# Chapter 1

# Introduction to Database System Concepts

***"This chapter provides a general overview of the nature and purpose of database system. We explain how the traditional file system (for maintaining records) was replaced with the computerized database management system. We also introduce some of the common features of Database Management System, its advantages, disadvantages and implications of the system. The various levels of abstraction is also introduced in database system architecture. It also introduces the various roles of a system administrator. The salient feature of the chapter is the Real Life Application and Hands on Practice."***

## 1.1 Evolution of Data Base Management Systems

To understand the fundamentals of database technology, we must start with the basics of traditional database applications. Humans have maintained records since the beginning of time. They kept track of births and deaths, tax payments, loan payments, prisoners and crimes, assets, visitors, and even natural disasters. Whether these events were written or transmitted orally, depended on the sophistication of a particular culture.

Through the years, the exponential growth in population, the advent of technology, ease of modern travel, and the emergence of large organizations and bureaucracies led to an explosion in the amount of information and its transfer. Just as it became almost too much for humans to handle all this information, computing started to emerge as a viable way to automate information storage and retrieval. Computers were faster in processing information than any human could, they do not make mistakes if programmed correctly, and they never forget.

Thus computer databases came into existence and hence the sophistication of modern database technology is the result of a decades-long evolution in data processing and information management.

*"A database is an integrated collection of related files that is entered into and stored on a computer together with the ways of viewing and analyzing this information. The stored information is called data while the database management system (DBMS) provides ways of viewing, sorting, and analyzing them."*

For instance, when you see a website like Ebay, you will know that a database management system (DBMS) provide us a way to store and retrieve computerized information in convenient and efficient way maintaining a track of all their information and display it so well.

What you see in Ebay is a far call from what databases were able to do at the time of their inception in the late 1950s. There were no DBMS then, just large flat files filled with huge amounts of data separated by commas or tab stops and associated customized programs that helped users analyze the data and generate reports. These programs were limited in scope and difficult to change or augment.

In the 1960s, data storage and access needs became momentous; the scope of database projects seemed much larger than anticipated. The Database Management System evolved as an answer to these needs. The first DBMS were marketed by IBM; these were tailored to the organizations that bought them and the sort of data these organizations dealt with. DBMS made it easier for users to manipulate data, take a backup, reorganize files, and reclassify data and so on. Researchers have taken great interest in the subject due to the expanding scope and size of database related projects and a unified body of theory and ideas regarding data management grew over the 1960s and 1970s, leading to the birth of the modern generic DBMS.

Database management systems have changed a great deal over the last half-century; examining this evolution is a good way to start learning more about DBMS.

*Today the most important function of database systems is to provide the basis for corporate management information systems. Thus, the objective of the DBMS is to provide a convenient and effective method of inserting, deleting, editing and retrieving the electronic information contained in the database.*

## 1.2 Real Life Database Applications

Today, database systems have become part of every industry domain whether it is in education, banking, finance, government, sales, manufacturing, inventory control, travel industry. Some of the real life applications are listed below:

***Banking*** – Today, all banks are computerized with bank tellers keying in the query or request of the customers and providing instant information to them. All this is possible because in the background, a database system is being used as a data storage and data retrieval.

***Airlines industry*** - Booking of travel itineries, re-scheduling travel plans all can be done instantly by the booking clerk using a GUI interface to extract, update, modify or delete a journey plan as per traveler request without much hassles. It is the database which is storing all the information in its repository.

***University setup*** – Activities like Course registration, likely course offerings, likely student preferences, rooms availability and scheduling are handled by the Registrar's office using Databases.

***Sales and Marketing*** - Gathering past data for consumer preferences, previous purchases made all can be retrieved using databases to arrive at useful predictions like future purchase trends and market basket analysis (MBA) statistics.

***Inventory Records*** – Keeping track of inventory analysis, check out / check in details to reduce storage costs and optimize on the overall revenues can be done using databases and related GUI tools.

***Employee Information system*** – Storing employee details, payroll details and generation, employee performance and work deliverables and subsequent benefit entitlements. All these details can provide useful insight into employee performance bonus and incentive entitlements from time to time.

Database systems when integrated with internet technologies have made possible online transactions, online purchases sitting from the comfort of our respective homes. Database systems have played a key role in integrated the world economy into a single entity.

## 1. 3 Approaches to Data Management

***File-Based Systems***

Conventionally, before the Database systems evolved, data in software systems was stored in and represented using flat files. In traditional file processing system, each user defines and implements the files needed for a specific software application as part of programming the application.

Database Systems evolved in the late 1960s to address common issues in applications handling large volumes of data which are also data intensive. Some of these issues could be traced back to the following disadvantages of File-based systems. For example, one user *checking account application* may keep a file on account holder name, account number and address, ID-proof. Programs to check and print account details are implemented as part of the application. A second user, the *savings application* may keep track of the balance sheet and credits. A third user, the Installment Loan Application keeps track of bank / account statements. Although the three users are interested in data about account holder, and bank statements, each user maintains separate files – and programs to manipulate these files because each require some data not available from the other user's file .

Consequently, the cost of computer systems that could perform these functions was easy to justify. The manual effort required for payroll or accounts receivable, for example, was so great that an automated system that could replace the manual system would pay for itself in short time.

Because these systems performed normal record-keeping functions, they were called *data processing systems.*

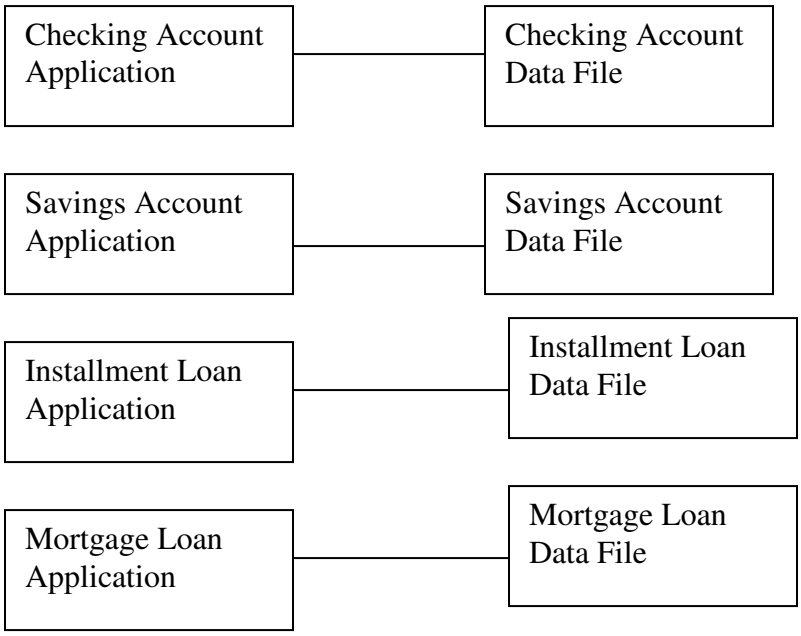


**Fig1.1: File Based System**

As shown in the figure 1.1, in a file-based system, different programs in the same application may be interacting with different private data files. There is no system enforcing any standardized control on the organization and structure of these data files.

***Drawbacks of File Based System***

i. **Data Redundancy and Inconsistency**

Since data resides in different private data files, there are chances of redundancy and resulting inconsistency. For example, in the above example shown, the same customer can have a savings account as well as a mortgage loan. Here the customer details may be duplicated since the programs for the two functions store their corresponding data in two different data files. This gives rise to redundancy in the customer's data. Since the same data is stored in two files, inconsistency arises if a change made in the data in one file is not reflected in the other.

**ii. Unanticipated Queries**

In a file-based system, handling sudden/ad-hoc queries can be difficult, since it requires changes in the existing programs.

### iii. Data Isolation

Though data used by different programs in the application may be related, they reside in isolated data files.

### iv. Concurrent Access Anomalies

In large multi-user systems the same file or record may need to be accessed by multiple users simultaneously. Handling this in a file-based systems is difficult.

### v. Security Problems

In data-intensive applications, security of data is a major concern. Users should be given access only to required data and not the whole database. In a file-based system, this can be handled only by additional programming in each application.

### vi. Integrity Problems

In any application, there will be certain data integrity rules which needs to be maintained. These could be in the form of certain conditions/constraints on the elements of the data records. In the savings bank application, one such integrity rule could be "Customer ID, which is the unique identifier for a customer record, should be non-empty". There can be several such integrity rules. In a file-based system, all these rules need to be explicitly programmed in the application program.

It may be noted that, we are not trying to say that handling the above issues like concurrent access, security, integrity problems, etc., is not possible in a file-based system. The real issue was that, though all these are common issues of concern to any data-intensive application, each application had to handle all these problems on its own. The application programmer needs to bother not only about implementing the application business rules but also about handling these common issues.

**Database Systems**

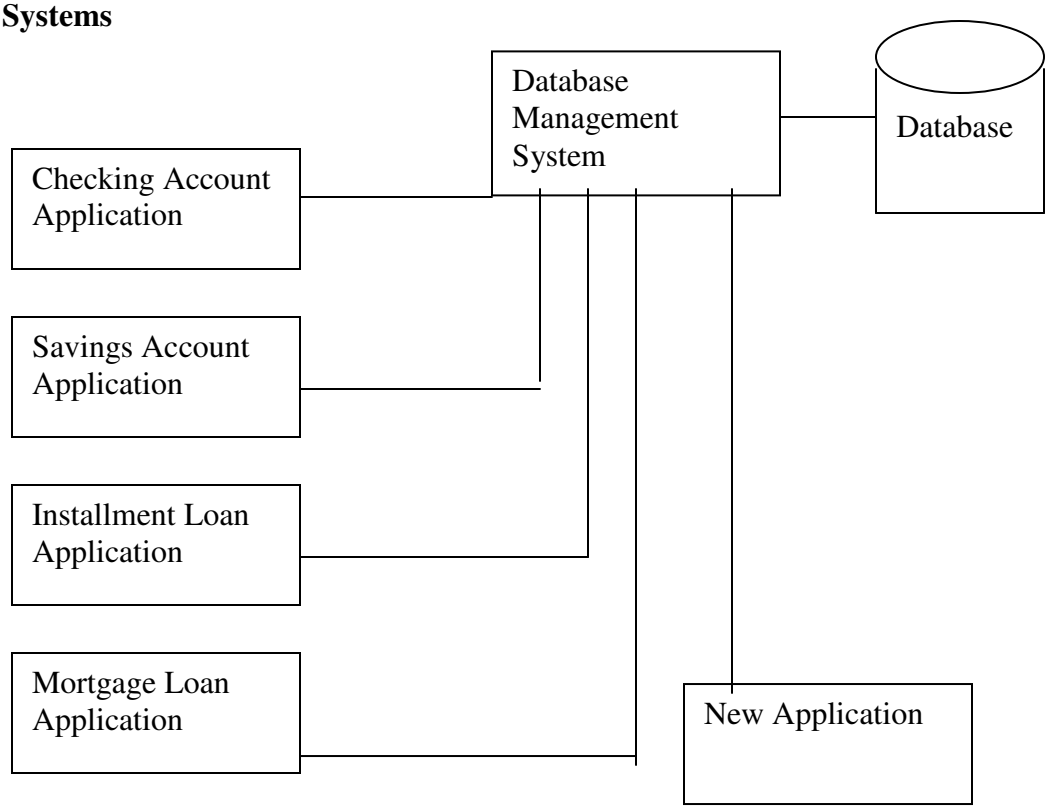


**Fig 1.2: Database Systems**

As shown in the figure 1.2, the DBMS is a central system which provides a common interface between the data and the various front-end programs in the application. It also provides a central location for the whole data in the application to reside.

## 1.4 Advantages of Database Systems

Due to its centralized nature, the database system can overcome the disadvantages of the file-based system as discussed below.

### i. Minimal Data Redundancy

Since the whole data resides in one central database, the various programs in the application can access data in different data files. Hence data present in one file need not be duplicated in another. This reduces data redundancy. However, this does not mean all redundancy can be eliminated. There could be business or technical reasons for having some amount of redundancy. Any such redundancy should be carefully controlled and the DBMS should be aware of it.

### ii. Data Consistency

Reduced data redundancy leads to better data consistency.

### iii. Data Integration

Since related data is stored in one single database, enforcing data integrity is much easier. Moreover, the functions in the DBMS can be used to enforce the integrity rules with minimum programming in the application programs.

### iv. Data Sharing

Related data can be shared across programs since the data is stored in a centralized manner. Even new applications can be developed to operate against the same data.

### v. Enforcement of Standards

Enforcing standards in the organization and structure of data files is required and also easy in a Database System, since it is one single set of programs which is always interacting with the data files.

### vi. Application Development Ease

The application programmer need not build the functions for handling issues like concurrent access, security, data integrity, etc. The programmer only needs to implement the application business rules. This brings in application development ease. Adding additional functional modules is also easier than in file-based systems.

### vii. Better Controls

Better controls can be achieved due to the centralized nature of the system.

### viii. Data Independence

The architecture of the DBMS can be viewed as a 3-level system comprising the following:

- The internal or the physical level where the data resides.

- The conceptual level which is the level of the DBMS functions

- The external level which is the level of the application programs or the end user.

Data Independence is isolating an upper level from the changes in the organization or structure of a lower level. For example, if changes in the file organization of a data file do not demand for changes in the functions in the DBMS or in the application programs, data independence is achieved. Thus Data Independence can be defined as immunity of applications to change in physical representation and access technique. The provision of data independence is a major objective for database systems.

**ix. Reduced Maintenance**

Maintenance is less and easy, again, due to the centralized nature of the system.

## 1.5 Organizational Implications of Database systems

***Standards Enforcement*** – By having a centralized control over the various components of the database system, the DBA helps ensure implementation of quality standards right from the stage of database creation, user roles defining, monitoring and crash recovery procedures. All this has been possible because of various components being bundled into one system namely, the database management system by a single vendor.

***Reduced Application Development Time*** - Over time, database systems have got standardized and accordingly a number of graphical user interface tools (GUI) have emerged in the market developed both by the database vendors and thirt party vendors. These front-end tools have made application development and database administration tasks easier and more efficient to handle by providing features like drag and drop tables, report generation wizard. Many CASE tools like Oracle designer, rational rose, Together are also available to further automate the task of the database developers right from the analysis and design phases of software development.

***Ease of integration with Third party tools*** – As databases now have become part of every domain, most of the evolving tools and technologies also provide third party plug-ins for smooth integration with existing IT infrastructure of any company,

## 1.6 View of Data

A database is an integrated collection of related files that is entered into and stored on a computer together with the ways of viewing and analyzing this information. The stored information is called data while the database management system (DBMS) provides ways of viewing, sorting, and analyzing them. This implies that some details are hidden from end users at various levels and only the required data is made available for view.

## 1.6.1 Data Abstraction

A major role of database system is to provide users with an *abstract* view of the data. That's, it acts like a Capsule (medicine), wherein, the system hides certain details of how the data are stored and maintained.

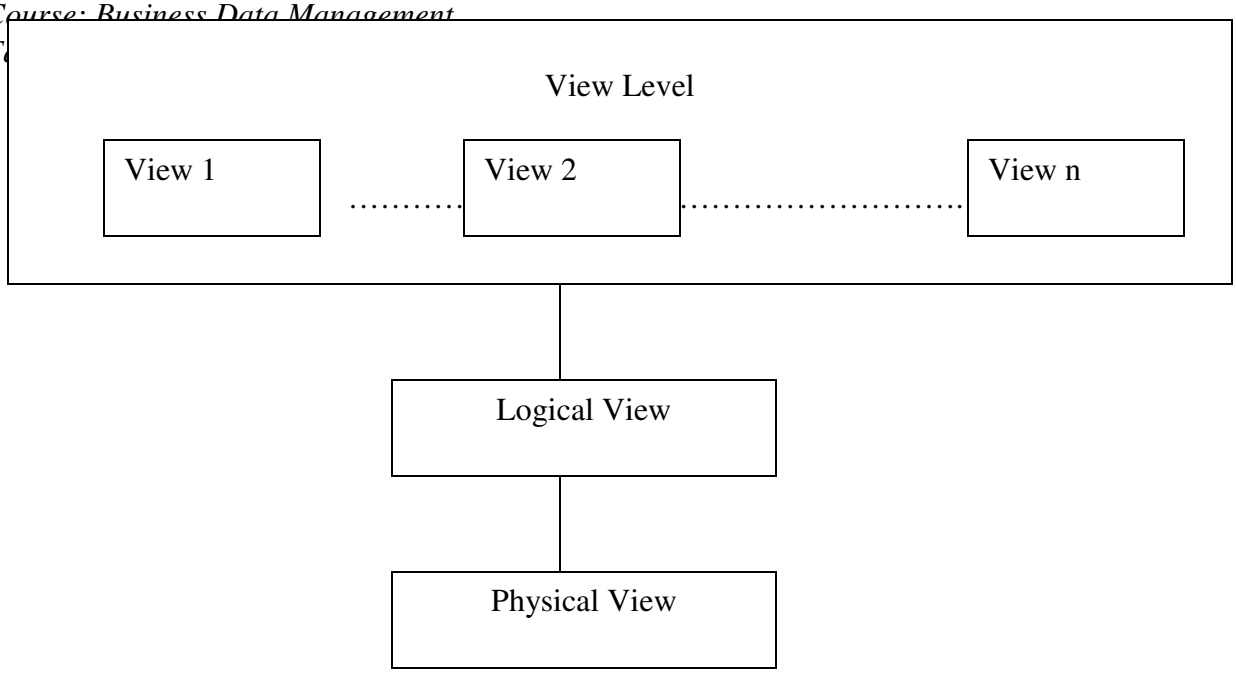


**Fig 1.3: View of Data**

The database management system has access to metadata, relieving users (or their application programs) of its maintenance and manipulation.

***Physical Level*** lies at the lowest level of abstraction and it describes how the data are actually stored. It describes complex low-level data structures in detail.

***Logical Level*** describes what data are stored in the database, and what relationships exist among those data. Database administrators, who decide what information is to keep in the database, usually use logical view of abstraction.

***View Level*** describes only part of the entire database. At this level the system may provide many views for the same database.

## 1.6.2 The Three Level View Architecture for a DBMS

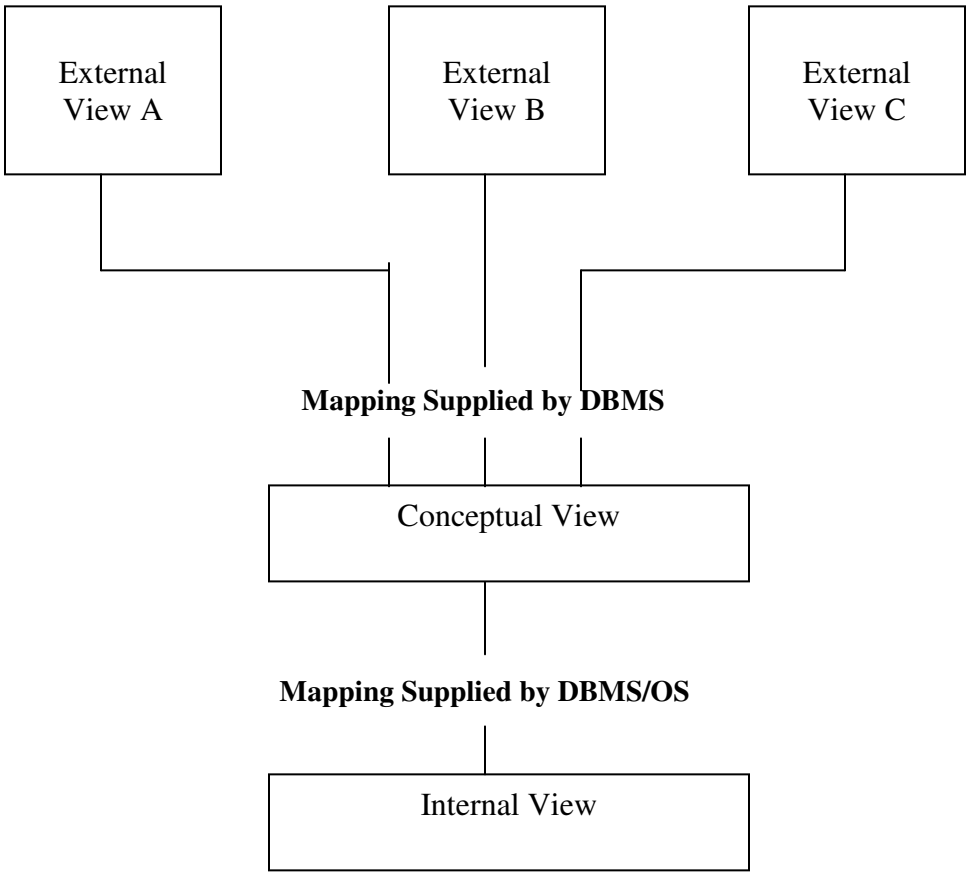


**Fig 1.4: Three Level View Architecture for a DBMS**

The figure 1.4 above describes the generalized architecture of a database system called the ANSI/ SPARC model.

The three level view architecture of a DBMS is divided into three levels, usually referred to as the Internal View, Conceptual view, and the external view.

The three level architecture of DBMS was proposed to help achieve and visualize the following characteristics:

1. insulation of programs and data i.e. program-data and program-operation independence
2. support of multiple user views, and
3. use of a catalog to store the database description (schema).

The view at each of the levels is described by a **scheme.** A scheme is an outline or a plan that describes the records and relationships existing in the view.

***External View:*** The external or user view is at the highest level of database abstraction where only those portions of the database of concern to a user or application program are included. The *external* or *view level* includes a number of *external schemas* or *user views* that consists of the definition of the logical records and the relationship in the external view. The External Level represents the collection of views available to different end-users. The external view describes the part of the database that a particular user group is interested in and hides the rest of the database from that user group.

***Conceptual View:*** At this level of database abstraction all the database entities and the relationship among them are included. The Conceptual level is the representation of the entire information content of the database. The *conceptual view* has a *conceptual schema*, which describes the structure of the whole database. Unlike external view, there is only one conceptual schema per database. The conceptual schema hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations, and constraints.

***Internal View:*** The Internal level is the physical level which shows how the data is stored, what are the representation of the fields etc.This view is the lowest level of abstraction. It has an *internal schema*, which describes the physical storage structure of the database. It describes the storage and access paths for the database.

## 1.7 Database Instance and Database Schema

From time to time, data in the database is either inserted, updated or deleted. In other words, the database keep changing from time to time. However, to do these tasks we have to define a skeleton or blueprint to contain this data. In other words, a **database schema** is the description of a database specified during the database design. It means creating tables, columns or attributes, defining rules or constraints that should hold on the columns of a table. Further, the data in the database at a particular moment in time is called a database state or database instance.
A database system can have many schemas at any moment of time depending on the level of abstraction defined. The **physical schema** defines the database at the physical level, while the **logical schema** defines the database at the logical or conceptual level. It is the logical schema at which the application programs are written or database design is carried out. It is by far the most important level also. Further, the ability to modify the logical schema without affecting the physical level is called **logical data independence**.

## 1.8 Functions of DBMS – Data Storage and Retrieval using Database Languages

The functions performed by a typical DBMS are the following:

### i. Data Definition

The DBMS provides functions to define the structure of the data in the application. These include defining and modifying the record structure, the type and size of fields and the various constraints/conditions to be satisfied by the data in each field. The create table statement does exactly that:

**CREATE TABLE <table name>** (
**<**attribute name 1> <data type 1>,
...
<attribute name n> <data type n>)**;**

e.g.

**create** **table** *account* (
*account-number* **char**(10),
*balance* **integer**)

### ii. Data Manipulation

DML statements are used to work with the data in tables. Once the data structure is defined, data needs to be inserted, modified or deleted. The functions which perform these operations are also part of the DBMS. Some of the functions performed are also part of DML as an illustration:

The *select* statement is used to select data from a table. The tabular result is stored in a result table (called the result-set).

*SELECT column_name(s)*
*FROM table_name*

The *insert* statement is used, obviously, to add new rows to a table.

*INSERT INTO <table name>*
*VALUES (<value 1>, ... <value n>);*

The *update* statement is used to change values that are already in a table.

*UPDATE <table name>*
*SET <attribute> = <expression>*
*WHERE <condition>;*

The *delete* statement does just that, for rows in a table.

*DELETE FROM <table name>*
*WHERE <condition>;*

### iii. Data Security & Integrity

The DBMS contains functions which handle the security and integrity of data in the application. These can be easily invoked by the application and hence the application programmer need not code these functions in his/her programs.

**iv. Data Recovery & Concurrency**

Recovery of data after a system failure and concurrent access of records by multiple users are also handled by the DBMS.

**v. Data Dictionary Maintenance**

A Data Dictionary is "centralized repository of information about data such as meaning, relationships to other data, origin, usage, and format.Maintaining the Data Dictionary which contains the data definition of the application is also one of the functions of a DBMS.

**vi. Performance**

Optimizing the performance of the queries is one of the important functions of a DBMS. Hence the DBMS has a set of programs forming the Query Optimizer which evaluates the different implementations of a query and chooses the best among them.

Thus the DBMS provides an environment that is both convenient and efficient to use when there is a large volume of data and many transactions to be processed.

## 1.9 Role of the Database Administrator

Typically there are three types of users for a DBMS. They are:

The End User who uses the application. Ultimately, this is the user who actually puts the data in the system into use in business. This user need not know anything about the organization of data in the physical level. She also need not be aware of the complete data in the system. She needs to have access and knowledge of only the data she is using.

The Application Programmer who develops the application programs. She has more knowledge about the data and its structure since she has manipulate the data using her programs. She also need not have access and knowledge of the complete data in the system.

The Database Administrator (DBA) who is like the super-user of the system. The role of the DBA is very important and is defined by the following functions.

**i. Defining the Schema**

The DBA defines the schema which contains the structure of the data in the application. The DBA determines what data needs to be present in the system ad how this data has to be represented and organized.

**ii. Liaising with Users**

The DBA needs to interact continuously with the users to understand the data in the system and its use.

**iii. Defining Security & Integrity Checks**

The DBA finds about the access restrictions to be defined and defines security checks accordingly. Data Integrity checks are also defined by the DBA.

**iv. Defining Backup / Recovery Procedures**

The DBA also defines procedures for backup and recovery. Defining backup procedures includes specifying what data is to backed up, the periodicity of taking backups and also the medium and storage place for the backup data.

**v. Monitoring Performance**

The DBA has to continuously monitor the performance of the queries and take measures to optimize all the queries in the application.

## 1.10 Data Models in DBMS

**Database Models**

**Data Model:** A data model is represents a set of concepts to describe the *structure* of database and certain *constraints* that the database should follow.

**Data Model Operations:**

A data model is not just a way of structuring data: it also defines a set of operations that can be performed on the data. The relational model, for example, defines operations such as select, project, and join. Although these operations may not be explicit in a particular query language, they provide the foundation on which a query language is built.

Operations on database may include *basic operations* and *user defined operations*.

**Categories of Database Models:**

1. Entity Relationship Model
2. Hierarchical Model
3. Network Model

4. Relational Model
5. Object/Relational Model
6. Object Oriented Model
7. Semistructured Model
8. Associative Model

***E-R Model:***

The entity-relationship model (or ER model) is a way of graphically representing the logical relationships of entities (or objects) in order to create a database. The ER model was first proposed by Peter Pin-Shan Chen of Massachusetts Institute of Technology (MIT) in the 1970s.

In ER modeling, the structure for a database is portrayed as a diagram, called an entity-relationship diagram (or ER diagram), that resembles the graphical breakdown of a sentence into its grammatical parts. Entities are rendered as points, polygons, circles, or ovals. Relationships are portrayed as lines connecting the points, polygons, circles, or ovals. Any ER diagram has an equivalent relational table, and any relational table has an equivalent ER diagram. ER diagramming is an invaluable aid to engineers in the design, optimization, and debugging of database programs.

The entity-relationship model is based on a perception of the world as consisting of a collection of basic **objects** (entities) and **relationships** among these objects.

- An **entity** is a distinguishable object that exists.
- Each entity has associated with it a set of **attributes** describing it.
- E.g. *number* and *balance* for an account entity.
- A **relationship** is an association among several entities.
- e.g. A *cust_acct* relationship associates a customer with each account he or she has.
- The set of all entities or relationships of the same type is called the **entity set** or **relationship set**.
- Another essential element of the E-R diagram is the **mapping cardinalities**, which express the number of entities to which another entity can be associated via a relationship set.

We can express the overall logical structure of a database **graphically** with an E-R diagram.

Its components are:

- **rectangles** representing entity sets.
- **ellipses** representing attributes.
- **diamonds** representing relationship sets.
- **lines** linking attributes to entity sets and entity sets to relationship sets.

e. g. The ABC firm has no. of sales offices in several states

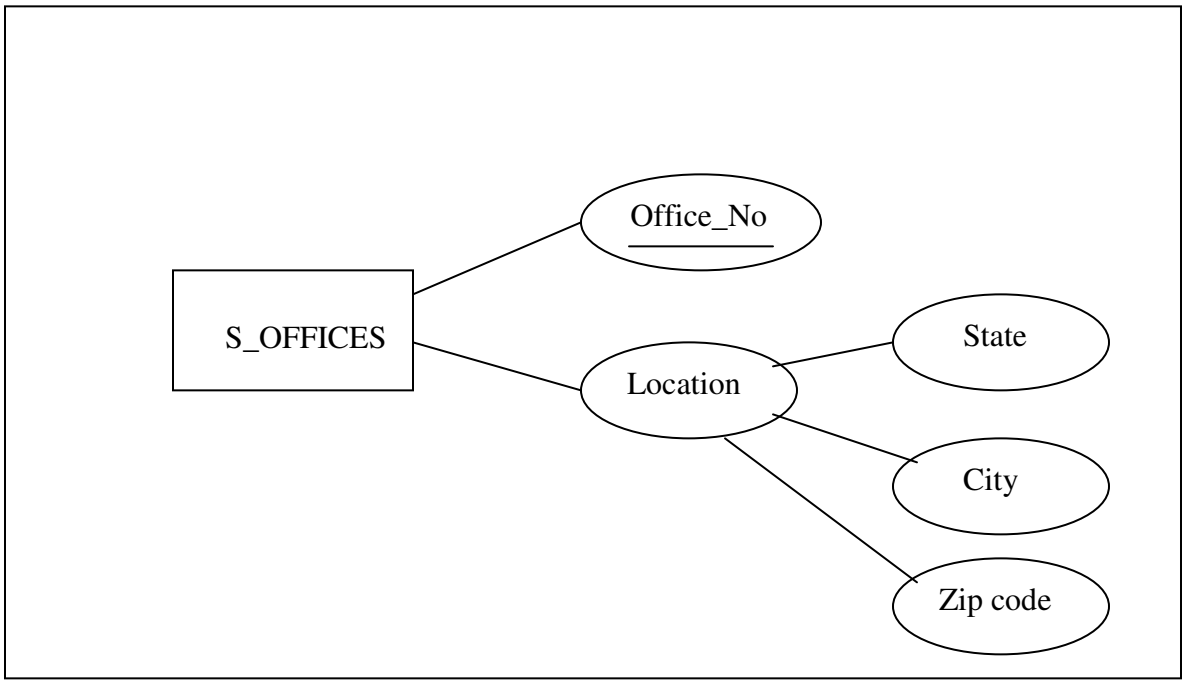


***Fig 1.5:E-R Model of Sales Office***

***Hierarchical Model***

The hierarchical data model organizes data in a tree structure. There is a hierarchy of parent and child data segments. This structure implies that a record can have repeating information, generally in the child data segments. Data in a series of records, which have a set of field values attached to it. It collects all the instances of a specific record together as a record type. These record types are the equivalent of tables in the relational model, and with the individual records being the equivalent of rows. To create links between these record types, the hierarchical model uses Parent Child Relationships. These are a 1:N mapping between record types. This is done by using trees, like set theory used in the relational model, "borrowed" from mathematics. For example, an organization might store information about an employee, such as name, employee number, department, salary. The organization might also store information about an employee's children, such as name and date of birth. The employee and children data forms a hierarchy, where the employee data represents the parent segment and the children data represents the child segment. If an employee has three children, then there would be three child segments associated with one employee segment. In a hierarchical database the parent-child relationship is one to many. This restricts a child segment to having only one parent

segment. Hierarchical DBMSs were popular from the late 1960s, with the introduction of IBM's Information Management System (IMS) DBMS, through the 1970s.

In the figure below either of the entity sets, namely EMPLOYEES and PROJECTS can be selected as the root of a tree. If we select projects as the root, then a logical record type T1 consisting of these two data items SERIAL_NO_1 and EMP_NO are attached to this root at its children. The data item EMP_NO is the key data item. Then we create another logical record type with all the attributes of EMPLOYEES. This record is placed at the root of its tree. To find only the employee working in a given project the above arrangement is ideal. If we include a new record type T2 consisting of the key fields of the record type PROJECTS as a child of the record type EMPLOYEES then the projects assigned to a given employee can also be found easily.

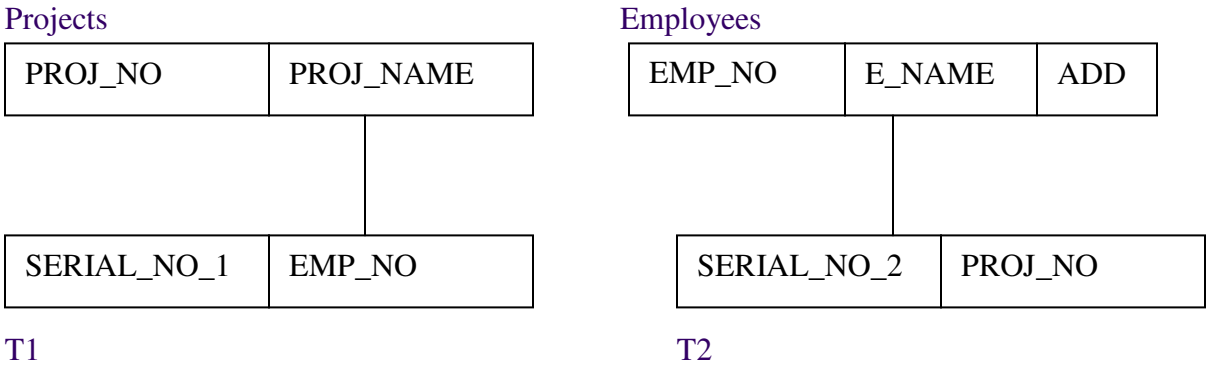


Fig 1.6: The schema diagram of Employee_Project database in Hierarchical Model

***Network Model***

The popularity of the network data model coincided with the popularity of the hierarchical data model. Some data were more naturally modeled with more than one parent per child. So, the network model permitted the modeling of many-to-many relationships in data. In 1971, the Conference on Data Systems Languages (CODASYL) formally defined the network model. The basic data modeling construct in the network model is the set construct. A set consists of an owner record type, a set name, and a member record type. A member record type can have that role in more than one set; hence the multiparent concept is supported. An owner record type can also be a member or owner in another set. The data model is a simple network, and link and intersection record types (called junction records by IDMS) may exist, as well as sets between them. Thus, the complete network of relationships is represented by several pairwise sets; in each set some (one) record type is owner (at the tail of the network arrow) and one or more record types are members (at the head of the relationship arrow). Usually, a set defines a 1:M relationship, although 1:1 is permitted.

***Relational Model***

(RDBMS - relational database management system) A database based on the relational model developed by E.F. Codd. A relational database allows the definition of data structures, storage and retrieval operations and integrity constraints. In such a database the data and relations between them are organized in tables. A table is a collection of records and each record in a table contains the same fields.

Properties of Relational Tables:

- Values Are Atomic
- Each Row is Unique
- Column Values Are of the Same Kind
- The Sequence of Columns is Insignificant
- The Sequence of Rows is Insignificant
- Each Column Has a Unique Name

Certain fields may be designated as keys, which means that searches for specific values of that field will use indexing to speed them up. Where fields in two different tables take values from the same set, a join operation can be performed to select related records in the two tables by matching values in those fields. Often, but not always, the fields will have the same name in both tables. For example, an "orders" table might contain (customer-ID, product-code) pairs and a "products" table might contain (product-code, price) pairs so to calculate a given customer's bill you would sum the prices of all products ordered by that customer by joining on the product-code fields of the two tables. This can be extended to joining multiple tables on multiple fields. Because these relationships are only specified at retreival time, relational databases are classed as dynamic database management system. The RELATIONAL database model is based on the Relational Algebra.

***Object/Relational Model***

Object DBMSs add database functionality to object programming languages. They bring much more than persistent storage of programming language objects. Object DBMSs extend the semantics of the C++, Smalltalk and Java object programming languages to provide full-featured database programming capability, while retaining native language compatibility. A major benefit of this approach is the unification of the application and database development into a seamless data model and language environment. As a result, applications require less code, use more natural data modeling, and code bases are easier to maintain. Object developers can write complete database applications with a modest amount of additional effort.

According to Rao (1994), "The object-oriented database (OODB) paradigm is the combination of object-oriented programming language (OOPL) systems and persistent systems. The power of the OODB comes from the seamless treatment of both persistent data, as found in databases, and transient data, as found in executing programs."

In contrast to a relational DBMS where a complex data structure must be flattened out to

fit into tables or joined together from those tables to form the in-memory structure, object DBMSs have no performance overhead to store or retrieve a web or hierarchy of interrelated objects. This one-to-one mapping of object programming language objects to database objects has two benefits over other storage approaches: it provides higher performance management of objects, and it enables better management of the complex interrelationships between objects. This makes object DBMSs better suited to support applications such as financial portfolio risk analysis systems, telecommunications service applications, world wide web document structures, design and manufacturing systems, and hospital patient record systems, which have complex relationships between data.

An RDBMS might commonly involve SQL statements such as these:

```
CREATE TABLE Customers  (
      Id          CHAR(12)    NOT NULL PRIMARY KEY,
      Surname     VARCHAR(32) NOT NULL,
      FirstName   VARCHAR(32) NOT NULL,
      DOB         DATE        NOT NULL
   );
   SELECT InitCap(Surname) || ', ' || InitCap(FirstName)
     FROM Customers
    WHERE Month(DOB) = Month(getdate())
    AND Day(DOB) = Day(getdate())
```

Most current SQL databases allow the crafting of custom functions, which would allow the query to appear as:

```
SELECT Formal(Id)
     FROM Customers
    WHERE Birthday(Id) = Today()
```

In an object-relational database, one might see something like this, with user-defined data-types and expressions such as BirthDay():

```
CREATE TABLE Customers (
     Id          Cust_Id     NOT NULL  PRIMARY KEY,
     Name        PersonName  NOT NULL,
     DOB         DATE        NOT NULL
   );
   SELECT Formal( C.Name )
     FROM Customers C
    WHERE BirthDay ( C.DOB ) = TODAY;
```

The object-relational model can offer another advantage in that the database can make use of the relationships between data to easily collect related records. In an address book application, an additional table would be added to the ones above to hold zero or more addresses for each user. Using a traditional RDBMS, collecting information for both the

user and their address requires a "join":

```
SELECT InitCap(C.Surname) || ', ' || InitCap(C.FirstName), A.city
     FROM Customers C JOIN Addresses A ON A.Cust_Id=C.Id -- the join
    WHERE A.city="New York"
```

**Object-Oriented Model**

When you integrate database capabilities with object programming language capabilities, the result is an object-oriented database management system or ODBMS. An ODBMS makes database objects appear as programming language objects in one or more existing programming languages. Object database management systems extend the object programming language with transparently persistent data, concurrency control, data recovery, associative queries, and other database capabilities.

To combat the limitations of RDBMS and meet the challenge of the increasing rise of the Internet and the Web, programmers developed object-oriented databases in the 1980s. The main objective of Object-Oriented Database Management Systems, commonly known as OODBMS, is to provide consistent, data independent, secure, controlled and extensible data management services to support the object-oriented model. They were created to handle big and complex data that relational databases could not.

There are important characteristics involved with object-oriented databases. The most important characteristic is the joining of object-oriented programming with database technology, which provides an integrated application development system. Object-oriented programming results in 4 main characteristics: inheritances, data encapsulation, object identity, and polymorphism. Inheritance allows one to develop solutions to complex problems incrementally by defining new objects in terms of previously defined objects.

Data encapsulation or simply encapsulation allows the hiding of the internal state of the objects. Encapsulated objects are those objects that can only be assessed by their methods instead of their internal states. There are three types of encapsulated objects users and developers should recognize. The first is full encapsulation, in which all the operations on objects are done through message sending and method execution. The second is write encapsulation, which is where the internal state of the object is visible only for reading operations. The third is partial encapsulation, which involves allowing direct access for reading and writing for only a part of the internal state.

Object identity allows objects of the database to be independent of each other. Polymorphism and dynamic binding allow one to define operations for one object and then to share the specification of the operation with other objects. This allows users and/or programmers to compose objects to provide solutions without having to write code that is specific to each object.

The language important to OODBMS is data definition and manipulation language (DDML). The use of this language allows persistent data to be created, updated, deleted, or retrieved. An OODBMS needs a computational versus a relational language because it

can be used to avoid impedance mismatch. DDML allows users to define a database, including creating, altering, and dropping tables and establishing constraints. DDMLs are used to maintain and query a database, including updating, inserting, modifying, and querying data.

The OODBMS has many advantages and benefits. First, object-oriented is a more natural way of thinking. Second, the defined operations of these types of systems are not dependent on the particular database application running at a given moment. Third, the data types of object-oriented databases can be extended to support complex data such as images, digital and audio/video, along with other multi-media operations. Different benefits of OODBMS are its reusability, stability, and reliability. Another benefit of OODBMS is that relationships are represented explicitly, often supporting both navigational and associative access to information. This translates to improvement in data access performance versus the relational model.

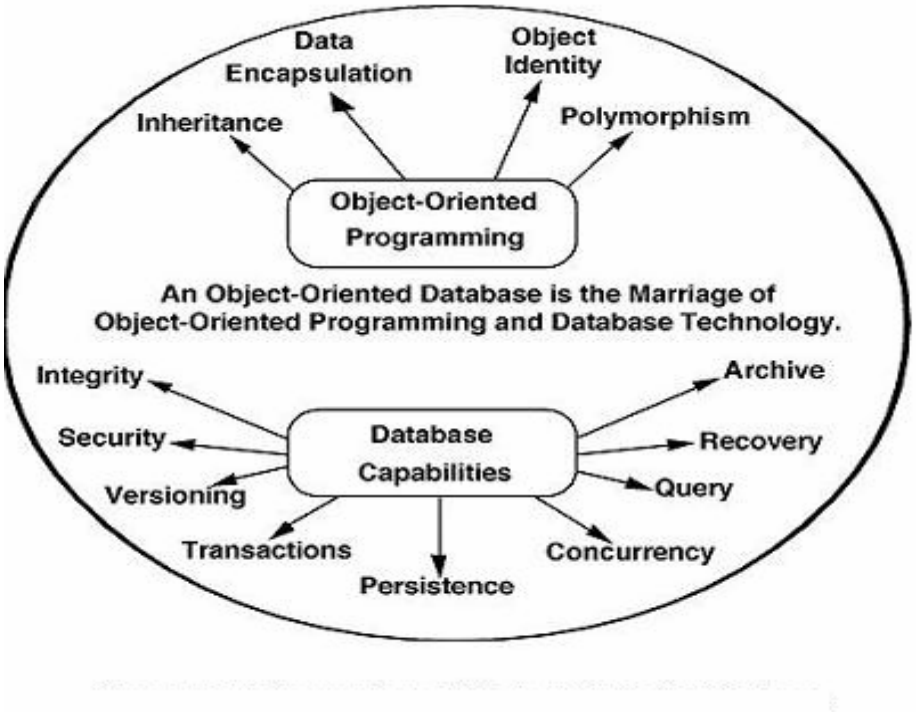


**Fig. 1.7 Makeup of an Object Oriented Database**

Another important benefit is that users are allowed to define their own methods of access to data and how it will be represented or manipulated. The most significant benefit of the OODBMS is that these databases have extended into areas not known by the RDBMS. Medicine, multimedia, and high-energy physics are just a few of the new industries relying on object-oriented databases.

As with the relational database method, object-oriented databases also has disadvantages or limitations. One disadvantage of OODBMS is that it lacks a common data model.

There is also no current standard, since it is still considered to be in the development stages.

An example of a Object database is ObjectDB developed fully in Java and compliant with the JDO standard. It also supports a sophisticated visual tool called Visual JDO Database Explorer that enables browsing, querying, editing and constructing Java objects in a database, visually, without writing a single line of code.

***Semistructured Model***

In semistructured data model, the information that is normally associated with a schema is contained within the data, which is sometimes called ``self-describing''. In such database there is no clear separation between the data and the schema, and the degree to which it is structured depends on the application. In some forms of semistructured data there is no separate schema, in others it exists but only places loose constraints on the data. Semi-structured data is naturally modeled in terms of graphs which contain labels which give semantics to its underlying structure. Such databases subsume the modeling power of recent extensions of flat relational databases, to nested databases which allow the nesting (or encapsulation) of entities, and to object databases which, in addition, allow cyclic references between objects.

Semistructured data has recently emerged as an important topic of study for a variety of reasons. First, there are data sources such as the Web, which we would like to treat as databases but which cannot be constrained by a schema. Second, it may be desirable to have an extremely flexible format for data exchange between disparate databases. Third, even when dealing with structured data, it may be helpful to view it as semistructured for the purposes of browsing.

***Associative Model***

The **associative model of data** is an alternative data model for database systems. Other data models, such as the relational model and the object data model, are record-based. These models involve encompassing attributes about a thing, such as a car, in a record structure. Such attributes might be registration, color, make, model, etc. In the associative model, everything which has "discrete independent existence" is modeled as an entity, and relationships between them are modeled as associations.The associative model divides the real-world things about which data is to be recorded into two sorts: Entities are things that have discrete, independent existence. An entity's existence does not depend on any other thing. Associations are things whose existence depends on one or more other things, such that if any of those things ceases to exist, then the thing itself ceases to exist or becomes meaningless.
An associative database comprises two data structures:

A set of items, each of which has a unique identifier, a name and a type, and,
A set of links, each of which has a unique identifier, together with the unique identifiers of three other things, that represent the source, verb and target of a fact that is

recorded about the source in the database. Each of the three things identified by the source, verb and target may be either a link or an item.

## 1.11 Accessing Database using Application Programs

To interact with a database, we need to write application programs in C, C++, Java or using a web scripting languages like ASP or JSP. However, to connect to a database each database vendor provides a different interface. Two ways to do so are using ODBC and JDBC.

**ODBC**

ODBC (Open Database Connectivity), a C-based interface to database engines, provides a consistent interface for communicating with a database and for accessing *database metadata* (information about the database system vendor, how the data is stored, and so on). Individual vendors provide specific drivers or "bridges" to their particular database management system. Unfortunately, you cannot easily write a program that will run on multiple platforms, even though the database connectivity standardization issue has been largely resolved. For example, if you wrote a database client in C++, you might have to totally rewrite the client for another platform; that is to say, your PC version would not run on a Macintosh. There are two reasons for this. First, C++ as a language is not portable because C++ is not completely specified (for example, how many bits does an int hold?). Second, and more importantly, support libraries such as network access and GUI (Graphical User Interface) frameworks are different on each platform. The alternative is to use JDBC in conjunction with Java programming language.

**JDBC**

A Java program, written properly and according to specification, can run on any Java technology-enabled platform without recompilation. The Java programming language is completely specified and, by definition, a Java technology-enabled platform must support a known core of libraries. One such library is the java.sql package or Java Database Connectivity (JDBC), which you can think of as a portable version of ODBC, and is itself a major standard. Using the Java programming language in conjunction with JDBC provides a truly portable solution to writing database applications.

## 1.12 Database System Architecture

The database system architecture can be best described in terms of the following two diagrams, namely, fig. 1.8 and fig. 1.9 i.e.,

- Architectural Components of a DBMS, and

- Components of a Database Manager

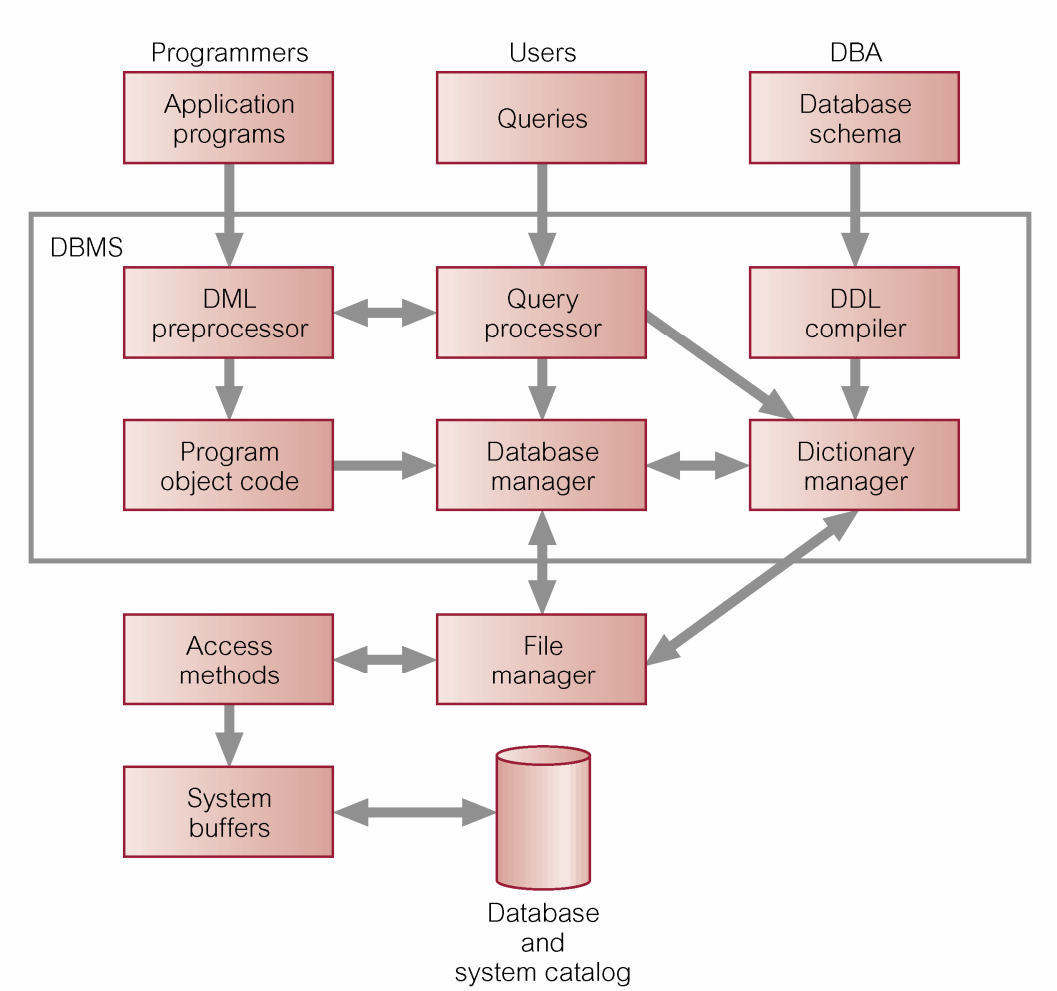


**Fig. 1.8 Architectural Components of a DBMS**

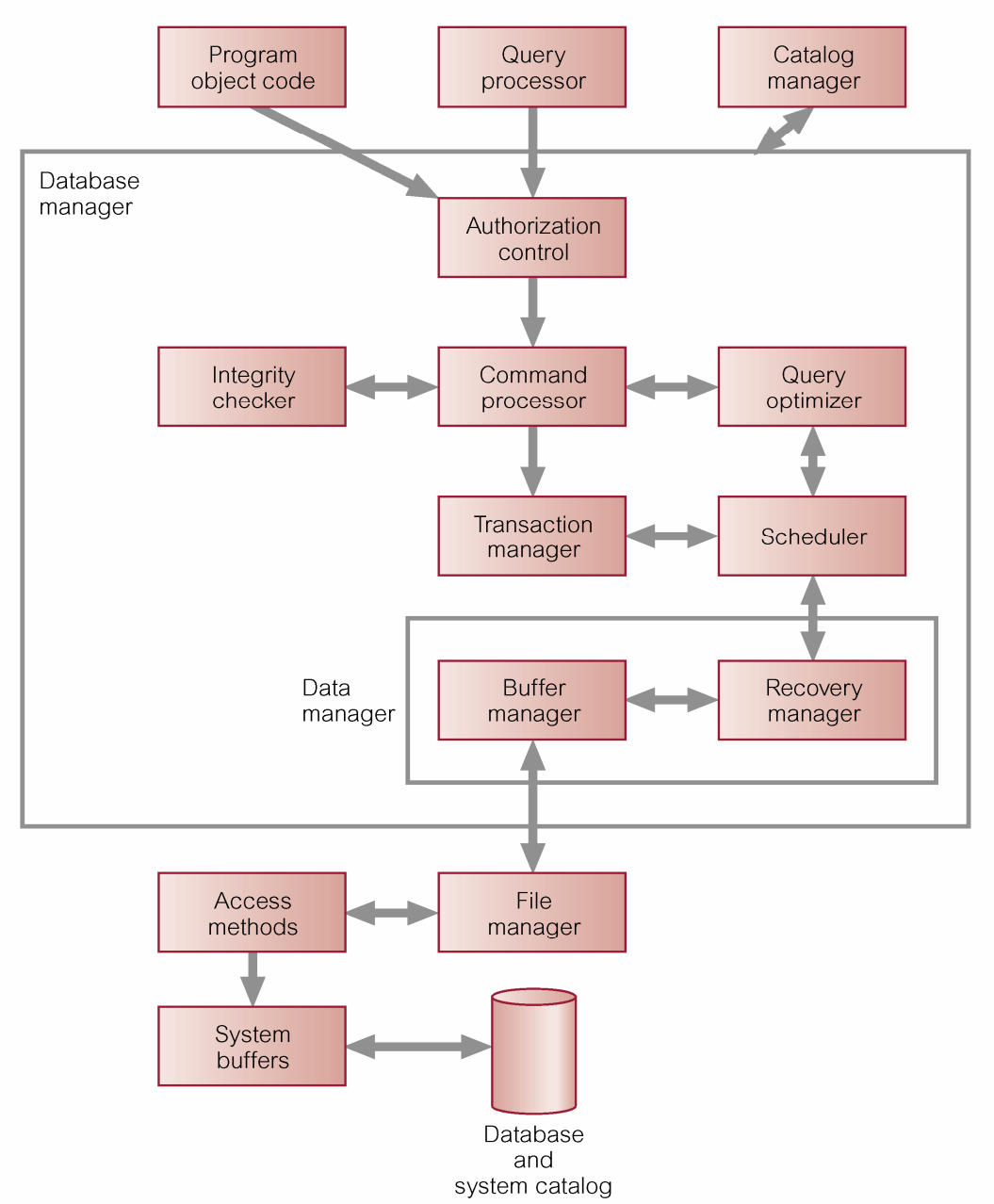


**Fig. 1.9 Components of Database Manager**

## 1.13 Database Application Architectures

Most end users of software applications or application programs to access databases are nowadays, not located at the same site as the database system is located. The end user may be located in a different room or a different building connected via a local area network (LAN). There is a high probability that the end users might be located in a

different city, country or a continent globally and accessing the database via a internet. What we are implying is that the database system may be integrated with application programs to form application architectures that may be a 2-tier, 3-tier or a N-tier architecture. They are briefly explained below:

***Two-tier Architecture*** - The most common example can be thought of as Front-end (GUI) tool like Oracle Developer or Microsoft Visual Basic with Oracle Database as the backend. Basically, a fat client with a fat server as all the application programs resides in these two components.

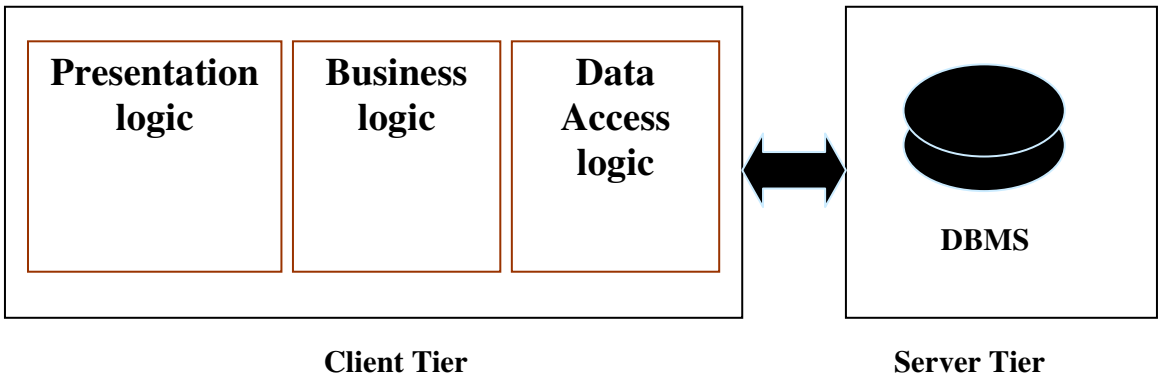


**Fig 1.10: Two Tier Architecture**

***Three-tier Architecture*** - This architecture partitions the application into three sets of services namely, a **User Interface Service Tier** using Client interface like HTML or JSP. The middle tier is the **Business Rule Services Tier** that performs both UI validation and data manipulation in the database. An example is of an servlet or JSP running in an application server like BEA Web logic, IBM Web sphere or just a web server like Jakarta Apache ot Jakarta Tomcat. The middle tier does both UI validation for the client side and execution of various SQL queries for data insertion / modification / deletion on the database side. The last component is the **Data Persistence Services Tier** that is basically the database system itself like Oracle, Microsoft SQLServer, PostgreSQL or MySQL.

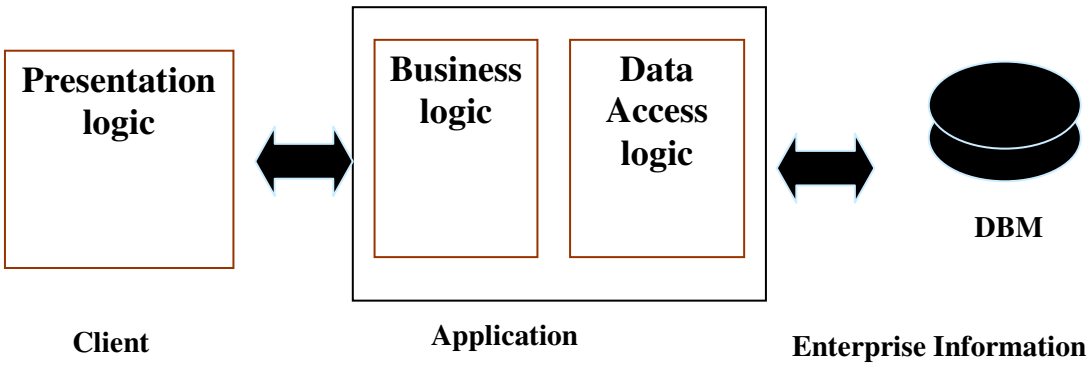

**Fig 1.11: Three Tier Architecture**

***N-Tier Architecture (Multi-Tier Architecture) -*** This architecture further divides the business rule services tier into 2 collaborating tiers - One for business rule processing that supports the UI and the other for business rule processing that integrates and manipulates data. Thus, the various components are the **User Interface Service Tier** using Client side UI like HTML. The second component is the **UI-Oriented Business Rule Services Tier** that uses JavaBeans or Servlets or JSPs to do the input validation of a Client HTML form. The third component is the **Data-Oriented Business Rule Services Tier** that uses JSP / Servlets or EJBs for doing data manipulation using SQL queries from databases. Finally, the last component is the **Data Persistence Services Tier** which is basically the Database like Oracle or any other database. It may also be a legacy mainframe database.

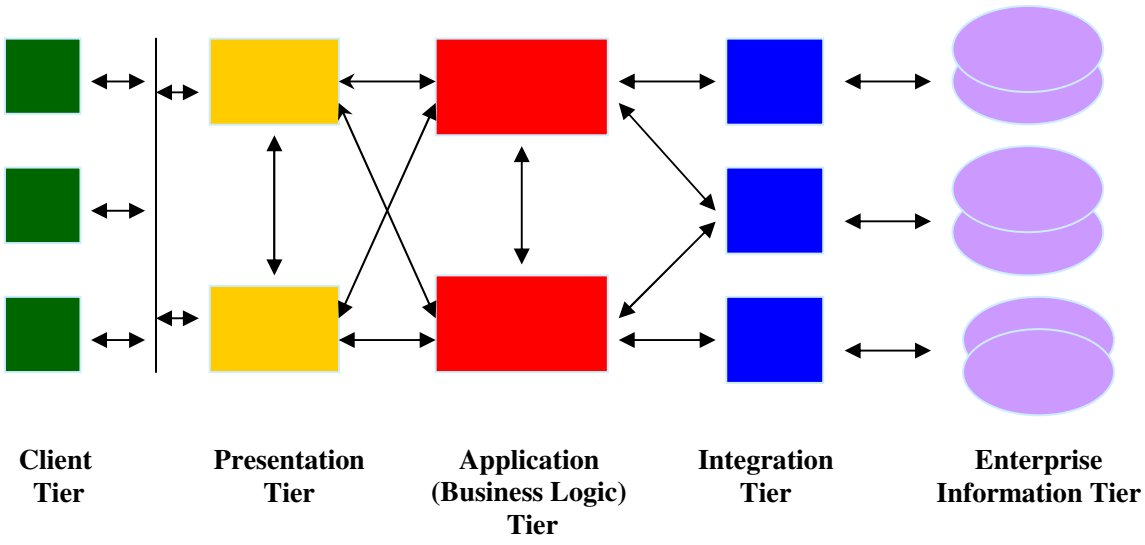


**Fig 1.12: N-Tier Architecture**

## 1.14 Real World Application

### Database Development Life Cycle

Database development lifecycle (DDLC) is a subset of the SDLC or we can say that the DDLC is part of the SDLC. The different phases of DDLC are requirements analysis, database design, DBMS evaluation, selection, implementation, data loading, testing, operation, performance tuning, and maintenance. We will also see the different activities performed in each phase and the output generated in each phase.

INTRODUCTION

The software development is that set of actions required for efficiently transforming the user's need into an effective software solution. Software development process defines the activities required for building the software systems, incorporating the methods and practices to be adopted. It also includes the activities essential for planning the project, tracking its progress and managing the complexities of building the software.

The life span of software systems varies from product to product. During its lifetime, the software goes through various phases. IEEE defines software lifecycle as the period of time that starts when a software product is conceived and ends when the product is no longer available for use. The software development lifecycle (SDLC) typically includes a requirements phase, design phase, implementation phase, testing phase, operation and maintenance phase, and sometimes, retirement phase. The database development life cycle (DDLC)is a part of (or embedded inside) the software development life cycle. Figure 1.11 gives the various SDLC phases and their relationship with DDLC.

## ***DDLC PHASES***

In this section we will consolidate these different database related activities and group them into phases that form part of the database development life cycle (DDLC). The different phases of DDLC are:

- Requirements Analysis
- Database Design
- Evaluation and Selection
- Logical Database Design
- Physical Database Design
- Implementation
- Data Loading
- Testing and Performance Tuning
- Operation
- Maintenance

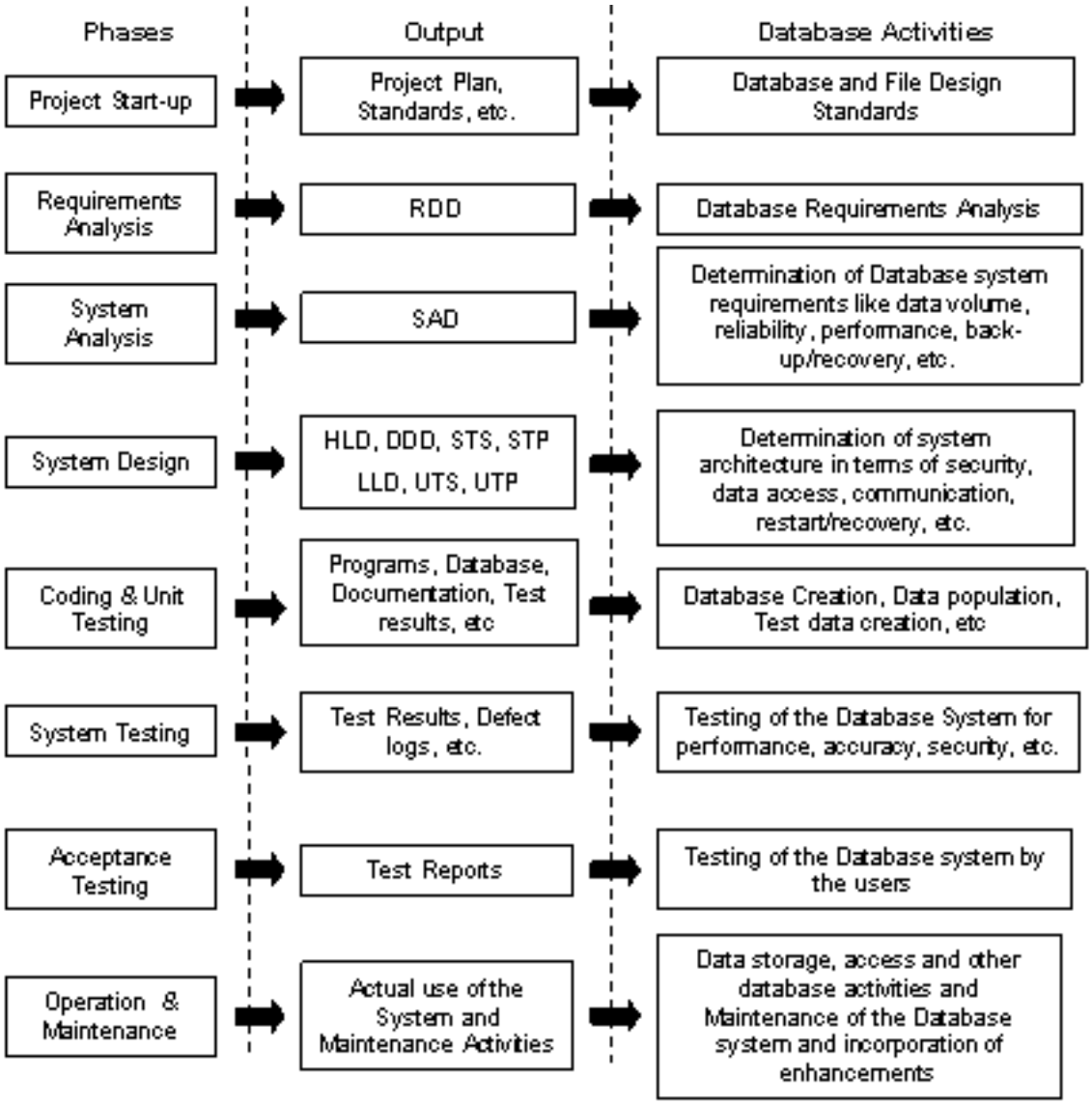


**Figure 1.13 SDLC phases and their relationship with DDLC**

We will now see each of these steps in some detail and find out what exactly is done in each of these phases.

**Requirements Analysis:**

The first step in implementing a database system is to find out what is required. What kind of a database is needed for the organization, what is the volume of the data that must be handled on a day-to-day basis (transaction data), how much data is to be stored in the

master files, and so on. To get these information the database analysts should spend a lot of time in the organization talking to people—the end users—assessing their day-to-day tasks and analyzing the data needs and the amount of data they process and produce. In order to get an accurate estimate of the data needs, the volume of data that is to be handled by the proposed system, the database analysts need to spend time studying the system. Besides the data volume, the analysts should also gather detailed and accurate information on the number of users, the number of people simultaneously accessing the system, the performance expectations of the users, the rate at which the database will grow and so on. The main goals of this phase of DDLC are:

- **Study the existing system** – Here the objective is to identify the data needs and requirements, the data volume, the performance requirements, the security issues, data access restrictions, number and types of users, the growth rate of the database and so on.
- **Define problems and constraints of the database environment** – Here the objective is to find out the problems that will have to be solved when the database is designed. What are the areas where the designers have to be careful, in order to avoid producing a bad database design. Here the analysts should study the paper forms that are being used in the organization and then decide which data items are needed and which are redundant. This is a challenging job as it involves converting the grievances and complaints and expectations of the end users (who will not be familiar with the intricacies of database design) into meaningful requirement definitions. Here the analysts should sit with the users, question the need and necessity of each data item, the problems that can arise if a particular data item is missing, and so on and then arrive at the problem areas, constraints and limitations of the database system.
- **Define the design objectives** – The proposed database system should help in solving the information requirements of the organization and facilitate the smooth and efficient flow of information and help in better decision-making. Most organizations that plan to have a database usually have islands of information—each department having its own database. These islands should be removed and the information should be centralized. In order to do this the analysts should gather the information stored in the departmental databases and then integrate all of them to create an enterprise-wide information system. To do this—to create a centralized database that contains the information for the entire organization—the designers should have the overall view of the organizational goals and the organization's structure. They should also address issues like data integrity, security, concurrency, performance and so on.
- **Define standards and guidelines** – The database design standards, the naming conventions, the documentation standards, diagramming standards and other conventions that are to be followed in the DDLC phases are also formulated during this phase.

**Database Design:**

In this phase the database designers will decide on the database model that is ideally suited for the organization's needs. The database designers will study the documents prepared by the analysts in the requirements analysis phase and then will go about developing a system that satisfies the requirements. In this phase the designers will try to find answers to the following questions:

- What are the problems in the existing system and how they could be overcome?
- What are the information needs of the different users of the system and how could the conflicting requirements be balanced?
- What data items are required for an efficient decision-making system?
- What are the performance requirements for the system?
- How should the data be structured?
- How will each user access the data?
- How is the data entered into the database or how is the data loaded to the database?
- How much data will be added to the database each day/week/year?

First a **conceptual design** of the database is created. In the conceptual design stage, data modeling is used to create an abstract database structure that represents the real-world scenario. The conceptual model will be a true representation of the real world, only if the requirements analysis is properly done, as it needs a thorough understanding of the business and functional areas. At this stage the hardware or the database model that is to be used are not decided—the conceptual design is hardware and software independent. This is important as the system can be implemented on any software/hardware platform chosen later. The different steps in the conceptual design are as follows:

- **Data Analysis and Requirements Definition** – In this step the data items and their characteristics are determined. The data items that are required for successful information processing and decision-making are identified and their characteristics are recorded. Questions like what kind of information is needed, what outputs (reports and queries) should the system generate, who will use the information, how and for what purpose it will be used, what are the sources of the information, etc. will be answered in this step.
- **Data Modeling and Normalization** – In this step the database designer creates a data model of the system. The business contains entities and relationships. Each entity will have attributes. In this step the business entities and relationships are transformed into a data model (usually an E-R model) using E-R diagrams. Now many designers have started using data modeling using UML (Unified Modeling Language) instead of the E-R diagrams. Once the data model is created, then the data will be available in a structured form. All objects (entities, attributes, relations and so on) are defined in a data dictionary and the data is normalized. During the process the designer will group the data items, define the tables, identify the primary keys, define the relationships (one-to-one, one-to-many or many-to-many), create the data model, normalize the data model and so on. Once the data model is created, it is verified against the proposed system in order to ascertain that the proposed model is capable of supporting the real-world system.

So the data model is tested to find out whether the model can perform the various database operations (data loading, access, querying, insertion, modification and so on) and whether the data model takes care of the issues of data security, integrity, concurrency and so on.

**Evaluation and Selection:**

Once the data model is created, tested and verified, the next step is to evaluate the different database management systems and select the one that is ideally suited for the needs of the organization. Here the very important thing to be remembered is that the end-user's representatives should be made part of the group that evaluates and selects the database system for the organization. The main factors that influence the selection of the DBMS are:

- **Cost of the System** – The cost includes the purchase price, cost of operation, maintenance, site license, installation, training, data migration, data conversion, etc.
- **Features and Tools** – Not all database management systems are created equal. Some systems have more features than others. Some have a lot of data administration, querying and report writing tools as part of the system. For example, the availability of Query-By-Example (QBE), screen painters, report generators, query generators, data loaders, data dictionaries and so on make the DBMS easy-to-use and pleasant to work with. Similarly DBA utilities, automated back-up/recovery systems, access control and management systems all make the DBMS more attractive to the buyer. But here a word of caution: you should not go out and buy the DBMS that has the most number of features and tools. You should buy the one that has the most number of features and tools that you want.
- **Customer Support and Training** – Another factor that will influence the selection of the DBMS is the efficiency of the customer service (after sales service) that the DBMS vendor offers. Also the ease of training, that is the ease with which users can be trained in the system is another factor.
- **Underlying Data Model** – The purchasing decision is to a very large extent influenced by the underlying data model—Hierarchical, Network, Relational, Object-Relational and so on.
- **Portability** – The DBMS selected should be portable across platforms and languages if there is a requirement for that.
- **Hardware Requirements** – The hardware requirements of the DBMS is another important factor. If the DBMS needs high-end systems to perform efficiently, then the cost of these hardware components should also be considered while making the selection.

**Logical Database Design**

Once the different database management systems are evaluated and the one best suited for the organization is selected, the next step in the DDLC is the logical database design. Logical design is dependent on the choice of the database model that is used. Once the

database model is identified, the conceptual design can be mapped to the logical design that is tailored to the selected database model. So the logical design is software dependent.

In the logical design stage, the conceptual design is translated into internal model for the selected DBMS. This includes mapping all objects in the model to the specific constructs used by the selected database software. For example, for a RDBMS, the logical design includes the design of tables, indexes, views, transactions, access privileges, etc. Thus the logical design transforms the software-independent conceptual model into a software dependent model.

**Physical Database Design**

Physical database design is the process of selecting the data storage and data access characteristics of the database. The storage characteristics depends on the type of devices supported by the hardware, the type of data access methods supported by the system and the DBMS. Physical design translates the logical design into hardware dependent one. Physical design is particularly important for older database models like hierarchical and network models. Relational, object-relational, object-oriented and deductive models are much more insulated from the physical layer details than the older database models. But even in the case of modern database models, physical design has great significance as a bad design can result in poor performance. In the case of distributed databases the physical design becomes more complex as the networking and communications issues also come into the picture.

**Implementation:**

In most databases a new database implementation requires the creation of special storage related constructs to house the end user tables. These constructs usually include storage group, tablespaces, data files, tables and so on.

**Data Loading:**

After creating the database, the data must be loaded into the database. If the data to be loaded into the database is currently stored in a different system or in a different format, then the data needs to be converted and then migrated to the new database. Data conversion and migration tools and utilities are available with almost all database management systems in the marketplace. There are also a host of third-party tools to accomplish these tasks.

**Testing and Performance Tuning:**

Once the data is loaded into the database the database is tested and fine-tuned for performance, integrity, concurrent access and security constraints. The testing and performance tuning occurs in parallel with the testing and performance tuning of the application programs. Sometimes, the performance degradation of the database is due to

the inefficient code in the application program. So however hard the database administrator tries to fine-tune the database parameters, unless and otherwise, the application program logic is changed to a more efficient one, the performance will not improve. So it is important that the database administrators and application programmers work hand-in-hand during this phase.

**Operation:**

Once the data is loaded to the database and it is tested, the database is released into production (along with the application programs). At this stage the database is considered to be operational and the database, its management, its users and the application programs together form an information system. During the operational phase, the database is accessed by the users and application programs, new data is added, the existing data is modified and some obsolete data is deleted. The database administrators perform the administrative tasks like performance tuning, storage space creation, access control, database back up and so on. It is during the operational phase that the database delivers its usefulness as a critical tool in management decision-making and help in the smooth and efficient functioning of the organization.

**Maintenance:**

Once the database is released into production, it will not remain as it was designed, New business requirements, need for new information, acquisition of new data and similar factors will make it necessary to make modifications and enhancements to the existing design. So the database administrators will definitely receive requests for more storage space, changes in the database design, addition of tables, addition of new users, removal of users who have left the organization, changes in the access privileges of the users and so on. The main tasks in this phase are:

- Database backup and recovery
- Performance tuning
- Database design modifications
- Database access management
- Database audits (access audits, usage audits, security audits, etc.)
- Usage monitoring
- Hardware maintenance
- DBMS Software upgradation

**SUMMARY**

We have seen the different phases of the software development life cycle and the database development life cycle. Database development lifecycle (DDLC) is a subset of the SDLC or we can say that the DDLC is part of the SDLC. The different phases of DDLC are requirements analysis, database design, DBMS evaluation, selection, implementation, data loading, testing, operation, performance tuning, and maintenance.

# 1.15 Hands on Practice

## How to apply in your project

There are a number of database tools available that are widely in use. Some of the most popular tools that are available for commercial use are Oracle from Oracle Corporation (www.oracle.com), Microsoft SQL Server from Microsoft Corporation (www.microsoft.com), DB2 from IBM (www.ibm.com). A few of these are available for free download for personal use or deployment through limited editions, but needs to be purchased for actual deployment.
There are also a number of free / public domain database systems that have become quite popular in recent times. Two such popular databases are MySQL (www.mysql.org) and PostgreSQL (www.postgresql.org). Many small software units as well as academia are using it successfully for both commercial use and research projects.

***Database Installation and Setup***

*In this book, we will use two widely used databases namely, Oracle and PostgreSQL to get you hands on and indirectly making the concepts crystal clear that are explained in the various chapters of this book..*
*To get started with, in this chapter we first give a step by step instructions to setting up of Oracle and PostgreSQL databases and their related graphical user interface (GUI) tools that are used for data insertion and report generation tasks.*

***(A) Installation and Setting up of Oracle Database and its GUI Tools***

*Here, we outline the step by step instructions for installation of the Oracle database and setting up of GUI tool* ***Developer 9i*** *for forms creation and reports generation.*

*Alongside, we also setup the Oracle CASE tool* ***Oracle Designer*** *that can be used to automate the analysis and design of any information systems by creating the necessary models like E-R diagrams, Data flow diagrams, flow charts etc.*

**********

*A.* ***Oracle 9i Database***

**********

*Global Database Name (service name) : gdn.oracle*
*SID (Database Name) : gdn*
*SYS password : manager*
*SYSTEM password : admin*

**********

## *B. Settings for Connecting Developer 9i to Oracle 9i Database*

**********

*During Developer 9i Installation (for net service configuration), select the following:*

*1. select "perform typical configuration for me"*

*2. select Local net service Name Configuration*

*3. select oracle 8i or later database or service*

*4. enter database service name = gdn.oracle*

*5. select protocol as "TCP"*

*6. Select hostname as "computer name"*

*7. select port as 1521*

*8. Click "Test" (It should succeed).*

*(Assuming username = "system and password= "admin")*

*9. Enter net service name as "gdn"*

**********

## *C. Deploying and Running Forms / Reports*

**********

*1. Start the oc4j instance from the Forms / Reports menu i.e.*

*Start --> Programs --> Developer --> Forms Builder --> start oc4j instance*

*(For reports, use*
*Start --> Programs --> Developer -->Reports Builder --> start oc4j instance )*

*2. Click Run Form*

*That's it !!!!!!!!!!!!!!!*

**********

*D.* ***Installing a Repository for Oracle Designer***

**********

*1. Refer the Oracle SCM Repository Installation Guide i.e.*

*Select start --> programs --> developer --> Oracle 9i software configuration manager*
*--> Oracle SCM Repository Installation Guide (chapter 'Server-Side Installation, Migration and Upgrade' under topic 'Installing a new Repository')*

*2. Use the 'interactive SQL scripts' method.*

*In addition, use the following information to install the repository.*

*3. connect as SYS user (at SQL prompt)*

*SQL>connect SYS as sysdba*
*password: manager*

*4. TNS name: gdn*

*Enter password for SYS .....'*
*password: manager*

*5. Create a user named 'repos_manager' (from SQL prompt) with password 'manager'*

*SQL>create user repos_manager identified by manager*

*Note: The sql script file 'ckcreate.sql' is failing in creating 'repos_manager' as user.*
*---- Hence, we are creating it manually and then running the script file 'ckcreate.sql'.*

*6. Perform Step 9 i.e. 'Start the Repository Administration Utility' of the topic 'Installing a new Repository'.*

*7. Perform Step 10 i.e. 'Check Privileges, Tablespaces and Parameters' of the topic 'Installing a new Repository'.*

*8. Perform steps 11, 12, 13 of the topic 'Installing a new Repository'.*

*9. Perform Step 14 i.e., 'Start the Installation' of the topic 'Installing a new Repository'.*

*10. Go thru the topic 'After You Have Installed a Repository' and under 'Test Basic Repository Operations' to test basic repository operations.*

*That's it !! !!!!!!!!!*

***(B) Installation and Setting up of PostgreSQL Database***

*Download the executable file from www.postgresql.org. Double click the file to successfully install it on your PC.*

***Installing JDBC Drivers for your database***

***(A) Setting up JDBC Drivers for Oracle***

*1. Copy the following database drivers in c:\j2sdk1.4.2\jre\lib\ext folder :*

*ojdbc14.jar and nls_charset12.zip files*

*2. Add both of them to oracle class path i.e.*

*c:\oracle\ora92\jdbc\lib\ojdbc14.jar.*

*3. Repeat the above step for nls_charset12.zip file.*

*4. Add c:\j2sdk1.4.2\jre\lib\ext to the classpath*

***(B) Setting up JDBC Drivers for PostgreSQL***

***Method 1 :***

*1. The JDBC driver for PostgreSQL 8.0 installation is available under*

*c:\postgresql\8.0\jdbc directory as postgresql-8.0-310.jdbc2.jar file.*

*2. Unjar it using the command :*

*jar -xvf postgresql-8.0-310.jdbc2.jar*

*3. Add to classpath=c:\postgresql\8.0\jdbc;*

*4. Run any java program using postgresql drivers to check jdbc connectivity i.e.*

*java <program name>*

***Method 2 :***

*1. The JDBC driver for PostgreSQL 8.0 installation is available under*

*c:\postgresql\8.0\jdbc directory as postgresql-8.0-310.jdbc2.jar file.*

*2. Unjar it using the command :*

*jar -xvf postgresql-8.0-310.jdbc2.jar*

*3. Add to classpath=c:\jdk1.4.2\jre\lib\ext;*

*4. Run any java program using postgresql drivers to check jdbc connectivity i.e.*

*java <program name>*

*More details of JDBC and related programs will be discussed in the chapter on SQL.*

*That's it !!!!!*

## 1.16 Points to Remember

***DBMS:*** (DataBase Management System) Software that controls the organization, storage, retrieval, security and integrity of data in a database. It accepts requests from the application and instructs the operating system to transfer the appropriate data.

***DBMS Vendor***: The major DBMS vendors are Oracle, IBM, Microsoft and Sybase (see Oracle database, DB2, SQL Server and ASE). MySQL is a very popular open source product (see MySQL).

***Data Security***
The DBMS can prevent unauthorized users from viewing or updating the database. Using passwords, users are allowed access to the entire database or a subset of it known as a "subschema." For example, in an employee database, some users may be able to view salaries while others may view only work history and medical data.

***Data Integrity***
The DBMS can ensure that no more than one user can update the same record at the same time. It can keep duplicate records out of the database; for example, no two customers with the same customer number can be entered.

***Interactive Query***
A DBMS provides a query language and report writer that lets users interactively interrogate the database. These essential components give users access to all management information as needed.

***Data Independence***
When a DBMS is used, the details of the data structure are not stated in each application program. The program asks the DBMS for data by field name; for example, a coded equivalent of "give me customer name and balance due" would be sent to the DBMS.

Without a DBMS, the programmer must reserve space for the full structure of the record in the program. Any change in data structure requires changing all application programs.

***Hierarchical database***
Hierarchical databases link records like an organization chart. A record type can be owned by only one owner.

***Network database***
In network databases, a record type can have multiple owners. In the example below, orders are owned by both customers and products, reflecting their natural relationship in business.

***Relational databases***

Relational databases do not link records together physically, but the design of the records must provide a common field, such as account number, to allow for matching. Often, the fields used for matching are indexed in order to speed up the process.

***Object databases***

Certain information systems may have complex data structures not easily modeled by traditional data structures. An "object database" can be employed when hierarchical, network and relational structures are too restrictive. Object databases can easily handle many-to-many relationships.

***ODBC***

ODBC (Open Database Connectivity), a C-based interface to database engines, provides a consistent interface for communicating with a database and for accessing database metadata (information about the database system vendor, how the data is stored, and so on).

***JDBC***

A Java program, written properly and according to specification, can run on any Java technology-enabled platform without recompilation. The Java programming language is completely specified and, by definition, a Java technology-enabled platform must support a known core of libraries. One such library is the java.sql package or Java Database Connectivity (JDBC), which you can think of as a portable version of ODBC, and is itself a major standard. Using the Java programming language in conjunction with JDBC provides a truly portable solution to writing database applications.

***Two-tier Architecture***

The most common example can be thought of as Front-end (GUI) tool like Oracle Developer or Microsoft Visual Basic with Oracle Database as the backend.

***Three-tier Architecture***

This architecture partitions the application into three sets of services namely, a User Interface Service Tier using Client interface like HTML or JSP. The middle tier is the Business Rule Services Tier that performs both UI validation and data manipulation in the database. The last component is the Data Persistence Services Tier that is basically the database system itself like Oracle, Microsoft SQLServer, PostgreSQL or MySQL.

***DDL***
The DBMS provides functions to define the structure of the data in the application. These include defining and modifying the record structure, the type and size of fields and the various constraints/conditions to be satisfied by the data in each field.

***DML***
DML statements are used to work with the data in tables. Once the data structure is defined, data needs to be inserted, modified or deleted. The functions which perform these operations are also part of the DBMS.

***Data Recovery & Concurrency***

Recovery of data after a system failure and concurrent access of records by multiple users are also handled by the DBMS.

***Data Dictionary***

A Data Dictionary is "centralized repository of information about data such as meaning, relationships to other data, origin, usage, and format. Maintaining the Data Dictionary which contains the data definition of the application is also one of the functions of a DBMS.

***Database Schema***

A database schema is the description of a database specified during the database design. It means creating tables, columns or attributes, defining rules or constraints that should hold on the columns of a table. Further, the data in the database at a particular moment in time is called a database state or database instance.

## 1.17 Exercises

1. Define the following terms:

a. Database b. Data Redundancy

c. atomicity d. logical schema

e. ODBC f. JDBC

g. DBA h. Data Dictionary

i. Data Independence j. physical data independence

2. List a few applications of database systems.

3. What is a database system? What are the advantages and disadvantages of using a database system?

4. What are the responsibilities of Database Administrator (DBA)?

5. Write queries in SQL for the following Relational Database:-
lives (Personname, street, city) works (personname, company name, salary)
manages (company name, manager name)
located in (company name,. city)

(i) Find the name of all people who work for "Universal Bank".
(ii) Find all people who live in same city as company they work for.

6. What is ER Diagram. Explain with suitable example? Give ER diagram that reduce into tables with explanation.

7. What are the disadvantages of using file based systems as compared to databases systems?

8. Explain ODBC and JDBC interface for database interaction with application programs.

9. What are the likely implications a database implementation would have on an organization ?

10. Explain the various application architectures in use with database systems?

11. Explain the role of a Data Base Administrator.

12. Explain the database system architecture.

13. What are the different data models used to implement database systems.

## 1.17 Hands-on Drill:

Grab any database system available for download and do the necessary installation and setup on your machine. Try to configure any compatible GUI client tool with this

database. For example, MySQL database with its client tool MySQL Administrator or MySQL Query Browser.