordering field – enrolment number. The primary index on this file is shown in Figure 4.6. Please note the following points in the Figure 4.6.

- An index entry is defined as the attribute value, pointer to the block where that record is stored. The pointer physically is represented as the binary address of the block.
- Since there are four student records, which of the key value should be stored as the index value? We have used the first key value stored in the block as the index key value. This is also called the anchor value. All the records stored in the given block have ordering attribute value as the same or more than this anchor value.
- The primary index may be smaller in size, as it contains one index entry for each storage data block. Also notice that not all the records need to have an

entry in the index file. This type of index is called non-dense index. Thus, the primary index is non-dense index.

- To locate the record of a student whose enrolment number is 2238422, you need to find two consecutive entries of indexes such that index value $1 < 2238422 <$ index value 2. In the Figure 4.6, you can find the third and fourth index values as: 2238412 and 2258015 respectively satisfying the properties as above. Thus, the required student record must be found in Block 3.

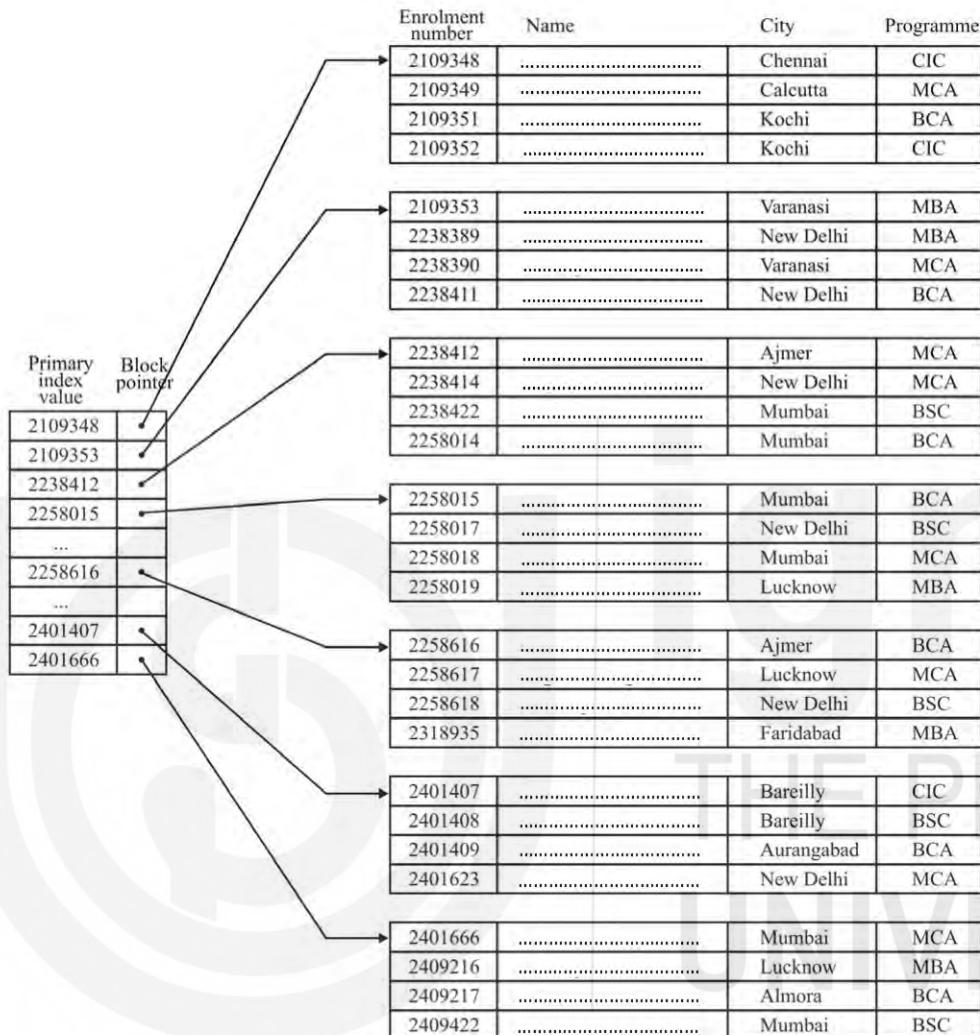


Figure 4.6: The Student file and the Primary Index on Enrolment Number

But does primary index enhance efficiency of searching? Let us explain this with the help of an example (Please note we will define savings in terms of the number of block transfers, as that is the most time-consuming operation during searching).

Example 1: An ordered student file (ordering field is enrolment number) has 20,000 records stored on a disk having the Block size as 1 K. Assume that each student record is of 100 bytes, the ordering field is of 8 bytes, and block pointer is also of 8 bytes, find how many block accesses on average may be saved on using primary index.

Answer:

Number of accesses without using Primary Index:

Number of records in the file = 20000

Block size = 1024 bytes

Record size = 100 bytes

Number of records per block = integer value of $[1024 / 100] = 10$

Number of disk blocks acquired by the file

$$\begin{aligned} &= [\text{Number of records / records per block}] \\ &= [20000/10] = 2000 \end{aligned}$$

Assuming a block level binary search, it would require $\log_2 2000$
= about 11 block accesses.

Number of accesses with Primary Index:

Size of an index entry = $8+8 = 16$ bytes

Number of index entries that can be stored per block
= integer value of $[1024 / 16] = 64$

Number of index entries = number of disk blocks = 2000

Number of index blocks = ceiling of $[2000 / 64] = 32$

Number of index block transfers to find the value in index blocks = $\log_2 32 = 5$

One block transfer will be required to get the data records using the index pointer after the required index value has been located.

So total number of block transfers with primary index = $5 + 1 = 6$.

Thus, the Primary index would save $11 - 6 = 5$ block transfers for the given size of data and index.

Is there any disadvantage of using primary index? Yes, a primary index requires the data file to be ordered, this causes problems during insertion and deletion of records in the file. This problem can be taken care of by selecting a suitable file organisation that allows logical ordering only.

Clustering Indexes.

It may be a good idea to keep records of the students in the order of the programme they have registered, as most of the data file accesses may require programme wise student data. A file can be ordered and physically stored on non-key attributes; an index that is created on such non-key attributes would have multiple records pointed to by a single index entry. Such index is called a clustering index. *Figure 4.7 and Figure 4.8* show the clustering indexes in the same file organised in different ways.

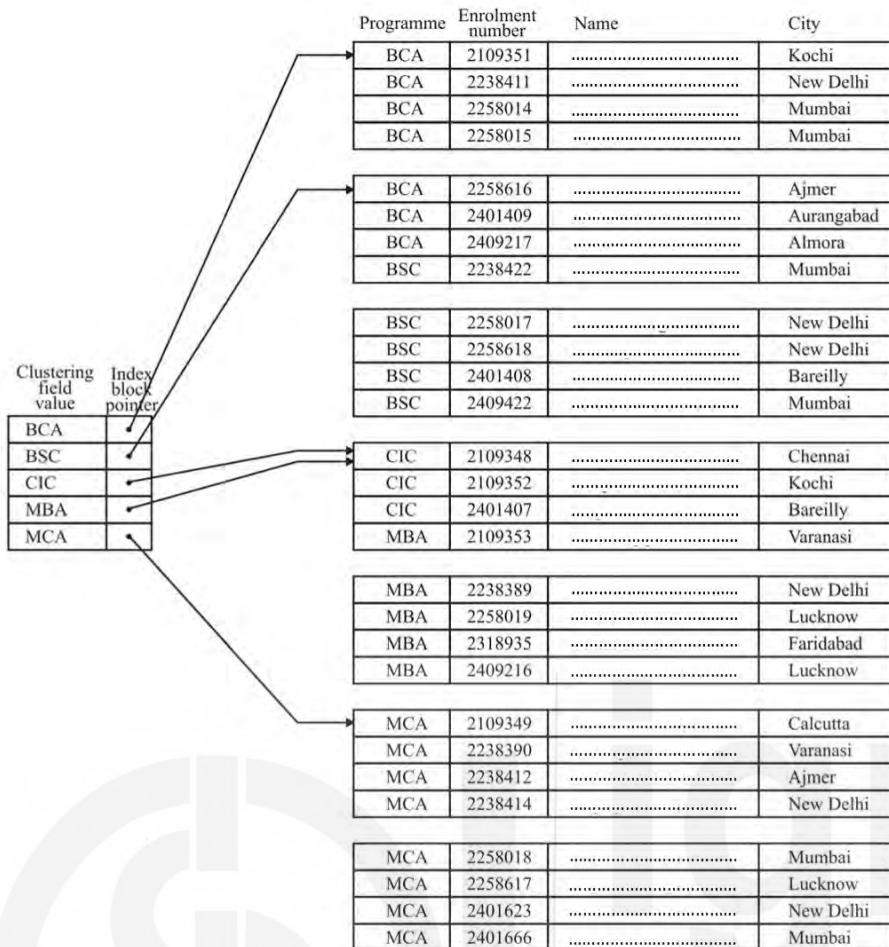


Figure 4.7: A clustering Index on Programme in the Student file

Please note the following points about the clustering index as shown in the *Figure 4.7*:

- The clustering index is an ordered file having the clustering index value and a block pointer to the first block where that clustering field value first appears.
- Clustering index is also a sparse index. The size of clustering index is smaller than primary index as far as number of entries is concerned.

Please note that in the *Figure 4.7*, the data file can have a single block in which data of students of multiple programmes are stored. You can improve upon this organisation by allowing only one Programme data in one block. Such an organisation and its clustering index is shown in the *Figure 4.8*:

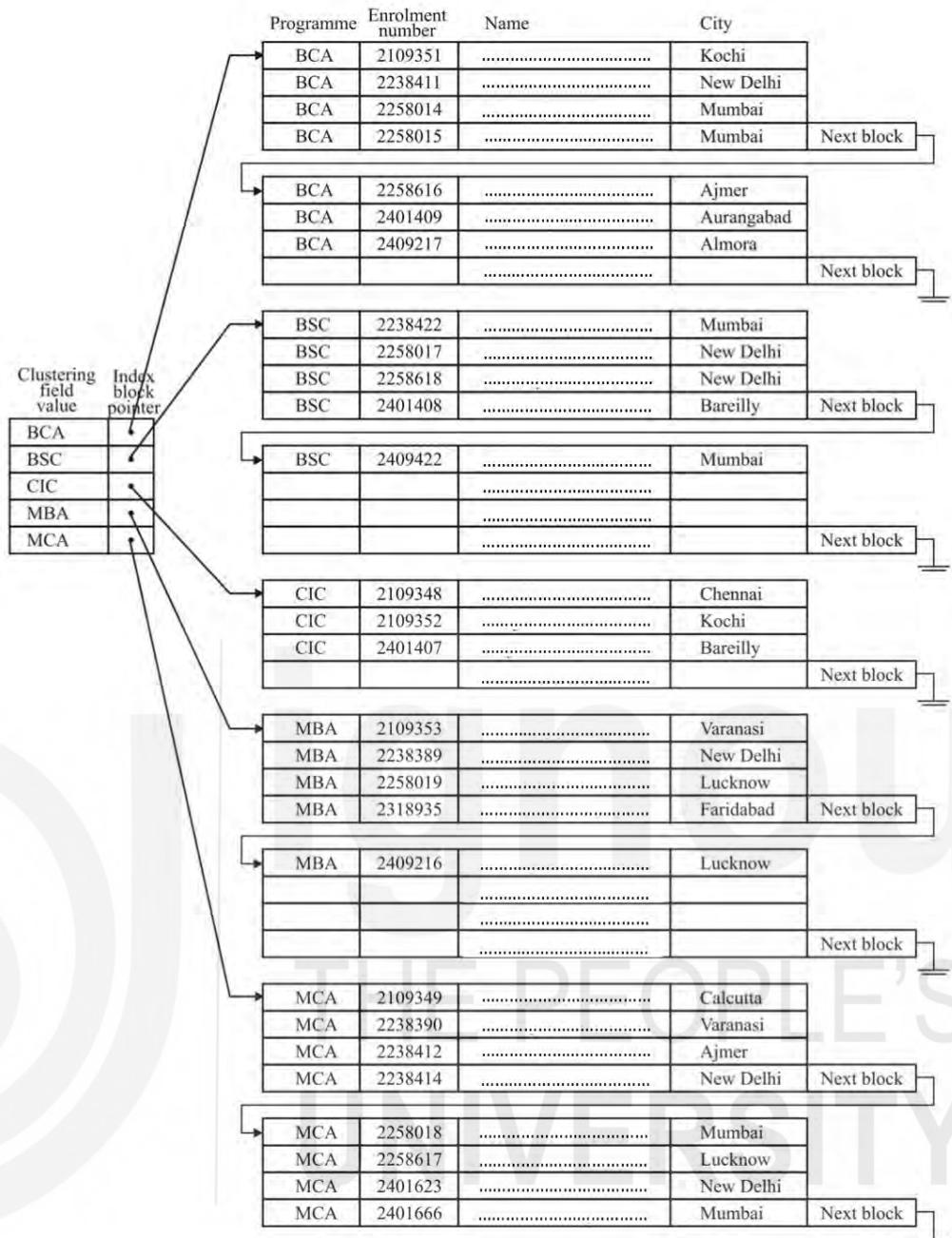


Figure 4.8: Clustering index with separate blocks for each clustering attribute value

Please note the following points for the above:

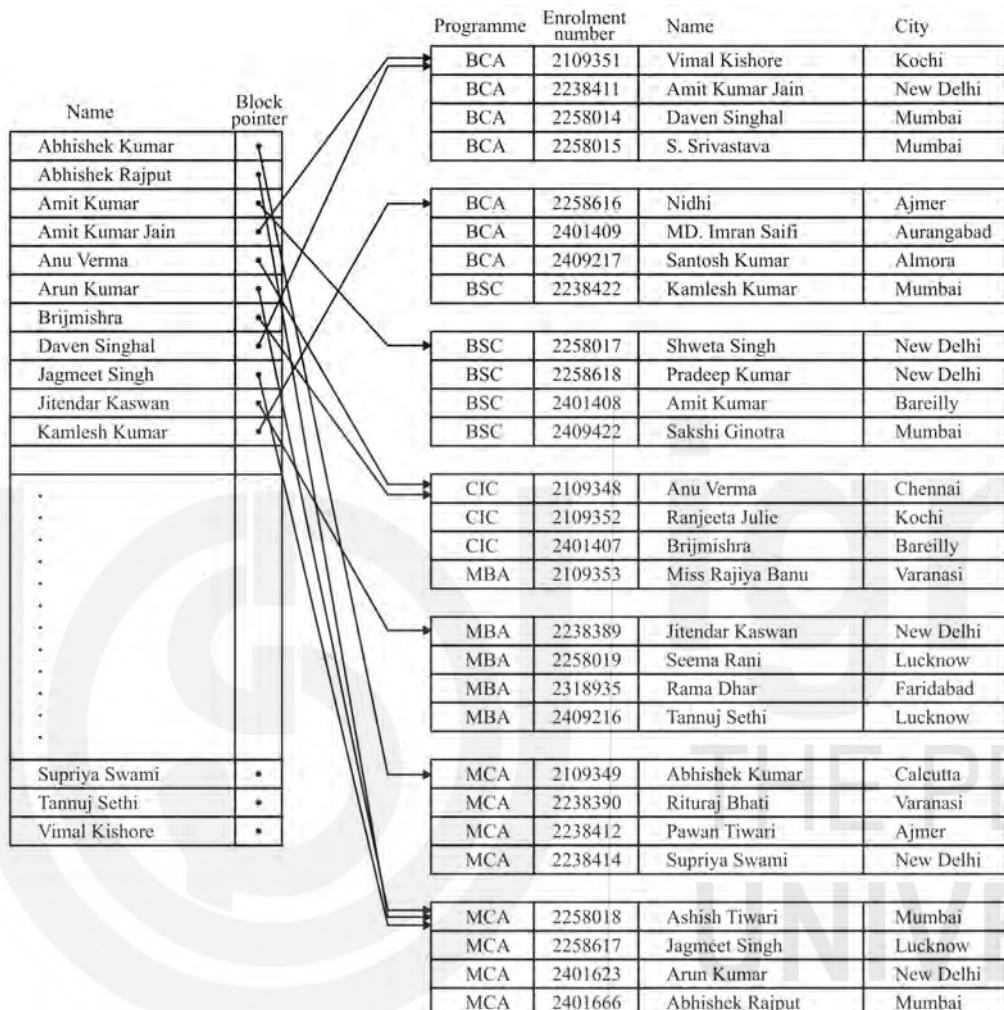
- Data insertion and deletion is easier than in the earlier clustering files, yet even now it is cumbersome.
- The blocks allocated for one index entry are in the form of linked list blocks.
- Clustering index is another example of a non-dense index as it has one entry for every distinct value of the clustering index attribute and not for every record in the file.

Secondary Indexes

Consider the student database and its primary and clustering index (only one will be applicable at a time). Now consider the situation when the database is to be searched or accessed in the alphabetical order of student names. Any search on a student name would require sequential data file search, as there is no ordering on student names. Thus, searching on student's name is going to be very time consuming. Such a search on an average would require reading of half of the total number of blocks. Thus, you need secondary indices in database systems. A secondary index is a file that contains records containing a secondary index field value which is not the ordering field of the data file, and a pointer to the block that contains the data record. Please note that

although a data file can have only one primary index (as there can be only one ordering of a database file), it can have many secondary indices.

Secondary index can be defined on an alternate key or non-key attributes. A secondary index that is defined on the alternate key will be dense while secondary index on non-key attributes would require a bucket of pointers for one index entry. Let us explain them in more detail with the help of *Figures 4.9*.



**Figure 4.9: A Dense Secondary Index on a non-ordering key field of a file
(The file is organised on the clustering field “Programme”)**

Please note the following in the *Figure 4.9*.

- The names in the data file are unique and thus are being assumed as the alternate key. Each name therefore is appearing as the secondary index entry.
- The pointers are block pointers, thus are pointing to the beginning of the block and not a record. For simplicity we have not shown all the pointers in the Figure 4.9.
- This type of secondary index file is dense index as it contains one entry for each record/distinct value.
- The secondary index is larger than the Primary index as we cannot use block anchor values here as the secondary index attributes are not the ordering attribute of the data file.
- To search a value in a data file using name, first the index file is (binary) searched to determine the block, where the record having the desired key value

can be found. Then this block is transferred to the main memory where the desired record is searched and accessed.

- A secondary index file, usually, has larger number of index entries than that of primary index. However, the secondary index improves the search time to a greater proportion than that of primary index. This is due to the reason - If primary index does not exist even then, you can perform binary search on the blocks of data records, as the records are ordered in the sequence of primary index value. However, if a secondary key does not exist then you may need to search the records sequentially. This fact is demonstrated with the help of Example 2.

Example 2: Let us reconsider the problem of Example 1 with a few changes. An un-ordered student file, which is not ordered on primary key, has 20,000 records stored on a disk having the Block size as 1 K. Assume that each student record is of 100 bytes, the secondary index field is of 8 bytes, and block pointer is also of 8 bytes, find how many block accesses on average may be saved on using secondary index on enrolment number.

Answer:

Number of accesses without using Secondary Index:

Number of records in the file = 20000

Block size = 1024 bytes

Record size = 100 bytes

Number of records per block = integer value of $[1024 / 100] = 10$

Number of disk blocks acquired by the file

$$= [\text{Number of records} / \text{records per block}]$$

$$= [20000/10] = 2000$$

Since the file is un-ordered any search on an average will require about half of the above blocks to be accessed. Thus, average number of block accesses = 1000

Number of accesses with Secondary Index:

Size of an index entry = $8+8 = 16$ bytes

Number of index entries that can be stored per block

$$= \text{integer value of } [1024 / 16] = 64$$

Number of index entries = number of records = 20000

Number of index blocks = ceiling of $[20000 / 64] = 320$

Number of index block transfers to find the value in index blocks

$$= \text{ceiling of } [\log_2 320] = 9$$

One block transfer will be required to get the data records using the index pointer after the required index value has been located. So total number of block transfers with secondary index = $9 + 1 = 10$

Thus, the Secondary index would save about 1990 block transfers for the given case. This is a huge saving compared to primary index. Please also compare the size of secondary index to primary index.

Let us now see an example of a secondary index that is on an attribute that is not an alternate key.

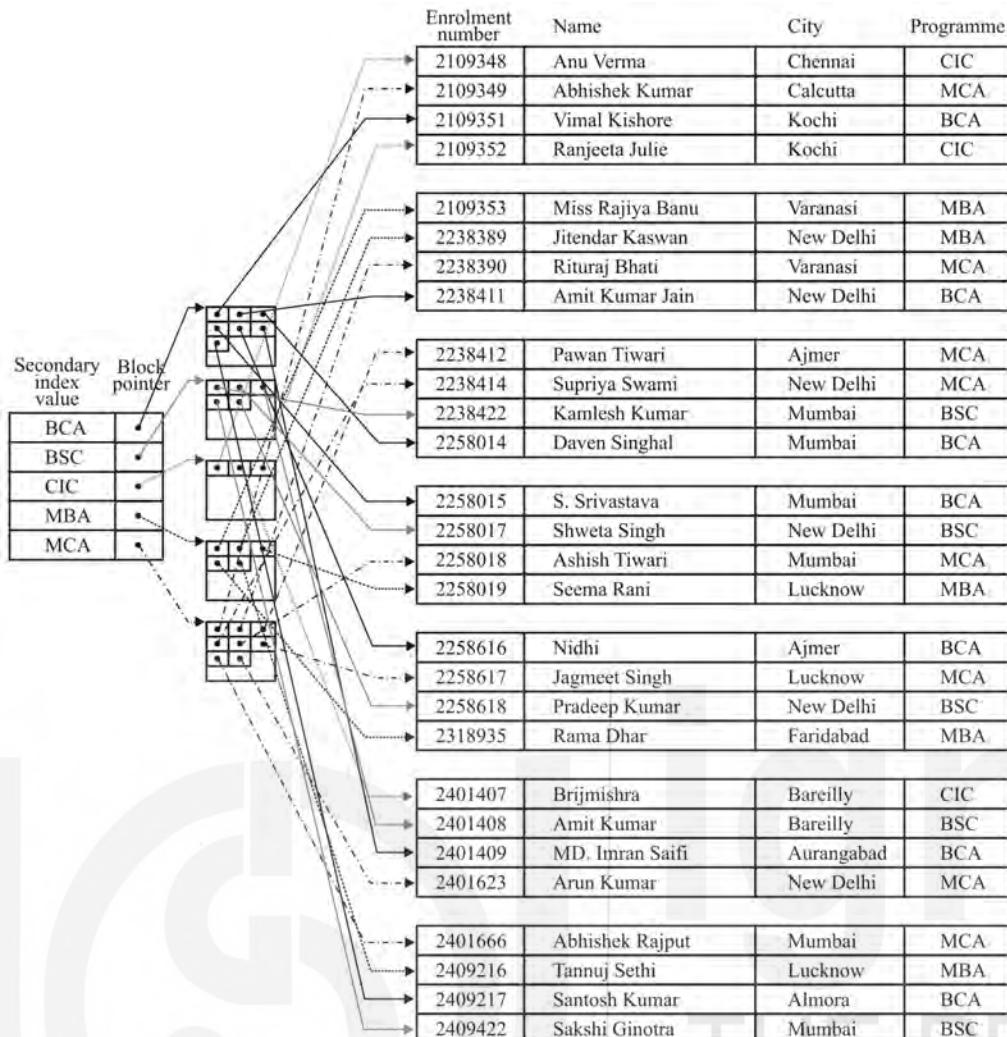


Figure 4.10: An example of Secondary Index with single level of indirection

(Index entries are of fixed length and have unique field values)

(The file is organised on the primary key)

A secondary index that needs to be created on a field that is not a candidate key can be implemented using several ways. We have shown here the way in which a block of pointer records is kept for implementing such index. This method allows the index entries to be of fixed length. It also allows only a single entry for value of the indexing attribute. In addition, the level of indirection allows multiple index pointers to be stored in a single block of data. The algorithms for searching the index, inserting and deleting new values into an index are very simple in such a scheme. Thus, this is the most popular scheme for implementing such secondary indexes.

Sparse and Dense Indexes

As discussed earlier an index is defined as the ordered $\langle \text{index value}, \text{address} \rangle$ pair. These indexes in principle are the same as that of indexes used at the back of the book. The key ideas of the indexes are:

- They are sorted on the order of the index value (ascending or descending) as per the choice of the creator.
- The indexes are logically separate files (just like separate index pages of the book).
- An index is primarily created for fast access of information.
- The primary index is the index on the ordering field of the data file, whereas a secondary index is the index on any other field, thus, is more useful.

But what are sparse and dense indexes?

A dense index contains one index entry for every value of the indexing attributes, whereas a sparse index also called non-dense index contains few index entries out of the available indexing attribute values. For example, the primary index on enrolment number is sparse, while secondary index on student name is dense.

Multilevel Indexing Scheme

For small files, indexing scheme keeps the address of the block file in each index entry. Such indices would be small and can be processed efficiently in the main memory. However, for a large file the size of index can also be very large. In such a case, you can create indexes at several levels, with the last level pointing to the data records. Figure 4.11 shows this scheme.

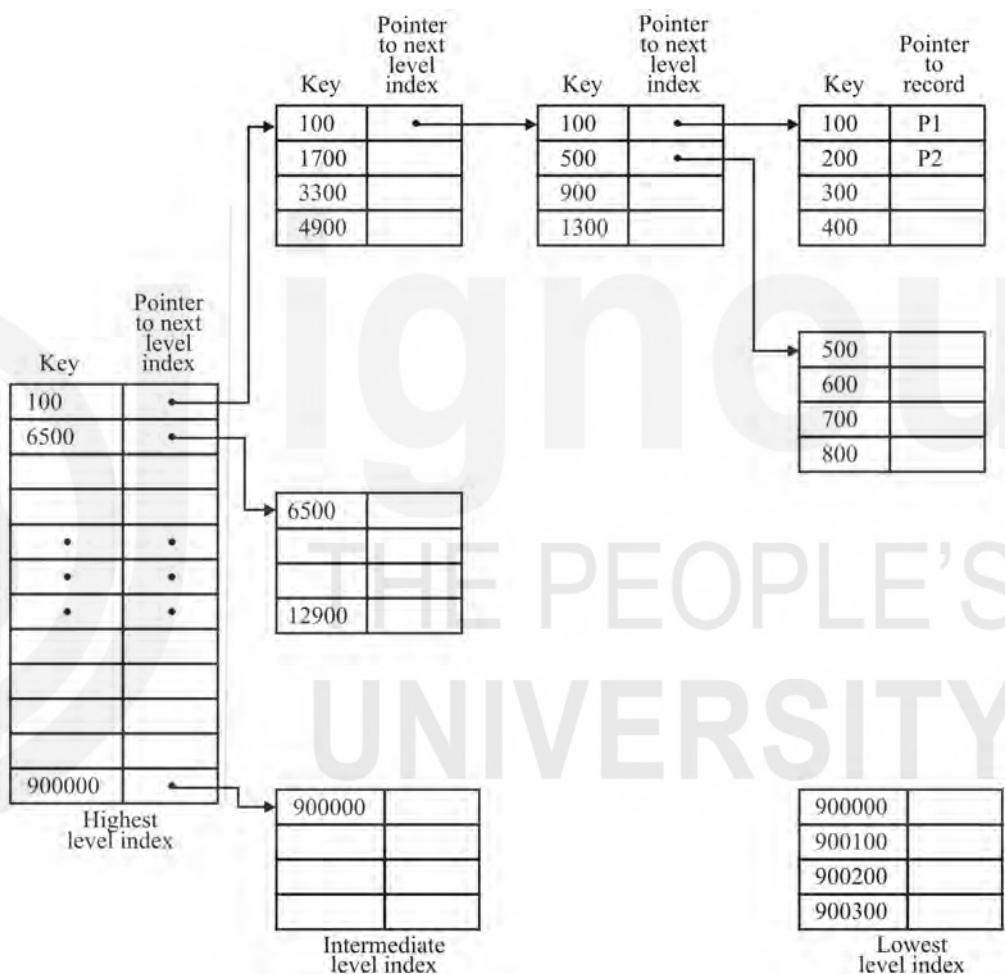


Figure 4.11: Hierarchy of Indexes

Please note the following relating to multi-level indexes:

- The lowest level index, as shown in Figure 4.11, has one entry for each record, though in different index block. Thus, lowest level index occupies the maximum space.
- Thus, maintenance of the multi-level indexes is expensive, as multiple indexes are to be maintained for every insertion and deletion of record.

After discussing so much about the indexes let us now turn our attention to how an index can be implemented in a database. The indexes are implemented through B-Tree. The following section examines the index implementation in more detail.

+ Check Your Progress 2

- 1) A file contains 40,000 student records of 200 bytes each having the 1 KB as the size of a block. What would be the size of primary index, if the size of primary

key is 4 byte and the block address is 8 bytes? What would be the average number of block accesses to access a record using this Index?

.....

- 2) What would be the size of the size of secondary index for the student file as stated in question 1 if it has a secondary index on its alternate key of size 8 bytes. What would be the average number of block accesses to access a record using this secondary Index?
-
-
-

- 3) Which of the following indexes are dense indexes? Give reasons.
 (i) Primary Index (ii) Clustering Index (iii) Secondary Index on alternative key
 (iv) Secondary index on non-key attributes with unique attribute values
-
-
-

4.6 IMPEMENTING INDEX USING TREE STRUCTURE

Let us discuss the data structure that is used for creating indexes.

Can we use Binary Search Tree (BST) as Indexes?

Let us first reconsider the binary search tree. A BST is a data structure that has a property that all the keys that are to the left of a node are smaller than the key value of the node and all the keys to the right are larger than the key value of the node.

To search a typical key value, you start from the root and move towards left or right depending on the value of key that is being searched. Since an index is a *<value, address>* pair, thus while using BST, you need to use the value as the key and address field must also be specified in order to locate the records in the file that is stored on the secondary storage devices. The following figure demonstrates the use of BST index for a University where a dense index exists on the enrolment number field.

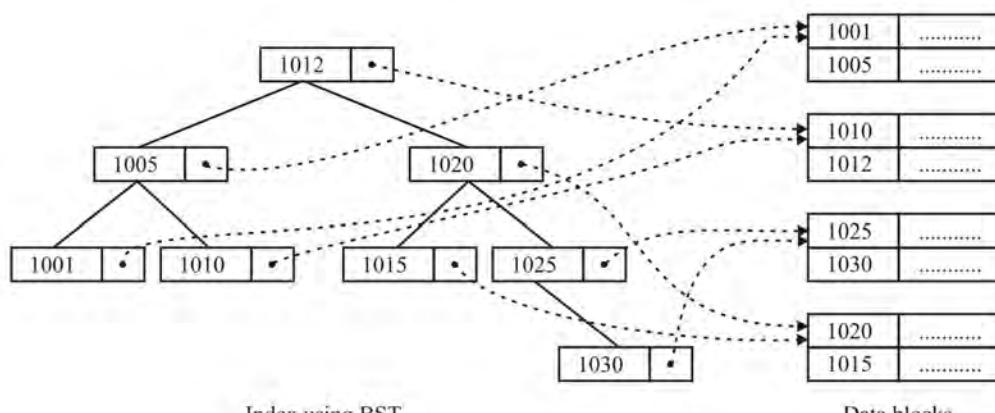


Figure 4.12: The Index structure using Binary Search Tree

Please note in Figure 4.12 that a key value is associated with a pointer to a record. A record consists of the key value and other information fields. Please note that a node on BST stores the *<key value, address>* pair.

Now, let us examine the suitability of BST as a data structure to implement index. A BST as a data structure is suitable for an index if the complete index is contained in the primary memory. However, indexes are quite large in nature and require a combination of primary and secondary storage. Therefore, you can use B-Tree data structure to implement the index.

A B-Tree as an index has two advantages:

- It is completely balanced
- Each node of B-Tree can have a number of keys. Ideal node size would be if it is somewhat equal to the block size of secondary storage.

The question that needs to be answered here is: what should be the order of B-Tree for an index? The suggested order is from 80-200 depending on various index structures and block size.

B-tree is a data structure, which was proposed by R. Bayer and E. McCreight of Bell Scientific Research Labs in 1970. The B-Tree and its variants are secondary storage structures and have been found to be very useful for implementing indexes. An N order B-tree has:

- A node of B-tree of order N can have children/paths in the range - ceiling of $[N/2]$ to N. However, the root node of the tree can have 2 to N children/paths.
- Each node can have one fewer key than the number of children/paths, but a maximum of $N-1$ keys can be stored in a node.
- The keys are normally arranged in a node in an increasing order.
- If a new key is inserted into a full node of order N (that is it already contains $N-1$ keys), then on addition of another key value, the node would have $N+1$ paths (N keys). This node is split into two nodes with the median key moved to the parent of this node. In case the node which is being split is the root node, the root node is split into two nodes, but a new root node is created using the median key of the node to be split.
- B-tree does not allow any empty sub-tree, therefore, all the leaves of B-tree are at the same level. Therefore, a B-tree is completely balanced tree.

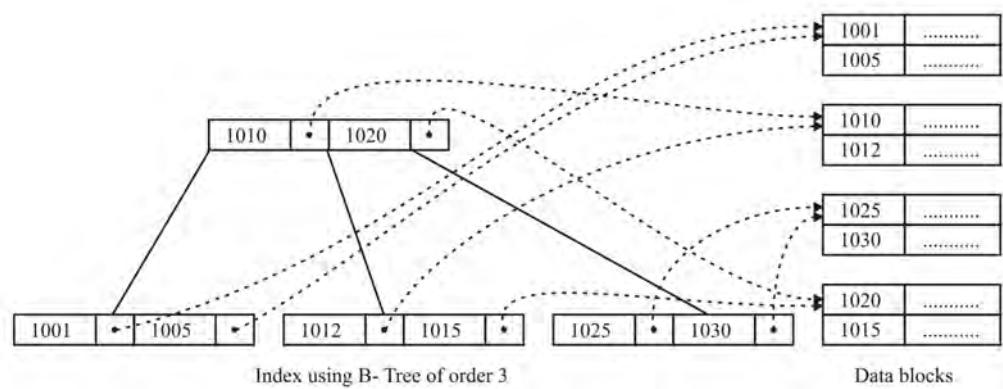


Figure 4.13: A B-Tree as an index

A B-Tree index is shown in Figure 4.13. The B-Tree has a very useful variant called B+Tree, which have all the key values at the leaf level also, in addition to the higher level. For example, the key value 1010 in *Figure 12* will also exist at leaf level. In addition, these lowest level leaves are linked through pointers. Thus, B+tree is a very

4.7 MULTI-KEY FILE ORGANISATIONS

Till now we have discussed file organisations having the single access key. But is it possible to have file organisations that allows access of records on more than one key fields? This section discusses the two file organisations that allow multiple access paths, with each path having a different key. These are called multi-key file Organisations. These file organisations, in general, are part of a real database management system. Two of the commonest techniques for this Organisation are:

- Multi-list file Organisation
- Inverted file Organisation

Let us discuss these techniques in more detail. But first let us discuss the need for the Multiple access paths.

4.7.1 Multiple Access Paths

In practice, most of the online information systems require the support of multi-key files. For example, consider a banking database application having many kinds of users such as:

- Teller
- Loan officers
- Branch manager
- Account holders

All these users access the bank data however in a different way. Let us assume a sample data format for the Account relation in a bank as:

Account Relation:

Account Number	Account Holder Name	Branch Code	Account type	Balance	Permissible Loan Limit

A teller may access the record above to check the balance at the time of withdrawal. S/he needs to access the account based on branch code and account number. A loan approver may be interested in finding the potential customer by accessing the records in decreasing order of permissible loan limits. A branch manager may need to find the top ten most preferred customers in each category of account, so s/he may access the database in the order of account type and balance. The account holder may be interested in her/his own record. Thus, all these applications are trying to refer to the same data but using different key values. Thus, all the applications as above require the database file to be accessed in different format and order.

Multiple indexes can be used to access a data file through multiple access paths. In such a scheme only one copy of the data is kept, only the number of paths is added with the help of indexes. Let us discuss two important approaches, viz. multi-list file organisation and Inverted file organisation.

4.7.2 Multi-list file Organisation

This file organisation, as the name suggests, consists of multiple lists or indexes. The records in each list are linked from the index value. The linking of records, in general, is done in the sorted sequence of the key attribute to facilitate searching, insertion and deletion operations. The following example explains the multi-list file organisation.

A sample data of employees of an organisation is given in *Figure 4.14*. Assume that the Empid is the key attribute. You can create multiple index lists using this data.

Assumed Record Number	Employee id (Empid)	Employee Name	Job Title	Highest Qualification	Gender (Female F /Male M)	City of posting	Married - M/ Single - S	Salary per month
A	795	Praveen	Engineer	B. Tech.	M	Dehradun	S	16,200/-
B	495	Rohini	Manager	B. Tech.	F	Dehradun	M	19,000/-
C	905	Rishika	Manager	MCA	F	Jaipur	S	17,100/-
D	705	Gaurav	Engineer	B. Tech.	M	Jaipur	M	13,200/-
E	595	Dipti	Manager	MCA	F	Jaipur	S	14,100/-

Figure 4.14: Sample Employee Data

The primary link order (in the order of primary key Empid) would be:

B (495), E(595), D(705), A(795), C(905)

The primary index for this file would be:

≥ 500 but < 700
 ≥ 700 but < 900
 ≥ 900 but < 1100

The index file for the example data as per Figure 4.14 is shown in *Figure 4.15*.

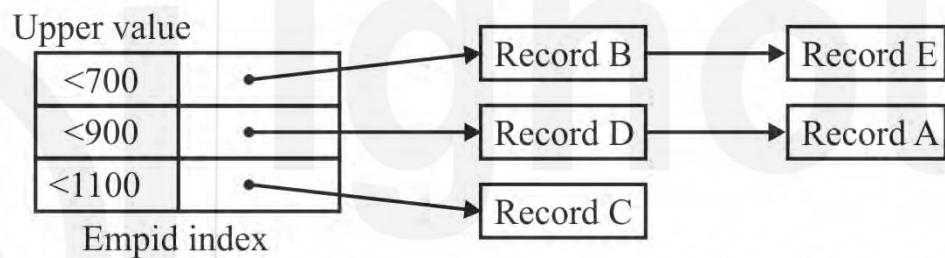


Figure 4.15: Linking together all the records in the same index value.

Please note that in the *Figure 4.15*, those records that fall in the same index value range of Empid are linked together. These lists are smaller than the total range, which will improve search performance.

This file can be supported by many more indexes that will enhance the search performance on various fields, thus, creating a multi-list file organisation. *Figure 4.16* shows various indexes and lists corresponding to those indexes. For simplicity we have just shown the links and not the complete record. Please note that nodes in the original file are assumed to be in the order of Empid's.

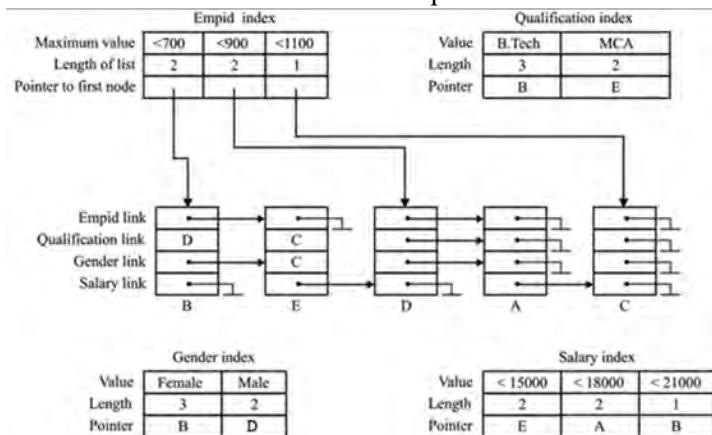


Figure 4.16: Multi-list representation for Figure 4.14

In Figure 4.16, the qualification index can be read as:

B. Tech. Starts at B → D → A.

MCA Starts at E → C.

Gender Index:

Female: Starts at B, → E, → C.

Male: Starts at B, → D, → A.

An interesting addition that can be done in the indexing scheme of multi-list organisation is that an index entry may contain the length of each sub-list and index entry should have a pointer to the first record of that list. The length information is useful when the query contains a Boolean expression. For example, if you need to find the list of Female employees who have MCA qualifications, you can find the results in two ways. Either you go to the Gender index and search the Female index list for MCA qualification, or you search the qualification index to find MCA list and in MCA list search for Female candidates. Since the size of MCA list is 2 while the size of Female list is 3 so the preferable search will be through smaller MCA index list. Thus, the information about the length of the list may help in reducing the search time in the complex queries. Performance of this structure is not good under heavy insertion and deletion of records. However, it is a good structure for searching records in case the appropriate index exists.

4.7.3 Inverted File Organisation

Inverted file organisation is one file organisation where the index structure is most important. In this organisation the basic structure of file records does not matter much. This file organisation is somewhat similar to that of multi-list file organisation with the key difference that in multi-list file organisation index points to a list, whereas in inverted file organisation the index itself contains the list. Thus, maintaining the proper index through proper structures is an important issue in the design of inverted file organisation. Let us show inverted file organisation with the help of data given in *Figure 4.14*.

Let us assume that the inverted file organisation for the data shown contains dense index. *Figure 4.17* shows how the data can be represented using inverted file organisation.

Empid index		Qualification index		Salary index	
495	B	B.Tech	B, D, A	13200	D
595	E	MCA	E, C	14100	E
705	D	Gender index		16200	A
795	A	Female	B, E, C	17100	C
905	C	Male	D, A	19000	B

Figure 4.17: Some of the indexes for fully inverted file

Please note the following points for the inverted file organisation:

- The index entries are of variable lengths as the number of records with the same key value is changing, thus, maintenance of index is more complex than that of multi-list file organisation.
- The queries that involve Boolean expressions require accesses only for those records that satisfy the query in addition to the block accesses needed for the indices. For example, the query about Female, MCA employees can be solved by the Gender and Qualification index. You just need to take intersection of record numbers on the two indices. (Please refer to *Figure 4.17*). Thus, any complex query requiring Boolean expression can be handled easily through the help of indices.

4.8 IMPORTANCE OF FILE ORGANISATION IN DATABASES

To implement a database efficiently, there are several design tradeoffs needed. One of the most important ones is the file Organisation. For example, if there were to be an application that required only sequential batch processing, then the use of indexing techniques would be pointless and wasteful.

There are several important consequences of an inappropriate file Organisation being used in a database. The wrong file Organisation will result in :

- much larger processing time for retrieving or modifying the required record.
- undue disk access that could stress the hardware.

Needless to say, there could be many undesirable consequences at the user level, such as making some applications impractical.

+ Check Your Progress 3

- 1) What is the difference if indexes are implemented using binary search tree or using B-tree?

.....
.....

- 2) What are the advantages of using B+ tree index over B tree index if the supported file organisation is required to access the records sequentially too.

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

- 3) What is the need of a multi-list organization? What is the advantage of using number of records in an index entry?

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

4.9 SUMMARY

In this unit, we discussed the physical database design issues in which we had addressed the file Organisation and file access method. The unit also discusses different types of file organization giving their advantages and their disadvantages.

An index is an important component of a database system, as one of the key requirements of DBMS is efficient access to data. This unit explains various types of indexes that may exist in database systems. Some of these are: Primary index, clustering index and secondary index. The secondary index results in better search performance but adds on the task of index updating. This unit also discusses two multi-key file organisations viz. multi-list and inverted file organisations. These are very useful for improving query performance.

4.10 SOLUTIONS / ANSWERS

Check Your Progress 1

1)

Operation	Comments
File Creation	It will be efficient if transaction records are ordered by record key
Record Location	As it follows the sequential approach it is inefficient. On an average, half of the records in the file must be processed to locate a record.
Record Creation	It will require you to browse through all the records to check if such record already exists or not. Thus, entire file must be read and written. Efficiency improves if a group of records are created together. This operation could be combined with deletion and modification transactions to achieve greater efficiency.
Record Deletion	Entire file must be read and write. Efficiency improves with greater number of deletions. This operation could be combined with addition and modification transactions to achieve greater efficiency.
Record Modification	Very efficient if the number of records to be modified is high and the records in the transaction file are ordered by the record key.

- 2) Direct-access systems do not search the entire file; rather they move directly to the record, which is to be accessed. To be able to achieve this, several strategies like relative addressing, hashing and indexing can be used.
- 3) It is a technique for physically arranging the records of a file on secondary storage devices. When choosing the file Organisation, you should consider the following factors:
1. Data retrieval speed
 2. Data processing speed
 3. Efficient use of storage space
 4. Protection from failures and data loss
 5. Scalability
 6. Security

Check Your Progress 2

- 1) Number of accesses without using Primary Index:

Number of records in the file = 40000

Block size = 1024 bytes

Record size = 200 bytes

Number of records per block = integer value of $[1024 / 200] = 05$

Number of disk blocks acquired by the file

$$= [\text{Number of records} / \text{records per block}]$$

$$= [40000/05] = 8000$$

The file is in the order of primary index, so binary search can be employed to access the file. As $2^{13} = 8192$. Thus, about 13 Block accesses would be required to locate the block containing the desired record.

Number of accesses with Primary Index:

Size of an index entry = $4+8 = 12$ bytes

Number of index entries that can be stored per block
= integer value of $[1024 / 12] = 85$

Number of index entries = number of disk Blocks of file = 8000

Number of index blocks = ceiling of $[8000 / 85] = 94$

Number of index block transfers to find the value in index blocks
= ceiling of $\lceil \log_2 94 \rceil = 7$

One block transfer will be required to get the data records using
the index pointer after the required index value has been
located. So total number of block transfers with secondary
index = $7 + 1 = 8$

Thus, the Primary index would save about 5 block transfers for the given case.

2) **Number of accesses without using Secondary Index:**

Number of disk Blocks = 8000 (as computed in Question 1)

The file is in the order of primary index, so search on alternative key
would require on an average half of the Blocks to be searched.

Average bumber of block transfer (without secondary index) = 4000

Number of accesses with Secondary Index on alternate key (dense index):

Size of an index entry = $8+8 = 16$ bytes

Number of index entries that can be stored per block
= integer value of $[1024 / 16] = 64$

Number of index entries = number of records = 40000

Number of index blocks = ceiling of $[40000 / 64] = 625$

Number of index block transfers to find the value in index blocks
= ceiling of $\lceil \log_2 625 \rceil = 10$

One block transfer will be required to get the data records using
the index pointer after the required index value has been
located. So total number of block transfers with secondary
index = $10 + 1 = 11$

Thus, the Secondary index would save a large number of block transfers.

(i) Sparse as only one entry per Block of data.

(ii) Dense, as one index entry for each attribute value, sparse if multi-level
index is used.

(iii) Dense, as one index entry for each attribute value

(iv) Dense, as one index entry for each unique attribute

Check Your Progress 3

- 1) In a B+ tree the leaves are linked together to form a sequence set; interior
nodes exist only for the purposes of indexing the sequence set (not to index
into data/records). The insertion and deletion algorithm differ slightly.
- 2) Sequential access to the keys of a B-tree is much slower than sequential
access to the keys of a B+ tree, since the latter have all the keys linked in
sequential order in the leave.
- 3) The multi-list organization enhances the query answering capability of
databases on multiple key values. The number of records in an index entry
determines the length of the index on a specific attribute value. This enhances
the performance of searching when more than one search terms are used, as
you can select the smaller list and then apply second search term on it.

UNIT 5 DATABASE INTEGRITY, FUNCTIONAL DEPENDENCY AND NORMALISATION

Structure	Page Nos.
5.0 Introduction	
5.1 Objectives	
5.2 Database Integrity	
5.2.1 The Keys	
5.2.2 Referential Integrity	
5.2.3 Entity Integrity	
5.3 Redundancy and Associated Problems	
5.4 Functional Dependencies	
5.5 Normalisation Using Functional Dependencies	
5.5.1 The First Normal Form	
5.5.2 The Second Normal Form	
5.5.3 The Third Normal Form	
5.5.4 Boyce Codd Normal Form	
5.6 Desirable Properties of Decomposition	
5.7 Rules of Data Normalisation	
5.7.1 Eliminate Repeating Groups	
5.7.2 Eliminate Redundant Data	
5.7.3 Eliminate Columns Not Dependent on Key	
5.8 Summary	
5.9 Answers/Solutions	

5.0 INTRODUCTION

The first block of this course discusses about the database concept, relational algebra, Entity relationship model and integrity constraints in the context of relational database design. One of the key aspects of database design is to create relations without any data redundancy.

This unit first explains the concept of entity and referential integrity. It also explains the anomalies in a relational database system. To remove anomalies is through decomposing the database, you need to decompose relations into smaller relations, which are free of those anomalies. This decomposition may be lossless and dependency preserving. The Unit explains the concept of functional dependency, which is the basis of lossless decomposition of a relation into smaller relations.

The unit deals with the standard form of database relations and discusses the process of decomposing relations into different normal forms up to BCNF. The higher normal forms are discussed in the next unit.

5.1 OBJECTIVES

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

- Explain entity integrity and referential integrity constraints of a relation;
- Explain various anomalies that exist in a database system;
- Define the desirable properties of decomposing a relation;
- Define and use functional dependencies to normalize database.

5.2 DATABASE INTEGRITY

Database integrity refers to maintaining the consistency of data in the database. Data integrity relates to the correctness of data and often is implemented using constraints on attributes in one or more relations. In this section, we will discuss more about two important integrity constraints of a database: the entity integrity constraint and the referential integrity constraint. To define these two, let us once again define the term Key with respect to a Database Management System.

5.2.1 The Keys

Candidate Key: In a relation R, a candidate key for R is a subset of the set of attributes of R, which have the following two properties:

- | | |
|------------------------|---|
| (1) <i>Uniqueness</i> | No two distinct tuples in R have the same value for the candidate key |
| (2) <i>Irreducible</i> | No proper subset of the candidate key has the uniqueness property. |

Every relation must have at least one candidate key which cannot be reduced further. Duplicate tuples are not allowed in relations. Any candidate key can be a composite key also. For Example, (student-id + course-id) together can form the candidate key of a relation called *Result* (*student-id, course-id, marks*).

Let us summarise the properties of a candidate key.

- A candidate key must be unique and irreducible
- A candidate may involve one or more than one attributes. A candidate key that involves more than one attribute is said to be composite.

But why are we interested in candidate keys?

Candidate keys are important because they provide the basic **tuple-level identification** mechanism in a relational system. For example, if the enrolment number is the candidate key of a STUDENT relation, then the answer of the query: "Find student details from the STUDENT relation having enrolment number A0123" will output at most one tuple.

Primary Key

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

Foreign Keys

Let us first give you the basic definition of foreign key.

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

1. There exists a relation R1 (R1 and R2 not necessarily distinct) with a candidate key, and
2. For all time, each value of a foreign key in the current state or instance of R2 is identical to the value of Candidate Key in some tuple in the current state of R1.

The definition above seems to be very heavy. Therefore, let us define it in more practical terms with the help of the following example.

Example 1: Assume that in an organisation, an employee may perform different roles in different projects. Say, XYZ is doing coding in one project and designing in another. Assume that the information is represented by the organisation in three different relations named EMPLOYEE, PROJECT and ROLE. The ROLE relation describes the different roles required in any project.

Assume that the relational schema for the above three relations are:

EMPLOYEE (EMPID, Name, Designation)
PROJECT (PROJID, Proj_Name, Details)
ROLE (ROLEID, Role_description)

In the relations above EMPID, PROJID and ROLEID are unique and not NULL. As you can clearly observe, you can identify the complete instance of the entity set employee through the attribute EMPID. Thus, EMPID is the primary key of the relation EMPLOYEE. Similarly, PROJID and ROLEID are the primary keys for the relations PROJECT and ROLE respectively.

Let ASSIGNMENT is a relationship between entities EMPLOYEE and PROJECT and ROLE, describing which employee is working on which project and what the role of the employee is in the respective project. Figure 5.1 shows part of E-R diagram for these entities and relationships (for simplicity, no attribute has been shown).

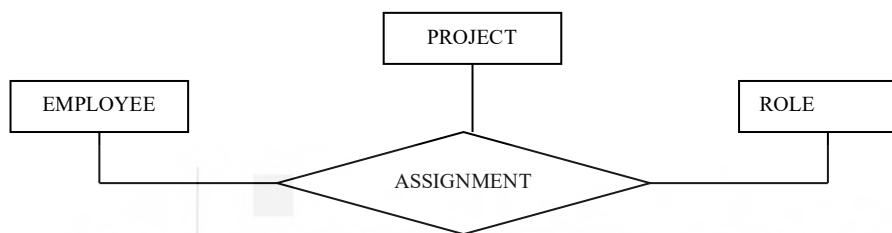


Figure 5.1: E-R diagram for employee role in development

Let us consider sample relation instances as:

EMPLOYEE

EMPID	Name	Designation
101	XYZ	Analyst
102	ABC	Receptionist
103	ARVIND	Manager

PROJECT

PROJID	Proj_name	Details
TCS	Traffic Control System	For traffic shaping.
LG	Load Generator	To simulate load for input in TCS.
B++1	B++ TREE	ISS/R turbo sys

ROLE

ROLEID	Role_description
1000	Design
2000	Coding
3000	Marketing

ASSIGNMENT

EMPID	PROJID	ROLEID
101	TCS	1000
101	LG	2000
102	B++1	3000

Figure 5.2: An example relation

You can define the relational scheme for the relation ASSIGNMENT as follows:

ASSIGNMENT (EMPID, PROJID, ROLEID)

Please note now that in the relation ASSIGNMENT (as per the definition of foreign key to be taken as R2) EMPID is the foreign key in ASSIGNMENT relation; it references the relation EMPLOYEE (as per the definition to be taken as R1) where EMPID is the primary key. Similarly, PROJID and ROLEID in the relation ASSIGNMENT are **foreign keys** referencing the relation PROJECT and ROLE respectively.

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

5.2.2 Referential Integrity

It can be simply defined as:

The database must not contain any unmatched foreign key values.

The term “unmatched foreign key value” means a foreign key value for which there does not exist a **matching value** of the relevant candidate key in the relevant target (referenced) relation. For example, any value existing in the EMPID attribute in ASSIGNMENT relation must exist in the EMPLOYEE relation. That is, the only EMPIDs that can exist in the EMPLOYEE relation are 101, 102 and 103 for the present state/ instance of the database given in *Figure 5.2*. If we want to add a tuple with EMPID value 104 in the ASSIGNMENT relation, it will cause violation of referential integrity constraint. Logically it is obvious, after all the employee 104 does not exist, so how can s/he be assigned any work.

Database modifications can cause violations of referential integrity. We list here the referential action that you may specify for each type of database modification to preserve the referential-integrity constraint:

Delete

During the deletion of a tuple two cases can occur:

Deletion of tuple in relation having the foreign key: In such a case simply delete the desired tuple. For example, in ASSIGNMENT relation you can easily delete the first tuple.

Deletion of the target of a foreign key reference: For example, an attempt to delete an employee tuple in EMPLOYEE relation whose EMPID is 101. This employee appears not only in the EMPLOYEE but also in the ASSIGNMENT relation. Can this tuple be deleted? If you delete the tuple in EMPLOYEE relation then two unmatched tuples are left in the ASSIGNMENT relation, thus causing violation of referential integrity constraint. Thus, the following two choices exist for such deletion:

RESTRICT – The delete operation is “restricted” to only the case where there are no matching tuples in the referencing relation. For example, you can delete the EMPLOYEE record of EMPID 103 as no matching tuple in ASSIGNMENT but not the record of EMPID 101.

CASCADE – The delete operation “cascades” to delete those matching tuples also. For example, if the delete mode is CASCADE, then deleting employee having EMPID as 101 from EMPLOYEE relation will also cause deletion of 2 more tuples from ASSIGNMENT relation.

Insert

The insertion of a tuple in the target of reference does not cause any violation. However, insertion of a tuple in the relation in which, we have the foreign key, for example, in ASSIGNMENT relation it needs to be ensured that all matching target candidate key exist; otherwise, the insert operation can be rejected. For example, one of the possible ASSIGNMENT insert operations would be (103, LG, 3000).

Modify

Modify or update operation changes the existing values. If these operations change the value that is the foreign key also, the only check required is the same as that of the Insert operation.

What should happen to an attempt to update a candidate key that is the target of a foreign key reference? For example, an attempt to update the PROJID “LG” for which there exists at least one matching ASSIGNMENT tuple? In general, there are the same possibilities as for DELETE operation:

RESTRICT: The update operation is “restricted” to the case where there are no matching ASSIGNMENT tuples. (It is rejected otherwise).

CASCADE – The update operation “cascades” to update the foreign key in those matching ASSIGNMENT tuples also.

5.2.3 Entity Integrity

Before describing the second type of integrity constraint, viz., Entity Integrity, you should be familiar with the concept of **NULL**.

Basically, NULL is intended as a basis for dealing with the problem of missing information. This kind of situation is frequently encountered in the real world. For example, historical records sometimes have entries such as “Date of birth unknown”. Hence it is necessary to have some way of dealing with such situations in database systems. Codd proposed an approach to this issue that makes use of special markers called NULL to represent such missing information.

A given attribute in the relation might or might not be allowed to contain NULL. But can the Primary key or any of its components (in case primary key is a composite key) contain a NULL? To answer this question an **Entity Integrity Rule** states: **No component of the primary key of a relation is allowed to accept NULL**. In other words, the definition of every attribute involved in the primary key of any basic relation must explicitly or implicitly include the specifications of **NULL NOT ALLOWED**.

Foreign Keys and NULL

Let us consider the relations:

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

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

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

So, the foreign key definition may be redefined to include NULL as an acceptable value in the foreign key for which there is no need to find a matching tuple.

+ Check Your Progress 1

Consider the following relations:

S		
SNO	SNAME	CITY
S1	Smita	Delhi
S2	Jim	Pune

P			
PNO	PNAME	COLOUR	CITY
P1	Nut	Red	Delhi
P2	Bolt	Blue	Pune

S3	Ballav	Pune
S4	Seema	Delhi
S5	Salim	Agra

P3	Part1	White	Mumbai
P4	Part2	Blue	Delhi
P5	Camera	Brown	Pune
P6	Part3	Grey	Delhi

J			SPJ			
<u>JNO</u>	<u>JNAME</u>	<u>CITY</u>	<u>SNO</u>	<u>PNO</u>	<u>JNO</u>	<u>QUANTITY</u>
J1	Sorter	Pune	S1	P1	J1	200
J2	Display	Bombay	S1	P1	J4	700
J3	OCR	Agra	S2	P3	J2	400
J4	Console	Agra	S2	P2	J7	200
J5	RAID	Delhi	S2	P3	J3	500
J6	EDP	Udaipur	S3	P3	J5	400
J7	Tape	Delhi	S4	P4	J3	900

- 1) For each of the relation, as given above, list the candidate keys. Also, identify the Primary key to each of the relations.

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

- 2) List the entity integrity constraints, which can be found in relations S, P, J, SPJ?
List the domain constraints, if any.

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

- 3) Which of the relation S, P, J, SPJ has referential constraints? List those constraints.

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

- 4) For the referential constraints as identified in question 3, suggest suitable referential actions.

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

5.3 REDUNDANCY AND ASSOCIATED PROBLEMS

Let us consider the following relation STUDENT.

Enrolment Number	Student Name	Student StAddress	Course Number	Course Name	Instructor	Office number of the Instructor
050112345	Rohan	D-27, Main Road, Ranchi	MCS-201	Problem Solution	Nayan Kumar	102
050112345	Rohan	D-27, Main Road, Ranchi	MCS-202	Computer Organisation	Anurag Sharma	105
050112345	Rohan	D-27, Main Road, Ranchi	MCS-203	OS	Preeti Anand	103
050111341	Aparna	B-III, Gurgaon	MCS-203	OS	Preeti Anand	103

Figure 5.3: A state of STUDENT relation

The above relation satisfies the properties of a relation and contains single value in each cell. Conceptually it is convenient to have all the information in one relation, as a single query to the database may produce complete information about a person. Does the student relation, as given in Figure 5.3 has any undesirable characteristics?

You may observe that Figure 5.3 contains duplicate information in several attributes. For example, the student, whose enrolment number is 050112345 has a name - Rohan and the student stays at an address “D-27, Main Road, Ranchi”. This information is repetitive in first three attributes of tuples 1, 2 and 3 (shown in Figure 5.3 in red colour). Similarly, the information that course name for number MCS-203 is OS and it is taught by “Preeti Anand”, whose office number is 103 (shown in Figure 5.3 in purple colour). You can observe that even this information is repetitive in tuple 3 and tuple 4. Thus, the relation of Figure 5.3 has the undesirable **Data Redundancy**.

In addition, when a new student takes admission and enrolls for the course MCS-203, then the entire information about the MCS-203 will be added to the relation. Such repetitive information not only increases the size of database, but also results in several problems, which are discussed below.

1. *Update Anomaly*: Consider the data redundancy of student name and address, as shown in Figure 5.3. Assume that the student Rohan shifts from Ranchi to New Delhi at a new address “F-102, Maidan Garhi, New Delhi”. This will require to update the address of Rohan in all the three tuples of the Student relation. All the three tuples must be updated consistently. If this does not happen, then the address of Rohan will be inconsistent. This inconsistency is the result of update operation on redundant data. Thus, this anomaly is termed as update anomaly, which causes **data inconsistency**.

2. *Insertion Anomaly*: The note that the primary key of the Student relation, as given in Figure 5.3, is composite key consisting of enrolment number and CoNo. Any new tuple to be inserted in the relation must have a value for both the attributes of the primary key, as entity integrity constraint requires that a key may not be totally or partially NULL. However, in the given relation if you want to insert the number and name of a new course in the database, it would not be possible until a student enrolls in that course. Similarly, information about a new student cannot be inserted in the database until the student enrolls in a course. These problems are called insertion anomalies.

3. *Deletion Anomalies*: Loss of Useful Information: In some instances, useful information may be lost when a tuple is deleted. For example, if you delete the tuple corresponding to student 050111341 enrolled for MCS-203, you will lose relevant information about the student viz. enrolment number, name and address of this

student. Similarly, deletion of tuple having StName “Rohan” and CoNo ‘MCS-202’ will result in loss of information that MCS-202 is named “Computer organization” having an instructor “Anurag Sharma”, whose office number is 105. This is called deletion anomaly.

The anomalies arise primarily because the relation STUDENT has information about students as well as courses. One solution to the problems is to decompose the relation into two or more smaller relations, so that a relation contains information about one thing. But what should be the basis of this decomposition? To answer the questions let us try to articulate dependence of data within a relation with the help of the following *Figure 5.4*:

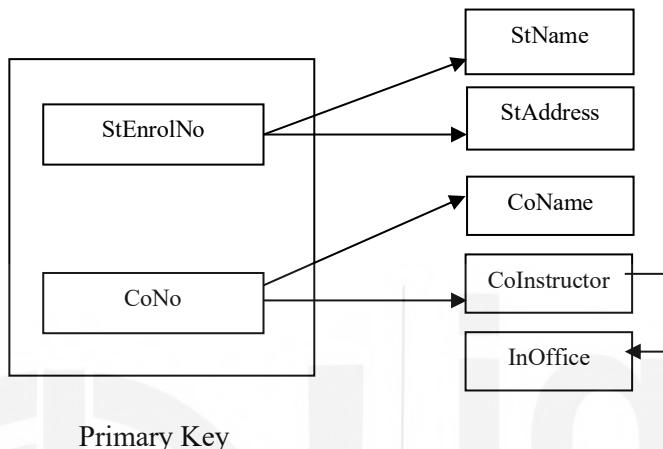


Figure 5.4: The dependencies of relation in Figure 5.3

In the *Figure 5.4*, an arrow shows dependence of a data attributes. For example, you may notice that two arrows are originating from an attribute student enrolment number (*StEnrolNo*). One of these arrows is to student name attribute (*StName*) and other link to student address (*StAddress*). The link between attribute *StEnrolNo* and student name attribute (*StName*) denotes that enrolment number attribute uniquely determines the student’s name. For example, in Figure 5.3 the enrolment number 050112345 uniquely determines that name of the student is Rohan. Likewise, you may determine the dependence among different attributes. You may notice that the dependence of data can exist even among the attributes, which are not the part of the primary key, such as, a dependence from course instructor (*CoInstructor*) attribute to instructor office number (*InOffice*) attribute. The dependence among the attributes forms the basis of decomposition of a relation into two or more relations, this is called the normalisation. The objective of this decomposition is to reduce the three anomalies in the decomposed relations. However, normalisation should not result in loss of information, which means that you should be able to obtain the original relation’s information by taking JOIN of the relations obtained after decomposition.

A relation that needs to be normalised may have a very large number of attributes. In such relations, it is almost impossible for a person to conceptualise all the information and suggest a suitable decomposition to overcome the problems. Such relations need an algorithmic approach of finding if there are problems in a proposed database design and how to eliminate them if they exist. The discussions of these algorithms are beyond the scope of this Unit, but we will first introduce you to the basic concept that supports the process of Normalisation of large databases. So let us first define the concept of functional dependence in the subsequent section and follow it up with the concepts of normalisation.

5.4 FUNCTIONAL DEPENDENCIES

A database is a collection of related information and it is therefore inevitable that some items of information in the database would depend on some other items of information. The information is either single-valued or multi-valued. The enrolment number of a student and his/her date of birth are single-valued information; qualifications of a person or subjects that an instructor teaches are multi-valued facts. In this section, we will deal with single-valued facts, which forms the basis of the concept of functional dependency. Let us define this concept logically.

Functional Dependency (FD)

Let us consider a single universal relation schema “A”. A functional dependency denoted by $X \rightarrow Y$, between two sets of attributes X and Y that are subset of universal relation “A” specifies a constraint on the possible tuples that can form a relational state of “A”. Consider any two tuples of a relation A, say t_1 and t_2 , a FD is said to exist between two set of attributes X to Y, if the following holds in A:

If $t_1(X) = t_2(X)$, then $t_1(Y) = t_2(Y)$ must be true.

It means that, if tuple t_1 and tuple t_2 have same values for attributes X, then to hold $X \rightarrow Y$ on “A”, t_1 and t_2 must have same values for attributes Y also.

Thus, FD $X \rightarrow Y$ means that the values of the Y component of a tuple in “A” depend on or is determined by the values of X component. In other words, the value of Y component is uniquely determined by the value of X component. This is functional dependency from X to Y (**but not Y to X**) that is, Y is functionally dependent on X.

The relation schema “A” determines the function dependency of Y on X ($X \rightarrow Y$) when and only when:

- 1) if tuples in “A”, which agree on their X value, then
- 2) they **must** agree on their Y values too.

Please note that if $X \rightarrow Y$ in “A”, does not mean $Y \rightarrow X$ in “A”.

Please note that functional dependency (FD) does not imply a one-to-one relationship between X and Y.

For example, the functional dependencies of Figure 5.4 are:

$$\begin{array}{lcl} \text{StEnrolNo} & \rightarrow & \text{StName, StAddress} \\ \text{CoNo} & \rightarrow & \text{CoName, CoInstructor} \\ \text{CoInstructor} & \rightarrow & \text{InOffice} \end{array}$$

These functional dependencies imply that there can be only one student name for each **StEnrolNo**, only one address for each student and only one course name for each **CoNo**. Please note, given this set of FDs, it is possible that two or more students, who have different enrolment numbers may have the same name. In addition, two or more students can reside at the same address. However, two different students cannot have same enrolment number.

Similarly, consider **CoNo → CoInstructor**, the dependency implies that no subject can have more than one instructor (perhaps this is not a very realistic assumption). Functional dependencies therefore place constraints on what information the database may store. In addition, in the example above, you may be wondering if the following FDs hold:

$$\text{StName} \rightarrow \text{StEnrolNo} \quad (1)$$

$$\text{CoName} \rightarrow \text{CoNo} \quad (2)$$

Certainly, there is nothing in the given instance of the database relation presented that contradicts the functional dependencies as above. However, whether these FDs hold or not would depend on whether the university or college, whose database you are considering, allows two different students to have the same name and two different courses to have the same course names. If it was the enterprise policy to have unique course names, then (2) holds. If two students have exactly the same name, then (1) does not hold.

Let us use an E-R diagram to show various FDs (Please refer to *Figure 5.5*). A simple example of the functional dependency in this ERD is when X is a primary key of an entity (e.g., enrolment number) and Y is some single-valued property or attribute of the entity (e.g., student name). $X \rightarrow Y$ then must always hold. (Why?)

Functional dependencies also arise in relationships. Let C and D be the primary keys of two strong entity participating in a relationship. If the relationship is one-to-one, both the FDs $C \rightarrow D$ and $D \rightarrow C$ will hold. If the relationship is many-to-one (C on many side), an FD $C \rightarrow D$ will hold but the FD $D \rightarrow C$ will NOT hold. In case, a relationship cardinality is many-to-many, then the key attributes of the participating entities would not have any functional dependency between them.

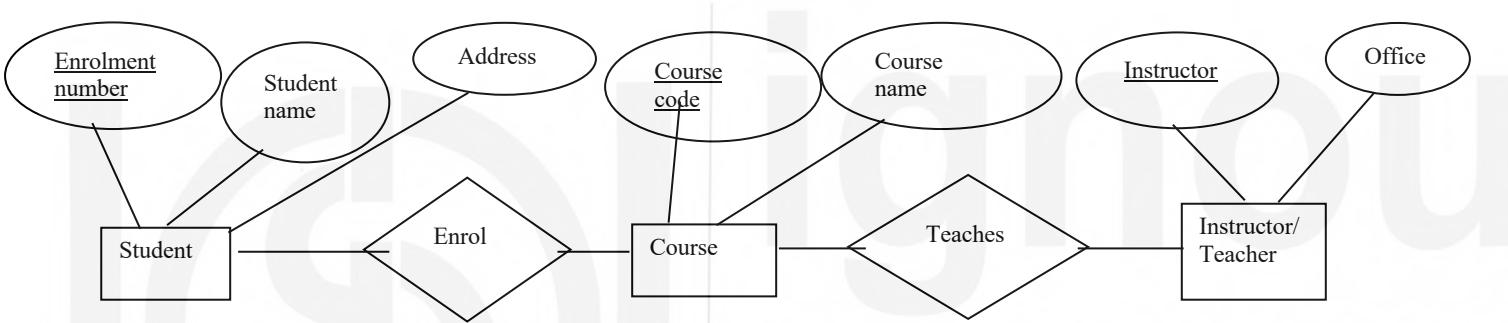


Figure 5.5: ERD of Entities – Student, Course and Teacher

ER diagram Figure 5.5 can be converted to the following relations:

Relations of Entities

STUDENT (enrno, StName, StAddress)

FDs: $\text{enrno} \rightarrow \text{StName}, \text{StAddress}$

COURSE (CoCode, CoName)

FD: $\text{CoCode} \rightarrow \text{CoName}$

INSTRUCTOR (inId, InOffice) // inID is instructor Id

FD: $\text{inId} \rightarrow \text{InOffice}$

Relations of Relationships

ENROL (enrno, CoCode) (assuming many-to-many cardinality)

FD: $\text{enrno}, \text{CoCode} \rightarrow \text{NULL}$

Assuming that one course can be taught by only one teacher, but one teacher can teach many courses, you need to redesign COURSE relation as:

COURSE (CoCode, CoName, inId)

FDs: $\text{CoCode} \rightarrow \text{CoName}, \text{inId}$

Identification of FDs:

Identification of FDs, in general, is not trivial. You need to study the domain of attributes and relationships among these attributes. You cannot identify FDs by using a set of rules. The following example explains this.

Consider the following relation:

STUDENT-COURSE (enrolno, sname, cname, classlocation, hours)

The following functional dependencies may exist on this relation (you should identify the FDs primarily from constraints, there is no thumb rule to do so otherwise):

- **enrolno → sname** (the enrolment number of a student uniquely determines the student names alternatively; you can say that sname is functionally determined/dependent on enrolment number).
- **classcode → cname, classlocation**, (the value of a class code uniquely determines the class name and class location).
- **enrolno, classcode → Hours** (a combination of enrolment number and class code values uniquely determines the number of hours and students' study in the class per week (Hours)).

The semantic property of functional dependency explains how the attributes in “A” are related to one another. A FD in “A” must be used to specify constraints on its attributes that must hold at all times.

For example, a FD (State, City, Place) → Pin_code should hold for any address in India. It is also possible that certain functional dependencies may cease to exist in the real world if the relationship changes, for example, the FD Pin_code → Area_code used to exist as a relationship between postal codes and telephone number area codes in India, however, with the proliferation of mobile telephone, the FD is no longer true.

The set of FDs over a relation can be optimised to obtain a minimal set of FDs called the canonical cover. However, these topics are beyond the scope of this course and can be studied by consulting further readings.

5.5 NORMALISATION USING FUNCTIONAL DEPENDENCIES

Codd in the year 1972 presented three normal forms (1NF, 2NF, and 3NF). These were based on functional dependencies among the attributes of a relation. Later Boyce and Codd proposed another normal form called the Boyce-Codd normal form (BCNF). The fourth and fifth normal forms are based on multivalue and join dependencies and were proposed later. In this section we will cover normal forms till BCNF only. Fourth and fifth normal forms are discussed in the next unit.

For all practical purposes, 3NF or the BCNF are quite adequate since they remove the anomalies discussed for most common situations. It should be clearly understood that there is no obligation to normalise relations to the highest possible level. Performance should be taken into account and sometimes an organisation may take a decision not to normalise, say, beyond third normal form. But it should be noted that such designs should be careful enough to take care of anomalies that would result because of the decision above.

Intuitively, the second and third normal forms are designed to result in relations such that each relation contains information about only one thing (either an entity or a relationship). A sound E-R model of the database would ensure that all relations either provide facts about an entity or about a relationship resulting in the relations that are obtained being in 2NF or 3NF.

The normalisation of a relation till BCNF is established using the FDs. The normalisation process depends on the assumptions that:

- 1) a set of functional dependencies is given for each relation, and
- 2) each relation has a designated primary key.

The normalisation process proceeds in a top-down fashion by evaluating each relation against the criteria for normal forms and decomposing relations as found necessary

during analysis. Therefore, normalisation upto BCNF is looked upon as a process of analysing the given relation schemas based on their condition (FDs and Primary Keys) to achieve the desirable properties:

- firstly, minimizing redundancy, and
- secondly minimizing the insertion, deletion and update anomalies.

Thus, the normalisation provides the database designer with:

- 1) a formal framework for analysing relation schemas.
- 2) a series of normal form tests that can be normalised to any desired degree.

Decomposition through normalization should include any or both of the following properties.

- 1) the lossless join and non-additive join property, and
- 2) the dependency preservation property.

Do you always normalize database to highest normalized form? Due to performance reasons, database designers sometimes leave relations in lower normalisation forms. This is called denormalisation.

Let us now define the normal forms in more detail.

5.5.1 The First Normal Form (1NF)

Let us first define 1NF:

Definition: A relation (table) R is in 1NF if every attribute of R takes atomic values. In other words the following conditions hold in R:

1. There are no duplicate rows or tuples in the relation.
2. Each data value stored in the relation is single-valued
3. Entries in a column (attribute) are of the same kind (type).

Please note that in a 1NF relation the order of the tuples (rows) and attributes (columns) does not matter.

The first requirement above means that the relation **must have a key**. The key may be single attribute or composite key. It may even, possibly, contain all the columns.

The first normal form defines only the basic structure of the relation and does not resolve the anomalies discussed in Section 5.3.

The relation STUDENT (StEnrolNo, StName, StAddress, CoNo, CoName, CoInstructor, InOffice) of *Figure 5.3* is in 1NF. The primary key of the relation is a composite key of attributes StEnrolNo and CoNo.

5.5.2 The Second Normal Form (2NF)

The relation of *Figure 5.3* is in 1NF, yet it suffers from all the anomalies. Therefore, a second normal form may be defined to overcome the anomalies.

Definition: A relation is in Second Normal Form (2NF) if it fulfils the following criteria (assuming the relation has only one candidate key, which is selected as the primary key of the relation):

- (i) The relation fulfils the criteria of the First Normal Form (1NF), and
- (ii) All those attributes, which are not part of any primary key, are fully functionally dependent on the primary keys.

Key features of 2NF:

- The partial dependency will exist, only if the primary key is composite. Therefore, if the primary key of a relation consists of a single attribute, then the given 1NF relation would be in 2NF also.
- This normal form decomposes a relation so that relations store information about one thing.

Please refer to *Figure 5.4*, which illustrates the FDs in the relation STUDENT (StEnrolNo, StName, StAddress, CoNo, CoName, CoInstructor, InOffice). The FDs, as shown in *Figure 5.4* are:

$$\begin{array}{lcl} \text{StEnrolNo} & \rightarrow & \text{StName, StAddress} \\ \text{CoNo} & \rightarrow & \text{CoName, CoInstructor} \\ \text{CoInstructor} & \rightarrow & \text{InOffice} \end{array} \quad \begin{array}{l} (1) \\ (2) \\ (3) \end{array}$$

Since, from the above FDs, you can create a FD:

$$\text{StEnrolNo, CoNo} \rightarrow \text{StName, StAddress, CoName, CoInstructor, InOffice}$$

Therefore, the attributes StEnrolNo and CoNo together forms the composite key to the relation STUDENT. The relation has only one candidate key, therefore, all the attributes, other than StEnrolNo and CoNo, of the relation are not part of a candidate key. These attributes are also called the non-key attributes. The relation STUDENT is in 1NF, but is it in 2NF? No, the FDs at (1) and (2) are clearly violating the fully functionally dependence criteria of 2NF (*Figure 5.4*). Therefore, the relation suffers from the anomalies as shown in Figure 5.3. Next, how will you decompose the STUDENT relation to 2NF relations? You may use FDs of (1), (2) and (3) to do so. FD (1) can be used to create a 2NF relation consisting of these three attributes from STUDENT relation. This part new relation is:

$$\text{STUDENT1 } (\underline{\text{StEnrolNo}}, \text{StName}, \text{StAddress})$$

Similarly, you can use FD (2) to decompose the STUDENT relation further, but what about the attribute ‘InOffice’? You find in FD (2) that Course code (CoNo) attribute uniquely determines the name of instructor (refer to FD 2(a)). Also, the FD (3) means that name of the instructor uniquely determines office number. This can be written as:

$$\begin{array}{lcl} \text{CoNo} & \rightarrow & \text{CoInstructor} \\ \text{CoInstructor} & \rightarrow & \text{InOffice} \end{array} \quad \begin{array}{l} (2 \text{ (a)) (without CoName)} \\ (3) \end{array}$$

The above two FDs imply a transitive dependency:

$$\text{CoNo} \rightarrow \text{InOffice} \quad (\text{This is transitive dependency})$$

The revised FD (2) is:

$$\text{CoNo} \rightarrow \text{CoName, CoInstructor, InOffice} \quad (4)$$

Use this FD to create another 2NF relation:

$$\text{COU_INST } (\underline{\text{CoNo}}, \text{CoName}, \text{CoInstructor}, \text{InOffice})$$

Now, you have the following two 2NF relations, which are obtained by decomposing the STUDENT relation:

$$\begin{array}{l} \text{STUDENT1 } (\underline{\text{StEnrolNo}}, \text{StName}, \text{StAddress}) \\ \text{COU_INST } (\underline{\text{CoNo}}, \text{CoName}, \text{CoInstructor}, \text{InOffice}) \end{array}$$

Are these two 2NF relations sufficient to represent the relations STUDENT? No, as there is no common attribute between the relations. Therefore, they cannot be joined together to get the data of STUDENT relation. You must have a relation that joins the two decomposed relations. This relation would have the primary key attributes of the STUDENT relation and any other attribute that is fully functionally dependent on this primary key. However, there is no attribute in STUDENT relation that is fully dependent on the composite primary key.

Thus, you will create a third relation using the composite primary key of the STUDENT relation, as shown below:

COURSE_STUDENT (StEnrolNo, CoNo)

Please note that the COURSE_STUDENT relation can be joined with STUDENT1 relation on StEnrolNo attribute and with COU_INST relation using CoNo attribute.

So, the relation STUDENT in 2NF form would be:

STUDENT1 (<u>StEnrolNo</u> , StName, StAddress)	2NF(a)
COU_INST (<u>CoNo</u> , CoName, CoInstructor, InOffice)	2NF(b)
COURSE_STUDENT (<u>StEnrolNo</u> , <u>CoNo</u>)	2NF(c)

5.5.3 The Third Normal Form (3NF)

Although, transforming a relation that is not in 2NF into a number of relations that are in 2NF removes many of the anomalies, it does not necessarily remove all anomalies. Thus, further Normalisation is sometimes needed to ensure further removal of anomalies. These anomalies arise because a 2NF relation may have attributes that are not directly related to the primary key of the relation.

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

But what is **non-transitive** dependence?

Let A, B and C be three attributes of a relation R such that $A \rightarrow B$ and $B \rightarrow C$. From these FDs, we may derive $A \rightarrow C$. This dependence $A \rightarrow C$ is transitive.

Now, let us reconsider the relation 2NF (b)

COU_INST (CoNo, CoName, Instruction, InOffice)

Assume that CoName is not unique and therefore CoNo is the only candidate key. The following functional dependencies exists

CoNo	\rightarrow	CoInstructor	(2 (a))
CoInstructor	\rightarrow	InOffice	(3)
CoNo	\rightarrow	InOffice	(This is transitive dependency)

You had derived $CoNo \rightarrow InOffice$ from the functional dependencies 2(a) and (3) for decomposition to 2NF. The relation is, however, not in 3NF since the attribute 'InOffice' is not directly dependent on attribute 'CoNo' but is transitively dependent on it and should, therefore, be decomposed as it has all the anomalies. The primary difficulty in the relation above is that an instructor might be responsible for several subjects, requiring one tuple for each course. Therefore, his/her office number will be repeated in each tuple. This leads to all the problems such as update, insertion, and deletion anomalies. To overcome these difficulties, you need to decompose the relation 2NF(b) into the following two relations:

COURSE (CoNo, CoName, CoInstructor) INST (CoInstructor, InOffice)

Please note these two relations and 2NF (a) and 2NF (c) are already in 3NF. Thus, the relation STUDENT in 3 NF would be:

STUDENT1 (<u>StEnrolNo</u> , StName, StAddress)
COURSE (<u>CoNo</u> , CoName, CoInstructor)
INST (<u>CoInstructor</u> , InOffice)
COURSE_STUDENT (<u>StEnrolNo</u> , <u>CoNo</u>)

The 3NF is usually quite adequate for most relational database designs. There are, however, some situations where a relation may be in 3 NF but have the anomalies. For example, consider a relation STUDENT5 (StEnrolNo, StName, CoNo, CoName). Assume it has the following set of FDs:

$$\begin{array}{lcl} \text{StEnrolNo} & \rightarrow & \text{StName} \\ \text{StName} & \rightarrow & \text{StEnrolNo} \\ \text{CoNo} & \rightarrow & \text{CoName} \\ \text{CoName} & \rightarrow & \text{CoNo} \end{array}$$

Is the relation STUDENT5 is in 3NF? The FDs of this relation can be written as:

$$\begin{array}{lcl} \text{StEnrolNo, CoNO} & \rightarrow & \text{StName, CoName} \\ \text{StEnrolNo, CoName} & \rightarrow & \text{StName, CoNO} \\ \text{StName, CoNO} & \rightarrow & \text{StEnrolNo, CoName} \\ \text{StName, CoName} & \rightarrow & \text{StEnrolNo, CoNO} \end{array}$$

Therefore, the relation STUDENT5 has the following candidate keys.

- (StEnrolNo, CoNo)
- (StEnrolNo, CoName)
- (StName, CoNo)
- (StName, CoName)

Figure 5.6 shows the functional dependency diagram for this relation.

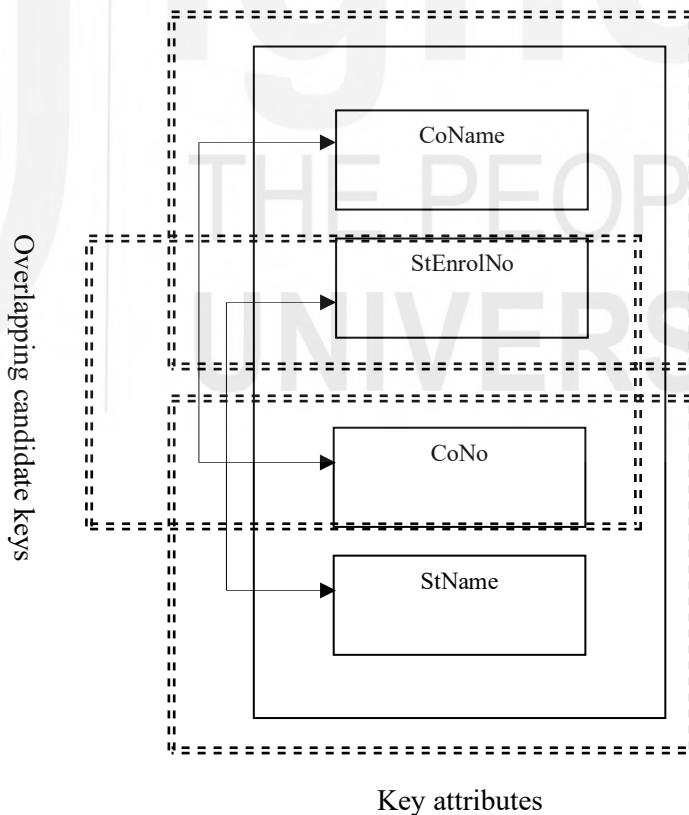


Figure 5.6: Functional Diagram for STUDENT5 relation

You may observe that all the attributes of STUDENT5 relations are prime attributes as they are part of some candidate key. You may also observe that the relation has overlapping candidate keys. Therefore, no non-key attribute exists in the relation. Hence, the relation follows the criteria of 2NF and 3NF and is in 3NF. However, this relation still suffers from the three anomalies (please make an instance for the

relation), as the attributes in the non-overlapping part of the candidate key can determine each other. Therefore, the relation is to be normalised into a more stronger normal form called BCNF.

5.5.4 Boyce-Codd Normal Form (BCNF)

As stated earlier, the relation STUDENT5 (StEnrolNo, StName, CoNo, CoName) has the following candidate keys:

$$\begin{array}{ll} (\text{StEnrolNo}, \text{CoNo}) ; & (\text{StEnrolNo}, \text{CoName}) \\ (\text{StName}, \text{CoNo}) ; & (\text{StName}, \text{CoName}) \end{array}$$

Since the relation has no non-key attributes, the relation is in 2NF and also in 3NF. However, the relation suffers from the anomalies (please check it yourself by making the relational instance of the STUDENT5 relation).

The difficulty in this relation is being caused by dependence within the attributes of composite candidate keys.

Definition: A relation is in Boyce-Codd Normal Form (BCNF) if it fulfils the following criteria:

- (i) The relation fulfils the criteria of the Third Normal Form (3NF), and
- (ii) All the determinants (the left side of an FD) are the candidate key.

A relation that is in 3NF and does not have overlapping candidate keys, will also be in BCNF. However, if a 3NF relation have following features then it is NOT in BCNF:

- (a) It has overlapping composite candidate keys.
- (b) The non-overlapping attributes of two candidate key are functionally dependent.

For example, in the relation STUDENT5, the candidate keys:

(StEnrolNo, CoName) and (StName, CoName) has non-overlapping attributes StEnrolNo and StName. Further, $\text{StEnrolNo} \rightarrow \text{StName}$

The relation STUDENT5 (StEnrolNo, StName, CoNo, CoName) is in 3NF, but not in BCNF.

You can decompose the relation into the following relations using FD

$\text{StEnrolNo} \rightarrow \text{StName}$ as:

$$\begin{array}{l} \text{STUDENT6}(\text{StEnrolNo}, \text{CoNo}, \text{CoName}) \\ \text{STUDENTNAME}(\text{StEnrolNo}, \text{StName}) \end{array}$$

STUDENT6 is still not in BCNF as it has overlapping candidate keys:

(StEnrolNo, CoNo) and (StEnrolNo, CoName) and the FD $\text{CoNo} \rightarrow \text{CoName}$

Thus, STUDENT6 can be decomposed using the FD as above to relations:

$$\begin{array}{l} \text{STUDENT7}(\text{StEnrolNo}, \text{CoNo}) \\ \text{STUDENTNAME}(\text{StEnrolNo}, \text{StName}) \\ \text{COURSENAME}(\text{CoNo}, \text{CoName}) \end{array}$$

The following is another example of BCNF. Consider the relation:

ENROL (StEnrolNo, StName, CoNo, CoName, Dateenrolled)

Let us assume that the relation has the following FDs (We have assumed that CoName are unique):

$$\begin{array}{ll} \text{StEnrolNo} & \rightarrow \text{StName} \\ \text{CoNo} & \rightarrow \text{CoName} \\ \text{CoName} & \rightarrow \text{CoNo} \\ \text{StEnrolNo, CoNo} & \rightarrow \text{Dateenrolled} \end{array}$$

The candidate keys of the relation would be:

$$\begin{array}{l} (\text{StEnrolNo}, \text{CoNo}) \\ (\text{StEnrolNo}, \text{CoName}) \end{array}$$

Due to dependency between the non-overlapping attributes of the candidate keys, the 3NF relation ENROL is NOT in BCNF. This relation will have all the stated anomalies (Please draw the relational instance and check these problems). The BCNF decomposition of the relation would be:

STUD1 (StEnrolNo, StName)
COU1 (CoNo, CoName)
ENROL1 (StEnrolNo, CoNo, Dateenrolled)

You now have a relation that only has information about students, another only about subjects and the third only about relationship enrolls.

Higher Normal Forms:

Are there more normal forms beyond BCNF? Yes, however, these normal forms are not based on the concept of functional dependence. Further normalisation is needed if the relation has multi-valued, join dependencies, etc. These are discussed in Unit 6.

+ Check Your Progress 2

- 1) Given the following relation of book issue and return in a Library:
BOOKISSUERETURN (student_id, student_name, bookID, book_title, issuedOn, returnedOn)
A student can get many books issued to him/her. Explain anomalies in the relation above with the help of a relational instance.

.....
.....

- 2) Identify the functional dependencies in the relation given in question 1. What are the candidate keys and which of these can be a primary key?

.....
.....

- 3) Normalise the relation of problem 1.

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

5.6 DESIRABLE PROPERTIES OF DECOMPOSITION

In the previous section, you have learnt the process of normalization using functional dependency. Normalizing a relation entails decomposing a relation into a number of non-disjoint projections of the relation based on the FDs, so that the three anomalies are minimised. But why these projections are non-disjoint? The non-disjoint relations allow reconstruction of original relation instance through JOIN operation. In this section, we discuss about the properties of a good decomposition.

The decomposition of a relation should fulfil the following:

Attribute Preservation: All the attributes of the relation, which is being decomposed, should be part of at least one of the decomposed relations. This is a necessary condition, otherwise the decomposition will be lossy.

Lossless-Join Decomposition: For explaining the concept of lossless-join decomposition, let us first explain a lossy decomposition with the help of an example. This example also explains, why FDs should be used to perform proper decomposition of a relation

Example: Consider the following instance of a STUDENT9 relation.

STUDENT9 (StEnrolNo, CoNo, Dateenrolled, CoInstructor, InRoom)

With the following set of FDs:

$$\begin{aligned} \underline{\text{StEnrolNo}}, \text{CoNo} &\rightarrow \text{Dateenrolled} \\ \text{CoNo} &\rightarrow \text{CoInstructor} \\ \text{CoInstructor} &\rightarrow \text{InRoom} \end{aligned}$$

Suppose you decompose the STUDENT9 relation randomly into two relations ST1 and ST2 as follows:

ST1 (StEnrolNo, CoNo, Dateenrolled, CoInstructor)

ST2 (StEnrolNo, InRoom)

Are there problems with this decomposition? Yes, but for the time being, let us focus on the question, whether this decomposition is lossless? Consider the following instance of the relation

STUDENT9

StEnrolNo	CoNo	Dateenrolled	CoInstructor	InRoom
1001	MCS-201	01-02-2022	Preeti	1
1001	MCS-202	01-02-2022	Salim	2
1002	MCS-201	13-02-2022	Preeti	1
1002	MCS-203	13-02-2022	Shashi	3
1003	MCS-202	15-02-2022	Salim	2

Figure 5.7: An instance of STUDENT9 relation

The decomposed relations ST1 and ST2 would be:

ST1

StEnrolNo	CoNo	Dateenrolled	CoInstructor
1001	MCS-201	01-02-2022	Preeti
1001	MCS-202	01-02-2022	Salim
1002	MCS-201	13-02-2022	Preeti
1002	MCS-203	13-02-2022	Shashi
1003	MCS-202	15-02-2022	Salim

ST2

StEnrolNo	InRoom
1001	1
1001	2
1002	1
1002	3
1003	2

Will you be able to reconstruct the original instance of relation using ST1 and ST2? For this simply take a JOIN of the two relations ST1 and ST2 on StEnrolNo, which is the only common attribute. The joined instance would be:

ST1JOINST2

StEnrolNo	CoNo	Dateenrolled	CoInstructor	InRoom
1001	MCS-201	01-02-2022	Preeti	1
1001	MCS-201	01-02-2022	Preeti	2
1001	MCS-202	01-02-2022	Salim	1
1001	MCS-202	01-02-2022	Salim	2
1002	MCS-201	13-02-2022	Preeti	1
1002	MCS-201	13-02-2022	Preeti	3
1002	MCS-203	13-02-2022	Shashi	1

1002	MCS-203	13-02-2022	Shashi	3
1003	MCS-202	15-02-2022	Salim	2

Thus, the resulting relation obtained is not the same as that of *Figure 5.7*. The joined relation contains a number of spurious tuples that were not in the original relation. Because of these additional tuples, you have lost the information, such as the instructor Preeti is located in Room number 1. Such decompositions are called **lossy decompositions**. A lossless join decomposition guarantees that the join will result in exactly the same relation as was decomposed. One might think that there might be other ways of recovering the original relation from the decomposed relations but, sadly, no other operators can recover the original relation if the join does not (why?). You need to analyse why the decomposition is lossy. The common attribute in the above decompositions was StEnrolNo. The common attribute is the glue that gives us the ability to find the relationships between different relations by joining the relations together. *If the common attribute has been the primary key of at least one of the two decomposed relations, the problem of losing information would not have existed.* You may use FDs to decompose the relation STUDENT9 into 2NF and then to 3NF, to produce the following relational instance:

STENROL (StEnrolNo, CoNo, Dateenrolled)
using the FD StEnrolNo, CoNo → Dateenrolled

STENROL

StEnrolNo	CoNo	Dateenrolled
1001	MCS-201	01-02-2022
1001	MCS-202	01-02-2022
1002	MCS-201	13-02-2022
1002	MCS-203	13-02-2022
1003	MCS-202	15-02-2022

COUINST (CoNo, CoInstructor) using the FD CoNo → CoInstructor

COUINST

CoNo	CoInstructor
MCS-201	Preeti
MCS-202	Salim
MCS-203	Shashi

INSROOM (CoInstructor, InRoom) using the FD CoInstructor → InRoom

INSROOM

CoInstructor	InRoom
Preeti	1
Salim	2

You may please join the three relations to check that that the decomposition is lossless-join decomposition. *The dependency-based decomposition scheme as discussed in the section 5.5 creates lossless decomposition.*

Dependency Preservation: It is clear that the decomposition must be lossless so that you do not lose any information from the relation that is decomposed. Dependency preservation is another important requirement since a *dependency is a constraint* on the database. If all the attributes appearing on the left and the right side of a dependency appear in the same relation, then a dependency is considered to be preserved. Thus, dependency preservation can be checked easily. Dependency preservation is important, because as stated earlier, dependency is a constraint on a relation. Thus, if a constraint is split over more than one relation (dependency is not preserved), the constraint would be difficult to meet. We will not discuss this in more detail in this unit. You may refer to the further readings for more details. However, let us state one basic point:

"Normalisation to 3NF is lossless decomposition. It also preserves the dependencies."

"Normalisation to BCNF is lossless decomposition. However, it may not preserve dependencies."

Reduction of Redundancy: Redundancy is the cause of anomalies. Normalisation results in reduction of redundancy.

5.6 RULES OF DATA NORMALISATION

In the previous sections, you have gone through various normal forms. In this section, the normalisation using FDs is presented as a set of practical rules:

1. Identify and eliminate Attributes with multiple data values into separate relation.
2. Identify and eliminate those attributes, which are not the part of any key attributes but are dependent on the part of the composite primary key.
3. Identify and eliminate those non-key attributes, which are dependent on other non-key attributes.
4. Identify and eliminate non-overlapping composite candidate key attributes, which are dependent on each other.

Let's explain these steps of Normalisation through an example. Let us make a list of all the employees in the company. In the original employee list, each employee name is followed by any databases that the employee has experience with. Some might know many, and others might not know any.

Employee ID (empid)	Employee Name (empname)	DBMS Known (DBMS)	Department (dept)	Department Location (loc)
101	Gurmeet	MySQL	Quality Assurance	Mumbai
102	Hanif	DB2	Database Design	Delhi
103	Manish	Oracle, PostgreSQL	Frontend Design	Hyderabad
104	Sameer Singh	SQLserver, Oracle	Database Design	Delhi

Figure 5.8: Table of Data Not in 1NF

For attributes Department and Department Location will be considered in Step-3.

5.6.1 Eliminate Repeating Groups

Please observe that Figure 5.8 has a repeating group – DBMS Known. The problem with such repeating group can be explained with the help of a query. Consider a query - “Find the list of employees, who know Oracle”.

To answer this query, you need to perform an awkward scan of the entire list of attributes Database-Known, looking for references to DB2. This is inefficient and an extremely untidy way to retrieve information.

You may convert the relation to 1NF (For the following discussion, we have ignored two attributes - Department and Department Location. These two attributes will be part of Employee relation). The two decomposed relation on eliminating the repeating groups are shown in Figure 5.9. The elimination of repeating group DBMS Known from the Employee relation, which has primary key - Employee ID, leaves the Employee relation in 1NF. However, this elimination requires that you add another relation, which can store information about the DBMS known by each employee. This new relation is named DBMSknown. It has three attributes, including a foreign key for relating the two relations with a JOIN operation. Now, you can answer the question by looking in the database relation for "Oracle" and getting the list of Employees. Please note that although the name of the DBMSs are unique, yet we have added a DBMS code field in the decomposed relation (Figure 5.8):

EMPLOYEE			
empid	empname	dept	loc
101	Gurmeet	Quality Assurance	Mumbai
102	Hanif	Database Design	Delhi
103	Manish	Frontend Design	Hyderabad
104	Sameer Singh	Database Design	Delhi

DBMSknown		
empid	DBMScode	DBMS
101	1	MySQL
102	2	DB2
103	3	Oracle
103	4	PostgreSQL
104	5	SQLserver
104	3	Oracle

Figure 5.9: 1NF relation after Eliminating Repeating Groups

5.6.2 Eliminate Redundant Data

In the “DBMSknown” relation above, the primary key is made up of the *empid* and the *DBMScode*. The attribute *DBMS* depends only on the *DBMScode*. The same *DBMS* will appear redundantly every time its associated *DBMScode* appears in the *DBMSknown* Relation. Thus, this database relation has redundancy, for example, *DBMScode* value 3 is Oracle, which is repeated twice. In addition, it also suffers insertion anomaly that is you cannot enter Sybase in the relation as no employee has that database skill.

The deletion anomaly also exists. For example, if you remove employee with *empid* 3; no employee has a skill in PostgreSQL and the information that *DBMScode* 4 is the code of PostgreSQL will be lost.

To avoid these problems, you need second normal form. To achieve this, you isolate the attributes that depends on the entire composite key from the attributes those depends only on the *DBMScode*. This results in decomposition of *DBMSknown* relation into two relations: “DBMSlist” and “EMPDBMS” which lists the databases for each employee. The EMPLOYEE relation is already in 2NF as all the *empid* determines all other attributes.

EMPDBMS		DBMSlist	
empid	DBMScode	DBMScode	DBMS
101	1	1	MySQL
102	2	2	DB2
103	3	3	Oracle
103	4	4	PostgreSQL
104	5	5	SQLserver
104	3	3	Oracle

5.6.3 Eliminate Columns Not Dependent on Key

The Employee Relation satisfies -

First normal form - As it contains no repeating groups.

Second normal form - As it does not have a multi-attribute key.

Now, let us add the remaining two attributes of Employee relation. The employee relation is in 2NF but not 3NF. Why?

EMPLOYEE			
empid	empname	dept	loc
101	Gurmeet	Quality Assurance	Mumbai
102	Hanif	Database Design	Delhi
103	Manish	Frontend Design	Hyderabad
104	Sameer Singh	Database Design	Delhi

The key to the Employee relation is *empid*. The attribute *loc* describes information about the Department and not about an Employee. To achieve the third normal form, *dept* and *loc* must be moved into a separate relation. Since they describe a department, thus, the attribute *dept* becomes the key of the new “Department” relation.

The motivation for this decomposition is that you want to avoid update, insertion and deletion anomalies.

EMPLOYEElist		
empid	empname	dept
101	Gurmeet	Quality Assurance
102	Hanif	Database Design
103	Manish	Frontend Design
104	Sameer Singh	Database Design

DEPARTMENT Relation		
deptid	dept	loc
1	Quality Assurance	Mumbai
2	Database Design	Delhi
3	Frontend Design	Hyderabad

You may use these steps for decomposition till BCNF.

+ Check Your Progress 3

- 1) Why functional dependencies should be preserved in decomposition?

.....

- 2) Explain the term lossless-join decomposition.

.....

- 3) List various steps that may be followed to decompose a relation up to BCNF.

.....

5.7 SUMMARY

This unit discusses some of the most important aspects of a good database design. The unit first defines the concept of referential integrity constraint and entity integrity constraints. It then discusses about different forms of the normalisation based on the concept of function dependency. A functional dependency highlights the relationships among the attributes of a relation. Different dependency among attributes leads to the problems of update anomalies, insertion anomalies and deletion anomalies. Different forms of normalisation decompose relations, using the FDs, to remove anomalies. The concept of FD is used for the process of normalisation.

This unit also defines the features of a good decomposition of a relation. The most important being the attribute preservation, dependency preservation and lossless-join decomposition. Finally, the unit summarises the rules of normalisation with the help of an example.

5.10 SOLUTIONS/ANSWERS

Check Your Progress 1

- 1) The following table shows the candidate keys and primary key of the relations

Relation	Candidate Keys	Primary key
S (Supplier)	SNO, SNAME*	SNO
P (Part)	PNO, PNAME*	PNO
J (Project)	PROJ NO, JNAME*	PROJNO
SPJ	(SNO+PNO+PROJNO)	(SNO+PNO+PROJNO)

* Only if the values are assumed to be unique, this may be incorrect for large systems.

- 2) SNO in S, PNO in P, PROJNO in J and (SNO+PNO+PROJNO) in SPJ should not contain NULL value. Also, no part of the primary key of SPJ, that is SNO or PNO or PROJNO should contain NULL value.
- 3) Foreign keys exist only in SPJ, where SNO references SNO of S; PNO references PNO of P; and PROJNO references PROJNO of J. The referential integrity necessitates that all matching foreign key values must exist.
- 4) You may select the following referential actions:

For operations like Delete and update, you should use referential action as RESTRICT. This selection would restrict loss of information from the SPJ relation, in case you delete a tuple in S or P or J relation. Insertion of records does not create a problem provided the referential integrity constraints are met.

Check Your Progress 2

- 1) The database suffers from all the anomalies; let us demonstrate these with the help of the following relation instance or state of the relation:

student_id	Student_name	bookID	Book_title	issuedOn	returnedOn
A 101	Abishek	0050	DBMS	15/01/05	25/01/05
R 102	Raman	0125	DS	25/01/05	29/01/05
A 101	Abishek	0060	Multimedia	20/01/05	NULL
R 102	Raman	0050	DBMS	28/01/05	NULL

Is there any data redundancy?

Yes, the information is getting repeated about student names and book Title. This may lead to an update anomaly in case of changes made to data value of the book. In addition, the library may be having many more books that have not been issued yet. The information of such books cannot be added to the relation as the primary key to the relation is: (student_id + bookID + issuedOn). (This would involve the assumption that the same book can be issued to the same student only once in a day). Thus, you cannot enter the information about bookID and book_title of those book, which has not been issued to a student. This is insertion anomaly. Similarly, you cannot enter student_id if a book is not issued to that student. This is also an insertion anomaly. As far as the deletion anomaly is concerned, suppose Abishek did not collect the Multimedia book, so this record needs to be deleted from the relation (tuple 3). This deletion will also remove the information about the Multimedia book that is its bookID and book_title. This is deletion anomaly for the given instance.

2) The FDs of the relation are:

$$\begin{aligned} \text{student_id} &\rightarrow \text{student_name} & (1) \\ \text{bookID} &\rightarrow \text{book_title} & (2) \\ \text{bookID}, \text{student_id}, \text{issuedOn} &\rightarrow \text{returnedOn} & (3) \end{aligned}$$

Why is the attribute issuedOn on the left hand of the FD above? Because a student, for example, Abishek can be issued the book having the bookID 0050 again on a later date, let say in February after Raman has returned it. Also note that returnedOn may have NULL value, but that is allowed in the FD. FD only necessitates that the value may be NULL or any specific data that is consistent across the instance of the relation.

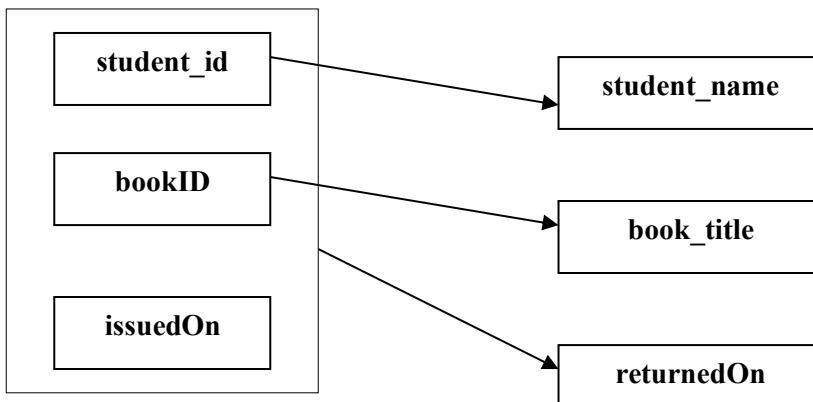
Just one candidate key, which is also chosen as the primary key, is : bookID, student_id, issuedOn.

Some interesting domain and procedural constraints in this database are:

- A book cannot be issued again unless it is returned
- A book can be returned only after the date on which it has been issued.
- A student can be issued a limited/maximum number of books at a time.

You will study about these constraints later in the Block.

3) The relation is in 1NF. The following is a FD diagram for the relation:



1NF to 2NF:

The composite key to the relation is (student_id + bookID). The attributes student_name depends on student_id, which is part of the composite key, likewise book_title attributes is dependent on bookID attribute, which is part of the composite key so the decomposition based on following FDs would be:

MEMBER (student_id, student_name)	[Reason: FD (1)]
BOOK (bookID, book_name)	[Reason: FD (2)]
ISSUE_RETURN (student_id, bookID, issuedOn, returnedOn)	[Reason: FD (3)]

2NF to 3 NF and BCNF:

All the relations are in 3NF and BCNF also. As there is no dependence among non-key attributes and there is no overlapping candidate keys.

Please note that the decomposed relations have no anomalies. Let us map the relation instance here:

MEMBER		BOOK		ISSUE_RETURN
Member - ID	Book - Code	Member - Name	Book - Name	
A 101	0050	Abhishek	DBMS	
R 102	0060	Raman	Multimedia	
	0125		OS	
Member - ID	Book - Code	Issue - Date		Return - Date
A 101	0050	15/01/05		25/01/05
R 102	0125	25/01/05		29/01/05
A 101	0060	20/01/05		NULL
R 102	0050	28/01/05		NULL

- i) There is no redundancy, so no update anomaly is possible in the relations above.
- ii) The insertion of new book and new student can be done in the BOOK and MEMBER tables respectively without any issue of book to student or vice versa. So, no insertion anomaly.
- iii) Even on deleting record 3 from ISSUE_RETURN it will not delete the information the book 0060 titled as Multimedia as this information is in separate table. So, no deletion anomaly.

Check Your Progress 3

- 1) Dependency preservation within a relation helps in enforcing constraints that are implied by dependency over a single relation. In case, you do not preserve dependency then constraints might be enforced on more than one relation that is quite troublesome and time consuming.
- 2) A lossless join decomposition is one in which you can reconstruct the original table without loss of information that means exactly the same tuples are obtained on taking join of the relations obtained on decomposition. Lossless join decomposition requires that decomposition should be carried out on the basis of FDs.
- 3) The steps of Normalisation are:

1. Remove repeating groups of each of the multi-valued attributes.
2. Then remove redundant data and its dependence on part key.
3. Remove columns from the table that are not dependent on the key, that is remove transitive dependency.
4. Check if there are overlapping composite candidate keys. If yes, check for any dependency among the non-overlapping attributes of the composite candidate keys; and remove such dependency.

**Database Integrity and
Normalisation**



UNIT 6 HIGHER NORMAL FORMS

Structure

- 6.0 Introduction
 - 6.1 Objectives
 - 6.2 Multivalued Dependency
 - 6.3 Fourth Normal Form (4NF)
 - 6.4 Join Dependency
 - 6.5 5NF
 - 6.6 Other Normal Forms
 - 6.7 Summary
 - 6.8 Solutions/Answers
-

6.0 INTRODUCTION

In the previous unit of this block, you have gone through the concept of single-valued dependency, called functional dependency (FD). Further, the previous Unit discussed about different forms of normalisations that can be arrived at by removing different types of single-valued dependency, viz. 1NF, 2NF, 3NF and BCNF. However, relational databases may have several other types of dependencies, which results in redundancy in a relation. Such dependencies, thus causes the update anomaly, insertion anomaly and deletion anomaly in a relation. Therefore, more normal forms has been designed to address this problem.

This Unit focusses on the higher normal forms, namely fourth normal form (4NF), which is based on the concept of multivalued dependency (MVD); and fifth normal form (5NF), which is based on the concept of Join dependency. The unit also introduces you to other normal forms. For more details on other normal forms, you may refer to the further readings. In general, these higher normal forms are not very popular among industries, however they propose a good theoretical perspectives for the database design.

6.1 OBJECTIVES

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

- Explain the concept of multivalued dependencies;
 - Perform normalisation upto 4NF;
 - Explain the join dependency;
 - Explain the 5NF;
 - Define other normal forms.
-

6.2 MULTIVALUED DEPENDENCIES

An attribute in a relational model represents only a single value information. How will you represent the information if a particular attribute is multivalued? Such multivalued attributes would require that

information of all other attribute is repeated in an instance of the relation. This will result in multiple tuples in a single relation to represent information about one entity. The primary key of such a relation would be a composite key involving the primary key of the entity and the multivalued attribute. This situation becomes worse, if an entity has more than one multivalued attributes. Such an entity will be represented by several tuples. The following example explains this situation:

Example 1: Let us consider a relation ‘employee’.

emp (e#, project, tool)

An employee can work on several projects and an employee has skills in several tools, therefore, there is no functional dependency in this relation. However, this relation has two multivalued attributes: *project* and *tool*.

The attributes *project* and *tool* are assumed to be independent of each other, as any project can use any tool. The following table (not relation) defines this situation:

e# (Employee Number)	Project	Tool
E001	DBMS, Ecommerce	Python, Virtualization
E002	Ecommerce, Bank Automation	PostgreSQL, Virtualization
E003	Bank Automation	PostgreSQL

Figure 6.1: Sample Data

However, to represent this information through 1NF, you need to create the following relation:

e#	project	tool
E001	DBMS	Python
E001	DBMS	Virtualization
E001	Ecommerce	Python
E001	Ecommerce	Virtualization
E002	Ecommerce	PostgreSQL
E002	Ecommerce	Virtualization
E002	Bank Automation	PostgreSQL
E002	Bank Automation	Virtualization
E003	Bank Automation	PostgreSQL

Figure 6.2: 1NF relation of data of Figure 6.1

This relation has no FD and primary key of the relation is composite key (*e#, project, tool*), therefore, the relation is in 3NF and BCNF. However, it suffers from the problem of redundancy, for example, the employee E001 is working on DBMS is appearing twice, so is that E001 knows the tool Python. This may lead to update anomaly. Further, you cannot insert information about a new employee, who knows the tools PostgreSQL, but has not been assigned to any project. In addition, if you remove the employee E003 from the Bank Automation project, the information that E003 has skill in PostgreSQL would be lost. Thus, the relation suffers from update, insertion and deletion anomalies.

How can you now normalise the relation, as this contains no FD? For addressing the issues related to such relations, we may first define the concept of multivalued dependency, which relates to relations that contains more than one multivalued attributes. Let us define the multivalued dependency formally for a relation $R(X, Y, Z)$, where X, Y and Z are a set of attributes.

Multivalued dependency (MVD): Given a relation $R(X, Y, Z)$, a MVD $X \rightarrow\rightarrow Y$ will hold in relation R if for a specific value of attribute set X , there is an associated set of values of attribute set Y , such that

these multiple-associated (zero or more) values of attributes Y depends only on value of attribute X. Further, values of attribute set Y have no dependence on the value of attribute set Z.

Hence, if $MVD X \rightarrow\rightarrow Y$ holds in R , another $MVD X \rightarrow\rightarrow Z$ would also hold, as function of the attribute set Y and attribute set Z is symmetrical.

In the *Example 1* given above, the employee number can determine all the projects, employee is working on, and also employee number can determine all the tools in which employee is skillful. Further, both the *project* and *tool* attributes are independent of each other. Therefore, the following MVDs holds in the relation of example 1:

$$\begin{aligned} e\# &\rightarrow\rightarrow project \\ e\# &\rightarrow\rightarrow tool \end{aligned}$$

Let us now define the concept of MVD in a different way. Consider the relation $R(X, Y, Z)$ having a multi-valued set of attributes Y associated with a value of X . Assume that the attributes Y and Z are independent, and Z is also multi-valued. Now, more formally, $X \rightarrow\rightarrow Y$ is said to hold for $R(X, Y, Z)$ if $t1$ and $t2$ are two tuples in R that have the same values for attributes X ($t1[X] = t2[X]$) then R also contains tuples $t3$ and $t4$ (not necessarily distinct) such that:

$$\begin{aligned} t1[X] &= t2[X] = t3[X] = t4[X] \\ t3[Y] &= t1[Y] \text{ and } t3[Z] = t2[Z] \\ t4[Y] &= t2[Y] \text{ and } t4[Z] = t1[Z] \end{aligned}$$

For example, consider the Figure 6.2, the two such tuples are:

Tuple 1: E001	DBMS	Python
Tuple 4: E001	Ecommerce	Virtualization

Then two more tuples must exists as follows:

E001	DBMS	Virtualization
E001	Ecommerce	Python

Please check these two are Tuple 2 and Tuple 3 in the Figure 6.2

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

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

The theory of multivalued dependencies is very similar to that for functional dependencies. Given a set of MVDs, say D , you can find D^+ , the closure of D using a set of axioms. We do not discuss the axioms here. You may refer this topic in further readings.

Before explaining the 4NF of a relation, one more term needs to be defined. It is the Trivial MVD, which is defined next.

Trivial MVD: A MVC $X \rightarrow\rightarrow Y$ is called trivial MVD if either Y is a subset of X or X and Y together form the relation R .

The MVD is trivial since it results in no constraints being placed on the relation. For example, consider a relation *dependent* ($e\#$, $e\text{dependent}\#$). An employee can have zero or more dependents, therefore, $e\#$ uniquely determines the **values** of $e\text{dependent}\#$. However, this MVD is trivial, as $e\#$, $e\text{dependent}\#$ together forms the relation *dependent*. This *dependent* relation cannot be decomposed any further.

Therefore, a relation having non-trivial MVDs must have at least three attributes; two of them would be multivalued and not dependent on each other. Non-trivial MVDs result in the relation having some constraints on it since all possible combinations of the multivalued attributes are then required to be present in the relation. In the next section, we discuss, how MVD can be used to decompose a relation into fourth normal form.

6.3 FOURTH NORMAL FORM

In this section, first we define the fourth normal form (4NF) and then present an example about how MVD can be used to decompose a relation to 4NF.

A relation R is in 4NF if for all the multivalued dependencies ($X \rightarrow\rightarrow Y$) any one of the following clauses holds:

- the multivalued dependencies ($X \rightarrow\rightarrow Y$) is trivial,
- X is a candidate key for R .

The dependency $X \rightarrow\rightarrow \emptyset$ or $X \rightarrow\rightarrow Y$ in a relation R (X, Y) is trivial, since they must hold for all R (X, Y). In this case, R (X, Y) is in 4NF. Similarly, if in a relations R (A, B, C) with only three attributes, if a trivial MVD $(A, B) \rightarrow\rightarrow C$ holds, then R (A, B, C) is in 4NF.

If a relation has more than one multivalued attributes, you should decompose it into fourth normal form using the following rules of decomposition:

For a relation $R(X, Y, Z)$, if it contains two nontrivial MVDs $X \rightarrow\rightarrow Y$ and $X \rightarrow\rightarrow Z$, then decompose the relation into $R_1(X, Y)$ and $R_2(X, Z)$ or more specifically, if there holds a non-trivial MVD in a relation R (X, Y, Z) of the form $X \rightarrow\rightarrow Y$, such that $X \cap Y = \emptyset$, that is the set of attributes X and Y are disjoint, then R must be decomposed to $R_1(X, Y)$ and $R_2(X, Z)$, where Z represents all attributes other than those in X and Y .

Intuitively R is in 4NF if

- (1) All dependencies are a result of keys.
- (2) When multivalued dependencies exist, a relation should not contain two or more independent multivalued attributes.

The decomposition of a relation to achieve 4NF would normally result in not only reduction of redundancies but also avoidance of anomalies.

Example 2: Normalise the relation *emp* ($e\#$, *project*, *tool*) shown in Figure 6.2 using the MVDs:

$e\# \rightarrow\rightarrow \text{project}$ and $e\# \rightarrow\rightarrow \text{tool}$.

The relation has no FD, but two MVDs, therefore, the relation is in BCNF but not 4NF. To decompose the relation into 4NF, you may use any of the MVD. The decomposed relation would be:

empproj ($e\#$, *project*) with MVD $e\# \rightarrow\rightarrow \text{project}$, which is now a trivial MVD; and

empproj (e#, project) with MVD $e\# \rightarrow\!\!\!\rightarrow$ tool, which is a trivial MVD.

On decomposition of the relation in Figure 6.2, by taking the projections as stated above, the relational state of the decomposed relations would be:

<i>empproj</i>		<i>emptool</i>	
<u>e#</u>	<u>project</u>	<u>e#</u>	<u>tool</u>
E001	DBMS	E001	Python
E001	Ecommerce	E001	Virtualization
E002	Ecommerce	E002	PostgreSQL
E002	Bank Automation	E002	Virtualization
E003	Bank Automation	E003	PostgreSQL

Figure 6.3: Decomposition of *emp (e#, project, tool)* relation to 4NF

You can observe the following in Figure 6.3

- The key to relations empproj and emptool are (e#, project) and (e#, tool) respectively.
- There is no redundancy in empproj and emptool relations.
- Both the relations do not suffer from any anomaly.

So far, you are able to decompose a relation based on FD and MVD. Are there any other form of dependencies? Researcher has shown several kinds of dependencies. However, for this course we will discuss about one more type of dependency, called the join dependency, which is discussed next.

☞ Check Your Progress 1

- 1) What are Multi-valued Dependencies? When can you say that a constraint X is multi-valued dependency?

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

- 2) Identify the MVDs in the following relation. Decompose the relation into 4NF using the MVDs

EMP

ENAME	PNAME	DNAME
Rohan	Big Data	AI Unit
Rohan	Machine Learning	Analytics
Rohan	Big Data	Analytics
Rohan	Machine Learning	AI Unit

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

- 3) Convert the following relation to 4NF relations.

SUPPLY

SNAME	PARTNAME	PROJNAME
XYZ Pvt Ltd	Bolt	Big Data
XYZ Pvt Ltd	Nut	Machine Learning
ABC Ltd	Bolt	Machine Learning
Info Comm Ltd	Nut	AI
ABC Ltd	Nail	Big Data
ABC Ltd	Bolt	Big Data
XYZ Pvt Ltd	Bolt	Machine Learning

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

6.4 JOIN DEPENDENCIES

As discussed in the previous section, a relation that suffers from insertion, update and deletion anomalies is decomposed using either FD or MVD. The normal forms require that a given relation R , if not in the given normal form, should be decomposed in two relations to meet the requirements of the normal form. However, in some cases, a relation can have problems like redundancy leading to anomalies, yet it cannot be decomposed in two relations without loss of information. In such cases, it may be possible to decompose the relation in three or more relations. When does such a situation arise? Such cases normally happen when a relation has at least three attributes such that all those values are totally independent of each other. It may also be the case when a ternary relationship exists among three entities.

Following example explains this in detail. Consider a relation

ProjToolEmp (projectid, toolid, empid)

There are no constraints on this relation that is:

- Any project can use any tools
- Any tool can be used by any employee
- Any employee can work on any project
- No employee would use all the tools
- No employee would work on all the projects
- No project uses all the tools
- All the three attributes are independent of each other

Assume that the relation has the following relational instance:

Tuple#	projectid	toolid	empid
1	P1	T1	E1
2	P2	T2	E2
3	P1	T2	E2

Figure 6.4: A ternary relation with all independent attributes

The relation above does not have any FDs and MVDs since the attributes projectid, toolid and empid are independent; they are related to each other only by the pairings that have significant information in them.

For example, the first tuple of relation states that employee E1 works on P1 project, which uses tool T1. The key to the relation is the composite key (projectid, toolid, empid). The relation is in 4NF, but still suffers from the insertion, deletion, and update anomalies, as the information that Employee E1 can use tool T1 is redundant in tuple 3. Similarly, information like project P3 uses tool T1 cannot be inserted in the relation, as no employee has started working on the project. Likewise, on deleting tuple 2 from the relation, may delete the information like Employee E2 can use tool T2. Therefore, you need to decompose the relation to minimize the anomalies. As there are no FDs or MVDs in this relation, there is no basis of decomposition. You may try to see what happens when you decompose the relation into the following two relations:

ProjectTool	
projectid	toolid
P1	T1
P2	T2
P1	T2

ProjectEmployee	
projectid	empid
P1	E1
P2	E2
P1	E2

Figure 6.5: A decomposition of ternary relation of Figure 6.4

What happens when you take Join of the two relations on the attribute *projectid*:

Tuple#	projectid	toolid	empid
1	P1	T1	E1
2	P1	T1	E2
3	P2	T2	E2
4	P1	T2	E1
5	P1	T2	E2

Figure 6.6: ProjectTool JOIN ProjectEmployee

You may notice that the tuples 2 and tuple 4 in the relation state in Figure 6.6 were not existent in original relational state given in Figure 6.4. Thus, this decomposition is a lossy decomposition. So, what should be done to remove the anomalies?

In such situation, you may have to decompose the relation into three relations. Two of which are already shown in Figure 6.5. A third relation as shown in Figure 6.7 must be added.

ToolEmployee	
toolid	empid
T1	E1
T2	E2

Figure 6.7: The third relation

In order to obtain the original relation, you need to perform the following join operation:

ProjectTool JOIN ProjectEmployee JOIN ToolEmployee

Figure 6.6 represents (*ProjectTool JOIN ProjectEmployee*) which can be joined with *ToolEmployee*

Tuple#	projectid	toolid	empid
1	P1	T1	E1
2	P1	T1	E2
2	P2	T2	E2

<u>4</u>	<u>P1</u>	<u>T2</u>	<u>E1</u>
3	P1	T2	E2

Figure 6.8: ProjectTool JOIN ProjectEmployee JOIN ToolEmployee

Please note that Tuple 2 and Tuple 4 of Figure 6.6 will not be part of Figure 6.8, as there are no matching tuples in Figure 6.7. Thus, you can observe that Figure 6.4 and Figure 6.8 are identical.

Therefore, the relation ProjToolEmp (projectid, toolid, empid), which has no FD and MVD can be decomposed into three relations, which can be joined to form the original relation. This forms the concept of join dependency, which is explained next.

Join Dependency (JD): A relation R satisfies join dependency over the projections (P_1, P_2, \dots, P_n) if and only if every instance of R, the join of P_1, P_2, \dots, P_n creates the instance of R.

For example, the instance of ternary relation ProjToolEmp (projectid, toolid, empid), as shown in Figure 6.4, when decomposed to two relations, as shown in Figure 6.5, does not satisfy the join dependency. This is shown in Figure 6.6. On addition of the third relation, as shown in Figure 6.7, the three projections ProjectTool (projectid, toolid), ProjectEmployee(projectid, empid) and ToolEmployee (toolid, empid) satisfies the join dependency as JOIN on the three projections, viz. ProjectTool (projectid, toolid), ProjectEmployee(projectid, empid) and ToolEmployee (toolid, empid), is same as the instance of the relation. Thus, the three projections, as stated above, satisfies the join dependency.

6.4 FIFTH NORMAL FORM

The fifth normal form (5NF) (Project-Join normal form (PJNF)) deals with join-dependencies, which is a generalisation of the MVD. The aim of fifth normal form is to have relations that cannot be decomposed further. A relation in 5NF cannot be constructed from several smaller relations.

A relation R is in 5NF if for all the join dependencies any one of the following clauses holds:

- (a) join-dependency (P_1, P_2, \dots, P_n) is trivial (that is, one of the P_i 's is R)
- (b) Every P_i is a super key of R.

For example, the instance of ternary relation ProjToolEmp (projectid, toolid, empid), as shown in Figure 6.4, has a Join Dependency (ProjectTool (projectid, toolid), ProjectEmployee(projectid, empid), ToolEmployee (toolid, empid)). Therefore, this relation is not in 5NF, as it violates the clauses of the 5NF, given above. You can decompose the relation ProjToolEmp (projectid, toolid, empid) into its projections as follows:

ProjectTool (projectid, toolid)
 ProjectEmployee(projectid, empid)
 ToolEmployee(toolid, empid)

Each of these three relations are in 5NF, as they have trivial join dependency. The instance of each of the decomposed relations is shown in Figure 6.5 and Figure 6.7. You may also observe that this decomposition is lossless join decomposition. It may be noted that every MVD is also a join dependency. Therefore, every PJNF (5NF) schema is also in 4NF. A relation schema that has Join Dependencies and suffers from anomalies, can be decomposed into projections, as per the join dependency. The new relations would be in PJNF schema.

Check Your Progress 2

1) What is join dependency?

.....
.....

2) Define 5NF.

.....
.....

3) Convert the relational instance of SUPPLY relation, as given in question 3 of check your progress 1. to 5NF.

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

6.5 OTHER NORMAL FORMS

The researchers of database systems have found additional dependencies and normal forms. This section introduces the basic concept behind these forms. First, we define some additional types of dependencies.

Inclusion Dependency

The inclusion dependency has been designed for two specific type of database constraints that are not defined by the concept of FD, MVD and Join dependency. These two constraints are:

- Foreign key constraints
- Class / subclass relationships

Please note that these constraints are between two relations. Therefore, requires a new form of formal definition. We define it in the context of foreign key constraints.

Consider two relations R1 and R2, which are related through a foreign key constraint such that a set of attributes in R1, say X, is the foreign key that references the relation R2 on set of attributes, say Y. Please note that attribute sets X and Y must have similar number of attributes and similar domains, so that foreign key constraint is applicable. The inclusion dependency for such a situation will be defined as follows:

An **inclusion dependency** (denoted as $R1.X < R2.Y$) if the following relationship between the projections holds:

$$\pi_X(r1) \subseteq \pi_Y(r2) \tag{1}$$

Where r1 and r2 are the instances of relation R1 and R2 respectively at the same instance of time.

For example, consider the following two relational instances:

Relation: DEPT

deptID	deptName	deptlocation
D01	Marketing	Delhi
D02	Production	Mumbai
D03	HR	Delhi

Relation: EMP

empID	empName	salary	department
E01	ABC	30000	D01
E02	BCD	40000	D01
E03	CDE	45000	D02
E04	EFG	35000	D02

There exists a foreign key between the two relations, viz. department in EMP relation references deptID in DEPT relation. Therefore, the inclusion dependency $EMP.department < DEPT.deptID$ must hold for the given relational instances.

The equation (1) for this may be (assuming that relational instance of DEPT is dept and EMP is emp):

$\pi_X(r1)$ is $\pi_{department}(emp)$, which would be:

emp
department
D01
D02

and $\pi_Y(r2)$ is $\pi_{deptID}(dept)$, which would be:

dept
deptID
D01
D02
D03

You may observe that the inclusion dependency $EMP.department < DEPT.deptID$ holds, as $\pi_{department}(emp) \subseteq \pi_{deptID}(dept)$.

Template Dependency

The template dependency is specified in the form of a template and can be used to represent any generic dependency. These dependencies can define any constraints in the form of a template. A template consists of two parts – the first part which shows the tuples that may exist in a relation is called the hypotheses, which are followed by conclusion of the template.

A template dependency can be used to represent any dependency in this form. The following example shows how a FD can be represented using a template dependency.

Example: Consider a relation RESULT (studentid, courseid, grade) with the FD

studentid, courseid \rightarrow grade

Represent this FD using template dependency.

Description	<u>studentid</u>	<u>courseid</u>	grade
Hypothesis	S1	C1	G1

	S1	C1	G2
Conclusion	G1 = G2		

The following example shows, how MVDs can be represented using template dependency.

Example: Consider a relation EMPLOYEE (e#, project, tool) with the set of MVDs

$e\# \rightarrow\!\!\! \rightarrow$ project and $e\# \rightarrow\!\!\! \rightarrow$ tool.

The following template dependency defines these MVDs:

Description	<u>e#</u>	project	tool
Hypothesis	E1	P1	T1
	E1	P2	T2
Conclusion	E1	P1	T2
	E1	P2	T1

One example of this MVD is shown with attribute values in the following table:

Description	<u>e#</u>	project	tool
Hypothesis	E001	DBMS	Python
	E001	Ecommerce	Virtualization
Conclusion	E001	DBMS	Virtualization
	E001	Ecommerce	Python

However, the actual use of template dependency may be to represent the constraints that cannot be represented using FD, MVD and join dependency. The following example explains this.

Example: Consider the relation EMP (empID, empName, salary, headID). In this relation the attribute headID represents empID of the head of the employee. You want to put a constraint in the relation that the employee cannot be given more salary than his/her head. The following template dependency would define this constraint:

Description	<u>empID</u>	empName	salary	headID
Hypothesis	E1	N1	S1	E2
	E2	N2	S2	E3
Conclusion	$S2 \geq S1$			

The Domain-Key Normal Form (DKNF)

The Domain-Key Normal Form (DKNF) is beyond 5NF and proposes to make a set of relations free of all anomalies. The DKNF was defined as a consequence of Fagin's theorem that states:

“A relation is in DKNF if every constraint on the relation is a logical consequence of the definitions of keys and domains.”

Let us define the key terms used in the definition above – *constraint*, *key* and *domain* in more detail. These terms were defined as follows:

Key can be either the primary keys or the candidate key. Let R be a relation schema with $CK \subseteq R$. A key CK requires that CK be a super key for schema R such that $CK \rightarrow R$. Please note that a key declaration is a functional dependency but not all functional dependencies are key declarations.

Domain is the set of definitions of the contents of attributes and any limitations on the kind of data to be stored in the attribute. Let A be an attribute and **dom** be a set of values. The domain declaration can be stated as $A \sqsubseteq \text{dom}$. It requires that the values of A in all the tuples of R be values from the set **dom**.

Constraint is a well-defined rule that is to be upheld by any set of legal data of R . A *general constraint* is defined as a predicate on the set of all the relations of a given schema. The MVDs, JDs are the examples of general constraints. However, a general constraint need not be just functional, multivalued, or join dependency. For example, the first two digits of your enrolment number represents the year in which you have taken admission. Assuming that MCA is the only programme of the university, whose maximum duration is 4 years. Also, assume that admission to this programme was started in 2021. Therefore, in the year 2026, there would be two types of students, viz. students whose enrolment number starts with 21 and students whose registration number starts with 22, 23, 24, 25 and 26. The registration of the students, whose registration starts with 21 would become invalid. Therefore, the general constraint for such a case may be: “If the first two digit of $t[\text{enrolmentnumber}]$ is 21, then $t[\text{marks}]$ are valid.” The t represents a tuple and *enrolmentnumber* and *marks* are the attributes.

The constraint suggests that the database design is not in DKNF. To convert this design to DKNF design, you need two schemas as:

Validstudentschema = (*enrolmentnumber*, *subject*, *marks*)

Invalidstudentschema = (*enrolmentnumber*, *subject*, *marks*)

Please note that the schema of valid students requires that the enrolment number of the students begin with 22. The resulting design is in DKNF.

Although DKNF is an aim of a database designer, it may not be implemented in a practical design.

☛ Check Your Progress 3

- 1) Define Inclusion Dependencies.

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

2) Define the template dependency.

.....
.....

3) What is the key idea behind DKNF?

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

6.6 SUMMARY

This unit explains the concept of multi-valued dependencies, which causes a relation to have data redundancy. This causes a relation to have data anomalies. MVD is a consequence of having a set of attributes in a relation that determines more than one multi-valued attributes, which are independent of each other. MVD forms the basis of decomposition of a relation into 4NF relations. Further, certain relations do not have any FDs and MVDs but have anomalies. Such relations, in general, consist of more than two independent attributes. These relations contain join dependency, that is the relation has several projections, which can be joined losslessly to produce original relation. The join dependency forms the basis for 5NF decomposition. The unit also discusses about the inclusion and template dependencies, which are designed to represent the constraints that cannot be assigned using FDs, MVDs and join dependencies. Finally, the unit introduces the concept of DKNF. You may refer to the further readings for more details on these topics.

6.7 SOLUTIONS/ANSWERS

Check Your Progress 1

- 1) An MVD is a constraint due to multi-valued attributes. A constraint is multi-valued if all the following conditions holds in a relation:
 - The relational must have at least three attributes out of which one should be multi-valued attribute.
 - One of the attributes can multi-determine the other two attributes.
 - The other two attributes, as stated above should be independent of each other.

- 2) There are two MVDs that exists in the relation. These are:

$\text{ENAME} \rightarrow\!\!\!\rightarrow \text{PNAME}$ and $\text{ENAME} \rightarrow\!\!\!\rightarrow \text{DNAME}$

Due to these two MVDs the relation suffers from insertion, update and deletion anomalies. This relation can be decomposed into the following projections, which are in 4NF.

EMP_PROJECT

ENAME	PNAME
Rohan	Big Data
Rohan	Machine Learning

EMP_DNAME

ENAME	DNAME
Rohan	AI Unit
Rohan	Analytics

- 3) The relational instance of SUPPLY relation shows that all three attributes are independent of each other as any supplier can supply any part to any project. Thus, there is no FD or MVD in the relation. Therefore, the relation is already in 4NF.

Check Your Progress 2

- 1) A join dependency is defined for a relation R and its projections, say R_1, R_2, \dots, R_n , as follows:
The join of relations R_1, R_2, \dots, R_n must be equal to the relation R .
- 2) A relation is said to be in 5NF if either it has trivial join dependencies, or every projection R_i of the relation R is a super key of R .
- 3) All the attributes of relation SUPPLY are independent of each other. Does the relation have join dependency $(\text{SNAME}, \text{PARTNAME}), (\text{PARTNAME}, \text{PROJNAME}), (\text{PROJNAME}, \text{SNAME})$? The three projections for the given instance of SUPPLY would be:

R1

SNAME	PARTNAME
XYZ Pvt Ltd	Bolt
XYZ Pvt Ltd	Nut
ABC Ltd	Bolt
Info Comm Ltd	Nut
ABC Ltd	Nail

R2

PARTNAME	PROJNAME
Bolt	Big Data
Nut	Machine Learning

Bolt	Machine Learning
Nut	AI
Nail	Big Data

R3

SNAME	PROJNAME
XYZ Pvt Ltd	Big Data
XYZ Pvt Ltd	Machine Learning
ABC Ltd	Machine Learning
Info Comm Ltd	AI
ABC Ltd	Big Data

The join of the first two projections (R1 and R2) on PARTNAME would be:

JoinR1R2

SNAME	PARTNAME	PROJNAME
XYZ Pvt Ltd	Bolt	Big Data
XYZ Pvt Ltd	Bolt	Machine Learning
XYZ Pvt Ltd	Nut	Machine Learning
XYZ Pvt Ltd	Nut	AI
ABC Ltd	Bolt	Big Data
ABC Ltd	Bolt	Machine Learning
Info Comm Ltd	Nut	Machine Learning
Info Comm Ltd	Nut	AI
ABC Ltd	Nail	Big Data

The join of relations JoinR1R2 with R3 on SNAME, PROJNAME would be:

SNAME	PARTNAME	PROJNAME
XYZ Pvt Ltd	Bolt	Big Data
XYZ Pvt Ltd	Bolt	Machine Learning
XYZ Pvt Ltd	Nut	Machine Learning
ABC Ltd	Bolt	Big Data
ABC Ltd	Bolt	Machine Learning
Info Comm Ltd	Nut	AI
ABC Ltd	Nail	Big Data

Please observe that the joined relation is same as the instance of relation SUPPLY. Therefore, the join dependency (SNAME, PARTNAME), (PARTNAME, PROJNAME), (PROJNAME, SNAME) holds over the relation SUPPLY. To convert SUPPLY relation to 5NF, you may decompose it into the three projections R1, R2 and R3 as shown above. Please notice that each of the relation R1, R2 and R3 now have trivial join dependency, therefore, are in 5NF.

Check Your Progress 3

- 1) An inclusion dependency defines the referential constraint on two attributes. An inclusion dependency holds if a projection on an attribute of a relation, which may be a foreign key, is a proper subset of projection of another attribute of another relation, where it is the primary key.
- 2) Template dependency is a generic representation of various dependencies. It consists of two parts – the hypothesis and conclusion.
- 3) DKNF is the “ultimate normal form”. It defines the terms keys, domains and constraints.



For Unit 7 & 8 : Please read the following unit of MCS-23 Block 2 Unit 1

UNIT 1 THE STRUCTURED QUERY

The Structured Query Language

Structure	Page Nos.
1.0 Introduction	5
1.1 Objectives	5
1.2 What is SQL?	6
1.3 Data Definition Language	7
1.4 Data Manipulation Language	9
1.5 Data Control	15
1.6 Database Objects: Views, Sequences, Indexes and Synonyms	16
1.6.1 Views	
1.6.2 Sequences	
1.6.3 Indexes and Synonyms	
1.7 Table Handling	19
1.8 Nested Queries	23
1.9 Summary	27
1.10 Solutions/Answer	28
1.11 Further Reading	34

1.0 INTRODUCTION

Database is an organised collection of information about an entity having controlled redundancy and serves multiple applications. DBMS (database management system) is an application software that is developed to create and manipulate the data in database. A query language can easily access a data in a database. SQL (Structured Query Language) is language used by most relational database systems. IBM developed the SQL language in mid-1979. All communication with the clients and the RDBMS, or between RDBMS is via SQL. Whether the client is a basic SQL engine or a disguised engine such as a GUI, report writer or one RDBMS talking to another, SQL statements pass from the client to the server. The server responds by processing the SQL and returning the results. The advantage of this approach is that the only network traffic is the initial query and the resulting response. The processing power of the client is reserved for running the application.

SQL is a data sub-language consisting of three built-in languages: Data definition language (DDL), Data manipulation language (DML) and Data control language (DCL). It is a fourth generation language. SQL has many more features and advantages. Let us discuss the SQL in more detail in this unit. It should be noted that many commercial DBMS may or may not implement all the details given in this unit. For example, MS-ACCESS does not support some of these features. Even some of the constructs may not be portable, please consult the DBMS Help for any such difference.

1.1 OBJECTIVES

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

- create, modify and delete database schema objects;
- update database using SQL command;
- retrieve data from the database through queries and sub-queries;
- handle join operations;
- control database access;
- deal with database objects like Tables, Views, Indexes, Sequences, and Synonyms using SQL.

1.2 WHAT IS SQL?

Structured Query Language (SQL) is a standard query language. It is commonly used with all relational databases for data definition and manipulation.

All the relational systems support SQL, thus allowing migration of database from one DBMS to another. In fact, this is one of the reasons of the major success of Relational DBMS. A user may be able to write a program using SQL for an application that involves data being stored in more than one DBMSs, provided these DBMS support standard SQL. This feature is commonly referred to as portability. However, not all the features of SQL implemented in one RDBMS are available in others because of customization of SQL. However, please note there is just ONE standard SQL.

SQL provides an interface where the user can specify “What” are the expected results. The query execution plan and optimisation is performed by the DBMS. The query plan and optimisation determines how a query needs to be executed. For example, if three tables X, Y and Z are to be joined together then which plan (X JOIN Y) and then Z or X JOIN (Y JOIN Z) may be executed. All these decisions are based on statistics of the relations and are beyond the scope of this unit.

SQL is called a non-procedural language as it just specifies what is to be done rather than how it is to be done. Also, since SQL is a higher-level query language, it is closer to a language like English. Therefore, it is very user friendly.

The American National Standard Institute (ANSI) has designed standard versions of SQL. The first standard in this series was created in 1986. It was called SQL-86 or SQL1. This standard was revised and enhanced later and SQL-92 or SQL-2 was released. A newer standard of SQL is SQL3 which is also called SQL- 99. In this unit we will try to cover features from latest standards. However, some features may be found to be very specific to certain DBMSs.

Some of the important features of SQL are:

- It is a non procedural language.
- It is an English-like language.
- It can process a single record as well as sets of records at a time.
- It is different from a third generation language (C& COBOL). All SQL statements define what is to be done rather than how it is to be done.
- SQL is a data sub-language consisting of three built-in languages: Data definition language (DDL), Data manipulation language (DML) and Data control language (DCL).
- It insulates the user from the underlying structure and algorithm.
- SQL has facilities for defining database views, security, integrity constraints, transaction controls, etc.

There are many variants of SQL, but the standard SQL is the same for any DBMS environment. The following table shows the differences between SQL and one of its superset SQL*Plus which is used in Oracle. This will give you an idea of how various vendors have enhanced SQL to an environment. The non-standard features of SQL are not portable across databases, therefore, should not be used preferably while writing SQL queries.

Difference between SQL and SQL*Plus

SQL	SQL*Plus
SQL is a language	SQL *Plus is an environment
It is based on ANSI (American National Standards Institute) standard	It is oracle proprietary interface for executing SQL statements

SQL	
It is entered into the SQL buffer on one or more lines	It is entered one line at a time, not stored in the SQL buffer
SQL cannot be abbreviated	SQL*Plus can be abbreviated
It uses a termination character to execute command immediately	It does not require termination character. Commands are executed immediately.
SQL statements manipulate data and table definition in the database	It does not allow manipulation of values in the database

1.3 DATA DEFINITION LANGUAGE

As discussed in the previous block, the basic storage unit of a relational database management system is a table. It organises the data in the form of rows and columns. But what does the data field column of table store? How do you define it using SQL?

The Data definition language (DDL) defines a set of commands used in the creation and modification of schema objects such as tables, indexes, views etc. These commands provide the ability to create, alter and drop these objects. These commands are related to the management and administrations of the databases. Before and after each DDL statement, the current transactions are implicitly committed, that is changes made by these commands are permanently stored in the databases. Let us discuss these commands in more detail:

CREATE TABLE Command

Syntax:

```
CREATE TABLE <table name>
Column_name1 data type (column width) [constraints],
Column_name2 data type (column width) [constraints],
Column_name3 data type (column width) [constraints],
.....,
);
```

Where table name assigns the name of the table, column name defines the name of the field, data type specifies the data type for the field and column width allocates specified size to the field.

Guidelines for creation of table:

- Table name should start with an alphabet.
- In table name, blank spaces and single quotes are not allowed.
- Reserve words of that DBMS cannot be used as table name.
- Proper data types and size should be specified.
- Unique column name should be specified.

Column Constraints: NOT NULL, UNIQUE, PRIMARY KEY, CHECK, DEFAULT, REFERENCES,

On delete Cascade: Using this option whenever a parent row is deleted in a referenced table then all the corresponding child rows are deleted from the referencing table. This constraint is a form of referential integrity constraint.

Example 1:

```
CREATE TABLE product
(
pno number (4) PRIMARY KEY,
pname char (20) NOT NULL,
qoh number (5) DEFAULT (100),
```

```
class char (1) NOT NULL,  
rate number (8,2) NOT NULL,  
CHECK ((class='A' AND rate<1000) OR  
(class='B' AND rate>1000 AND rate<4500) OR  
(class='C' AND rate>4500))  
);
```

The command above creates a table. Primary key constraint ensures that product number (pno) is not null and unique (both are the properties of primary key). Please note the use of data type char (20). In many implementations of SQL on commercial DBMS like SQL server and oracle, a data type called varchar and varchar2 is used respectively. Varchar basically is variable length character type subject to a maximum specified in the declarations. We will use them at most of the places later.

Please note the use of check constraints in the table created above. It correlates two different attribute values.

Example 2:

```
CREATE TABLE prodtrans  
(  
    pno number (4)  
    ptype char (1) CHECK (ptype in ('T','R','S')),  
    qty number (5)  
    FOREIGN KEY pno REFERENCES product (pno)  
    ON DELETE CASCADE);
```

In the table so created please note the referential constraint on the foreign key **pno** in **prodtrans** table to **product** table. Any product record if deleted from the product table will trigger deletion of all the related records (ON DELETE CASCADE) in the **prodtrans** table.

ALTER TABLE Command: This command is used for modification of existing structure of the table in the following situation:

- When a new column is to be added to the table structure.
- When the existing column definition has to be changed, i.e., changing the width of the data type or the data type itself.
- When integrity constraints have to be included or dropped.
- When a constraint has to be enabled or disabled.

Syntax

```
ALTER TABLE <table name> ADD (<column name> <data type>...);  
ALTER TABLE <table name> MODIFY (<column name> <data type>...);  
ALTER TABLE <table name> ADD CONSTRAINT <constraint name> <constraint type>(field name);  
ALTER TABLE <table name> DROP<constraint name>;  
ALTER TABLE <table name> ENABLE/DISABLE <constraint name>;
```

You need to put many constraints such that the database integrity is not compromised.

Example 3:

```
ALTER TABLE emp MODIFY (empno NUMBER (7));
```

DROP TABLE Command:

When an existing object is not required for further use, it is always better to eliminate it from the database. To delete the existing object from the database the following command is used.

Syntax:

```
DROP TABLE<table name>;
```

Example 4:

```
DROP TABLE emp;
```

The Structured Query
Language

1.4 DATA MANIPULATION LANGUAGE

Data manipulation language (DML) defines a set of commands that are used to query and modify data within existing schema objects. In this case commit is not implicit that is changes are not permanent till explicitly committed. DML statements consist of SELECT, INSERT, UPDATE and DELETE statements.

SELECT Statement

This statement is used for retrieving information from the databases. It can be coupled with many clauses. Let us discuss these clauses in more detail:

1. Using Arithmetic operator

Example 5:

```
SELECT ENAME, SAL, SAL+300  
FROM EMP;
```

2. Operator Precedence

The basic operators used in SQL are * / + -

Operators of the same priority are evaluated From Left to right

Parentheses are used to force prioritized evaluation.

Example 6:

```
SELECT ENAME, SAL, 12*SAL+100  
FROM EMP;
```

```
SELECT ENAME, SAL, 12*(SAL+100)  
FROM EMP;
```

3. Using Column aliases

Example 7:

To print column names as NAME and ANNUAL SALARY

```
SELECT ENAME "NAME", SAL*12 "ANNUAL SALARY"  
FROM EMP;
```

4. Concatenation operator

Example 8:

Printing name and job as one string as column name employees:

```
SELECT ENAME||JOB "EMPLOYEES"  
FROM EMP;
```

5. Using Literal Character String

Example 9:

To print <name> IS A <job> as one string with column name employee

```
SELECT ENAME || ' IS A ' || JOB AS "EMPLOYEE"  
FROM EMP;
```

6. To eliminate duplicate rows (distinct operator)

Example 10:

```
SELECT DISTINCT DEPTNO  
FROM EMP;
```

7. **Special comparison operator** used in where Clause

- a. **between...and...** It gives range between two values (inclusive)

Example 11:

```
SELECT ENAME, SAL  
FROM EMP  
WHERE SAL BETWEEN 1000 AND 1500;
```

- b. **In (list):** match any of a list of values

Example 12:

```
SELECT EMPNO, ENAME, SAL, MGR  
FROM EMP  
WHERE MGR IN (7902, 7566, 7788);  
7902, 7566, and 7788 are Employee numbers
```

- c. **Like:** match a character pattern

- Like operator is used only with char and Varchar2 to match a pattern
- % Denotes zero or many characters
- _ Denotes one character
- Combination of % and _ can also be used

Example 13:

- (I) List the names of employees whose name starts with 's'

```
SELECT ENAME FROM EMP  
WHERE ENAME LIKE 'S%';
```

- (II) List the ename ending with 's'

```
SELECT ENAME FROM EMP  
WHERE ENAME LIKE '%S';
```

- (III) List ename having I as a second character

```
SELECT ENAME FROM EMP  
WHERE ENAME LIKE '_I%';
```

- d. **Is null operator**

Example 14:

to find employee whose manage-id is not specified

```
SELECT ENAME, MGR FROM EMP  
WHERE MGR IS NULL;
```

8 Logical Operators

Rules of Precedence:

Order evaluated	Operator
1	All comparison operators

2	NOT
3	AND
4	OR

Example 15: To print those records of salesman or president who are having salary above 15000/-

Select ename, job, sal from emp
Where job = ‘SALESMAN’ or job = ‘PRESIDENT’
And sal>15000;

The query formulation as above is incorrect for the problem statement. The correct formulation would be:

```
SELECT ENAME, JOB, SAL FROM EMP
WHERE (JOB = ‘SALESMAN’ OR JOB = ‘PRESIDENT’)
AND SAL>15000;
```

9. Order by clause

- It is used in the last portion of select statement
- By using this rows can be sorted
- By default it takes ascending order
- DESC: is used for sorting in descending order
- Sorting by column which is not in select list is possible
- Sorting by column Alias

Example 16:

```
SELECT EMPNO, ENAME, SAL*12 “ANNUAL”
FROM EMP
ORDER BY ANNUAL;
```

Example 17: Sorting by multiple columns; ascending order on department number and descending order of salary in each department.

```
SELECT ENAME, DEPTNO, SAL
FROM EMP
ORDER BY DEPTNO, SAL DESC;
```

10. Aggregate functions

- Some of these functions are count, min, max, avg.
- These functions help in getting consolidated information from a group of tuples.

Example 18: Find the total number of employees.

```
SELECT COUNT(*)
FROM EMP;
```

Example 19: Find the minimum, maximum and average salaries of employees of department D1.

```
SELECT MIN(SAL), MAX(SAL), AVG(SAL)
FROM EMP
WHERE DEPTNO = ‘D1’ ;
```

11. Group By clauses

- It is used to group database tuples on the basis of certain common attribute value such as employees of a department.
- WHERE clause still can be used, if needed.

Example 20: Find department number and Number of Employees working in that department.

```
SELECT DEPTNO, COUNT(EMPNO)
FROM EMP
GROUP BY DEPTNO;
```

Please note that while using group by and aggregate functions the only attributes that can be put in select clause are the aggregated functions and the attributes that have been used for grouping the information. For example, in the example 20, we cannot put ENAME attribute in the SELECT clause as it will not have a distinct value for the group. Please note the group here is created on DEPTNO.

12. Having clause

- This clause is used for creating conditions on grouped information.

Example 21: Find department number and maximum salary of those departments where maximum salary is more than Rs 20000/-.

```
SELECT DEPTNO, MAX(SAL)
FROM EMP
GROUP BY DEPTNO
HAVING MAX(SAL) > 20000;
```

INSERT INTO command:

- Values can be inserted for all columns or for the selected columns
- Values can be given through sub query explained in section 1.8
- In place of values parameter substitution can also be used with insert.
- If data is not available for all the columns, then the column list must be included following the table name.

Example 22: Insert the employee numbers, an increment amount of Rs.500/- and the increment date-today (which is being entered through function SYSDATE) of all the managers into a table INCR (increment due table) from the employee file.

```
INSERT INTO INCR
SELECT EMPNO, 500, SYSDATE FROM EMP
WHERE JOB = 'MANAGER';
```

Example 23: Insert values in a table using parameter substitution (& operator is used for it 1, 2 are the field numbers).

```
INSERT INTO EMP
VALUES (&1,'&2','&3', &4, &5, &6,NULL, &7);
Please note these values needs to be supplied at run time.
```

UPDATE Command:

Syntax is
UPDATE <table name>
SET <column name> = <value>
WHERE <condition>;

Sub query in the UPDATE command:

Example 24: Double the commission for employees, who have got at least 2 increments.

```
UPDATE EMP
SET COMM = COMM * 2
WHERE 2 <= (
    SELECT COUNT (*) FROM INCR
    WHERE INCR.EMPNO = EMP.EMPNO
    GROUP BY EMPNO);
```

Please note the use of subquery that counts the number of increments given to each employee stored in the INCR table. Please note the comparison, instead of \geq 2, we have written reverse of it as 2 \leq

DELETE Command

In the following example, the deletion will be performed in EMP table. No deletion will be performed in the INCR table.

Example 25: Delete the records of employees who have got no increment.

```
DELETE FROM EMP
WHERE EMPNO NOT IN (SELECT EMPNO FROM INCR);
```

☛ Check Your Progress 1

1) What are the advantages of SQL? Can you think of some disadvantages of SQL?

2) Create the Room, Booking, and Guest tables using the integrity enhancement features of SQL with the following constraints:

- (a) Type must be one of Single, Double, or Family.
- (b) Price must be between Rs.100/- and Rs.1000/-.
- (c) roomNo must be between 1 and 100.
- (d) booking dateFrom and dateTo must be greater than today's date.
- (e) The same room cannot be double booked.
- (f) The same guest cannot have overlapping bookings.

3) Define the function of each of the clauses in the SELECT statement. What are the restrictions imposed on these clauses?

4) Consider the supplier relations.

S

SNO (Supplier Number)	SNAME (Supplier Name)	STATUS	CITY
S1	Prentice Hall	30	Calcutta
S2	McGraw Hill	30	Mumbai
S3	Wiley	20	Chennai
S4	Pearson	40	Delhi
S5	Galgotia	10	Delhi

SP

SNO	PNO (Part Number)	Quantity
S1	P1	300
S1	P2	200
S2	P1	100
S2	P2	400
S3	P2	200
S4	P2	200

- a) Get supplier numbers for suppliers with status > 20 and city is Delhi
- b) Get Supplier Numbers and status for suppliers in Delhi in descending order of status.
- c) Get all pairs of supplier numbers such that the two suppliers are located in the same city. (Hint: It is retrieval involving join of a table with itself.)
- d) Get unique supplier names for suppliers who supply part P2 without using IN operator.
- e) Give the same query above by using the operator IN.
- f) Get part numbers supplied by more than one supplier. (Hint : It is retrieval with sub-query block, with inter block reference and same table involved in both blocks)
- g) Get supplier numbers for suppliers who are located in the same city as supplier S1. (Hint: Retrieval with sub query and unqualified comparison operator).
- h) Get supplier names for suppliers who supply part P1. (Hint : Retrieval using EXISTS)
- i) Get part numbers for parts whose quantity is greater than 200 or are currently supplied by S2. (Hint: It is a retrieval using union).
- j) Suppose for the supplier S5 the value for status is NULL instead of 10. Get supplier numbers for suppliers greater than 25. (Hint: Retrieval using NULL).
- k) Get unique supplier numbers supplying parts. (Hint: This query is using the built-in function count).
- l) For each part supplied, get the part number and the total quantity supplied for that part. (Hint: The query using GROUP BY).
- m) Get part numbers for all parts supplied by more than one supplier. (Hint: It is GROUP BY with HAVING).
- n) For all those parts, for which the total quantity supplied is greater than 300, get the part number and the maximum quantity of the part supplied in a single supply. Order the result by descending part number within those maximum quantity values.
- o) Double the status of all suppliers in Delhi. (Hint: UPDATE Operation).
- p) Let us consider the table TEMP has one column, called PNO. Enter into TEMP part numbers for all parts supplied by S2.
- q) Add part p7.
- r) Delete all the suppliers in Mumbai and also the suppliers concerned.

1.5 DATA CONTROL

The data control basically refers to commands that allow system and data privileges to be passed to various users. These commands are normally available to database administrator. Let us look into some data control language commands:

Create a new user:

```
CREATE USER < user name > IDENTIFIED BY < Password>
```

Example 26:

```
CREATE USER MCA12 IDENTIFIED BY W123
```

Grant: It is used to provide database access permission to users. It is of two types (1) system level permission (2) Object level permission.

Example 27:

```
GRANT CREATE SESSION TO MCA12;
```

(This command provides system level permission on creating a session – not portable)

```
GRANT SELECT ON EMP TO MCA12;
```

(Object level permission on table EMP)

```
GRANT SELECT, INSERT, UPDATE, DELETE ON EMP TO MCA12;
```

```
GRANT SELECT, UPDATE ON EMP TO MCA12, MCA13;
```

(Two users)

```
GRANT ALL ON EMP TO PUBLIC;
```

(All permission to all users, do not use it. It is very dangerous for database)

Revoke: It is used to cancel the permission granted.

Example 28:

```
REVOKE ALL ON EMP FROM MCA12;
```

(All permissions will be cancelled)

You can also revoke only some of the permissions.

Drop: A user-id can be deleted by using drop command.

Example 29:

```
DROP USER MCA12;
```

Accessing information about permissions to all users

- (1) Object level permissions: With the help of data dictionary you can view the permissions to user. Let us take the table name from oracle. In oracle the name of the table containing these permissions is user_tab_privs.

```
DESCRIBE USER_TAB_PRIVS ;  
SELECT * FROM USER_TAB_PRIVS;
```

- (2) System level permissions: With the help of data dictionary you can see them. Let us take the table name as user_sys_privs (used in oracle).

```
DESCRIBE USER_SYS_PRIVS ;
```

SELECT * FROM USER_SYS_PRIVS ;

All these commands are very specific to a data base system and may be different on different DBMS.

1.6 DATABASE OBJECTS: VIEWS, SEQUENCES, INDEXES AND SYNONYMS

Some of the important concepts in a database system are the views and indexes. Views are a mechanism that can support data independence and security. Indexes help in better query responses. Let us discuss about them along with two more concepts sequences and synonyms in more details.

1.6.1 Views

A view is like a window through which data from tables can be viewed or changed. The table on which a view is based is called Base table. The view is stored as a SELECT statement in the data dictionary. When the user wants to retrieve data, using view. Then the following steps are followed.

- 1) View definition is retrieved from data dictionary table. For example, the view definitions in oracle are to be retrieved from table name USER-VIEWS.
- 2) Checks access privileges for the view base table.
- 3) Converts the view query into an equivalent operation on the underlying base table

Advantages:

- It restricts data access.
- It makes complex queries look easy.
- It allows data independence.
- It presents different views of the same data.

Type of views:

Simple views and Complex Views

Feature	Simple Views	Complex Views
Number of tables	One	One or more
Contain Functions	No	Yes
Contain groups of data	No	Yes
Data Manipulation	IS allowed	Not always

Let us look into some of the examples. To see all the details about existing views in Oracle:

SELECT* FROM USER_VIEWS;

Creating a view:

- A query can be embedded within the CREATE VIEW STATEMENT
- A query can contain complex select statements including join, groups and sub-queries
- A query that defines the view **cannot contain** an order by clause.
- DML operation (delete/modify/add) cannot be applied if the view contains any of the following:

<i>Delete (You can't delete if view contains following)</i>	<i>Modify (you cannot modify if view contains following)</i>	<i>Insert (you cannot insert if view contains following)</i>
<ul style="list-style-type: none"> • Group functions • A group by clause • A distinct keyword 	<ul style="list-style-type: none"> • Group functions • A group by clause • A distinct keyword • Columns defined by Expressions 	<ul style="list-style-type: none"> • Group functions • A group by clause • A distinct keyword • Columns defined by Expressions • There are Not Null Columns in the base tables that are not selected by view.

Example 30: Create a view named employee salary having minimum, maximum and average salary for each department.

```
CREATE VIEW EMPSAL (NAME, MINSAL, MAXSAL, AVGSAL) AS
SELECT D.DNAME, MIN(E.SAL),MAX(E.SAL),AVG(E.SAL)
FROM EMP E, DEPT D
WHERE E.DEPTNO=D.DEPTNO
GROUP BY D.DNAME;
```

To see the result of the command above you can give the following command:

```
SELECT * FROM EMPSAL;
You may get some sample output like:
NAME      MINSAL    MAXSA    AVGSAL
-----  -----  -----
ACCOUNTING   1300     5000  2916.6667
RESEARCH     800      3000   2175
SALES        950     2850  1566.6667
```

To see the structure of the view so created, you can give the following command:

```
DESCRIBE EMPSAL;
Name          Null? Type
-----  -----
NAME          VARCHAR2 (14)
MINSAL        NUMBER
MAXSAL        NUMBER
AVGSAL        NUMBER
```

Creating views with check option: This option restricts those updates of data values that cause records to go off the view. The following example explains this in more detail:

Example 31: To create a view for employees of Department = 30, such that user cannot change the department number from the view:

```
CREATE OR REPLACE VIEW EMPD30 AS
SELECT EMPNO EMPL_NUM, ENAME NAME, SAL SALARY
FROM EMP
WHERE DEPTNO=30
WITH CHECK OPTION;
```

Now the user cannot change the department number of any record of view EMPD30. If this change is allowed then the record in which the change has been made will go off the view as the view is only for department-30. This is restricted because of use of WITH CHECK OPTION clause

Creating views with Read only option: In the view definition this option is used to ensure that no DML operations can be performed on the view.

Creating views with Replace option: This option is used to change the definition of the view without dropping and recreating it or regranting object privileges previously granted on it.

1.6.2 Sequences

Sequences:

- automatically generate unique numbers
- are sharable
- are typically used to create a primary key value
- replace application code
- speed up the efficiency of accessing sequence Values when cached in memory.

Example 32: Create a sequence named SEQSS that starts at 105, has a step of 1 and can take maximum value as 2000.

```
CREATE SEQUENCE SEQSS
START WITH 105
INCREMENT BY 1
MAX VALUE 2000;
```

How the sequence so created is used? The following sequence of commands try to demonstrate the use of the sequence SEQSS.

Suppose a table person exists as:

```
SELECT * FROM PERSON;
Output:      CODE    NAME      ADDRESS
-----        -----
          104     RAMESH   MUMBAI
```

Now, if we give the command:

```
INSERT INTO PERSON
```

```
VALUES (SEQSS.NEXTVAL, &NAME, &ADDRESS)
```

On execution of statement above do the following input:

Enter value for name: 'Rakhi'

Enter value for address: 'New Delhi'

Now, give the following command to see the output:

```
SELECT * FROM PERSON;
CODE NAME      ADDRESS
-----        -----
          104 RAMESH   MUMBAI
          105 Rakhi   NEW DELHI
```

The descriptions of sequences such as minimum value, maximum value, step or increment are stored in the data dictionary. For example, in oracle it is stored in the table user_sequences. You can see the description of sequences by giving the SELECT command.

Gaps in sequence values can occur when:

- A rollback occurs that is when a statement changes are not accepted.
- The system crashes
- A sequence is used in another table.

Modify a sequence:

```
ALTER SEQUENCE SEQSS
INCREMENT 2
MAXVALUE 3000;
```

Removing a sequence:

```
DROP SEQUENCE SEQSS;
```

1.6.3 Indexes and Synonyms

Some of the basic properties of indexes are:

- An Index is a schema Object
- Indexes can be created explicitly or automatically
- Indexes are used to speed up the retrieval of rows
- Indexes are logically and physically independent of the table. It means they can be created or dropped at any time and have no effect on the base tables or other indexes.
- However, when a table is dropped corresponding indexes are also dropped.

Creation of Indexes

Automatically: When a primary key or Unique constraint is defined in a table definition then a unique index is created automatically.

Manually: User can create non-unique indexes on columns to speed up access time to rows.

Example 33: The following commands create index on employee name and employee name + department number respectively.

```
CREATE INDEX EMP_ENAME_IDX ON EMP (ENAME);
CREATE INDEX EMP_MULTI_IDX ON EMP (ENAME, DEPTNO);
```

Finding details about created indexes: The data dictionary contains the name of index, table name and column names. For example, in Oracle a user-indexes and user-ind-columns view contains the details about user created indexes.

Remove an index from the data dictionary:

```
DROP INDEX EMP_ENAME_IDX;
```

Indexes cannot be modified.

Synonyms

It permits short names or alternative names for objects.

Example 34:

```
CREATE SYNONYM D30
FOR EMPD30;
```

Now if we give command:

```
SELECT * FROM D30;
```

The output will be:

NAME	MINSAL	MAXSAL	AVGSAL
ACCOUNTING	1300	5000	2916.6667
RESEARCH	800	3000	2175
SALES	950	2850	1566.6667

Removing a Synonym:

```
DROP SYNONYM D30;
```

1.7 TABLE HANDLING

In RDBMS more than one table can be handled at a time by using join operation. Join operation is a relational operation that causes two tables with a common domain to be combined into a single table or view. SQL specifies a join implicitly by referring the matching of common columns over which tables are joined in a WHERE clause. Two tables may be joined when each contains a column that shares a common domain with the other. The result of join operation is a single table. Selected columns from all the tables are included. Each row returned contains data from rows in the different input tables where values for the common columns match. An important rule of table handling is that there should be one condition within the WHERE clause for each pair of tables being joined. Thus if two tables are to be combined, one condition would be necessary, but if three tables (X, Y, Z) are to be combined then two conditions would be necessary because there are two pairs of tables (X-Y and Y-Z) OR (X-Z and Y-Z), and so forth. There are several possible types of joins in relational database queries. Four types of join operations are described below:

- (1) **Equi Join:** A join in which the joining condition is based on equality between values in the common columns. Common columns appear (redundantly) in the result table. Consider the following relations:

- customer (custid, custname,)
- order (custid, ordered,.....)

Example 35: What are the names of all customers who have placed orders?

The required information is available in two tables, customer and order. The SQL solution requires joining the two table using equi join.

```
SELECT CUSTOMER.CUSTOID, ORDER.CUSTOID,
       CUSTOMNAME, ORDERID
  FROM CUSTOMER, ORDER
 WHERE CUSTOMER.CUSTOID=ORDER.CUSTOID;
```

The output may be:

Customer.custoid	order.custoid	customname	orderid
10	10	Pooja Enterprises	1001
12	12	Estern Enterprises	1002
3	3	Impressions	1003

- (2) **Natural Join:** It is the same like Equi join except one of the duplicate columns is eliminated in the result table. The natural join is the most commonly used form of join operation.

Example 36:

```
SELECT CUSTOMER.CUSTOID, CUSTOMNAME, ORDERID
  FROM CUSTOMER, ORDER
 WHERE CUSTOMER.CUSTOID=ORDER.CUSTOID;
```

Output:

custoid	customname	orderid
10	Pooja Enterprises	1001
12	Estern Enterprises	1002
3	Impressions	1003

- (3) **Outer Join:** The use of Outer Join is that it even joins those tuples that do not have matching values in common columns are also included in the result table. Outer join places null values in columns where there is not a match between

tables. A condition involving an outer join is that it cannot use the IN operator or cannot be linked to another condition by the OR operator.

Example 37: The following is an example of left outer join (which only considers the non-matching tuples of table on the left side of the join expression).

```
SELECT CUSTOMER.CUSTOID, CUSTOMERNAME, ORDERID
FROM CUSTOMER LEFT OUTER JOIN ORDER
WHERE CUSTOMER.CUSTOID = ORDER.CUSTOID;
```

Output: The following result assumes a CUSTID in CUSTOMER table who have not issued any order so far.

CUSTOID	CUSTOMERNAME	ORDERID
10	Pooja Enterprises	1001
12	Estern Enterprises	1002
3	Impressions	1003
15	South Enterprises	NULL

The other types of outer join are the Right outer join or complete outer join.

- (4) **Self-Join:** It is a join operation where a table is joined with itself. Consider the following sample partial data of EMP table:

EMPNO	ENAME	MGRID
1	Nirmal	4	
2	Kailash	4	
3	Veena	1	
4	Boss	NULL	
.....	

Example 38: Find the name of each employee's manager name.

```
SELECT WORKER.ENAME || 'WORK FOR' || MANAGER.ENAME
FROM EMP WORKER, EMP MANAGER
WHERE WORKER.MGR=MANAGER.EMPNO;
```

Output:

Nirmal works for Boss
Kailash works for Boss
Veena works for Nirmal

☛ Check Your Progress 2

- 1) Discuss how the Access Control mechanism of SQL works.
-
-
-

- 2) Consider Hotel schema consisting of three tables Hotel, Booking and Guest,
CREATE TABLE Hotel

```
hotelNo      HotelNumber      NOT NULL,
hotelName    VARCHAR(20)      NOT NULL,
city         VARCHAR(50)      NOT NULL,
```

PRIMARY KEY (hotelNo));

```
CREATE TABLE Booking(
    hotelNo      HotelNumbers      NOT NULL,
    guestNo      GuestNumbers      NOT NULL,
    dateFrom     BookingDate       NOT NULL,
    dateTo       BookingDate       NULL,
    roomNo       RoomNumber        NOT NULL,
    PRIMARY KEY (hotelNo, guestNo, dateFrom),
    FOREIGN KEY (hotelNo) REFERENCES Hotel
        ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (guestNo) REFERENCES Guest
        ON DELETE NO ACTION ON UPDATE CASCADE,
    FOREIGN KEY (hotelNo, roomNo) REFERENCES Room
        ON DELETE NO ACTION ON UPDATE CASCADE);
```

```
CREATE TABLE Guest(
    guestNo      GuestNumber        NOT NULL,
    guestName    VARCHAR(20)        NOT NULL,
    guestAddress VARCHAR(50)        NOT NULL
    PRIMARY KEY (guestno));
```

```
CREATE TABLE Room(
    roomNo      RoomNumber        NOT NULL,
    hotelNo      HotelNumbers      NOT NULL,
    type         RoomType          NOT NULL DEFAULT 'S',
    price        RoomPrice         NOT NULL,
    PRIMARY KEY (roomNo, hotelNo),
    FOREIGN KEY (hotelNo) REFERENCES Hotel
        ON DELETE CASCADE ON UPDATE CASCADE);
```

Create a view containing the hotel name and the names of the guests staying at the hotel.

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

- 3) Give the users Manager and Director full access to views HotelData and BookingOutToday, with the privilege to pass the access on to other users.
-
.....
.....
.....

1.8 NESTED QUERIES

By now we have discussed the basic commands including data definition and data manipulations. Now let us look into some more complex queries in this section.

Sub-queries

Some of the basic issues of sub-queries are:

- A sub-query is a SELECT statement that is embedded in a clause of another SELECT statement. They are often referred to as a NESTED SELECT or SUB SELECT or INNER SELECT.
- The sub-query (inner query) executes first before the main query. The result of the sub-query is used by the main query (outer query).
- Sub-query can be placed in WHERE or HAVING or FROM clauses.
- Format of using sub-queries:

```
SELECT<select_list>
FROM<table>
WHERE expr OPERATOR
      (SELECT <select_list>
       FROM <TABLE>WHERE);
```

Operator includes a comparison operator (single or multiple row operators)

Single row operators: $>$, $=$, $>=$, $<$, $<=$, \neq

Multiple row operators: IN, ANY, ALL

- Order by clause cannot be used in sub-query, if specified it must be the last clause in the main select statement.
- Types of sub-queries:
 - Single row sub-query: It returns only one row from the inner select statement.
 - Multiple row sub-queries: it returns more than one row from the inner select statement
 - Multiple column sub-queries: it returns more than one column from the inner select statement.

Single row operators are used with single row sub queries and multiple row operators are used with multiple row sub queries.

- The Outer and Inner queries can get data from different tables.
- Group Functions can be used in sub queries.

Consider the following partial relation EMP. Let us create some sub-queries for them

EMPNO	ENAME	JOB	SAL	DEPTNO
7566	Nirmal	MANAGER	2975	10
7788	Kailash	ANALYST	3000	10
7839	Karuna	PRESIDENT	5000	20
7902	Ashwin	ANALYST	3000	20
7905	Ashwini	MANAGER	4000	20

Example 39: Get the details of the person having the minimum salary.

```
SELECT ENAME, JOB, SAL
FROM EMP
WHERE SAL = (    SELECT MIN (SAL)
                  FROM EMP);
```

Output:

ENAME	JOB	SAL
Nirmal	MANAGER	2975

Example 40: Display the employees whose job title is the same as that of employee 7566 and salary is more than the salary of employee 7788.

```
SELECT ENAME, JOB
FROM EMP
WHERE JOB = (    SELECT JOB
                  FROM EMP
                  WHERE EMPPNO = 7566)
AND SAL > ( SELECT SAL
              FROM EMP
              WHERE EMPPNO=7788);
```

Output: Job title for the employee 7566 happens to be ‘MANAGER’)

ENAME	JOB
Ashwini	MANAGER

Having Clause with sub queries: First we recollect the GROUP BY clause. The following query finds the minimum salary in each department.

```
SELECT DEPTNO, MIN(SAL)
FROM EMP
GROUP BY DEPTNO;
```

Output:

DEPTNO	SAL
10	2975
20	3000

Example 41: To find the minimum salary in those departments whose minimum salary is greater than minimum salary of department number 10.

```
SELECT DEPTNO, MIN(SAL)
FROM EMP
GROUP BY DEPTNO
HAVING MIN(SAL) > (
```

```
SELECT MIN (SAL)
FROM EMP
WHERE DEPTNO = 10);
```

Output:

DEPTNO	SAL
20	3000

Example 42: Find the name, department number and salary of employees drawing minimum salary in that department.

```
SELECT ENAME, SAL, DEPTNO
FROM EMP
WHERE SAL IN (SELECT MIN (SAL)
                  FROM EMP
                  GROUP BY DEPTNO);
```

Output:

ENAME	SAL	DEPTNO
Nirmal	2975	10
Ashwin	3000	20

Find the salary of employees employed as an ANALYST
SELECT SAL FROM EMP WHERE JOB = 'ANALYST'

Output:

SAL
3000
3000

Example 43: Find the salary of employees who are not 'ANALYST' but get a salary less than or equal to any person employed as 'ANALYST'.

```
SELECT EMPNO, ENAME, JOB, SAL
FROM EMP
WHERE SAL <= ANY ( SELECT SAL
                   FROM EMP WHERE JOB = 'ANALYST' )
AND JOB <> 'ANALYST';
```

Output:

EMPNO	ENAME	JOB	SAL
7566	Nirmal	MANAGER	2975

Find the average salary in each department

SELECT DEPTNO, AVG(SAL) FROM EMP GROUP BY DEPTNO;
Result:

DEPTNO	SAL
10	2987.5
20	4000

Example 44: Find out the employee who draws a salary more than the average salary of all the departments.

```
SELECT EMPNO, ENAME, JOB, SAL
FROM EMP
WHERE SAL > ALL (SELECT AVG (SAL)
                  FROM EMP
                  GROUP BY DEPTNO);
```

Output:

EMPNO	ENAME	JOB	SAL
7839	Karuna	PRESIDENT	5000

Example 45: Find the employee name, salary, department number and average salary of his/her department, for those employees whose salary is more than the average salary of that department.

```
SELECT A.ENAME, A.SAL, A.DEPTNO, B.AVGSAL
FROM EMP A, ( SELECT DEPTNO, AVG (SAL) AVGSAL
               FROM EMP
               GROUP BY DEPTNO ) B
WHERE A.DEPTNO=B.DEPTNO AND A.SAL > B.AVGSAL;
```

Output:

ENAME	SAL	DEPTNO	AVGSAL
Kailash	3000	10	2987.5
Karuna	5000	20	4000

Multiple column Queries:

Syntax:

```
SELECT COLUMN1, COL2,....  
FROM TABLE  
WHERE (COLUMN1, COL2, ...) IN  
(SELECT COLUMN1, COL2,....  
FROM TABLE  
WHERE <CONDITION>);
```

Example 46: Find the department number, name, job title and salary of those people who have the same job title and salary as those are in department 10.

```
SELECT DEPTNO,ENAME,JOB,SAL  
FROM EMP  
WHERE (JOB, SAL) IN (      SELECT JOB, SAL  
                        FROM EMP  
                       WHERE DEPTNO=10);
```

Output:

DEPTNO	ENAME	JOB	SAL
10	Nirmal	MANAGER	2975
10	Kailash	ANALYST	3000
20	Ashwin	ANALYST	3000

Check Your Progress 3

- 1) What is the difference between a sub-query and a join? Under what circumstances would you not be able to use a sub-query?

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

- 2) Use the Hotel schema defined in question number 2 (Check Your Progress 2) and answer the following queries:

- List the names and addresses of all guests in Delhi, alphabetically ordered by name.
- List the price and type of all rooms at the GRAND Hotel.
- List the rooms that are currently unoccupied at the Grosvenor Hotel.
- List the number of rooms in each hotel.
- What is the most commonly booked room type for hotel in Delhi?
- Update the price of all rooms by 5%.

.....
.....

- 3) 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.

- (i) SELECT ename
 FROM employees
 WHERE eno IN (SELECT eno
 FROM workin
 GROUP BY eno
 HAVING COUNT (*) = (SELECT COUNT (*) FROM projects));
- (ii) SELECT Pname
 FROM projects
 WHERE Pno IN (SELECT Pno
 FROM projects
 MINUS
 GROUP BY eno
 (SELECT DISTINCT Pno FROM workin));
-
-
-
-

1.9 SUMMARY

This unit has introduced the SQL language for relational database definition, manipulation and control. The SQL environment includes an instance of an SQL DBMS with accessible databases and associated users and programmers. Each schema definition that describes the database objects is stored in data dictionary/ system catalog. Information contained in system catalog is maintained by the DBMS itself, rather than by the users of DBMS.

The data definition language commands are used to define a database, including its creation and the creation of its tables, indexes and views. Referential integrity constraints are also maintained through DDL commands. The DML commands of SQL are used to load, update and query the database. DCL commands are used to establish user access to the database. SQL commands directly affect the base tables, which contain the raw data, or they may affect database view, which has been created. The basic syntax of an SQL SELECT statement contains the following keywords: SELECT, FROM, WHERE, ORDER BY, GROUP BY and HAVING.

SELECT determines which attributes will be displayed in the query result table. FROM determines which tables or views will be used in the query. WHERE sets the criteria of the query, including any joins of multiple tables, which are necessary. ORDER BY determines the order in which the result will be displayed. GROUP BY is used to categorize results. HAVING is used to impose condition with GROUP BY.

1.10 SOLUTIONS / ANSWERS

Check Your Progress 1

1)

Advantages

- A standard for database query languages
- (Relatively) Easy to learn
- Portability
- SQL standard exists
- Both interactive and embedded access
- Can be used by specialist and non-specialist.

Yes, SQL has disadvantages. However, they are primarily more technical with reference to Language features and relational model theories. We are just putting them here for reference purposes.

Disadvantages

- Impedance mismatch – mixing programming paradigms with embedded access
- Lack of orthogonality – many different ways to express some queries
- Language is becoming enormous (SQL-92 is 6 times larger than predecessor)
- Handling of nulls in aggregate functions is not portable
- Result tables are not strictly relational – can contain duplicate tuples, imposes an ordering on both columns and rows.

2. CREATE DOMAIN RoomType AS CHAR(1)*[Constraint (a)]*
CHECK(VALUE IN (S, F, D));

CREATE DOMAIN HotelNumbers AS HotelNumber

CHECK(VALUE IN (SELECT hotelNo FROM Hotel));

*[An additional constraint for
hotel number for the application]*

CREATE DOMAIN RoomPrice AS DECIMAL(5, 2)
CHECK(VALUE BETWEEN 1000 AND 10000);

CREATE DOMAIN RoomNumber AS VARCHAR(4)
CHECK(VALUE BETWEEN '1' AND '100');

*[Constraint (c), one additional character is kept instead of 3
we have used 4characters but no space wastage as varchar]*

CREATE TABLE Room(

roomNo	RoomNumber	NOT NULL,
hotelNo	HotelNumbers	NOT NULL,
type	RoomType	NOT NULL DEFAULT S,
price	RoomPrice	NOT NULL,

PRIMARY KEY (roomNo, hotelNo),

FOREIGN KEY (hotelNo) REFERENCES Hotel

ON DELETE CASCADE ON UPDATE CASCADE);

CREATE DOMAIN GuestNumber AS CHAR(4);

```
CREATE TABLE Guest(
    guestNo      GuestNumber      NOT NULL,
    guestName    VARCHAR(20)      NOT NULL,
    guestAddress VARCHAR(50)      NOT NULL);
```

```
CREATE DOMAIN GuestNumbers AS GuestNumber
    CHECK(VALUE IN (SELECT guestNo FROM Guest));
    [A sort of referential constraint expressed within domain]
```

```
CREATE DOMAIN BookingDate AS DATETIME
    CHECK(VALUE > CURRENT_DATE);           [constraint (d)]
```

```
CREATE TABLE Booking(
    hotelNo      HotelNumbers      NOT NULL,
    guestNo      GuestNumbers      NOT NULL,
    dateFrom    BookingDate      NOT NULL,
    dateTo      BookingDate      NULL,
    roomNo      RoomNumber      NOT NULL,
    PRIMARY KEY (hotelNo, guestNo, dateFrom),
    FOREIGN KEY (hotelNo) REFERENCES Hotel
        ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (guestNo) REFERENCES Guest
        ON DELETE NO ACTION ON UPDATE CASCADE,
    FOREIGN KEY (hotelNo, roomNo) REFERENCES Room
        ON DELETE NO ACTION ON UPDATE CASCADE,
    CONSTRAINT RoomBooked
        CHECK (NOT EXISTS ( SELECT *
            FROM Booking b
            WHERE b.dateTo > Booking.dateFrom AND
            b.dateFrom < Booking.dateTo AND
            b.roomNo = Booking.roomNo AND
            b.hotelNo = Booking.hotelNo)),
    CONSTRAINT GuestBooked
        CHECK (NOT EXISTS ( SELECT *
            FROM Booking b
            WHERE b.dateTo > Booking.dateFrom AND
            b.dateFrom < Booking.dateTo AND
            b.guestNo = Booking.guestNo)));
```

- | | | |
|----|-----------------|--|
| 3. | FROM | Specifies the table or tables to be used. |
| | WHERE | Filters the rows subject to some condition. |
| | GROUP BY | Forms groups of rows with the same column value. |
| | HAVING | Filters the groups subject to some condition. |
| | SELECT | Specifies which columns are to appear in the output. |
| | ORDER BY | Specifies the order of the output. |

If the SELECT list includes an aggregate function and no GROUP BY clause is being used to group data together, then no item in the SELECT list can include any reference to a column unless that column is the argument to an aggregate function.

When GROUP BY is used, each item in the SELECT list must be single-valued per group. Further, the SELECT clause may only contain:

- Column names.
- Aggregate functions.
- Constants.
- An expression involving combinations of the above.

All column names in the SELECT list must appear in the GROUP BY clause unless the name is used only in an aggregate function.

3. Please note that some of the queries are sub-queries and queries requiring join. The meaning of these queries will be clearer as you proceed further with the Unit.

a) `SELECT SNO
FROM S
WHERE CITY = 'Delhi'
AND STATUS > 20;`

Result:

SNO
S4

b) `SELECT SNO, STATUS
FROM S
WHERE CITY = 'Delhi'
ORDER BY STATUS DESC;`

Result:

SNO	STATUS
S4	40
S5	10

c) `SELECT FIRST.SNO, SECOND.SNO
FROM S FIRST, S SECOND
WHERE FIRST.CITY = SECOND.CITY AND FIRST.SNO <
SECOND.SNO;`

Please note that if you do not give the condition after AND you will get some unnecessary tuples such as: (S4, S4), (S5, S4) and (S5, S5).

Result:

SNO	SNO
S4	S5

d) `SELECT DISTINCT SNAME
FROM S, SP
WHERE S.SNO = SP.SNO
AND SP.PNO = 'P2';`

Result:

SNAME
Prentice Hall

McGraw Hill
Wiley
Pearson

OR
 SELECT SNAME
 FROM S
 WHERE SNO = ANY (SELECT SNO
 FROM SP
 WHERE PNO = 'P2');

- e) SELECT SNAME
 FROM S
 WHERE SNO IN (SELECT SNO
 FROM SP
 WHERE PNO = 'P2');
- f) SELECT DISTINCT PNO
 FROM SP SPX
 WHERE PNO IN (SELECT PNO
 FROM SP
 WHERE SP.SNO = SPX.SNO AND SPX.SNO < SP.SNO);

This query can also be answered using count and group by. Please formulate that.

Result:

PNO
P1
P2

- g) SELECT SNO
 FROM S
 WHERE CITY = (SELECT CITY
 FROM S
 WHERE SNO = 'S1');

Result:

SNO
S1

- h) SELECT SNAME
 FROM S
 WHERE EXISTS (SELECT *
 FROM SP
 WHERE SNO = S.SNO AND PNO = 'P1');

Result:

SNAME
Prentice Hall
McGraw Hill

- i) SELECT PNO
 FROM SP
 WHERE QUANTITY > 200 UNION (SELECT PNO
 FROM SP
 WHERE SNO = S2);

Result:

PNO
P1
P2

j) SELECT SNO
FROM S
WHERE STATUS > 25 OR STATUS IS NULL;

Result:

SNO
S1
S2
S4
S5

k) SELECT COUNT (DISTINCT SNO)
FROM SP;

Result: 4

l) SELECT PNO, SUM(QUANTITY)
FROM SP
GROUP BY PNO;

Result:

PNO	SUM
P1	400
P2	1000

m) SELECT PNO
FROM SP
GROUP BY PNO
HAVING COUNT(*) > 1 ;

The query is a same as that of part (f)

n) SELECT PNO, MAX(QUANTITY)
FROM SP
WHERE QUANTITY > 200
GROUP BY PNO
HAVING SUM(QUANTITY) > 300
ORDER BY 2, PNO DESC;

o) UPDATE S
SET STATUS = 2 * STATUS
WHERE CITY = 'Delhi' ;

p) INSERT INTO TEMP
SELECT PNO
FROM SP
WHERE SNO = 'S2' ;

q) INSERT INTO SP(SNO,PNO,QUANTITY) < 'S5' , 'P7' , 100 > ;

Please note that part cannot be added without a supply in the present case.

Actually there should be another table for Parts

r) DELETE S, SP
WHERE SNO = (SELECT SNO
FROM S
WHERE CITY = 'Mumbai');

Check Your Progress 2

- 1) Each user has an authorization identifier (allocated by DBA).
Each object has an owner. Initially, only owner has access to an object but the owner can pass privileges to carry out certain actions on to other users via the GRANT statement and take away given privileges using REVOKE.
- 2) CREATE VIEW HotelData(hotelName, guestName) AS
SELECT h.hotelName, g.guestName
FROM Hotel h, Guest g, Booking b
WHERE h.hotelNo = b.hotelNo AND g.guestNo = b.guestNo AND
b.dateFrom <= CURRENT_DATE AND
b.dateTo >= CURRENT_DATE;
- 3) GRANT ALL PRIVILEGES ON HotelData
TO Manager, Director WITH GRANT OPTION;
GRANT ALL PRIVILEGES ON BookingOutToday
TO Manager, Director WITH GRANT OPTION;

Check Your Progress 3

- 1) With a sub-query, the columns specified in the SELECT list are restricted to one table. Thus, cannot use a sub-query if the SELECT list contains columns from more than one table. But with a join operation SELECT list contains columns from more than two tables.
- 2) Answers of the queries are:

- SELECT guestName, guestAddress FROM Guest
WHERE address LIKE '%Delhi%'
ORDER BY guestName;
- SELECT price, type FROM Room
WHERE hotelNo =
(SELECT hotelNo FROM Hotel
WHERE hotelName = 'GRAND Hotel');
- SELECT * FROM Room r
WHERE roomNo NOT IN
(SELECT roomNo FROM Booking b, Hotel h
WHERE (dateFrom <= CURRENT_DATE AND
dateTo >= CURRENT_DATE) AND
b.hotelNo = h.hotelNo AND hotelName = 'GRAND Hotel');
- SELECT hotelNo, COUNT(roomNo) AS count
FROM Room
GROUP BY hotelNo;
- SELECT MAX(X)

FROM (SELECT type, COUNT(type) AS X

FROM Booking b, Hotel h, Room r

WHERE r.roomNo = b.roomNo AND b.hotelNo = h.hotelNo AND
city = 'LONDON'

GROUP BY type);

- UPDATE Room SET price = price*1.05;

3) (i) – Give names of employees who are working on all projects.

(ii) - Give names of the projects which are currently not being worked upon.

1.11 FURTHER READINGS

Fundamentals of Database Systems; Almasri and Navathe; Pearson Education Limited; Fourth Edition; 2004.

A Practical Approach to Design, Implementation, and Management; Thomas Connolly and Carolyn Begg; Database Systems, Pearson Education Limited; Third Edition; 2004.

The Complete Reference; Kevin Lonely and George Koch; Oracle 9i, Tata McGraw-Hill; Fourth Edition; 2003.

Jeffrey A. Hoffer, Marry B. Prescott and Fred R. McFadden; Modern Database Management; Pearson Education Limited; Sixth Edition; 2004.

For Unit 9 : Please read the following unit of MCS-43 Block 1 Unit 3

UNIT 3 ADVANCED SQL

Structure	THE PEOPLE'S UNIVERSITY	Page Nos.
3.0 Introduction		47
3.1 Objectives		47
3.2 Assertions and Views		47
3.2.1 Assertions		
3.2.2 Views		
3.3 Embedded SQL and Dynamic SQL		51
3.3.1 Embedded SQL		
3.3.2 Cursors and Embedded SQL		
3.3.3 Dynamic SQL		
3.3.4 SQLJ		
3.4 Stored Procedures and Triggers		58
3.4.1 Stored Procedures		
3.4.2 Triggers		
3.5 Advanced Features of SQL		61
3.6 Summary		62
3.7 Solutions/Answers		62

3.0 INTRODUCTION

The Structured Query Language (SQL) is a standard query language for database systems. It is considered as one of the factors contributing to the success of commercial database management systems, primarily, because of the availability of this standard language on most commercial database systems. We have described SQL in details in the course MCS-023 Block-2, Unit-1 where we discussed data definition, data manipulation, data control, queries, joins, group commands, sub-queries, etc.

In this unit, we provide details of some of the advanced features of Structured Query Language. We will discuss Assertions and Views, Triggers, Standard Procedure and Cursors. The concepts of embedded and dynamic SQL and SQLJ, which is used along with JAVA, are also been introduced. Some of the advanced features of SQL have been covered. We will provide examples in various sections rather than including a separate section of examples. The examples given here are in a SQL3 standard and will be applicable for any commercial database management system that supports SQL3 standards.

3.1 OBJECTIVES

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

- define Assertions and explain how they can be used in SQL;
- explain the concept of views, SQL commands on views and updates on views;
- define and use Cursors;
- discuss Triggers and write stored procedures, and
- explain Dynamic SQL and SQLJ.

3.2 ASSERTIONS AND VIEWS

One of the major requirements in a database system is to define constraints on various tables. Some of these simple constraints can be specified as primary key, NOT NULL, check value and range constraints. Such constraints can be specified within



the data or table creation statements with the help of statements like NOT NULL, PRIMARY KEY, UNIQUE, CHECK etc. Referential constraints can be specified with the help of foreign key constraints. However, there are some constraints, which may relate to more than one field or table. These are called assertions.

In this section we will discuss about two important concepts that we use in database systems viz., views and assertions. Assertions are general constraints while views are virtual tables. Let us discuss about them in more detail.

3.2.1 Assertions

Assertions are constraints that are normally of general nature. For example, the age of the student in a hypothetical University should not be more than 25 years or the minimum age of the teacher of that University should be 30 years. Such general constraints can be implemented with the help of an assertion statement. The syntax for creating assertion is:

Syntax:

```
CREATE ASSERTION <Name>
CHECK (<Condition>);
```

Thus, the assertion on age for the University as above can be implemented as:

```
CREATE ASSERTION age-constraint
CHECK (NOT EXISTS (
    SELECT *
    FROM STUDENT s
    WHERE s.age > 25
    OR s.age > (
        SELECT MIN (f.age)
        FROM FACULTY f
    )));

```

The assertion name helps in identifying the constraints specified by the assertion. These names can be used to modify or delete an assertion later. But how are these assertions enforced? The database management system is responsible for enforcing the assertion on to the database such that the constraints stated in the assertion are not violated. Assertions are checked whenever a related relation changes.

Now try writing an assertion for a university system that stores a database of faculty as:

FACULTY (code, name, age, basic salary, medical-allow, other benefits)

MEDICAL-CLAIM (code, date, amount, comment)

Assertion: The total medical claims made by a faculty member in the current financial year should not exceed his/her medial allowance.

```
CREATE ASSERTION med-claim
CHECK (NOT EXISTS (
    SELECT code, SUM (amount), MIN(medical-allow)
    FROM (FACULTY NATRUAL JOIN MEDICAL-CLAIM)
    WHERE date>"31-03-2006"
    GROUP BY code
    HAVING MIN(medical-allow) < SUM(amount)
));
```

OR

```
CREATE ASSERTION med-claim
CHECK (NOT EXISTS (
    SELECT *
    FROM FACULTY f
    WHERE (f.code IN
        (SELECT code, SUM(amount)
        FROM MEDICAL-CLAIM m
        WHERE date>"31-03-2006" AND f.code=m.code AND
        f.medical-allow<SUM(amount)))
```

Please analyse both the queries above and find the errors if any.

So, now you can create an assertion. But how can these assertions be used in database systems? The general constraints may be designed as assertions, which can be put into the stored procedures. Thus, any violation of an assertion may be detected.

3.2.2 Views

A view is a virtual table, which does not actually store data. But if it does not store any data, then what does it contain?

A view actually is a query and thus has a `SELECT FROM WHERE` clause which works on physical table which stores the data. Thus, the view is a collection of relevant information for a specific entity. The ‘view’ has been introduced as a topic in MCS-023, Block 2, Unit-1. Let us recapitulate the SQL commands for creating views, with the help of an example.

Example: A student’s database may have the following tables:

```
STUDENT (name, enrolment-no, dateofbirth)
MARKS (enrolment-no, subjectcode, smarks)
```

For the database above a view can be created for a Teacher who is allowed to view only the performance of the student in his/her subject, let us say MCS-043.

```
CREATE VIEW SUBJECT-PERFORMANCE AS
```

```
(SELECT s.enrolment-no, name, subjectcode, smarks
FROM STUDENT s, MARKS m
WHERE s.enrolment-no = m.enrolment-no AND
subjectcode 'MCS-043' ORDER BY s.enrolment-no;
```

A view can be dropped using a `DROP` statement as:

```
DROP VIEW SUBJECT-PERFORMANCE;
```

The table, which stores the data on which the statement of the view is written, is sometimes referred to as the base table. You can create views on two or more base tables by combining the data using joins. Thus, a view hides the logic of joining the tables from a user. You can also index the views too. This may speed up the performance. Indexed views may be beneficial for very large tables. Once a view has been created, it can be queried exactly like a base table. For example:

```
SELECT *
FROM STUDENT-PERFORMANCE
WHERE smarks >50
```

How the views are implemented?



There are two strategies for implementing the views. These are:

- Query modification
- View materialisation.

In the query modification strategy, any query that is made on the view is modified to include the view defining expression. For example, consider the view STUDENT-PERFORMANCE. A query on this view may be: The teacher of the course MCS-043 wants to find the maximum and average marks in the course. The query for this in SQL will be:

```
SELECT MAX(smarks), AVG(smarks)  
FROM SUBJECT-PERFORMANCE
```

Since SUBJECT-PERFORMANCE is itself a view the query will be modified automatically as:

```
SELECT MAX (smarks), AVG (smarks)  
FROM STUDENT s, MARKS m  
WHERE s.enrolment-no=m.enrolment-no AND subjectcode= "MCS-043";
```

However, this approach has a major disadvantage. For a large database system, if complex queries have to be repeatedly executed on a view, the query modification will have to be done each time, leading to inefficient utilisation of resources such as time and space.

The view materialisation strategy solves this problem by creating a temporary physical table for a view, thus, materialising it. However, this strategy is not useful in situations where many database updates are made on the tables, which are used for view creation, as it will require suitable updating of a temporary table each time the base table is updated.

Can views be used for Data Manipulations?

Views can be used during DML operations like INSERT, DELETE and UPDATE. When you perform DML operations, such modifications need to be passed to the underlying base table. However, this is not allowed on all the views. Conditions for the view that may allow Data Manipulation are:

A view allows data updating, if it follows the following conditions:

- 1) If the view is created from a single table, then:
 - For INSERT operation, the PRIMARY KEY column(s) and all the NOT NULL columns must be included in the view.
 - View should not be defined using any aggregate function or GROUP BY or HAVING or DISTINCT clauses. This is due to the fact that any update in such aggregated attributes or groups cannot be traced back to a single tuple of the base table. For example, consider a view avgmarks (coursecode, avgmark) created on a base table student(st_id, coursecode, marks). In the avgmarks table changing the class average marks for coursecode "MCS 043" to 50 from a calculated value of 40, cannot be accounted for a single tuple in the Student base table, as the average marks are computed from the marks of all the Student tuples for that coursecode. Thus, this update will be rejected.
- 2) The views in SQL that are defined using joins are normally NOT updatable in general.
- 3) WITH CHECK OPTION clause of SQL checks the updatability of data from views, therefore, must be used with views through which you want to update.



Views and Security

Views are useful for security of data. A view allows a user to use the data that is available through the view; thus, the hidden data is not made accessible. Access privileges can be given on views. Let us explain this with the help of an example. Consider the view that we have created for teacher-STUDENT-PERFORMANCE. We can grant privileges to the teacher whose name is 'ABC' as:

```
GRANT SELECT, INSERT, DELETE ON STUDENT-PERFORMANCE TO ABC  
WITH GRANT OPTION;
```

Please note that the teacher ABC has been given the rights to query, insert and delete the records on the given view. Please also note s/he is authorised to grant these access rights (WITH GRANT OPTION) to any data entry user so that s/he may enter data on his/her behalf. The access rights can be revoked using the REVOKE statement as:

```
REVOKE ALL ON STUDENT-PERFORMANCE FROM ABC;
```

☞ Check Your Progress 1

- 1) Consider a constraint – the value of the age field of the student of a formal University should be between 17 years and 50 years. Would you like to write an assertion for this statement?

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

- 2) Create a view for finding the average marks of the students in various subjects, for the tables given in section 3.2.2.

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

- 3) Can the view created in problem 2 be used to update subjectcode?

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

3.3 EMBEDDED SQL AND DYNAMIC SQL

SQL commands can be entered through a standard SQL command level user interface. Such interfaces are interactive in nature and the result of a command is shown immediately. Such interfaces are very useful for those who have some knowledge of SQL and want to create a new type of query. However, in a database application where a naïve user wants to make standard queries, that too using GUI type interfaces, probably an application program needs to be developed. Such interfaces sometimes require the support of a programming language environment.

Please note that SQL normally does not support a full programming paradigm (although the latest SQL has full API support), which allows it a full programming interface. In fact, most of the application programs are seen through a programming interface, where SQL commands are put wherever database interactions are needed. Thus, SQL is embedded into programming languages like C, C++, JAVA, etc.

Let us discuss the different forms of embedded SQL in more detail.



3.3.1 Embedded SQL

The embedded SQL statements can be put in the application program written in C, Java or any other host language. These statements sometime may be called static. Why are they called static? The term ‘static’ is used to indicate that the embedded SQL commands, which are written in the host program, do not change automatically during the lifetime of the program. Thus, such queries are determined at the time of database application design. For example, a *query statement* embedded in C to determine the status of train booking for a train will not change. However, this query may be executed for many different trains. Please note that it will only change the input parameter to the query that is train-number, date of boarding, etc., and not the query itself.

But how is such embedding done? Let us explain this with the help of an example.

Example: Write a C program segment that prints the details of a student whose enrolment number is input.

Let us assume the relation

```
STUDENT (enrolno:char(9), name:Char(25), phone:integer(12), prog-code:char(3))
/* add proper include statements*/
/*declaration in C program */
EXEC SQL BEGIN DECLARE SECTION;
    Char enrolno[10], name[26], p-code[4];
    int phone;
    int SQLCODE;
    char SQLSTATE[6]
EXEC SQL END DECLARE SECTION;
/* The connection needs to be established with SQL*/
/* program segment for the required function */
printf ("enter the enrolment number of the student");
scanf ("%s", &enrolno);
EXEC SQL
    SELECT name, phone, prog-code INTO
        :name, :phone, :p-code
    FROM STUDENT
    WHERE enrolno = :enrolno;
If (SQLCODE ==0)
    printf ("%d, %s, %s, %s", enrolno, name, phone, p-code)
else
    printf ("Wrong Enrolment Number");
```

Please note the following points in the program above:

- The program is written in the host language ‘C’ and contains embedded SQL statements.
- Although in the program an SQL query (SELECT) has been added. You can embed any DML, DDL or views statements.
- The distinction between an SQL statement and host language statement is made by using the key word EXEC SQL; thus, this key word helps in identifying the Embedded SQL statements by the pre-compiler.
- Please note that the statements including (EXEC SQL) are terminated by a semi-colon (;),
- As the data is to be exchanged between a host language and a database, there is a need of shared variables that are shared between the environments. Please note that enrolno[10], name[20], p-code[4]; etc. are shared variables, colon (:) declared in ‘C’.

- Please note that the shared host variables enrolno is declared to have char[10] whereas, an SQL attribute enrolno has only char[9]. Why? Because in ‘C’ conversion to a string includes a ‘\ 0’ as the end of the string.
- The type mapping between ‘C’ and SQL types is defined in the following table:

‘C’ TYPE	SQL TYPE
long	INTEGER
short	SMALLINT
float	REAL
double	DOUBLE
char [i+1]	CHAR (i)

- Please also note that these shared variables are used in SQL statements of the program. They are prefixed with the colon (:) to distinguish them from database attribute and relation names. However, they are used without this prefix in any C language statement.
- Please also note that these shared variables have almost the same name (except p-code) as that of the attribute name of the database. The prefix colon (:) this distinguishes whether we are referring to the shared host variable or an SQL attribute. Such similar names is a good programming convention as it helps in identifying the related attribute easily.
- Please note that the shared variables are declared between BEGIN DECLARE SECTION and END DECLARE SECTION and their types are defined in ‘C’ language.

Two more shared variables have been declared in ‘C’. These are:

- SQLCODE as int
- SQLSTATE as char of size 6
- These variables are used to communicate errors and exception conditions between the database and the host language program. The value 0 in SQLCODE means successful execution of SQL command. A value of the SQLCODE =100 means ‘no more data’. The value of SQLCODE if less than 0 indicates an error. Similarly, SQLSTATE is a 5 char code the 6th char is for ‘\0’ in the host language ‘C’. Value “00000” in an SQLSTATE indicate no error. You can refer to SQL standard in more detail for more information.
- In order to execute the required SQL command, connection with the database server need to be established by the program. For this, the following SQL statement is used:

CONNECT <name of the server> AS <name of the connection>
AUTHORISATION <username, password>,

TO DISCONNECT we can simply say

DISCONNECT <name of the connection>;

However, these statements need to be checked in the commercial database management system, which you are using.

Execution of SQL query in the given program: To create the SQL query, first, the given value of enrolment number is transferred to SQL attribute value, the query then is executed and the result, which is a single tuple in this case, is transferred to shared host variables as indicated by the key word INTO after the SELECT statement.

The SQL query runs as a standard SQL query except the use of shared host variables. Rest of the C program has very simple logic and will print the data of the students whose enrolment number has been entered.



Please note that in this query, as the enrolment number is a key to the relation, only one tuple will be transferred to the shared host variables. But what will happen if more than one tuple results on the execution of embedded SQL query. Such situations are handled with the help of a cursor. Let us discuss such queries in more detail.

3.3.2 Cursors and Embedded SQL

Let us first define the terminate ‘cursor’. The Cursor can be defined as a pointer to the current tuple. It can also be defined as a portion of RAM allocated for the internal processing that has been set aside by the database server for database interactions. This portion may be used for query processing using SQL. But, what size of memory is to be allotted for the cursor? Ideally, the size allotted for the cursor should be equal to the memory required to hold the tuples that result from a query. However, the available memory puts a constraint on this size. Whenever a query results in a number of tuples, we can use a cursor to process the currently available tuples one by one. How? Let us explain the use of the cursor with the help of an example:

Since most of the commercial RDBMS architectures are client-server architectures, on execution of an embedded SQL query, the resulting tuples are cached in the cursor. This operation is performed on the server. Sometimes the cursor is opened by RDBMS itself – these are called **implicit cursors**. However, in embedded SQL you need to declare these cursors explicitly – these are called **explicit cursors**. Any cursor needs to have the following operations defined on them:

DECLARE – to declare the cursor

OPEN AND CLOSE - to open and close the cursor

FETCH – get the current records one by one till end of tuples.

In addition, cursors have some attributes through which we can determine the state of the cursor. These may be:

ISOPEN – It is true if cursor is OPEN, otherwise false.

FOUND/NOT FOUND – It is true if a row is fetched successfully/not successfully.

ROWCOUNT – It determines the number of tuples in the cursor.

Let us explain the use of the cursor with the help of an example:

Example: Write a C program segment that inputs the final grade of the students of MCA programme.

Let us assume the relation:

```
STUDENT (enrolno:char(9), name:Char(25), phone:integer(12),
          prog-code:char(3)); grade: char(1));
```

The program segment is:

```
/* add proper include statements*/
/*declaration in C program */
```

```
EXEC SQL BEGIN DECLARE SECTION;
    Char enrolno[10], name[26], p-code[4], grade /* grade is just one character*/
    int phone;
    int SQLCODE;
    char SQLSTATE[6]
EXEC SQL END DECLARE SECTION;
/* The connection needs to be established with SQL*/
/* program segment for the required function */
```

```
printf ("enter the programme code);  
scanf ("%s, &p-code);  
EXEC SQL DECLARE CURSOR GUPDATE  
    SELECT enrolno, name, phone, grade  
    FROM STUDENT  
    WHERE progcode =: p-code  
    FOR UPDATE OF grade;  
EXEC SQL OPEN GUPDATE;  
EXEC SQL FETCH FROM GUPDATE  
    INTO :enrolno, :name, :phone, :grade;  
WHILE (SQLCODE==0) {  
    printf ("enter grade for enrolment number, \"%s\", enrolno);  
    scanf ("%c", grade);  
    EXEC SQL  
        UPDATE STUDENT  
        SET grade=:grade  
        WHERE CURRENT OF GUPDATE  
    EXEC SQL FETCH FROM GUPDATE;  
}  
EXEC SQL CLOSE GUPDATE;
```

- Please note that the declared section remains almost the same. The cursor is declared to contain the output of the SQL statement. Please notice that in this case, there will be many tuples of students database, which belong to a particular programme.
- The purpose of the cursor is also indicated during the declaration of the cursor.
- The cursor is then opened and the first tuple is fetch into shared host variable followed by SQL query to update the required record. Please note the use of CURRENT OF which states that these updates are for the current tuple referred to by the cursor.
- WHILE Loop is checking the SQLCODE to ascertain whether more tuples are pending in the cursor.
- Please note the SQLCODE will be set by the last fetch statement executed just prior to while condition check.

How are these SQL statements compiled and error checked during embedded SQL?

- The SQL pre-compiler performs the type of checking of the various shared host variables to find any mismatches or errors on each of the SQL statements. It then stores the results into the SQLCODE or SQLSTATE variables.

Is there any limitation on these statically embedded SQL statements?

They offer only limited functionality, as the query must be known at the time of application development so that they can be pre-compiled in advance. However, many queries are not known at the time of development of an application; thus we require dynamically embedded SQL also.

3.3.3 Dynamic SQL

Dynamic SQL, unlike embedded SQL statements, are built at the run time and placed in a string in a host variable. The created SQL statements are then sent to the DBMS for processing. Dynamic SQL is generally slower than statically embedded SQL as they require complete processing including access plan generation during the run time.

However, they are more powerful than *embedded SQL* as they allow run time application logic. The basic advantage of using dynamic embedded SQL is that we need not compile and test a new program for a new query.



Let us explain the use of dynamic SQL with the help of an example:

Example: Write a dynamic SQL interface that allows a student to get and modify permissible details about him/her. The student may ask for subset of information also. Assume that the student database has the following relations.

STUDENT (enrolno, name, dob)

RESULT (enrolno, coursecode, marks)

In the table above, a student has access rights for accessing information on his/her enrolment number, but s/he cannot update the data. Assume that user names are enrolment number.

Solution: A sample program segment may be (please note that the syntax may change for different commercial DBMS).

```
/* declarations in SQL */  
EXEC SQL BEGIN DECLARE SECTION;  
    char inputfields(50);  
    char tablename(10)  
    char sqlquery ystring(200)  
EXEC SQL END DECLARE SECTION;  
    printf ("Enter the fields you want to see \n");  
    scanf ("SELECT% s", inputfields);  
    printf ("Enter the name of table STUDENT or RESULT");  
    scanf ("FROM% s", tablename);  
    sqlqueryystring = "SELECT" +inputfields +" "+  
        "FROM" + tablename  
        + "WHERE enrolno + :USER"  
/*Plus is used as a symbol for concatenation operator; in some DBMS it may be ||*/  
/* Assumption: the user name is available in the host language variable USER*/  
  
EXEC SQL PREPARE sqlcommand FROM :sqlqueryystring;  
EXEC SQL EXECUTE sqlcommand;
```

Please note the following points in the example above.

- The query can be entered completely as a string by the user or s/he can be suitably prompted.
- The query can be fabricated using a concatenation of strings. This is language dependent in the example and is not a portable feature in the present query.
- The query modification of the query is being done keeping security in mind.
- The query is prepared and executed using a suitable SQL EXEC commands.

3.3.4 SQLJ

Till now we have talked about embedding SQL in C, but how can we embed SQL statements into JAVA Program? For this purpose we use SQLJ. In SQLJ, a preprocessor called SQLJ translator translates SQLJ source file to JAVA source file. The JAVA file compiled and run on the database. Use of SQLJ improves the productivity and manageability of JAVA Code as:

- The code becomes somewhat compact.
- No run-time SQL syntax errors as SQL statements are checked at compile time.
- It allows sharing of JAVA variables with SQL statements. Such sharing is not possible otherwise.

Please note that SQLJ cannot use dynamic SQL. It can only use simple embedded SQL. SQLJ provides a standard form in which SQL statements can be embedded in

JAVA program. SQLJ statements always begin with a #sql keyword. These embedded SQL statements are of two categories – Declarations and Executable Statements.

Declarations have the following syntax:

```
#sql <modifier> context context_classname;
```

The executable statements have the following syntax:

```
#sql {SQL operation returning no output};
```

OR

```
#sql result = {SQL operation returning output};
```

Example:

Let us write a JAVA function to print the student details of student table, for the student who have taken admission in 2005 and name are like ‘Shyam’. Assuming that the first two digits of the 9-digit enrolment number represents a year, the required input conditions may be:

- The enrolno should be more than “05000000” and
- The name contains the sub string “Shyam”.

Please note that these input conditions will not be part of the Student Display function, rather will be used in the main () function that may be created separately by you. The following display function will accept the values as the parameters supplied by the main () .

```
Public void DISPSTUDENT (String enrolno, String name, int phone)
{
    try {
        if ( name equals ( " " ) )
            name = "%";
        if(enrolno equals ( " " ) )
            enrolno = "%";
        SelRowIter      srows = null;
        # sql srows = { SELECT Enrolno, name, phone
                        FROM STUDENT
                        WHERE enrolno > :enrolno AND name like :name
                    };
        while ( srows.next ( ) ) {
            int enrolno      = srows.enrolno ( );
            String name      = srows.name ( );

            System.out.println ( "Enrollment_No = " + enrolno);
            System.out.println ("Name =" +name);
            System.out.println ("phone =" +phone);
        }
    } Catch (Exception e) {
        System.out.println ( " error accessing database" + e.to_string);
    }
}
```



☛ Check Your Progress 2

- 1) A University decided to enhance the marks for the students by 2 in the subject MCS-043 in the table: RESULT (enrolno, coursecode, marks). Write a segment of embedded SQL program to do this processing.

- 2) What is dynamic SQL?

- 3) Can you embed dynamic SQL in JAVA?

3.4 STORED PROCEDURES AND TRIGGERS

In this section, we will discuss some standard features that make commercial databases a strong implementation tool for information systems. These features are triggers and stored procedures. Let us discuss about them in more detail in this section.

3.4 .1 Stored Procedures

Stored procedures are collections of small programs that are stored in compiled form and have a specific purpose. For example, a company may have rules such as:

- A code (like enrolment number) with one of the digits as the check digit, which checks the validity of the code.
- Any date of change of value is to be recorded.

These rules are standard and need to be made applicable to all the applications that may be written. Thus, instead of inserting them in the code of each application they may be put in a stored procedure and reused.

The use of procedure has the following advantages from the viewpoint of database application development.

- They help in removing SQL statements from the application program thus making it more readable and maintainable.
- They run faster than SQL statements since they are already compiled in the database.

Stored procedures can be created using CREATE PROCEDURE in some commercial DBMS.

Syntax:

```
CREATE [or replace] PROCEDURE [user]PROCEDURE_NAME  
[(argument datatype  
[, argument datatype]....)]
```



BEGIN

Host Language statements;

END;

For example, consider a data entry screen that allows entry of enrolment number, first name and the last name of a student combines the first and last name and enters the complete name in upper case in the table STUDENT. The student table has the following structure:

STUDENT (enrolno:char(9), name:char(40));

The stored procedure for this may be written as:

```
CREATE PROCEDURE studententry (
    enrolment IN char (9);
    f-name     INOUT char (20);
    l-name     INOUT char (20)
BEGIN
    /* change all the characters to uppercase and trim the length */
    f-name TRIM = UPPER (f-name);
    l-name TRIM = UPPER (l-name);
    name   TRIM = f-name || . . . || l-name;
    INSERT INTO CUSTOMER
        VALUES (enrolment, name);
END;
```

INOUT used in the host language indicates that this parameter may be used both for input and output of values in the database.

While creating a procedure, if you encounter errors, then you can use the ***show errors*** command. It shows all the error encountered by the most recently created procedure object.

You can also write an SQL command to display errors. The syntax of finding an error in a commercial database is:

```
SELECT *
FROM USER_ERRORS
WHERE Name='procedure name' and type='PROCEDURE';
```

Procedures are compiled by the DBMS. However, if there is a change in the tables, etc. referred to by the procedure, then the procedure needs to be recompiled. You can recompile the procedure explicitly using the following command:

`ALTER PROCEDURE procedure_name COMPILE;`

You can drop a procedure by using `DROP PROCEDURE` command.

3.4.2 Triggers

Triggers are somewhat similar to stored procedures except that they are activated automatically. When a trigger is activated? A trigger is activated on the occurrence of a particular event. What are these events that can cause the activation of triggers?



These events may be database update operations like INSERT, UPDATE, DELETE etc. A trigger consists of these essential components:

- An event that causes its automatic activation.
- The condition that determines whether the event has called an exception such that the desired action is executed.
- The action that is to be performed.

Triggers do not take parameters and are activated automatically, thus, are different to stored procedures on these accounts. Triggers are used to implement constraints among more than one table. Specifically, the triggers should be used to implement the constraints that are not implementable using referential integrity/constraints. An instance, of such a situation may be when an update in one relation affects only few tuples in another relation. However, please note that you should not be over enthusiastic for writing triggers – if any constraint is implementable using declarative constraints such as PRIMARY KEY, UNIQUE, NOT NULL, CHECK, FOREIGN KEY, etc. then it should be implemented using those declarative constraints rather than triggers, primarily due to performance reasons.

You may write triggers that may execute once for each row in a transaction – called Row Level Triggers or once for the entire transaction Statement Level Triggers. Remember, that you must have proper access rights on the table on which you are creating the trigger. For example, you may have all the rights on a table or at least have UPDATE access rights on the tables, which are to be used in trigger. The following is the syntax of triggers in one of the commercial DBMS:

```
CREATE TRIGGER <trigger_name>
[BEFORE | AFTER]
<Event>
ON <tablename>
[WHEN <condition> | FOR EACH ROW]
<Declarations of variables if needed is – may be used when creating trigger using host language>
BEGIN
    <SQL statements OR host language SQL statements>
[EXCEPTION]
    <Exceptions if any>
END;
```

Let us explain the use of triggers with the help of an example:

Example:

Consider the following relation of a students database

STUDENT(enrolno, name, phone)
RESULT (enrolno, coursecode, marks)
COURSE (course-code, c-name, details)

Assume that the marks are out of 100 in each course. The passing marks in a subject are 50. The University has a provision for 2% grace marks for the students who are failing marginally – that is if a student has 48 marks, s/he is given 2 marks grace and if a student has 49 marks then s/he is given 1 grace mark. Write the suitable trigger for this situation.

Please note the requirements of the trigger:

Event: UPDATE of marks



OR

INSERT of marks

Condition: When student has 48 OR 49 marks

Action: Give 2 grace marks to the student having 48 marks and 1 grace mark to the student having 49 marks.

The trigger for this thus can be written as:

```
CREATE TRIGGER grace
AFTER INSERT OR UPDATE OF marks ON RESULT
WHEN (marks = 48 OR marks =49)
UPDATE RESULT
SET marks =50;
```

We can drop a trigger using a DROP TRIGGER statement

```
DROP TRIGGER trigger_name;
```

The triggers are implemented in many commercial DBMS. Please refer to them in the respective DBMS for more details.

3.5 ADVANCED FEATURES OF SQL

The latest SQL standards have enhanced SQL tremendously. Let us touch upon some of these enhanced features. More details on these would be available in the sources mentioned in ‘further readings’.

SQL Interfaces: SQL also has a good programming level interfaces. The SQL supports a library of functions for accessing a database. These functions are also called the Application Programming Interface (API) of SQL. The advantage of using an API is that it provides flexibility in accessing multiple databases in the same program irrespective of DBMS, while the disadvantage is that it requires more complex programming. The following are two common functions called interfaces:

SQL/CLI (SQL – call level interface) is an advanced form of Open Database Connectivity (ODBC).

Java Database Connectivity (JDBC) – allows object-oriented JAVA to connect to the multiple databases.

SQL support for object-orientation: The latest SQL standard also supports the object-oriented features. It allows creation of abstract data types, nested relations, object identifies etc.

Interaction with Newer Technologies: SQL provides support for XML (eXtended Markup Language) and Online Analytical Processing (OLAP) for data warehousing technologies.

☛ Check Your Progress 3

- 1) What is stored procedure?

.....
.....



- 2) Write a trigger that restricts updating of STUDENT table outside the normal working hours/holiday.
-

3.6 SUMMARY

This unit has introduced some important advanced features of SQL. The unit has also provided information on how to use these features.

An assertion can be defined as the general constraint on the states of the database. These constraints are expected to be satisfied by the database at all times. The assertions can be stored as stored procedures.

Views are the external schema windows of the data from a database. Views can be defined on a single table or multiple tables and help in automatic security of hidden data. All the views cannot be used for updating data in the tables. You can query a view.

The embedded SQL helps in providing a complete host language support to the functionality of SQL, thus making application programming somewhat easier. An embedded query can result in a single tuple, however, if it results in multiple tuple then we need to use cursors to perform the desired operation. Cursor is a sort of pointer to the area in the memory that contains the result of a query. The cursor allows sequential processing of these result tuples. The SQLJ is the embedded SQL for JAVA. The dynamic SQL is a way of creating queries through an application and compiling and executing them at the run time. Thus, it provides dynamic interface and hence the name.

Stored procedures are the procedures that are created for some application purpose. These procedures are precompiled procedures and can be invoked from application programs. Arguments can be passed to a stored procedure and it can return values. A trigger is also like a stored procedure except that it is invoked automatically on the occurrence of a specified event.

You should refer to the books further readings list for more details relating to this unit.

3.7 SOLUTIONS/ANSWERS

Check Your Progress 1

- 1) The constraint is a simple Range constraint and can easily be implemented with the help of declarative constraint statement. (You need to use CHECK CONSTRAINT statement). Therefore, there is no need to write an assertion for this constraint.
- 2)

```
CREATE VIEW avgmarks AS (
    SELECT subjectcode, AVG(smarks)
    FROM MARKS
    GROUP BY subjectcode);
```
- 3) No, the view is using a GROUP BY clause. Thus, if we try to update the subjectcode. We cannot trace back a single tuple where such a change needs to take place. Please go through the rules for data updating though views.

Check Your Progress 2

1)

```
/*The table is RESULT (enrolno.coursecode, marks). */
EXEC SQL BEGIN DECLARE SECTION;
    char enrolno [10], coursecode[7]; /* grade is just one character*/
    int marks;
    char SQLSTATE[6]
EXEC SQL END DECLARE SECTION;
/*The connection needs to be established with sQL*/
/* program segment for the required function*/
    printf ("enter the course code for which 2 grace marks are to be added")
    scanf ("%s", &coursecode);
EXEC SQL DECLARE CURSOR GRACE
    SELECT enrolno, coursecode, marks
        FROM RESULT
        WHERE coursecode=:coursecode
        FOR UPDATE OF marks;
EXEC SQL OPEN GRACE;
EXEC SQL FETCH FROM GRACE
    INTO :enrolno, :coursecode, :marks;
WHILE (SQL CODE==0) {
    EXEC SQL
        UPDATE RESULT
        SET marks = marks+2
        WHERE CURRENT OF GRACE;
    EXEC SQL FETCH FROM GRACE;
}
EXEC SQL CLOSE GRACE;
```

An alternative implementation in a commercial database management system may be:

```
DECLARE CURSOR grace IS
    SELECT enrolno, coursecode, marks
        FROM RESULT
        WHERE coursecode ='MCS043';
str_enrolno RESULT.enrolno%type;
str_coursecode RESULT.coursecode%type;
str_marks RESULT.marks%type;
BEGIN
    OPEN grace;
IF GRACE %OPEN THEN
    LOOP
        FETCH grace INTO str_enrolno, str_coursecode, str_marks;
        Exit when grace%NOTFOUND;
        UPDATE student SET marks=str_marks +2;
        INSERT INTO resultmcs-43 VALUES (str_enrolno,
            str_coursecode, str_marks);
    END LOOP;
    COMMIT;
    CLOSE grace;
ELSE
    Dbms_output.put_line ('Unable to open cursor');
END IF;
END;
```



- 2) Dynamic SQL allows run time query making through embedded languages. The basic step here would be first to create a valid query string and then to execute that query string. Since the queries are compiled and executed at the run time thus, it is slower than simple embedded SQL.
- 3) No, at present JAVA cannot use dynamic SQL.

Check Your Progress 3

- 1) Stored procedure is a compiled procedure in a host language that has been written for some purpose.

- 2) The trigger is some pseudo DBMS may be written as:

```
CREATE TRIGGER resupdstudent
BEFORE INSERT OR UPDATE OR DELETE ON STUDENT
BEGIN
IF (DAY ( SYSDATE ) IN ('SAT', 'SUN')) OR
(HOURS (SYSDATE) NOT BETWEEN '09:00' AND 18:30')
THEN
RAISE_EXCEPTION AND OUTPUT ('OPERATION NOT
ALLOWED AT THIS TIME/DAY');
END IF;
END
```

Please note that we have used some hypothetical functions, syntax of which will be different in different RDBMS.

UNIT 10 TRANSACTIONS AND CONCURRENCY MANAGEMENT

(Adopted from MCS-023 Block-2 Unit-2)

Structure	Page Nos.
10.0 Introduction	
10.1 Objectives	
10.2 The Transactions	
10.3 The Concurrent Transactions	
10.4 The Locking Protocol	
10.4.1 Serialisable Schedules	
10.4.2 Locks	
10.4.3 Two Phase Locking (2PL)	
10.5 Deadlock and its Prevention	
10.6 Optimistic Concurrency Control	
10.7 Summary	
10.8 Solutions/ Answers	

10.0 INTRODUCTION

One of the main advantages of storing data in an integrated repository or a database is to allow sharing of it among multiple users. Several users access the database or perform transactions at the same time. What if a user's transactions try to access a data item that is being used /modified by another transaction? This unit attempts to provide details on how concurrent transactions are executed under the control of DBMS. However, in order to explain the concurrent transactions, first we must describe the term transaction.

Concurrent execution of user programs is essential for better performance of DBMS, as concurrent running of several user programs keeps utilizing CPU time efficiently, since disk accesses are frequent and are relatively slow in case of DBMS. Also, a user's program may carry out many operations on the data returned from DB, but DBMS is only concerned about what data is being read /written from/ into the database. This unit discusses the issues of concurrent transactions in more detail.

10.1 OBJECTIVES

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

- describe the term CONCURRENCY;
- define the term transaction and concurrent transactions;
- discuss about concurrency control mechanism;
- describe the principles of locking and serialisability, and
- describe concepts of deadlock & its prevention.

10.2 THE TRANSACTIONS

A transaction is defined as the unit of work in a database system. Database systems

that deal with a large number of transactions are also termed as transaction processing systems.

What is a transaction? Transaction is a unit of data processing. For example, some of the transactions at a bank may be withdrawal or deposit of money; transfer of money from A's account to B's account etc. A transaction would involve manipulation of one or more data values in a database. Thus, it may require reading and writing of database value. For example, the withdrawal transactions can be written in pseudocode as:

Example 1:

; Assume that we are doing this transaction for person
; whose account number is X.

```
TRANSACTION WITHDRAWAL (withdrawal_amount)
Begin transaction
    IF X exist then
        READ X.balance
        IF      X.balance > withdrawal_amount
        THEN SUBTRACT withdrawal_amount
        WRITE X.balance
        COMMIT
    ELSE
        DISPLAY "TRANSACTION CANNOT BE PROCESSED"
    ELSE DISPLAY "ACCOUNT X DOES NOT EXIST"
End transaction;
```

Another similar example may be transfer of money from Account no x to account number y. This transaction may be written as:

Example 2:

; transfers transfer_amount from x's account to y's account
; assumes x&y both accounts exist

```
TRANSACTION (x, y, transfer_amount)
Begin transaction
    IF X AND Y exist then
        READ x.balance
        IF x.balance > transfer_amount THEN
            x.balance = x.balance - transfer_amount
            READ y.balance
            y.balance = y.balance + transfer_amount
            COMMIT
        ELSE DISPLAY ("BALANCE IN X NOT OK")
        ROLLBACK
    ELSE DISPLAY ("ACCOUNT X OR Y DOES NOT EXIST")
End_transaction
```

Please note the use of two keywords here COMMIT and ROLLBACK. Commit makes sure that all the changes made by transactions are made permanent.

ROLLBACK terminates the transactions and rejects any change made by the transaction. Transactions have certain desirable properties. Let us look into those properties of a transaction.

Properties of a Transaction

A transaction has four basic properties. These are:

- Atomicity
- Consistency
- Isolation or Independence
- Durability or Permanence

Atomicity: It defines a transaction to be a single unit of processing. In other words either a transaction will be done *completely or not at all*. In the transaction example 1 & 2 please note that transaction 2 is reading and writing more than one data items, the atomicity property requires either operations on both the data item be performed or not at all.

Consistency: This property ensures that a complete transaction execution takes a database from one consistent state to another consistent state. If a transaction fails even then the database should come back to a consistent state.

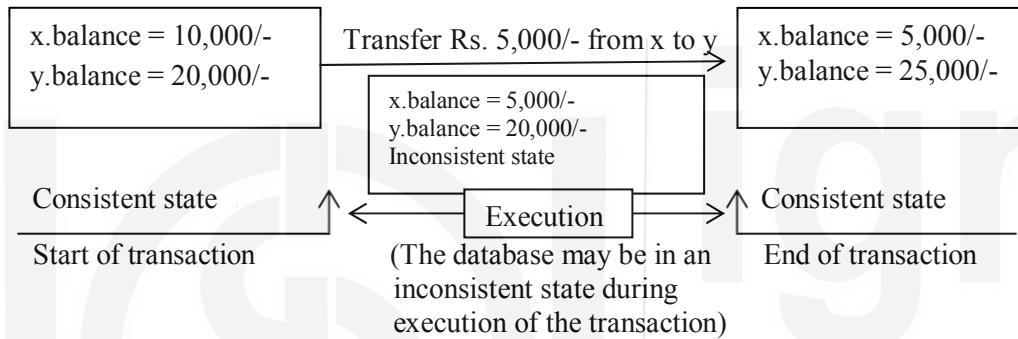


Figure 1: A Transaction execution

Isolation or Independence: The isolation property states that the updates of a transaction should not be visible till they are committed. Isolation guarantees that the progress of other transactions do not affect the outcome of this transaction. For example, if another transaction that is a withdrawal transaction which withdraws an amount of Rs. 5000 from X account is in progress, whether fails or commits, should not affect the outcome of this transaction. Only the state that has been read by the transaction last should determine the outcome of this transaction.

Durability: This property necessitates that once a transaction has committed, the changes made by it be never lost because of subsequent failure. Thus, a transaction is also a basic unit of recovery. The details of transaction-based recovery are discussed in the next unit.

A transaction has many states of execution. These states are displayed in *Figure 2*.

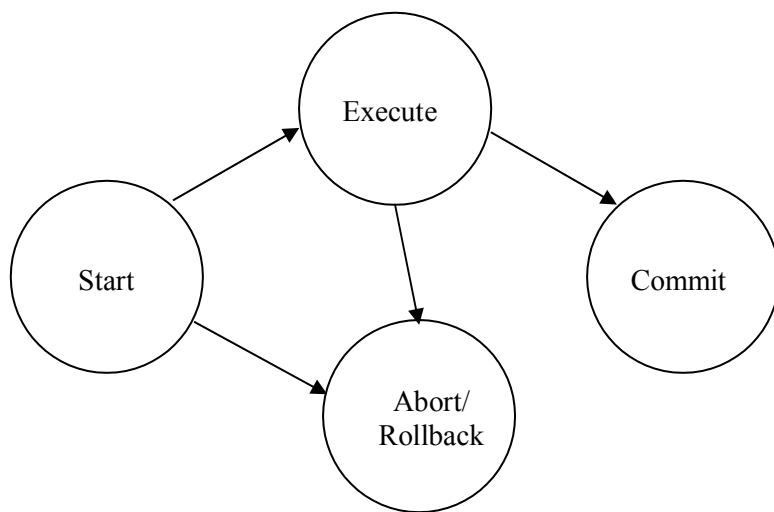


Figure 2: States of transaction execution

A transaction is started as a program. From the start state as the transaction is scheduled by the CPU it moves to the Execute state, however, in case of any system error at that point it may also be moved into the Abort state. During the execution transaction changes the data values and database moves to an inconsistent state. On successful completion of transaction it moves to the Commit state where the durability feature of transaction ensures that the changes will not be lost. In case of any error the transaction goes to Rollback state where all the changes made by the transaction are undone. Thus, after commit or rollback database is back into consistent state. In case a transaction has been rolled back, it is started as a new transaction. All these states of the transaction are shown in *Figure 2*.

10.3 THE CONCURRENT TRANSACTIONS

Almost all the commercial DBMS support multi-user environment. Thus, allowing multiple transactions to proceed simultaneously. The DBMS must ensure that two or more transactions do not get into each other's way, i.e., transaction of one user doesn't effect the transaction of other or even the transactions issued by the same user should not get into the way of each other. Please note that concurrency related problem may occur in databases only if **two transactions are contending for the same data item and at least one of the concurrent transactions wishes to update a data value in the database**. In case, the concurrent transactions only read same data item and no updates are performed on these values, then it does NOT cause any concurrency related problem. Now, let us first discuss why you need a mechanism to control concurrency.

Consider a banking application dealing with checking and saving accounts. A Banking Transaction T1 for Mr. Sharma moves Rs.100 from his checking account balance X to his savings account balance Y, using the transaction T1:

Transaction T1:

```
A:Read X  
    Subtract 100  
    Write X  
B:Read Y  
    Add 100  
    Write Y
```

Let us suppose an auditor wants to know the total assets of Mr. Sharma. He executes the following transaction:

Transaction T2:

```
Read X  
Read Y  
Display X+Y
```

Suppose both of these transactions are issued simultaneously, then the execution of these instructions can be mixed in many ways. This is also called the **Schedule**. Let us define this term in more detail.

A schedule S is defined as the sequential ordering of the operations of the 'n' interleaved transactions. A schedule maintains the order of operations within the individual transaction.

Conflicting Operations in Schedule: Two operations of different transactions conflict if they access the same data item AND one of them is a write operation.

For example, the two transactions TA and TB as given below, if executed in parallel, may produce a schedule:

TA
READ X
WRITE X

TB
READ X
WRITE X

SCHEDULE	TA	TB
READ X	READ X	
READ X		READ X
WRITE X		WRITE X
WRITE X	WRITE X	

One possible schedule for interleaved execution of TA and TB

Let us show you three simple ways of interleaved instruction execution of transactions T1 and T2. Please note that in the following tables the first column defines the sequence of instructions that are getting executed, that is the schedule of operations.

- a) T2 is executed completely before T1 starts, then sum X+Y will show the correct assets:

Schedule	Transaction T1	Transaction T2	Example Values
Read X		Read X	X = 50000
Read Y		Read Y	Y= 100000
Display X+Y		Display X+Y	150000
Read X	Read X		X = 50000
Subtract 100	Subtract 100		49900
Write X	Write X		X = 49900
Read Y	Read Y		Y= 100000
Add 100	Add 100		100100
Write Y	Write Y		Y= 100100

- b) T1 is executed completely before T2 starts, then sum X+Y will still show the correct assets:

Schedule	Transaction T1	Transaction T2	Example Values
Read X	Read X		X = 50000
Subtract 100	Subtract 100		49900
Write X	Write X		X = 49900
Read Y	Read Y		Y= 100000
Add 100	Add 100		100100
Write Y	Write Y		Y= 100100
Read X		Read X	X = 49900
Read Y		Read Y	Y= 100100
Display X+Y		Display X+Y	150000

- c) Block A in transaction T1 is executed, followed by complete execution of T2, followed by the Block B of T1.

Schedule	Transaction T1	Transaction T2	Example Values
Read X	Read X		X = 50000
Subtract 100	Subtract 100		49900
Write X	Write X		X = 49900
Read X		Read X	X = 49900
Read Y		Read Y	Y= 100000
Display X+Y		Display X+Y	149900
Read Y	Read Y		Y= 100000
Add 100	Add 100		100100
Write Y	Write Y		Y= 100100

In this execution an incorrect value is being displayed. This is because Rs.100 although removed from X, has not been put in Y, and is thus missing. Obviously, if T1 had been written differently, starting with block B and following up with block A, even then such an interleaving would have given a different but incorrect result.

Please note that for the given transaction there are many more ways of this interleaved instruction execution.

Thus, there may be a problem when the transactions T1 and T2 are allowed to execute in parallel. Let us define the problems of concurrent execution of transaction more precisely.

Let us assume the following transactions (assuming there will not be errors in data while execution of transactions)

Transaction T3 and T4: T3 reads the balance of account X and subtracts a withdrawal amount of Rs. 5000, whereas T4 reads the balance of account X and adds an amount of Rs. 3000

T3	T4
READ X	READ X
SUB 5000	ADD 3000
WRITE X	WRITE X

Problems of Concurrent Transactions

1. **Lost Updates:** Suppose the two transactions T3 and T4 run concurrently and they happen to be interleaved in the following way (assume the initial value of X as 10000):

T3	T4	Value of X	
		T3	T4
READ X		10000	
	READ X		10000
SUB 5000		5000	
	ADD 3000		13000
WRITE X		5000	
	WRITE X		13000

After the execution of both the transactions the value X is 13000 while the

semantically correct value should be 8000. The problem occurred as the update made by T3 has been overwritten by T4. The root cause of the problem was the fact that both the transactions had read the value of X as 10000. Thus one of the two updates has been lost and we say that a **lost update** has occurred.

There is one more way in which the lost updates can arise. Consider the following part of some transactions:

T5	T6	Value of x originally 2000	
		T5	T6
UPDATE X		3000	
	UPDATE X		4000
ROLLBACK		2000	

Here T5 & T6 updates the same item X. Thereafter T5 decides to undo its action and rolls back causing the value of X to go back to the original value that was 2000. In this case also the update performed by T6 had got lost and a lost update is said to have occurred.

2. **Unrepeatable reads:** Suppose T7 reads X twice during its execution. If it did not update X itself it could be very disturbing to see a different value of X in its next read. But this could occur if, between the two read operations, another transaction modifies X.

T7	T8	Assumed value of X=2000	
		T7	T8
READ X		2000	
	UPDATE X		3000
READ X		3000	

Thus, the inconsistent values are read and results of the transaction may be in error.

3. **Dirty Reads:** T10 reads a value which has been updated by T9. This update has not been committed and T9 aborts.

T9	T10	Value of x old value = 200	
		T9	T10
UPDATE X		500	
	READ X		500
ROLLBACK		200	?

Here T10 reads a value that has been updated by transaction T9 that has been aborted. Thus T10 has read a value that would never exist in the database and hence the problem. Here the problem is primarily of isolation of transaction.

4. **Inconsistent Analysis:** The problem as shown with transactions T1 and T2 where two transactions interleave to produce incorrect result during an analysis by Audit is the example of such a problem. This problem occurs when more than one data items are being used for analysis, while another transaction has modified some of those values and some are yet to be modified. Thus, an analysis transaction reads values from the inconsistent state of the database that results in inconsistent analysis.

Thus, we can conclude that the prime reason of problems of concurrent transactions is that a transaction reads an inconsistent state of the database that has been created by

other transaction.

But how do we ensure that execution of two or more transactions have not resulted in a concurrency related problem?

Well one of the commonest techniques used for this purpose is to restrict access to data items that are being read or written by one transaction and is being written by another transaction. This technique is called locking. Let us discuss locking in more detail in the next section.

Check Your Progress 1

- 1) What is a transaction? What are its properties? Can a transaction update more than one data values? Can a transaction write a value without reading it? Give an example of transaction.

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

- 2) What are the problems of concurrent transactions? Can these problems occur in transactions which do not read the same data values?

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

- 3) What is a Commit state? Can you rollback after the transaction commits?

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

10.4 THE LOCKING PROTOCOL

To control concurrency related problems we use locking. A lock is basically a variable that is associated with a data item in the database. A lock can be placed by a transaction on a shared resource that it desires to use. When this is done, the data item is available for the exclusive use for that transaction, i.e., other transactions are locked out of that data item. When a transaction that has locked a data item does not desire to use it any more, it should unlock the data item so that other transactions can use it. If a transaction tries to lock a data item already locked by some other transaction, it cannot do so and waits for the data item to be unlocked. The component of DBMS that controls and stores lock information is called the Lock Manager. The locking mechanism helps us to convert a schedule into a serialisable schedule. We had defined what a schedule is, but what is a serialisable schedule? Let us discuss about it in more detail:

10.4.1 Serialisable Schedules

If the operations of two transactions conflict with each other, how to determine that no concurrency related problems have occurred? For this, serialisability theory has been developed. Serialisability theory attempts to determine the **correctness** of the schedules. The rule of this theory is:

"A schedule S of n transactions is serialisable if it is equivalent to some **serial schedule of the same 'n' transactions".**

A serial schedule is a schedule in which either transaction T1 is completely done before T2 or transaction T2 is completely done before T1. For example, the following figure shows the two possible serial schedules of transactions T1 & T2.

Schedule A: T2 followed by T1		Schedule B: T1 followed by T2		
Schedule	T1	T2	Schedule	T1
Read X		Read X	Read X	
Read Y		Read Y	Subtract 100	
Display X+Y		Display X+Y	Write X	
Read X	Read X		Read Y	Read Y
Subtract 100	Subtract 100		Add 100	Add 100
Write X	Write X		Write Y	
Read Y	Read Y		Read X	
Add 100	Add 100		Read Y	Read Y
Write Y	Write Y		Display X+Y	Display X+Y

Figure 3: Serial Schedule of two transactions

Schedule C: An Interleaved Schedule		
Schedule	T1	T2
Read X	Read X	
Subtract 100		Subtract 100
Read X		Read X
Write X	Write X	
Read Y		Read Y
Read Y	Read Y	
Add 100	Add 100	
Display X+Y		Display X+Y
Write Y	Write Y	

Figure 4: An Interleaved Schedule

Now, we have to figure out whether this interleaved schedule would be performing read and write in the same order as that of a serial schedule. If it does, then it is equivalent to a serial schedule, otherwise not. In case it is not equivalent to a serial schedule, then it may result in problems due to concurrent transactions.

Serialisability

Any schedule that produces the same results as a serial schedule is called a serialisable schedule. But how can a schedule be determined to be serialisable or not? In other words, other than giving values to various items in a schedule and checking if the results obtained are the same as those from a serial schedule, is there an algorithmic way of determining whether a schedule is serialisable or not?

The basis of the algorithm for serialisability is taken from the notion of a serial schedule. There are two possible serial schedules in case of two transactions (T1 - T2 OR T2 - T1). Similarly, in case of three parallel transactions the number of possible serial schedules is $3!$, that is, 6. These serial schedules can be:

T1-T2-T3 T1-T3-T2 T2-T1-T3
 T2-T3-T1 T3-T1-T2 T3-T2-T1

Using the notion of precedence graph, an algorithm can be devised to determine whether an interleaved schedule is serialisable or not. In this graph, the transactions of the schedule are represented as the nodes. This graph also has directed edges. An edge

from the node representing transactions T_i to node T_j means that there exists a **conflicting operation** between T_i and T_j and T_i precedes T_j in some conflicting operations. It has been proved that a serialisable schedule is the one that contains no cycle in the graph.

Given a graph with no cycles in it, there must be a serial schedule corresponding to it.

The steps of constructing a precedence graph are:

1. Create a node for every transaction in the schedule.
2. Find the precedence relationships in conflicting operations. Conflicting operations are (read-write) or (write-read) or (write-write) on the same data item in two different transactions. But how to find them?
 - 2.1 For a transaction T_i which *reads* an item A, find a transaction T_j that *writes* A later in the schedule. If such a transaction is found, draw an edge from T_i to T_j .
 - 2.2 For a transaction T_i which has *written* an item A, find a transaction T_j later in the schedule that *reads* A. If such a transaction is found, draw an edge from T_i to T_j .
 - 2.3 For a transaction T_i which has *written* an item A, find a transaction T_j that *writes* A later than T_i . If such a transaction is found, draw an edge from T_i to T_j .
3. If there is any cycle in the graph, the schedule is not serialisable, otherwise, find the equivalent serial schedule of the transaction by traversing the transaction nodes starting with the node that has no input edge.

Let us use this algorithm to check whether the schedule as given in Figure 4 is Serialisable. Figure 5 shows the required graph. Please note as per step 1, we draw the two nodes for T_1 and T_2 . In the schedule given in Figure 4, please note that the transaction T_2 reads data item X, which is subsequently written by T_1 , thus there is an edge from T_2 to T_1 (clause 2.1). Also, T_2 reads data item Y, which is subsequently written by T_1 , thus there is an edge from T_2 to T_1 (clause 2.1). However, that edge already exists, so we do not need to redo it. Please note that there are no cycles in the graph, thus, the schedule given in Figure 4 is serialisable. The equivalent serial schedule (as per step 3) would be T_2 followed by T_1 .

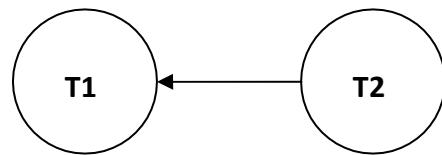


Figure 5: Test of Serialisability for the Schedule of Figure 4

Please note that the schedule given in part (c) of section 2.3 is not serialisable, because in that schedule, the two edges that exist between nodes T_1 and T_2 are:

- T_1 writes X which is later read by T_2 (clause 2.2), so there exists an edge from T_1 to T_2 .
- T_2 reads X which is later written by T_1 (clause 2.1), so there exists an edge from T_2 to T_1 .

Thus the graph for the schedule will be:

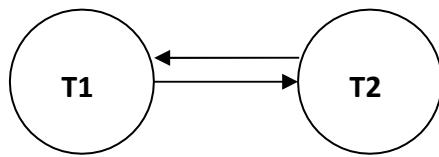


Figure 6: Test of Serialisability for the Schedule (c) of section 2.3

Please note that the graph above has a cycle T1-T2-T1, therefore it is not serialisable.

10.4.2 Locks

Serialisability is just a test whether a given interleaved schedule is ok or has a concurrency related problem. However, it does not ensure that the interleaved concurrent transactions do not have any concurrency related problem. This can be done by using locks. So let us discuss about what the different types of locks are, and then how locking ensures serialisability of executing transactions.

Types of Locks

There are two basic types of locks:

- Binary lock: This locking mechanism has two states for a data item: locked or unlocked
- Multiple-mode locks: In this locking type each data item can be in three states read locked or shared locked, write locked or exclusive locked or unlocked.

Let us first take an example for binary locking and explain how it solves the concurrency related problems. Let us reconsider the transactions T1 and T2 for this purpose; however we will add to required binary locks to them.

Schedule	T1	T2
Lock X	Lock X	
Read X	Read X	
Subtract 100	Subtract 100	
Write X	Write X	
Unlock X	Unlock X	
Lock X		Lock X
Lock Y		Lock Y
Read X		Read X
Read Y		Read Y
Display X+Y		Display X+Y
Unlock X		Unlock X
Unlock Y		Unlock Y
Lock Y	Lock Y	
Read Y	Read Y	
Add 100	Add 100	
Write Y	Write Y	
Unlock Y	Unlock Y	

Figure 7: An incorrect locking implementation

Does the locking as done above solve the problem of concurrent transactions? No the same problems still remains. Try working with the old value. Thus, locking should be done with some logic in order to make sure that locking results in no concurrency related problem. One such solution is given below:

Schedule	T1	T2
Lock X	Lock X	
Lock Y	Lock Y	
Read X	Read X	
Subtract 100	Subtract 100	
Write X	Write X	
Lock X (issued by T2)	Lock X: denied as T1 holds the lock. The transaction T2 Waits and T1 continues.	
Read Y	Read Y	
Add 100	Add 100	
Write Y	Write Y	
Unlock X	Unlock X	
	The lock request of T2 on X can now be granted it can resumes by locking X	
Unlock Y	Unlock Y	
Lock Y		Lock Y
Read X		Read X
Read Y		Read Y
Display X+Y		Display X+Y
Unlock X		Unlock X
Unlock Y		Unlock Y

Figure 8: A correct but restrictive locking implementation

Thus, the locking as above when you obtain all the locks at the beginning of the transaction and release them at the end ensures that transactions are executed with no concurrency related problems. However, such a scheme limits the concurrency. We will discuss a two-phase locking method in the next subsection that provides sufficient concurrency. However, let us first discuss multiple mode locks.

Multiple-mode locks: It offers two locks: shared locks and exclusive locks. But why do we need these two locks? There are many transactions in the database system that never update the data values. These transactions can coexist with other transactions that update the database. In such a situation multiple reads are allowed on a data item, so multiple transactions can lock a data item in the shared or read lock. On the other hand, if a transaction is an updating transaction, that is, it updates the data items, it has to ensure that no other transaction can access (read or write) those data items that it wants to update. In this case, the transaction places an exclusive lock on the data items. Thus, a somewhat higher level of concurrency can be achieved in comparison to the binary locking scheme.

The properties of shared and exclusive locks are summarised below:

a) **Shared lock or Read Lock**

- It is requested by a transaction that wants to just read the value of data item.
- A shared lock on a data item does not allow an exclusive lock to be placed but permits any number of shared locks to be placed on that item.

b) Exclusive lock

- It is requested by a transaction on a data item that it needs to update.
- No other transaction can place either a shared lock or an exclusive lock on a data item that has been locked in an exclusive mode.

Let us describe the above two modes with the help of an example. We will once again consider the transactions T1 and T2 but in addition a transaction T11 that finds the total of accounts Y and Z.

Schedule	T1	T2	T11
S_Lock X		S_Lock X	
S_Lock Y		S_Lock Y	
Read X		Read X	
S_Lock Y			S_Lock Y
S_Lock Z			S_Lock Z
			Read Y
			Read Z
X_Lock X	X_Lock X. The exclusive lock request on X is denied as T2 holds the Read lock. The transaction T1 Waits.		
Read Y		Read Y	
Display X+Y		Display X+Y	
Unlock X		Unlock X	
X_Lock Y	X_Lock Y. The previous exclusive lock request on X is granted as X is unlocked. But the new exclusive lock request on Y is not granted as Y is locked by T2 and T11 in read mode. Thus T1 waits till both T2 and T11 will release the read lock on Y.		
Display Y+Z			Display Y+Z
Unlock Y		Unlock Y	
Unlock Y			Unlock Y
Unlock Z			Unlock Z
Read X	Read X		
Subtract 100	Subtract 100		
Write X	Write X		
Read Y	Read Y		
Add 100	Add 100		
Write Y	Write Y		
Unlock X	Unlock X		
Unlock Y	Unlock Y		

Figure 9: Example of Locking in multiple-modes

Thus, the locking as above results in a serialisable schedule. Now the question is can we release locks a bit early and still have no concurrency related problem? Yes, we can do it if we lock using two-phase locking protocol. This protocol is explained in the next sub-section.

10.4.3 Two Phase Locking (2PL)

The two-phase locking protocol consists of two phases:

Phase 1: The lock acquisition phase: If a transaction T wants to read an object, it needs to obtain the S (shared) lock. If T wants to modify an object, it needs to obtain X (exclusive) lock. No conflicting locks are granted to a transaction. **New locks on items can be acquired but no lock can be released till all the locks required by the transaction are obtained.**

Phase 2: Lock Release Phase: The existing locks can be released in any order but no new lock can be acquired **after a lock has been released**. The locks are held only till they are required.

Normally the locks are obtained by the DBMS. Any legal schedule of transactions that follows 2 phase locking protocol is guaranteed to be serialisable. The two phase locking protocol has been proved for its correctness. However, the proof of this protocol is beyond the scope of this Unit. You can refer to further readings for more details on this protocol.

There are two types of 2PL:

- (1) The Basic 2PL
- (2) Strict 2PL

The basic 2PL allows release of lock at any time after all the locks have been acquired. For example, we can release the locks in schedule of *Figure 8*, after we have Read the values of Y and Z in transaction 11, even before the display of the sum. This will enhance the concurrency level. The basic 2PL is shown graphically in *Figure 10*.

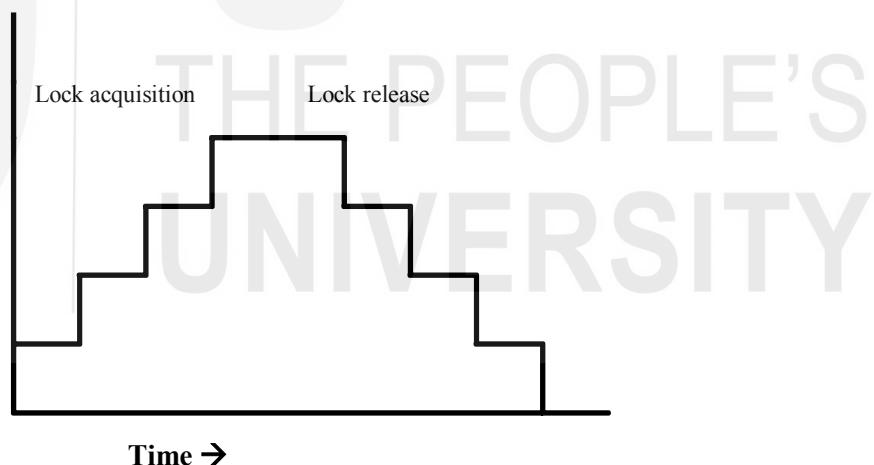


Figure 10: Basic Two Phase Locking

However, this basic 2PL suffers from the problem that it can result into loss of atomic / isolation property of transaction as theoretically speaking once a lock is released on data item it can be modified by another transaction before the first transaction commits or aborts.

To avoid such a situation we use strict 2PL. The strict 2PL is graphically depicted in *Figure 11*. However, the basic disadvantage of strict 2PL is that it restricts concurrency as it locks the item beyond the time it is needed by a transaction.

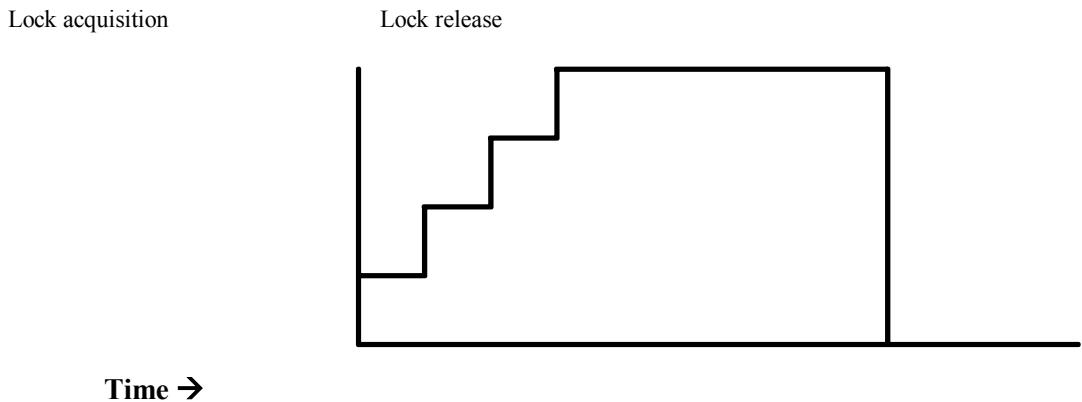


Figure 11: Strict Two Phase Locking

Does the 2PL solve all the problems of concurrent transactions? No, the strict 2PL solves the problem of concurrency and atomicity, however it introduces another problem: “Deadlock”. Let us discuss this problem in next section.

☞ Check Your Progress 2

- 1) Let the transactions T1, T2, T3 be defined to perform the following operations:

T1: Add one to A
 T2: Double A
 T3: Display A on the screen and set A to one.

Suppose transactions T1, T2, T3 are allowed to execute concurrently. If A has initial value zero, how many possible correct results are there? Enumerate them.

.....

.....

- 2) Consider the following two transactions, given two bank accounts having a balance A and B.

Transaction T1: Transfer Rs. 100 from A to B

Transaction T2: Find the multiple of A and B.

Create a non-serialisable schedule.

.....

.....

- 3) Add lock and unlock instructions (exclusive or shared) to transactions T1 and T2 so that they observe the serialisable schedule. Make a valid schedule.
-
-

10.5 DEADLOCK AND ITS PREVENTION

As seen earlier, though 2PL protocol handles the problem of serialisability, but it causes some problems also. For example, consider the following two transactions and a schedule involving these transactions:

TA	TB
X_lock A	X_lock A
X_lock B	X_lock B
⋮	⋮
⋮	⋮
Unlock A	Unlock A
Unlock B	Unlock B

Schedule

T1: X_lock A
 T2: X_lock B
 T1: X_lock B
 T2: X_lock A

As is clearly seen, the schedule causes a problem. After T1 has locked A, T2 locks B and then T1 tries to lock B, but unable to do so waits for T2 to unlock B. Similarly, T2 tries to lock A but finds that it is held by T1 which has not yet unlocked it and thus waits for T1 to unlock A. At this stage, neither T1 nor T2 can proceed since both of these transactions are waiting for the other to unlock the locked resource.

Clearly the schedule comes to a halt in its execution. The important thing to be seen here is that both T1 and T2 follow the 2PL, which guarantees serialisability. So whenever the above type of situation arises, we say that a deadlock has occurred, since two transactions are **waiting for a condition that will never occur**.

Also, the deadlock can be described in terms of a directed graph called a “*wait for*” graph, which is maintained by the lock manager of the DBMS. This graph G is defined by the pair (V, E). It consists of a set of vertices/nodes V and a set of edges/arcs E. Each transaction is represented by node and an arc from $T_i \rightarrow T_j$, if T_j holds a lock and T_i is waiting for it. When transaction T_i requests a data item currently being held by transaction T_j then the edge $T_i \rightarrow T_j$ is inserted in the “*wait for*” graph. This edge is removed only when transaction T_j is no longer holding the data item needed by transaction T_i .

A deadlock in the system of transactions occurs, if and only if the wait-for graph contains a cycle. Each transaction involved in the cycle is said to be deadlocked. To detect deadlocks, a periodic check for cycles in graph can be done. For example, the “*wait-for*” for the schedule of transactions TA and TB as above can be made as:

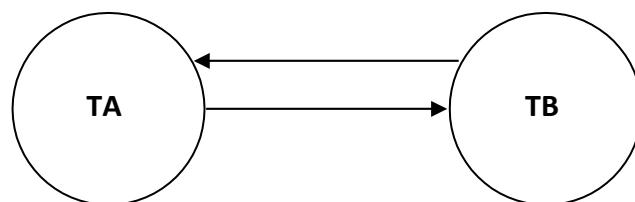


Figure 12: Wait For graph of TA and TB

In the figure above, TA and TB are the two transactions. The two edges are present between nodes TA and TB since each is waiting for the other to unlock a resource held by the other, forming a cycle, causing a deadlock problem. The above case shows a direct cycle. However, in actual situation more than two nodes may be there in a cycle.

A deadlock is thus a situation that can be created because of locks. It causes transactions to wait forever and hence the name deadlock. A deadlock occurs because of the following conditions:

- Mutual exclusion: A resource can be locked in exclusive mode by only one transaction at a time.
- Non-preemptive locking: A data item can only be unlocked by the transaction that locked it. No other transaction can unlock it.
- Partial allocation: A transaction can acquire locks on database in a piecemeal fashion.
- Circular waiting: Transactions lock part of data resources needed and then wait indefinitely to lock the resource currently locked by other transactions.

In order to prevent a deadlock, one has to ensure that at least one of these conditions does not occur.

A deadlock can be prevented, avoided or controlled. Let us discuss a simple method for deadlock prevention.

Deadlock Prevention

One of the simplest approaches for avoiding a deadlock would be to acquire all the locks at the start of the transaction. However, this approach restricts concurrency greatly, also you may lock some of the items that are not updated by that transaction (the transaction may have if conditions). Thus, better prevention algorithm have been evolved to prevent a deadlock having the basic logic: *not to allow circular wait to occur*. This approach rolls back some of the transactions instead of letting them wait.

There exist two such schemes. These are:

“Wait-die” scheme: The scheme is based on non-preventive technique. It is based on a simple rule:

```
If Ti requests a database resource that is held by Tj
    then if Ti has a smaller timestamp than that of Tj
        it is allowed to wait;
    else Ti aborts.
```

A timestamp may loosely be defined as the system generated sequence number that is unique for each transaction. Thus, a smaller timestamp means an older transaction.

For example, assume that three transactions T1, T2 and T3 were generated in that sequence, then if T1 requests for a data item which is currently held by transaction T2, it is allowed to wait as it has a smaller time stamping than that of T1. However, if T3 requests for a data item which is currently held by transaction T2, then T3 is rolled back (die).

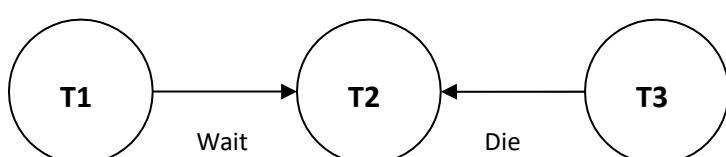


Figure 13: Wait-die Scheme of Deadlock prevention

“Wound-wait” scheme: It is based on a preemptive technique. It is based on a simple rule:

If T_i requests a database resource that is held by T_j
then if T_i has a larger timestamp (T_i is younger) than that of T_j
it is allowed to wait;
else T_j is wounded up by T_i .

For example, assume that three transactions T_1 , T_2 and T_3 were generated in that sequence, then if T_1 requests for a data item which is currently held by transaction T_2 , then T_2 is rolled back and data item is allotted to T_1 as T_1 has a smaller time stamping than that of T_2 . However, if T_3 requests for a data item which is currently held by transaction T_2 , then T_3 is allowed to wait.

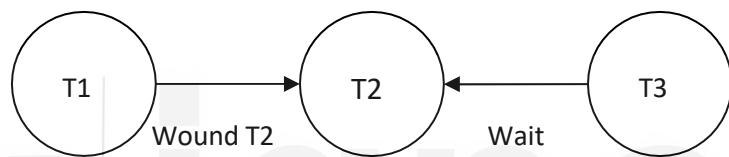


Figure 14: Wound-wait Scheme of Deadlock prevention

It is important to see that whenever any transaction is rolled back, it would not make a starvation condition, that is no transaction gets rolled back repeatedly and is never allowed to make progress. Also both “wait-die” & “wound-wait” scheme avoid starvation. The number of aborts & rollbacks will be higher in wait-die scheme than in the wound-wait scheme. But one major problem with both of these schemes is that these schemes may result in unnecessary rollbacks. You can refer to further readings for more details on deadlock related schemes.

2.6 OPTIMISTIC CONCURRENCY CONTROL

Is locking the only way to prevent concurrency related problems? There exist some other methods too. One such method is called an Optimistic Concurrency control. Let us discuss it in more detail in this section.

The basic logic in optimistic concurrency control is to allow the concurrent transactions to update the data items assuming that the concurrency related problem will not occur. However, we need to reconfirm our view in the validation phase. Therefore, the optimistic concurrency control algorithm has the following phases:

- a) **READ Phase:** A transaction T reads the data items from the database into its private workspace. All the updates of the transaction can only change the local copies of the data in the private workspace.
- b) **VALIDATE Phase:** Checking is performed to confirm whether the read values have changed during the time transaction was updating the local values. This is performed by comparing the current database values to the values that were read in the private workspace. In case, the values have changed the local copies

are thrown away and the transaction aborts.

- c) WRITE Phase: If validation phase is successful the transaction is committed and updates are applied to the database, otherwise the transaction is rolled back.

Some of the terms defined to explain optimistic concurrency controls are:

- write-set(T): all data items that are written by a transaction T
- read-set(T): all data items that are read by a transaction T
- Timestamps: for each transaction T, the start-time and the end time are kept for all the three phases.

More details on this scheme are available in the further readings. But let us show this scheme here with the help of the following examples: Consider the set for transaction T1 and T2.

T1		T2	
Phase	Operation	Phase	Operation
-	-	Read	Reads the read set (T2). Let say variables X and Y and performs updating of local values
Read	Reads the read set (T1) lets say variable X and Y and performs updating of local values	-	-
Validate	Validate the values of (T1)	-	-
-	-	Validate	Validate the values of (T2)
Write	Write the updated values in the database and commit	-	-
-	-	Write	Write the updated values in the database and commit

In this example both T1 and T2 get committed. Please note that Read set of T1 and Read Set of T2 are both disjoint, also the Write sets are also disjoint and thus no concurrency related problem can occur.

T1	T2	T3
Operation	Operation	Operation
Read R(A)	--	--
--	Read R(A)	--
--	--	Read (D)
--	--	Update(D)
--	--	Update (A)
--	--	Validate (D,A) finds OK Write (D,A), COMMIT
--	Validate(A):Unsuccessful Value changed by T3	--
Validate(A):Unsuccessful Value changed by T3	--	--
ABORT T1	--	--
--	Abort T2	--

In this case both T1 and T2 get aborted as they fail during validate phase while only T3 is committed. Optimistic concurrency control performs its checking at

the transaction commits point in a validation phase. The serialization order is determined by the time of transaction validation phase.

Check Your Progress 3

- 1) Draw suitable graph for following locking requests, find whether the transactions are deadlocked or not.

T1: S_lock A	--	--
--	T2: X_lock B	--
--	T2: S_lock C	--
--	--	T3: X_lock C
--	T2: S_lock A	--
T1: S_lock B	--	--
T1: S_lock A	--	--
--	--	T3: S_lock A
All the unlocking requests start from here		

- 2) What is Optimistic Concurrency Control?

10.7 SUMMARY

In this unit you have gone through the concepts of transaction and Concurrency Management. A transaction is a sequence of many actions. Concurrency control deals with ensuring that two or more users do not get into each other's way, i.e., updates of transaction one doesn't affect the updates of other transactions.

Serializability is the generally accepted criterion for correctness for the concurrency control. It is a concept related to concurrent schedules. It determines how to analyse whether any schedule is serialisable or not. Any schedule that produces the same results as a serial schedule is a serialisable schedule.

Concurrency Control is usually done via locking. If a transaction tries to lock a resource already locked by some other transaction, it cannot do so and waits for the resource to be unlocked.

Locks are of two type a) shared lock b) Exclusive lock. Then we move on to a method known as Two Phase Locking (2PL). A system is in a deadlock state if there exist a set of transactions such that every transaction in the set is waiting for another transaction in the set. We can use a deadlock prevention protocol to ensure that the system will never enter a deadlock state.

Finally we have discussed the method Optimistic Concurrency Control, another concurrency management mechanism.

10.8 SOLUTIONS/ANSWERS

Check Your Progress 1

- 1) A transaction is the basic unit of work on a Database management system. It defines the data processing on the database. IT has four basic properties:

- a. Atomicity: transaction is done completely or not at all.
- b. Consistency: Leaves the database in a consistent state
- c. Isolation: Should not see uncommitted values
- d. Durability: Once committed the changes should be reflected.

A transaction can update more than one data values. Some transactions can do writing of data without reading a data value.

A simple transaction example may be: Updating the stock inventory of an item that has been issued. Please create a sample pseudo code for it.

- 2) The basic problems of concurrent transactions are:

- Lost updates: An update is overwritten
- Unrepeatable read: On reading a value later again an inconsistent value is found.
- Dirty read: Reading an uncommitted value
- Inconsistent analysis: Due to reading partially updated value.

No these problems cannot occur if the transactions do not read the same data values. The conflict occurs only if one transaction updates a data value while another is reading or writing the data value.

Commit state is defined as when transaction has done everything correctly and shows the intent of making all the changes as permanent. No, you cannot rollback after commit.

Check Your Progress 2

- 1) There are six possible results, corresponding to six possible serial schedules:

Initially:	A = 0
T1-T2-T3:	A = 1
T1-T3-T2:	A = 2
T2-T1-T3:	A = 1
T2-T3-T1:	A = 2
T3-T1-T2:	A = 4
T3-T2-T1:	A = 3

- 2)

Schedule	T1	T2
Read A	Read A	
A = A - 100	A = A - 100	
Write A	Write A	
Read A		Read A
Read B		Read B
Read B	Read B	
Result = A * B		Result = A * B
Display Result		Display Result
B = B + 100	B = B + 100	
Write B	Write B	

Please make the precedence graph and find out that the schedule is not serialisable.3)

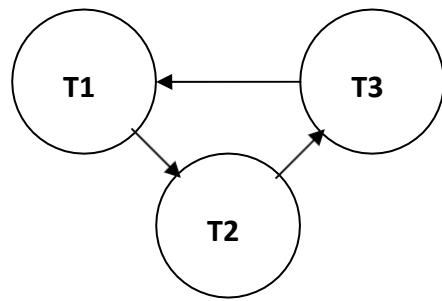
Schedule	T1	T2
Lock A	Lock A	
Lock B	Lock B	
Read A	Read A	
A = A - 100	A = A - 100	
Write A	Write A	
Unlock A	Unlock A	
Lock A		Lock A: Granted
Lock B		Lock B: Waits
Read B	Read B	
B = B + 100	B = B + 100	
Write B	Write B	
Unlock B	Unlock B	
Read A		Read A
Read B		Read B
Result = A * B		Result = A * B
Display Result		Display Result
Unlock A		Unlock A
Unlock B		Unlock B

You must make the schedules using read and exclusive lock and a schedule in strict 2PL.

Check Your Progress 3

- 1) The transaction T1 gets the shared lock on A, T2 gets exclusive lock on B and Shared lock on A, while the transactions T3 gets exclusive lock on C.
 - Now T2 requests for shared lock on C which is exclusively locked by T3, so it cannot be granted. So T2 waits for T3 on item C.
 - T1 now requests for Shared lock on B which is exclusively locked by T2, thus, it waits for T2 for item B. The T1 request for shared lock on C is not processed.
 - Next T3 requests for exclusive lock on A which is share locked by T1, so it cannot be granted. Thus, T3 waits for T1 for item A.

The Wait for graph for the transactions for the given schedule is:



Since there exists a cycle, therefore, the schedule is deadlocked.

- 2) The basic philosophy for optimistic concurrency control is the optimism that nothing will go wrong so let the transaction interleave in any fashion, but to avoid any concurrency related problem we just validate our assumption before we make changes permanent. This is a good model for situations having a low rate of transactions.

IGNOU
THE PEOPLE'S
UNIVERSITY

UNIT 11 DATABASE RECOVERY AND SECURITY

(Adopted from MCS-023 Block-2 Unit-3)

- 11.0 Introduction
 - 11.1 Objectives
 - 11.2 What is Recovery?
 - 11.2.1 Kinds of failures
 - 11.2.2 Failure controlling methods
 - 11.2.3 Database errors
 - 11.3 Recovery Techniques
 - 11.4 Security & Integrity
 - 11.4.1 Relationship between Security and Integrity
 - 11.4.2 Difference between Operating System and Database Security
 - 11.5 Authorisation
 - 11.6 Summary
 - 11.7 Solutions/Answers
-

11.0 INTRODUCTION

In the previous unit of this block, you have gone through the concepts of transactions and Concurrency management. In this unit we will introduce two important issues relating to database management systems.

A computer system suffers from different types of failures. A DBMS controls very critical data of an organisation and therefore must be reliable. However, the reliability of the database system is linked to the reliability of the computer system on which it runs. In this unit we will discuss recovery of the data contained in a database system following failure of various types and present the different approaches to database recovery. The types of failures that the computer system is likely to be subjected to include failures of components or subsystems, software failures, power outages, accidents, unforeseen situations and natural or man-made disasters. Database recovery techniques are methods of making the database consistent till the last possible consistent state. The aim of recovery scheme is to allow database operations to be resumed after a failure with minimum loss of information at an economically justifiable cost.

The second main issue that is being discussed in this unit is Database security. “Database security” is protection of the information contained in the database against unauthorised access, modification or destruction. The first condition for security is to have Database integrity. “Database integrity” is the mechanism that is applied to ensure that the data in the database is consistent.

Let us discuss all these terms in more detail in this unit.

11.1 OBJECTIVES

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

- describe the terms RECOVERY and INTEGRITY;
- describe Recovery Techniques;

- define Error and Error detection techniques, and
- describe types of Authorisation

11.2 WHAT IS RECOVERY?

During the life of a transaction, that is, after the start of a transaction but before the transaction commits, several changes may be made in a database state. The database during such a state is in an inconsistent state. What happens when a failure occurs at this stage? Let us explain this with the help of an example:

Assume that a transaction transfers Rs.2000/- from A's account to B's account. For simplicity we are not showing any error checking in the transaction. The transaction may be written as:

Transaction T1:

```

READ A
A = A - 2000
WRITE A
Failure →
READ B
B = B + 2000
WRITE B
COMMIT

```

What would happen if the transaction fails after account A has been written back to database? As far as the holder of account A is concerned s/he has transferred the money but that has never been received by account holder B.

Why did this problem occur? Because although a transaction is considered to be atomic, yet it has a life cycle during which the database gets into an inconsistent state and failure has occurred at that stage.

What is the solution? In this case where the transaction has not yet committed the changes made by it, the partial updates need to be undone.

How can we do that? By remembering information about a transaction such as when did it start, what items it updated etc. All such details are kept in a log file. We will study about log in Section 3.3. But first let us analyse the reasons of failure.

Failures and Recovery

In practice several things might happen to prevent a transaction from completing. Recovery techniques are used to bring database, which does not satisfy consistency requirements, into a consistent state. If a transaction completes normally and commits then all the changes made by the transaction on the database are permanently registered in the database. They should not be lost (please recollect the durability property of transactions given in Unit 2). But, if a transaction does not complete normally and terminates abnormally then all the changes made by it should be discarded. An abnormal termination of a transaction may be due to several reasons, including:

- a) user may decide to abort the transaction issued by him/ her
- b) there might be a deadlock in the system

- c) there might be a system failure.

The recovery mechanisms must ensure that a consistent state of database can be restored under all circumstances. In case of transaction abort or deadlock, the system remains in control and can deal with the failure but in case of a system failure the system loses control because the computer itself has failed. Will the results of such failure be catastrophic? A database contains a huge amount of useful information and any system failure should be recognised on the restart of the system. The DBMS should recover from any such failures. Let us first discuss the kinds of failure for identifying how to recover.

11.2.1 Kinds of Failures

The kinds of failures that a transaction program during its execution can encounter are:

- 1) **Software failures:** In such cases, a software error abruptly stops the execution of the current transaction (or all transactions), thus leading to losing the state of program execution and the state/ contents of the buffers. But what is a buffer? A buffer is the portion of RAM that stores the partial contents of database that is currently needed by the transaction. The software failures can further be subdivided as:
 - a) Statement or application program failure
 - b) Failure due to viruses
 - c) DBMS software failure
 - d) Operating system failure

A statement of program may cause abnormal termination if it does not execute completely. This happens if during the execution of a statement, an integrity constraint gets violated. This leads to abnormal termination of the transaction due to which any prior updates made by the transaction may still get reflected in the database leaving it in an inconsistent state.

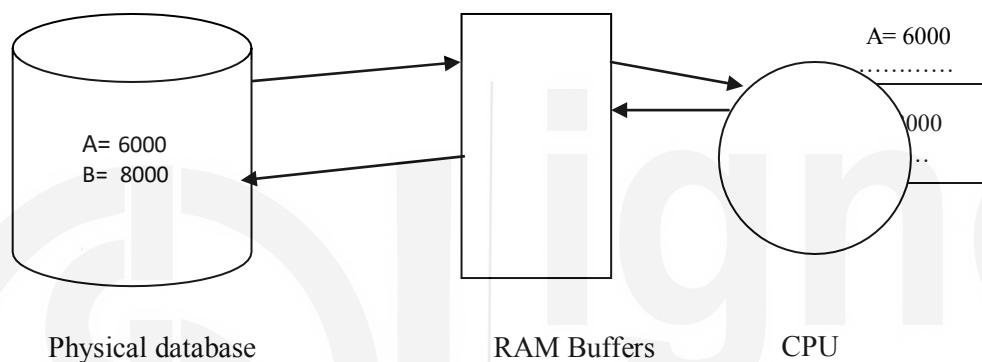
A failure of transaction can occur if some code in a transaction program leads to its abnormal termination. For example, a transaction can go into an infinite loop. In such a case the only way to break the loop is to abort the program. Similarly, the failure can be traced to the operating system or DBMS and transactions are aborted abruptly. Thus part of the transaction that was executed before abort may cause some updates in database, and hence the database is updated only partially which leads to an inconsistent state of database.

- 2) **Hardware failure:** Hardware failures are those failures when some hardware chip or disk fails. This may result in loss of data. One such problem can be that a disk gets damaged and cannot be read any more. This may be due to many reasons. For example, a voltage fluctuation in the power supply to the computer makes it go off or some bad sectors may come on disk or there is a disk crash. In all these cases, the database gets into an inconsistent state.
- 3) **External failure:** A failure can also result due to an external cause, such as fire, earthquakes, floods, etc. The database must be duly backed up to avoid problems occurring due to such failures.

In practice software failures are more common than hardware failures. Fortunately, recovery from software failures is much quicker.

The basic unit of recovery is a transaction. But, how are the transactions handled during recovery? Consider that some transactions are deadlocked, then at least one of

these transactions has to be restarted to break the deadlock and thus the partial updates made by such restarted program in the database need to be **undone** so that the database does not go to an inconsistent state. So the transaction may have to be rolled back which makes sure that the transaction does not bring the database to an inconsistent state. This is one form of recovery. Let us consider a case when a transaction has committed but the changes made by the transaction have not been communicated to permanently stored physical database. A software failure now occurs and the contents of the CPU/ RAM are lost. This leaves the database in an inconsistent state. Such failure requires that on restarting the system the database be brought to a consistent state using **redo** operation. The redo operation makes the changes made by the transaction again to bring the system to a consistent state. The database system then can be made available to the users. The point to be noted here is that the database updates are performed in the buffer in the memory. *Figure 1* shows some cases of undo and redo. You can create more such cases.



	Physical Database	RAM	Activity
Case 1	A=6000 B=8000	A=4000 B=8000	Transaction T1 has just changed the value in RAM. Now it aborts, value in RAM is lost. No problem. But we are not sure that the physical database has been written back, so must undo.
Case 2	A=4000 B=8000	A=4000 B=8000	The value in physical database has got updated due to buffer management, now the transaction aborts. The transaction must be undone.
Case 3	A=6000 B=8000	A=4000 B=10000 Commit	The value B in physical database has not got updated due to buffer management. In case of failure now when the transaction has committed. The changes of transaction must be redone to ensure transfer of correct values to physical database.

Figure 1: Database Updates And Recovery

11.2.2 Failure Controlling Methods

Failures can be handled using different recovery techniques that are discussed later in

the unit. But the first question is do we really need recovery techniques as a failure control mechanism? The recovery techniques are somewhat expensive both in terms of time and in memory space for small systems. In such a case it is more beneficial to better avoid the failure by some checks instead of deploying recovery technique to make database consistent. Also, recovery from failure involves manpower that can be used in some other productive work if failure can be avoided. It is, therefore, important to find out some general precautions that help in controlling failure. Some of these precautions may be:

- having a regulated power supply.
- having a better secondary storage system such as RAID.
- taking periodic backup of database states and keeping track of transactions after each recorded state.
- properly testing the transaction programs prior to use.
- setting important integrity checks in the databases as well as user interfaces etc.

However, it may be noted that if the database system is critical it must use a DBMS that is suitably equipped with recovery procedures.

11.2.3 Database Errors

An error is said to have occurred if the execution of a command to manipulate the database cannot be successfully completed either due to inconsistent data or due to state of program. For example, there may be a command in program to store data in database. On the execution of command, it is found that there is no space/place in database to accommodate that additional data. Then it can be said that an error has occurred. This error is due to the physical state of database storage.

Broadly errors are classified into the following categories:

- 1) **User error:** This includes errors in the program (e.g., Logical errors) as well as errors made by online users of database. These types of errors can be avoided by applying some check conditions in programs or by limiting the access rights of online users e.g., read only. So only updating or insertion operation require appropriate check routines that perform appropriate checks on the data being entered or modified. In case of an error, some prompts can be passed to user to enable him/her to correct that error.
- 2) **Consistency error:** These errors occur due to the inconsistent state of database caused may be due to wrong execution of commands or in case of a transaction abort. To overcome these errors the database system should include routines that check for the consistency of data entered in the database.
- 3) **System error:** These include errors in database system or the OS, e.g., deadlocks. Such errors are fairly hard to detect and require reprogramming the erroneous components of the system software.

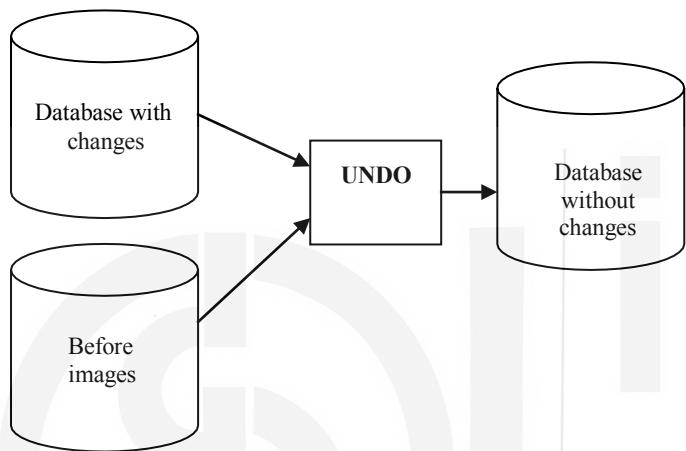
Database errors can result from failure or can cause failure and thus will require recovery. However, one of the main tasks of database system designers is to make sure that errors minimised. These concepts are also related to database integrity and have also been discussed in a later section.

11.3 RECOVERY TECHNIQUES

After going through the types of failures and database errors, let us discuss how to recover from the failures. Recovery can be done using/restoring the previous consistent state (backward recovery) or by moving forward to the next consistent state as per the committed transactions (forward recovery) recovery. Please note that a system can recover from software and hardware failures using the forward and backward recovery only if the system log is intact. What is system log? We will discuss it in more detail, but first let us define forward and backward recovery.

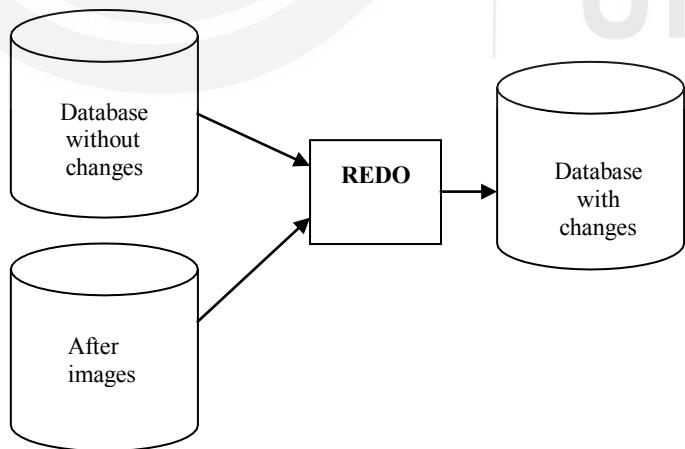
1) Backward Recovery (UNDO)

In this scheme the uncommitted changes made by a transaction to a database are undone. Instead the system is reset to the previous consistent state of database that is free from any errors.



2) Forward Recovery (Redo)

In this scheme the committed changes made by a transaction are reapplied to an earlier copy of the database.



In simpler words, when a particular error in a system is detected, the recovery system makes an accurate assessment of the state of the system and then makes the appropriate adjustment based on the anticipated results - had the system been error free.

One thing to be noted is that the Undo and Redo operations must be idempotent, i.e., executing them several times must be equivalent to executing them once. This

characteristic is required to guarantee correct behaviour of database even if a failure occurs during the recovery process.

Depending on the above discussed recovery scheme, several types of recovery methods have been used. However, we define the most important recovery schemes used in most of the commercial DBMSs

Log based recovery

Let us first define the term transaction log in the context of DBMS. A transaction log is a record in DBMS that keeps track of all the transactions of a database system that update any data values in the database. A log contains the following information about a transaction:

- A transaction begin marker
- The transaction identification: The transaction id, terminal id or user id etc.
- The operations being performed by the transaction such as update, delete, insert.
- The data items or objects that are affected by the transaction including name of the table, row number and column number.
- The before or previous values (also called UNDO values) and after or changed values (also called REDO values) of the data items that have been updated.
- A pointer to the next transaction log record, if needed.
- The COMMIT marker of the transaction.

In a database system several transactions run concurrently. When a transaction commits, the data buffers used by it need not be written back to the physical database stored on the secondary storage as these buffers may be used by several other transactions that have not yet committed. On the other hand, some of the data buffers that may have updates by several uncommitted transactions might be forced back to the physical database, as they are no longer being used by the database. So the transaction log helps in remembering which transaction did which changes. Thus the system knows exactly how to separate the changes made by transactions that have already committed from those changes that are made by the transactions that did not yet commit. Any operation such as begin transaction, insert /delete/update and end transaction (commit), adds information to the log containing the transaction identifier and enough information to undo or redo the changes.

But how do we recover using log? Let us demonstrate this with the help of an example having three concurrent transactions that are active on ACCOUNTS table as:

Transaction T1	Transaction T2	Transaction T3
Read X	Read A	Read Z
Subtract 100	Add 200	Subtract 500
Write X	Write A	Write Z
Read Y		
Add 100		
Write Y		

Figure 2: The sample transactions

Assume that these transactions have the following log file (hypothetical) at a point:

Transaction Begin Marker	Transaction Id	Operation on ACCOUNTS table	UNDO values (assumed)	REDO values	Transaction Commit Marker
Y	T1	Sub on X Add on Y	500 800	400 Not done yet	N
Y	T2	Add on A	1000	1200	N
Y	T3	Sub on Z	900	400	Y

Figure 3: A sample (hypothetical) Transaction log

Now assume at this point of time a failure occurs, then how the recovery of the database will be done on restart.

Values	Initial	Just before the failure	Operation Required for recovery	Recovered Database Values
X	500	400 (assuming update has been done in physical database also)	UNDO	500
Y	800	800	UNDO	800
A	1000	1000 (assuming update has not been done in physical database)	UNDO	1000
Z	900	900 (assuming update has not been done in physical database)	REDO	400

Figure 4: The database recovery

The selection of REDO or UNDO for a transaction for the recovery is done on the basis of the state of the transactions. This state is determined in two steps:

- Look into the log file and find all the transactions that have started. For example, in *Figure 3*, transactions T1, T2 and T3 are candidates for recovery.
- Find those transactions that have committed. REDO these transactions. All other transactions have not committed so they should be rolled back, so UNDO them. For example, in *Figure 3* UNDO will be performed on T1 and T2; and REDO will be performed on T3.

Please note that in *Figure 4* some of the values may not have yet been communicated to database, yet we need to perform UNDO as we are not sure what values have been written back to the database.

But how will the system recover? Once the recovery operation has been specified, the system just takes the required REDO or UNDO values from the transaction log and changes the inconsistent state of database to a consistent state. (Please refer to *Figure 3* and *Figure 4*).

Let us consider several transactions with their respective start & end (commit) times as shown in *Figure 5*.

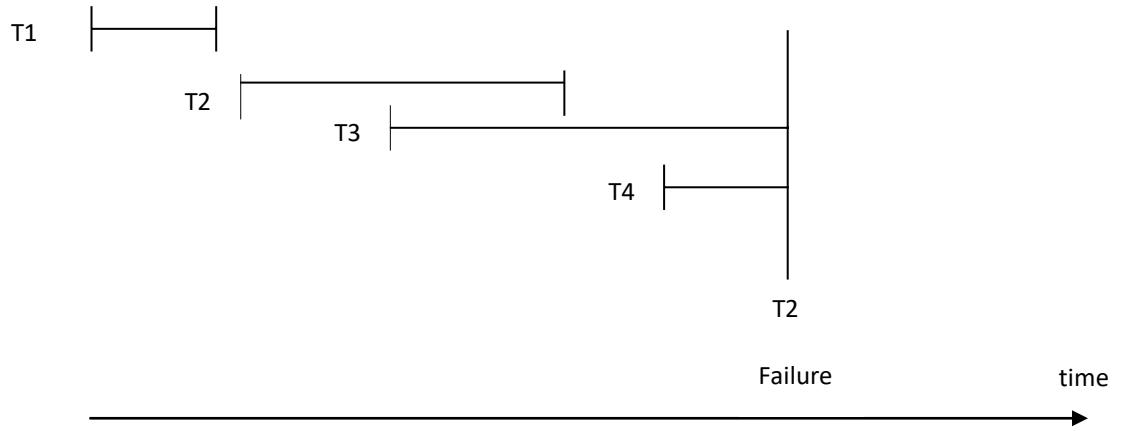


Figure 5: Execution of Concurrent Transactions

In the figure above four transactions are executing concurrently, on encountering a failure at time t_2 , the transactions T_1 and T_2 are to be REDONE and T_3 and T_4 will be UNDONE. But consider a system that has thousands of parallel transactions then all those transactions that have been committed may have to be redone and all uncommitted transactions need to be undone. That is not a very good choice as it requires redoing of even those transactions that might have been committed even hours earlier. So can we improve on this situation? Yes, we can take checkpoints. Figure 6 shows a checkpoint mechanism:

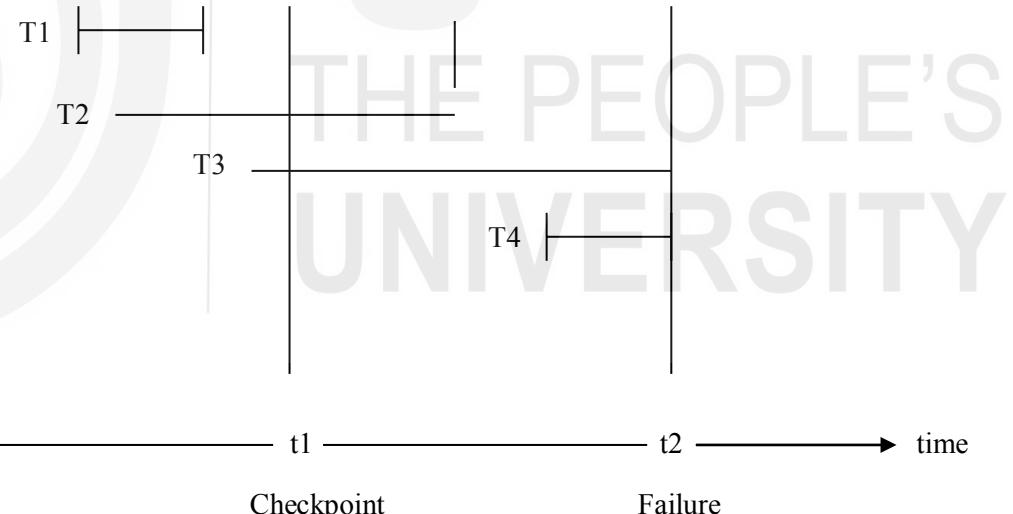


Figure 6: Checkpoint In Transaction Execution

A checkpoint is taken at time t_1 and a failure occurs at time t_2 . Checkpoint transfers all the committed changes to database and all the system logs to stable storage (it is defined as the storage that would not be lost). At restart time after the failure the stable check pointed state is restored. Thus, we need to only REDO or UNDO those transactions that have completed or started after the checkpoint has been taken. The only possible disadvantages of this scheme may be that during the time of taking the checkpoint the database would not be available and some of the uncommitted values may be put in the physical database. To overcome the first problem the checkpoints should be taken at times when system load is low. To avoid the second problem some systems allow some time to the ongoing transactions to complete without restarting

new transactions.

In the case of *Figure 6* the recovery from failure at t_2 will be as follows:

- The transaction T1 will not be considered for recovery as the changes made by it have already been committed and transferred to physical database at checkpoint t_1 .
- The transaction T2 since it has not committed till the checkpoint t_1 but have committed before t_2 , will be REDONE.
- T3 must be UNDONE as the changes made by it before checkpoint (we do not know for sure if any such changes were made prior to checkpoint) must have been communicated to the physical database. T3 must be restarted with a new name.
- T4 started after the checkpoint, and if we strictly follow the scheme in which the buffers are written back only on the checkpoint, then nothing needs to be done except restarting the transaction T4 with a new name.

The restart of a transaction requires the log to keep information of the new name of the transaction and maybe give higher priority to this newer transaction.

But one question is still unanswered that is during a failure we lose database information in RAM buffers, we may also lose log as it may also be stored in RAM buffers, so how does log ensure recovery?

The answer to this question lies in the fact that for storing transaction log we follow a **Write Ahead Log Protocol**. As per this protocol, the transaction logs are written to stable storage before any item is updated. Or more specifically it can be stated as; **the undo portion of log is written to stable storage prior to any updates and redo portion of log is written to stable storage prior to commit.**

Log based recovery scheme can be used for any kind of failure provided you have stored the most recent checkpoint state and most recent log as per write ahead log protocol into the stable storage. Stable storage from the viewpoint of external failure requires more than one copy of such data at more than one location. You can refer to the further readings for more details on recovery and its techniques.

Check Your Progress 1

1) What is the need of recovery? What is it the basic unit of recovery?

.....
.....

2) What is a checkpoint? Why is it needed? How does a checkpoint help in recovery?

.....
.....

3) What are the properties that should be taken into consideration while selecting recovery techniques?

.....
.....

11.4 SECURITY AND INTEGRITY

After going through the concepts of database recovery in the previous section, let us now deal with an important concept that helps in minimizing consistency errors in database systems. These are the concepts of database security and integrity.

Information security is the protection of information against unauthorised disclosure, alteration or destruction. Database security is the protection of information that is maintained in a database. It deals with ensuring only the “right people” get the right to access the “right data”. By right people we mean those people who have the right to access/update the data that they are requesting to access/update with the database. This should also ensure the confidentiality of the data. For example, in an educational institution, information about a student’s grades should be made available only to that student, whereas only the university authorities should be able to update that information. Similarly, personal information of the employees should be accessible only to the authorities concerned and not to everyone. Another example can be the medical records of patients in a hospital. These should be accessible only to health care officials.

Thus, one of the concepts of database security is primarily a specification of access rules about who has what type of access to what information. This is also known as the problem of Authorisation. These access rules are defined at the time database is defined. The person who writes access rules is called the authoriser. The process of ensuring that information and other protected objects are accessed only in authorised ways is called access control. There may be other forms of security relating to physical, operating system, communication aspects of databases. However, in this unit, we will confine ourselves mainly to authorisation and access control using simple commands.

The term integrity is also applied to data and to the mechanism that helps to ensure its consistency. Integrity refers to the avoidance of accidental loss of consistency.

Protection of database contents from unauthorised access includes legal and ethical issues, organization policies as well as database management policies. To protect database several levels of security measures are maintained:

- 1) **Physical:** The site or sites containing the computer system must be physically secured against illegal entry of unauthorised persons.
- 2) **Human:** An Authorisation is given to a user to reduce the chance of any information leakage and unwanted manipulations.
- 3) **Operating System:** Even though foolproof security measures are taken to secure database systems, weakness in the operating system security may serve as a means of unauthorised access to the database.
- 4) **Network:** Since databases allow distributed or remote access through terminals or network, software level security within the network software is an important issue.
- 5) **Database system:** The data items in a database need a fine level of access control. For example, a user may only be allowed to read a data item and is allowed to issue queries but would not be allowed to deliberately modify the data. It is the responsibility of the database system to ensure that these access restrictions are not violated. Creating database views as discussed in Unit 1 Section 1.6.1 of this block is a very useful mechanism of ensuring database security.

To ensure database security requires implementation of security at all the levels as

above. The Database Administrator (DBA) is responsible for implementing the database security policies in a database system. The organisation or data owners create these policies. DBA creates or cancels the user accounts assigning appropriate security rights to user accounts including power of granting and revoking certain privileges further to other users.

11.4.1 Relationship between Security and Integrity

Database security usually refers to access, whereas database integrity refers to avoidance of accidental loss of consistency. But generally, the turning point or the dividing line between security and integrity is not always clear. *Figure 7* shows the relationship between data security and integrity.



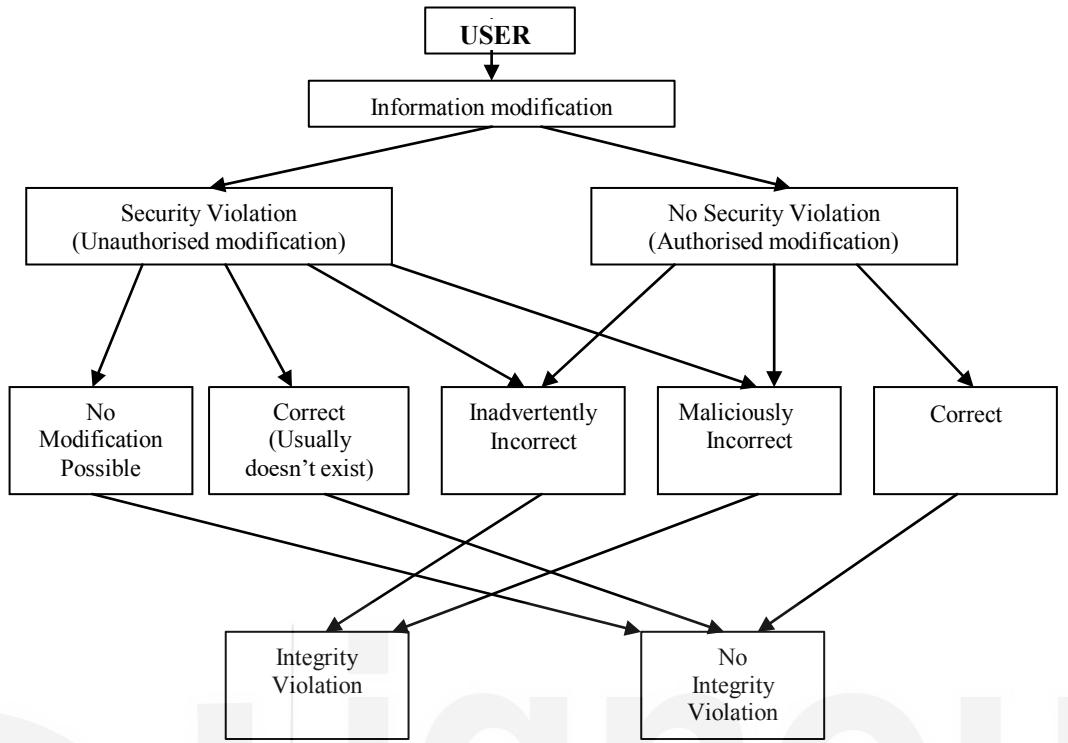


Figure 7: Relationship between security and Integrity

11.4.2 Difference between Operating System and Database Security

Security within the operating system can be implemented at several levels ranging from passwords for access to system, to the isolation of concurrent executing processes with the system. However, there are a few differences between security measures taken at operating system level as compared to those of database system. These are:

- Database system protects more objects, as the data is persistent in nature. Also database security is concerned with different levels of granularity such as file, tuple, an attribute value or an index. Operating system security is primarily concerned with the management and use of resources.
- Database system objects can be complex logical structures such as views, a number of which can map to the same physical data objects. Moreover different architectural levels viz. internal, conceptual and external levels, have different security requirements. Thus, database security is concerned with the semantics – meaning of data, as well as with its physical representation. Operating system can provide security by not allowing any operation to be performed on the database unless the user is authorized for the operation concerned.

Figure 8 shows the architecture of a database security subsystem that can be found in any commercial DBMS.

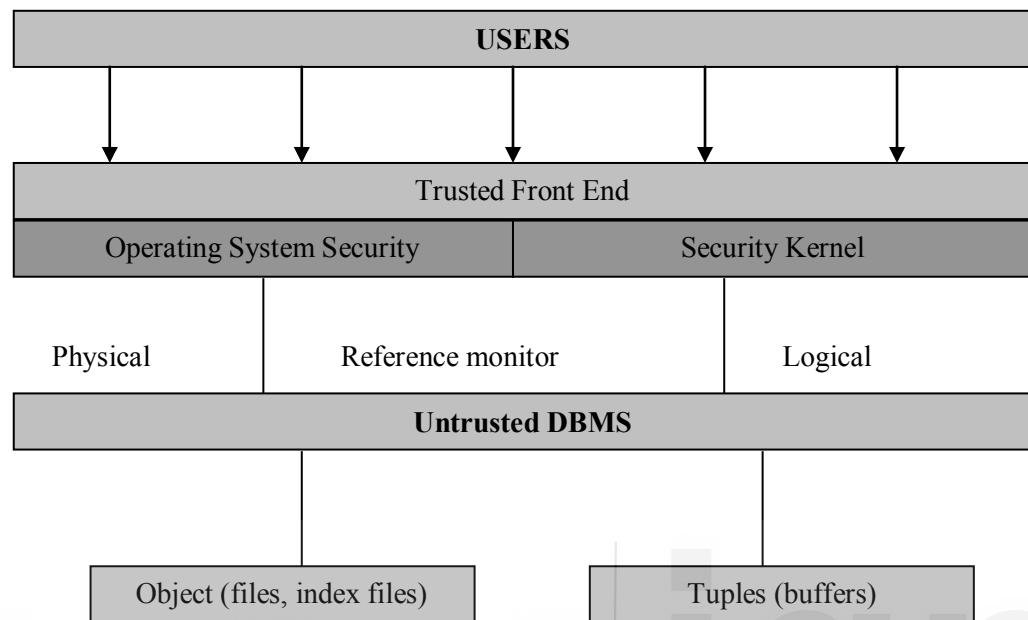


Figure 8: Database Security subsystem

11.5 AUTHORISATION

Authorisation is the culmination of the administrative policies of the organisation. As the name specifies, authorisation is a set of rules that can be used to determine which user has what type of access to which portion of the database. The following forms of authorisation are permitted on database items:

- 1) **READ:** it allows reading of data object, but not modification, deletion or insertion of data object.
- 2) **INSERT:** allows insertion of new data, but not the modification of existing data, e.g., insertion of tuple in a relation.
- 3) **UPDATE:** allows modification of data, but not its deletion. But data items like primary-key attributes may not be modified.
- 4) **DELETE:** allows deletion of data only.

A user may be assigned all, none or a combination of these types of Authorisation, which are broadly called access authorisations.

In addition to these manipulation operations, a user may be granted control operations like

- 1) Add: allows adding new objects such as new relations.
- 2) Drop: allows the deletion of relations in a database.
- 3) Alter: allows addition of new attributes in a relations or deletion of existing

attributes from the database.

- 4) Propagate Access Control: this is an additional right that allows a user to propagate the access control or access right which s/he already has to some other, i.e., if user A has access right R over a relation S, then if s/he has propagate access control, s/he can propagate her/his access right R over relation S to another user B either fully or part of it. In SQL you can use WITH GRANT OPTION for this right.

You must refer to Section 1.5 of Unit 1 of this block for the SQL commands relating to data and user control.

The ultimate form of authority is given to the database administrator. He is the one who may authorize new users, restructure the database and so on. The process of Authorisation involves supplying information known only to the person the user has claimed to be in the identification procedure.

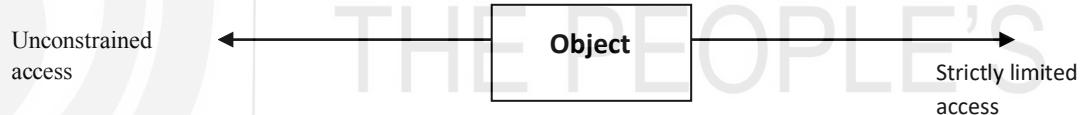
A basic model of Database Access Control

Models of database access control have grown out of earlier work on protection in operating systems. Let us discuss one simple model with the help of the following example:

Security problem: Consider the relation:

Employee (Empno, Name, Address, Deptno, Salary, Assessment)

Assuming there are two users: Personnel manager and general user . What access rights may be granted to each user? One extreme possibility is to grant an unconstrained access or to have a limited access.



One of the most influential protection models was developed by Lampson and extended by Graham and Denning. This model has 3 components:

- 1) A set of objects: where objects are those entities to which access must be controlled.
- 2) A set of subjects: where subjects are entities that request access to objects.
- 3) A set of all access rules: which can be thought of as forming an access (often referred to as authorisation matrix).

Let us create a sample authorisation matrix for the given relation:

Object \ Subject	Empno	Name	Address	Deptno	Salary	Assessment
Personnel Manager	Read	All	All	All	All	All
General User	Read	Read	Read	Read	Not accessible	Not accessible

As the above matrix shows, Personnel Manager and general user are the two subjects. Objects of database are Empno, Name, Address, Deptno, Salary and Assessment. As per the access matrix, Personnel Manager can perform any operation on the database of an employee except for updating the Empno that may be self-generated and once given can never be changed. The general user can only read the data but cannot update, delete or insert the data into the database. Also the information about the salary and assessment of the employee is not accessible to the general user.

In summary, it can be said that the basic access matrix is the representation of basic access rules. These rules may be implemented using a view on which various access rights may be given to the users.

Check Your Progress 2

- 1) What are the different types of data manipulation operations and control operations?

.....
.....

- 2) What is the main difference between data security and data integrity?

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

- 3) What are the various aspects of security problem?

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

- 4) Name the 3 main components of Database Access Control Model?

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

11.6 SUMMARY

In this unit we have discussed the recovery of the data contained in a database system after failures of various types. The types of failures that the computer system is likely to be subject to include that of components or subsystems, software failures, power

outages, accidents, unforeseen situations, and natural or man-made disasters. Database recovery techniques are methods of making the database fault tolerant. The aim of the recovery scheme is to allow database operations to be resumed after a failure with the minimum loss of information and at an economically justifiable cost.

The basic technique to implement database recovery is by using data redundancy in the form of logs, and archival copies of the database. Checkpoint helps the process of recovery.

Security and integrity concepts are crucial since modifications in a database require the replacement of the old values. The DBMS security mechanism restricts users to only those pieces of data that are required for the functions they perform. Security mechanisms restrict the type of actions that these users can perform on the data that is accessible to them. The data must be protected from accidental or intentional (malicious) corruption or destruction. In addition, there is a privacy dimension to data security and integrity.

Security constraints guard against accidental or malicious tampering with data; integrity constraints ensure that any properly authorized access, alteration, deletion, or insertion of the data in the database does not change the consistency and validity of the data. Database integrity involves the correctness of data and this correctness has to be preserved in the presence of concurrent operations, error in the user's operation and application programs, and failures in hardware and software.

11.7 SOLUTIONS/ANSWERS

Check Your Progress 1

- 1) Recovery is needed to take care of the failures that may be due to software, hardware and external causes. The aim of the recovery scheme is to allow database operations to be resumed after a failure with the minimum loss of information and at an economically justifiable cost. One of the common techniques is log-based recovery. A transaction is the basic unit of recovery.
- 2) A checkpoint is a point when all the database updates and logs are written to stable storage. A checkpoint ensures that not all the transactions need to be REDONE or UNDONE. Thus, it helps in faster recovery from failure. You should create a sample example, for checkpoint having a sample transaction log.
- 3) The following properties should be taken into consideration:
 - Loss of data should be minimal
 - Recovery should be quick
 - Recovery should be automatic
 - Affect small portion of database.

Check Your Progress 2

- 1) Data manipulation operations are:
 - Read

- Insert
- Delete
- Update

Data control operations are:

- Add
- Drop
- Alter
- Propagate access control

2) Data security is the protection of information that is maintained in database against unauthorised access, modification or destruction. Data integrity is the mechanism that is applied to ensure that data in the database is correct and consistent.

- 3)
- Legal, social and ethical aspects
 - Physical controls
 - Policy questions
 - Operational problems
 - Hardware control
 - Operating system security
 - Database administration concerns
- 4) The three components are:
- Objects
 - Subjects
 - Access rights

ignou
THE PEOPLE'S
UNIVERSITY

UNIT 12 QUERY PROCESSING AND EVALUATION

(Adopted from MCS-043 Block-2 Unit-1)

Structure	Page Nos.
12.0 Introduction	
12.1 Objectives	
12.2 Query Processing : An Introduction	
12.2.1 Optimisation	
12.2.2 Measure of Query Cost	
12.3 Select Operation	
12.4 Sorting	
12.5 Join Operation	
12.5.1 Nested-Loop Join	
12.5.2 Block Nested-Loop Join	
12.5.3 Indexed Nested-Loop Join	
12.5.4 Merge-Join	
12.5.5 Hash-Join	
12.5.6 Complex Join	
12.6 Other Operations	
12.7 Representation and Evaluation of Query Expressions	
12.8 Creation of Query Evaluation Plans	
12.8.1 Transformation of Relational Expressions	
12.8.2 Query Evaluation Plans	
12.8.3 Choice of Evaluation Plan	
12.8.4 Cost Based Optimisation	
12.8.5 Storage and Query Optimisation	
12.9 View And Query Processing	
12.9.1 Materialised View	
12.9.2 Materialised Views and Query Optimisation	
12.10 Summary	
12.11 Solutions/Answers	

12.0 INTRODUCTION

The Query Language – SQL is one of the main reasons of success of RDBMS. A user just needs to specify the query in SQL that is close to the English language and does not need to say how such query is to be evaluated. However, a query needs to be evaluated efficiently by the DBMS. But how is a query-evaluated efficiently? This unit attempts to answer this question. The unit covers the basic principles of query evaluation, the cost of query evaluation, the evaluation of join queries, etc. in detail. It also provides information about query evaluation plans and the role of storage in query evaluation and optimisation. This unit thus, introduces you to the complexity of query evaluation in DBMS.

12.1 OBJECTIVES

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

- measure query cost;

- define algorithms for individual relational algebra operations;
 - create and modify query expression;
 - define evaluation plan choices, and
 - define query processing using views.

12.2 QUERY PROCESSING: AN INTRODUCTION

Before defining the measures of query cost, let us begin by defining query processing. Let us take a look at the *Figure 1*.

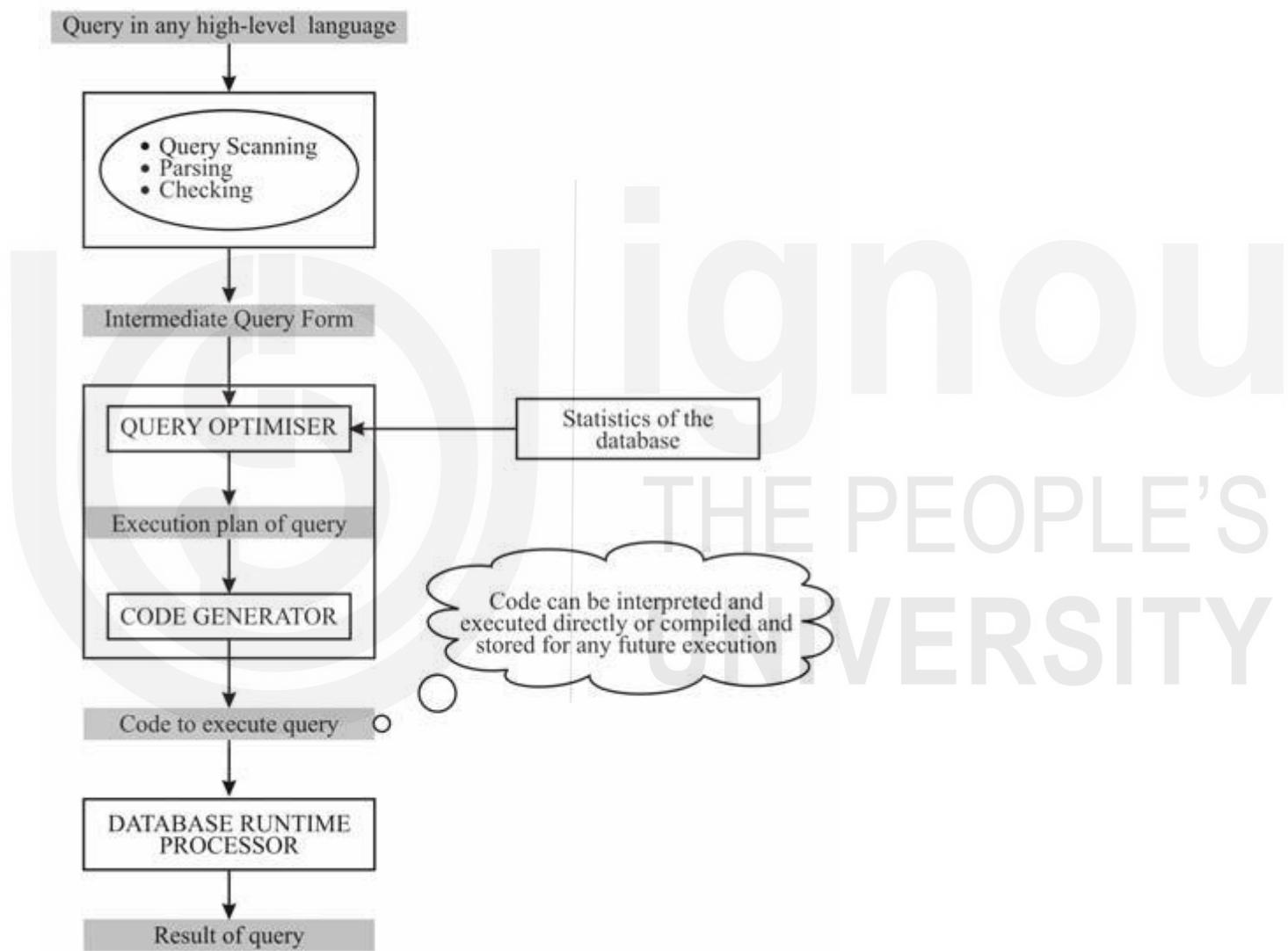


Figure 1: Query processing

In the first step Scanning, Parsing, and Validating is done to translate the query into its internal form. This is then further translated into relational algebra (an intermediate query form). Parser checks syntax and verifies relations. The query then is optimised with a query plan, which then is compiled into a code that can be executed by the database runtime processor.

We can define query evaluation as the query-execution engine taking a query-

evaluation plan, executing that plan, and returning the answers to the query. The query processing involves the study of the following concepts:

- how to measure query costs?
- algorithms for evaluating relational algebraic operations.
- how to evaluate a complete expression using algorithms on individual operations?

12.2.1 Optimisation

A relational algebra expression may have many equivalent expressions. For example, $\sigma_{(\text{salary} < 5000)} (\pi_{\text{salary}} (\text{EMP}))$ is equivalent to $\pi_{\text{salary}} (\sigma_{\text{salary} < 5000} (\text{EMP}))$.

Each relational algebraic operation can be evaluated using one of the several different algorithms. Correspondingly, a relational-algebraic expression can be evaluated in many ways.

An expression that specifies detailed evaluation strategy is known as evaluation-plan, for example, you can use an index on *salary* to find employees with $\text{salary} < 5000$ or we can perform complete relation scan and discard employees with $\text{salary} \geq 5000$. The basis of selection of any of the scheme will be the cost.

Query Optimisation: Amongst all equivalent plans choose the one with the lowest cost. Cost is estimated using statistical information from the database catalogue, for example, number of tuples in each relation, size of tuples, etc.

Thus, in query optimisation we find an evaluation plan with the lowest cost. The cost estimation is made on the basis of heuristic rules.

12.2.2 Measure of Query Cost

Cost is generally measured as total elapsed time for answering the query. There are many factors that contribute to time cost. These are *disk accesses*, CPU time, or even network *communication*.

Typically disk access is the predominant cost as disk transfer is a very slow event and is also relatively easy to estimate. It is measured by taking into account the following activities:

Number of seeks	×	average-seek-cost
Number of blocks read	×	average-block-read-cost
Number of blocks written	×	average-block-written-cost.

Please note that the cost for writing a block is higher than the cost for reading a block. This is due to the fact that the data is read back after being written to ensure that the write was successful. However, for the sake of simplicity we will just use *number of block transfers from disk as the cost measure*. We will also ignore the difference in cost between sequential and random I/O, CPU and communication costs. The I/O cost **depends** on the search criteria i.e., point/range query on an ordering/other fields and the file structures: heap, sorted, hashed. It is also dependent on the use of indices such as primary, clustering, secondary, B+ tree, multilevel, etc. There are other cost factors also, these may include buffering, disk placement, materialisation, overflow / free space management etc.

In the subsequent section, let us try to find the cost of various operations.

12.3 SELECT OPERATION

The selection operation can be performed in a number of ways. Let us discuss the algorithms and the related cost of performing selection operations.

File scan: These are the search algorithms that locate and retrieve records that fulfil a selection condition in a file. The following are the two basic file scan algorithms for selection operation:

- 1) *Linear search:* This algorithm scans each file block and tests all records to see whether they satisfy the selection condition.

The cost of this algorithm (in terms of block transfer) is defined as:

$$\text{Cost of searching records satisfying a condition} = \text{Number of blocks in a database} = N_b.$$

$$\text{Cost for searching a key attribute value} = \text{Average number of block transfer for locating the value (on an average, half of the file needs to be traversed)} \text{ so it is } = N_b/2.$$

Linear search can be applied regardless of selection condition or ordering of records in the file, or availability of indices.

- 2) *Binary search:* It is applicable when the selection is an equality comparison on the attribute on which file is ordered. Assume that the blocks of a relation are stored continuously then, cost can be estimated as:

Cost = Cost of locating the first tuple by a binary search on the blocks + sequence of other blocks that continue to satisfy the condition.

$$= \lceil \log_2 (N_b) \rceil + \frac{\text{average number of tuples with the same value}}{\text{Blocking factor (Number of tuples in a block) of the relation}}$$

These two values may be calculated from the statistics of the database.

Index scan: Search algorithms that use an index are restricted because the selection condition must be on the search-key of the index.

- 3) (a) *Primary index-scan for equality:* This search retrieves a single record that satisfies the corresponding equality condition. The cost here can be calculated as:

Cost = Height traversed in index to locate the block pointer +1(block of the primary key is transferred for access).

(b) *Hash key:* It retrieves a single record in a direct way thus, cost in hash key may also be considered as Block transfer needed for finding hash target +1

- 4) *Primary index-scan for comparison:* Assuming that the relation is sorted on the attribute(s) that are being compared, ($<$, $>$ etc.), then we need to locate the

first record satisfying the condition after which the records are scanned forward or backward as the condition may be, displaying all the records. Thus cost in this case would be:

Cost = Number of block transfer to locate the value in index + Transferring all the blocks of data satisfying that condition.

Please note we can calculate roughly (from the cost point of view) the number of blocks satisfying the condition as:

Number of values that satisfy the condition \times average number of tuples per attribute value/blocking factor of the relation.

- 5) *Equality on clustering index* to retrieve multiple records: The cost calculations in this case are somewhat similar to that of algorithm (4).
- 6) (a) *Equality on search-key of secondary index*: Retrieves a single record if the search-key is a candidate key.

$Cost = \text{cost of accessing index} + 1.$

It retrieves multiple records if search-key is not a candidate key.

$Cost = \text{cost of accessing index} + \text{number of records retrieved}$ (It can be very expensive).

Each record may be on a different block, thus, requiring one block access for each retrieved record (this is the worst case cost).

(b) *Secondary index, comparison*: For the queries of the type that use comparison on secondary index value \geq a value, then the index can be used to find first index entry which is greater than that value, scan index sequentially from there till the end and also keep finding the pointers to records.

For the \leq type query just scan leaf pages of index, also keep finding pointers to records, till first entry is found satisfying the condition.

In either case, retrieving records that are pointed to, may require an I/O for each record. Please note linear file scans may be cheaper if many records are to be fetched.

Implementation of Complex Selections

Conjunction: Conjunction is basically a set of AND conditions.

- 7) *Conjunctive selection using one index*: In such a case, select any algorithm given earlier on one or more conditions. If possible, test other conditions on these tuples after fetching them into memory buffer.
- 8) *Conjunctive selection using multiple-key index*: Use appropriate composite (multiple-key) index if they are available.
- 9) *Conjunctive selection by intersection of identifiers* requires indices with record pointers. Use corresponding index for each condition, and take the intersection

of all the obtained sets of record pointers. Then fetch records from file if, some conditions do not have appropriate indices, test them after fetching the tuple from the memory buffer.

Disjunction: Specifies a set of OR conditions.

- 10) *Disjunctive selection by union of identifiers* is applicable if *all* conditions have available indices, otherwise use linear scan. Use corresponding index for each condition, take the union of all the obtained sets of record pointers, and eliminate duplicates, then fetch data from the file.

Negation: Use linear scan on file. However, if very few records are available in the result and an index is applicable on attribute, which is being negated, then find the satisfying records using index and fetch them from the file.

12.4 SORTING

Now we need to take a look at sorting techniques that can be used for calculating costing. There are various methods that can be used in the following ways:

- 1) Use an existing applicable ordered index (e.g., B+ tree) to read the relation in sorted order.
- 2) Build an index on the relation, and then use the index to read the relation in sorted order. (Options 1&2 may lead to one block access per tuple).
- 3) For relations that fit in the memory, techniques like quicksort can be used.
- 4) For relations that do not fit in the memory, *external sort-merge* is a good choice.

Let us go through the algorithm for External Sort-Merge.

i) **Create Sorted Partitions:**

Let i be 0 initially.

Repeat steps (a) to (d) until the end of the relation:

- (a) Read M blocks of relation into the memory. (Assumption M is the number of available buffers for the algorithm).
- (b) Sort these buffered blocks using internal sorting.
- (c) Write sorted data to a temporary file – temp (i)
- (d) $i = i + 1;$

Let the final value of i be denoted by N ;

Please note that each temporary file is a sorted partition.

ii) Merging Partitions (N-way merge):

// We assume (for now) that $N < M$.
 // Use N blocks of memory to buffer temporary files and 1 block to buffer output.

Read the first block of each temporary file (partition) into its input buffer block;

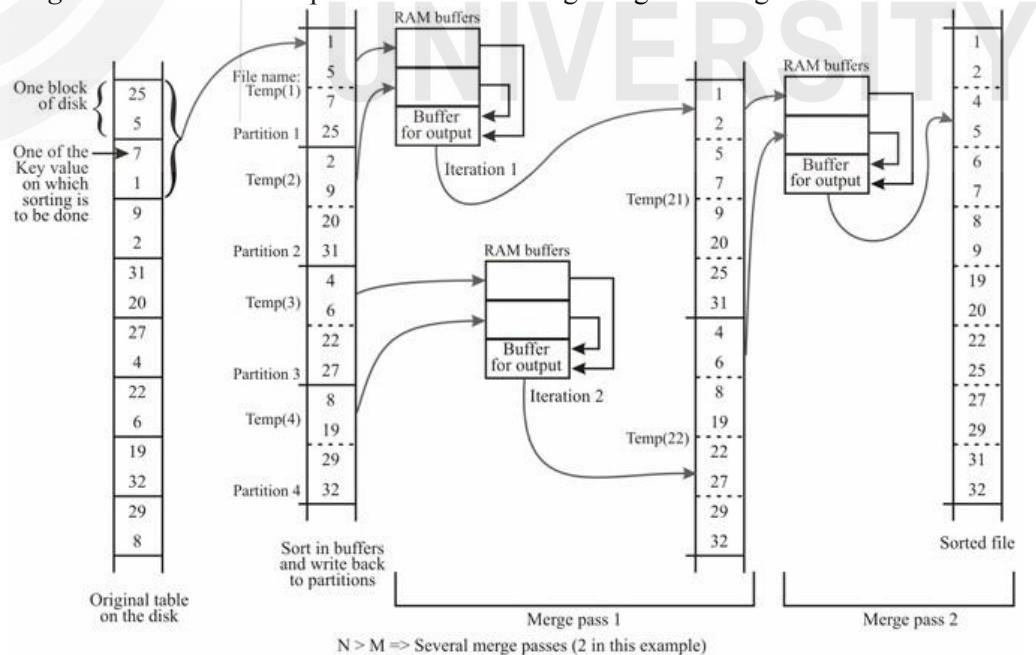
Repeat steps (a) to (e) until all input buffer blocks are empty;

- (a) Select the first record (in sort order) among all input buffer blocks;
- (b) Write the record to the output buffer block;
- (c) If the output buffer block is full then write it to disk and empty it for the next set of data. This step may be performed automatically by the OperatingSystem;
- (d) Delete the record from its input buffer block;
- (e) If the buffer block becomes empty then read the next block (if any) of the temporary file into the buffer.

If $N \geq M$, several merge *passes* are required, in each pass, contiguous groups of $M - 1$ partitions are merged and a pass reduces the number of temporary files temp (i) by a factor of $M - 1$. For example, if $M=11$ and there are 90 temporary files, one pass reduces the number of temporary files to 9, each temporary file begin 10 times the size of the earlier partitions.

Repeated passes are performed till all partitions have been merged into one.

Figure 2 shows an example for external sorting using sort-merge.



Assumption $M = 3 \Rightarrow$ Only two blocks to be considered at a time for sorting. One block is kept for output.

Figure 2: External merge-sort example

Cost Analysis:

Cost Analysis is may be performed, according to the following activities:

- Assume the file has a total of Z blocks.
- Z block input to buffers + Z block output – for temporary file creation.
- Assuming that $N \geq M$, then a number of merge passes are required
- Number of merge passes = $\lceil \log_{M-1} (Z/M) \rceil$. Please note that of M buffers 1 is used for output.
- So number of block transfers needed for merge passes = $2 \times Z (\lceil \log_{M-1} (Z/M) \rceil)$ as all the blocks will be read and written back of the buffer for each merge pass.
- Thus, the total number of passes for sort-merge algorithm = $2Z + 2Z (\lceil \log_{M-1} (Z/M) \rceil) = 2Z \times (\lceil \log_{M-1} (Z/M) \rceil + 1)$.

☞ Check Your Progress 1

1) What are the basic steps in query processing?

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

2) How can the cost of query be measured?

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

3) What are the various methods adopted in select operation?

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

4) Define External-Sort-Merge algorithm.

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

12.5 JOIN OPERATION

There are number of algorithms that can be used to implement joins:

- Nested-loop join
- Block nested-loop join
- Indexed nested-loop join
- Merge-join
- Hash-join
- Complex join.

The choice of join algorithm is based on the cost estimate. Let us use the following information to elaborate the same:

MARKS (enroll no, subject code, marks):10000 rows, 500 blocks

STUDENT (enroll no, name, dob): 2000 rows, 100 blocks

12.5.1 Nested-Loop Join

Let us show the algorithm for the given relations.

To compute the theta or equi-join

```
for each tuple s in STUDENT
{
    for each tuple m in MARKS
    {
        test s.enrolno = m.enrolno to see if they satisfy the
        joincondition if they do, output joined tuple to the
        result.
    };
}
```

- In the nested loop join there is one *outer* relation and one *inner* relation.
- It does not use or require indices. It can be used with any kind of join condition. However, it is expensive as it examines every pair of tuples in the two relations.
- If there is only enough memory to hold one block of each relation, the number of disk accesses can be calculated as:

For each tuple of STUDENT, all the MARKS tuples (blocks) that need to be accessed.

However, if both or one of the relations fit entirely in the memory, block transfer will be needed only once, so the total number of transfers in such a case, may be calculated as:

$$\begin{aligned} &= \text{Number of blocks of STUDENT} + \text{Number of blocks of MARKS} \\ &= 100 + 500 = 600. \end{aligned}$$

If only the smaller of the two relations fits entirely in the memory then use that as the inner relation and the bound still holds.

Cost for the worst case:

Number of tuples of outer relation \times Number of blocks of inner relation + Number of blocks of outer relation.

$2000 * 500 + 100 = 1,000,100$ with STUDENT as outer relation.

There is one more possible bad case when MARKS is on outer loop and STUDENT in the inner loop. In this case, the number of Block transfer will be:

$10000 * 100 + 500 = 1,000,500$ with MARKS as the outer relation.

12.5.2 Block Nested-Loop Join

This is a variant of nested-loop join in which a complete block of outer loop is joined with the block of inner loop.

The algorithm for this may be written has:

```
for each block s of STUDENT
{
    for each block m of MARKS
    {
        for each tuple si in s
        {
            for each tuple mi in m
            {
                Check if (si and mi) satisfy the join condition
                if they do output joined tuple to the result
            };
        };
    };
}
```

Worst case of block accesses in this case = Number of Blocks of outer relation (STUDENT) \times Number of blocks of inner relation (MARKS) + Number of blocks of outer relation (STUDENT).

Best case: Blocks of STUDENT + Blocks of MARKS Number of block transfers

assuming worst case:

$100 * 500 + 100 = 50,100$ (much less than nested-loop join)

Number of block transfers assuming best case:

$400 + 100 = 500$ (same as with nested-loop join)

Improvements to Block Nested-Loop Algorithm

The following modifications improve the block Nested method:

Use $M - 2$ disk blocks as the blocking unit for the outer relation, where M = memory size in blocks.

Use one buffer block to buffer the inner relation.

Use one buffer block to buffer the output.

This method minimizes the number of iterations.

12.5.3 Indexed Nested-Loop Join

Index scans can replace file scans if the join is an equi-join or natural join, and an index is available on the inner relation's join attribute.

For each tuple si in the outer relation STUDENT, use the index to look up tuples in MARKS that satisfy the join condition with tuple si .

In a worst case scenarios, the buffer has space for only one page of STUDENT, and, for each tuple in MARKS, then we should perform an index lookup on *MARKS index*.

Worst case: Block transfer of STUDENT+ number of records in STUDENT * cost of searching through index and retrieving all matching tuples for each tuple of STUDENT.

If a supporting index does not exist than it can be constructed as and when needed.

If indices are available on the join attributes of both STUDENT and MARKS, then use the relation with fewer tuples as the outer relation.

Example of Index Nested-Loop Join Costs

Compute the cost for STUDENT and MARKS join, with STUDENT as the outer relation. Suppose MARKS has a primary B+-tree index on enroll no, which contains 10 entries in each index node. Since MARKS has 10,000 tuples, the height of the tree is 4, and one more access is needed to the actual data. The STUDENT has 2000 tuples. Thus, the cost of indexed nested loops join as:

$$100 + 2000 * 5 = 10,100 \text{ disk accesses}$$

12.5.4 Merge-Join

The merge-join is applicable to equi-joins and natural joins only. It has the following process:

- 1) Sort both relations on their join attribute (if not already sorted).
- 2) Merge the sorted relations to join them. The join step is similar to the merge stage of the sort-merge algorithm, the only difference lies in the manner in which duplicate values in join attribute are treated, i.e., every pair with same value on join attribute must be matched.

STUDENT		
Enrol no	Name	-----
1001	Ajay
1002	Aman
1005	Rakesh
1100	Raman
.....

MARKS		
Enrol no	subject code	Marks
1001	MCS-011	55
1001	MCS-012	75
1002	MCS-013	90
1005	MCS-015	75
.....

Figure 3: Sample relations for computing join

The number of block accesses:

Each block needs to be read only once (assuming all tuples for any given value of the

join attributes fit in memory). Thus number of block accesses for merge-join is:

Blocks of STUDENT + Blocks of MARKS + the cost of sorting if relations are unsorted.

Hybrid Merge-Join

This is applicable only when the join is an equi-join or a natural join and one relation is sorted and the other has a secondary B+-tree index on the join attribute.

The algorithm is as follows:

Merge the sorted relation with the leaf entries of the B+tree. Sort the result on the addresses of the unsorted relation's tuples. Scan the unsorted relation in physical address order and merge with the previous results, to replace addresses by the actual tuples. Sequential scan in such cases is more efficient than the random lookup method.

12.5.5 Hash-Join

This is applicable to both the equi-joins and natural joins. A hash function h is used to partition tuples of both relations, where h maps joining attribute (enroll no in our example) values to $\{0, 1, \dots, n-1\}$.

The join attribute is hashed to the join-hash partitions. In the example of *Figure 4* we have used mod 100 function to hashing, and $n = 100$.

Error!

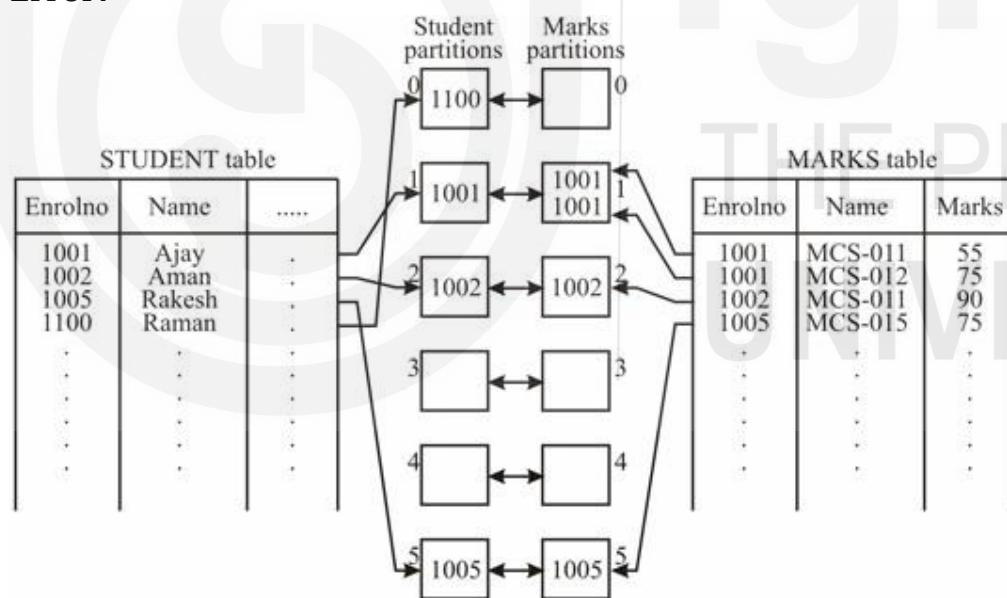


Figure 4: A hash-join example

Once the partition tables of STUDENT and MARKS are made on the enrolment number, then only the corresponding partitions will participate in the join as:

A STUDENT tuple and a MARKS tuple that satisfy the join condition will have the same value for the join attributes. Therefore, they will be hashed to equivalent partition and thus can be joined easily.

Algorithm for Hash-Join

The hash-join of two relations r and s is computed as follows:

- Partition the relation r and s using hashing function h . (When partitioning a relation, one block of memory is reserved as the output buffer for each partition).
- For each partition si of s , load the partition into memory and build an in-memory hash index on the join attribute.
- Read the tuples in ri from the disk, one block at a time. For each tuple in ri locate each matching tuple in si using the in-memory hash index and output the concatenation of their attributes.

In this method, the relation s is called the *build* relation and r is called the *probe* relation. The value n (the number of partitions) and the hash function h is chosen in such a manner that each si should fit in to the memory. Typically n is chosen as Number of blocks of s /Number of memory buffers] *f (M) where f is a “fudge factor”, typically around 1.2. The probe relation partitions ri need not fit in memory.

Average size of a partition si will be less than M blocks using the formula for n as above thereby allowing room for the index. If the build relation s is very huge, then the value of n as given by the above formula may be greater than $M - 1$ i.e., the number of buckets is $>$ the number of buffer pages. In such a case, the relation s can be recursively partitioned, instead of partitioning n ways, use $M - 1$ partitions for s and further partition the $M - 1$ partitions using a different hash function. You should use the same partitioning method on r . This method is rarely required, as recursive partitioning is not needed for relations of 1GB or less for a memory size of 2MB, with block size of 4KB.

Cost calculation for Simple Hash-Join

- (i) Cost of partitioning r and s : all the blocks of r and s are read once and after partitioning written back, so cost 1 = 2 (blocks of r + blocks of s).
- (ii) Cost of performing the hash-join using build and probe will require at least one block transfer for reading the partitions
Cost 2 = (blocks of r + blocks of s)
- (iii) There are a few more blocks in the main memory that may be used for evaluation, they may be read or written back. We ignore this cost as it will be too less in comparison to cost 1 and cost 2.
Thus, the total cost = cost 1 + cost 2
= 3 (blocks of r + blocks of s)

Cost of Hash-Join requiring recursive partitioning:

- (i) The cost of partitioning in this case will increase to number of recursion required, it may be calculated as:

$$\text{Number of iterations required} = ([\log_{M-1} (\text{blocks of } s)] - 1)$$

Thus, cost 1 will be modified as:

$$= 2 \text{ (blocks of } r + \text{blocks of } s) \times ([\log_{M-1}(\text{blocks of } s)] - 1)$$

The cost for step (ii) and (iii) here will be the same as that given in steps (ii) and (iii) above.

Thus, total cost = $2(\text{blocks of } r + \text{blocks of } s) \times ([\log_{M-1}(\text{blocks of } s)] - 1) + (\text{blocks of } r + \text{blocks of } s)$.

Because s is in the inner term in this expression, it is advisable to choose the smaller relation as the build relation. If the entire build input can be kept in the main memory, n can be set to 1 and the algorithm need not partition the relations but may still build an in-memory index, in such cases the cost estimate goes down to (Number of blocks r + Number of blocks of s).

Handling of Overflows

Even if s is recursively partitioned *hash-table overflow* can occur, i.e., some partition s_i may not fit in the memory. This may happen if there are many tuples in s with the same value for join attributes or a bad hash function.

Partitioning is said to be *skewed* if some partitions have significantly more tuples than the others. This is the overflow condition. The overflow can be handled in a variety of ways:

Resolution (during the build phase): The overflow partition s is further partitioned using different hash function. The equivalent partition of r must be further partitioned similarly.

Avoidance (during build phase): Partition build relations into many partitions, then combines them.

However, such approaches for handling overflows fail with large numbers of duplicates. One option of avoiding such problems is to use block nested-loop join on the overflowed partitions.

Let us explain the hash join and its cost for the natural join STUDENT MARKS. Assume a memory size of ~~25~~ blocks $\Rightarrow M=25$;

SELECT build s as STUDENT as it has less number of blocks (100 blocks) and r probe as MARKS (500 blocks).

Number of partitions to be created for STUDENT = (block of STUDENT/M)* fudge factor (1.2) = $(100/25) \times 1.2 = 4.8$

Thus, STUDENT relation will be partitioned into 5 partitions of 20 blocks each. MARKS will also have 5 partitions of 100 blocks each. The 25 buffers will be used as -20 blocks for one complete partition of STUDENT plus 4 more blocks for one block of each of the other 4 partitions. One block will be used for input of MARKS partitions.

The total cost = $3(100+500) = 1800$ as no recursive partitioning is needed.

Hybrid Hash-Join

This is useful when the size of the memory is relatively large, and the build input is larger than the memory. Hybrid hash join keeps the first partition of the build relation in the memory. The first partition of STUDENT is maintained in the first 20 blocks of the buffer, *and not written to the disk*. The first block of MARKS is used right away

for probing, instead of being written and read back. Thus, it has a cost of $3(80 + 400) + 20 + 100 = 1560$ block transfers for hybrid hash-join, instead of 1800 with plain hash-join.

Hybrid hash-join is most useful if M is large, such that we can have bigger partitions.

12.5.6 Complex Joins

A join with a conjunctive condition can be handled, either by using the nested loop or block nested loop join, or alternatively, the result of one of the simpler joins (on a few conditions) can be computed and the final result may be evaluated by finding the tuples that satisfy the remaining conditions in the result.

A join with a disjunctive condition can be calculated either by using the nested loop or block nested loop join, or it may be computed as the union of the records in individual joins.

12.6 OTHER OPERATIONS

There are many other operations that are performed in database systems. Let us introduce these processes in this section.

Duplicate Elimination: Duplicate elimination may be implemented by using hashing or sorting. On sorting, duplicates will be adjacent to each other thus, may be identified and deleted. An optimised method for duplicate elimination can be deletion of duplicates during generation as well as at intermediate merge steps in external sort-merge. Hashing is similar – duplicates will be clubbed together in the same bucket and therefore may be eliminated easily.

Projection: It may be implemented by performing the projection process on each tuple, followed by duplicate elimination.

Aggregate Function Execution: Aggregate functions can be implemented in a manner similar to duplicate elimination. Sorting or hashing can be used to bring tuples in the same group together, and then aggregate functions can be applied to each group. An optimised solution could be to combine tuples in the same group during part time generation and intermediate merges, by computing partial aggregate values. For count, min, max, sum, you may club aggregate values on tuples found so far in the group. When combining partial aggregates for counting, you would need to add up the aggregates. For calculating the average, take the sum of the aggregates and the count/number of aggregates, and then divide the sum with the count at the end.

Set operations (\cup , \cap and $-$) can either use a variant of merge-join after sorting, or a variant of hash-join.

Hashing:

- 1) Partition both relations using the same hash function, thereby creating r_0, \dots, r_{n-1} and s_0, \dots, s_{n-1}

- 2) Process each partition i as follows:
Using a different hashing function, build an-in-memory hash index on r_i after it is brought into the memory.
 $r \cup s$: Add tuples in s_i to the hash index if they are not already in it. At the end of s_i add the tuples in the hash index to the result.
 $r \cap s$: Output tuples in s_i to the result if they are already there in the hash index.

$r - s$: For each tuple in si , if it is there in the hash index, delete it from the index.

At end of si add remaining tuples in the hash index to the result.

There are many other operations as well. You may wish to refer to them in further readings.

☛ Check Your Progress 2

- 1) Define the algorithm for Block Nested-Loop Join for the worst-case scenario.

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

- 2) Define Hybrid Merge-Join.

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

- 4) Give the method for calculating the cost of Hash-Join?

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

- 5) Define other operations?

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

12.7 REPRESENTATION AND EVALUATION OF QUERY EXPRESSIONS

Before we discuss the evaluation of a query expression, let us briefly explain how a SQL query may be represented. Consider the following student and marks relations:

STUDENT (enrolno, name, phone)

MARKS (enrolno, subjectcode, grade)

To find the results of the student(s) whose phone number is ‘1129250025’, the following query may be given.

```
SELECT enrolno, name, subjectcode, grade  
FROM STUDENT s, MARKS m  
WEHRE s.enrolno=m.enrolno AND phone= '1129250025'
```

The equivalent relational algebraic query for this will be:

$$\pi_{\text{enrolno}, \text{name}, \text{subjectcode}} (\sigma_{\text{phone}='1129250025'} (\text{STUDENT}) \bowtie \text{MARKS})$$

This is a very good internal representation however, it may be a good idea to represent the relational algebraic expression to a query tree on which algorithms for query optimisation can be designed easily. In a query tree, nodes are the operators and relations represent the leaf. The query tree for the relational expression above would be:

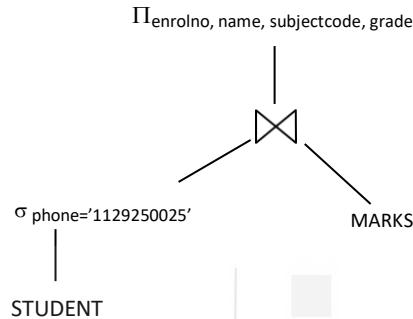


Figure 5: A Sample query tree

In the previous section we have seen the algorithms for individual operations. Now let us look at the methods for evaluating an entire expression. In general we use two methods:

- Materialisation
- Pipelining.

Materialisation: Evaluate a relational algebraic expression from the bottom-up, by explicitly generating and storing the results of each operation in the expression. For example, in *Figure 5* compute and store the result of the selection operation on STUDENT relation, then take the join of this result with MARKS relation and then finally compile the projection operation.

Materialised evaluation is always possible though; the cost of writing/reading results to/from disk can be quite high.

Pipelining

Evaluate operations in a multi-threaded manner, (i.e., passes tuples output from one operation to the next parent operation as input) even as the first operation is being executed. In the previous expression tree, it does not store (materialise) results instead, it passes tuples directly to the join. Similarly, does not store results of join, and passes tuples directly to the projection. Thus, there is no need to store a temporary relation on a disk for each operation. Pipelining may not always be possible or easy for sort, hash-join.

One of the pipelining execution methods may involve a buffer filled by the result tuples of lower level operation while, records may be picked up from the buffer by the higher level operation.

Complex Joins

When an expression involves three relations then we have more than one strategy for the evaluation of the expression. For example, join of relations such as:

STUDENT \bowtie MARKS \bowtie SUBJECTS
may involve the following three strategies:

Strategy 1: Compute STUDENT \bowtie MARKS; use result to compute result
 \bowtie SUBJECTS

Strategy 2: Compute MARKS \bowtie SUBJECTS first, and then join the result with STUDENT

Strategy 3: Perform the pair of joins at the same time. This can be done by building an index of enroll no in STUDENT and on subject code in SUBJECTS.

For each tuple m in MARKS, look up the corresponding tuples in STUDENT and the corresponding tuples in SUBJECTS. Each tuple of MARKS will be examined only once.

Strategy 3 combines two operations into one special-purpose operation that may be more efficient than implementing the joins of two relations.

12.8 CREATION OF QUERY EVALUATION PLANS

We have already discussed query representation and its final evaluation in earlier section of this unit, but can something be done during these two stages that optimises the query evaluation? This section deals with this process in detail.

Generation of query-evaluation plans for an expression involves several steps:

- 1) Generating logically equivalent expressions using **equivalence rules**
- 2) Annotating resultant expressions to get alternative query plans
- 3) Choosing the cheapest plan based on **estimated cost**.

The overall process is called **cost based optimisation**.

The cost difference between a good and a bad method of evaluating a query would be enormous. We would therefore, need to estimate the cost of operations and statistical information about relations. For example a number of tuples, a number of distinct values for an attribute etc. Statistics helps in estimating intermediate results to compute the cost of complex expressions.

Let us discuss all the steps in query-evaluation plan development in more details next.

12.8.1 Transformation of Relational Expressions

Two relational algebraic expressions are said to be **equivalent** if on every legal database instance the two expressions generate the same set of tuples (order of tuples is irrelevant).

Let us define certain equivalence rules that may be used to generate equivalent relational expressions.

Equivalence Rules

- 1) The conjunctive selection operations can be equated to a sequence of individual selections. It can be represented as:

$$\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$$

- 2) The selection operations are commutative, that is,

$$\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$$

- 3) Only the last of the sequence of projection operations is needed, the others can be omitted.

$$\pi_{\text{attriblist1}}(\pi_{\text{attriblist2}}(\pi_{\text{attriblist3}} \dots (E) \dots) = \pi_{\text{attriblist1}}(E)$$

- 4) The selection operations can be combined with Cartesian products and theta join operations.

$$\sigma_{\theta_1}(E_1 \times E_2) = E_1 \bowtie_{\theta_1} E_2$$

and

$$\sigma_{\theta_2}(E_1 \bowtie_{\theta_1} E_2) = E_1 \bowtie_{\theta_2 \wedge \theta_1} E_2$$

- 5) The theta-join operations and natural joins are commutative.

$$E_1 \bowtie_{\theta} E_2 = E_2 \bowtie_{\theta} E_1$$

- 6) The Natural join operations are associative. Theta joins are also associative but with the proper distribution of joining condition:

$$(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$$

- 7) The selection operation distributes over the theta join operation, under conditions when all the attributes in selection predicate involve only the attributes of one of the expressions (E_1) being joined.

$$\sigma_{\theta_1}(E_1 \bowtie_{\theta} E_2) = (\sigma_{\theta_1}(E_1)) \bowtie_{\theta} E_2$$

- 8) The projections operation distributes over the theta join operation with only those attributes, which are present in that relation.

$$\pi_{\text{attriblist1} \cup \text{attriblist2}}(E_1 \bowtie_{\theta} E_2) = (\pi_{\text{attriblist1}}(E_1) \bowtie_{\theta} \pi_{\text{attriblist2}}(E_2))$$

- 9) The set operations of union and intersection are commutative. But set difference is not commutative.

$$E_1 \cup E_2 = E_2 \cup E \text{ and similarly for the intersection.}$$

- 10) Set union and intersection operations are also associative.

$$(E_1 \cup E_2) \cup E_3 = E_1 \cup (E_2 \cup E_3) \text{ and similarly for intersection.}$$

- 11) The selection operation can be distributed over the union, intersection, and set-differences operations.

$$\sigma_{\theta_1}(E_1 - E_2) = ((\sigma_{\theta_1}(E_1) - (\sigma_{\theta_1}(E_2)))$$

- 12) The projection operation can be distributed over the union.

$$\pi_{\text{attriblist1}}(E_1 \cup E_2) = \pi_{\text{attriblist1}}(E_1) \cup \pi_{\text{attriblist1}}(E_2)$$

The rules as above are too general and a few heuristics rules may be generated from

these rules, which help in modifying the relational expression in a better way. These rules are:

- (1) Combining a cascade of selections into a conjunction and testing all the predicates on the tuples at the same time:

$$\sigma_{\theta_2}(\sigma_{\theta_1}(E)) \text{ convert to } \sigma_{\theta_2 \wedge \theta_1}(E)$$

- (2) Combining a cascade of projections into single outer projection:

$$\pi_4(\pi_3(\dots(E))) = \pi_4(E)$$

- (3) Commutating the selection and projection or vice-versa sometimes reduces cost

- (4) Using associative or commutative rules for Cartesian product or joining to find various alternatives.

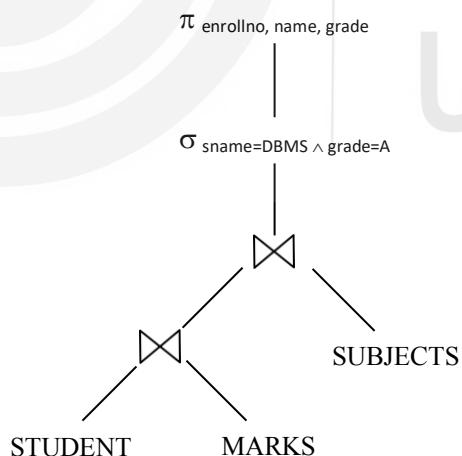
- (5) Moving the selection and projection (it may have to be modified) before joins. The selection and projection results in the reduction of the number of tuples and therefore may reduce cost of joining.

- (6) Commuting the projection and selection with Cartesian product or union.

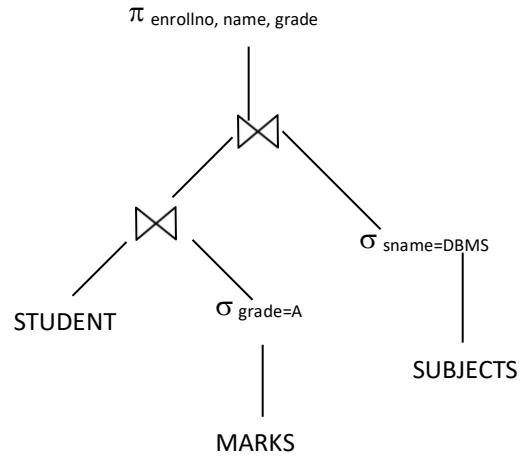
Let us explain use of some of these rules with the help of an example. Consider the query for the relations:

STUDENT (enrollno, name, phone)
 MARKS (enrollno, subjectcode, grade)
 SUBJECTS (subjectcode, sname)

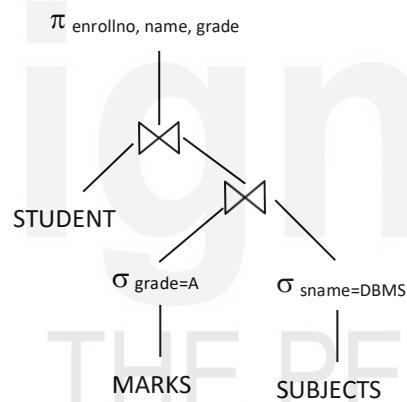
Consider the query: Find the enrolment number, name, and grade of those students who have secured an A grade in the subject DBMS. One of the possible solutions to this query may be:



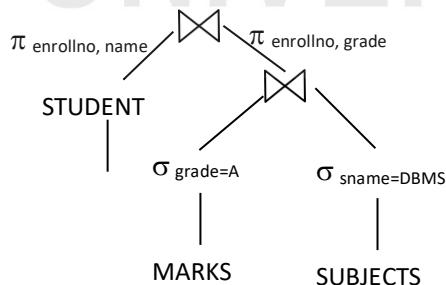
The selection condition may be moved to the join operation. The selection condition given in the *Figure* above is: *sname = DBMS and grade = A*. Both of these conditions belong to different tables, as *sname* is available only in the SUBJECTS table and *grade* in the MARKS table. Thus, the conditions of selection will be mapped accordingly as shown in the *Figure* below. Thus, the equivalent expression will be:



The expected size of SUBJECTS and MARKS after selection will be small so it may be a good idea to first join MARKS with SUBJECTS. Hence, the associative law of JOIN may be applied.



Finally projection may be moved inside. Thus the resulting query tree may be:



Please note the movement of the projections.

Obtaining Alternative Query Expressions

Query optimisers use equivalence rules to systematically generate expressions equivalent to the given expression. Conceptually, they generate all equivalent expressions by repeatedly executing the equivalence rules until no more expressions are to be found. For each expression found, use all applicable equivalence rules and

add newly generated expressions to the set of expressions already found. However, the approach above is very expensive both in time space requirements. The heuristics rules given above may be used to reduce cost and to create a few possible but good equivalent query expression.

12.8.2 Query Evaluation Plans

Let us first define the term Evaluation Plan.

An evaluation plan defines exactly which algorithm is to be used for each operation, and how the execution of the operation is coordinated. For example, *Figure 6* shows the query tree with evaluation plan.

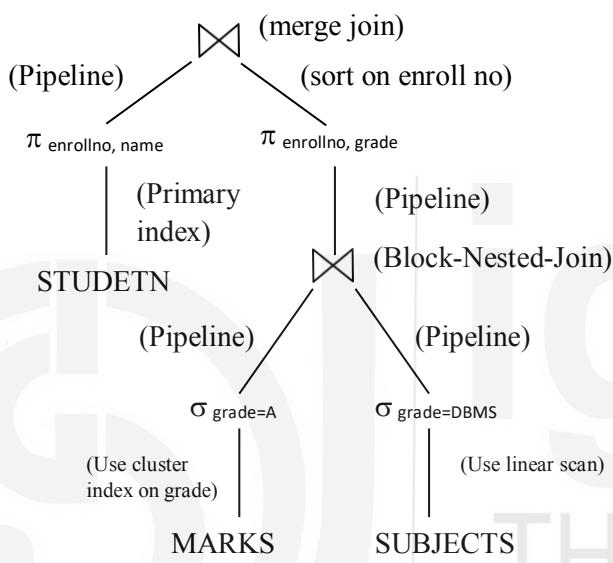


Figure 6: Query evaluation plan

12.8.3 Choice of Evaluation Plans

For choosing an evaluation technique, we must consider the interaction of evaluation techniques. Please note that choosing the cheapest algorithm for each operation independently may not yield best overall algorithm. For example, merge-join may be costlier than hash-join, but may provide a sorted output which reduces the cost for a higher level aggregation. A nested-loop join may provide opportunity for pipelining. Practical query optimisers incorporate elements of the following two broad approaches:

- 1) Searches all the plans and chooses the best plan in a cost-based fashion.
- 2) Uses heuristic rules to choose a plan.

12.8.4 Cost Based Optimisation

Cost based optimisation is performed on the basis of the cost of various individual operations that are to be performed as per the query evaluation plan. The cost is calculated as we have explained in the earlier section with respect to the method and operation (JOIN, SELECT, etc.).

12.8.5 Storage and Query Optimisation

Cost calculations are primarily based on disk access, thus, storage has an important

role to play in cost. In addition, some of the operations also require intermediate storage thus; cost is further enhanced in such cases. The cost of finding an optimal query plan is usually more than offset by savings at query-execution time, particularly by reducing the number of slow disk accesses.

12.9 VIEWS AND QUERY PROCESSING

A view must be prepared, passing its parameters, which describe the query and the manner in which tuples should be evaluated. This takes the form of a pre-evaluation window, which gives the user of the program the ability to trade off memory usage for faster navigation, or an attempt to balance both of these resources.

The view may maintain the complete set of tuples following evaluation. This requires a lot of memory space, therefore, it may be a good idea to partially pre-evaluate it.

This is done by hinting at how the number of that should be present in the evaluated view tuples before and after the current tuple. However, as the current tuple changes, further evaluation changes, thus, such scheme is harder to plan.

12.9.1 Materialised View

A materialised view is a view whose contents are computed and stored.

Materialising the view would be very useful if the result of a view is required frequently as it saves the effort of finding multiple tuples and totalling them up.

Further the task of keeping a materialised view up-to-date with the underlying data is known as materialised view maintenance. Materialised views can be maintained by recomputation on every update. A better option is to use incremental view maintenance, that is, where only the affected part of the view is modified. View maintenance can be done manually by defining triggers on insert, delete, and update of each relation in the view definition. It can also be written manually to update the view whenever database relations are updated or supported directly by the database.

12.9.2 Materialised Views and Query Optimisation

We can perform query optimisation by rewriting queries to use materialised views. For example, assume that a materialised view of the join of two tables b and c is available as:

$$a = b \text{ NATURAL JOIN } c$$

Any query that uses natural join on b and c can use this materialised view ' a ' as:

Consider you are evaluating a query:

$$z = r \text{ NATURAL JOIN } b \text{ NATURAL JOIN } c$$

Then this query would be rewritten using the materialised view ' a ' as:

$$z = r \text{ NATURAL JOIN } a$$

Do we need to perform materialisation? It depends on cost estimates for the two alternatives viz., use of a materialised view by view definition, or simple evaluation.

Query optimiser should be extended to consider all the alternatives of view evaluation

and choose the best overall plan. This decision must be made on the basis of the system workload. Indices in such decision-making may be considered as specialised views. Some database systems provide tools to help the database administrator with index and materialised view selection.

☛ Check Your Progress 3

- 1) Define methods used for evaluation of expressions?

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

- 2) How you define cost based optimisation?

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

- 2) How you define evaluation plan?

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

12.10 SUMMARY

In this unit we have discussed query processing and evaluation. A query in a DBMS is a very important operation, as it needs to be efficient. Query processing involves query parsing, representing query in alternative forms, finding the best plan of evaluation of a query and then actually evaluating it. The major query evaluation cost is the disk access time. In this unit, we have discussed the cost of different operations in details. However, an overall query cost will not be a simple addition of all such costs.

12.11 SOLUTIONS/ANSWERS

Check Your Progress 1

- 1) (a) In the first step scanning, parsing & validating is done
(b) Translation into relational algebra
(c) Parser checks syntax, verifies relations
(d) Query execution engine take a query evaluation plan, executes it and returns answer to the query.

2) Query cost is measured by taking into account following activities:

* Number of seeks * Number of blocks * Number of block written

For simplicity we just use number of block transfers from disk as the cost measure. During this activity we generally ignore the difference in cost between sequential and random I/O and CPU/communication cost.

3) Selection operation can be performed in a number of ways such as:

File Scan

1. Linear Search
2. Binary Search

Index Scan

1. (a) Primary index, equality
(b) Hash Key
2. Primary index, Comparison
3. Equality on clustering index
4. (a) Equality on search key of secondary index
(b) Secondary index comparison

4) Algorithm for External Sort-Merge

1. Create sorted partitions
 - (a) Read M blocks of relation into memory
 - (b) Write sorted data to partition R_i
2. Merge the partitions (N-way Merge) until all input buffer blocks are empty.

Check Your Progress 2

```
1) For each block  $B_r$  of  $r$  {  
    For each block  $B_s$  of  $s$  {  
        For each tuple  $t_i$  in  $B_r$  {  
            For each tuple  $t_j$  in  $B_s$  {  
                Test pair( $t_i, t_j$ ) to see if they satisfy the join condition  
                If they do, add the joined tuple to result  
            };  
        };  
    };  
};
```

2) Hybrid Merge-Join is applicable when the join is equi-join or natural join and one relation is sorted.

Merge the sorted relation with leaf of B+tree.

- (i) Sort the result on address of sorted relations tuples.
- (ii) Scan unsorted relation and merge with previous result.

3) Cost of Hash-join.

(i) **If recursive partitioning not required**

3(Blocks of r + blocks of s)

(ii) If recursive partitioning required then

2(blocks of r + blocks of s) ($\lceil \log_{m-1}(\text{blocks of } s) \rceil - 1$) + blocks of r + blocks of s

- 4) Other operations
 - (a) Duplicate elimination
 - (b) Projection
 - (c) Aggregate functions.

Check Your Progress 3

- 1) Methods used for evaluation of expressions:
 - (a) Materialisation
 - (b) Pipelining
- 2) Cost based optimisation
 - (a) Generalising logically equivalent expressions using equivalence rules
 - (b) Annotating resultant expressions to get alternative query plans.
 - (c) Choosing the cheapest plan based on estimated cost.
- 3) Evaluation plan defines exactly what algorithms are to be used for each operation and the manner in which the operations are coordinated.



For Unit 13 : Please read the following unit of MCS-43 Block 3 Unit 1

UNIT 1 OBJECT ORIENTED DATABASE

Object Oriented Database

Structure	Page No.
1.0 Introduction	5
1.1 Objectives	5
1.2 Why Object Oriented Database?	6
1.2.1 Limitation of Relational Databases	
1.2.2 The Need for Object Oriented Databases	
1.3 Object Relational Database Systems	8
1.3.1 Complex Data Types	
1.3.2 Types and Inheritances in SQL	
1.3.3 Additional Data Types of OOP in SQL	
1.3.4 Object Identity and Reference Type Using SQL	
1.4 Object Oriented Database Systems	15
1.4.1 Object Model	
1.4.2 Object Definition Language	
1.4.3 Object Query Language	
1.5 Implementation of Object Oriented Concepts in Database Systems	22
1.5.1 The Basic Implementation issues for Object-Relational Database Systems	
1.5.2 Implementation Issues of OODBMS	
1.6 OODBMS Vs Object Relational Database	23
1.7 Summary	24
1.8 Solutions/Answers	24

1.0 INTRODUCTION

Object oriented software development methodologies have become very popular in the development of software systems. Database applications are the backbone of most of these commercial business software developments. Therefore, it is but natural that, object technologies also, have their impact on database applications. Database models are being enhanced in computer systems for developing complex applications. For example, a true hierarchical data representation like generalisation hierarchy scheme in a rational database would require a number of tables, but could be a very natural representation for an object oriented system. Thus, object oriented technologies have found their way into database technologies. The present day commercial RDBMS supports the features of object orientation.

This unit provides an introduction to various features of object oriented databases. In this unit, we shall discuss, the need for object oriented databases, the complex types used in object oriented databases, how these may be supported by inheritance etc. In addition, we also define object definition language (ODL) and object manipulation language (OML). We shall discuss object-oriented and object relational databases as well.

1.1 OBJECTIVES

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

- define the need for object oriented databases;
- explain the concepts of complex data types;
- use SQL to define object oriented concepts;



- familiarise yourself with object definition and query languages, and
- define object relational and object-oriented databases.

1.2 WHY OBJECT ORIENTED DATABASE?

An object oriented database is used for complex databases. Such database applications require complex interrelationships among object hierarchies to be represented in database systems. These interrelationships are difficult to be implement in relational systems. Let us discuss the need for object oriented systems in advanced applications in more details. However, first, let us discuss the weakness of the relational database systems.

1.2.1 Limitation of Relational Databases

Relational database technology was not able to handle complex application systems such as Computer Aided Design (CAD), Computer Aided Manufacturing (CAM), and Computer Integrated Manufacturing (CIM), Computer Aided Software Engineering (CASE) etc. The limitation for relational databases is that, they have been designed to represent entities and relationship in the form of two-dimensional tables. Any complex interrelationship like, multi-valued attributes or composite attribute may result in the decomposition of a table into several tables, similarly, complex interrelationships result in a number of tables being created. Thus, the main asset of relational databases viz., its simplicity for such applications, is also one of its weaknesses, in the case of complex applications.

The data domains in a relational system can be represented in relational databases as standard data types defined in the SQL. However, the relational model does not allow extending these data types or creating the user's own data types. Thus, limiting the types of data that may be represented using relational databases.

Another major weakness of the RDMS is that, concepts like inheritance/hierarchy need to be represented with a series of tables with the required referential constraint. Thus they are not very natural for objects requiring inheritance or hierarchy.

However, one must remember that relational databases have proved to be commercially successful for text based applications and have lots of standard features including security, reliability and easy access. Thus, even though they, may not be a very natural choice for certain applications, yet, their advantages are far too many. Thus, many commercial DBMS products are basically relational but also support object oriented concepts.

1.2.2 The Need for Object Oriented Databases

As discussed in the earlier section, relational database management systems have certain limitations. But how can we overcome such limitations? Let us discuss some of the basic issues with respect to object oriented databases.

The objects may be complex, or they may consist of low-level objects (for example, a window object may consist of many simpler objects like menu bars scroll bar etc.). However, to represent the data of these complex objects through relational database models you would require many tables – at least one each for each inherited class and a table for the base class. In order to ensure that these tables operate correctly we would need to set up referential integrity constraints as well. On the other hand, object



oriented models would represent such a system very naturally through, an inheritance hierarchy. Thus, it is a very natural choice for such complex objects.

Consider a situation where you want to design a class, (let us say a Date class), the advantage of object oriented database management for such situations would be that they allow representation of not only the structure but also the operation on newer user defined database type such as finding the difference of two dates. Thus, object oriented database technologies are ideal for implementing such systems that support complex inherited objects, user defined data types (that require operations in addition to standard operation including the operations that support polymorphism).

Another major reason for the need of object oriented database system would be the seamless integration of this database technology with object-oriented applications. Software design is now, mostly based on object oriented technologies. Thus, object oriented database may provide a seamless interface for combining the two technologies.

The Object oriented databases are also required to manage complex, highly interrelated information. They provide solution in the most natural and easy way that is closer to our understanding of the system. **Michael Brodie** related the object oriented system to human conceptualisation of a problem domain which enhances communication among the system designers, domain experts and the system end users.

The concept of object oriented database was introduced in the late 1970s, however, it became significant only in the early 1980s. The initial commercial product offerings appeared in the late 1980s. Today, many object oriented databases products are available like Objectivity/DB (developed by Objectivity, Inc.), ONTOS DB (developed by ONTOS, Inc.), VERSANT (developed by Versant Object Technology Corp.), ObjectStore (developed by Object Design, Inc.), GemStone (developed by Servio Corp.) and ObjectStore PSE Pro (developed by Object Design, Inc.). An object oriented database is presently being used for various applications in areas such as, e-commerce, engineering product data management; and special purpose databases in areas such as, securities and medicine.

Figure 1 traces the evolution of object oriented databases. *Figure 2* highlights the strengths of object oriented programming and relational database technologies. An object oriented database system needs to capture the features from both these world. Some of the major concerns of object oriented database technologies include access optimisation, integrity enforcement, archive, backup and recovery operations etc.

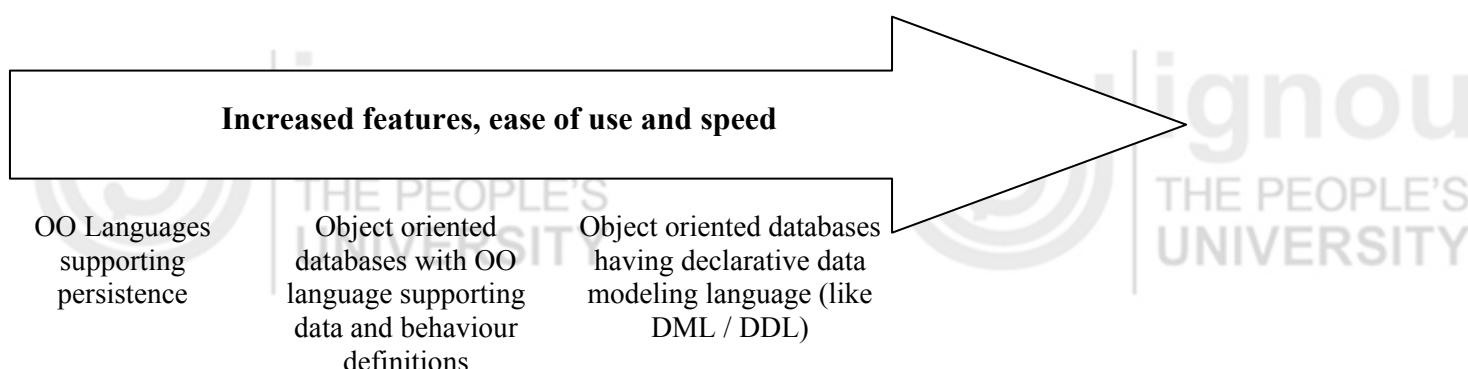


Figure 1: The evolution of object-oriented databases

The major standard bodies in this area are Object Management Group (OMG), Object Database Management Group (ODMG) and X3H7.



Object Oriented Database Technologies

Object Oriented Programming

- Inheritance
- Encapsulation
- Object identity
- Polymorphism

Relational Database

Features

- Security
- Integrity
- Transactions
- Concurrency
- Recovery
- Persistence

Figure 2: Makeup of an Object Oriented Database

Now, the question is, how does one implement an Object oriented database system? As shown in *Figure 2* an object oriented database system needs to include the features of object oriented programming and relational database systems. Thus, the two most natural ways of implementing them will be either to extend the concept of object oriented programming to include database features – OODBMS or extend the relational database technology to include object oriented related features – Object Relational Database Systems. Let us discuss these two viz., the object relational and object oriented databases in more details in the subsequent sections.

1.3 OBJECT RELATIONAL DATABASE SYSTEMS

Object Relational Database Systems are the relational database systems that have been enhanced to include the features of object oriented paradigm. This section provides details on how these newer features have been implemented in the SQL. Some of the basic object oriented concepts that have been discussed in this section in the context of their inclusion into SQL standards include, the complex types, inheritance and object identity and reference types.

1.3.1 Complex Data Types

In the previous section, we have used the term complex data types without defining it. Let us explain this with the help of a simple example. Consider a composite attribute – *Address*. The address of a person in a RDBMS can be represented as:

House-no and apartment
Locality
City
State
Pin-code



When using RDBMS, such information either needs to be represented as set attributes as shown above, or, as just one string separated by a comma or a semicolon. The second approach is very inflexible, as it would require complex string related operations for extracting information. It also hides the details of an address, thus, it is not suitable.

If we represent the attributes of the address as separate attributes then the problem would be with respect to writing queries. For example, if we need to find the address of a person, we need to specify all the attributes that we have created for the address viz., House-no, Locality.... etc. The question is –Is there any better way of representing such information using a single field? If, there is such a mode of representation, then that representation should permit the distinguishing of each element of the address? The following may be one such possible attempt:

```
CREATE TYPE Address AS (
    House      Char(20)
    Locality   Char(20)
    City       Char(12)
    State      Char(15)
    Pincode    Char(6)
);
```

Thus, *Address* is now a new type that can be used while showing a database system scheme as:

```
CREATE TABLE STUDENT (
    name      Char(25)
    address   Address
    phone     Char(12)
    programme Char(5)
    dob       ????
);
```

* Similarly, complex data types may be extended by including the date of birth field (dob), which is represented in the discussed scheme as???. This complex data type should then, comprise associated fields such as, day, month and year. This data type should also permit the recognition of difference between two dates; the day; and the year of birth. But, how do we represent such operations. This we shall see in the next section.

But, what are the advantages of such definitions?

Consider the following queries:

Find the name and address of the students who are enrolled in MCA programme.

```
SELECT      name, address
FROM        student
WHERE       programme = 'MCA' ;
```

Please note that the attribute ‘address’ although composite, is put only once in the query. But can we also refer to individual components of this attribute?

Find the name and address of all the MCA students of Mumbai.

```
SELECT      name, address
FROM        student
WHERE       programme = 'MCA' AND address.city = 'Mumbai';
```



Thus, such definitions allow us to handle a composite attribute as a single attribute with a user defined type. We can also refer to any of the component of this attribute without any problems so, the data definition of attribute components is still intact.

Complex data types also allow us to model a table with multi-valued attributes which would require a new table in a relational database design. For example, a library database system would require the representation following information for a book.

Book table:

- ISBN number
- Book title
- Authors
- Published by
- Subject areas of the book.

Clearly, in the table above, authors and subject areas are multi-valued attributes. We can represent them using tables (ISBN number, author) and (ISBN number, subject area) tables. (Please note that our database is not considering the author position in the list of authors).

Although this database solves the immediate problem, yet it is a complex design. This problem may be most naturally represented if, we use the object oriented database system. This is explained in the next section.

1.3.2 Types and Inheritances in SQL

In the previous sub-section we discussed the data type – Address. It is a good example of a structured type. In this section, let us give more examples for such types, using SQL. Consider the attribute:

- Name – that includes given name, middle name and surname
- Address – that includes address details, city, state and pincode.
- Date – that includes day, month and year and also a method for distinguish one data from another.

SQL uses Persistent Stored Module (PSM)/PSM-96 standards for defining functions and procedures. According to these standards, functions need to be declared both within the definition of type and in a CREATE METHOD statement. Thus, the types such as those given above, can be represented as:

```
CREATE TYPE      Name AS (
    given-name Char(20),
    middle-name   Char(15),
    sur-name     Char(20)
)
FINAL
```

```
CREATE TYPE      Address AS (
    add-det      Char(20),
    city         Char(20),
    state        Char(20),
    pincode      Char(6)
)
```

NOT FINAL



```

CREATE TYPE Date AS (
    dd Number(2),
    mm Number(2),
    yy Number(4)
)
FINAL
METHOD difference (present Date)
RETURNS INTERVAL days ;

```

This method can be defined separately as:

```

CREATE INSTANCE METHOD difference (present Date)
    RETURNS INTERVAL days FOR Date
BEGIN
// Code to calculate difference of the present date to the date stored in the object. //
// The data of the object will be used with a prefix SELF as: SELF.yy, SELF.mm etc.
//
// The last statement will be RETURN days that would return the number of days//
END

```

These types can now be used to represent class as:

```

CREATE TYPE Student AS (
    name Name,
    address Address,
    dob Date
)

```

'FINAL' and 'NOT FINAL' key words have the same meaning as you have learnt in JAVA. That is a final class cannot be inherited further.

There also exists the possibility of using constructors but, a detailed discussion on that is beyond the scope of this unit.

Type Inheritance

In the present standard of SQL, you can define inheritance. Let us explain this with the help of an example.

Consider a type University-person defined as:

```

CREATE TYPE University-person AS (
    name Name,
    address Address
)

```

Now, this type can be inherited by the Staff type or the Student type. For example, the Student type if inherited from the class given above would be:

```

CREATE TYPE Student
    UNDER University-person (
        programme Char(10),
        dob Number(7)
)

```

Similarly, you can create a sub-class for the staff of the University as:

```

CREATE TYPE Staff

```



UNDER University-person (
 designation Char(10),
 basic-salary Number(7))

Notice, that, both the inherited types shown above-inherit the name and address attributes from the type University-person. Methods can also be inherited in a similar way, however, they can be overridden if the need arises.

Table Inheritance

The concept of table inheritance has evolved to incorporate implementation of generalisation/ specialisation hierarchy of an E-R diagram. SQL allows inheritance of tables. Once a new type is declared, it could be used in the process of creation of new tables with the usage of keyword “OF”. Let us explain this with the help of an example.

Consider the University-person, Staff and Student as we have defined in the previous sub-section. We can create the table for the type University-person as:

```
CREATE TABLE University-members OF University-person ;
```

Now the table inheritance would allow us to create sub-tables for such tables as:

```
CREATE TABLE student-list OF Student  
UNDER University-members ;
```

Similarly, we can create table for the University-staff as:

```
CREATE TABLE staff OF Staff  
UNDER University-members ;
```

Please note the following points for table inheritance:

- The type that associated with the sub-table must be the sub-type of the type of the parent table. This is a major requirement for table inheritance.
- All the attributes of the parent table – (University-members in our case) should be present in the inherited tables.
- Also, the three tables may be handled separately, however, any record present in the inherited tables are also implicitly present in the base table. For example, any record inserted in the student-list table will be implicitly present in university-members tables.
- A query on the parent table (such as university-members) would find the records from the parent table and all the inherited tables (in our case all the three tables), however, the attributes of the result table would be the same as the attributes of the parent table.
- You can restrict your query to – only the parent table used by using the keyword – ONLY. For example,

SELECT NAME FROM university-member ONLY ;



1.3.3 Additional Data Types of OOP in SQL

The object oriented/relational database must support the data types that allows multi-valued attributes to be represented easily. Two such data types that exist in SQL are:

- Arrays – stores information in an order, and
- Multisets – stores information in an unordered set.

Let us explain this with the help of example of book database as introduced in section 1.3. This database can be represented using SQL as:

```
CREATE TYPE Book AS (
    ISBNNO      Char (14),
    TITLE       Char (25),
    AUTHORS     Char (25) ARRAY [5],
    PUBLISHER   Char (20),
    KEYWORDS    Char (10) MULTISET
)
```

Please note, the use of the type ARRAY. Arrays not only allow authors to be represented but, also allow the sequencing of the name of the authors. Multiset allows a number of keywords without any ordering imposed on them.

But how can we enter data and query such data types? The following SQL commands would help in defining such a situation. But first, we need to create a table:

```
CREATE TABLE library OF Book ;
```

```
INSERT INTO library VALUES.
('008-124476-x', 'Database Systems', ARRAY ['Silberschatz', 'Elmasri'], 'XYZ
PUBLISHER', multiset ['Database', 'Relational', 'Object Oriented']);
```

The command above would insert information on a hypothetical book into the database.

Let us now write few queries on this database:

Find the list of books related to area Object Oriented:

```
SELECT ISBNNO, TITLE
FROM library
WHERE 'Object Oriented' IN (UNNEST (KEYWORDS));
```

Find the first author of each book:

```
SELECT ISBNNO, TITLE, AUTHORS [1]
FROM library
```

You can create many such queries, however, a detailed discussion on this, can be found in the SQL 3 standards and is beyond the scope of this unit.

1.3.4 Object Identity and Reference Type Using SQL

Till now we have created the tables, but what about the situation when we have attributes that draw a reference to another attribute in the same table. This is a sort of referential constraint. The two basic issues related such a situation may be:

- How do we indicate the referenced object? We need to use some form of identity, and
- How do we establish the link?



Let us explain this concept with the help of an example; consider a book procurement system which provides an accession number to a book:

```
CREATE TABLE      book-purchase-table (
    ACCESSION-NO  CHAR (10),
    ISBNNO REF (Book) SCOPE (library)
);
```

The command above would create the table that would give an accession number of a book and will also refer to it in the library table.

However, now a fresh problem arises how do we insert the books reference into the table? One simple way would be to search for the required ISBN number by using the system generated object identifier and insert that into the required attribute reference. The following example demonstrates this form of insertion:

```
INSERT INTO book-purchase-table VALUES ('912345678', NULL);
```

```
UPDATE book-table
SET ISBNNO = (SELECT book_id
               FROM library
              WHERE ISBNNO = '83-7758-476-6')
WHERE ACCESSION-NO = '912345678'
```

Please note that, in the query given above, the sub-query generates the object identifier for the ISBNNO of the book whose accession number is 912345678. It then sets the reference for the desired record in the book-purchase-table.

This is a long procedure, instead in the example as shown above, since, we have the ISBNNO as the key to the library table, therefore, we can create a user generated object reference by simply using the following set of SQL statements:

```
CREATE TABLE      book-purchase-table (
    ACCESSION-NO  CHAR (10),
    ISBNNO REF (Book) SCOPE (library) USER GENERATED
);
```

```
INSERT INTO book-purchase-table VALUES ('912345678', '83-7758-476-6');
```

☛ Check Your Progress 1

- 1) What is the need for object-oriented databases?

- 2) How will you represent a complex data type?



- 3) Represent an address using SQL that has a method for locating pin-code information.
-
.....
.....

- 4) Create a table using the type created in question 3 above.
-
.....
.....

- 5) How can you establish a relationship with multiple tables?
-
.....
.....

1.4 OBJECT ORIENTED DATABASE SYSTEMS

Object oriented database systems are the application of object oriented concepts into database system model to create an object oriented database model. This section describes the concepts of the object model, followed by a discussion on object definition and object manipulation languages that are derived SQL.

1.4.1 Object Model

The ODMG has designed the object model for the object oriented database management system. The Object Definition Language (ODL) and Object Manipulation Language (OML) are based on this object model. Let us briefly define the concepts and terminology related to the object model.

Objects and Literal: These are the basic building elements of the object model. An object has the following four characteristics:

- A unique identifier
- A name
- A lifetime defining whether it is persistent or not, and
- A structure that may be created using a type constructor. The structure in OODBMS can be classified as atomic or collection objects (like Set, List, Array, etc.).

A literal does not have an identifier but has a value that may be constant. The structure of a literal does not change. Literals can be atomic, such that they correspond to basic data types like int, short, long, float etc. or structured literals (for example, current date, time etc.) or collection literal defining values for some collection object.

Interface: Interfaces defines the operations that can be inherited by a user-defined object. Interfaces are non-instantiable. All objects inherit basic operations (like copy object, delete object) from the interface of Objects. A collection object inherits operations – such as, like an operation to determine empty collection – from the basic collection interface.



Atomic Objects: An atomic object is an object that is not of a collection type. They are user defined objects that are specified using *class* keyword. The properties of an atomic object can be defined by its attributes and relationships. An example is the book object given in the next sub-section. **Please note** here that a *class* is instantiable.

Inheritance: The interfaces specify the abstract operations that can be inherited by classes. This is called behavioural inheritance and is represented using “:” symbol. Sub-classes can inherit the state and behaviour of super-class(s) using the keyword EXTENDS.

Extents: An extent of an object that contains all the persistent objects of that class. A class having an extent can have a key.

In the following section we shall discuss the use of the ODL and OML to implement object models.

1.4.2 Object Definition Language

Object Definition Language (ODL) is a standard language on the same lines as the DDL of SQL, that is used to represent the structure of an object-oriented database. It uses unique object identity (OID) for each object such as library item, student, account, fees, inventory etc. In this language objects are treated as records. Any class in the design process has three properties that are attribute, relationship and methods. A class in ODL is described using the following syntax:

```
class <name>
{
    <list of properties>
};
```

```
class Book
{
    attribute string ISBNNO;
    attribute string TITLE;
    attribute enum CATEGORY
        {text,reference,journal} BOOKTYPE;
    attribute struct AUTHORS
        {string fauthor, string sauthor, string
        tauthor}
        AUTHORLIST;
```

Please note that, in this case, we have defined authors as a structure, and a new field on book type as an enum.

These books need to be issued to the students. For that we need to specify a relationship. The relationship defined in ODL specifies the method of connecting one object to another. We specify the relationship by using the keyword “relationship”. Thus, to connect a student object with a book object, we need to specify the relationship in the student class as:

```
relationship set <Book> receives
```



Here, for each object of the class student there is a reference to book object and the set of references is called receives.

But if we want to access the student based on the book then the “inverse relationship” could be specified as

```
relationship set <Student> receivedby
```

We specify the connection between the relationship receives and receivedby by, using a keyword “inverse” in each declaration. If the relationship is in a different class, it is referred to by the relationships name followed by a double colon(::) and the name of the other relationship.

The relationship could be specified as:

```
class Book
{
    attribute string ISBNNO;
    attribute string TITLE;
    attribute integer PRICE;
    attribute string PUBLISHER;
    attribute enum CATEGORY
        {text,reference}BOOKTYPE;
    attribute struct AUTHORS
        {string fauthor, string sauthor, string
        tauthor} AUTHORLIST;
    relationship set <Student> receivedby
        inverse Student:::receives;
    relationship set <Supplier> suppliedby
        inverse Supplier::supplies;
};

class Student
{
    attribute string ENROLMENT_NO;
    attribute string NAME;
    attribute integer MARKS;
    attribute string COURSE;
    relationship set <Book> receives
        inverse Book::receivedby;
};

class Supplier
{
    attribute string SUPPLIER_ID;
    attribute string SUPPLIER_NAME;
    attribute string SUPPLIER_ADDRESS;
    attribute string SUPPLIER_CITY;
    relationship set <Book> supplies
        inverse Book::suppliedby;
};
```

Methods could be specified with the classes along with input/output types. These declarations are called “signatures”. These method parameters could be in, out or inout. Here, the first parameter is passed by value whereas the next two parameters are passed by reference. Exceptions could also be associated with these methods.

```
class Student
{
    attribute string ENROLMENT_NO;
    attribute string NAME;
    attribute string st_address;
    relationship set <book> receives
```



```
        inverse Book::receivedby;
void findcity(in set<string>,out set<string>)
    raises(notfoundcity);
};
```

In the method find city, the name of city is passed referenced, in order to find the name of the student who belongs to that specific city. In case blank is passed as parameter for city name then, the exception notfoundcity is raised.

The ODL could be atomic type or class names. The basic type uses many class constructors such as set, bag, list, array, dictionary and structure. We have shown the use of some in the example above. You may wish to refer to the further readings section.

Inheritance is implemented in ODL using subclasses with the keyword “extends”.

```
class Journal extends Book
{
    attribute string VOLUME;
    attribute string emailauthor1;
    attribute string emailauthor2;
};
```

Multiple inheritance is implemented by using extends separated by a colon (:). If there is a class Fee containing fees details then multiple inheritance could be shown as:

```
class StudentFeeDetail extends Student:Fee
{
    void deposit(in set <float>, out set <float>)
        raises(refundToBeDone)
};
```

Like the difference between relation schema and relation instance, ODL uses the class and its extent (set of existing objects). The objects are declared with the keyword “extent”.

```
class Student (extent firstStudent)
{
    attribute string ENROLMENT_NO;
    attribute string NAME;
    .....
};
```

It is not necessary in case of ODL to define keys for a class. But if one or more attributes have to be declared, then it may be done with the declaration on key for a class with the keyword “key”.

```
class student (extent firstStudent key ENROLMENT_NO)
{
    attribute string ENROLMENT_NO;
    attribute string NAME;
    .....
};
```

Assuming that the ENROLMENT_NO and ACCESSION_NO forms a key for the issue table then:

```
class Issue (extent thisMonthIssue key (ENROLMENT_NO,
ACCESSION_NO))
```



```
{
    attribute string ENROLMENT_NO;
    attribute string ACCESSION_NO;
    .....
};
```

The major considerations while converting ODL designs into relational designs are as follows:

- It is not essential to declare keys for a class in ODL but in Relational design now attributes have to be created in order for it to work as a key.
- Attributes in ODL could be declared as non-atomic whereas, in Relational design, they have to be converted into atomic attributes.
- Methods could be part of design in ODL but, they can not be directly converted into relational schema although, the SQL supports it, as it is not the property of a relational schema.
- Relationships are defined in inverse pairs for ODL but, in case of relational design, only one pair is defined.

For example, for the book class schema the relation is:

```
Book (ISBNNO, TITLE, CATEGORY, fauthor, sauthor, tauthor)
```

Thus, the ODL has been created with the features required to create an object oriented database in OODBMS. You can refer to the further readings for more details on it.

1.4.3 Object Query Language

Object Query Language (OQL) is a standard query language which takes high-level, declarative programming of SQL and object-oriented features of OOPs. Let us explain it with the help of examples.

Find the list of authors for the book titled “The suitable boy”

```
SELECT b.AUTHORS
FROM Book b
WHERE b.TITLE="The suitable boy"
```

The more complex query to display the title of the book which has been issued to the student whose name is Anand, could be

```
SELECT b.TITLE
FROM Book b, Student s
WHERE s.NAME ="Anand"
```

This query is also written in the form of relationship as

```
SELECT b.TITLE
FROM Book b
WHERE b.receivedby.NAME ="Anand"
```

In the previous case, the query creates a bag of strings, but when the keyword DISTINCT is used, the query returns a set.

```
SELECT DISTINCT b.TITLE
FROM Book b
```



```
WHERE b.receivedby.NAME = "Anand"
```

When we add ORDER BY clause it returns a list.

```
SELECT b.TITLE
FROM Book b
WHERE b.receivedby.NAME = "Anand"
ORDER BY b.CATEGORY
```

In case of complex output the keyword “Struct” is used. If we want to display the pair of titles from the same publishers then the proposed query is:

```
SELECT DISTINCT Struct(book1:b1,book2:b2)
FROM Book b1,Book b2
WHERE b1.PUBLISHER = b2.PUBLISHER
AND b1.ISBNNO < b2.ISBNNO
```

Aggregate operators like SUM, AVG, COUNT, MAX, MIN could be used in OQL. If we want to calculate the maximum marks obtained by any student then the OQL command is

```
Max(SELECT s.MARKS FROM Student s)
```

Group by is used with the set of structures, that are called “immediate collection”.

```
SELECT cour, publ,
       AVG(SELECT p.b.PRICE FROM partition
            p)
FROM Book b
GROUP BY cour:b.receivedby.COURSE, publ:b.PUBLISHER
```

HAVING is used to eliminate some of the groups created by the GROUP BY commands.

```
SELECT cour, publ,
       AVG(SELECT p.b.PRICE FROM partition
            p)
FROM Book b
GROUP BY cour:b.receivedby.COURSE, publ:b.PUBLISHER
HAVING AVG(SELECT p.b.PRICE FROM partition p)>=60
```

Union, intersection and difference operators are applied to set or bag type with the keyword UNION, INTERSECT and EXCEPT. If we want to display the details of suppliers from PATNA and SURAT then the OQL is

```
(SELECT DISTINCT su
FROM Supplier su
WHERE su.SUPPLIER_CITY="PATNA")
UNION
(SELECT DISTINCT su
FROM Supplier su
WHERE su.SUPPLIER_CITY="SURAT")
```

The result of the OQL expression could be assigned to host language variables. If, costlyBooks is a set <book> variable to store the list of books whose price is below Rs.200 then

```
costlyBooks = SELECT DISTINCT b
```



FROM Book b
WHERE b.PRICE > 200

In order to find a single element of the collection, the keyword “ELEMENT” is used.
If costlySBook is a variable then

```
costlySBook = ELEMENT (SELECT DISTINCT b
                           FROM Book b
                           WHERE b.PRICE > 200
                           )
```

The variable could be used to print the details a customised format.

```
bookDetails = SELECT DISTINCT b
               FROM Book b
               ORDER BY b.PUBLISHER,b.TITLE;
bookCount = COUNT(bookDetails);
for (i=0;i<bookCount;i++)
{
    nextBook = bookDetails[i];
    cout<<i<<"\t"<<nextBook.PUBLISHER <<"\t"<<
                           nextBook.TITLE<<"\n";
}
```

☛ Check Your Progress 2

- 1) Create a class staff using ODL that also references the Book class given in section 1.5.
-
.....
.....

- 2) What modifications would be needed in the Book class because of the table created by the above query?
-
.....
.....

- 3) Find the list of books that have been issued to “Shashi”.
-
.....
.....

1.5 IMPLEMENTATION OF OBJECT ORIENTED CONCEPTS IN DATABASE SYSTEMS

Database systems that support object oriented concepts can be implemented in the following ways:

- Extend the existing RDBMSs to include the object orientation; Or



- Create a new DBMS that is exclusively devoted to the Object oriented database.
- Let us discuss more about them.

1.5.1 The Basic Implementation Issues for Object-Relational Database Systems

The RDBMS technology has been enhanced over the period of last two decades. The RDBMS are based on the theory of relations and thus are developed on the basis of proven mathematical background. Hence, they can be proved to be working correctly. Thus, it may be a good idea to include the concepts of object orientation so that, they are able to support object-oriented technologies too. The first two concepts that were added include the concept of complex types, inheritance, and some newer types such as multisets and arrays. One of the key concerns in object-relational database are the storage of tables that would be needed to represent inherited tables, and representation for the newer types.

One of the ways of representing inherited tables may be to store the inherited primary key attributes along with the locally defined attributes. In such a case, to construct the complete details for the table, you need to take a join between the inherited table and the base class table.

The second possibility here would be, to allow the data to be stored in all the inherited as well as base tables. However, such a case will result in data replication. Also, you may find it difficult at the time of data insertion.

As far as arrays are concerned, since they have a fixed size their implementation is straight forward. However, the cases for the multiset would desire to follow the principle of normalisation in order to create a separate table which can be joined with the base table as and when required.

1.5.2 Implementation Issues of OODBMS

The database system consists of persistent data. To manipulate that data one must either use data manipulation commands or a host language like C using embedded command. However, a persistent language would require a seamless integration of language and persistent data.

Please note: The embedded language requires a lot many steps for the transfer of data from the database to local variables and vice-versa. The question is, can we implement an object oriented language such as C++ and Java to handle persistent data? Well a persistent object-orientation would need to address some of the following issues:

Object persistence: A practical approach for declaring a persistent object would be to design a construct that declares an object as persistent. The difficulty with this approach is that it needs to declare object persistence at the time of creation. An alternative of this approach may be to mark a persistent object during run time. An interesting approach here would be that once an object has been marked persistent then all the objects that are reachable from that object should also be persistent automatically.

Object Identity: All the objects created during the execution of an object oriented program would be given a system generated object identifier, however, these identifiers become useless once the program terminates. With the persistent objects it is necessary that such objects have meaningful object identifiers. Persistent object identifiers may be implemented using the concept of persistent pointers that remain valid even after the end of a program.



Storage and access: The data of each persistent object needs to be stored. One simple approach for this may be to store class member definitions and the implementation of methods as the database schema. The data of each object, however, needs to be stored individually along with the schema. A database of such objects may require the collection of the persistent pointers for all the objects of one database together. Another, more logical way may be to store the objects as collection types such as sets. Some object oriented database technologies also define a special collection as **class extent** that keeps track of the objects of a defined schema.

1.6 OODBMS VERSUS OBJECT RELATIONAL DATABASE

An object oriented database management system is created on the basis of persistent programming paradigm whereas, a object relational is built by creating object oriented extensions of a relational system. In fact both the products have clearly defined objectives. The following table shows the difference among them:

Object Relational DBMS	Object Oriented DBMS
The features of these DBMS include: <ul style="list-style-type: none"> Support for complex data types Powerful query languages support through SQL Good protection of data against programming errors 	The features of these DBMS include: <ul style="list-style-type: none"> Supports complex data types, Very high integration of database with the programming language, Very good performance But not as powerful at querying as Relational.
One of the major assets here is SQL. Although, SQL is not as powerful as a Programming Language, but it is none-the-less essentially a fourth generation language, thus, it provides excellent protection of data from the Programming errors.	It is based on object oriented programming languages, thus, are very strong in programming, however, any error of a data type made by a programmer may effect many users.
The relational model has a very rich foundation for query optimisation, which helps in reducing the time taken to execute a query.	These databases are still evolving in this direction. They have reasonable systems in place.
These databases make the querying as simple as in relational even, for complex data types and multimedia data.	The querying is possible but somewhat difficult to get.
Although the strength of these DBMS is SQL, it is also one of the major weaknesses from the performance point of view in memory applications.	Some applications that are primarily run in the RAM and require a large number of database accesses with high performance may find such DBMS more suitable. This is because of rich programming interface provided by such DBMS. However, such applications may not support very strong query capabilities. A typical example of one such application is databases required for CAD.

Check Your Progress 3

State True or False.

- 1) Object relational database cannot represent inheritance but can represent complex database types. T F
- 2) Persistence of data object is the same as storing them into files. T F
- 3) Object- identity is a major issue for object oriented database especially in the context of referencing the objects. T F



- | | |
|---|---|
| 4) The class extent defines the limit of a class. | T <input type="checkbox"/> F <input type="checkbox"/> |
| 5) The query language of object oriented DBMS is stronger than object relational databases. | T <input type="checkbox"/> F <input type="checkbox"/> |
| 6) SQL commands cannot be optimised. | T <input type="checkbox"/> F <input type="checkbox"/> |
| 7) Object oriented DBMS support very high integration of database with OOP. | T <input type="checkbox"/> F <input type="checkbox"/> |

1.7 SUMMARY

Object oriented technologies are one of the most popular technologies in the present era. Object orientation has also found its way into database technologies. The object oriented database systems allow representation of user defined types including operation on these types. They also allow representation of inheritance using both the type inheritance and the table inheritance. The idea here is to represent the whole range of newer types if needed. Such features help in enhancing the performance of a database application that would otherwise have many tables. SQL supports these features for object relational database systems.

The object definition languages and object query languages have been designed for the object oriented DBMS on the same lines as that of SQL. These languages tries to simplify various object related representations using OODBMS.

The object relational and object oriented databases do not compete with each other but have different kinds of applications areas. For example, relational and object relational DBMS are most suited for simple transaction management systems, while OODBMS may find applications with e-commerce, CAD and other similar complex applications.

1.8 SOLUTIONS/ANSWERS

Check Your Progress 1

- 1) The object oriented databases are need for:
 - Representing complex types.
 - Representing inheritance, polymorphism
 - Representing highly interrelated information
 - Providing object oriented solution to databases bringing them closer to OOP.
- 2) Primarily by representing it as a single attribute. All its components should also be referenced separately.
- 3)

```
CREATE TYPE Addrtype AS
(
    houseNo      CHAR(8),
    street       CHAR(10),
    colony        CHAR(10),
    city          CHAR(8),
    state CHAR(8),
    pincode      CHAR(6),
);
```



```

METHOD pin() RETURNS CHAR(6);
CREATE METHOD pin() RETURNS CHAR(6);
FOR Addrtype
BEGIN
    . . .
END
4)
CREATE TABLE address OF Addrtype
(
    REF IS addid SYSTEM GENERATED,
    PRIMARY KEY (houseNo,pincode)
);

```

- 5) The relationship can be established with multiple tables by specifying the keyword “SCOPE”. For example:

```

Create table mylibrary
{
    mybook REF(Book) SCOPE library;
    myStudent REF(Student) SCOPE student;
    mySupplier REF(Supplier) SCOPE supplier;
};

```

Check Your Progress 2

1)

```

class Staff
{
    attribute string STAFF_ID;
    attribute string STAFF_NAME;
    attribute string DESIGNATION;
    relationship set <Book> issues
        inverse Book::issuedto;
};

```

- 2) The Book class needs to represent the relationship that is with the Staff class. This would be added to it by using the following commands:

```

RELATIONSHIP SET <Staff> issuedto
    INVERSE :: issues Staff

```

- 3) SELECT DISTINCT b.TITLE
FROM BOOK b
WHERE b.issuedto.NAME = “Shashi”

Check Your Progress 3

- 1) False 2) False 3) True 4) False 5) False 6) False 7) True

For Unit 14 : Please read the following units of MCS-43 Block 3 Unit 3 & Unit 4

UNIT 3 INTRODUCTION TO DATA WAREHOUSING

Introduction to Data

Warehousing

Structure	Page Nos.
3.0 Introduction	59
3.1 Objectives	59
3.2 What is Data Warehousing?	60
3.3 The Data Warehouse: Components and Processes	62
3.3.1 Basic Components of a Data Warehouse	
3.3.2 Data Extraction, Transformation and Loading (ETL)	
3.4 Multidimensional Data Modeling for Data Warehouse	67
3.5 Business Intelligence and Data Warehousing	70
3.5.1 Decision Support System (DSS)	
3.5.2 Online Analytical Processing (OLAP)	
3.6 Building of Data Warehouse	73
3.7 Data Marts	75
3.8 Data Warehouse and Views	76
3.9 The Future: Open Issues for Data Warehouse	77
3.10 Summary	77
3.11 Solutions/Answers	78

3.0 INTRODUCTION

Information Technology (IT) has a major influence on organisational performance and competitive standing. With the ever increasing processing power and availability of sophisticated analytical tools and techniques, it has built a strong foundation for the product - data warehouse. But, why should an organisation consider investing in a data warehouse? One of the prime reasons, for deploying a data warehouse is that, the data warehouse is a kingpin of business intelligence.

The data warehouses provide storage, functionality and responsiveness to queries, that is far superior to the capabilities of today's transaction-oriented databases. In many applications, users only need read-access to data, however, they need to access larger volume of data very rapidly – much more than what can be conveniently handled by traditional database systems. Often, such data is extracted from multiple operational databases. Since, most of these analyses performed do occur periodically, therefore, software developers and software vendors try to design systems to support these functions. Thus, there is a definite need for providing decision makers at middle management level and higher level with information as per the level of details to support decision-making. The data warehousing, online analytical processing (OLAP) and data mining technologies provide this functionality.

This unit covers the basic features of data warehousing and OLAP. Data Mining has been discussed in more details in unit 4 of this Block.

3.1 OBJECTIVES

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

- explain the term data warehouse;
- define key concepts surrounding data warehousing systems;



- compare database warehouse with operational information systems;
- discuss data warehousing architecture;
- identify the main stages in the life cycle of data warehousing, and
- discuss the concepts of OLAP, MOLAP, ROLAP.

3.2 WHAT IS DATA WAREHOUSING?

Let us first try to answer the question: What is a data warehouse? A simple answer could be: A data warehouse is a tool that manages data *after and outside* of operational systems. Thus, they are not replacements for the operational systems but are major tools that acquire data from the operational system. Data warehousing technology has evolved in business applications for the process of strategic decision-making. Data warehouses may be sometimes considered as the key components of IT strategy and architecture of an organisation. We will give the more formal definition of data warehouse in the next paragraph.

A data warehouse as defined by **W.H. Inmon** is a *subject-oriented, integrated, nonvolatile, time-variant collection* of data that supports decision-making of the management. Data warehouses provide controlled access to data for complex analysis, knowledge discovery, and decision-making.

Figure 1 presents some uses of data warehousing in various industries

S.No.	Industry	Functional Areas of Use	Strategic Uses
1	Banking	Creating new schemes for loans and other banking products, helps in operations, identifies information for marketing	Finding trends for customer service, service promotions, reduction of expenses.
2	Airline	Operations, marketing	Crew assignment, aircraft maintenance plans, fare determination, analysis of route profitability, frequent - flyer program design
3	Hospital	Operation optimisation	Reduction of operational expenses, scheduling of resources
4	Investment and Insurance	Insurance product development, marketing	Risk management, financial market analysis, customer tendencies analysis, portfolio management

Figure 1: Uses of Data Warehousing

A data warehouse offers the following advantages:

- It provides historical information that can be used in many different forms of comparative and competitive analysis.
- It enhances the quality of the data and tries to make it complete.
- It can help in supporting disaster recovery although not alone but with other back up resources.



One of the major advantages a data warehouse offers is that it allows a large collection of historical data of many operational databases, which may be heterogeneous in nature, that can be analysed through one data warehouse interface, thus, it can be said to be a ONE STOP portal of historical information of an organisation. It can also be used in determining many trends through the use of data mining techniques.

Remember a data warehouse does not create value of its own in an organisation. However, the value can be generated by the users of the data of the data warehouse. For example, an electric billing company, by analysing data of a data warehouse can predict frauds and can reduce the cost of such determinations. In fact, this technology has such great potential that any company possessing proper analysis tools can benefit from it. Thus, a data warehouse supports Business Intelligence (that is), the technology that includes business models with objectives such as reducing operating costs, increasing profitability by improving productivity, sales, services and decision-making. Some of the basic questions that may be asked from a software that supports business intelligence include:

- What would be the income, expenses and profit for a year?
- What would be the sales amount this month?
- Who are the vendors for a product that is to be procured?
- How much of each product is manufactured in each production unit?
- How much is to be manufactured?
- What percentage of the product is defective?
- Are customers satisfied with the quality? etc.

Data warehouse supports various business intelligence applications. Some of these may be - online analytical processing (**OLAP**), decision-support systems (DSS), data mining etc. We shall be discussing these terms in more detail in the later sections.

A data warehouse has many characteristics. Let us define them in this section and explain some of these features in more details in the later sections.

Characteristics of Data Warehouses

Data warehouses have the following important features:

- 1) **Multidimensional conceptual view:** A data warehouse contains data of many operational systems, thus, instead of a simple table it can be represented in multidimensional data form. We have discussed this concept in more detail in section 3.3.
- 2) **Unlimited dimensions and unrestricted cross-dimensional operations:** Since the data is available in multidimensional form, it requires a schema that is different from the relational schema. Two popular schemas for data warehouse are discussed in section 3.3.
- 3) **Dynamic sparse matrix handling:** This is a feature that is much needed as it contains huge amount of data.
- 4) **Client/server architecture:** This feature help a data warehouse to be accessed in a controlled environment by multiple users.
- 5) **Accessibility and transparency, intuitive data manipulation and consistent reporting performance:** This is one of the major features of the data warehouse. A Data warehouse contains, huge amounts of data, however, that should not be the reason for bad performance or bad user interfaces. Since the objectives of data warehouse are clear, therefore, it has to support the following



easy to use interfaces, strong data manipulation, support for applying and reporting of various analyses and user-friendly output.

3.3 THE DATA WAREHOUSE: COMPONENTS AND PROCESSES

A data warehouse is defined as *subject-oriented, integrated, nonvolatile, time-variant collection*, but how can we achieve such a collection? To answer this question, let us define the basic architecture that helps a data warehouse achieve the objectives as given/stated above. We shall also discuss the various processes that are performed by these components on the data.

3.3.1 The Basic Components of a Data Warehouse

A data warehouse basically consists of three components:

The Data Sources

The ETL and

The Schema of data of data warehouse including meta data.

Figure 2 defines the basic architecture of a data warehouse. The analytical reports are not a part of the data warehouse but are one of the major business application areas including OLAP and DSS.

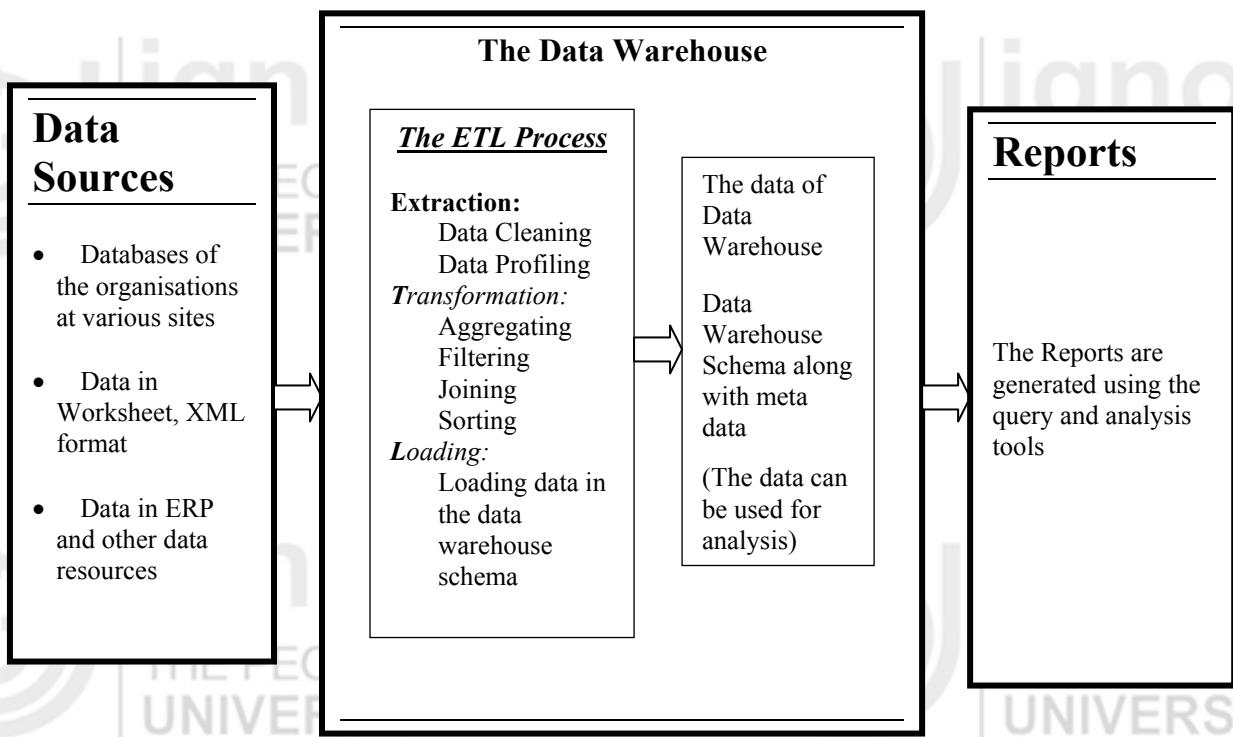


Figure 2: The Data Warehouse Architecture

The Data Sources

The data of the data warehouse can be obtained from many operational systems. A data warehouse interacts with the environment that provides most of the source data for the data warehouse. By the term environment, we mean, traditionally developed applications. In a large installation, hundreds or even thousands of these database systems or files based system exist with plenty of redundant data.



The warehouse database obtains most of its data from such different forms of legacy systems – files and databases. Data may also be sourced from external sources as well as other organisational systems, for example, an office system. This data needs to be integrated into the warehouse. But how do we integrate the data of these large numbers of operational systems to the data warehouse system? We need the help of ETL tools to do so. These tools capture the data that is required to be put in the data warehouse database. We shall discuss the ETL process in more detail in section 3.3.2.

Data of Data Warehouse

A data warehouse has an integrated, “subject-oriented”, “time-variant” and “non-volatile” collection of data. The basic characteristics of the data of a data warehouse can be described in the following way:

i) Integration: Integration means bringing together data of multiple, dissimilar operational sources on the basis of an enterprise data model. The enterprise data model can be a basic template that identifies and defines the organisation’s key data items uniquely. It also identifies the logical relationships between them ensuring organisation wide consistency in terms of:

Data naming and definition: Standardising for example, on the naming of “student enrolment number” across systems.

Encoding structures: Standardising on gender to be represented by “M” for male and “F” for female or that the first two digit of enrolment number would represent the year of admission.

Measurement of variables: A Standard is adopted for data relating to some measurements, for example, all the units will be expressed in metric system or all monetary details will be given in Indian Rupees.

ii) Subject Orientation: The second characteristic of the data warehouse’s data is that its design and structure can be oriented to important objects of the organisation. These objects such as STUDENT, PROGRAMME, REGIONAL CENTRES etc., are in contrast to its operational systems, which may be designed around applications and functions such as ADMISSION, EXAMINATION and RESULT DECLARATIONS (in the case of a University). Refer to *Figure 3*.

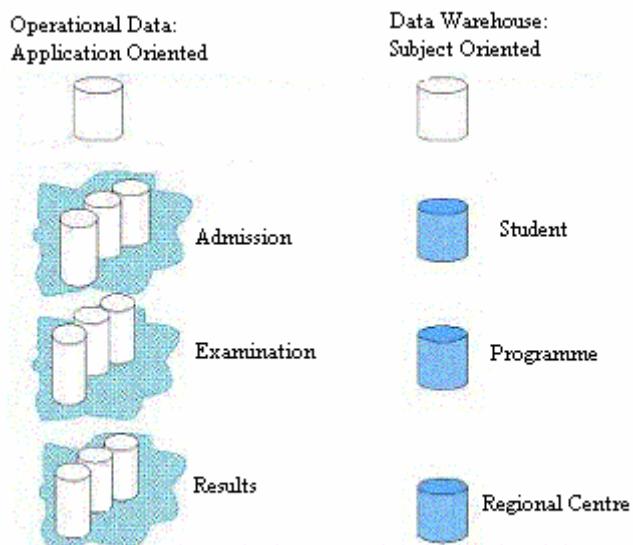


Figure 3: Operations system data orientation vs. Data warehouse data orientation

iii) Time-Variance: The third defining characteristic of the database of data warehouse is that it is time-variant, or historical in nature. The entire data in the data



warehouse is/was accurate at some point of time. This is, in contrast with operational data that changes over a shorter time period. The data warehouse's data contains data that is date-stamped, and which is historical data. *Figure 4* defines this characteristic of data warehouse.

OPERATIONAL DATA	DATA WAREHOUSE DATA
• It is the current value data	• Contains a snapshot of historical data
• Time span of data = 60-90 days	• Time span of data = 5-10 years or more
• Data can be updated in most cases	• After making a snapshot the data record cannot be updated
• May or may not have a timestamp	• Will always have a timestamp

Figure 4: Time variance characteristics of a data of data warehouse and operational data

iv) **Non-volatility (static nature) of data:** Data warehouse data is loaded on to the data warehouse database and is subsequently scanned and used, but is not updated in the same classical sense as operational system's data which is updated through the transaction processing cycles.

Decision Support and Analysis Tools

A data warehouse may support many OLAP and DSS tools. Such decision support applications would typically access the data warehouse database through a standard query language protocol; an example of such a language may be SQL. These applications may be of three categories: simple query and reporting, decision support systems and executive information systems. We will define them in more details in the later sections.

Meta Data Directory

The meta data directory component defines the repository of the information stored in the data warehouse. The meta data can be used by the general users as well as data administrators. It contains the following information:

- i) the structure of the contents of the data warehouse database,
- ii) the source of the data,
- iii) the data transformation processing requirements, such that, data can be passed from the legacy systems into the data warehouse database,
- iv) the process summarisation of data,
- v) the data extraction history, and
- vi) how the data needs to be extracted from the data warehouse.

Meta data has several roles to play and uses in the data warehouse system. For an end user, meta data directories also provide some additional information, such as what a particular data item would mean in business terms. It also identifies the information on reports, spreadsheets and queries related to the data of concern. All database management systems (DBMSs) have their own data dictionaries that serve a similar purpose. Information from the data dictionaries of the operational system forms a valuable source of information for the data warehouse's meta data directory.

3.3.2 Data Extraction, Transformation and Loading (ETL)



The first step in data warehousing is, to perform data extraction, transformation, and loading of data into the data warehouse. This is called ETL that is Extraction, Transformation, and Loading. ETL refers to the methods involved in accessing and manipulating data available in various sources and loading it into a target data warehouse. Initially the ETL was performed using SQL programs, however, now there are tools available for ETL processes. The manual ETL was complex as it required the creation of a complex code for extracting data from many sources. ETL tools are very powerful and offer many advantages over the manual ETL. ETL is a step-by-step process. As a first step, it maps the data structure of a source system to the structure in the target data warehousing system. In the second step, it cleans up the data using the process of data transformation and finally, it loads the data into the target system.

What happens during the ETL Process?

The ETL is three-stage process. During the *Extraction* phase the desired data is identified and extracted from many different sources. These sources may be different databases or non-databases. Sometimes when it is difficult to identify the desirable data then, more data than necessary is extracted. This is followed by the identification of the relevant data from the extracted data. The process of extraction sometimes, may involve some basic transformation. For example, if the data is being extracted from two Sales databases where the sales in one of the databases is in Dollars and in the other in Rupees, then, simple transformation would be required in the data. The size of the extracted data may vary from several hundreds of kilobytes to hundreds of gigabytes, depending on the data sources and business systems. Even the time frame for the extracted data may vary, that is, in some data warehouses, data extraction may take a few days or hours to a real time data update. For example, a situation where the volume of extracted data even in real time may be very high is a web server.

The *extraction* process involves data cleansing and data profiling. Data cleansing can be defined as the process of removal of inconsistencies among the data. For example, the state name may be written in many ways also they can be misspelt too. For example, the state Uttar Pradesh may be written as U.P., UP, Uttar Pradesh, Utter Pradesh etc. The cleansing process may try to correct the spellings as well as resolve such inconsistencies. But how does the cleansing process do that? One simple way may be, to create a Database of the States with some possible fuzzy matching algorithms that may map various variants into one state name. Thus, cleansing the data to a great extent. Data profiling involves creating the necessary data from the point of view of data warehouse application. Another concern here is to eliminate duplicate data. For example, an address list collected from different sources may be merged as well as purged to create an address profile with no duplicate data.

One of the most time-consuming tasks - data *transformation* and *loading* follows the extraction stage. This process includes the following:

- Use of data filters,
- Data validation against the existing data,
- Checking of data duplication, and
- Information aggregation.

Transformations are useful for transforming the source data according to the requirements of the data warehouse. The process of transformation should ensure the quality of the data that needs to be loaded into the target data warehouse. Some of the common transformations are:



Filter Transformation: Filter transformations are used to filter the rows in a mapping that do not meet specific conditions. For example, the list of employees of the Sales department who made sales above Rs.50,000/- may be filtered out.

Joiner Transformation: This transformation is used to join the data of one or more different tables that may be stored on two different locations and could belong to two different sources of data that may be relational or from any other sources like XML data.

Aggregator Transformation: Such transformations perform aggregate calculations on the extracted data. Some such calculations may be to find the sum or average.

Sorting Transformation: requires creating an order in the required data, based on the application requirements of the data warehouse.

Once the data of the data warehouse is properly extracted and transformed, it is *loaded* into a data warehouse. This process requires the creation and execution of programs that perform this task. One of the key concerns here is to propagate updates. Some times, this problem is equated to the problem of maintenance of the materialised views.

When should we perform the ETL process for data warehouse? ETL process should normally be performed during the night or at such times when the load on the operational systems is low. **Please note** that, the integrity of the extracted data can be ensured by synchronising the different operational applications feeding the data warehouse and the data of the data warehouse.

☛ Check Your Progress 1

1) What is a Data Warehouse?

-
.....
.....

3) What are the important characteristics of Data Warehousing?

-
.....
.....

4) Name the component that comprise the data warehouse architecture?



3.4 MULTIDIMENSIONAL DATA MODELING FOR DATA WAREHOUSING

A data warehouse is a huge collection of data. Such data may involve grouping of data on multiple attributes. For example, the enrolment data of the students of a University may be represented using a student schema such as:

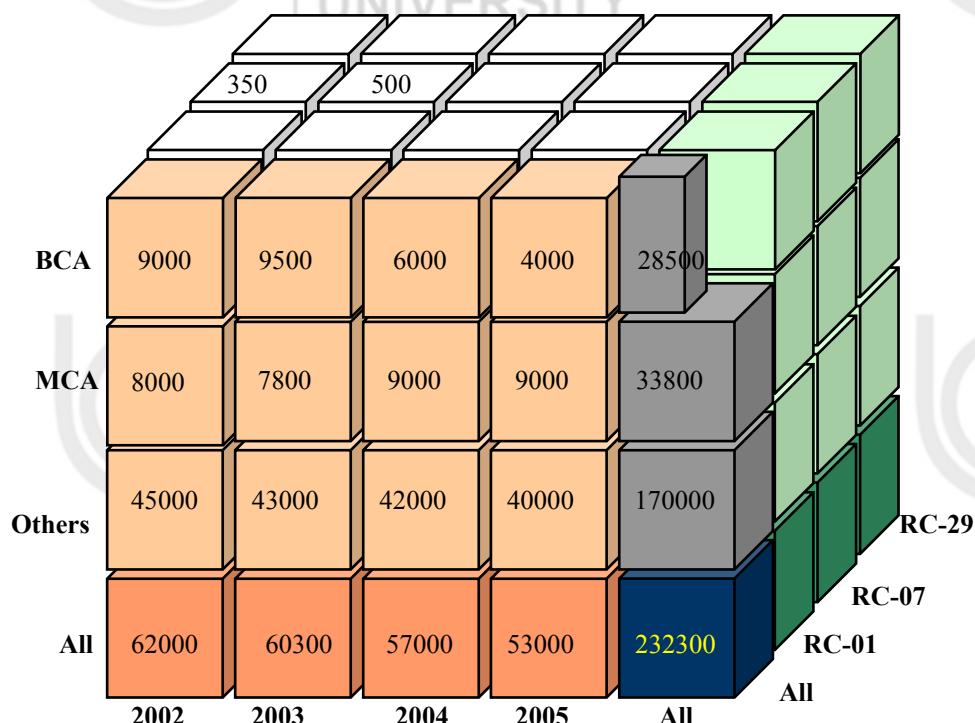
Student_enrolment (year, programme, region, number)

Here, some typical data value may be (These values are shown in *Figure 5* also. Although, in an actual situation almost all the values will be filled up):

- In the year 2002, BCA enrolment at Region (Regional Centre Code) RC-07 (Delhi) was 350.
- In year 2003 BCA enrolment at Region RC-07 was 500.
- In year 2002 MCA enrolment at all the regions was 8000.

Please note that, to define the student number here, we need to refer to three attributes: the year, programme and the region. Each of these attributes is identified as the dimension attributes. Thus, the data of student_enrolment table can be modeled using dimension attributes (year, programme, region) and a measure attribute (number). Such kind of data is referred to as a Multidimensional data. Thus, a data warehouse may use multidimensional matrices referred to as a data cubes model. The multidimensional data of a corporate data warehouse, for example, would have the fiscal period, product and branch dimensions. If the dimensions of the matrix are greater than three, then it is called a hypercube. Query performance in multidimensional matrices that lend themselves to dimensional formatting can be much better than that of the relational data model.

The following figure represents the Multidimensional data of a University:





Multidimensional data may be a little difficult to analyse. Therefore, Multidimensional data may be displayed on a certain pivot, for example, consider the following table:

Region: ALL THE REGIONS				
	BCA	MCA	Others	All the Programmes
2002	9000	8000	45000	62000
2003	9500	7800	43000	60300
2004	6000	9000	42000	57000
2005	4000	9000	40000	53000
ALL the Years	28500	33800	170000	232300

The table given above, shows, the multidimensional data in *cross-tabulation*. This is also referred to as a *pivot-table*. Please note that cross-tabulation is done on any two dimensions keeping the other dimensions fixed as ALL. For example, the table above has two dimensions Year and Programme, the third dimension Region has a fixed value ALL for the given table.

Please note that, the cross-tabulation as we have shown in the table above is, different to a relation. The relational representation for the data of the table above may be:

Table: Relational form for the Cross table as above

Year	Programme	Region	Number
2002	BCA	All	9000
2002	MCA	All	8000
2002	Others	All	45000
2002	All	All	62000
2003	BCA	All	9500
2003	MCA	All	7800
2003	Others	All	43000
2003	All	All	60300
2004	BCA	All	6000
2004	MCA	All	9000
2004	Others	All	42000
2004	All	All	57000
2005	BCA	All	4000
2005	MCA	All	9000
2005	Others	All	40000
2005	All	All	53000
All	BCA	All	28500
All	MCA	All	33800
All	Others	All	170000
All	All	All	232300



A cross tabulation can be performed on any two dimensions. The operation of changing the dimensions in a cross tabulation is termed as pivoting. In case a cross tabulation is done for a value other than ALL for the fixed third dimension, then it is called *slicing*. For example, a slice can be created for Region code RC-07 instead of ALL the regions in the cross tabulation of regions. This operation is called *dicing* if values of multiple dimensions are fixed.

Multidimensional data allows data to be displayed at various level of granularity. An operation that converts data with a fine granularity to coarse granularity using aggregation is, termed *rollup* operation. For example, creating the cross tabulation for All regions is a rollup operation. On the other hand an operation that moves from a coarse granularity to fine granularity is known as *drill down* operation. For example, moving from the cross tabulation on All regions back to Multidimensional data is a drill down operation. **Please note:** For the drill down operation, we need, the original data or any finer granular data.

Now, the question is, how can multidimensional data be represented in a data warehouse? or, more formally, what is the schema for multidimensional data?

Two common multidimensional schemas are the star schema and the snowflake schema. Let us, describe these two schemas in more detail. A multidimensional storage model contains two types of tables: the dimension tables and the fact table. The dimension tables have tuples of dimension attributes, whereas the fact tables have one tuple each for a recorded fact. In order to relate a fact to a dimension, we may have to use pointers. Let us demonstrate this with the help of an example. Consider the University data warehouse where one of the data tables is the **Student enrolment table**. The three dimensions in such a case would be:

- Year
- Programme, and
- Region

The star schema for such a data is shown in *Figure 6*.

**Dimension Table:
Programme**

ProgramCode
Name
Duration

**Fact Table:
Enrolment**

Year
Programme
Region
Enrolment

**Dimension Table:
Year**

Year
Semester
Start date
.

**Dimension Table:
Region**

RCCode
RCname
RCaddress
RCphone

Figure 6: A Star Schema



Please note that in *Figure 6*, the fact table points to different dimension tables, thus, ensuring the reliability of the data. Please notice that, each Dimension table is a table for a single dimension only and that is why this schema is known as a star schema. However, a dimension table may not be normalised. Thus, a new schema named the snowflake schema was created. A snowflake schema has normalised but hierarchical dimensional tables. For example, consider the star schema shown in *Figure 6*, if in the Region dimension table, the value of the field Rcphone is multivalued, then the Region dimension table is not normalised.

Thus, we can create a snowflake schema for such a situation as:

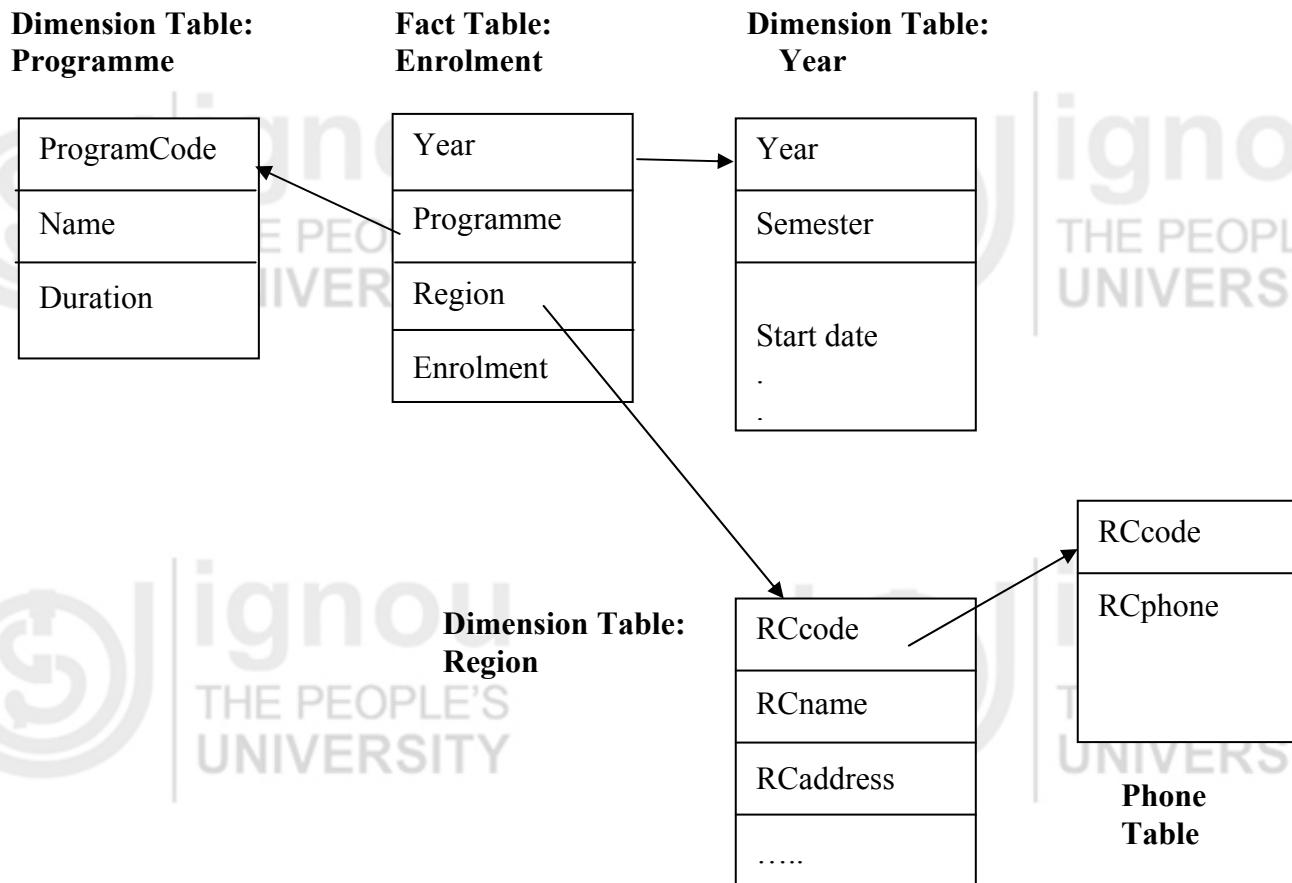


Figure 7: Snowflake Schema

Data warehouse storage can also utilise indexing to support high performance access. Dimensional data can be indexed in star schema to tuples in the fact table by using a join index. Data warehouse storage facilitates access to summary data due to the non-volatile nature of the data.

3.5 BUSINESS INTELLIGENCE AND DATA WAREHOUSING

A data warehouse is an integrated collection of data and can help the process of making better business decisions. Several tools and methods are available to that enhances advantage of the data of data warehouse to create information and knowledge that supports business decisions. Two such techniques are Decision-support systems and online analytical processing. Let us discuss these two in more details in this section.

3.5.1 Decision Support Systems (DSS)



The DSS is a decision support system and NOT a decision-making system. DSS is a specific class of computerised information systems that support the decision-making activities of an organisation. A properly designed DSS is an *interactive* software based system that helps decision makers to compile useful information from raw data, documents, personal knowledge, and/or business models to identify and solve problems and make decisions.

A decision support system may gather or present the following information:

- Accessing current information from data warehouse or legacy databases or other data resources.
- Presenting comparative sales figures of several months for an organisation.
- Creating projected revenue details based on new product sales assumptions.
- Demonstrating the consequences of different decision alternatives based on past experiences.

The DSS assists users in evaluating appropriate analysis or performing different types of studies on the datasets. For example, a spreadsheet can be used to store answers to a series of questionnaires in the form of Excel spreadsheets. This information then, can be passed on to decision makers. More specifically, the feedback data collected on a programme like CIC may be given to subject matter experts for making decisions on the quality, improvement, and revision of that programme. The DSS approach provides a self-assessment weighing tool to facilitate the determining of the value of different types of quality and quantity attributes. Decision support systems are sometimes also referred to as the Executive Information Systems (EIS).

Executive Information System (EIS): Executive information systems (EIS) are created for purpose of providing executives with the information they require to run their businesses. An EIS is intended to facilitate and support information and decision-making at senior executives level by, providing easy access to both *internal and external* information. Of course, this information should be relevant and should help them in establishing and meeting the strategic goals of the organisation.

The emphasis of DSS/EIS is mainly on graphical displays and easy-to-use user interfaces as they are there chiefly, to provide help. They offer strong reporting and drill-down capabilities. In general, EIS are enterprise-wide DSS that help top-level executives analyse, compare, and bring to light trends in important market/operational variables so that, they can monitor performance and identify opportunities and future problems. EIS and data warehousing technologies converge are convergent.

The concept of providing information to the executive management is not a new concept except for the ease with which they can get it. Given that top management has succeeded in acquiring the information till date, they can run their business without direct access to computer-based information systems. So why does one need a DSS/EIS? Well, there are a number of factors in support of DSS/EIS. These seem to be more managerial in nature. Some of these factors are:

- The first factor is a strange but true ‘pull’ factor, that is, executives are suggested to be more computer-literate and willing to become direct users of computer systems. For example, a survey suggests that more than twenty percent of senior executives have computers on their desks but rarely 5% use the system, although there are wide variations in the estimates yet, there is a define pull towards this simple easy to use technology.
- The other factor may be the increased use of computers at the executive level. For example, it has been suggested that middle managers who have been directly using computers in their daily work are being promoted to the executive level.



This new breed of executives do not exhibit the fear of computer technology that has characterised executive management up to now and are quite willing to be direct users of computer technology.

- The last factor is more on the side of technology. Technology is gradually becoming extremely simple to use from the end users point of view and it is now finding more users attracted towards it.

3.5.2 Online Analytical Processing (OLAP)

Data warehouses are not suitably designed for transaction processing, however, they support increased efficiency in query processing. Therefore, a data warehouse is a very useful support for the analysis of data. But are there any such tools that can utilise the data warehouse to extract useful analytical information?

On Line Analytical Processing (OLAP) is an approach for performing analytical queries and statistical analysis of multidimensional data. OLAP tools can be put in the category of business intelligence tools along with data mining. Some of the typical applications of OLAP may include reporting of sales projections, judging the performance of a business, budgeting and forecasting etc.

OLAP tools require multidimensional data and distributed query-processing capabilities. Thus, OLAP has data warehouse as its major source of information and query processing. But how do OLAP tools work?

In an OLAP system a data analyst would like to see different cross tabulations by interactively selecting the required attributes. Thus, the queries in an OLAP are expected to be executed extremely quickly. The basic data model that may be supported by OLAP is the star schema, whereas, the OLAP tool may be compatible to a data warehouse.

Let us, try to give an example on how OLAP is more suitable to a data warehouse rather than to a relational database. An OLAP creates an aggregation of information, for example, the sales figures of a sales person can be grouped (aggregated) for a product and a period. This data can also be grouped for sales projection of the sales person over the regions (North, South) or states or cities. Thus, producing enormous amount of aggregated data. If we use a relational database, we would be generating such data many times. However, this data has many dimensions so it is an ideal candidate for representation through a data warehouse. The OLAP tool thus, can be used directly on the data of the data warehouse to answer many analytical queries in a short time span. The term OLAP is sometimes confused with OLTP. OLTP is online transaction processing. OLTP systems focus on highly concurrent transactions and better commit protocols that support high rate of update transactions. On the other hand, OLAP focuses on good query-evaluation and query-optimisation algorithms.

OLAP Implementation

This classical form of OLAP implementation uses multidimensional arrays in the memory to store multidimensional data. Such implementation of OLAP is also referred to as Multidimensional OLAP (MOLAP). MOLAP is faster as it stores data in an already processed aggregated data form using dimension and fact tables. The other important type of OLAP implementation is Relational OLAP (ROLAP), which stores data directly in the relational databases. ROLAP creates multidimensional views upon request rather than in advance as in MOLAP. ROLAP may be used on complex data with a wide number of fields.

3.6 BUILDING OF DATA WAREHOUSE



The first basic issue for building a data warehouse is to identify the USE of the data warehouse. It should include information on the expected outcomes of the design. A good data warehouse must support meaningful query facility on the attributes of dimensional and fact tables. A data warehouse design in addition to the design of the schema of the database has to address the following three issues:

- How will the data be acquired?
- How will the data be stored?
- What would be the environment of the data warehouse?

Some of the key concerns of the issues above are:

Data Acquisition: A data warehouse must acquire data so that it can fulfil the required objectives. Some of the key issues for data acquisition are:

- Whether the data is to be extracted from multiple, heterogeneous sources? The location of these sources and the kind of data they contain?
- The method of acquiring the data contained in the various sources in a standard data warehouse schema. Remember, you must have consistent data in a data warehouse.
- How will the data be cleaned so that its validity can be ensured?
- How is the data going to be transformed and converted into the data warehouse multidimensional schema model?
- How will the data be loaded in the warehouse. After all, the data is huge and the amount of time the loading will take needs to be ascertained? Here, we need to find the time required for data cleaning, formatting, transmitting, creating additional indexes etc. and also the issues related to data consistency such as, the currency of data, data integrity in multidimensional space, etc.

Data storage: The data acquired by the data is also to be stored as per the storage schema. This data should be easily accessible and should fulfil the query needs of the users efficiently. Thus, designers need to ensure that there are appropriate indexes or paths that allow suitable data access. Data storage must be updated as more data is acquired by the data warehouse, but it should still provide access to data during this time. Data storage also needs to address the issue of refreshing a part of the data of the data warehouse and purging data from the data warehouse.

Environment of the data warehouse: Data warehouse designers should also keep in mind the data warehouse environment considerations. The designers must find the expected use of the data and predict if it is consistent with the schema design. Another key issue here would be the design of meta data directory component of the data warehouse. The design should be such that it should be maintainable under the environmental changes.

DATA WAREHOUSING LIFE CYCLE

The data warehouse technologies use very diverse vocabulary. Although the vocabulary of data warehouse may vary for different organisations, the data warehousing industry is in agreement with the fact that the data warehouse lifecycle model fundamentally can be defined as the model consisting of five major phases – design, prototype, deploy, operation and enhancement.

Let us introduce these terms:

- 1) **Design:** The design of database is to be done for available data inventories, DSS analyst requirements and analytical needs. It needs to produce a robust star schema or snowflake schema. Key activities in the design phase may include



communication with the end users, finding the available catalogues, defining key performance and quality indicators, mapping of decision-making processes as per the information needs at various end user levels, logical and physical schema design etc.

- 2) **Prototype:** A data warehouse is a high cost project, thus, it may be a good idea to deploy it partially for a select group of decision-makers and database practitioners in the end user communities. This will help in developing a system that will be easy to accept and will be mostly as per the user's requirements.
- 3) **Deploy:** Once the prototype is approved, then the data warehouse can be put to actual use. A deployed data warehouse comes with the following processes; documentation, training, maintenance.
- 4) **Operation:** Once deployed the data warehouse is to be used for day-to-day operations. The operation in data warehouse includes extracting data, putting it in database and output of information by DSS.
- 5) **Enhancement:** These are needed with the updating of technology, operating processes, schema improvements etc. to accommodate the change.

Please note you can apply any software life cycle model on the warehouse life cycle.

Data Warehouse Implementation

After the design of the data warehouse, the next step for building the data warehouse may be its implementation. Please remember that implementing a data warehouse is a very challenging process. It tests the ability of an organisation to adjust to change. The implementation of the data warehouse may require the following stages:

Implementation of Data Model: The data model that is to be implemented should be checked to ensure that it has the key entities and their interrelationships. It also should see that the system records of the data warehouse must be as per the data warehouse data model and should be possible best matches for the operational system data. The physical design should support the schema design.

Implementation of Data Transformation Programs: Now, the transformation programs that will extract and transform the data from the system of record should be implemented. They should be tested to load the required data in the data warehouse database.

Populating Data Warehouse and Maintenance: Once the data transformation programs are found to be ok, they can be used to populate the data warehouse. Once the data warehouse is operation at it needs to be maintained properly.

Some general Issues for Warehouse Design and Implementation

The programs created during the previous phase are executed to populate the data warehouse's database.

The Development and Implementation Team: A core team for such implementation may be:

A Project Leader responsible for managing the overall project and the one who helps in obtaining resources and participates in the design sessions.

Analysts documents the end user requirements and creates the enterprise data models for the data warehouse.

A Data Base Administrator is responsible for the physical data base creation, and



Programmers responsible for programming the data extraction and transformation programs and end user access applications.

Training: Training will be required not only for end users, once the data warehouse is in place, but also for various team members during the development stages of the data warehouse.

☛ Check Your Progress 2

- 1) What is a dimension, how is it different from a fact table?

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

- 2) How is snowflake schema different from other schemes?

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

- 3) What are the key concerns when building a data warehouse?

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

- 4) What are the major issues related to data warehouse implementation?

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

- 5) Define the terms: DSS and ESS.

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

- 6) What are OLAP, MOLAP and ROLAP?

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

3.7 DATA MARTS

Data marts can be considered as the database or collection of databases that are designed to help managers in making strategic decisions about business and the organisation. Data marts are usually smaller than data warehouse as they focus on some subject or a department of an organisation (a data warehouse combines databases across an entire enterprise). Some data marts are also called dependent data marts and may be the subsets of larger data warehouses.

A data mart is like a data warehouse and contains operational data that helps in making strategic decisions in an organisation. The only difference between the two is



that data marts are created for a certain limited predefined application. Even in a data mart, the data is huge and from several operational systems, therefore, they also need a multinational data model. In fact, the star schema is also one of the popular schema choices for a data mart.

A dependent data mart can be considered to be a logical subset (view) or a physical subset (extraction) of a large data warehouse. A dependent data mart may be isolated for the following reasons.

- (i) For making a separate schema for OLAP or any other similar system.
- (ii) To put a portion of the data warehouse or a separate machine to enhance performance.
- (iii) To create a highly secure subset of data warehouse.

In fact, to standardise data analysis and usage patterns, data warehouses are generally organised as task specific small units the data marts. The data organisation of a data mart is a very simple star schema. For example, the university data warehouse that we discussed in section 3.4 can actually be a data mart on the problem “The prediction of student enrolments for the next year.” A simple data mart may extract its contents directly from operational databases. However, in complex multilevel data warehouse architectures the data mart content may be loaded with the help of the warehouse database and Meta data directories.

3.8 DATA WAREHOUSE AND VIEWS

Many database developers classify data warehouse as an extension of a view mechanism. If that is the case, then how do these two mechanisms differ from one another? For, after all even in a database warehouse, a view can be materialised for the purpose of query optimisation. A data warehouse may differ from a view in the following ways:

- A data warehouse has a multi-dimensional schema and tries to integrate data through fact-dimension star schema, whereas views on the other hand are relational in nature.
- Data warehouse extracts and transforms and then stores the data into its schema; however, views are only logical and may not be materialised.
- You can apply mechanisms for data access in an enhanced way in a data warehouse, however, that is not the case for a view.
- Data warehouse data is time-stamped, may be differentiated from older versions, thus, it can represent historical data. Views on the other hand are dependent on the underlying DBMS.
- Data warehouse can provide extended decision support functionality, views normally do not do it automatically unless, an application is designed for it.

3.9 THE FUTURE: OPEN ISSUE FOR DATA WAREHOUSE



The administration of a data warehouse is a complex and challenging task. Some of the open issues for data warehouse may be:

- Quality control of data despite having filtration of data.
- Use of heterogeneous data origins is still a major problem for data extraction and transformation.
- During the lifetime of the data warehouse it will change, hence, management is one of the key issues here.
- Data warehouse administration is a very wide area and requires diverse skills, thus, people need to be suitably trained.
- Managing the resources of a data warehouse would require a large distributed team.
- The key research areas in data warehouse is, data cleaning, indexing, view creation, queries optimisation etc.

However, data warehouses are still an expensive solution and are typically found in large firms. The development of a central warehouse is capital intensive with high risks. Thus, at present data marts may be a better choice.

☛ Check Your Progress 3

1) How is data mart different from data warehouse?

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

2) How does data warehouse differ from materialised views?

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

3.10 SUMMARY

This unit provided an introduction to the concepts of data warehousing systems. The data warehouse is a technology that collects operational data from several operational systems, refines it and stores it in its own multidimensional model such as star schema or snowflake schema. The data of a data warehouse can be indexed and can be used for analyses through various DSS and EIS. The architecture of data warehouse supports contains – an interface that interact with operational system, transformation processing, database, middleware and DSS interface at the other end. However, data warehouse architecture is incomplete if, it does not have meta data directory which is extremely useful for each and every step of the data warehouse. The life cycle of a data warehouse has several stages for designing, prototyping, deploying and maintenance. The database warehouse's life cycle, however, can be clubbed with SDLC. Data mart is a smaller version of a data warehouse designed for a specific purpose. Data warehouse is quite different from views. A data warehouse is complex and offers many challenges and open issues, but, in the future data warehouses will be-extremely important technology that will be deployed for DSS. Please go through further readings for more details on data warehouse.



3.11 SOLUTIONS/ANSWERS

Check Your Progress 1

- 1) A Data Warehouse can be considered to be a “corporate memory”. It is a repository of processed but integrated information that can be used for queries and analysis. Data and information are extracted from heterogeneous sources as they are generated. Academically, it is subject – oriented, time-variant, and a collection of operational data.

Relational databases are designed in general, for on-line transactional processing (OLTP) and do not meet the requirements for effective on-line analytical processing (OLAP). The data warehouses are designed differently from relational databases and are suitable for OLAP.

- 2). ETL is Extraction, transformation, and loading. ETL refers to the methods involved in accessing and manipulating data available in various sources and loading it into target data warehouse. The following are some of the transformations that may be used during ETL:

- Filter Transformation
- Joiner Transformation
- Aggregator transformation
- Sorting transformation.

- 3) Some important characteristics of Data Warehousing are
 - i) Multidimensional view
 - ii) Unlimited dimensions and aggregation levels and unrestricted cross-dimensional operations.
 - iii) Dynamic sparse matrix handling
 - iv) Client/server architecture
 - v) Accessibility and transparency, intuitive data manipulation and consistent reporting performance.

- 4) The data warehouse architecture consists of six distinct components that include:
 - i) Operational systems
 - ii) Transformation processing
 - iii) Database
 - iv) Middleware
 - v) Decision support and presentation processing and
 - vi) Meta data directory.

Check Your Progress 2

- 1) A dimension may be equated with an object. For example, in a sales organisation, the dimensions may be salesperson, product and period of a quarterly information. Each of these is a dimension. The fact table will represent the fact relating to the dimensions. For the dimensions as above, a fact table may include sale (in rupees) made by a typical sales person for the specific product for a specific period. This will be an actual date, thus is a fact. A fact, thus, represents an aggregation of relational data on the dimensions.
- 2) The primary difference lies in representing a normalised dimensional table.



- 3)
 - How will the data be acquired?
 - How will it be stored?
 - The type of environment in which the data warehouse will be implemented?
- 4)
 - Creation of proper transformation programs
 - Proper training of development team
 - Training of data warehouse administrator and end users
 - Data warehouse maintenance.
- 5) The DSS is a decision support system and not a decision-making system. It is a specific class of information system that supports business and organisational decision-making activities. A DSS is an interactive software-based system that helps decision makers compile useful information from raw data or documents, personal knowledge, etc. This information helps these decision makers to identify and solve problems and take decisions.

An Executive Information System (EIS) facilitates the information and decision making needs of senior executives. They provide easy access to relevant information (both internal as well as external), towards meeting the strategic goals of the organisation. These are the same as for the DSS.

- 6) OLAP refers to the statistical processing of multidimensional such that the results may be used for decision-making. MOLAP and ROLAP are the two implementations of the OLAP. In MOLAP the data is stored in the multidimensional form in the memory whereas in the ROLAP it is stored in the relational database form.

Check Your Progress 3

- 1) The basic constructs used to design a data warehouse and a data mart are the same. However, a Data Warehouse is designed for the enterprise level, while Data Marts may be designed for a business division/department level. A data mart contains the required subject specific data for local analysis only.
- 2) The difference may be:
 - Data warehouse has a multi-dimensional schema whereas views are relational in nature.
 - Data warehouse extracts and transforms and then stores the data into its schema that is not true for the materialised views.
 - Materialised views need to be upgraded on any update, whereas, a data warehouse does not need updation.
 - Data warehouse data is time-stamped, thus, can be differentiated from older versions that is not true for the materialised views.

For Unit 14 : Please read the following units of MCS-43 Block 3 Unit 3 & Unit 4

UNIT 4 INTRODUCTION TO DATA MINING

Structure

	Page Nos.
4.0 Introduction	80
4.1 Objectives	80
4.2 Data Mining Technology	81
4.2.1 Data, Information, Knowledge	
4.2.2 Sample Data Mining Problems	
4.2.3 Database Processing vs. Data Mining Processing	
4.2.4 Data Mining vs KDD	
4.3 Approaches to Data Mining Problems	84
4.4 Classification	85
4.4.1 Classification Approach	
4.4.2 Classification Using Distance (K-Nearest Neighbours)	
4.4.3 Decision or Classification Tree	
4.4.4 Bayesian Classification	
4.5 Clustering	93
4.5.1 Partitioning Clustering	
4.5.2 Nearest Neighbours Clustering	
4.5.2 Hierarchical Clustering	
4.6 Association Rule Mining	96
4.7 Applications of Data Mining Problem	99
4.8 Commercial Tools of Data Mining	100
4.9 Summary	102
4.10 Solutions/Answers	102
4.11 Further Readings	103

4.0 INTRODUCTION

Data mining is emerging as a rapidly growing interdisciplinary field that takes its approach from different areas like, databases, statistics, artificial intelligence and data structures in order to extract hidden knowledge from large volumes of data. The data mining concept is now a days not only used by the research community but also a lot of companies are using it for predictions so that, they can compete and stay ahead of their competitors.

With rapid computerisation in the past two decades, almost all organisations have collected huge amounts of data in their databases. These organisations need to understand their data and also want to discover useful knowledge as patterns, from their existing data.

This unit aims at giving you some of the fundamental techniques used in data mining. This unit emphasises on a brief overview of data mining as well as the application of data mining techniques to the real world. We will only consider structured data as input in this unit. We will emphasise on three techniques of data mining:

- (a) Classification,
- (b) Clustering, and
- (c) Association rules.

4.1 OBJECTIVES

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

- explain what is data mining;
- explain how data mining is applied in the real world;
- define the different approaches to data mining;
- use the classification approach in data mining;

- use the clustering approach;
- explain how association rules are used in data mining, and
- identify some of the leading data mining tools.

4.2 DATA MINING TECHNOLOGY

Data is growing at a phenomenal rate today and the users expect more sophisticated information from this data. There is need for new techniques and tools that can automatically generate useful information and knowledge from large volumes of data. Data mining is one such technique of generating hidden information from the data. Data mining can be defined as: “an automatic process of extraction of non-trivial or implicit or previously unknown but potentially useful information or patterns from data in large databases, data warehouses or in flat files”.

Data mining is related to data warehouse in this respect that, a data warehouse is well equipped for providing data as input for the data mining process. The advantages of using the data of data warehouse for data mining are or many some of them are listed below:

- Data quality and consistency are essential for data mining, to ensure, the accuracy of the predictive models. In data warehouses, before loading the data, it is first extracted, cleaned and transformed. We will get good results only if we have good quality data.
- Data warehouse consists of data from multiple sources. The data in data warehouses is integrated and subject oriented data. The data mining process performed on this data.
- In data mining, it may be the case that, the required data may be aggregated or summarised data. This is already there in the data warehouse.
- Data warehouse provides the capability of analysing data by using OLAP operations. Thus, the results of a data mining study can be analysed for hitherto, uncovered patterns.

As defined earlier, data mining generates potentially useful information or patterns from data. In fact, the information generated through data mining can be used to create knowledge. So let us, first, define the three terms data, information and knowledge.

4.2.1 Data, Information, Knowledge

Before going into details on data mining, let us, first, try to discuss the differences between data, information and knowledge.

1. **Data (Symbols):** It simply exists. It has no significance beyond its existence. It is raw information. For example, “it is raining”.
2. **Information:** Information is the processed data. It provides answer to “who”, “what”, “where”, and “when” questions. For example, “The temperature dropped 12 degrees centigrade and then it started raining” is an example of information.
3. **Knowledge:** Knowledge is the application of data and information and it answers the “how” questions. This is not explicit in the database - it is implicit. For example “If humidity is very high and the temperature drops suddenly, then, the atmosphere is often unlikely to be able to hold the moisture so, it rains”, is an example of knowledge.



4.2.2 Sample Data Mining Problems

Now that we have defined data, information and knowledge let us define some of the problems that can be solved through the data mining process.

a) Mr Ramniwas Gupta manages a supermarket and the cash counters, he adds transactions into the database. Some of the questions that can come to Mr. Gupta's mind are as follows:

- a) Can you help me visualise my sales?
- b) Can you profile my customers?
- c) Tell me something interesting about sales such as, what time sales will be maximum etc.

He does not know statistics, and he does not want to hire statisticians.

The answer of some of the above questions may be answered by data mining.

b) Mr. Avinash Arun is an astronomer and the sky survey has 3 tera-bytes (10^{12}) of data, 2 billion objects. Some of the questions that can come to the mind of Mr. Arun are as follows:

- a) Can you help me recognise the objects?
- b) Most of the data is beyond my reach. Can you find new/unusual items in my data?
- c) Can you help me with basic manipulation, so I can focus on the basic science of astronomy?

He knows the data and statistics, but that is not enough. The answer to some of the above questions may be answered once again, by data mining.

Please note: The use of data mining in both the questions given above lies in finding certain patterns and information. Definitely the type of the data in both the database as given above will be quite different.

4.2.3 Database Processing Vs. Data Mining Processing

Let us, first, differentiate between database processing and data mining processing: The query language of database processing is well defined and it uses SQL for this, while, the data mining, the query is poorly defined and there is no precise query language. The data used in data processing is operational data, while, in data mining, it is historical data i.e., it is not operational data.

The output of the query of database processing is precise and is the subset of the data, while, in the case of data mining the output is fuzzy and it is not a subset of the data.

Some of the examples of database queries are as follows:

- Find all credit card applicants with the last name Ram.
- Identify customers who have made purchases of more than Rs.10,000/- in the last month.
- Find all customers who have purchased shirt(s).

Some data mining queries may be:

- Find all credit card applicants with poor or good credit risks.
- Identify the profile of customers with similar buying habits.
- Find all items that are frequently purchased with shirt (s).

4.2.4 Data Mining Vs. Knowledge Discovery in Databases (KDD)

Knowledge Discovery in Databases (KDD) is the process of finding useful information, knowledge and patterns in data while data mining is the process of using algorithms to automatically extract desired information and patterns, which are derived by the Knowledge Discovery in Databases process. Let us define KDD in more details.

Knowledge Discovery in Databases (KDD) Process

The different steps of KDD are as follows:

- **Extraction:** Obtains data from various data sources.
- **Preprocessing:** It includes cleansing the data which has already been extracted by the above step.
- **Transformation:** The data is converted in to a common format, by applying some technique.
- **Data Mining:** Automatically extracts the information/patterns/knowledge.
- **Interpretation/Evaluation:** Presents the results obtained through data mining to the users, in easily understandable and meaningful format.

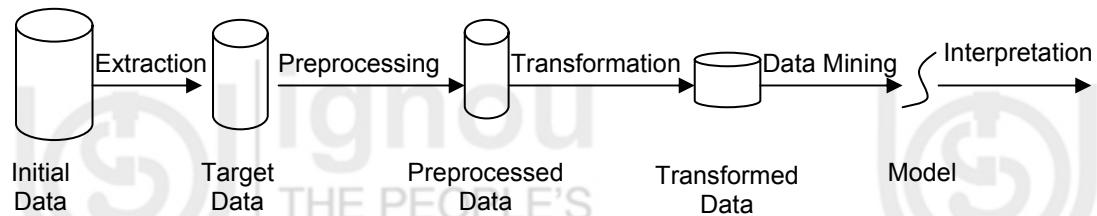


Figure 1: KDD process

Tasks in Knowledge Discovery in Databases (KDD) Process

The different tasks in KDD are as follows:

- **Obtains information on application domain:** It gathers knowledge from the domain relevant to the user.
- **Extracting data set:** It includes extracting required data which will later, be used for analysis.
- **Data cleansing process:** It involves basic operations such as, the removal of noise, collecting necessary information from noisy data, such as, deciding on strategies for handling missing data fields.
- **Data reduction and projection:** Using dimensionality reduction or transformation methods it reduces the effective number of dimensions under consideration.
- **Selecting data mining task:** In this stage we decide what the objective of the KDD process is. Whether it is classification, clustering, association rules etc.
- **Selecting data mining method:** In this stage, we decide the methods and the parameter to be used for searching for desired patterns in the data.



Data organised by
function

The KDD Process

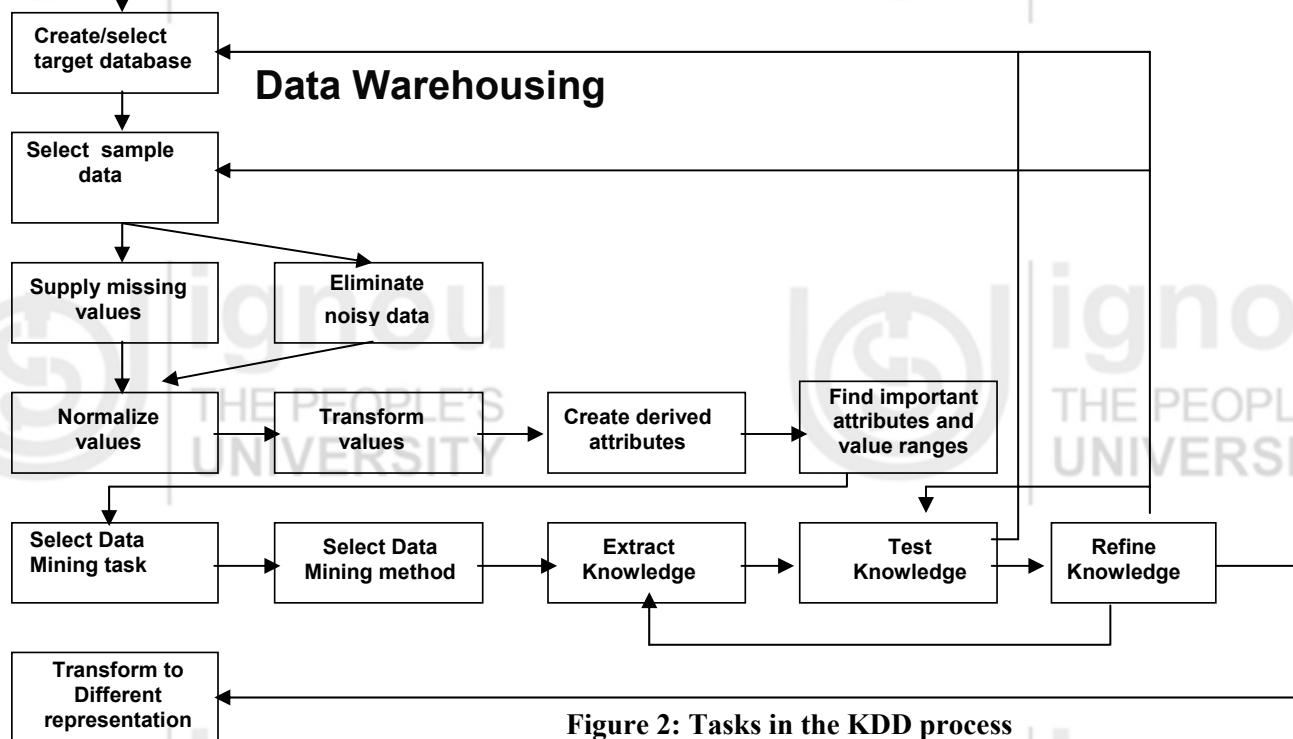


Figure 2: Tasks in the KDD process

- **Extraction of patterns:** It includes searching for desired patterns only, because, the data-mining model may generate a lot of patterns.
- Interpretation and presentation of pattern/model.

4.3 APPROACHES TO DATA MINING PROBLEMS

The approaches to data mining problems are based on the type of information/knowledge to be mined. We will emphasize on three different approaches: Classification, Clustering, and Association Rules.

The classification task maps data into predefined groups or classes. The class of a tuple is indicated by the value of a user-specified goal attribute. Tuples consist of a set of predicated attributes and a goal attribute. The task is to discover some kind of relationship between the predicated attributes and the goal attribute, so that the discovered information/knowledge can be used to predict the class of new tuple(s).

The task of clustering is to group the tuples with similar attribute values into the same class. Given a database of tuples and an integer value k, the Clustering is to define a mapping such that, tuples are mapped to different cluster.

The principle is to maximize intra-class similarity and minimize the interclass similarity. In clustering, there is no goal attribute. So, classification is supervised by the goal attribute, while clustering is an unsupervised classification.

The task of association rule mining is to search for interesting relationships among items in a given data set. Its original application is on “market basket data”. The rule has the form X → Y, where X and Y are sets of items and they do not intersect. Each

rule has two measurements, support and confidence. Given the user-specified minimum support and minimum confidence, the task is to find, rules with support and confidence above, minimum support and minimum confidence.

The distance measure finds, the distance or dissimilarity between objects the measures that are used in this unit are as follows:

- Euclidean distance: $\text{dis}(t_i, t_j) = \sqrt{\sum_{h=1}^k (t_{ih} - t_{jh})^2}$
- Manhattan distance: $\text{dis}(t_i, t_j) = \sum_{h=1}^k |(t_{ih} - t_{jh})|$

where t_i and t_j are tuples and h are the different attributes which can take values from 1 to k

Check Your Progress 1

- 1) What do you mean by data mining?
.....
.....
- 2) How is data mining different from Knowledge discovery in databases? What are the different steps of KDD?
.....
.....

- 3) What is the difference between data mining and OLTP?
.....
.....

- 4) What are different data mining tasks?
.....
.....

4.4 CLASSIFICATION

The classification task maps data into predefined groups or classes.

Given a database/dataset $D = \{t_1, t_2, \dots, t_n\}$ and a set of classes $C = \{C_1, \dots, C_m\}$, the classification Problem is to define a mapping $f: D \rightarrow C$ where each t_i is assigned to one class, that is, it divides database/dataset D into classes specified in the Set C .

A few very simple examples to elucidate classification could be:

- Teachers classify students' marks data into a set of grades as A, B, C, D, or F.
- Classification of the height of a set of persons into the classes tall, medium or short.

4.4.1 Classification Approach



The basic approaches to classification are:

- To create specific models by, evaluating training data, which is basically the old data, that has already been classified by using the domain of the experts' knowledge.
- Now applying the model developed to the new data.

Please note that in classification, the classes are predefined.

Some of the most common techniques used for classification may include the use of Decision Trees, Neural Networks etc. Most of these techniques are based on finding the distances or uses statistical methods.

4.4.2 Classification Using Distance (K-Nearest Neighbours)

This approach, places items in the class to which they are “closest” to their neighbour. It must determine distance between an item and a class. Classes are represented by centroid (Central value) and the individual points.

One of the algorithms that is used is K-Nearest Neighbors. Some of the basic points to be noted about this algorithm are:

- The training set includes *classes* along with other attributes. (Please refer to the training data given in the *Table* given below).
- The value of the K defines the number of *near items* (items that have less distance to the attributes of concern) that should be used from the given set of training data (just to remind you again, training data is already classified data). This is explained in point (2) of the following example.
- A new item is placed in the class in which the most number of close items are placed. (Please refer to point (3) in the following example).
- The value of K should be $\leq \sqrt{\text{Number_of_training_items}}$ However, in our example for limiting the size of the sample data, we have not followed this formula.

Example: Consider the following data, which tells us the person's class depending upon gender and height

Name	Gender	Height	Class
Sunita	F	1.6m	Short
Ram	M	2m	Tall
Namita	F	1.9m	Medium
Radha	F	1.88m	Medium
Jully	F	1.7m	Short
Arun	M	1.85m	Medium
Shelly	F	1.6m	Short
Avinash	M	1.7m	Short
Sachin	M	2.2m	Tall
Manoj	M	2.1m	Tall
Sangeeta	F	1.8m	Medium
Anirban	M	1.95m	Medium
Krishna	F	1.9m	Medium
Kavita	F	1.8m	Medium
Pooja	F	1.75m	Medium

- 1) You have to classify the tuple $\langle \text{Ram}, \text{M}, 1.6 \rangle$ from the training data that is given



to you.

- 2) Let us take only the **height** attribute for distance calculation and suppose K=5 then the following are the near five tuples to the data that is to be classified (using Manhattan distance as a measure on the height attribute).

Name	Gender	Height	Class
Sunita	F	1.6m	Short
Jully	F	1.7m	Short
Shelly	F	1.6m	Short
Avinash	M	1.7m	Short
Pooja	F	1.75m	Medium

- 3) On examination of the tuples above, we classify the tuple <Ram, M, 1.6> to *Short* class since most of the tuples above belongs to *Short* class.

4.4.3 Decision or Classification Tree

Given a data set $D = \{t_1, t_2, \dots, t_n\}$ where $t_i = \langle t_{i1}, \dots, t_{ih} \rangle$, that is, each tuple is represented by h attributes, assume that, the database schema contains attributes as $\{A_1, A_2, \dots, A_h\}$. Also, let us suppose that the classes are $C = \{C_1, \dots, C_m\}$, then:

Decision or Classification Tree is a tree associated with D such that

- Each internal node is labeled with attribute, A_i
- Each arc is labeled with the predicate which can be applied to the attribute at the parent node.
- Each leaf node is labeled with a class, C_j

Basics steps in the Decision Tree are as follows:

- Building the tree by using the training set dataset/database.
- Applying the tree to the new dataset/database.

Decision Tree Induction is the process of learning about the classification using the inductive approach. During this process, we create a decision tree from the training data. This decision tree can, then be used, for making classifications. To define this we need to define the following.

Let us assume that we are given probabilities p_1, p_2, \dots, p_s whose sum is 1. Let us also define the term Entropy, which is the measure of the amount of randomness or surprise or uncertainty. Thus our basic goal in the classification process is that, the entropy for a classification should be zero, that, if no surprise then, entropy is equal to zero. Entropy is defined as:

$$H(p_1, p_2, \dots, p_s) = \sum_{i=1}^s (p_i * \log(1/p_i)) \quad \dots \quad (1)$$

ID3 Algorithm for Classification

This algorithm creates a tree using the algorithm given below and tries to reduce the expected number of comparisons.

Algorithm: ID3 algorithm for creating decision tree from the given training data.

Input: The *training data* and the *attribute-list*.



Output: A decision tree.

Process:

Step 1: Create a node N

Step 2: If sample data are all of the same class, C (that is probability is 1)
then return N as a leaf node labeled class C

Step 3: If *attribute-list* is empty

then return N as a leaf node label it with the most common class in
the training data; // majority voting

Step 4: Select *split-attribute*, which is the attribute in the *attribute-list* with the
highest information gain;

Step 5: label node N with *split-attribute*;

Step 6: for each known value A_i , of *split-attribute* // partition the samples

Create a branch from node N for the condition: *split-attribute* = A_i ;

// Now consider a partition and recursively create the decision tree:

Let x_i be the set of data from training data that satisfies the condition:

split-attribute = A_i

if the set x_i is empty then

attach a leaf labeled with the most common class in the prior
set of training data;

else

attach the node returned after recursive call to the program
with training data as x_i and

new attribute list = present attribute-list – *split-attribute*;

End of Algorithm.

Please note: The algorithm given above, chooses the split attribute with the highest
information gain, that is, calculated as follows:

$$\text{Gain } (D,S) = H(D) - \sum_{i=1}^s (P(D_i) * H(D_i)) \quad \dots \dots \dots (2)$$

where S is new states = { $D_1, D_2, D_3, \dots, D_s$ } and $H(D)$ finds the amount of order in that
state

Consider the following data in which *Position* attribute acts as class

Department	Age	Salary	Position
Personnel	31-40	Medium Range	Boss
Personnel	21-30	Low Range	Assistant
Personnel	31-40	Low Range	Assistant
MIS	21-30	Medium Range	Assistant
MIS	31-40	High Range	Boss
MIS	21-30	Medium Range	Assistant
MIS	41-50	High Range	Boss
Administration	31-40	Medium Range	Boss
Administration	31-40	Medium Range	Assistant
Security	41-50	Medium Range	Boss
Security	21-30	Low Range	Assistant

Figure 3: Sample data for classification

We are applying ID3 algorithm, on the above dataset as follows:

The initial entropy of the dataset using formula at (1) is

$$H(\text{initial}) = (6/11)\log(11/6) + (5/11)\log(11/5) = 0.29923$$

(Assistant) (Boss)

Now let us calculate gain for the departments using the formula at (2)

$$\text{Gain(Department)} = H(\text{initial}) - [P(\text{Personnel}) * H(\text{MIS}) + P(\text{MIS}) * H(\text{Personnel}) + P(\text{Administration}) * H(\text{Administration}) + P(\text{Security}) * H(\text{Security})]$$

$$\begin{aligned}
 &= 0.29923 - \{ (3/11)[(1/3)\log 3 + (2/3)\log(3/2)] + (4/11)[(2/4)\log 2 + (2/4)\log 2] + \\
 &\quad (2/11)[(1/2)\log 2 + (1/2)\log 2] + (2/11)[(1/2)\log 2 + (1/2)\log 2] \} \\
 &= 0.29923 - 0.2943 \\
 &\equiv 0.0049
 \end{aligned}$$

Similarly:

$$\begin{aligned}
 \text{Gain(Age)} &= 0.29923 - \{ (4/11)[(4/4)\log(4/4)] + (5/11)[(3/5)\log(5/3) + \\
 &\quad (2/5)\log(5/2)] + (2/11)[(2/2)\log(2/2)] \} \\
 &= 0.29923 - 0.1328 \\
 &= 0.1664
 \end{aligned}$$

$$\begin{aligned}
 \text{Gain}(\text{Salary}) &= 0.29923 - \{ (3/11)[(3/3)\log 3] + (6/11)[(3/6) \log 2 + (3/6)\log 2] + \\
 &\quad (2/11) [(2/2) \log(2/2)] \} \\
 &= 0.29923 - 0.164 \\
 &\equiv 0.1350
 \end{aligned}$$

Since age has the maximum gain, so, this attribute is selected as the first splitting attribute. In age range 31-40, class is not defined while for other ranges it is defined.

So, we have to again calculate the splitting attribute for this age range (31-40). Now, the tuples that belong to this range are as follows:

Department	Salary	Position
Personnel	Medium Range	Boss
Personnel	Low Range	Assistant
MIS	High Range	Boss
Administration	Medium Range	Boss
Administration	Medium Range	Assistant

Again the initial entropy = $(2/5)\log(5/2) + (3/5)\log(5/3) = 0.29922$
(Assistant) (Boss)

$$\begin{aligned} \text{Gain(Department)} &= 0.29922 - \left\{ \frac{2}{5} \left[\frac{1}{2} \log 2 + \frac{1}{2} \log 2 \right] + \frac{1}{5} \left[\frac{1}{1} \log 1 \right] + \right. \\ &\quad \left. \frac{2}{5} \left[\frac{1}{2} \log 2 + \frac{1}{2} \log 2 \right] \right\} \\ &= 0.29922 - 0.240 \\ &= 0.05922 \end{aligned}$$

$$\text{Gain (Salary)} = 0.29922 - \left\{ (1/5) [(1/1)\log 1] + (3/5) [(1/3)\log 3 + (2/3)\log(3/2)] + \right.$$



$$\begin{aligned}
 & (1/5) [(1/1)\log 1] \} \\
 & = 0.29922 - 0.1658 \\
 & = 0.13335
 \end{aligned}$$

The Gain is maximum for salary attribute, so we take salary as the next splitting attribute. In middle range salary, class is not defined while for other ranges it is defined. So, we have to again calculate the splitting attribute for this middle range. Since only department is left, so, department will be the next splitting attribute. Now, the tuples that belong to this salary range are as follows:

Department	Position
Personnel	Boss
Administration	Boss
Administration	Assistant

Again in the Personnel department, all persons are Boss, while, in the Administration there is a tie between the classes. So, the person can be either Boss or Assistant in the Administration department.

Now the decision tree will be as follows:

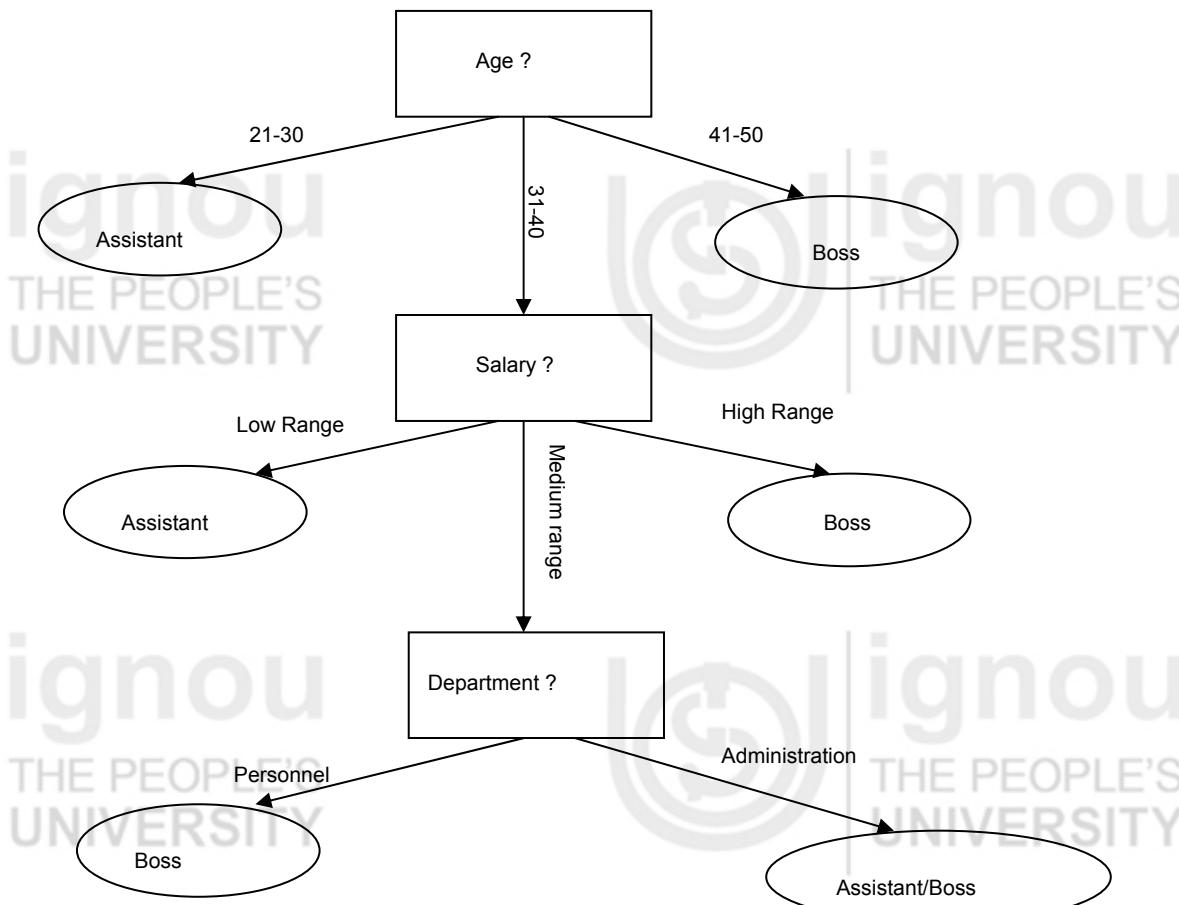


Figure 4: The decision tree using ID3 algorithm for the sample data of Figure 3.

Now, we will take a new dataset and we will classify the class of each tuple by applying the decision tree that we have built above.

Let us discuss, another important classification method called Bayesian classification in the next subsection.

4.4.4 Bayesian Classification



This is a statistical classification, which predicts the probability that a given sample is a member of a particular class. It is based on the Bayes theorem. The Bayesian classification shows better accuracy and speed when applied to large databases. We will discuss here the simplest form of Bayesian classification.

The basic underlying assumptions (also called class conditional independence) for this simplest form of classification known as the native Bayesian classification is:

“The effect of an attribute value on a given class is independent of the values of other attributes”

Let us discuss naive Bayesian classification in more details. But before that, let us, define the basic theorem on which this classification is based.

Bayes Theorem:

Let us assume the following:

X is a data sample whose class is to be determined

H is the hypothesis such that the data sample X belongs to a class C.

$P(H | X)$ is the probability that hypothesis H holds for data sample X . It is also called the posterior probability that condition H holds for the sample X.

$P(H)$ is the prior probability of H condition on the training data.

$P(X | H)$ is the posterior probability of X sample, given that H is true.

$P(X)$ is the prior probability on the sample X.

Please note: We can calculate $P(X)$, $P(X | H)$ and $P(H)$ from the data sample X and training data. It is only $P(H | X)$ which basically defines the probability that X belongs to a class C, and cannot be calculated. Bayes theorem does precisely this function. The Bayes' theorem states:

$$P(H | X) = \frac{P(X | H) P(H)}{P(X)}$$

Now after defining the Bayes theorem, let us explain the Bayesian classification with the help of an example.

- i) Consider the sample having an n-dimensional feature vector. For our example, it is a 3-dimensional (Department, Age, Salary) vector with training data as given in the Figure 3.
- ii) Assume that there are m classes C_1 to C_m . And an unknown sample X. The problem is to data mine which class X belongs to. As per Bayesian classification, the sample is assigned to the class, if the following holds:

$$P(C_i | X) > P(C_j | X) \text{ where } j \text{ is from 1 to } m \text{ but } j \neq i$$

In other words the class for the data sample X will be the class, which has the maximum probability for the unknown sample. **Please note:** The $P(C_i | X)$ will be found using:

$$P(C_i | X) = \frac{P(X | C_i) P(C_i)}{P(X)} \quad (3)$$

In our example, we are trying to classify the following data:

$X = (\text{Department} = \text{"Personal"}, \text{Age} = \text{"31 - 40"} \text{ and } \text{Salary} = \text{"Medium Range})$

into two classes (based on position) $C_1 = \text{BOSS}$ OR $C_2 = \text{Assistant}$.

- iii) The value of $P(X)$ is constant for all the classes, therefore, only $P(X | C_i) P(C_i)$ needs to be found to be maximum. Also, if the classes are equally, then,



$P(C_1)=P(C_2)=\dots=P(C_n)$, then we only need to maximise $P(X|C_i)$.
How is $P(C_i)$ calculated?

$$P(C_i) = \frac{\text{Number of training samples for Class } C_i}{\text{Total Number of Training Samples}}$$

In our example,

$$P(C_1) = \frac{5}{11}$$

and

$$P(C_2) = \frac{6}{11}$$

So $P(C_1) \neq P(C_2)$

- iv) $P(X|C_i)$ calculation may be computationally expensive if, there are large numbers of attributes. To simplify the evaluation, in the naïve Bayesian classification, we use the condition of class conditional independence, that is the values of attributes are independent of each other. In such a situation:

$$P(X|C_i) = \prod_{k=1}^n P(x_k|C_i) \quad \dots(4)$$

where x_k represent the single dimension or attribute.

The $P(x_k|C_i)$ can be calculated using mathematical function if it is continuous, otherwise, if it is categorical then, this probability can be calculated as:

$$P(x_k|C_i) = \frac{\text{Number of training samples of class } C_i \text{ having the value } x_k \text{ for the attribute } A_k}{\text{Number of training samples belonging to } C_i}$$

For our example, we have x_1 as Department= “Personnel”
 x_2 as Age= “31 – 40” and
 x_3 as Salary= “Medium Range”

$P(\text{Department= “Personnel”} \text{Position = “BOSS”})$	= 1/5
$P(\text{Department= “Personnel”} \text{Position = “Assistant”})$	= 2/6
$P(\text{Age= “31 – 40”} \text{Position = “BOSS”})$	= 3/5
$P(\text{Age= “31 – 40”} \text{Position = “Assistant”})$	= 2/6
$P(\text{Salary= “Medium Range”} \text{Position = “BOSS”})$	= 3/5
$P(\text{Salary= “Medium Range”} \text{Position = “Assistant”})$	= 3/6

Using the equation (4) we obtain:

$$\begin{aligned} P(X | \text{Position = “BOSS”}) &= 1/5 * 3/5 * 3/5 \\ P(X | \text{Position = “Assistant”}) &= 2/6 * 2/6 * 3/6 \end{aligned}$$

Thus, the probabilities:

$$\begin{aligned} P(X | \text{Position = “BOSS”}) P(\text{Position = “BOSS”}) &= (1/5 * 3/5 * 3/5) * 5/11 \\ &= 0.032727 \\ P(X | \text{Position = “Assistant”}) P(\text{Position = “Assistant”}) &= (2/6 * 2/6 * 3/6) * 6/11 \\ &= 0.030303 \end{aligned}$$

Since, the first probability of the above two is higher, the sample data may be classified into the BOSS position. Kindly check to see that you obtain the same result from the decision tree of *Figure 4*.

4.5 CLUSTERING

Clustering is grouping thing with similar attribute values into the same group. Given a database $D = \{t_1, t_2, \dots, t_n\}$ of tuples and an integer value k , the Clustering problem is to define a mapping where each tuple t_i is assigned to one cluster K_j , $1 \leq j \leq k$.

A **Cluster**, K_j , contains precisely those tuples mapped to it. Unlike the classification problem, clusters are not known in advance. The user has to enter the value of the number of clusters k .

In other words a *cluster* can be defined as the collection of data objects that are similar in nature, as per certain defining property, but these objects are dissimilar to the objects in other clusters.

Some of the clustering examples are as follows:

- To segment the customer database of a departmental store based on similar buying patterns.
- To identify similar Web usage patterns etc.

Clustering is a very useful exercise specially for identifying similar groups from the given data. Such data can be about buying patterns, geographical locations, web information and many more.

Some of the clustering Issues are as follows:

- **Outlier handling:** How will the outlier be handled? (outliers are the objects that do not comply with the general behaviour or model of the data) Whether it is to be considered or it is to be left aside while calculating the clusters?
- **Dynamic data:** How will you handle dynamic data?
- **Interpreting results:** How will the result be interpreted?
- **Evaluating results:** How will the result be calculated?
- **Number of clusters:** How many clusters will you consider for the given data?
- **Data to be used:** whether you are dealing with quality data or the noisy data? If, the data is noisy how is it to be handled?
- **Scalability:** Whether the algorithm that is used is to be scaled for small as well as large data set/database.

There are many different kinds of algorithms for clustering. However, we will discuss only three basic algorithms. You can refer to more details on clustering from the further readings.

4.5.1 Partitioning Clustering

The partitioning clustering algorithms constructs k partitions from a given n objects of the data. Here $k \leq n$ and each partition must have at least one data object while one object belongs to **only one** of the partitions. A partitioning clustering algorithm normally requires users to input the desired number of clusters, k .

Some of the partitioning clustering algorithms are as follows:

- Squared Error
- K-Means



Now in this unit, we will briefly discuss these algorithms.

Squared Error Algorithms

The most frequently used criterion function in partitioning clustering techniques is the squared error criterion. The method of obtaining clustering by applying this approach is as follows:

Squared Error Clustering Method:

- (1) Select an initial partition of the patterns with a fixed number of clusters and cluster centers.
- (2) Assign each pattern to its closest cluster center and compute the new cluster centers as the centroids of the clusters. Repeat this step until convergence is achieved, i.e., until the cluster membership is stable.
- (3) Merge and split clusters based on some heuristic information, optionally repeating step 2.

Some of the parameters that are used in clusters are as follows:

$$\text{Centriod}(C_m) = \frac{\sum_{i=1}^N t_{mi}}{N}$$

$$\text{Radius } (R_m) = \sqrt{\sum_{i=1}^N (t_{mi} - C_m)^2 / N}$$

$$\text{Diameter } (D_m) = \sqrt{\sum_{i=1}^N \sum_{j=1}^N (t_{mi} - t_{mj})^2 / (N * (N - 1))}$$

A detailed discussion on this algorithm is beyond the scope of this unit. You can refer to more details on clustering from the further readings.

K-Means clustering

In the K-Means clustering, initially a set of clusters is randomly chosen. Then iteratively, items are moved among sets of clusters until the desired set is reached. A high degree of similarity among elements in a cluster is obtained by using this algorithm. For this algorithm a set of clusters $K_i = \{t_{i1}, t_{i2}, \dots, t_{im}\}$ is given, the cluster mean is:

$$m_i = (1/m)(t_{i1} + \dots + t_{im}) \quad \dots(5)$$

Where t_i represents the tuples and m represents the mean

The K-Means algorithm is as follows:

Input :

$D = \{t_1, t_2, \dots, t_n\}$ //Set of elements

A //Adjacency matrix showing distance between elements.

k //Number of desired clusters.

Output :

K //Set of Clusters

K-Means Algorithm:

Assign initial values for means m_1, m_2, \dots, m_k ;

Repeat

 Assign each item t_i to the cluster which has the closest mean;

 Calculate new mean for each cluster;

Until convergence criteria is met.

K-Means Example:

Let us take the number of clusters as 2 and the following input set is given to us:

Input set = {1, 2, 3, 5, 10, 12, 22, 32, 16, 18}

- Step 1: We randomly assign means: $m_1=3, m_2=5$
- Step 2: $K_1=\{1,2,3\}, K_2=\{5,10,12,22,32,16,18\}$, $m_1=2, m_2=16.43$ (calculated mean using the formula (5)).

Now redefine cluster as per the closest mean:

- Step 3: $K_1=\{1,2,3,5\}, K_2=\{10,12,22,32,16,18\}$

Calculate the mean once again:

- $m_1=2.75, m_2=18.33$
- Step 4: $K_1=\{1,2,3,5\}, K_2=\{10,12,22,32,16,18\}$, $m_1=2.75, m_2=18.33$
- Stop as the clusters with these means are the same.

4.5.2 Nearest Neighbour Clustering

In this approach, items are iteratively merged into the existing clusters that are closest. It is an incremental method. The threshold, t , used to determine if items are added to existing clusters or a new cluster is created. This process continues until all patterns are labeled or no additional labeling occurs.

The Nearest Neighbour algorithm is as follows:

Input :

```
D= { t1,t2,...tn} //Set of elements
A           //Adjacency matrix showing distance between elements.
k           //Number of desired clusters.
```

Output :

```
K           //Set of Clusters
```

Nearest Neighbour algorithm :

```
K1={t1};
K={K1};
k=1;
for i=2 to n do
    Find the tm in some cluster Km in K such that distance(ti,tm) is the
    smallest;
    If dis(ti,tm)<= t then
        Km=Km U ti;
    Else
        k=k +1;
        Kk={ti};
```

4.5.3 Hierarchical Clustering

In this method, the clusters are created in levels and depending upon the threshold value at each level the clusters are again created.

An agglomerative approach begins with each tuple in a distinct cluster and successively merges clusters together until a stopping criterion is satisfied. This is the bottom up approach.

A divisive method begins with all tuples in a single cluster and performs splitting until a stopping criterion is met. This is the top down approach.

A hierarchical algorithm yields a dendrogram representing the nested grouping of tuples and similarity levels at which groupings change. Dendrogram is a tree data structure which illustrates hierarchical clustering techniques. Each level shows clusters for that level. The leaf represents individual clusters while the root represents one cluster.



Most hierarchical clustering algorithms are variants of the single-link, average link and complete-link algorithms. Out of these the single-link and complete-link algorithms are the most popular. These two algorithms differ in the way they characterise the similarity between a pair of clusters.

In the single-link method, the distance between two clusters is the minimum of the distances between all pairs of patterns drawn from the two clusters (one pattern from the first cluster, the other from the second).

In the complete-link algorithm, the distance between two clusters is the maximum of all pair-wise distances between patterns in the two clusters.

In either case, two clusters are merged to form a larger cluster based on minimum distance criteria.

You can refer to more detail on the hierarchical clustering algorithms from the further readings.

☛ Check Your Progress 2

- 1) What is the classification of data? Give some examples of classification.

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

- 2) What is clustering?

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

4.6 ASSOCIATION RULE MINING

The task of association rule mining is to find certain association relationships among a set of items in a dataset/database. The association relationships are described in association rules. In association rule mining there are two measurements, *support* and *confidence*. The confidence measure indicates the rule's strength, while support corresponds to the frequency of the pattern.

A typical example of an association rule created by data mining often termed to as “market basket data” is: “ 80% of customers who purchase bread also purchase butter.”

Other applications of data mining include cache customisation, advertisement personalisation, store layout and customer segmentation etc. All these applications try to determine the associations between data items, if it exists to optimise performance.

Formal Definition:

Let $I = \{i_1, i_2, \dots, i_m\}$ be a set of literals, called items. Let D be a set of transactions, where each transaction T is a set of items such that $T \subseteq I$. TID indicates a unique

transaction identifier. An association rule is an implication of the form $X \rightarrow Y$, where $X \subset I$, $Y \subset I$, and $X \cap Y = \emptyset$. X is called the antecedent while Y is called the consequence of the rule.

The rule $X \rightarrow Y$ has support s in the transaction set D if $s\%$ of transactions in D contains $X \cup Y$. The rule has confidence c if $c\%$ of transactions in D that contains X also contains Y . Support indicates how frequently the pattern occurs, while confidence indicates the strength of the rule.

Given a user specified minimum support and minimum confidence, the problem of mining association rules is to find all the association rules whose support and confidence are larger than the minimum support and minimum confidence. Thus, this approach can be broken into two sub-problems as follows:

- (1) Finding the frequent itemsets which have support above the predetermined minimum support.
- (2) Deriving all rules, based on each frequent itemset, which have confidence more than the minimum confidence.

There are a lots of ways to find the large itemsets but we will only discuss the Apriori Algorithm.

Apriori Algorithm: For finding frequent itemsets

The apriori algorithm applies the concept that if an itemset has minimum support, then all its subsets also have minimum support. An itemset having minimum support is called frequent itemset or large itemset. So any subset of a frequent itemset must also be frequent.

Apriori algorithm generates the candidate itemsets to be counted in the pass, by using only the large item set found in the previous pass – without considering the transactions in the database.

It starts by finding all frequent 1-itemsets (itemsets with 1 item); then consider 2-itemsets from these 1-itemsets, and so forth. During each iteration only candidates found to be frequent in the previous iteration are used to generate a new candidate set during the next iteration. The algorithm terminates when there are no frequent k-itemsets.

Notations that are used in Apriori algorithm are given below:

k-itemset	An itemset having k items
L_k	Set of frequent k-itemset (those with minimum support)
C_k	Set of candidate k-itemset (potentially frequent itemsets)

Apriori algorithm function takes as argument L_{k-1} and returns a superset of the set of all frequent k-itemsets. It consists of a join step and a prune step. The Apriori algorithm is given below :

APRIORI

1. $k = 1$
2. Find frequent set L_k from C_k of all candidate itemsets
3. Form C_{k+1} from L_k ; $k = k + 1$
4. Repeat 2-3 until C_k is empty

Details about steps 2 and 3

Step 2: Scan the data set D and count each itemset in C_k , if it is greater than minimum support, it is frequent



Step 3:

- For $k=1$, C_1 = all frequent 1-itemsets. (all individual items).
- For $k>1$, generate C_k from L_{k-1} as follows:

The join step

C_k = k -2 way join of L_{k-1} with itself

If both $\{a_1, \dots, a_{k-2}, a_{k-1}\} \& \{a_1, \dots, a_{k-2}, a_k\}$ are in L_{k-1} , then
add $\{a_1, \dots, a_{k-2}, a_{k-1}, a_k\}$ to C_k
(We keep items sorted).

The prune step

Remove $\{a_1, \dots, a_{k-2}, a_{k-1}, a_k\}$ if it contains a non-frequent
($k-1$) subset.

{In the prune step, delete all itemsets $c \in C_k$ such that some
($k-1$)-subset of C is not in L_{k-1} .}

Example: Finding frequent itemsets:

Consider the following transactions with minimum support $s=30\%$ for finding the frequent itemsets by applying Apriori algorithm:

Transaction ID	Item(s) purchased
1	Shirt, Trouser
2	Shirt, Trouser, Coat
3	Coat, Tie, Tiepin
4	Coat, Shirt, Tie, Trouser
5	Trouser, Belt
6	Coat, Tiepin, Trouser
7	Coat, Tie
8	Shirt
9	Shirt, Coat
10	Shirt, Handkerchief

Method of finding the frequent itemset is as follows:

Pass Number	Candidates	Large itemsets (≥ 3)
1	$C_1 = \{\text{Belt}, 1, \text{Coat}, 6, \text{Handkerchief}, 1, \text{Shirt}, 6, \text{Tie}, 3, \text{Tiepin}, 2, \text{Trouser}, 5\}$	$L_1 = \{\text{Coat}, 6, \text{Shirt}, 6, \text{Tie}, 3, \text{Trouser}, 5\}$
2	$C_2 = \{\{\text{Coat}, \text{Shirt}\}, 3, \{\text{Coat}, \text{Tie}\}, 3, \{\text{Coat}, \text{Trouser}\}, 3, \{\text{Shirt}, \text{Tie}\}, 1, \{\text{Shirt}, \text{Trouser}\}, 3, \{\text{Tie}, \text{Trouser}\}, 1\}$	$L_2 = \{\{\text{Coat}, \text{Shirt}\}, 3, \{\text{Coat}, \text{Tie}\}, 3, \{\text{Coat}, \text{Trouser}\}, 3, \{\text{Shirt}, \text{Trouser}\}, 3\}$
3	$C_3 = \{\{\text{Coat}, \text{Shirt}, \text{Trouser}\}, 2\}$	$L_3 = \emptyset$

The calculation of 3-itemsets is mentioned below:

Join operation yields 3 item sets as: $\{\{\text{Coat}, \text{Shirt}, \text{Tie}\}, \{\text{Coat}, \text{Shirt}, \text{Trouser}\}, \{\text{Coat}, \text{Tie}, \text{Trouser}\}\}$

However, the Prune operation removes two of these items from the set due to the following reasons:

- {Coat, Shirt, Tie} is pruned as {Shirt, Tie} is not in L₂
- {Coat, Shirt, Trouser} is retained as {Coat, Shirt}, {Coat, Trouser} and {Shirt, Trouser} all three are in L₂
- {Coat, Tie, Trouser} is pruned as {Tie, Trouser} is not in L₂

The set L={ L₁, L₂, L₃ }

The following algorithm creates the association rules from the set L so created by the Apriori algorithm.

Algorithm to generate the Association Rules:

Input:

D	//Database of transactions
I	//Items
L	//Large itemsets
s	// Support
c	// Confidence

Output:

R	//Association rules satisfying minimum s and c
---	--

AR algorithm:

```

R=∅
For each l ∈ L do // for each large item (l) in the set L
    For each x ⊂ I such that x ⊈ ∅ and x ⊈ l do
        if support(l)/support(x) ≥ c then
            R=R ∪ {x → (l - x)};

```

Apriori Advantages/Disadvantages:

The following are the advantages and disadvantages of the Apriori algorithm:

- **Advantages:**
 - It uses large itemset property.
 - It is easy to implement.
- **Disadvantages:**
 - It assumes transaction database is memory resident.

4.7 APPLICATIONS OF DATA MINING PROBLEM

Some of the applications of data mining are as follows:

- **Marketing and sales data analysis:** A company can use customer transactions in their database to segment the customers into various types. Such companies may launch products for specific customer bases.
- **Investment analysis:** Customers can look at the areas where they can get good returns by applying the data mining.
- **Loan approval:** Companies can generate rules depending upon the dataset they have. On that basis they may decide to whom, the loan has to be approved.
- **Fraud detection:** By finding the correlation between faults, new faults can be detected by applying data mining.



- **Network management:** By analysing pattern generated by data mining for the networks and its faults, the faults can be minimised as well as future needs can be predicted.
- **Risk Analysis:** Given a set of customers and an assessment of their risk-worthiness, descriptions for various classes can be developed. Use these descriptions to classify a new customer into one of the risk categories.
- **Brand Loyalty:** Given a customer and the product he/she uses, predict whether the customer will change their products.
- **Housing loan prepayment prediction:** Rule discovery techniques can be used to accurately predict the aggregate number of loan prepayments in a given quarter as a function of prevailing interest rates, borrower characteristics and account data.

4.8 COMMERCIAL TOOLS OF DATA MINING

Commercial Tools:

- 1) **AC2**, provides graphical tools for data preparation and building decision trees.
- 2) **Business Miner**, data mining product positioned for the mainstream business user.
- 3) **C4.5**, the "classic" decision-tree tool, developed by **J. R. Quinlan**
- 4) **C5.0/See5**, constructs classifiers in the form of decision trees and rulesets.
- 5) **CART**, decision-tree software, combines an easy-to-use GUI with advanced features for data mining, data pre-processing and predictive modeling.
- 6) **Cognos Scenario**, allows you to quickly identify and rank the factors that have a significant impact on your key business measures.
- 7) **Decisionhouse**, provides data extraction, management, pre-processing and visualisation, plus customer profiling, segmentation and geographical display.
- 8) **Kernel Miner**, decision-tree-based classifier with fast DB access.
- 9) **Knowledge Seeker**, high performance interactive decision tree analytical tool.
- 10) **SPSS AnswerTree**, easy to use package with four decision tree algorithms - two types of CHAID, CART, and QUEST.
- 11) **XpertRule Miner** (Attar Software), provides graphical decision trees with the ability to embed as ActiveX components.
- 12) **AIRA**, a rule discovery, data and knowledge visualisation tool. AIRA for Excel extracts rules from MS-Excel spreadsheets.
- 13) **Datamite**, enables rules and knowledge to be discovered in ODBC-compliant relational databases.
- 14) **SuperQuery**, business Intelligence tool; works with Microsoft Access and Excel and many other databases.
- 15) **WizWhy**, automatically finds all the if-then rules in the data and uses them to summarise the data, identify exceptions, and generate predictions for new cases.
- 16) **XpertRule Miner** (Attar Software) provides association rule discovery from any ODBC data source.
- 17) **DMSK: Data-Miner Software Kit :Task:** Collection of tools for efficient mining of big data (Classification, Regression, Summarisation, Deviation Detection multi-task tools).



- 16) **OSHAM Task**: Task (Clustering) interactive-graphic system for discovering concept hierarchies from unsupervised data
- 17) **DBMiner** is a data mining system for interactive mining of multiple-level knowledge in large relational databases.

Free Tools:

- 1) **EC4.5**, a more efficient version of C4.5, which uses the best among three strategies at each node construction.
- 2) **IND**, provides CART and C4.5 style decision trees and more. Publicly available from NASA but with export restrictions.
- 3) **ODBCMINE**, shareware data-mining tool that analyses ODBC databases using the C4.5, and outputs simple IF..ELSE decision rules in ASCII.
- 4) **OC1**, decision tree system continuous feature values; builds decision trees with linear combinations of attributes at each internal node; these trees then partition the space of examples with both oblique and axis-parallel hyper planes.
- 5) **PC4.5**, a parallel version of C4.5 built with Persistent Linda system.
- 6) **SE-Learn**, Set Enumeration (SE) trees generalise decision trees. Rather than splitting by a single attribute, one recursively branches on all (or most) relevant attributes. (LISP)
- 7) **CBA**, mines association rules and builds accurate classifiers using a subset of association rules.
- 8) **KINOsuite-PR** extracts rules from trained neural networks.
- 9) **RIPPER**, a system that learns sets of rules from data

☛ Check Your Progress 3

- 1) What is association rule mining?
-
.....
.....

- 2) What is the application of data mining in the banking domain?
-
.....
.....

- 3) Apply the Apriori algorithm for generating large itemset on the following dataset:

Transaction ID	Items purchased
T100	A ₁ a ₃ a ₄
T200	A ₂ a ₃ a ₅
T300	A ₁ a ₂ a ₃ a ₅
T400	A ₂ a ₅



4.9 SUMMARY

- 1) Data mining is the process of automatic extraction of interesting (non trivial, implicit, previously unknown and potentially useful) information or pattern from the data in large databases.
- 2) Data mining is one of the steps in the process of Knowledge Discovery in databases.
- 3) In data mining tasks are classified as: Classification, Clustering and Association rules.
- 4) The classification task maps data into predefined classes.
- 5) Clustering task groups things with similar properties/ behaviour into the same groups.
- 6) Association rules find the association relationship among a set of objects.
- 7) Data mining is applied in every field whether it is Games, Marketing, Bioscience, Loan approval, Fraud detection etc.

4.10 SOLUTIONS /ANSWERS

Check Your Progress 1

- 1) Data mining is the process of automatic extraction of interesting (non trivial, implicit, previously unknown and potentially useful) information or patterns from the data in large databases.
- 2) Data mining is only one of the many steps involved in knowledge discovery in databases. The various steps in KDD are data extraction, data cleaning and preprocessing, data transformation and reduction, data mining and knowledge interpretation and representation.
- 3) The query language of OLTP is well defined and it uses SQL for it, while, for data mining the query is poorly defined and there is no precise query language. The data used in OLTP is operational data while in data mining it is historical data. The output of the query of OLTP is precise and is the subset of the data while in the case of data mining the output is fuzzy and it is not a subset of the data.
- 4) The different data-mining tasks are: Classification, Clustering and Association Rule Mining.

Check Your Progress 2

- 1) The classification task maps data into predefined groups or classes. The class of a tuple is indicated by the value of a user-specified goal attribute. Tuples consists of a set of predication attributes and a goal attribute. The task is to discover some kind of relationship between the predication attributes and the goal attribute, so that the discovered knowledge can be used to predict the class of new tuple(s).

Some of the examples of classification are: Classification of students grades depending upon their marks, classification of customers as good or bad customer in a bank.

- 2) The task of clustering is to group the tuples with similar attribute values into the same class. Given a database of tuples and an integer value k, Clustering defines mapping, such that, tuples are mapped to different clusters.
- 3) In classification, the classes are predetermined, but, in the case of clustering the groups are not predetermined. The number of clusters has to be given by the user.

Check Your Progress 3

- 1) The task of association rule mining is to search for interesting relationships among items in a given data set. Its original application is on “market basket data”. The rule has the form $X \rightarrow Y$, where X and Y are sets of items and they do not intersect.
- 2) The data mining application in banking are as follows:
 1. Detecting patterns of fraudulent credit card use.
 2. Identifying good customers.
 3. Determining whether to issue a credit card to a person or not.
 4. Finding hidden correlations between different financial indicators.
- 3) The dataset D given for the problem is:

Transaction ID	Items purchased
T100	a ₁ a ₃ a ₄
T200	a ₂ a ₃ a ₅
T300	a ₁ a ₂ a ₃ a ₅
T400	a ₂ a ₅

Assuming the minimum support as 50% for calculating the large item sets. As we have 4 transaction, at least 2 transaction should have the data item.

1. scan D
 - C₁: a₁:2, a₂:3, a₃:3, a₄:1, a₅:3
 - L₁: a₁:2, a₂:3, a₃:3, a₅:3
 - C₂: a₁a₂, a₁a₃, a₁a₅, a₂a₃, a₂a₅, a₃a₅
2. scan D
 - C₂: a₁a₂:1, a₁a₃:2, a₁a₅:1, a₂a₃:2, a₂a₅:3, a₃a₅:2
 - L₂: a₁a₃:2, a₂a₃:2, a₂a₅:3, a₃a₅:2
 - C₃: a₁a₂a₃, a₁a₂a₅, a₂a₃a₅
 - Pruned C₃: a₂a₃a₅
3. scan D
 - L₃: a₂a₃a₅:2

Thus L={L₁,L₂,L₃}

4.11 FURTHER READINGS

- 1) *Data Mining Concepts and Techniques*, J Han, M Kamber, Morgan Kaufmann Publishers, 2001.
- 2) *Data Mining*, A K Pujari, 2004.

For Unit 16 : Please read the following unit of MCS-43 Block 4 Unit 1

UNIT 1 EMERGING DATABASE MODELS, TECHNOLOGIES AND APPLICATIONS-I

Structure	Page Nos.
1.0 Introduction	5
1.1 Objectives	5
1.2 Multimedia Database	6
1.2.1 Factors Influencing the Growth of Multimedia Data	
1.2.2 Applications of Multimedia Database	
1.2.3 Contents of MMDB	
1.2.4 Designing MMDBs	
1.2.5 State of the Art of MMDBMS	
1.3 Spatial Database and Geographic Information Systems	10
1.4 Gnome Databases	12
1.4.1 Genomics	
1.4.2 Gene Expression	
1.4.3 Proteomics	
1.5 Knowledge Databases	17
1.5.1 Deductive Databases	
1.5.2 Semantic Databases	
1.6 Information Visualisation	18
1.7 Summary	19
1.8 Solutions/Answers	20

1.0 INTRODUCTION

Database technology has advanced from the relational model to the distributed DBMS and Object Oriented databases. The technology has also advanced to support data formats using XML. In addition, data warehousing and data mining technology has become very popular in the industry from the viewpoint of decision making and planning.

Database technology is also being used in advanced applications and technologies. Some of this new application includes multimedia based database applications, geographic applications, Gnome databases, knowledge and spatial databases and many more such applications. These applications require some additional features from the DBMS as they are special in nature and thus are categorised as emerging database technologies.

This unit provides a brief introduction of database requirements of these newer applications.

1.1 OBJECTIVES

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

- define the requirements of a multimedia database systems;
- identify the basic features of geographic databases;
- list the features of Gnome databases;
- differentiate various knowledge databases and their advantages, and
- define the terms information visualisation and spatial databases.



1.2 MULTIMEDIA DATABASE

Multimedia and its applications have experienced tremendous growth. Multimedia data is typically defined as containing digital images, audio, video, animation and graphics along with the textual data. In the past decade, with the advances in network technologies, the acquisition, creation, storage and processing of multimedia data and its transmission over networks have grown tremendously.

A Multimedia Database Management System (MMDBMS) provides support for multimedia data types. It also provides the facilities for traditional DBMS functions like database creation, data modeling, data retrieval, data access and organisation, and data independence. With the rapid development of network technology, multimedia database system, multimedia information exchange is becoming very important. Any such application would require the support for a strong multimedia database technology. Let us look into some of the factors that influence the growth of multimedia data.

1.2.1 Factors Influencing the Growth of Multimedia Data

(i) Technological Advancements

Some of the technological advances that attributed to the growth of multimedia data are:

- computers, their computational power and availability,
- availability of high-resolution devices for the capture and display of multimedia data (digital cameras, scanners, monitors, and printers),
- development of high-density storage devices, and
- integration of all such technologies through digital means.

(ii) High Speed Data Communication Networks and Software

Secondly high-speed data communication networks are common these days. These networks not only support high bandwidth but also are more reliable and support digital data transfer. Even the World Wide Web has rapidly grown and software for manipulating multimedia data is now available.

(iii) Applications

With the rapid growth of computing and communication technologies, many applications have come to the forefront. Thus, any such applications in future will support life with multimedia data. This trend is expected to go on increasing in the days to come.

1.2.2 Applications of Multimedia Database

Multimedia data contains some exciting features. They are found to be more effective in dissemination of information in science, engineering, medicine, modern biology, and social sciences. They also facilitate the development of new paradigms in distance learning, and interactive personal and group entertainment.

Some of the typical applications of multimedia databases are:

- media commerce
- medical media databases
- bioinformatics
- ease of use of home media
- news and entertainment
- surveillance



- wearable computing
- management of meeting/presentation recordings
- biometrics (people identification using image, video and/or audio data).

The huge amount of data in different multimedia-related applications needs databases as the basic support mechanism. This is primarily due to the fact that the databases provide consistency, concurrency, integrity, security and availability of data. On the other hand from a user perspective, databases provide ease of use of data manipulation, query and retrieval of meaningful and relevant information from a huge collection of stored data.

Multimedia Databases (MMDBs) must cope with the large volume of multimedia data, being used in various software applications. Some such applications may include digital multimedia libraries, art and entertainment, journalism and so on. Some of these qualities of multimedia data like size, formats etc. have direct and indirect influence on the design and development of a multimedia database.

Thus, a MMDBs needs to provide features of a traditional database as well as some new and enhanced functionalities and features. They must provide a homogenous framework for storing, processing, retrieving, transmitting and presenting a wide variety of multiple media data types available in a large variety of formats.

1.2.3 Contents of MMDB

A MMDB needs to manage the following different types of information with respect to the multimedia data:

Media Data: It includes the media data in the form of images, audio and video. These are captured, digitised, processes, compressed and stored. Such data is the actual information that is to be stored.

Media Format Data: This data defines the format of the media data after the acquisition, processing, and encoding phases. For example, such data may consist of information about sampling rate, resolution, frame rate, encoding scheme etc. of various media data.

Media Keyword Data: This contains the keyword related to the description of media data. For example, for a video, this might include the date, time, and place of recording, the person who recorded, the scene description, etc. This is also known as content description data.

Media Feature Data: This contains the features derived from the media data. A feature characterises the contents of the media. For example, this could contain information on the distribution of colours, the kinds of textures and the different shapes present in an image. This is also referred to as content dependent data.

The last three types are known as meta data as, they describe several different aspects of the media data. The media keyword data and media feature data are used as indices for searching purpose. The media format data is used to present the retrieved information.

1.2.4 Designing MMDBs

The following characteristics of multimedia data have direct and indirect impacts on the design of MMDBs:

- the huge size of MMDBs,
- temporal nature of the data,
- richness of content through media, and



- complexity of representation and subjective interpretation specially from the viewpoint of the meta data.

Challenges in Designing of Multimedia Databases

The major challenges in designing multimedia databases are due to the requirements they need to satisfy. Some of these requirements are:

- 1) The database should be able to manage different types of input, output, and storage devices. For example, the data may be input from a number of devices that could include scanners, digital camera for images, microphone, MIDI devices for audio, video cameras. Typical output devices are high-resolution monitors for images and video, and speakers for audio.
- 2) The database needs to handle a variety of data compression and storage formats. Please note that data encoding has a variety of formats even within a single application. For example, in a medical application, the MRI image of the brain should be loss less, thus, putting very stringent quality on the coding technique, while the X-ray images of bones can be coded with lossy techniques as the requirements are less stringent. Also, the radiological image data, the ECG data, other patient data, etc. have widely varying formats.
- 3) The database should be able to support different computing platforms and operating systems. This is due to the fact that multimedia databases are huge and support a large variety of users who may operate computers and devices suited to their needs and tastes. However, all such users need the same kind of user-level view of the database.
- 4) Such a database must integrate different data models. For example, the textual and numeric data relating to a multimedia database may be best handled using a relational database model, while linking such data with media data as well as handling media data such as video documents are better done using an object-oriented database model. So these two models need to co-exist in MMDBs.
- 5) These systems need to offer a variety of query systems for different kinds of media. The query system should be easy-to-use, fast and deliver accurate retrieval of information. The query for the same item sometimes is requested in different forms. For example, a portion of interest in a video can be queried by using either:
 - (a) a few sample video frames as an example
 - (b) a clip of the corresponding audio track or
 - (c) a textual description using keywords.
- 6) One of the main requirements for such a Database would be to handle different kinds of indices. The multimedia data is inexact and subjective in nature, thus, the keyword-based indices and exact range searches used in traditional databases are ineffective in such databases. For example, the retrieval of records of students based on enrolment number is precisely defined, but the retrieval of records of student having certain facial features from a database of facial images, requires, content-based queries and similarity-based retrievals. Thus, the multimedia database may require indices that are content dependent keyword indices.
- 7) The Multimedia database requires developing measures of data similarity that are closer to perceptual similarity. Such measures of similarity for different media types need to be quantified and should correspond to perceptual similarity. This will also help the search process.
- 8) Multimedia data is created all over world, so it could have distributed database features that cover the entire world as the geographic area. Thus, the media data may reside in many different distributed storage locations.



- 9) Multimedia data may have to be delivered over available networks in real-time. Please note, in this context, the audio and video data is temporal in nature. For example, the video frames need to be presented at the rate of about 30 frames/sec for smooth motion.
- 10) One important consideration with regard to Multimedia is that it needs to synchronise multiple media types relating to one single multimedia object. Such media may be stored in different formats, or different devices, and have different frame transfer rates.

Multimedia data is now being used in many database applications. Thus, multimedia databases are required for efficient management and effective use of enormous amounts of data.

1.2.5 State of the Art of MMDBMS

The first multimedia database system **ORION** was developed in 1987. The mid 90s saw several commercial MMDBMS being implemented from scratch. Some of them were **MediaDB**, now **MediaWay**, **JASMINE**, and **ITASCA** (the commercial successor of **ORION**). They were able to handle different kinds of data and support mechanisms for querying, retrieving, inserting, and updating data. However, most of these products are not on offer commercially and only some of them have adapted themselves successfully to hardware, software and application changes.

These software are used to provide support for a wide variety of different media types, specifically different media file formats such as image formats, video etc. These files need to be managed, segmented, linked and searched.

The later commercial systems handle multimedia content by providing complex object types for various kinds of media. In such databases the object orientation provides the facilities to define new data types and operations appropriate for the media, such as video, image and audio. Therefore, broadly MMDBMSs are extensible **Object-Relational DBMS (ORDBMSs)**. The most advanced solutions presently include **Oracle 10g**, **IBM DB2** and **IBM Informix**. These solutions purpose almost similar approaches for extending the search facility for video on similarity-based techniques.

Some of the newer projects address the needs of applications for richer semantic content. Most of them are based on the new MPEG-standards MPEG-7 and MPEG-21.

MPEG-7

MPEG-7 is the ISO/IEC 15938 standard for multimedia descriptions that was issued in 2002. It is XML based multimedia meta-data standard, and describes various elements for multimedia processing cycle from the capture, analysis/filtering, to the delivery and interaction.

MPEG-21 is the ISO/IEC 21000 standard and is expected to define an open multimedia framework. The intent is that the framework will cover the entire multimedia content delivery chain including content creation, production, delivery, presentation etc.

Challenges for the Multimedia Database Technologies: Multimedia technologies need to evolve further. Some of the challenges posed by multimedia database applications are:

- the applications utilising multimedia data are very diverse in nature. There is a need for the standardisation of such database technologies,



- technology is ever changing, thus, creating further hurdles in the way of multimedia databases,
- there is still a need to refine the algorithms to represent multimedia information semantically. This also creates problems with respect to information interpretation and comparison.

☛ Check Your Progress 1

- 1) What are the reasons for the growth of multimedia data?

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

- 2) List four application areas of multimedia databases.

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

- 3) What are the contents of multimedia database?

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

- 4) List the challenges in designing multimedia databases.

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

1.3 SPATIAL DATABASE AND GEOGRAPHIC INFORMATION SYSTEMS

A spatial database keeps track of an object in a multi-dimensional space. A spatial database may be used to represent the map of a country along with the information about railways, roads, irrigation facilities, and so on. Such applications are known as Geographic Information Systems (GIS). Let us discuss GIS in this section.

The idea of a geographic database is to provide geographic information; therefore, they are referred to as the Geographic Information System (GIS). A GIS is basically a collection of the Geographic information of the world. This information is stored and analysed on the basis of data stored in the GIS. The data in GIS normally defines the physical properties of the geographic world, which includes:

- spatial data such as political boundaries, maps, roads, railways, airways, rivers, land elevation, climate etc.
- non-spatial data such as population, economic data etc.

But what are the Applications of the Geographic Databases?

The applications of the geographic databases can be categorised into three broad categories. These are:

- **Cartographic Applications:** These applications revolve around the capture and analysis of cartographic information in a number of layers. Some of the basic applications in this category would be to analyse crop yields, irrigation facility



planning, evaluation of land use, facility and landscape management, traffic monitoring system etc. These applications need to store data as per the required applications. For example, irrigation facility management would require study of the various irrigation sources, the land use patterns, the fertility of the land, soil characteristics, rain pattern etc. some of the kinds of data stored in various layers containing different attributes. This data will also require that any changes in the pattern should also be recorded. Such data may be useful for decision makers to ascertain and plan for the sources and types of and means of irrigation.

- **3-D Digital Modelling Applications:** Such applications store information about the digital representation of the land, and elevations of parts of earth surfaces at sample points. Then, a surface model is fitted in using the interpolation and visualisation techniques. Such models are very useful in earth science oriented studies, air and water pollution studies at various elevations, water resource management etc. This application requires data to be represented as attribute based just as in the case of previous applications.
- The third kind of application of such information systems is using the geographic objects applications. Such applications are required to store additional information about various regions or objects. For example, you can store the information about the changes in buildings, roads, over a period of time in a geographic area. Some such applications may include the economic analysis of various products and services etc.

Requirements of a GIS

The data in GIS needs to be represented in graphical form. Such data would require any of the following formats:

- **Vector Data:** In such representations, the data is represented using some geometric objects such as line, square, circle, etc. For example, you can represent a road using a sequence of line segments.
- **Raster Data:** Here, data is represented using an attribute value for each pixel or voxel (a three dimensional point). Raster data can be used to represent three-dimensional elevation using a format termed digital elevation format. For object related applications a GIS may include a temporal structure that records information about some movement related detail such as traffic movement.

A GIS must also support the analysis of data. Some of the sample data analysis operations that may be needed for typical applications are:

- analysing soil erosion
- measurement of gradients
- computing shortest paths
- use of DSS with GIS etc.

One of the key requirements of GIS may be to represent information in an integrated fashion, using both the vector and raster data. In addition it also takes care of data at various temporal structures thus, making it amenable to analysis.

Another question here is the capturing of information in two-dimensional and three-dimensional space in digital form. The source data may be captured by a remote sensing satellite, which can then be, further appended by ground surveys if the need arises. Pattern recognition in this case, is very important for the capture and automating of information input.



Once the data is captured in GIS it may be processed through some special operations. Some such operations are:

- Interpolation for locating elevations at some intermediate points with reference to sample points.
- Some operations may be required for data enhancement, smoothing the data, interpreting the terrain etc.
- Creating a proximity analysis to determine the distances among the areas of interest.
- Performing image enhancement using image processing algorithms for the raster data.
- Performing analysis related to networks of specific type like road network.

The GIS also requires the process of visualisation in order to display the data in a proper visual.

Thus, GIS is not a database that can be implemented using either the relational or object oriented database alone. Much more needs to be done to support them. A detailed discussion on these topics is beyond the scope of this unit.

1.4 GNOME DATABASES

One of the major areas of application of information technology is in the field of Genetics. Here, the computer can be used to create models based on the information obtained about genes. This information models can be used to study:

- the transmission of characteristics from one generation to next,
- the chemical structure of genes and the related functions of each portion of structure, and
- the variations of gene information of all organisms.

Biological data by nature is enormous. Bioinformation is one such key area that has emerged in recent years and which, addresses the issues of information management of genetic data related to DNA sequence. A detailed discussion on this topic is beyond the scope of this unit. However, let us identify some of the basic characteristics of the biological data.

Biological Data – Some Characteristics:

- Biological data consists of complex structures and relationships.
- The size of the data is very large and the data also has a lot of variations across the same type.
- The schema of the database keeps on evolving once or twice a year moreover, even the version of schema created by different people for the same data may be different.
- Most of the accesses to the database would be read only accesses.
- The context of data defines the data and must be preserved along with the data.
- Old value needs to be kept for future references.
- Complex queries need to be represented here.

The Human Genome Initiative is an international research initiative for the creation of detailed genetic and physical maps for each of the twenty-four different human chromosomes and the finding of the complete deoxyribonucleic acid (DNA) sequence of the human genome. The term Genome is used to define the complete genetic information about a living entity. A genetic map shows the linear arrangement of genes or genetic marker sites on a chromosome. There are two types of genetic maps—genetic linkage maps and physical maps. Genetic linkage maps are created on the



basis of the frequency with which genetic markers are co-inherited. Physical maps are used to determine actual distances between genes on a chromosome.

The Human Genome Initiative has six strong scientific objectives:

- to construct a high-resolution genetic map of the human genome,
- to produce a variety of physical maps of the human genome,
- to determine the complete sequence of human DNA,
- for the parallel analysis of the genomes of a selected number of well-characterised non-human model organisms,
- to create instrumentation technology to automate genetic mapping, physical mapping and DNA sequencing for the large-scale analysis of complete genomes,
- to develop algorithms, software and databases for the collection, interpretation and dissemination of the vast quantities of complex mapping and sequencing data that are being generated by human genome research.

Genome projects generate enormous quantities of data. Such data is stored in a molecular database, which is composed of an annotated collection of all publicly available DNA sequences. One such database is the Genbank of the National Institutes of Health (NIH), USA. But what would be the size of such a database? In February 2000 the Genbank molecular database contained 5,691,000 DNA sequences, which were further composed of approximately 5,805,000,000 deoxyribonucleotides.

One of the major uses of such databases is in computational Genomics, which refers to the applications of computational molecular biology in genome research. On the basis of the principles of the molecular biology, computational genomics has been classified into three successive levels for the management and analysis of genetic data in scientific databases. These are:

- Genomics.
- Gene expression.
- Proteomics.

1.4.1 Genomics

Genomics is a scientific discipline that focuses on the systematic investigation of the complete set of chromosomes and genes of an organism. Genomics consists of two component areas:

- **Structural Genomics** which refers to the large-scale determination of DNA sequences and gene mapping, and
- **Functional Genomics**, which refers to the attachment of information concerning functional activity to existing structural knowledge about DNA sequences.

Genome Databases

Genome databases are used for the storage and analysis of genetic and physical maps. Chromosome genetic linkage maps represent distances between markers based on meiotic re-combination frequencies. Chromosome physical maps represent distances between markers based on numbers of nucleotides.



Genome databases should define four data types:

- Sequence
- Physical
- Genetic
- Bibliographic

Sequence data should include annotated molecular sequences.

Physical data should include eight data fields:

- Sequence-tagged sites
- Coding regions
- Non-coding regions
- Control regions
- Telomeres
- Centromeres
- Repeats
- Metaphase chromosome bands.

Genetic data should include seven data fields:

- Locus name
- Location
- Recombination distance
- Polymorphisms
- Breakpoints
- Rearrangements
- Disease association
- Bibliographic references should cite primary scientific and medical literature.

Genome Database Mining

Genome database mining is an emerging technology. The process of genome database mining is referred to as computational genome annotation. Computational genome annotation is defined as the process by which an uncharacterised DNA sequence is documented by the location along the DNA sequence of all the genes that are involved in genome functionality.

1.4.2 Gene Expression

Gene expression is the use of the quantitative messenger RNA (mRNA)-level measurements of gene expression in order to characterise biological processes and explain the mechanisms of gene transcription. The objective of gene expression is the quantitative measurement of mRNA expression particularly under the influence of drugs or disease perturbations.

Gene Expression Databases

Gene expression databases provide integrated data management and analysis systems for the transcriptional expression of data generated by large-scale gene expression experiments. Gene expression databases need to include fourteen data fields:

- Gene expression assays
- Database scope
- Gene expression data
- Gene name
- Method or assay
- Temporal information



- Spatial information
- Quantification
- Gene products
- User annotation of existing data
- Linked entries
- Links to other databases
 - Internet access
 - Internet submission.

Gene expression databases have not established defined standards for the collection, storage, retrieval and querying of gene expression data derived from libraries of gene expression experiments.

Gene Expression Database Mining

Gene expression database mining is used to identify intrinsic patterns and relationships in gene expression data.

Gene expression data analysis uses two approaches:

- Hypothesis testing and
- Knowledge discovery.

Hypothesis testing makes a hypothesis and uses the results of perturbation of a biological process to match predicted results. The objective of knowledge discovery is to detect the internal structure of the biological data. Knowledge discovery in gene expression data analysis employs two methodologies:

- Statistics functions such as cluster analysis, and
- Visualisation.

Data visualisation is used to display the partial results of cluster analysis generated from large gene expression database cluster.

1.4.3 Proteomics

Proteomics is the use of quantitative protein-level measurements of gene expression in order to characterise biological processes and describe the mechanisms of gene translation. The objective of proteomics is the quantitative measurement of protein expression particularly under the influence of drugs or disease perturbations. Gene expression monitors gene transcription whereas proteomics monitors gene translation. Proteomics provides a more direct response to functional genomics than the indirect approach provided by gene expression.

Proteome Databases

Proteome databases also provide integrated data management and analysis systems for the translational expression data generated by large-scale proteomics experiments. Proteome databases integrate expression levels and properties of thousands of proteins with the thousands of genes identified on genetic maps and offer a global approach to the study of gene expression.

Proteome databases address five research problems that cannot be resolved by DNA analysis:

- Relative abundance of protein products,
- Post-translational modifications,
- Subcellular localisations,
- Molecular turnover and
- Protein interactions.



The creation of comprehensive databases of genes and gene products will lay the foundation for further construction of comprehensive databases of higher-level mechanisms, e.g., regulation of gene expression, metabolic pathways and signalling cascades.

Proteome Database Mining

Proteome database mining is used to identify intrinsic patterns and relationships in proteomics data. Proteome database mining has been performed in areas such as Human Lymphoid Proteins and the evaluation of Toxicity in drug users.

Some Databases Relating to Genome

The following table defines some important databases that have been developed for the Genome.

Database Name	Characteristics	Database Problem Areas
GenBank	Keeps information on the DNA/RNA sequences and information on proteins	Schema is always evolving. This database requires linking to many other databases
GDB	Stores information on genetic map linkages as well as non-human sequence data	It faces the same problem of schema evolution and linking of database. This database also has very complex data objects
ACEDB	Stores information on genetic map linkages as well as non-human sequence data. It uses object oriented database technology	It also has the problem of schema evolution and linking of database. This database also has very complex data objects

A detailed discussion on these databases is beyond the scope of this Unit. You may wish to refer to the further readings for more information.

☛ Check Your Progress 2

- 1) What is GIS? What are its applications?

.....
.....

- 2) List the requirements of a GIS.

.....
.....

- 3) What are the database requirements for Genome?

.....
.....



1.5 KNOWLEDGE DATABASES

Knowledge databases are the database for knowledge management. But what is knowledge management? Knowledge management is the way to gather, manage and use the knowledge of an organisation. The basic objectives of knowledge management are to achieve improved performance, competitive advantage and higher levels of innovation in various tasks of an organisation.

Knowledge is the key to such systems. Knowledge has several aspects:

- Knowledge can be implicit (called tacit knowledge) which are internalised or can be explicit knowledge.
- Knowledge can be captured before, during, or even after knowledge activity is conducted.
- Knowledge can be represented in logical form, semantic network form or database form.
- Knowledge once properly represented can be used to generate more knowledge using automated deductive reasoning.
- Knowledge may sometimes be incomplete. In fact, one of the most important aspects of the knowledge base is that it should contain upto date and excellent quality of information.

Simple knowledge databases may consist of the explicit knowledge of an organisation including articles, user manuals, white papers, troubleshooting information etc. Such a knowledge base would provide basic solutions to some of the problems of the less experienced employees.

A good knowledge base should have:

- good quality articles having up to date information,
- a good classification structure,
- a good content format, and
- an excellent search engine for information retrieval.

One of the knowledge base technologies is based on deductive database technology. Let us discuss more about it in the next sub-section.

1.5.1 Deductive Databases

A deductive database is a database system that can be used to make deductions from the available rules and facts that are stored in such databases. The following are the key characteristics of the deductive databases:

- the information in such systems is specified using a declarative language in the form of rules and facts,
- an inference engine that is contained within the system is used to deduce new facts from the database of rules and facts,
- these databases use concepts from the relational database domain (relational calculus) and logic programming domain (Prolog Language),
- the variant of Prolog known as Datalog is used in deductive databases. The Datalog has a different way of executing programs than the Prolog and
- the data in such databases is specified with the help of facts and rules. For example, The fact that Rakesh is the manager of Mohan will be represented as:

Manager(Rakesh, Mohan) having the schema:
Manager(Mgrname, beingmangaed)



Similarly the following represents a rule:

Manager(Rakesh, Mohan) :- Managedby(Mohan, Rakesh)

- Please note that during the representation of the fact the data is represented using the attribute value only and not the attribute name. The attribute name determination is on the basis of the position of the data. For instance, in the example above Rakesh is the Mgrname.
- The rules in the Datalog do not contain the data. These are evaluated on the basis of the stored data in order to deduce more information.

Deductive databases normally operate in very narrow problem domains. These databases are quite close to expert systems except that deductive databases use the database to store facts and rules, whereas expert systems store facts and rules in the main memory. Expert systems also find their knowledge through experts whereas deductive database have their knowledge in the data. Deductive databases are applied to knowledge discovery and hypothesis testing.

1.5.2 Semantic Databases

Information in most of the database management systems is represented using a simple table with records and fields. However, simple database models fall short of applications that require complex relationships and rich constructs to be represented using the database. So how do we address such a problem? Do we employ object oriented models or a more natural data model that represents the information using semantic models? Semantic modeling provides a far too rich set of data structuring capabilities for database applications. A semantic model contains far too many constructs that may be able to represent structurally complex inter-relations among data in a somewhat more natural way. Please note that such complex inter-relationships typically occur in commercial applications.

Semantic modeling is one of the tools for representing knowledge especially in Artificial Intelligence and object-oriented applications. Thus, it may be a good idea to model some of the knowledge databases using semantic database system.

Some of the features of semantic modeling and semantic databases are:

- these models represent information using high-level modeling abstractions,
- these models reduce the semantic overloading of data type constructors,
- semantic models represent objects explicitly along with their attributes,
- semantic models are very strong in representing relationships among objects, and
- they can also be modeled to represent IS A relationships, derived schema and also complex objects.

Some of the applications that may be supported by such database systems in addition to knowledge databases may be applications such as bio-informatics, that require support for complex relationships, rich constraints, and large-scale data handling.

1.6 INFORMATION VISUALISATION

Relational database offers one of the simplest forms of information visualisation in the form of the tables. However, with the complex database technologies and complex database inter-relationship structures, it is important that the information is presented to the user in a simple and understandable form. Information visualisation is the branch of Computer Graphics that deals with the presentation of digital images and interactions to the users in the form that s/he can handle with ease. Information visualisation may result in presentation of information using trees or graph or similar data structures.

Another similar term used in the context of visualisation is knowledge visualisation the main objective of which is to improve transfer of knowledge using visual formats that include images, mind maps, animations etc.

Please note the distinction here. Information visualisation mainly focuses on the tools that are supported by the computer in order to explore and present large amount of data in formats that may be easily understood.

You can refer to more details on this topic in the fifth semester course.

☛ Check Your Progress 3

- 1) What is a good knowledge base?

- 2) What are the features of deductive databases?

- 3) State whether the following are True or False:

- (a) Semantic model is the same as an object.
- (b) IS A relationship cannot be represented in a semantic model.
- (c) Information visualisation is used in GIS.
- (d) Knowledge visualisation is the same as information visualisation.

1.7 SUMMARY

This unit provides an introduction to some of the later developments in the area of database management systems. Multimedia databases are used to store and deal with multimedia information in a cohesive fashion. Multimedia databases are very large in size and also require support of algorithms for searches based on various media components. Spatial database primarily deals with multi-dimensional data. GIS is a spatial database that can be used for many cartographic applications such as irrigation system planning, vehicle monitoring system etc. This database system may represent information in a multi-dimensional way.

Genome database is another very large database system that is used for the purpose of genomics, gene expression and proteomics. Knowledge database store information either as a set of facts and rules or as semantic models. These databases can be utilised in order to deduce more information from the stored rules using an inference engine. Information visualisation is an important area that may be linked to databases from the point of visual presentation of information for better user interactions.



1.8 SOLUTIONS/ANSWERS

Check Your Progress 1

- 1)
 - a) Advanced technology in terms of devices that were digital in nature and support capture and display equipment.
 - b) High speed data communication network and software support for multimedia data transfer.
 - c) Newer application requiring multimedia support.
- 2) Medical media databases
Bio-informatics
Home media
News etc.
- 3) Content can be of two basic types:
 - a) Media Content
 - b) Meta data, which includes media, format data, media keyword data and media feature data.
- 4) Some of the challenges are:
 - a) Support for different types of input/output
 - b) Handling many compressions algorithms and formats
 - c) Differences in OS and hardware
 - d) Integrating to different database models
 - e) Support for queries for a variety of media types
 - f) Handling different kinds of indices
 - g) Data distribution over the world etc.

Check Your Progress 2

- 1) GIS is a spatial database application where the spatial and non-spatial data is represented along with the map. Some of the applications of GIS are:
 - Cartographic applications
 - 3-D Digital modeling applications like land elevation records
 - Geographic object applications like traffic control system.
- 2) A GIS has the following requirements:
 - Data representation through vector and raster
 - Support for analysis of data
 - Representation of information in an integrated fashion
 - Capture of information
 - Visualisation of information
 - Operations on information
- 3) The data may need to be organised for the following three levels:
 - **Geonomics:** Where four different types of data are represented. The physical data may be represented using eight different fields.
 - **Gene expression:** Where data is represented in fourteen different fields
 - **Proteomics:** Where data is used for five research problems.

Check Your Progress 3

- 1) A good knowledge database will have good information, good classification and structure and an excellent search engine.
- 2) They represent information using facts and rules
New facts and rules can be deduced
Used in expert system type of applications.
- 3) a) False b) False c) True d) False.

