**Unit-1**

**Introduction to database: File Systems organization: Sequential- Pointer- Indexed-Direct, purpose of database system – Database Characteristics- Users of Database system – Advantage of DBMS Approach – Scheme and Instances – Three Scheme Architecture and Data Independence – The Database System Environment- Relational Algebra.**

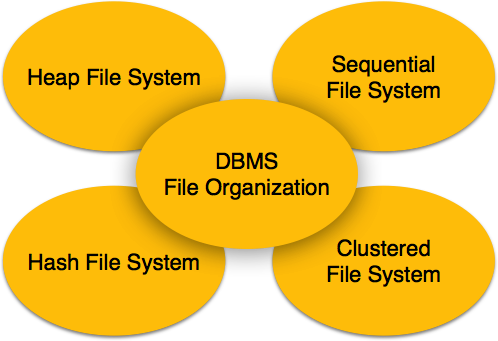
**File System Organization:**

A database consists of a huge amount of data. The data is grouped within a table in RDBMS, and each table has related records. A user can see that the data is stored in form of tables, but in actual this huge amount of data is stored in physical memory in form of files.

**File –** A file is named collection of related information that is recorded on secondary storage such as magnetic disks, magnetic tables and optical disks.

**What is File Organization?**  
File Organization refers to the logical relationships among various records that constitute the file, particularly with respect to the means of identification and access to any specific record. In simple terms, Storing the files in certain order is called file Organization. **File Structure** refers to the format of the label and data blocks and of any logical control record.

### Types of File Organizations –

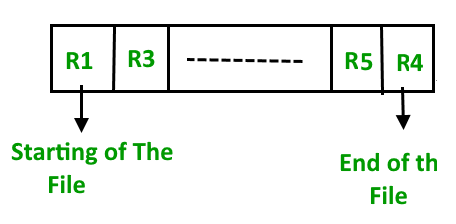
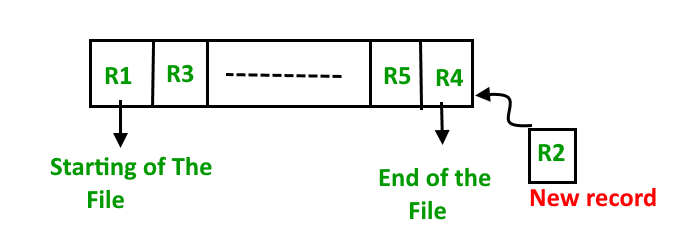
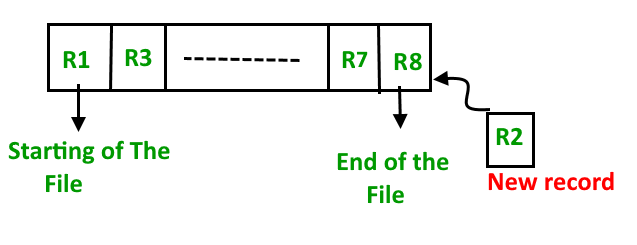


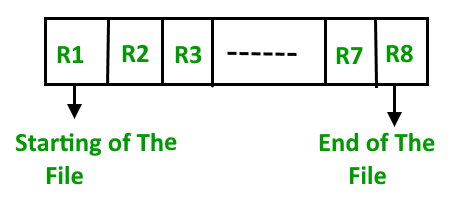
Various methods have been introduced to Organize files. These particular methods have advantages and disadvantages on the basis of access or selection. Thus it is all upon the programmer to decide the best suited file Organization method according to his requirements.  
Some types of File Organizations are:

* Sequential File Organization
* Heap File Organization
* Hash File Organization
* Clustered File Organization

### Sequential File Organization –

The easiest method for file Organization is Sequential method. In this method the file are stored one after another in a sequential manner. There are two ways to implement this method:

1. **Pile File Method –** This method is quite simple, in which we store the records in a sequence i.e one after other in the order in which they are inserted into the tables.  
     
   **Insertion of new record –**  
   Let the R1, R3 and so on upto R5 and R4 be four records in the sequence. Here, records are nothing but a row in any table. Suppose a new record R2 has to be inserted in the sequence, then it is simply placed at the end of the file.  
   
2. **Sorted File Method –**In this method, As the name itself suggest whenever a new record has to be inserted, it is always inserted in a sorted (ascending or descending) manner. Sorting of records may be based on any primary key or any other key.  
   

**Insertion of new record –**  
Let us assume that there is a preexisting sorted sequence of four records R1, R3, and so on upto R7 and R8. Suppose a new record R2 has to be inserted in the sequence, then it will be inserted at the end of the file and then it will sort the sequence.  


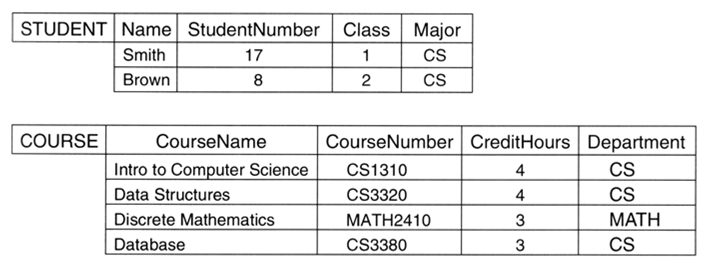
**Pros and Cons of Sequential File Organization –**  
**Pros –**

* Fast and efficient method for huge amount of data.
* Simple design.
* Files can be easily stored in magnetic tapes i.e cheaper storage mechanism.

**Cons –**

* Time wastage as we cannot jump on a particular record that is required, but we have to move in a sequential manner which takes our time.
* Sorted file method is inefficient as it takes time and space for sorting records.

Database: Collection of related data. By data, we mean known facts that can be recorded and that have implicit meaning.



Database Management System is basically software that manages the collection of related data. It is used for storing data and retrieving the data effectively when it is needed. It also provides proper security measures for protecting the data from unauthorized access. In Database Management System the data can be fetched by SQL queries and relational algebra. It also provides mechanisms for data recovery and data backup.

Examples: Oracle, MySQL, MS SQL server.

**Purpose of Database Systems**

1. To see why database management systems are necessary, let's look at a typical ``file-processing system'' supported by a conventional operating system.

The application is a savings bank:

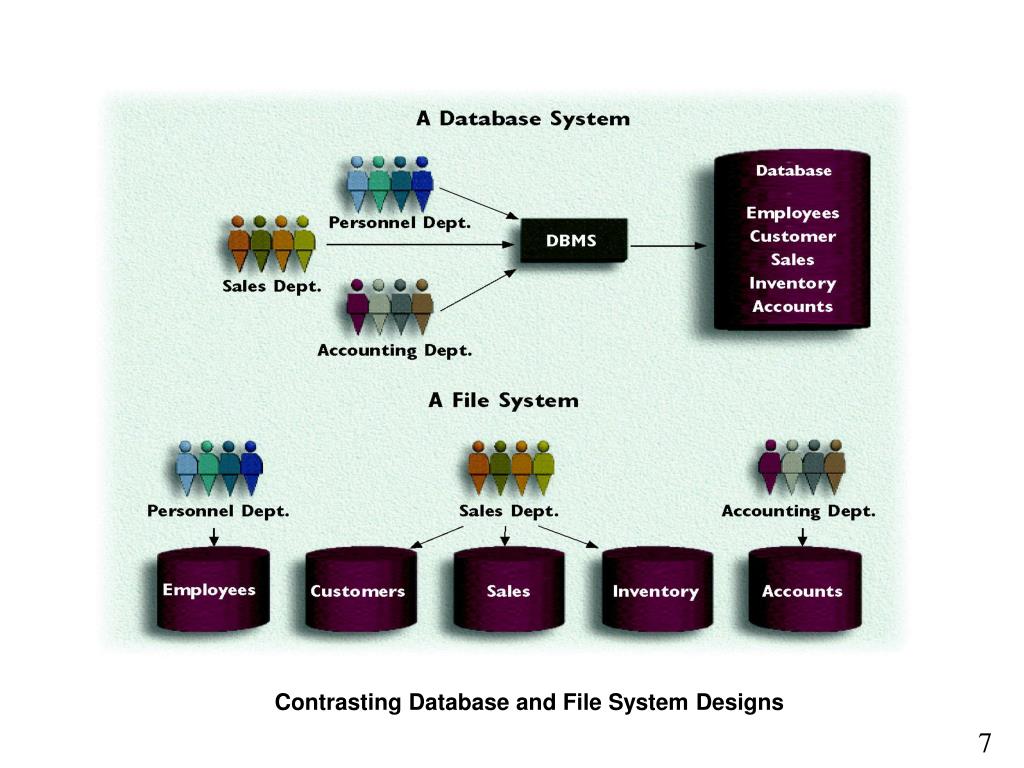
* + Savings account and customer records are kept in permanent system files.
  + Application programs are written to manipulate files to perform the following tasks:
    - Debit or credit an account.
    - Add a new account.
    - Find an account balance.
    - Generate monthly statements.

1. Development of the system proceeds as follows:
   * New application programs must be written as the need arises.
   * New permanent files are created as required.
   * **but** over a long period of time files may be in different formats, and
   * Application programs may be in different languages.
2. So we can see there are problems with the straight file-processing approach:
   * Data redundancy and inconsistency
     + Same information may be duplicated in several places.
     + All copies may not be updated properly.
   * Difficulty in accessing data
     + May have to write a new application program to satisfy an unusual request.
     + E.g. find all customers with the same postal code.
     + Could generate this data manually, but a long job...
   * Data isolation
     + Data in different files.
     + Data in different formats.
     + Difficult to write new application programs.
   * Multiple users
     + Want concurrency for faster response time.
     + Need protection for concurrent updates.
     + E.g. two customers withdrawing funds from the same account at the same time - account has $500 in it, and they withdraw $100 and $50. The result could be $350, $400 or $450 if no protection.
   * Security problems
     + Every user of the system should be able to access only the data they are permitted to see.
     + E.g. payroll people only handle employee records, and cannot see customer accounts; tellers only access account data and cannot see payroll data.
     + Difficult to enforce this with application programs.
   * Integrity problems
     + Data may be required to satisfy constraints.
     + E.g. no account balance below $25.00.
     + Again, difficult to enforce or to change constraints with the file-processing approach.

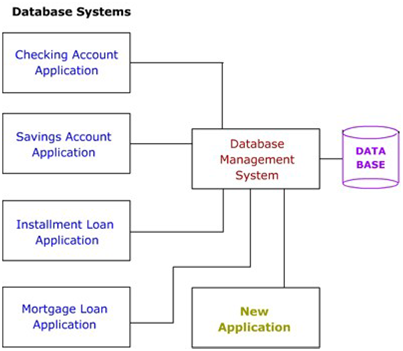
These problems and others led to the development of **database management systems**.

**Difference between File System and DBMS:**

| **S.NO.** | **File System** | **DBMS** |
| --- | --- | --- |
| 1. | File system is software that manages and organizes the files in a storage medium within a computer. | DBMS is a software for managing the database. |
| 2. | Redundant data can be present in a file system. | In DBMS there is no redundant data. |
| 3. | It doesn’t provide backup and recovery of data if it is lost. | It provides backup and recovery of data even if it is lost. |
| 4. | There is no efficient query processing in file system. | Efficient query processing is there in DBMS. |
| 5. | There is less data consistency in file system. | There is more data consistency because of the process of normalization. |
| 6. | It is less complex as compared to DBMS. | It has more complexity in handling as compared to file system. |
| 7. | File systems provide less security in comparison to DBMS. | DBMS has more security mechanisms as compared to file system. |
| 8. | It is less expensive than DBMS. | It has a comparatively higher cost than a file system. |



**Advantages of Database**



As shown in the figure, 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

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

* 1. 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
  2. Data Consistency:- Reduced data redundancy leads to better data consistency
  3. 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,
  4. 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
  5. 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.
  6. 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
  7. Better Controls:-Better controls can be achieved due to the centralized nature of the system
  8. 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

* 1. Reduced Maintenance:-Maintenance is less and easy, again, due to the centralized nature of the system.

***Database Users and Administrator***

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 using predefined user interfaces, such as web or mobile applications. 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 view read reports generated from the database.

As an example, consider a student, who during class registration period, wishes to register for a class by using a web interface. Such a user connects to a web application program that runs at a web server. The application first verifies the identity of the user and then allows her to access a form where she enters the desired information. The form information is sent back to the web application at the server, which then determines if there is room in the class (by retrieving information from the database) and if so adds the student information to the class roster in the database.

• Application programmers are computer professionals who write application programs. Application programmers can choose from many tools to develop user interfaces.

• Sophisticated users interact with the system without writing programs. Instead, they form their requests either using a database query language or by using tools such as data analysis software. Analysts who submit queries to explore data in the database fall in this category.

• Specialized Users

**Database Administrator**

One of the main reasons for using DBMSs is to have central control of both the data and the programs that access those data. A person who has such central control over the system is called a database administrator (DBA). The functions of a DBA include:

• Schema definition. The DBA creates the original database schema by executing a set of data definition statements in the DDL.

• Storage structure and access-method definition. The DBA may specify some parameters pertaining to the physical organization of the data and the indices to be created.

• Schema and physical-organization modification. The DBA carries out changes to the schema and physical organization to reflect the changing needs of the organization, or to alter the physical organization to improve performance.

• Granting of authorization for data access. By granting different types of authorization, the database administrator can regulate which parts of the database various users can access. The authorization information is kept in a special system structure that the database system consults whenever a user tries to access the data in the system.

• Routine maintenance. Examples of the database administrator’s routine maintenance activities are:

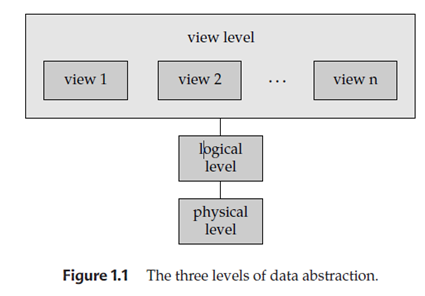
° Periodically backing up the database onto remote servers, to prevent loss of data in case of disasters such as flooding.

° Ensuring that enough free disk space is available for normal operations, and upgrading disk space as required.

° Monitoring jobs running on the database and ensuring that performance is not degraded by very expensive tasks submitted by some users.

***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-systems users are not computer trained, developers hide the complexity from users through several levels of abstraction, to simplify users interactions with the system:



*Physical level*: 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*: 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. Database administrators, who must decide what information to keep in the database, use the logical level of abstraction

*View level*: 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.

***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.

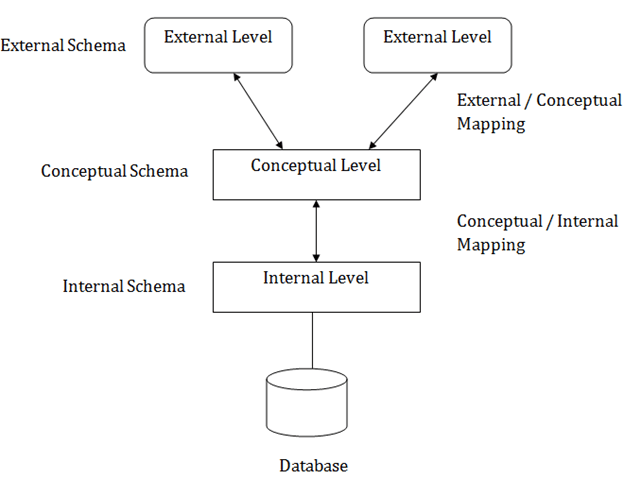
The concept of database schemas and instances can be understood by analogy to a program written in a programming language. A database schema corresponds to the variable declarations (along with associated type definitions) in a program. Each variable has a particular value at a given instant. The values 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 subschema’s, that describe different views of the database. 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.

# Three schema Architecture

* The three schema architecture is also called ANSI/SPARC architecture or three-level architecture.
* This framework is used to describe the structure of a specific database system.
* The three schema architecture is also used to separate the user applications and physical database.
* The three schema architecture contains three-levels. It breaks the database down into three different categories.

**The three-schema architecture is as follows:**



**In the above diagram:**

* It shows the DBMS architecture.
* Mapping is used to transform the request and response between various database levels of architecture.
* Mapping is not good for small DBMS because it takes more time.
* In External / Conceptual mapping, it is necessary to transform the request from external level to conceptual schema.
* In Conceptual / Internal mapping, DBMS transform the request from the conceptual to internal level.

### 1. Internal Level

* The internal level has an internal schema which describes the physical storage structure of the database.
* The internal schema is also known as a physical schema.
* It uses the physical data model. It is used to define that how the data will be stored in a block.
* The physical level is used to describe complex low-level data structures in detail.

### 2. Conceptual Level

* The conceptual schema describes the design of a database at the conceptual level. Conceptual level is also known as logical level.
* The conceptual schema describes the structure of the whole database.
* The conceptual level describes what data are to be stored in the database and also describes what relationship exists among those data.
* In the conceptual level, internal details such as an implementation of the data structure are hidden.
* Programmers and database administrators work at this level.

### 3. External Level

* At the external level, a database contains several schemas that sometimes called as subschema. The subschema is used to describe the different view of the database.
* An external schema is also known as view schema.
* Each view schema describes the database part that a particular user group is interested and hides the remaining database from that user group.
* The view schema describes the end user interaction with database systems.

# *Data Independence*

* Data independence can be explained using the three-schema architecture.
* Data independence refers characteristic of being able to modify the schema at one level of the database system without altering the schema at the next higher level.

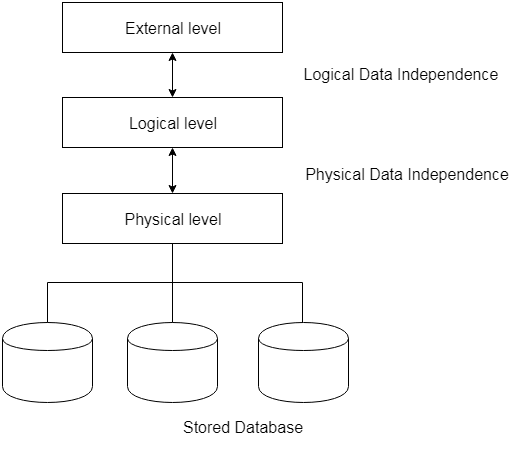
There are two types of data independence:

## 1. Logical Data Independence

* Logical data independence refers characteristic of being able to change the conceptual schema without having to change the external schema.
* Logical data independence is used to separate the external level from the conceptual view.
* If we do any changes in the conceptual view of the data, then the user view of the data would not be affected.
* Logical data independence occurs at the user interface level.

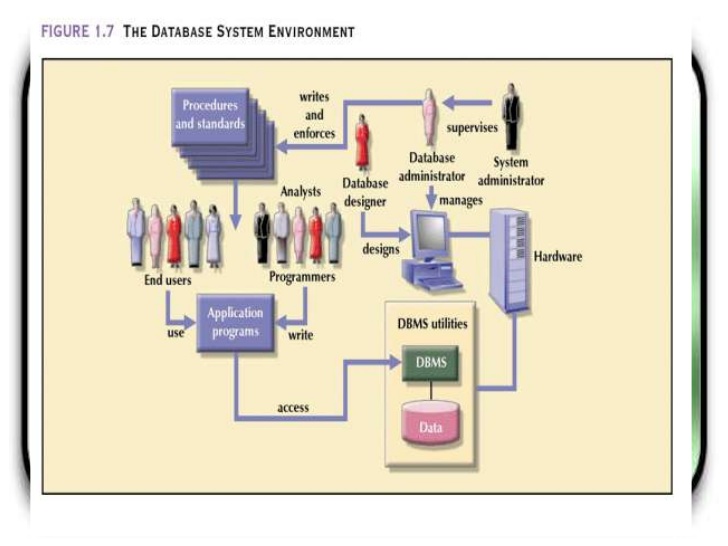
## 2. Physical Data Independence

* Physical data independence can be defined as the capacity to change the internal schema without having to change the conceptual schema.
* If we do any changes in the storage size of the database system server, then the Conceptual structure of the database will not be affected.
* Physical data independence is used to separate conceptual levels from the internal levels.
* Physical data independence occurs at the logical interface level.



**Fig: Data Independence**

***Database System Environment:***



The term database system refers to an organization of components that define and regulate the collection, storage, management, and use of data within a database environment. From a general management point of view, the database system is composed of the five major parts shown in Figure: hardware, software, people, procedures, and data. Let’s take a closer look at the five components shown in Figure:

* Hardware. Hardware refers to all of the system’s physical devices, including computers (PCs, tablets, workstations, servers, and supercomputers), storage devices, printers, network devices (hubs, switches, routers, fiber optics), and other devices (automated teller machines, ID readers, and so on).
* Software. Although the most readily identified software is the DBMS itself, three types of software are needed to make the database system function fully: operating system software, DBMS software, and application programs and utilities.
* Operating system software manages all hardware components and makes it possible for all other software to run on the computers. Examples of operating system software are Microsoft Windows, Linux, Mac OS, UNIX, and MVS.
* DBMS software manages the database within the database system. Some examples of DBMS software are Microsoft’s SQL Server, Oracle Corporation’s Oracle, Oracle’s MySQL, and IBM’s DB2.
* Application programs and utility software are used to access and manipulate data in the DBMS and to manage the computer environment in which data access and manipulation take place. Application programs are most commonly used to access data within the database to generate reports, tabulations, and other information to facilitate decision making. Utilities are the software tools used to help manage the database system’s computer components. For example, all of the major DBMS vendors now provide graphical user interfaces (GUIs) to help create database structures, control database access, and monitor database operations.
* People. This component includes all users of the database system. On the basis of primary job functions, five types of users can be identified in a database system: system administrators, database administrators, database designers, system analysts and programmers, and end users. Each user type, described next, performs both unique and complementary functions.

– System administrators oversee the database system’s general operations.

– Database administrators, also known as DBAs, manage the DBMS and ensure that the database is functioning properly.

– Database designers design the database structure. They are, in effect, the database architects. If the database design is poor, even the best application programmers and the most dedicated DBAs cannot produce a useful database environment. Because organizations strive to optimize their data resources, the database designer’s job description has expanded to cover new dimensions and growing responsibilities.

– System analysts and programmers design and implement the application programs. They design and create the data-entry screens, reports, and procedures through which end users access and manipulate the database’s data.

– End users are the people who use the application programs to run the organization’s daily operations. For example, sales clerks, supervisors, managers, and directors are all classified as end users. High-level end users employ the information obtained from the database to make tactical and strategic business decisions.

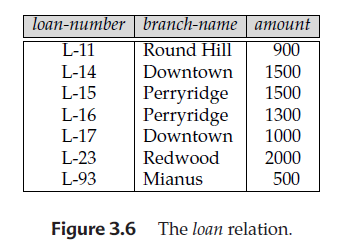
* Procedures. Procedures are the instructions and rules that govern the design and use of the database system. Procedures are a critical, although occasionally forgotten, component of the system. Procedures play an important role in a company because they enforce the standards by which business is conducted within the organization and with customers. Procedures also help to ensure that companies have an organized way to monitor and audit the data that enter the database and the information generated from those data.
* Data. The word data covers the collection of facts stored in the database. Because data is the raw material from which information is generated, determining which data to enter into the database and how to organize that data is a vital part of the database designer’s job.

A database system adds a new dimension to an organization’s management structure. The complexity of this managerial structure depends on the organization’s size, its functions, and its corporate culture. Therefore, database systems can be created and managed at different levels of complexity and with varying adherence to precise standards.

Relational Algebra:

The relational algebra is a procedural query language. It consists of a set of operations that take one or two relations as input and produce a new relation as their result. The fundamental operations in the relational algebra are select, project, union, set difference, Cartesian product, and rename.

Fundamental Operations: The select, project, and rename operations are called unary operations, because they operate on one relation. The other three operations operate on pairs of relations and are therefore, called binary operations.



The Select Operation

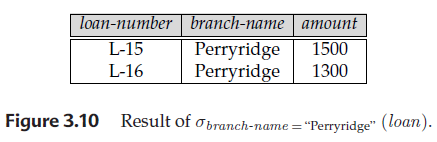
The select operation selects tuples that satisfy a given predicate. We use the lowercase Greek letter sigma (σ) to denote selection. The predicate appears as a subscript to σ. The argument relation is in parentheses after the σ.

**Notation** − σ*p*(r)

Thus, to select those tuples of the loan relation where the branch is”Perryridge”, write

http://www.faadooengineers.com/notes/images/1/63/24f52c4f0d1814bdbd4d1139b245a8ed1.png

If the loan relation is as shown in Figure 3.6, then the relation that results from the preceding query is as shown in Figure 3.10



We can find all tuples in which the amount lent is more than $1200 by writing

http://www.faadooengineers.com/notes/images/1/63/a65899434305fc7ac8fd1f0a7bbddb411.png

In general, we allow comparisons using =, =, &lt;, &le;, &gt;, &ge; in the selection predicate. Furthermore, we can combine several predicates into a larger predicate by using the connectives and (^), or (ⱽ), and not (⌐). Thus, to find those tuples pertaining to loans of more than $1200 made by the Perryridge branch, we write

http://www.faadooengineers.com/notes/images/1/63/59d5a4498d7e82a6e3f1d9b60f1d0c711.png

The selection predicate may include comparisons between two attributes. Consider the relation loan-officer that consists of three attributes: customer-name, banker-name, and loan-number, which specifies that a particular banker is the loan officer for a loan that belongs to some customer. To find all customers who have the same name as their loan officer, we can write

http://www.faadooengineers.com/notes/images/1/63/c9e166d7afd8b4017077be29bf9c6f6f1.png

Project Operation

The project operation allows us to produce this relation. The project operation is a unary operation that returns its argument relation, with certain attributes left out. Since a relation is a set, any duplicate rows are eliminated. Projection is denoted by the uppercase Greek letter pi (Π). We list those attributes that we wish to appear in the result as a subscript to Π. The argument relation follows in parentheses. We write the query to produce such a list as:

Π ID, name, salary(instructor)

Figure 2.11 shows the relation that result from this query.

|  |
| --- |
| ID name salary |
| 10101 Srinivasan 65000 |
| 12121 Wu 90000 |
| 15151 Mozart 40000 |
| 22222 Einstein 95000 |
| 32343 El Said 60000 |
| 33456 Gold 87000 |
| 45565 Katz 75000 |
| 58583 Califieri 62000 |
| 76543 Singh 80000 |
| 76766 Crick 72000 |
| 83821 Brandt 92000 |
| 98345 Kim 80000 |

Figure 2.11 Result of ΠID, name, salary (instructor).

Composition of Relational Operations

The fact that the result of a relational operation is itself a relation is important. Consider the more complicated query “Find the names of all instructors in the Physics department.”

We write: Πname (σdept name =“Physics” (instructor))

Notice that, instead of giving the name of a relation as the argument of the projection operation, we give an expression that evaluates to a relation. In general, since the result of a relational-algebra operation is of the same type (relation) as its inputs, relational-algebra operations can be composed together into a relational-algebra expression.

Composing relational-algebra operations into relational algebra expressions is just like composing arithmetic operations (such as +, −, ∗, and ÷) into arithmetic expressions.

We will use STUDENT\_SPORTS, EMPLOYEE and STUDENT relations as given in Table 1, Table 2 and Table 3 respectively to understand the various operators.

**Table 1 : STUDENT\_SPORTS**

|  |  |
| --- | --- |
| **ROLL\_NO** | **SPORTS** |
| 1 | Badminton |
| 2 | Cricket |
| 2 | Badminton |
| 4 | Badminton |

**Table 2 : EMPLOYEE**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EMP\_NO** | **NAME** | **ADDRESS** | **PHONE** | **AGE** |
| 1 | RAM | DELHI | 9455123451 | 18 |
| 5 | NARESH | HISAR | 9782918192 | 22 |
| 6 | SWETA | RANCHI | 9852617621 | 21 |
| 4 | SURESH | DELHI | 9156768971 | 18 |

**Table 3 : STUDENT**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ROLL\_NO** | **NAME** | **ADDRESS** | **PHONE** | **AGE** |
| 1 | RAM | DELHI | 9455123451 | 18 |
| 2 | RAMESH | GURGAON | 9652431543 | 18 |
| 3 | SUJIT | ROHTAK | 9156253131 | 20 |
| 4 | SURESH | DELHI | 9156768971 | 18 |

Cartesian-Product Operation

The Cartesian-product operation, denoted by a cross (×), allows us to combine information from any two relations. We write the Cartesian product of relations r1 and r2 as r1 × r2.

To apply Cross Product on STUDENT relation given in Table 1 and STUDENT\_SPORTS relation given in Table 2,

**STUDENT X STUDENT\_SPORTS**

**RESULT:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ROLL\_NO** | **NAME** | **ADDRESS** | **PHONE** | **AGE** | **ROLL\_NO** | **SPORTS** |
| 1 | RAM | DELHI | 9455123451 | 18 | 1 | Badminton |
| 1 | RAM | DELHI | 9455123451 | 18 | 2 | Cricket |
| 1 | RAM | DELHI | 9455123451 | 18 | 2 | Badminton |
| 1 | RAM | DELHI | 9455123451 | 18 | 4 | Badminton |
| 2 | RAMESH | GURGAON | 9652431543 | 18 | 1 | Badminton |
| 2 | RAMESH | GURGAON | 9652431543 | 18 | 2 | Cricket |
| 2 | RAMESH | GURGAON | 9652431543 | 18 | 2 | Badminton |
| 2 | RAMESH | GURGAON | 9652431543 | 18 | 4 | Badminton |
| 3 | SUJIT | ROHTAK | 9156253131 | 20 | 1 | Badminton |
| 3 | SUJIT | ROHTAK | 9156253131 | 20 | 2 | Cricket |
| 3 | SUJIT | ROHTAK | 9156253131 | 20 | 2 | Badminton |
| 3 | SUJIT | ROHTAK | 9156253131 | 20 | 4 | Badminton |
| 4 | SURESH | DELHI | 9156768971 | 18 | 1 | Badminton |
| 4 | SURESH | DELHI | 9156768971 | 18 | 2 | Cricket |
| 4 | SURESH | DELHI | 9156768971 | 18 | 2 | Badminton |
| 4 | SURESH | DELHI | 9156768971 | 18 | 4 | Badminton |

A Cartesian product of database relations differs in its definition slightly from the mathematical definition of a Cartesian product of sets. Instead of r1 × r2 producing pairs (t1, t2) of tuples from r1 and r2, the relational algebra concatenates t1 and t2 into a single tuple, as shown in TABLE

***Union (U):***Union on two relations R1 and R2 can only be computed if R1 and R2 are **union compatible** (These two relation should have same number of attributes and corresponding attributes in two relations have same domain) . Union operator when applied on two relations R1 and R2 will give a relation with tuples which are either in R1 or in R2. The tuples which are in both R1 and R2 will appear only once in result relation. Syntax:

**Relation1 U Relation2**

Find person who are either student or employee, we can use Union operator like:

**STUDENT U EMPLOYEE**

**RESULT:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ROLL\_NO** | **NAME** | **ADDRESS** | **PHONE** | **AGE** |
| 1 | RAM | DELHI | 9455123451 | 18 |
| 2 | RAMESH | GURGAON | 9652431543 | 18 |
| 3 | SUJIT | ROHTAK | 9156253131 | 20 |
| 4 | SURESH | DELHI | 9156768971 | 18 |
| 5 | NARESH | HISAR | 9782918192 | 22 |
| 6 | SWETA | RANCHI | 9852617621 | 21 |

***Set Diff(Minus) (-):*** Minus on two relations R1 and R2 can only be computed if R1 and R2 are **union compatible**. Minus operator when applied on two relations as R1-R2 will give a relation with tuples which are in R1 but not in R2. Syntax:

**Relation1 - Relation2**

Find person who are student but not employee, we can use minus operator like:

**STUDENT - EMPLOYEE**

**RESULT:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ROLL\_NO** | **NAME** | **ADDRESS** | **PHONE** | **AGE** |
| 2 | RAMESH | GURGAON | 9652431543 | 18 |
| 3 | SUJIT | ROHTAK | 9156253131 | 20 |

***Rename(ρ):***Rename operator is used to give another name to a relation. Syntax:

**ρ(Relation2, Relation1)**

To rename STUDENT relation to STUDENT1, we can use rename operator like:

**ρ(STUDENT1, STUDENT)**

If you want to create a relation STUDENT\_NAMES with ROLL\_NO and NAME from STUDENT, it can be done using rename operator as:

**ρ(STUDENT\_NAMES, ∏(ROLL\_NO, NAME)(STUDENT))**

***Join Operation***

A Join operation combines related tuples from different relations, if and only if a given join condition is satisfied. It is denoted by ⋈.

### Example:

**EMPLOYEE**

|  |  |
| --- | --- |
| **EMP\_CODE** | **EMP\_NAME** |
| 101 | Stephan |
| 102 | Jack |
| 103 | Harry |

**SALARY**

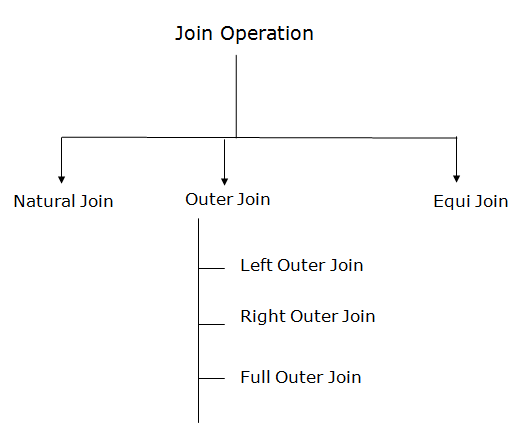
|  |  |
| --- | --- |
| **EMP\_CODE** | **SALARY** |
| 101 | 50000 |
| 102 | 30000 |
| 103 | 25000 |

1. Operation: (EMPLOYEE ⋈ SALARY)

**Result:**

|  |  |  |
| --- | --- | --- |
| **EMP\_CODE** | **EMP\_NAME** | **SALARY** |
| 101 | Stephan | 50000 |
| 102 | Jack | 30000 |
| 103 | Harry | 25000 |

## Types of Join operations:



### 1. Natural Join:

* A natural join is the set of tuples of all combinations in R and S that are equal on their common attribute names.
* It is denoted by ⋈.

**Example:** Let's use the above EMPLOYEE table and SALARY table:

**Input:**

1. ∏EMP\_NAME, SALARY (EMPLOYEE ⋈ SALARY)

**Output:**

|  |  |
| --- | --- |
| **EMP\_NAME** | **SALARY** |
| Stephan | 50000 |
| Jack | 30000 |
| Harry | 25000 |

### 2. Outer Join:

The outer join operation is an extension of the join operation. It is used to deal with missing information.

**Example:**

**EMPLOYEE**

|  |  |  |
| --- | --- | --- |
| **EMP\_NAME** | **STREET** | **CITY** |
| Ram | Civil line | Mumbai |
| Shyam | Park street | Kolkata |
| Ravi | M.G. Street | Delhi |
| Hari | Nehru nagar | Hyderabad |

**FACT\_WORKERS**

|  |  |  |
| --- | --- | --- |
| **EMP\_NAME** | **BRANCH** | **SALARY** |
| Ram | Infosys | 10000 |
| Shyam | Wipro | 20000 |
| Kuber | HCL | 30000 |
| Hari | TCS | 50000 |

**Input:**

1. (EMPLOYEE ⋈ FACT\_WORKERS)

**Output:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EMP\_NAME** | **STREET** | **CITY** | **BRANCH** | **SALARY** |
| Ram | Civil line | Mumbai | Infosys | 10000 |
| Shyam | Park street | Kolkata | Wipro | 20000 |
| Hari | Nehru nagar | Hyderabad | TCS | 50000 |

An outer join is basically of three types:

1. Left outer join
2. Right outer join
3. Full outer join

### a. Left outer join:

* Left outer join contains the set of tuples of all combinations in R and S that are equal on their common attribute names.
* In the left outer join, tuples in R have no matching tuples in S.
* It is denoted by ⟕.

**Example:** Using the above EMPLOYEE table and FACT\_WORKERS table

**Input:**

1. EMPLOYEE ⟕ FACT\_WORKERS

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EMP\_NAME** | **STREET** | **CITY** | **BRANCH** | **SALARY** |
| Ram | Civil line | Mumbai | Infosys | 10000 |
| Shyam | Park street | Kolkata | Wipro | 20000 |
| Hari | Nehru street | Hyderabad | TCS | 50000 |
| Ravi | M.G. Street | Delhi | NULL | NULL |

### b. Right outer join:

* Right outer join contains the set of tuples of all combinations in R and S that are equal on their common attribute names.
* In right outer join, tuples in S have no matching tuples in R.
* It is denoted by ⟖.

**Example:** Using the above EMPLOYEE table and FACT\_WORKERS Relation

**Input:**

1. EMPLOYEE ⟖ FACT\_WORKERS

**Output:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EMP\_NAME** | **BRANCH** | **SALARY** | **STREET** | **CITY** |
| Ram | Infosys | 10000 | Civil line | Mumbai |
| Shyam | Wipro | 20000 | Park street | Kolkata |
| Hari | TCS | 50000 | Nehru street | Hyderabad |
| Kuber | HCL | 30000 | NULL | NULL |

### c. Full outer join:

* Full outer join is like a left or right join except that it contains all rows from both tables.
* In full outer join, tuples in R that have no matching tuples in S and tuples in S that have no matching tuples in R in their common attribute name.
* It is denoted by ⟗.

**Example:** Using the above EMPLOYEE table and FACT\_WORKERS table

**Input:**

1. EMPLOYEE ⟗ FACT\_WORKERS

**Output:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EMP\_NAME** | **STREET** | **CITY** | **BRANCH** | **SALARY** |
| Ram | Civil line | Mumbai | Infosys | 10000 |
| Shyam | Park street | Kolkata | Wipro | 20000 |
| Hari | Nehru street | Hyderabad | TCS | 50000 |
| Ravi | M.G. Street | Delhi | NULL | NULL |
| Kuber | NULL | NULL | HCL | 30000 |

### 3. Equi join:

It is also known as an inner join. It is the most common join. It is based on matched data as per the equality condition. The equi join uses the comparison operator(=).

**Example:**

**CUSTOMER RELATION**

|  |  |
| --- | --- |
| **CLASS\_ID** | **NAME** |
| 1 | John |
| 2 | Harry |
| 3 | Jackson |

**PRODUCT**

|  |  |
| --- | --- |
| **PRODUCT\_ID** | **CITY** |
| 1 | Delhi |
| 2 | Mumbai |
| 3 | Noida |

**Input:**

1. CUSTOMER ⋈ PRODUCT

**Output:**

|  |  |  |  |
| --- | --- | --- | --- |
| CLASS\_ID | NAME | PRODUCT\_ID | CITY |
| 1 | John | 1 | Delhi |
| 2 | Harry | 2 | Mumbai |
| 3 | Harry | 3 | Noida |