UNIT-1

**Introduction to Database Management System**

As the name suggests, the database management system consists of two parts. They are:

* 1. Database and
  2. Management System

What is a Database?

To find out what database is, we have to start from data, which is the basic building block of any DBMS.

**Data**: Facts, figures, statistics etc. having no particular meaning (e.g. 1, ABC, 19 etc).

**Record**: Collection of related data items, e.g. in the above example the three data items had no meaning. But if we organize them in the following way, then they collectively represent meaningful information.

|  |  |  |
| --- | --- | --- |
| **Roll** | **Name** | **Age** |
| 1 | ABC | 19 |

**Table** or **Relation**: Collection of related records.

|  |  |  |
| --- | --- | --- |
| **Roll** | **Name** | **Age** |
| 1 | ABC | 19 |
| 2 | DEF | 22 |
| 3 | XYZ | 28 |

The columns of this relation are called **Fields**, **Attributes** or **Domains**. The rows are called **Tuples**

or **Records**.

**Database**: Collection of related relations. Consider the following collection of tables:

T1 T2

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Roll** | **Name** | **Age** |
| 1 | ABC | 19 |
| 2 | DEF | 22 |
| 3 | XYZ | 28 |

|  |  |
| --- | --- |
| **Roll** | **Address** |
| 1 | KOL |
| 2 | DEL |
| 3 | MUM |

T3 T4

|  |  |
| --- | --- |
| **Roll** | **Year** |
| 1 | I |
| 2 | II |
| 3 | I |

|  |  |
| --- | --- |
| **Year** | **Hostel** |
| I | H1 |
| II | H2 |

We now have a collection of 4 tables. They can be called a “related collection” because we can clearly find out that there are some common attributes existing in a selected pair of tables. Because of these common attributes we may combine the data of two or more tables together to find out the complete details of a student. Questions like “Which hostel does the youngest student live in?” can be answered now, although

***Age*** and ***Hostel*** attributes are in different tables.

A database in a DBMS could be viewed by lots of different people with different responsibilities.

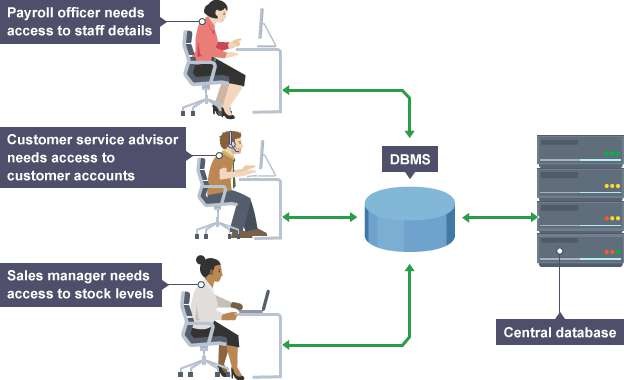


Figure 1.1: Empolyees are accessing Data through DBMS

For example, within a company there are different departments, as well as customers, who each need to see different kinds of data. Each employee in the company will have different levels of access to the database with their own customized **front-end** application.

In a database, data is organized strictly in row and column format. The rows are called **Tuple** or **Record**. The data items within one row may belong to different data types. On the other hand, the columns are often called **Domain** or **Attribute**. All the data items within a single attribute are of the same data type.

**What is Management System?**

A **database-management system** (DBMS) is a collection of interrelated data and a set of programs to access those data. This is a collection of related data with an implicit meaning and hence is a database. The collection of data, usually referred to as the **database**, contains information relevant to an enterprise. The primary goal of a DBMS is to provide a way to store and retrieve database information that is both *convenient* and *efficient*. By **data,** we mean known facts that can be recorded and that have implicit meaning.

The management system is important because without the existence of some kind of rules and regulations it is not possible to maintain the database. We have to select the particular attributes which should be included in a particular table; the common attributes to create relationship between two tables; if a new record has to be inserted or deleted then which tables should have to be handled etc. These issues must be resolved by having some kind of rules to follow in order to maintain the integrity of the database.

Database systems are designed to manage large bodies of information. Management of data involves both defining structures for storage of information and providing mechanisms for the manipulation of information. In addition, the database system must ensure the safety of the information stored, despite system crashes or attempts at unauthorized access. If data are to be shared among several users, the system must avoid possible anomalous results.

Because information is so important in most organizations, computer scientists have developed a large body of concepts and techniques for managing data. These concepts and technique form the focus of this book. This

chapter briefly introduces the principles of database systems.

**Database Management System (DBMS) and Its Applications:**

A Database management system is a computerized record-keeping system. It is a repository or a container for collection of computerized data files. The overall purpose of DBMS is to allow he users to define, store, retrieve and update the information contained in the database on demand. Information can be anything that is of significance to an individual or organization.

Databases touch all aspects of our lives. Some of the major areas of application are as follows:

1. Banking
2. Airlines
3. Universities
4. Manufacturing and selling
5. Human resources

***Enterprise Information***

* + *Sales*: For customer, product, and purchase information.
  + *Accounting*: For payments, receipts, account balances, assets and other accounting information.
  + *Human resources*: For information about employees, salaries, payroll taxes, and benefits, and for generation of paychecks.
  + *Manufacturing*: For management of the supply chain and for tracking production of items in factories, inventories of items inwarehouses and stores, and orders for items.

*Online retailers*: For sales data noted above plus online order tracking,generation of recommendation lists, and

maintenance of online product evaluations.

***Banking and Finance***

* + *Banking*: For customer information, accounts, loans, and banking transactions.
  + *Credit card transactions*: For purchases on credit cards and generation of monthly statements.
  + *Finance*: For storing information about holdings, sales, and purchases of financial instruments such as stocks and bonds; also for storing real-time market data to enable online trading by customers and automated trading by the firm.
* *Universities*: For student information, course registrations, and grades (in addition to standard enterprise information such as human resources and accounting).
* *Airlines*: For reservations and schedule information. Airlines were among the first to use databases in a geographically distributed manner.
* *Telecommunication*: For keeping records of calls made, generating monthly bills, maintaining balances on prepaid calling cards, and storing information about the communication networks.

**Purpose of Database Systems**

Database systems arose in response to early methods of computerized management of commercial data. As an example of such methods, typical of the 1960s, consider part of a university organization that, among other data, keeps information about all instructors, students, departments, and course offerings. One way to keep the information on a computer is to store it in operating system files. To allow users to manipulate the information, the system has a number of application programs that manipulate the files, including programs to:

* + Add new students, instructors, and courses
  + Register students for courses and generate class rosters
  + Assign grades to students, compute grade point averages (GPA), and generate transcripts

System programmers wrote these application programs to meet the needs of the university.

New application programs are added to the system as the need arises. For example, suppose that a university decides to create a new major (say, computer science).As a result, the university creates a new department and creates new permanent files (or adds information to existing files) to record information about all the instructors in the department, students in that major, course offerings, degree requirements, etc. The university may have to write new application programs to deal with rules specific to the new major. New application programs may also have to be written to handle new rules in the university. Thus, as time goes by, the system acquires more files and more application programs.

This typical **file-processing system** is supported by a conventional operating system. The system stores permanent records in various files, and it needs different application programs to extract records from, and add records to, the appropriate files. Before database management systems (DBMSs) were introduced, organizations usually stored information in such systems. Keeping organizational information in a file- processing system has a number of major disadvantages:

**Data redundancy and inconsistency**. Since different programmers create the files and application programs over a long period, the various files are likely to have different structures and the programs may be written in several programming languages. Moreover, the same information may be duplicated in several places (files). For example, if a student has a double major (say, music and mathematics) the address and telephone number of that student may appear in a file that consists of student records of students in the Music department and in a file that consists of student records of students in the Mathematics department. This redundancy leads to higher storage and access cost. In addition, it may lead to **data inconsistency**; that is, the various copies of the same data may no longer agree. For example, a changed student address may be reflected in the Music department records but not elsewhere in the system.

**Difficulty in accessing data**. Suppose that one of the university clerks needs to find out the names of all students who live within a particular postal-code area. The clerk asks the data-processing department to generate such a list. Because the designers of the original system did not anticipate this request, there is no application program on hand to meet it. There is, however, an application program to generate the list of *all* students.

The university clerk has now two choices: either obtain the list of all students and extract the needed information manually or ask a programmer to write the necessary application program. Both alternatives are obviously unsatisfactory. Suppose that such a program is written, and that, several days later, the same clerk needs to trim that list to include only those students who have taken at least 60 credit hours. As expected, a program to generate such a list does not exist. Again, the clerk has the preceding two options, neither of which is satisfactory. The point here is that conventional file-processing environments do not allow needed data to be retrieved in a convenient and efficient manner. More responsive data-retrieval systems are required for general use.

**Data isolation**. Because data are scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult.

**Integrity problems**. The data values stored in the database must satisfy certain types of **consistency constraints**. Suppose the university maintains an account for each department, and records the balance amount in each account. Suppose also that the university requires that the account balance of a department may never fall below zero. Developers enforce these constraints in the system by adding appropriate code in the various application programs. However, when new constraints are added, it is difficult to change the programs to enforce them. The problem is compounded when constraints involve several data items from different files.

**Atomicity problems**. A computer system, like any other device, is subject to failure. In many applications, it is crucial that, if a failure occurs, the data be restored to the consistent state that existed prior to the failure.

Consider a program to transfer $500 from the account balance of department *A* to the account balance of department *B*. If a system failure occurs during the execution of the program, it is possible that the $500 was removed from the balance of department *A* but was not credited to the balance of department *B*, resulting in an inconsistent database state. Clearly, it is essential to database consistency that either both the credit and debit occur, or that neither occur.

That is, the funds transfer must be *atomic*—it must happen in its entirety or not at all. It is difficult to ensure atomicity in a conventional file-processing system.

**Concurrent-access anomalies**. For the sake of overall performance of the system and faster response, many systems allow multiple users to update the data simultaneously. Indeed, today, the largest Internet retailers may have millions of accesses per day to their data by shoppers. In such an environment, interaction of concurrent updates is possible and may result in inconsistent data. Consider department *A*, with an account balance of $10,000. If two department clerks debit the account balance (by say $500 and $100, respectively) of department *A* at almost exactly the same time, the result of the concurrent executions may leave the budget in an incorrect (or inconsistent) state. Suppose that the programs executing on behalf of each withdrawal read the old balance, reduce that value by the amount being withdrawn, and write the result back. If the two programs run concurrently, they may both read the value $10,000, and write back $9500 and $9900, respectively. Depending on which one writes the value last, the account balance of department *A* may contain either $9500 or $9900, rather than the correct value of $9400. To guard against this possibility, the system must maintain some form of supervision.

But supervision is difficult to provide because data may be accessed by many different application programs that have not been coordinated previously.

As another example, suppose a registration program maintains a count of students registered for a course, in order to enforce limits on the number of students registered. When a student registers, the program reads the current count for the courses, verifies that the count is not already at the limit, adds one to the count, and stores the count back in the database. Suppose two students register concurrently, with the count at (say) 39. The two program executions may both read the value 39, and both would then write back 40, leading to an incorrect increase of only 1, even though two students successfully registered for the course and the count should be 41. Furthermore, suppose the course registration limit was 40; in the above case both students would be able to register, leading to a violation of the limit of 40 students.

**Security problems**. Not every user of the database system should be able to access all the data. For example, in a university, payroll personnel need to see only that part of the database that has financial information. They do not need access to information about academic records. But, since application programs are added to the file-processing system in an ad hoc manner, enforcing such security constraints is difficult.

These difficulties, among others, prompted the development of database systems. In what follows, we shall see the concepts and algorithms that enable database systems to solve the problems with file-processing systems.

Advantages of DBMS:

**Controlling of Redundancy:** Data redundancy refers to the duplication of data (i.e storing same data multiple times). In a database system, by having a centralized database and centralized control of data by the DBA the unnecessary duplication of data is avoided. It also eliminates the extra time for processing the large volume of data. It results in saving the storage space.

**Improved Data Sharing** : DBMS allows a user to share the data in any number of application programs.

**Data Integrity** : Integrity means that the data in the database is accurate. Centralized control of the data helps in permitting the administrator to define integrity constraints to the data in the database. For example: in customer database we can can enforce an integrity that it must accept the customer only from Noida and Meerut city.

**Security :** Having complete authority over the operational data, enables the DBA in ensuring that the only mean of access to the database is through proper channels. The DBA can define authorization checks to be carried out whenever access to sensitive data is attempted.

**Data Consistency :** By eliminating data redundancy, we greatly reduce the opportunities for inconsistency. For example: is a customer address is stored only once, we cannot have disagreement on the stored values. Also updating data values is greatly simplified when each value is stored in one place only. Finally, we avoid the wasted storage that results from redundant data storage.

**Efficient Data Access :** In a database system, the data is managed by the DBMS and all access to the data is through the DBMS providing a key to effective data processing

**Enforcements of Standards** : With the centralized of data, DBA can establish and enforce the data standards which may include the naming conventions, data quality standards etc.

**Data Independence** : Ina database system, the database management system provides the interface between the application programs and the data. When changes are made to the data representation, the meta data obtained by the DBMS is changed but the DBMS is continues to provide the data to application program in the previously used way. The DBMs handles the task of transformation of data wherever necessary.

**Reduced Application Development and Maintenance Time :** DBMS supports many important functions that are common to many applications, accessing data stored in the DBMS, which facilitates the quick development of application.

Disadvantages of DBMS

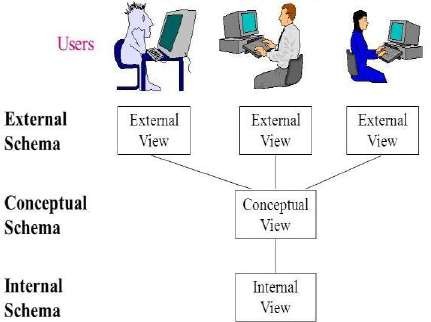
1. It is bit complex. Since it supports multiple functionality to give the user the best, the underlying software has become complex. The designers and developers should have thorough knowledge about the software to get the most out of it.
2. Because of its complexity and functionality, it uses large amount of memory. It also needs large memory to run efficiently.
3. DBMS system works on the centralized system, i.e.; all the users from all over the world access this database. Hence any failure of the DBMS, will impact all the users.
4. DBMS is generalized software, i.e.; it is written work on the entire systems rather specific one. Hence some of the application will run slow.

**View of Data**

A database system is a collection of interrelated data and a set of programs that allow users to access and modify these data. A major purpose of a database system is to provide users with an *abstract* view of the data. That is, the system hides certain details of how the data are stored and maintained.

Data Abstraction

For the system to be usable, it must retrieve data efficiently. The need for efficiency has led designers to use complex data structures to represent data in the database. Since many database-system users are not computer trained, developers hide the complexity from users through several levels of abstraction, to simplify users’ interactions with the system:



Database DISK

Figure 1.2 : Levels of Abstraction in a DBMS

* + **Physical level (or Internal View / Schema)**: The lowest level of abstraction describes *how* the data are actually stored. The physical level describes complex low-level data structures in detail.
  + **Logical level (or Conceptual View / Schema)**: The next-higher level of abstraction describes *what* data are stored in the database, and what relationships exist among those data. The logical level thus describes the entire database in terms of a small number of relatively simple structures. Although implementation of the simple structures at the logical level may involve complex physical-level structures, the user of the logical level does not need to be aware of this complexity. This is referred to as **physical data independence**. Database administrators, who must decide what information to keep in the database, use the logical level of abstraction.
  + **View level (or External View / Schema):** The highest level of abstraction describes only part of the entire database. Even though the logical level uses simpler structures, complexity remains because of the variety of information stored in a large database. Many users of the database system do not need all this information; instead, they need to access only a part of the database. The view level of abstraction exists to simplify their interaction with the system. The system may provide many views for the same database. Figure 1.2 shows the relationship among the three levels of abstraction.

An analogy to the concept of data types in programming languages may clarify the distinction among levels of abstraction. Many high-level programming languages support the notion of a structured type. For example, we may describe a record as follows:

**type** *instructor* = **record**

*ID* : **char** (5);

*name* : **char** (20); *dept name* : **char** (20); *salary* : **numeric** (8,2);

**end**;

This code defines a new record type called *instructor* with four fields. Each field has a name and a type associated with it. A university organization may have several such record types, including

* + - *department*, with fields *dept\_name*, *building*, and *budget*
    - *course*, with fields *course\_id*, *title*, *dept\_name*, and *credits*
    - *student*, with fields *ID*, *name*, *dept\_name*, and *tot\_cred*

At the physical level, an *instructor*, *department*, or *student* record can be described as a block of consecutive storage locations. The compiler hides this level of detail from programmers. Similarly, the database system hides many of the lowest-level storage details from database programmers. Database administrators, on the other hand, may be aware of certain details of the physical organization of the data.

At the logical level, each such record is described by a type definition, as in the previous code segment, and the interrelationship of these record types is defined as well. Programmers using a programming language work at this level of abstraction. Similarly, database administrators usually work at this level of abstraction.

Finally, at the view level, computer users see a set of application programs that hide details of the data types. At the view level, several views of the database are defined, and a database user sees some or all of these views. In addition

to hiding details of the logical level of the database, the views also provide a security mechanism to prevent users from accessing certain parts of the database. For example, clerks in the university registrar office can see only that part of the database that has information about students; they cannot access information about salaries of instructors.

**Instances and Schemas**

Databases change over time as information is inserted and deleted. The collection of information stored in the database at a particular moment is called an **instance** of the database. The overall design of the database is called the database **schema**. Schemas are changed infrequently, if at all. The concept of database schemas and instances can be understood by analogy to a program written in a programming language. A database schema corresponds to the variable declarations (along with associated type definitions) in a program.

Each variable has a particular value at a given instant. The values of the variables in a program at a point in time correspond to an *instance* of a database schema. Database systems have several schemas, partitioned according to the levels of abstraction. The **physical schema** describes the database design at the physical level, while the **logical schema** describes the database design at the logical level. A database may also have several schemas at the view level, sometimes called **subschemas**, which describe different views of the database. Of these, the logical schema is by far the most important, in terms of its effect on application programs, since programmers construct applications by using the logical schema. The physical schema is hidden beneath the logical schema, and can usually be changed easily without affecting application programs. Application programs are said to exhibit **physical data independence** if they do not depend on the physical schema, and thus need not be rewritten if the physical schema changes.

**Data Models**

Underlying the structure of a database is the **data model**: a collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints. A data model provides a way to describe the design of a database at the physical, logical, and view levels.

The data models can be classified into four different categories:

* + **Relational Model**. The relational model uses a collection of tables to represent both data and the relationships among those data. Each table has multiple columns, and each column has a unique name. Tables are also known as **relations**. The relational model is an example of a record-based model.

Record-based models are so named because the database is structured in fixed-format records of several types. Each table contains records of a particular type. Each record type defines a fixed number of fields, or attributes. The columns of the table correspond to the attributes of the record type. The relational data model is the most widely used data model, and a vast majority of current database systems are based on the relational model.

**Entity-Relationship Model**. The entity-relationship (E-R) data model uses a collection of basic objects, called

*entities*, and *relationships* among these objects.

An entity is a “thing” or “object” in the real world that is distinguishable from other objects. The entity- relationship model is widely used in database design.

**Object-Based Data Model**. Object-oriented programming (especially in Java, C++, or C#) has become the dominant software-development methodology. This led to the development of an object-oriented data model that can be seen as extending the E-R model with notions of encapsulation, methods (functions), and object identity. The object-relational data model combines features of the object-oriented data model and relational data model.

**Semi-structured Data Model**. The semi-structured data model permits the specification of data where individual data items of the same type may have different sets of attributes. This is in contrast to the data models mentioned earlier, where every data item of a particular type must have the same set of attributes. The **Extensible Markup Language (XML)** is widely used to represent semi-structured data.

Historically, the **network data model** and the **hierarchical data model** preceded the relational data model. These models were tied closely to the underlying implementation, and complicated the task of modeling data. As a result they are used little now, except in old database code that is still in service in someplaces.

Database Languages

A database system provides a **data-definition language** to specify the database

schema and a **data-manipulation language** to express database queries and updates. In practice, the data- definition and data-manipulation languages are not

two separate languages; instead they simply form parts of a single database language, such as the widely used SQL language.

**Data-Manipulation Language**

A **data-manipulation language (DML)** is a language that enables users to access or manipulate data as organized by the appropriate data model. The types of access are:

* Retrieval of information stored in the database
* Insertion of new information into the database
* Deletion of information from the database
* Modification of information stored in the database

There are basically two types:

* + **Procedural DMLs** require a user to specify *what* data are needed and *how* to get those data.
  + **Declarative DMLs** (also referred to as **nonprocedural DMLs**) require a user to specify *what* data are needed

*without* specifying how to get those data.

Declarative DMLs are usually easier to learn and use than are procedural DMLs. However, since a user does not have to specify how to get the data, the database system has to figure out an efficient means of accessing

data. A **query** is a statement requesting the retrieval of information. The portion of a DML that involves information retrieval is called a **query language**. Although technically incorrect, it is common practice to use the terms *query language* and *data-manipulation language* synonymously.

Data-Definition Language (DDL)

We specify a database schema by a set of definitions expressed by a special language called a **data-definition language** (**DDL**). The DDL is also used to specify additional properties of the data.

We specify the storage structure and access methods used by the database system by a set of statements in a special type of DDL called a **data storage and definition** language. These statements define the implementation details of the database schemas, which are usually hidden from the users.

The data values stored in the database must satisfy certain **consistency constraints**.

For example, suppose the university requires that the account balance of a department must never be negative. The DDL provides facilities to specify such constraints. The database system checks these constraints every time the database is updated. In general, a constraint can be an arbitrary predicate pertaining to the database. However, arbitrary predicates may be costly to test. Thus, database systems implement integrity constraints that can be tested with minimal overhead.

* + **Domain Constraints**. A domain of possible values must be associated with every attribute (for example, integer types, character types, date/time types). Declaring an attribute to be of a particular domain acts as a constraint on the values that it can take. Domain constraints are the most elementary form of integrity constraint. They are tested easily by the system whenever a new data item is entered into the database.
  + **Referential Integrity**. There are cases where we wish to ensure that a value that appears in one relation for a given set of attributes also appears in a certain set of attributes in another relation (referential integrity). For example, the department listed for each course must be one that actually exists. More precisely, the *dept name* value in a *course* record must appear in the *dept name* attribute of some record of the *department* relation.

Database modifications can cause violations of referential integrity. When a referential-integrity constraint is violated, the normal procedure is to reject the action that caused the violation.

* + **Assertions**. An assertion is any condition that the database must always satisfy. Domain constraints and referential-integrity constraints are special forms of assertions. However, there are many constraints that we cannot express by using only these special forms. For example, “Every department must have at least five courses offered every semester” must be expressed as an assertion. When an assertion is created, the system tests it for validity. If the assertion is valid, then any future modification to the database is allowed only if it does not cause that assertion to be violated.
  + **Authorization**. We may want to differentiate among the users as far as the type of access they are permitted on various data values in the database. These differentiations are expressed in terms of **authorization**, the most common being: **read authorization**, which allows reading, but not modification, of data; **insert authorization**, which allows insertion of new data, but not modification of existing data; **update authorization**, which allows modification, but not deletion, of data; and **delete authorization**, which allows deletion of data. We may assign the user all, none, or a combination of these types of authorization.

The DDL, just like any other programming language, gets as input some instructions (statements) and generates some output. The output of the DDL is placed in the **data dictionary**,which contains **metadata**—that is, data about data. The data dictionary is considered to be a special type of table that can only be accessed and updated by the database system itself (not a regular user). The database system consults the data dictionary before reading or modifying actual data.

Data Dictionary

We can define a data dictionary as a DBMS component that stores the definition of data characteristics and relationships. You may recall that such “data about data” were labeled metadata. The DBMS data dictionary provides the DBMS with its self describing characteristic. In effect, the data dictionary resembles and X-ray of the company’s entire data set, and is a crucial element in the data administration function.

The two main types of data dictionary exist, integrated and stand alone. An integrated data dictionary is included with the DBMS. For example, all relational DBMSs include a built in data dictionary or system catalog that is frequently accessed and updated by the RDBMS. Other DBMSs especially older types, do not have a built in data dictionary instead the DBA may use third party stand alone data dictionary systems.

Data dictionaries can also be classified as active or passive. An active data dictionary is automatically updated by the DBMS with every database access, thereby keeping its access information up-to-date. A passive data dictionary is not updated automatically and usually requires a batch process to be run. Data dictionary access information is normally used by the DBMS for query optimization purpose.

The data dictionary’s main function is to store the description of all objects that interact with the database. Integrated data dictionaries tend to limit their metadata to the data managed by the DBMS. Stand alone data dictionary systems are more usually more flexible and allow the DBA to describe and manage all the organization’s data, whether or not they are computerized. Whatever the data dictionary’s format, its existence provides database designers and end users with a much improved ability to communicate. In addition, the data dictionary is the tool that helps the DBA to resolve data conflicts.

Although, there is no standard format for the information stored in the data dictionary several features are common. For example, the data dictionary typically stores descriptions of all:

* Data elements that are define in all tables of all databases. Specifically the data dictionary stores the name, datatypes, display formats, internal storage formats, and validation rules. The data dictionary tells where an element is used, by whom it is used and so on.
* Tables define in all databases. For example, the data dictionary is likely to store the name of the table creator, the date of creation access authorizations, the number of columns, and so on.
* Indexes define for each database tables. For each index the DBMS stores at least the index name the attributes used, the location, specific index characteristics and the creation date.
* Define databases: who created each database, the date of creation where the database is located, who the DBA is and so on.
* End users and The Administrators of the data base
* Programs that access the database including screen formats, report formats application formats, SQL queries and so on.
* Access authorization for all users of all databases.
* Relationships among data elements which elements are involved: whether the relationship are mandatory or optional, the connectivity and cardinality and so on.

**Database Administrators and Database Users**

A primary goal of a database system is to retrieve information from and store new information in the database. People who work with a database can be categorized as database users or database administrators.

Database Users and User Interfaces

There are four different types of database-system users, differentiated by the way they expect to interact with the system. Different types of user interfaces have been designed for the different types of users.

**Naive users** are unsophisticated users who interact with the system by invoking one of the application programs that have been written previously. For example, a bank teller who needs to transfer $50 from account *A* to account *B* invokes a program called *transfer*. This program asks the teller for the amount of money to be transferred, the account from which the money is to be transferred, and the account to which the money is to be transferred.

As another example, consider a user who wishes to find her account balance over the World Wide Web. Such a user may access a form, where she enters her account number. An application program at the Web server then retrieves the account balance, using the given account number, and passes this information back to the user. The typical user interface for naive users is a forms interface, where the user can fill in appropriate fields of the form. Naive users may also simply read *reports* generated from the database.

**Application programmers** are computer professionals who write application programs. Application programmers can choose from many tools to develop user interfaces. **Rapid application development (RAD)** tools are tools that enable an application programmer to construct forms and reports without writing a program. There are also special types of programming languages that combine imperative control structures (for example, for loops, while loops and if-then-else statements) with statements of the data manipulation language. These languages, sometimes called *fourth-generation languages*, often include special features to facilitate the generation of forms and the display of data on the screen. Most major commercial database systems include a fourth generation language.

**Sophisticated users** interact with the system without writing programs. Instead, they form their requests in a database query language. They submit each such query to a **query processor**, whose function is to break down DML statements into instructions that the storage manager understands. Analysts who submit queries to explore data in the database fall in this category.

**Online analytical processing (OLAP)** tools simplify analysts’ tasks by letting them view summaries of data in different ways. For instance, an analyst can see total sales by region (for example, North, South, East, and West), or by product, or by a combination of region and product (that is, total sales of each product in each region). The tools also permit the analyst to select specific regions, look at data in more detail (for example, sales by city within a region) or look at the data in less detail (for example, aggregate products together by category).

Another class of tools for analysts is **data mining** tools, which help them find certain kinds of patterns in data. **Specialized users** are sophisticated users who write specialized database applications that do not fit into the traditional data-processing framework.

Among these applications are computer-aided design systems, knowledge base and expert systems, systems that store data with complex data types (for example, graphics data and audio data), and environment-modeling systems.

**Database Architecture:**

We are now in a position to provide a single picture (Figure 1.3) of the various components of a database system and the connections among them.

The architecture of a database system is greatly influenced by the underlying computer system on which the database system runs. Database systems can be centralized, or client-server, where one server machine executes work on behalf of multiple client machines. Database systems can also be designed to exploit parallel computer architectures. Distributed databases span multiple geographically separated machines.

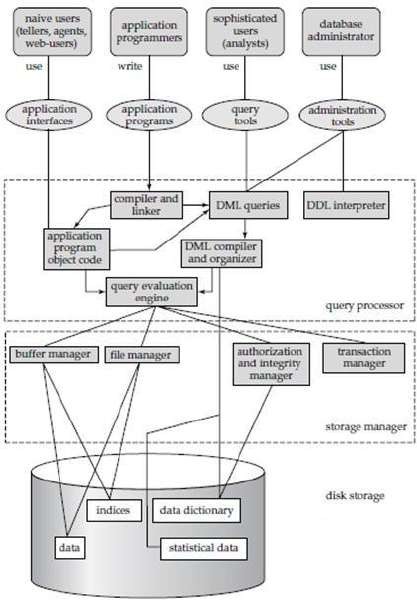


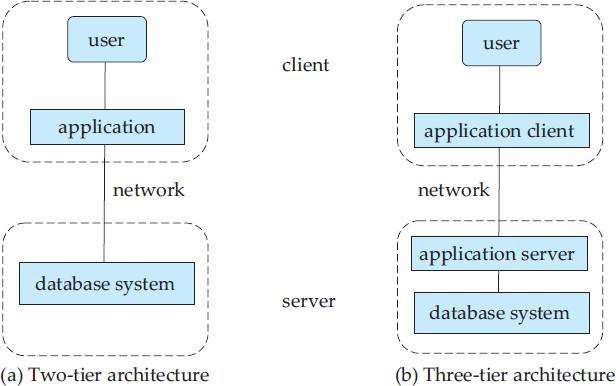
Figure 1.3: Database System Architecture

A database system is partitioned into modules that deal with each of the responsibilities of the overall system. The functional components of a database system can be broadly divided into the **storage manager** and the **query processor** components. The storage manager is important because databases typically require a large amount of storage space. The query processor is important because it helps the database system simplify and facilitate access to data.

It is the job of the database system to translate updates and queries written in a nonprocedural language, at the logical level, into an efficient sequence of operations at the physical level.

Database applications are usually partitioned into two or three parts, as in Figure 1.4. In a two-tier architecture, the application resides at the client machine, where it invokes database system functionality at the server machine through query language statements. Application program interface standards like ODBC and JDBC are used for interaction between the client and the server. In contrast, in a three-tier architecture, the client machine acts as merely a front end and does not contain any direct database calls. Instead, the client end communicates with an application server, usually through a forms interface.

The application server in turn communicates with a database system to access data. The business logic of the application, which says what actions to carry out under what conditions, is embedded in the application server, instead of being distributed across multiple clients. Three-tier applications are more appropriate for large applications, and for applications that run on the WorldWideWeb.



**Figure 1.4: Two-tier and three-tier architectures.**

**Query Processor:**

The query processor components include

* **DDL interpreter,** which interprets DDL statements and records the definitions in the data dictionary.
* **DML compiler,** which translates DML statements in a query language into an evaluation plan consisting of low-level instructions that the query evaluation engine understands.

A query can usually be translated into any of a number of alternative evaluation plans that all give the same result. The DML compiler also performs **query optimization**, that is, it picks the lowest cost evaluation plan from among the alternatives.

**Query evaluation engine,** which executes low-level instructions generated by the DML compiler.

**Storage Manager:**

A *storage manager* is a program module that provides the interface between the lowlevel data stored in the database and the application programs and queries submitted to the system. The storage manager is responsible for the interaction with the file manager. The raw data are stored on the disk using the file system, which is usually provided by a conventional operating system. The storage manager translates the various DML statements into low-level file-system commands. Thus, the storage manager is responsible for storing, retrieving, and updating data in the database.

The storage manager components include:

* **Authorization and integrity manager**, which tests for the satisfaction of integrity constraints and checks the authority of users to access data.
* **Transaction manager**, which ensures that the database remains in a consistent (correct) state despite system failures, and that concurrent transaction executions proceed without conflicting.
* **File manager**, which manages the allocation of space on disk storage and the data structures used to represent information stored on disk.
* **Buffer manager**, which is responsible for fetching data from disk storage into main memory, and deciding what data to cache in main memory. The buffer manager is a critical part of the database system, since it enables the database to handle data sizes that are much larger than the size of main memory.

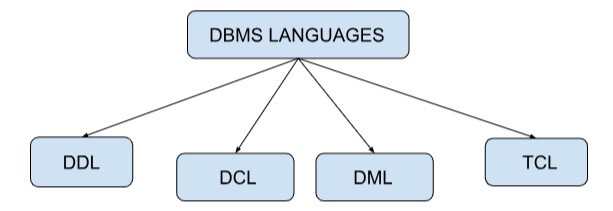
**Transaction Manager:**

A **transaction** is a collection of operations that performs a single logical function in a database application. Each transaction is a unit of both atomicity and consistency. Thus, we require that transactions do not violate any database-consistency constraints. That is, if the database was consistent when a transaction started, the database must be consistent when the transaction successfully terminates. **Transaction - manager** ensures that the database remains in a consistent (correct) state despite system failures (e.g., power failures and operating system crashes) and transaction failures.

Database Languages

Database languages are used to read, store and update the data in the database. Specific languages are used to perform various operations of the database.

Types of Database Languages



DDL(Data Definition Language)

Data Definition Language(DDL) is used for describing structures or patterns and its relationship in a database. It is also used to define the database schema, tables, index, Constraints, etc. It can also be used to store information like the number of tables, names, columns, indexes, etc. The commands only affect the database structure and not the data.

**The commands used in DDL are:**

**Create:**It is used to create a database or table.

**Alter:**It is used to make a change in the structure of a database.

**Drop:**It is used to completely delete a table from the database

**Rename:**It is used to rename a table.

**Truncate:**It is used to delete the entities inside the table while holding the structure of the table.

**Comment:**It is used to comment on the data dictionary.

DML(Data Manipulation Language)

DML is used to manipulate the data present in the table or database. We can easily perform operations such as store, modify, update, and delete on the database.

**The commands used in DML are:**

**Select:**It shows the record of the specific table. Also, it can be used with a WHERE clause to get the particular record.

**Insert:** It allows users to insert data into the database or tables.

**Update:**It is used to update or modify the existing data in database tables.

**Delete:** It is used to delete records from the database tables. Also, it can be used with a WHERE clause to delete a particular row from the table.

**Merge:**It allows the insert and update(UPSERT) operations.

DCL(Data Control Language)

DCL works to deal with SQL commands that are used to permit a user to access, modify and work on a database. it is used to access stored data. It gives access, revokes access, and changes the permission to the owner of the database as per the requirement.

**The commands used in DCL are:**

**Grant:**It is used to give access to security privileges to a specific database user.

**Revoke:**It is used to revoke the access from the user that is being granted by the grant command.

TCL(Transaction Control Language)

It can be grouped into a logical transaction and is used to run the changes made by the DML command in the database.

**Commit:**Transaction on the database is saved using Commit.

**Rollback:**The database gets restored to the original since the last commit.

Interface

An interface is a program that allows users to input queries into a database without writing the code in the query language. An interface can be used to manipulate the database for adding, deleting, updating, or viewing the data.

Types of Interface are

Form−based Interface

A form is displayed to each user by the form−based interface. The user fills in the details and submits the form to make a new entry into the database. It can also be done when the user only fills in some details and the system will help by retrieving the rest of the details from the database. The form−based interface is built for the naive user(inexperienced user) which deals with a limited number of operations. Many DBMS have specification language which helps the programmer define such forms.

**Example**

Student entering his roll. no, branch in the form to get the grade card.

Menu−based User Interface

In this interface, the user was provided with a list of options (called a menu) through which the user forms a request. The user doesn’t need to memorize the command and syntax and the query is composed step by step by picking options from a menu. Pull−down menu interfaces are mostly used in web−based user interfaces and are often used in browsing interfaces by which the database content can be looked through.

**Example**

In a shopping website, categories are selected from the menu, brands are selected from the menu of brands, and budget ranges are applied from the menu of budget range.

GUI(Graphical User Interface)

Users are provided a schema of diagrammatic form by which query can be specified through manipulating the diagram. GUI utilizes both menu and form in several cases. Schema Diagram's specific parts are selected using devices used by GUI.

**Example**

You liked a video on Instagram by tapping with your finger, and the color changes to red. The visual graphic gets changed due to user action.

Natural Language Interface

A natural language interface contains its unique schema more like the high−level conceptual schema. It also has a directory of important words. It generates a query based on the interpretation of important words in the input by the user and if the interpretation is successful, then it displays the result to the user.

**Example**

A user googled the fastest car in India, and now the natural language interface will look for the important words i. e. fastest, car, India, and show the result accordingly.

Speech Input and Output

The users query the interface with speech and get the answer in speech. The input is detected using predefined words and conversions are done into speech to provide the output. Nowadays, it has become the most common type of interface.

**Example**

OK Google, Siri on Apple, and Alexa is used in the form of speech.

Interface for DBA

DBA staff are provided commands that can only be used by them only to create an account, grant account authorization, and change a schema, and storage structure reorganization.