Discipline:	BS (IT) - 4th Semester
Subject:	Database Systems [**NEW Course - Updated**]
Course Code:	IT – 411
Mid Term Notes:	From Week No. 01 – 06
Prepared by:	**ARSHAD IQBAL**, Lecturer (CS/IT), ICS/IT - FMCS, The University of Agriculture, Peshawar

Course Objectives

Welcome to the course of Database Systems

At the end of the course the students will be able to:

1. **Explain** fundamental database concepts.
2. **Design** conceptual, logical and physical database schemas using different data models.
3. **Know** about Relational Data Model (RDM)
4. **Know** Entity Relation Model (ERM)
5. **Identify** functional dependencies and resolve database anomalies by normalizing database tables.
6. **Know** about Data integrity, security, concurrency and recovery and recovery techniques.
7. **Use** Structured Query Language (SQL) for database definition and manipulation in any DBMS

Week No. 01 & 02: Introduction to Database

Data:
Collection of facts and figures concerning an object or event or Stored representation of objects and events that have meaning and importance in the user's environment. It is always in **raw** form **OR** Data is a value or set of values. **OR** Data are facts that concern with people, person, place, event, objects etc.

Data Item:
A single unit of value in data is called **data item** e.g., Name, Address, Roll No. etc.

Group Data Item:
The data items divided into sub items are called **Group data items** e.g., Name is divided into First name, Middle name, Last name so name is a **Group Data Item**.

Elementary Data Item:
The data items that are not divided into sub items are called **Elementary Data Items** e.g., Roll Number which is not sub divided so Roll Number is an elementary item.

Meta Data:
Data about data. Data that describe the properties or characteristics of data, and the context (background) of that data. Some examples of Meta Data are Data Type, Length/Size, Maximum and Minimum values etc.

Table: Example of Meta Data

Field Name	**Data Type**	**Length**	**Description**	**Constraint**
Roll No.	Integer	3	Roll No. of the student	Value from 1 to 100
Name	Alphabetic	50	Name of the student	
Address	Alphanumeric	100	Address of the student	
Email	Alphanumeric	25	Email of the student	Must contain @ and .
Phone No.	Numeric	25	Phone No. of the student	

Types of Data:

1. **Textual/Alphabetic Data:** Data in the form of written phrases and sentences.
2. **Numeric Data:** Numeric data consists of numeric digits from 0 to 9 like 10, 245 or -5 etc. The numeric type of data may either be positive or negative.
3. **Alphanumeric Data:** Alphanumeric data consists of numeric digits (0 to 9), letters (A to Z) and all special characters like +, %, and @ etc. like "87%", "$300" and "H#17".
4. **Audio Data:** Data in the form of spoken phrases and sentences.
5. **Graphical/Image Data:** Pictorial (Graphic) data i.e., data in the form of images, charts, graphs, pictures etc.
6. **Video Data:** Data in the form of Digital **video** i.e., an electronic representation of moving visual images (**video**) in the form of encoded digital data.

Information:
Meaningful form of data. Processed form of data that can be used for decision making. Data that have been processed in such a way as to increase the knowledge of the person who uses the data.

$$\begin{tikzcd} \boxed{\text{Data}} \arrow[r] & \boxed{\text{Processing}} \arrow[r] & \boxed{\text{Information}} \end{tikzcd}$$

Field:
A group of related characters to represent some unit of information is called a **field** e.g., person name is a **field**. **Field** means a **column**. There are three types of **fields**:

i.	**Numeric Field:**	Weight is 50 Kg.
ii.	**Alphabetic Field:**	Name is Ali, Asad etc.
iii.	**Alpha Numeric Field:**	Address is H. No. 77, Peshawar.

Key Field:
A **key field** is used to identify the record for location & processing purposes. For **example,** in a sale ledger file the **key field** might be the customer code, in a payroll file the **key field** might be employee number, in student file the **key field** might be roll no. etc.
Record:
Fields are grouped together to provide information about a single entity (object/unit/thing/person) is called **record. Record** means a **row** e.g., student record.

Roll No.	Name	DOB	Marks
35	Fahad	10th Aug. 1986	485

File:
A **file** is a named collection of records means records are grouped together to provide a complete information about all entities. **File** means a **table** e.g., the **file name** may be **student** and it will contain **four records** e.g.

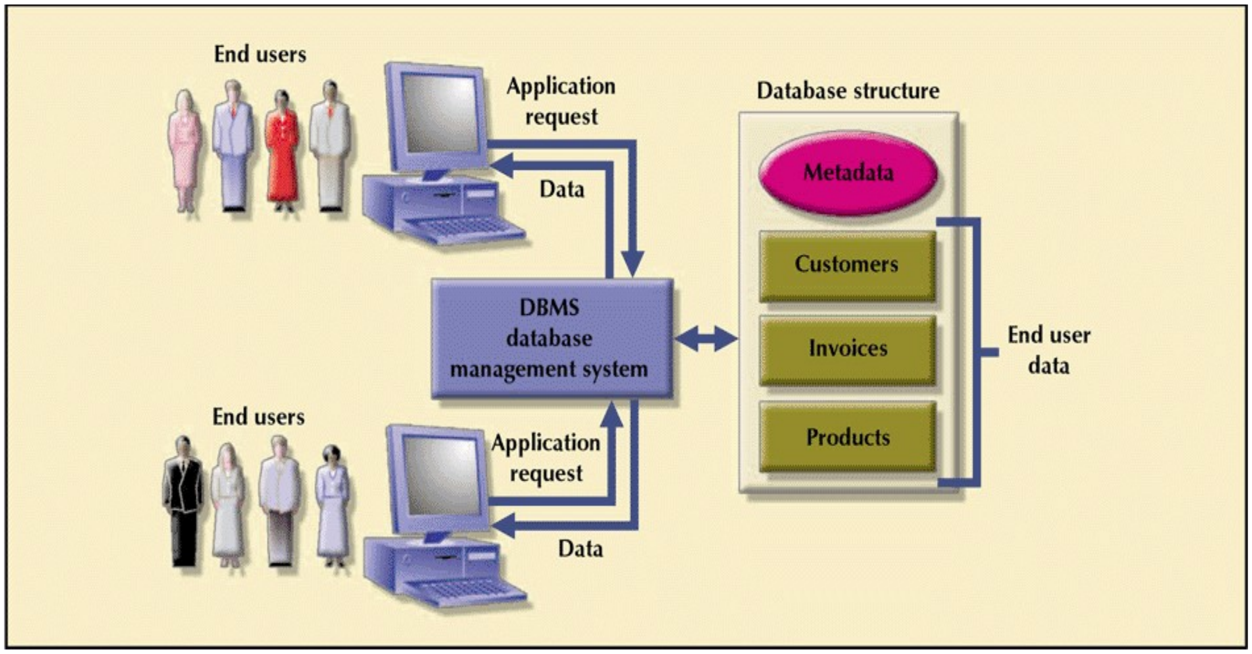
Roll No.	Name	DOB	Marks
35	Fahad	10th Aug. 1986	485
14	Asad	21st March 1984	415
30	Iqbal	22nd April 1982	420
40	Wasim	23rd June 1985	430

Database:
An organized collection of information in computerized format. An organized collection of logically related data. A database is a shared collection of logically related data, designed to meet the information needs of multiple users in an organization. To manage **database**, DataBase Management System (DBMS) will be needed.

Data Base Management System (DBMS):
DBMS is a **program** that stores, retrieves, and modifies data in **database** on request. **DBMS** is the **software** or **tool** that is used to manage the **database** and its **users**. **DBMS** is a software system that is used to create and maintain database, and provide controlled access to users. **DBMS** is a software system that acts as an interface between the Database and user of the Database. **Database management system** is software of collection of small programs to perform certain operation on data and manage the data.
Two basic operations performed by the DBMS are:
- Management of Data in the Database.
- Management of Users associated with the database.

Management of the data means to specify that how data will be stored, structured and accessed in the database.
Management of database users means to manage the users in such a way that they can perform any desired operations on the database. **DBMS** also ensures that a user cannot perform any operation for which he is not allowed. And also, an authorized user is not allowed to perform any action which is restricted to that user.

In **General DBMS** is a collection of Programs performing all necessary actions associated to a database.



Entity:
Entity is basic building block of the **Entity Relation Data Model**. The term **Entity** is used in three different meanings or for three different terms and that are:

- Entity Type
- Entity Instance
- Entity Set

Entity Type:
A collection or a set of entities (Entity Instances) that shares common properties. The **Entity Type** can be defined as a name/label assigned to items/objects that exist in an environment and that have similar properties. It could be a person, place, object or event in the user environment about which the organization wishes to maintain data. **Examples** are: departments, employees, students, and orders etc. To represent an **Entity Type** in a **Model**, use the following conventions:

- Soft box with any dimensions.
- Singular, Unique entity name.
- Entity name in Uppercase.

Entity Instance:
A single occurrence (existence) of an **Entity Type**. A particular object belonging to a particular **Entity Type** and an item becomes an **Instance** of or belongs to an **Entity Type** by possessing (having) the defining properties associated with an **Entity Type**. For **example**, following **table** lists the **Entity Types** and their defining Properties and Instances.

Table No. 1.1: Entity Types, their Properties and Instances

Entity Types	Properties	Instances
EMPLOYEE	Human being, has name, has father name, has a registration number, has qualification, designation	M. Sharif, Sh. Akmal and many others.
FURNITURE	Used to sit or work on, different material, having legs, cost, purchased	Chair, Table etc.
ELECTRIC APPLIANCES	Need electricity to work, purchased	Bulb, Fan, AC etc.
OFFICE EQUIPMENT	Used for office work, consumable or non-consumable	Papers, pencil, paper weight etc.

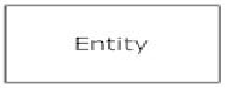
Each **Entity Instance** possesses (have) certain values against the properties with the **Entity Type** to which it belongs. For **example**, the above table identified that **Entity Type** EMPLOYEE has name, father name, registration number, qualification, designation. Now an **Instance** of this **Entity Type** will have values against each of these properties, like (M. Sajjad, Abdul Rehman, and Programmer) and may be one **Instance** of **Entity Type** EMPLOYEE. There could be many others.

Entity Set:
A **group** of **Entity Instances** of a particular **Entity Type** is called an **Entity Set**. For **example**, all **employees** of an organization form an **Entity Set**. Like all students, all courses, all of them form **Entity Set** of different **Entity Types**.

Entity Types:
There are two types of Entity Types: Strong/Regular Entity Type and Weak Entity Type.

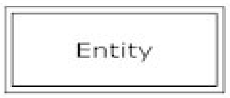
Strong/Regular Entity Type:
An **Entity Type** that exists independently of other **Entity Types**. The **Strong Entity Type** is called the **Identifying Owner**. An **Entity Type** whose **Instances** can exist **independently**, that is, without being linked to the **Instances** of any other **Entity Type** is called **Strong Entity Type.** A **major property** of the **Strong Entity Type** is that they have their **own** identification, which is not always the case with **Weak Entity Type**. For **example**, employee in an organization, is an independent or **Strong Entity Type**, since its **Instances** can exist independently.

Symbol for **Strong Entity Type**.



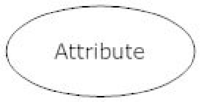
Weak Entity Type:
An **Entity Type** whose **existence** depends on another **Entity Type**. An **Entity Type** whose **Instances** cannot exist without being linked with **Instances** of some other **Entity Type**, i.e., they cannot exist independently. For **example**, in an organization, to maintain data about the **Vehicles** owned by the **employees**. Now a particular **Vehicle** can exist in this organization only if the **owner** already exists there as **employee**. **Similarly**, if an **employee** leaves the job and the organization decides to delete the record of the **employee** then the record of the **Vehicle** will also be deleted since it cannot exist without being linked to an **Instance** of **employee**.

Symbol for **Weak Entity Type**



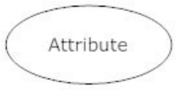
Attributes:
A property or characteristic of an **Entity Type** that is of interest to the organization. Something that describes an **Entity Type**. For **example**, the employee **Entity Type**, the **attributes** would be the employee number, name, job title, hire date, department number and so on.

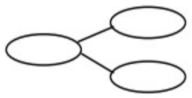
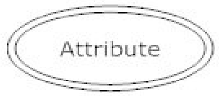
To represent an **attribute** in a **Model**, use the following conventions:

- Use Singular Name in Lowercase
- Symbol for Attribute 

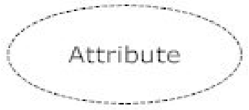
Types of Attributes:

1. Required Attribute
2. Optional Attribute
3. Simple Attribute
4. Composite Attribute
5. Single Valued Attribute
6. Multi Valued Attribute
7. Stored Attribute
8. Derived Attribute

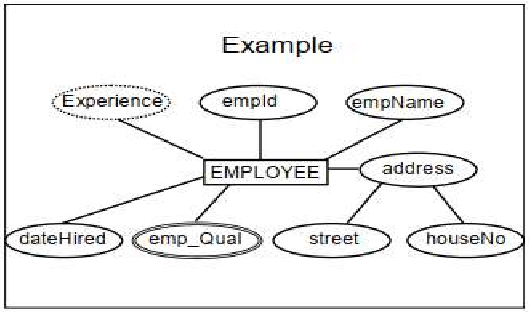
1. **Required Attribute:**
   - A Required Attribute is an Attribute that must have a data value e.g., Student Registration Number, CNIC, Name etc.
   - These Attributes are required because they describe what is important in the entity.
2. **Optional Attribute:**
   - An Optional Attribute may or may not have a data value in it and can be left blank e.g., Student Phone Number, email address, Passport Number etc.
3. **Simple Attribute:**
   - An Attribute that cannot be decomposed into smaller Attributes e.g., age, roll#, CNIC, Phone# etc.
   - Symbol for Simple Attribute 

4. **Composite Attribute:**
   - An Attribute that can be decomposed into smaller Attributes e.g. Name, address etc.
   - Name can be divided into first name, middle name, last name.
   - Address can be divided into House#, Street#, Sector#, Phase#, Tehsil, District etc.
   - Symbol for Composite Attribute. 
5. **Single Valued Attribute:**
   - An Attribute that has only one value for an Entity e.g., Roll# CNIC, Passport#, name, father name etc.
6. **Multi Valued Attribute:**
   - An Attribute that can have more than one values for an Entity e.g., Address, Email Address, Phone#, Subject, Hobbies, Skills etc.
   - Symbol for Multi valued Attribute. 

7. **Stored Attribute:**
   - Normally Attributes are stored Attributes, that is, their values are stored and accessed as such from the database.
   - An Attribute whose value is stored in the database and cannot be derived from another Attribute e.g., Date of Birth (D.O.B).

8. **Derived Attribute:**
   - Sometimes Attributes values are not stored as such, rather they are computed or derived based on some other value which is stored in database.
   - An Attribute whose value can be derived/calculated from the Stored Attribute e.g., age which is derived from D.O.B, experience etc.
   - A Derived Attribute is based on another Attribute.
   - Symbol for Derived Attribute 

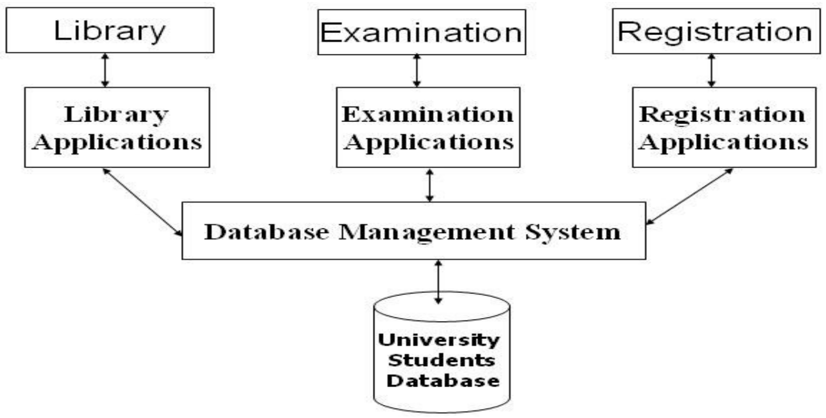
For Example, an Entity Type with Attributes of different types:



Relation:
A named two-dimensional table of data. Each relation consists of a set of named columns and an arbitrary (random) number of rows.

Relational Database:
A Relational Database uses relations or two-dimensional tables to store information. For **example**, to store information about all the employees in a company. In a relational database, create several tables to store different pieces of information about an employee, such as an employee table, a department table, and a salary table. A Relational Database is a database that represents data as a collection of tables in which the relationships are represented by common fields in the related tables.

Example of a Database System:


Library
Examination
Registration
Library Applications
Examination Applications
Registration Applications
Database Management System
University Students Database

Manual System:

- The principle of **Manual Database** is the same as **Electronic Database**.
- It's a file system that consists of many records.
- The main difference is that it is not computerized, so all the records and database itself are tangible.
- Practically **Manual Databases** are file boxes or paper records and folders.
- **Manual Databases** are still used in some smaller libraries and also in places where client register is needed for example hospitals.
- **Manual** or **Paper-Based Databases** are the simplest and most common type of database.
    - An Address Books
    - Birthday Book
    - Diary
    - Telephone Directory
- A **Manual Database** is a hard-file storage system that consists of paper records, folders and filing cabinets or storage boxes. A quality **Manual Database system** makes it easy to retrieve documents and information when they are needed.
- The records in **Manual Database** can be ordered in many ways. Most popular ways of organization are alphabetically, chronologically and numerically.

Merits of Manual Database:

- **Simplest:** It is the simplest type of database.
- **Cheaper:** It is cheaper.
- **Consume Less Electricity:** It doesn't consume a lot of electricity.
- **Not easily deleted:** It can't be deleted easily.

Demerits of Manual Database:

- **Harder:** It is harder.
- **Consume a lot of time:** It consumes a lot of time.
- **Human Error:** There can be a chance of Human Error.
- **Speed:** Need Speed for data entry.

Week No. 03: Traditional File Processing Approach

Traditional File Processing System:
Traditional File Processing System or simple file processing system refers to the first computer-based approach of handling the commercial or business applications. That is why it is also called a replacement of the Manual File System. Before the use of computers, the data in the offices or business was maintained in the files. Obviously, it was laborious (difficult), time consuming, inefficient, especially in case of large organizations. Computers, initially designed for the engineering purposes but file processing environment simply transformed Manual file work to computers. So, processing became very fast and efficient.

Traditional File Processing System was totally computer-based system where all the information is store in different computer files. Files in Traditional File Processing Systems are called **Flat Files**. Also, Traditional Files System stores data in a manner that all the departments of an organization have their own set of files that creates data redundancy. It stores data of an organization in group of files. Files carrying data are independent on each other. **COBOL, C, C++** programming languages were used to design the files. Each file contains data for some specific area or department like library, student fees, and student examinations.

Typical file processing environment:

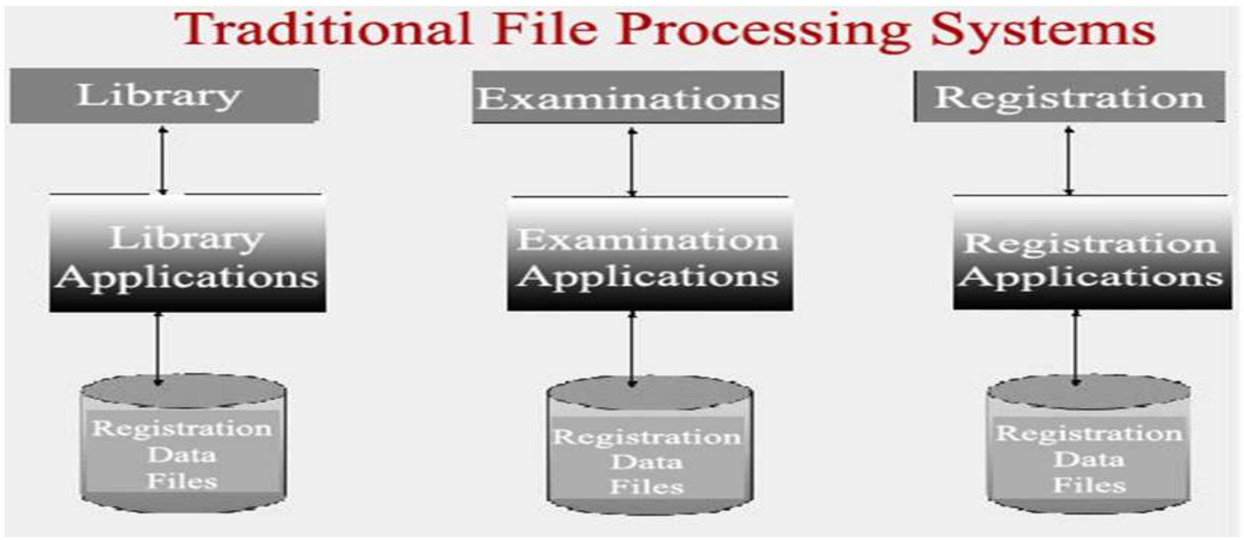


Figure No. 3.1: A typical File Processing Environment

The diagram presents a typical Traditional File Processing Environment. The main point being highlighted is the **program and data interdependence**, that is, **program** and **data** depend on each other, well they depend too much on each other. As a result, any change in one affects the other as well. This is something that makes a change very painful or problematic for the designers or developers of the system.

For example, previously the billing was performed in an organization on the monthly basis, now company has decided to bill the customers after every ten days. Since the bills are being generated from the computer (using File Processing System), this change has to be incorporated in the system. Yet another example is that, initially bills did not contain the address of the customer, now the company wants the address to be placed on the bill, so here is change. There could be many more examples. So, changes need in the system, but due to **program-data interdependence** these changes in the systems were very hard to make. A change in one will affect the other whether related or not.

For example, suppose data about the customer bills is stored in the file, and different programs use this file for different purposes, like adding data into the bills file, to compute the bill and to print the bill. Now the company asks to add the customers' address in the bills, for this, have to change the structure of the bill file and also the program that prints the bill.

The painful thing is that the other programs that are using these bills files but are not concerned with the printing of the bills or the change in the bill will also have to be changed and this is needless and causes extra unnecessary effort.

Another major drawback in the Traditional File System Environment is the non-sharing of data. It means if different systems of an organization are using some common data, then rather than storing it once and sharing it, each system stores data in separate files. This creates the problem of redundancy or wastage of storage.

Advantages of Traditional File Processing System:

1. **Simple:** It is very simple to use.
2. **No need of external storage:** There is no need of external storage.
3. **No need of a highly technical person:** There is no need of a highly technical person to handle the files.
4. **Processing Speed:** Processing Speed is high as compared to DBMS.

Disadvantages of Traditional File Processing System:

1. **Maintenance:** It is very difficult to maintain File Processing System.
2. **Less Flexible:** It is less flexible and has many limitations.
3. **Updating Problem:** High complexity in updating of File Processing System.
4. **Data and Program Interdependence:** Any change in one file affects all the files that creates burden on the programmer.
5. **Data Isolation:** Means that all the related data is not available in one file.
6. **Duplication of Data:** Due to the decentralized approach, Programs are often developed independently in the file processing system, and these application programs has then its own set of data. So, the same data is duplicated.
7. **Limited Data Sharing:** With the Traditional File Approach, each application program has its own private data file and users have little opportunity to share data outside their own applications.
8. **Excessive Program Maintenance:** Most of the time, programmer is busy in maintaining the existing programs, leaving little opportunity for developing new application.
9. **Less Security:** Provides less security.
10. **Redundancy:** Redundancy is more.
11. **Less Integrity:** There is less Integrity.

Database Approach:
A database is a shared collection of logically related data, and designed to meet the requirements of different users of an organization.

Advantages of Database Approach:

1. **Program Data Independence:**

The separation of data descriptions from the application programs that uses the data is called Data Independence. With the Database Approach, data descriptions are stored in a central location called the Repository. This property of database systems allows an organization's data to change and evolve (develop) without changing the application programs that process the data.

2. **Controlled Data Redundancy:**

The Database Approach does not eliminate redundancy entirely, but it enables the designer to control the duplication data to improve database performance.

3. **Better Data Integrity:**

Very important feature, means the validity of the data being entered in the database. Integrity of data is very important, since all the processing and the information produced in return are based on the data. Now if the data entered is not valid, how can be sure that the processing in the database is correct and the results or the information produced is valid. The businesses make decisions on the basis of information produced from the database and the wrong information leads to wrong decisions, and business collapse. In the database system environment, DBMS provides many features to ensure the data integrity, hence provides more reliable data processing environment.

4. **Improved Data Consistency:**

By eliminating or controlling data redundancy, greatly reduce the opportunities for inconsistency. For example, if a customer's address is stored only once, we cannot disagree about the customer's address. When the customer's address changes, recording the new address is greatly simplified because the address is stored in a single place. Finally, avoid the wasted storage space that results from redundant data storage.

5. **Improved Data Sharing:**

A database is designed as a shared corporate resource. Authorized Internal and External users are granted permission to use the database, and each user (or group of users) is provided one or more user views into the database to facilitate this use. A user view is a logical description of some portion of the database that is required by a user to perform some tasks. A user view is often developed by identifying a form or report that the user needs on a regular basis.

6. **Increased programmer productivity:**

A major advantage of the Database Approach is that it greatly reduces the cost and time for developing new business applications.

7. **Enhanced Enforcement (implementation) of Data Standards:**

When the database approach is implemented with full management support, the database administration function should be granted single point authority and responsibility for establishing and enforcing data standards. These standards will include naming conventions (agreements), data quality standards, and uniform procedures for accessing, updating, and protecting data.

8. **Improved Data Quality:**

Concern with poor quality data is a common theme in strategic planning and database administration today. The database approach provides a number of tools and processes to improve data quality. Database designers can specify integrity constraints that are enforced by the DBMS. A constraint is a rule that cannot be violated by database users.

9. **Improved Data Accessibility:**

With a relational database, end users without programming experience can often retrieve and display data, even when it crosses traditional departmental boundaries. The language used in this query is called Structured Query Language, or SQL. Although the queries constructed can be much more complex, the basic structure of the query is easy for even novice(beginner/trainee/learner) or nonprogrammers to grasp/grip/hold. If they understand the structure and names of the data that fit within their view of the database, they soon gain the ability to retrieve answers to new questions without having to rely on a professional application developer.

10. **Reduced Program Maintenance:**

Stored data must be changed frequently for a variety of reasons: new data item types are added, data formats are changed, and so on. An example of this problem was the well-known "year 2000" problem, in which common two-digit year fields were extended to four digits to accommodate the rollover from the year 1999 to the year 2000. In a database environment, data are more independent of the application programs that use them. Within limits, we can change either the data or the application programs that use the data without necessitating (demanding/requiring) a change in the other factor. As a result, program maintenance can be significantly reduced in a modern database environment.

Week No. 04: Components of DBMS and Database Environment

Components of DBMS: The database management system can be divided into five major components, they are:

1. **Hardware**
2. **Software**
3. **Data**
4. **Procedures**
5. **Database Access Language**
6. **User**

A **simple diagram** to show all components together to form a **Database Management System**.

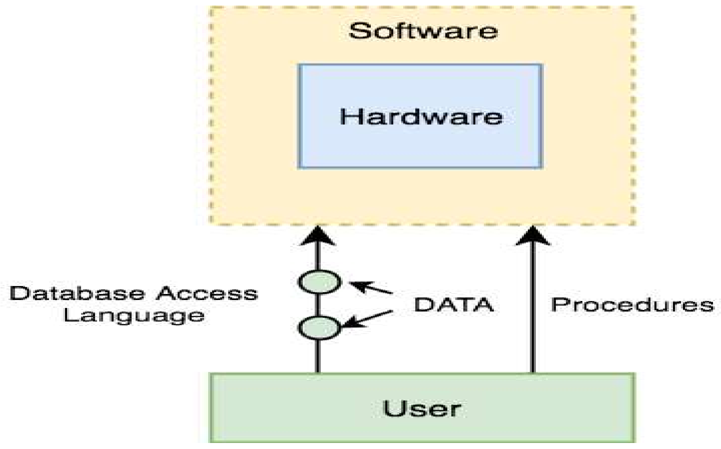


Figure No. 4.1: Components of DBMS

1. Hardware:

Hardware means computer, hard disks, I/O channels for data, and any other physical component involved before any data is successfully stored into the memory. Consist of a set of physical electronic devices such as computers, I/O devices, storage devices etc., provides the interface between computers and the real-world systems.

2. Software:

Software component, as this is the program which controls everything. The **DBMS software** which provides us with an easy-to-use interface to store, access and update data. **Software** is the set of programs used to control and manage the overall database. This includes the **DBMS software** itself, the Operating System, the network software being used to share the data among users, and the application programs used to access data in the **DBMS**.

3. Data:

Data is that resource, for which **DBMS** was designed. The motive (reason) behind the creation of **DBMS** was to store and utilize data.

Metadata is data about the data. This is information stored by the **DBMS** to better understand the data stored in it.

4. Procedures:
Procedures refer to general instructions to use a Database Management System (DBMS). This includes procedures to setup and install a DBMS, to login and logout of DBMS software, to manage databases, to take backups, generating reports etc.

5. Database Access Language:
Database Access Language is a simple language designed to write commands to access, insert, update and delete data, stored in any database. User can create new databases, tables, insert data, fetch stored data, update data and delete the data using the access language.

6. Users:
Database Administrators: Database Administrator or DBA is the one who manages the complete Database Management System (DBMS). DBA takes care of the security of the DBMS, its availability, managing the license keys, managing user accounts and access etc.
Application Programmer or Software Developer: This user group is involved in developing and designing the parts of DBMS.
End User: These days all the modern applications, web or mobile store user data. Applications are programmed in such a way that they collect user data and store the data on DBMS systems running on their server. **End users** are the one who store, retrieve, update and delete data.

Data Dictionary:
This is a reserved space within a database used to store information about the database itself. Data Dictionaries are created during the design phase to define the structure of a database. Data Dictionary is a repository of information that describes the logical structure of the database. It contains name of each entity, name of all attributes, size of each attribute, indication of attributes (Keys), validation rules that are to be applied to attributes, record types, data item types (exact meanings of data items) and data aggregates (collections) etc.

Data Dictionary contains metadata. Data Dictionary stores all information about different authorized users. It can also keep record of each access to the database. A database that containing data about all the databases in the database system. Data Dictionary store all the various file specifications and their locations. They also contain information about which programs use which data and which users are interested in which reports.

A Data Dictionary is a set of read-only table and views, containing the different information about the data used in the enterprise to ensure that database representation of the data follow one standard as defined in the dictionary.

Types of Data Dictionaries:

There are basically two types of data dictionaries which are available for use by a DBMS, with respect to their existence.

1. Integrated Data Dictionary:
The first type of **Data Dictionary** is the **Integrated Data Dictionary**. A **Data Dictionary** that is part of DBMS is called **Integrated Data Dictionary**. Such a **Data Dictionary** is embedded into the database system, and is created by the DBMS for its usage under the directions and requirements provided by the DBA. The DBMS uses the Data Dictionary to access the database at each layer or model, for this purpose the data dictionary of any type can be used but the Integrated Data Dictionary is far more efficient than any Free-Standing Data Dictionary because an Integrated Data Dictionary is created by the DBMS itself and uses the same data accessing techniques.

There are two types of Integrated Data Dictionary:

i. **Active Data Dictionary:** The Integrated Data Dictionary is called active if it is checked by DBMS every time when a database is accessed. It is always consistent (reliable) with actual database structure. It is automatically maintained by the system.
ii. **Passive Data Dictionary:** The Integrated Data Dictionary is called Passive if it is not used in day-to-day database processing.

2. Freestanding Data Dictionary:
Second type of Data Dictionary is Freestanding (unconnected/self-supporting) Data Dictionary and created by any Computer-Aided Software Engineering (CASE) Tool and then attached to the Database Management Systems (DBMS). A number of CASE Tools are available for this purpose and help user to design the database and the database applications. It can be a commercial product or a simple file developed by the designer.

Database Environment:
The database operational environment is an Integrated (combined/joined) System of:
– Hardware,
– Software, and
– People,

These components are designed to facilitate the:
– Data Storage
– Data Retrieval, and
– Control of the information resource and to improve the productivity of the organization.

Components of Database Environment:

The following figure shows typical components of Database Environment:

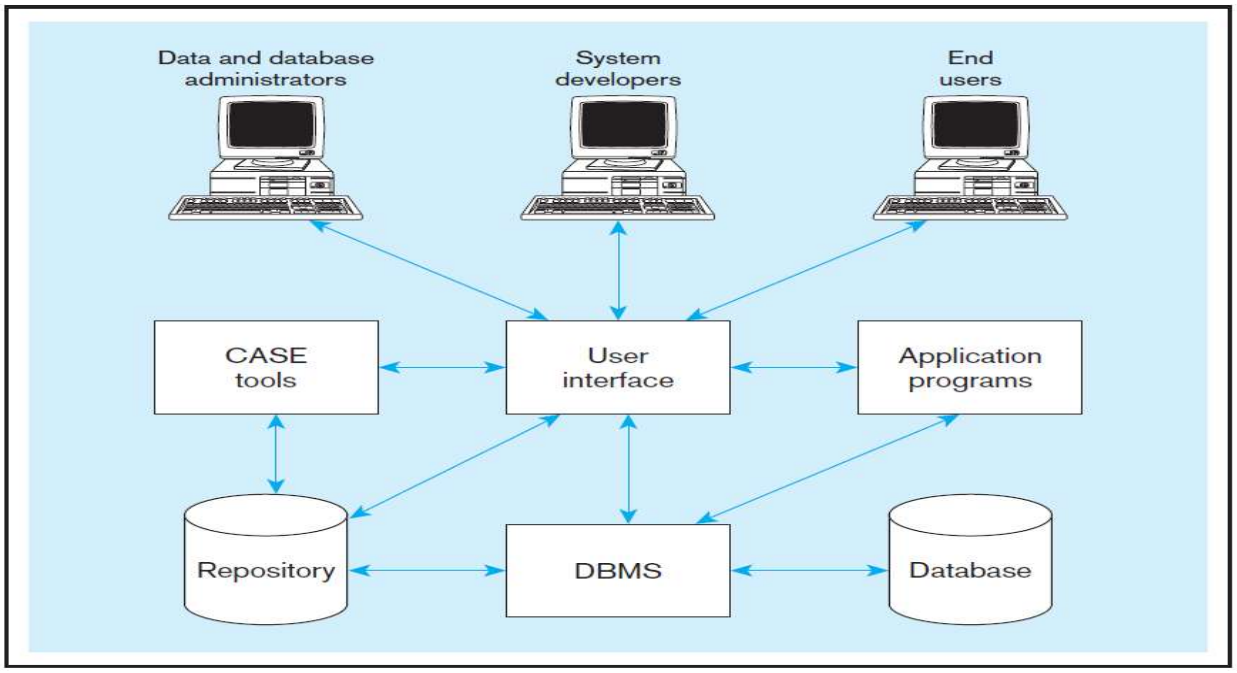


Figure No. 4.2: Components of Database Environment

1. Data Base Management System (DBMS):

DBMS is a collection of programs that are used to create, store, retrieve, and modify data in database on request. **DBMS** is the software or tool that is used to manage the database and its users. **DBMS** is a software system that is used to create, maintain, and provide controlled access to users. **DBMS** is a software system that acts as an interface between the Database and user of the Database. **Database management system** is software of collection of small programs to perform certain operation on data and manage the data.

Two basic operations performed by the DBMS are:

- Management of Data in the Database.
- Management of Users associated with the database.

Management of the data means to specify that how data will be stored, structured and accessed in the database.

Management of database users means to manage the users in such a way that they can perform any desired operations on the database. DBMS also ensures that a user cannot perform any operation for which he is not allowed. And also, an authorized user is not allowed to perform any action which is restricted to that user. **DBMS examples:**

- MySQL
- Microsoft Access
- SQL Server
- FileMaker
- Oracle
- RDBMS

2. Database:

A **database** is an organized collection of logically related data, usually designed to meet the information needs of multiple users in an organization. An organized collection of information in computerized format. It is important to distinguish between the database and the repository. The repository contains definitions of data, whereas the database contains occurrences of data.

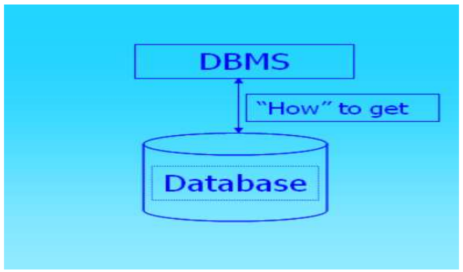


Figure No. 4.3: DBMS and Database

Database is used to store data and **DBMS** uses mechanisms to get data from the database.

3. Repository:

A **Repository** is a centralized collection of all data definitions, data relationships, output styles and report formats, and other system components. All this information is the metadata that is important to manage database. A **Repository** contains an extended set of metadata, important for managing databases as well as other components of an information system.

4. CASE Tools:

Computer-Aided Software Engineering (CASE) Tools are automated (Computerized) tools used to design databases and Application Programs. **Example of CASE Tools are:** Design Tools, Analysis Tools, Project Management Tools, Processing Modeling Tools, Diagram Tools, Database Management Tools, Documentation Tools. These tools help with creation of Data Models and in some cases can also help automatically to generate the "code", needed to create the database.

5. DataBase Administrators:

Data Administrators are persons who are responsible for the overall management of data resources in an organization. Database Administrators are responsible for physical database design and for managing technical issues in the database environment. This class of database users is the most technical class of database users. They need to have the knowledge of how to design and manage the database use as well as to manage the data in the database. DBA is a very responsible position in an organization. He is responsible for proper working of the database and DBMS, has the responsibility of making proper database backups and make necessary actions for recovering the database in case of a database crash. To fulfill the requirements of a DBA position, a DBA needs vast experience and very elegant technical skills.

Duties of a DataBase Administrator:

Once Database has been installed and is functioning properly in a production environment of an organization, the Database Administrator takes over the charge and performs specific DBA related activities including:

- Database Maintenance
- Database Backup
- Grant of rights to database users
- Monitoring of Running Jobs
- Managing Print jobs
- Ensuring quality of Service to all users
- Monitoring Disk Space

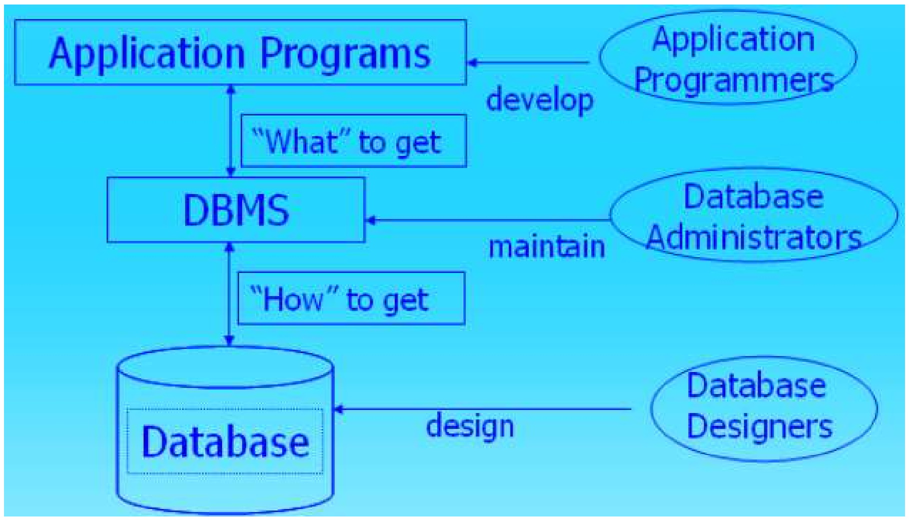


Figure No. 4.4: Database Administrator

6. **Application programs:**
Computer-based Application Programs are used to create and maintain the database and provide information to users. An Application Program is a program that is used to send commands to the Database Management System to manipulate database. These commands are sent to the DBMS through graphical user interface. The user interacts with Application Program and the Application Program further interacts with the Database Management System. Two important Application Programs are Developer2000 and Power Builder.

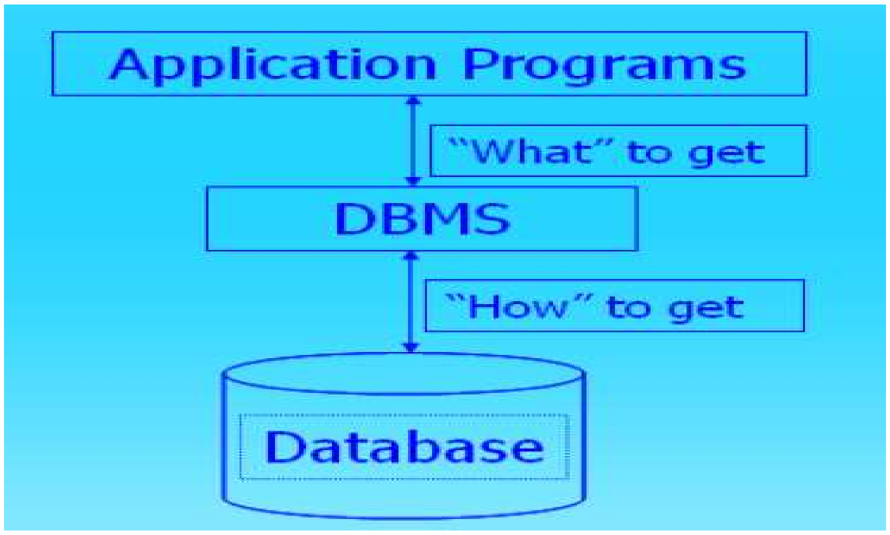


Figure No. 4.5: Application Programs

7. **Database Designers:**
Database Designers design (for large organizations) the database and install the DBMS for use by the users of the database in any specific organization.

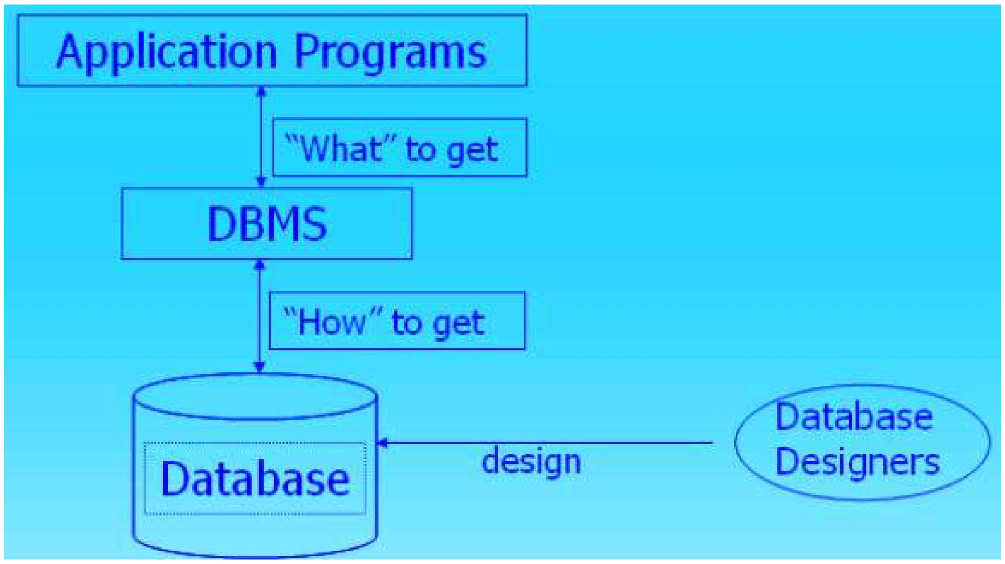


Figure No. 4.6: Database Designers

8. User Interface:
The User Interface is a visual environment that is used by the user to communicate with the computer. All windows-based software uses graphical User Interface. The user interface includes languages, menus, forms, buttons, and other facilities by which users interact with various system components, such as:

- CASE Tools.
- Application Programs.
- DBMS
- Repository.

9. Application Programmers:
Application Programmer is a professional who writes computer programs in a high-level language. These programs can be used to interact with DBMS. Application Programmer designs application programs according to the requirements of the users. Application Programmers are persons such as systems analysts and programmers who design new application programs. Application Programmers design the application according to the needs of the other users of the database in a certain environment. Application Programmers are skilled people who have clear idea of the structure of the database and know clearly about the needs of the organizations Application Programmers often use CASE tools for system requirements, analysis and program design.

10. End Users:
End Users are those persons who interact with the application directly. End Users are persons throughout the organization who add, delete, and modify data in the database and who request or receive information from it. All user interactions with the database must be routed through the DBMS. This group of users contains the people who use the database application programs developed by the Application programmers.

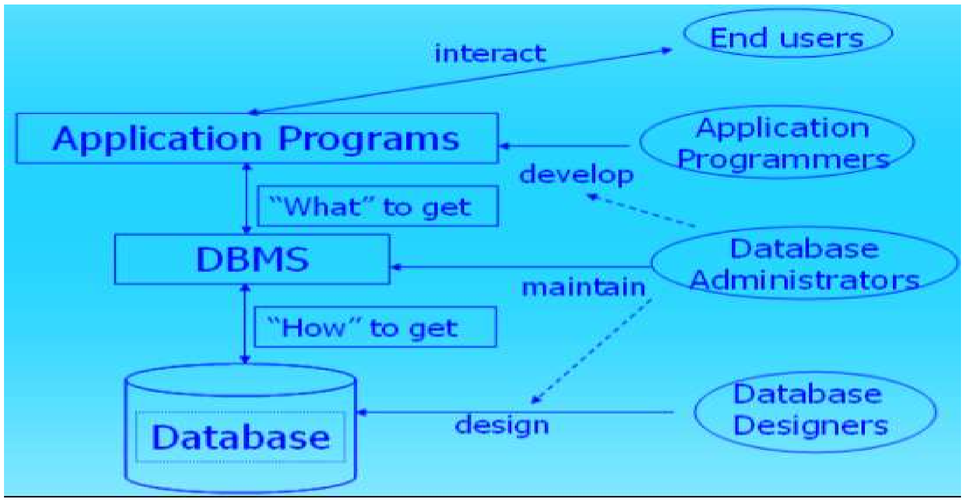


Figure No. 4.7: Database Administration's interaction with other users

Database Administrator can interact with the database designer during database design phase. DBA also interacts with the Application Programmers during the application development process and provides his services for better design of applications. End Users also interact with the system using application programs and other tools.

Week No. 05: Key and its types

Key:
A **Key** is a set of one or more attributes that can be used to identify or access a particular **Entity Instance** of an **Entity Type** (or out of an Entity Set). An **Entity Type** may have many instances, from a few to several thousands and even more. Now out of many instances, when to pick a particular/single instance then the **Key** is the used.

For example, think of whole population of **Pakistan**, the data of all Pakistanis lying at one place, say with **NADRA** people. Now if at some time, need to identify a particular person out of all this data. So, through **National ID Card** number, it can search easily, no matter whatever is the population of **Pakistan** and that is the **Key**.

A **Key** can be simple, that is, consisting of single attribute, or it could be composite which consists of two or more attributes. A **Key** is a data item used to identify a record. **Keys** are, as their name suggests, a key part of a Relational Database and a vital (important) part of the structure of a table. They ensure each record within a table can be uniquely identified by one or a combination of fields within the table. They help to enforce integrity and help to identify the relationship between tables.

Types of Keys:
There are the following main types of Keys:

1. Super Key
2. Candidate Keys
3. Primary Key
4. Foreign Key
5. Secondary Key/Alternate Key
6. Simple Key
7. Compound Key
8. Composite Key

1. Super key:
A **Super Key** is a set of one or more attributes which taken collectively, allow us to identify uniquely an **Entity Instance** in the **Entity Set**. This definition is same as of a **Key**, it means that the **Super Key** is the most **General Type** of **Key**.

For **example**, consider the Entity Type **STUDENT** with attributes registration number, name, father name, address, phone, class, admission date. Now which attribute can use that can uniquely identify any instance of **STUDENT** Entity Type. Of course, none of the name, father name, address, phone number, class, admission date can be used for this purpose. Because if consider name as **Super Key**, and situation arises that we need to contact the parents of a particular student. Now if we say to our registration department that give us the phone number of the student whose name is Ilyas Hussain, the registration department conducts a search and comes up with 10 different Ilyas Hussain, could be anyone. So, the value of the name attribute cannot be used to pick a particular instance. Same happens with other attributes. However, if we use the registration number, then it is 100% sure that with a particular value of registration number will always find exactly a single unique **Entity Instance**. Once you identified the instance, you have all its attributes available, name, father name, everything. The Entity Type **STUDENT** and its attributes are shown graphically in the figure no. 5.1 below, with its **Super Key** "regNo." underlined.

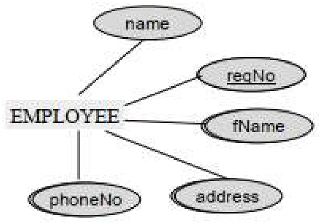
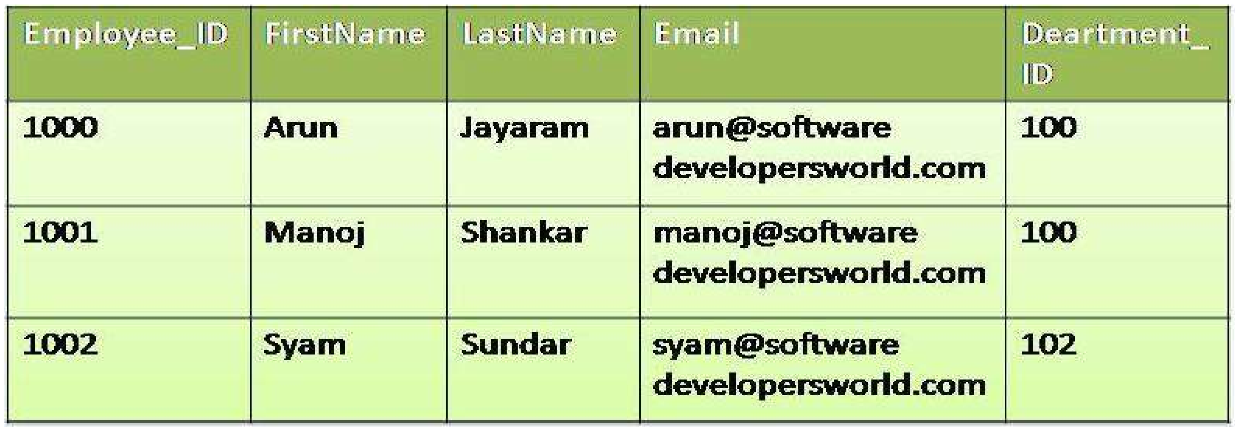


Figure No. 5.1: An Entity Type, its defining attributes and Super Key (underlined)

2. Candidate Key:

A **Candidate Key** is a single field or the least (smallest) combination of fields that uniquely identifies each record in the table. It is called **Candidate Key** because these **Keys** are the **candidate** for being a **Primary Key**. The most suitable **Candidate Key** become the **Primary Key**.

A **Candidate Key** is a set of one or more attributes that can uniquely identify a row in a given table. A **Candidate Key** is a **Minimal Super Key**. Let's explain **Candidate Key** in detail using an example. Consider a table **EMPLOYEE** as given below:

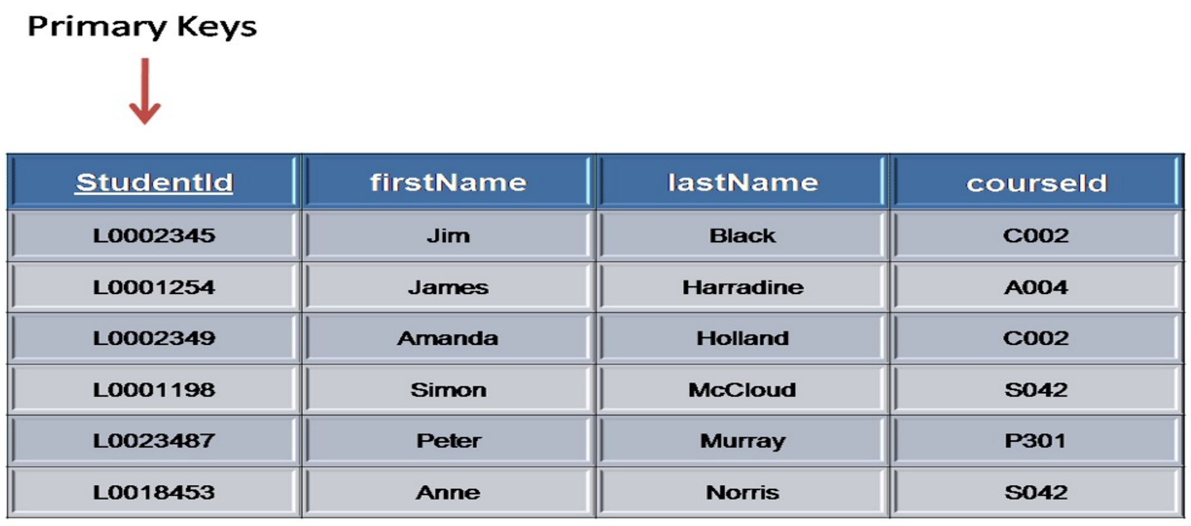


Employee_ID	FirstName	LastName	Email	Deartment_ID
1000	Arun	Jayaram	arun@software developersworld.com	100
1001	Manoj	Shankar	manoj@software developersworld.com	100
1002	Syam	Sundar	syam@software developersworld.com	102

Assume that **Employee_ID** and **Email** are unique. In this example, attribute **Employee_ID** can uniquely identify a row of the table **EMPLOYEE**. Hence **Employee_ID** is a **Candidate Key**. Similarly, the attribute **Email** can also uniquely identify a row of the table **EMPLOYEE**. Hence **Email** is also a **Candidate Key**. Hence for the table **EMPLOYEE**, have two **Candidate Keys**: Employee_ID and Email.

3. Primary Key:

Primary Key is a **Key** which uniquely identify each record in the table. **Primary Key** is the one **Super Key** that has minimum number of columns because of which **Primary Key** is also known as **Minimal Super Key**. **Primary Key** is a **Candidate Key** that is most appropriate to be the main **Reference Key** for the table. **Primary Key** is the **Reference** for the table and is used throughout the **database** to help, establish relationships with other tables. **Primary Key** must contain **Unique** values, must never be **Null** and uniquely identify each record in the table.


Primary Keys

StudentId	firstName	lastName	courseId
L0002345	Jim	Black	C002
L0001254	James	Harradine	A004
L0002349	Amanda	Holland	C002
L0001198	Simon	McCloud	S042
L0023487	Peter	Murray	P301
L0018453	Anne	Norris	S042

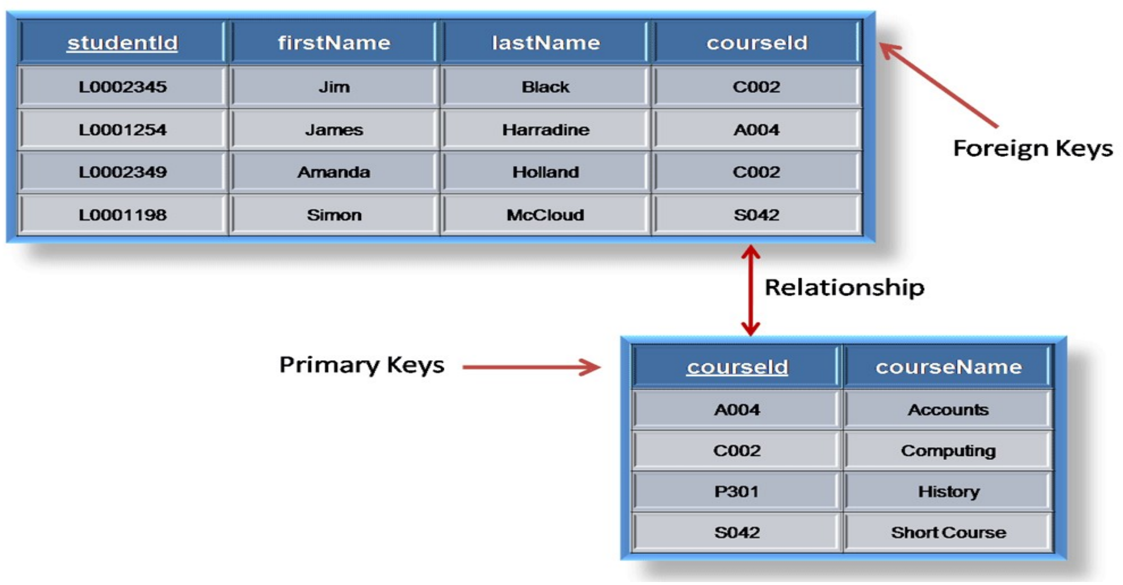
Primary Keys are mandatory for every table, each record must have a value for its **Primary Key**.

To select Primary Key from all Candidate Keys:

- Preference is given to numeric column(s).
- Preference is given to single attribute (Simple Key) over Composite Key.
- Preference is given to Minimal (smallest) Composite Key.

4. Foreign Key:

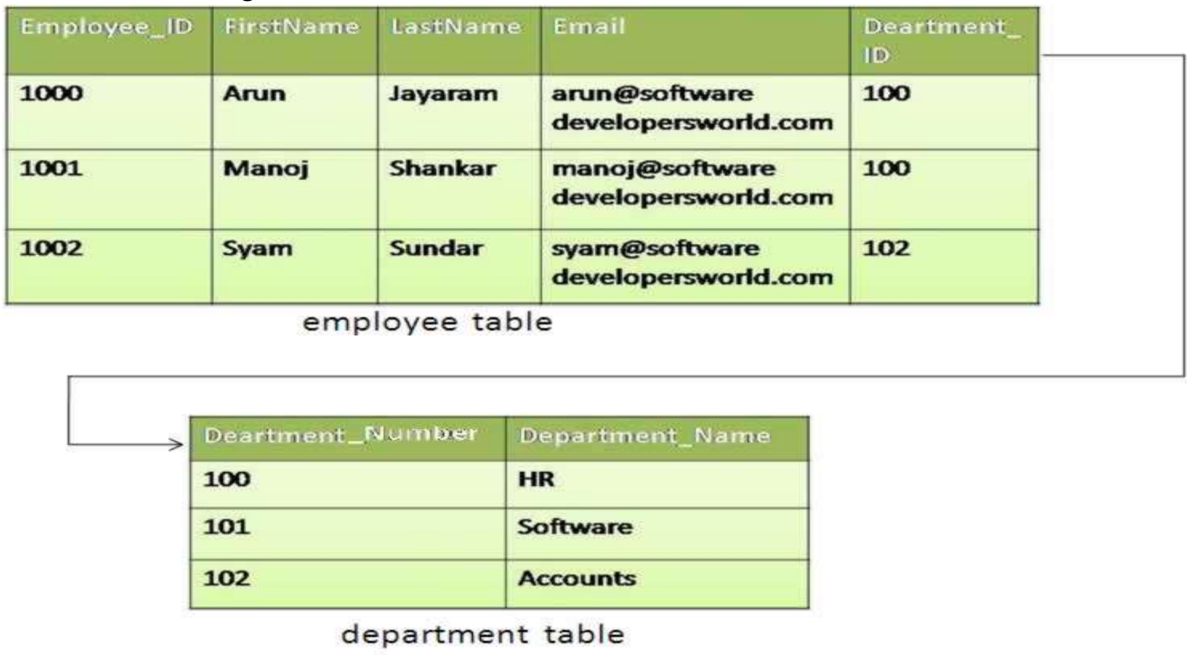
A **Foreign Key** is an attribute or a set of attributes in a relation whose values match a **Primary Key** in another relation. The relation in which **Foreign Key** is created is known as **Dependent Table** or **Child Table**. The relation to which the **Foreign Key** refers is known as **Parent Table**. A **Foreign Key** is generally a **Primary Key** from one table that appears as a field in another where the first table has a relationship to the second. In other words, if a **Table A** had a **Primary Key** X that linked to a **Table B** where X was a field in **B**, then **X** would be a **Foreign Key** in **table B**. An **example** might be a **Student Table** that contains the course_id, the student is attending. Another table lists the courses offer with course_id being the **Primary Key**. The two tables are linked through course_id and as such course_id would be a **Foreign Key** in the **Student Table**. **Foreign Key: Example-1**



studentId	firstName	lastName	courseId
L0002345	Jim	Black	C002
L0001254	James	Harradine	A004
L0002349	Amanda	Holland	C002
L0001198	Simon	McCloud	S042

courseId	courseName
A004	Accounts
C002	Computing
P301	History
S042	Short Course

Foreign Key: Example-2
Let's explain **Foreign Key** in detail using another example. Consider an **Employee Table** and a **Department Table** as given below:



Employee_ID	FirstName	LastName	Email	Deartment_ID
1000	Arun	Jayaram	arun@software developersworld.com	100
1001	Manoj	Shankar	manoj@software developersworld.com	100
1002	Syam	Sundar	syam@software developersworld.com	102

employee table

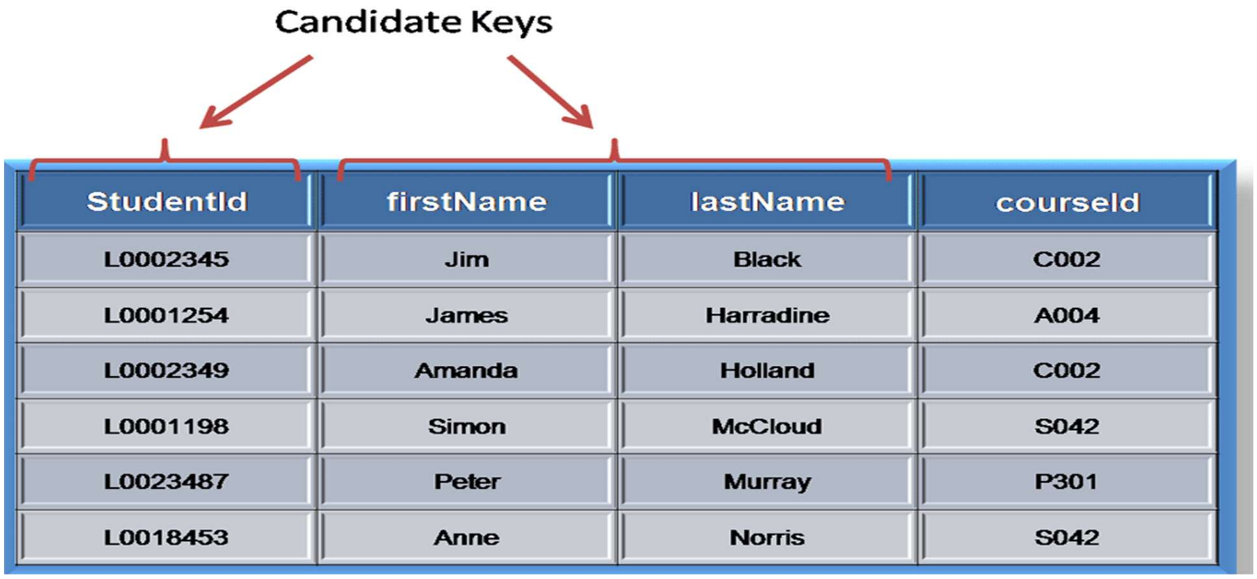
Deartment_Number	Department_Name
100	HR
101	Software
102	Accounts

department table

Here, Department_ID of **Employee Table** is a **Foreign Key** referring Department_Number which is a **Primary Key** of the **Department Table**.

5. Secondary Key/Alternate Key:
A table may have one or more choices for the **Primary Key**. Collectively these are known as **Candidate Keys**. One is selected as a **Primary Key**. Those not selected are known as **Secondary Keys** or **Alternative Keys**. **Candidate Keys** which are not chosen as a **Primary Key** are known as **Alternate Keys**. It is also called **Alternate Key** because these are **Alternate Keys** which may be **Primary Key**.

Example: Let's consider this example again. To explain Secondary/Alternate Key.



StudentId	firstName	lastName	courseId
L0002345	Jim	Black	C002
L0001254	James	Harradine	A004
L0002349	Amanda	Holland	C002
L0001198	Simon	McCloud	S042
L0023487	Peter	Murray	P301
L0018453	Anne	Norris	S042

In above table, a **student_id** that uniquely identifies the students in a **student table**, so this would be a **Candidate Key**. But in the same table, the student's **first name** and **last name** that also, when combined, uniquely identify the student in a **student table**. These would both be **Candidate Keys**. Here **Student_ID** is chosen as **Primary Key** and student's **first name** and **last name** is not selected so it is called **Secondary Key/Alternate Key**.

6. Simple Key:

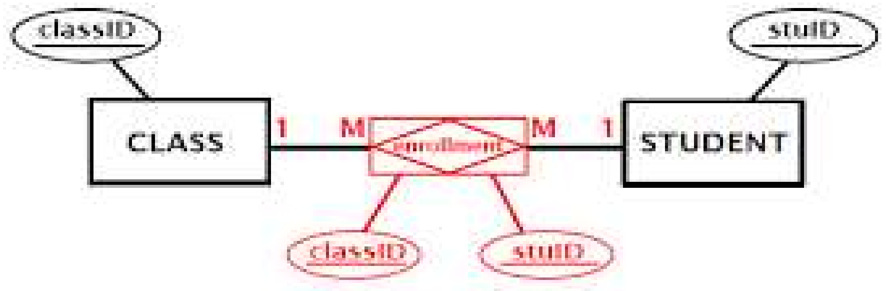
A **Simple Key** consists of a **single field** to uniquely identify a record. For **example**, studentId, which uniquely identifies a particular student, is a single field and therefore, it is called a **Simple Key**.

7. Compound Key:

A **Compound Key** consists of more than one field to uniquely identify a record. A **Compound Key** is distinguished from a **Composite Key** because each field, which makes up the **Primary Key**, is also a **Simple Key** in its own right. A **Compound Key** is created when two or more **Primary Keys** from different tables are present as **Foreign Key** within an entity.

Compound Keys are always created by combing two or more **Primary Keys** from two different tables. Both of these **Keys** uniquely identify data in their own tables, but in the table using the **Compound Key**, they are both required to uniquely identify data. A **Compound Key** is a **Composite Key** for which each attribute that makes up the **Key** is a **Simple** (Foreign) **Key** in its own right.

An **example** might be a **Table**: Enrollment that represents the Class, a student is attending. This **Table** has a studentId and a Class_ID as its **Primary Key**. Each of the fields that make up the **Primary Key** are **Simple Keys** because each represents a unique record when identifying a student and a class. The combination of both **Primary Keys** (Studend_ID and Class_ID) makes up a **Compound Key**.



8. Composite Key:

A **Composite Key** is a **Candidate Key** that consists of more than one attributes to uniquely identify a record. A **Composite Key** is a composition of two or more columns that uniquely identify rows in an entity. This differs from a **Compound Key** in that one or more of the attributes, which make up the **Key**, are not **Simple Keys** in their own right. Taking the example from **Compound Key**, imagine that a student uniquely identified by their **First Name** + **Last Name**. Both First and Last are not **Simple Keys** because they cannot uniquely identify a record by their own right. So, both First and Last attributes are **Composite Key**.

Relationship/Association:
Relationship represents an **association** between two or more entities.
An example of a **Relationship** would be:

- Employees are assigned to projects
- Projects have subtasks
- Departments manage one or more projects.

Relationships are the connections and interactions between the **Entity Instances** e.g., DEPT_EMP associates Department and Employee. Association between data. A **named** association between entities showing **Optionality** (May be) and **Degree** (How Many). Examples are employees and departments, and orders and items.

Each direction of the **Relationship** contains:

- A **Name:** For example, taught by or assigned to.
- An **Optionality:** Either must be or may be.
- A **Degree:** Either one and only one | or one or more.
- **Symbol** of **Relationship**
Relationship

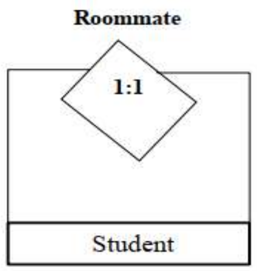
Types/Degree of Relationships/Association:

A **Relationship** may consist of many **Entity Types**. The **number** of **Entity Types** in a **Relationship** is called **Degree** of **Relationship**. The **Relationships** of **Degree 2** are most common and are also called **Binary Relationships**. The **Types** of **Relationships** with respect to **Degree** are as **follows**:

1. Unary Relationship
2. Binary Relationship
3. Ternary Relationship

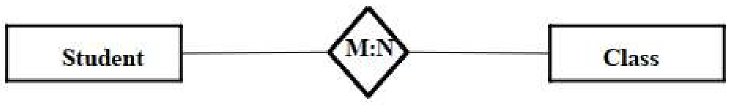
1. Unary Relationship/Association:
Unary Relationship is a type of **Relationship** that is established between the **Instances** of same **Entity Type**. An **Entity Type** linked with **itself**, also called **Unary**/**Recursive Relationship** e.g., Roommate, where STUDENT is linked with STUDENT. The **Degree** of **Unary Relationship** is **1**.

Example:



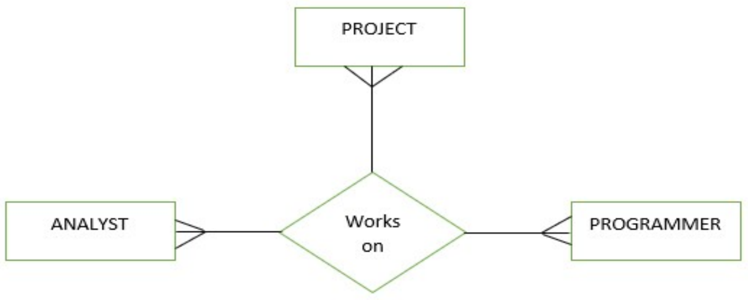
2. Binary Relationship/Association:
Binary Relationships exist between the **Instances** of two **Entity Types**. A **Binary Relationship** is the one that links two Entities Sets e.g., STUDENT - CLASS. The **Degree** of **Binary Relationship** is **2**.

Example:

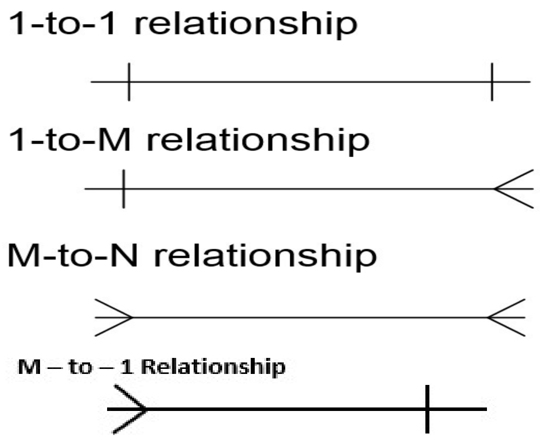


3. Ternary Relationship/Association:
Ternary Relationship exists among the **Instances** of three **Entity Types**. A **Ternary Relationship** is the one that involves three **Entity Types.** The **Degree** of **Ternary Relationship** is **3**. The following **Relationship** means that one or many **Analysts** with one or many **Programmers** work on one or more **Projects**.

Example:

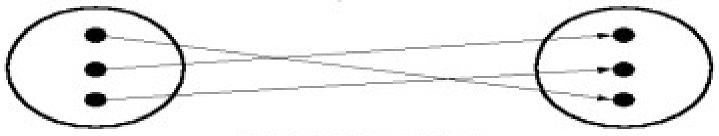


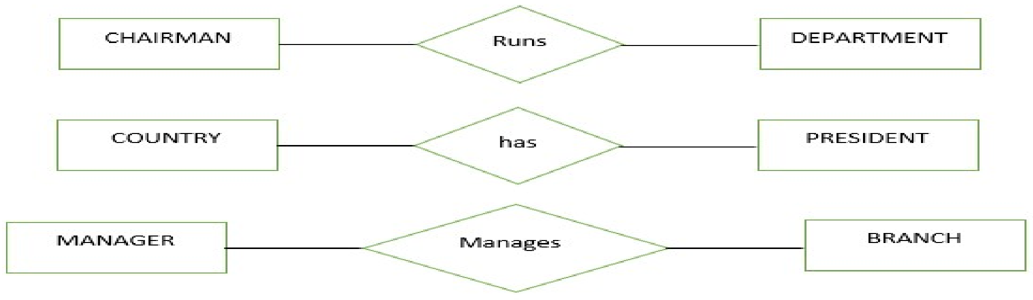
Relationship Basic Cardinalities:
The **Cardinality** of a **Relationship** is the **number** of **Entity Instances** that can be associated with another **Entity Instance** under that **Relationship**. **Symbols for Cardinalities are:**



Types of Relationship Cardinality:
There are **four types** of **Relationship Cardinality** as **follows**:

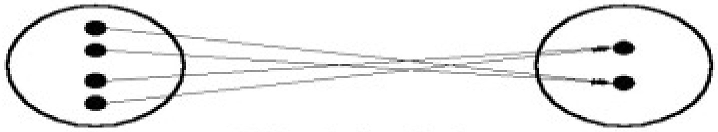
i. One-to-One Relationship:
A **Relationship** from **X** to **Y** is **one-to-one** if **each Entity Instance** in **X** is associated with at most **one Entity Instance** in **Y** and **each Entity Instance** in **Y** is associated with at most **one Entity Instance** in **X**.


(a) One-to-One Mapping

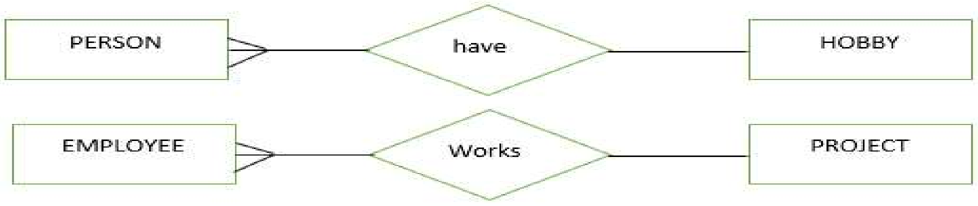


ii. Many-to-One Relationship:

A **Relationship** from **X** to **Y** is **many-to-one** if **each Entity Instance** in **X** is associated with at most **one Entity Instance** in **Y** but **each Entity Instance** in **Y** is associated with **many Entity Instances** in **X**.

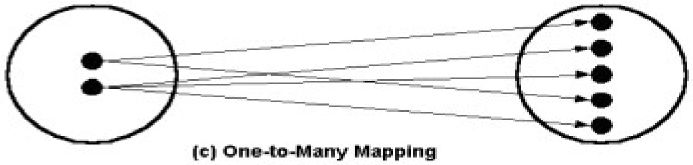


(a) Many-to-One Mapping

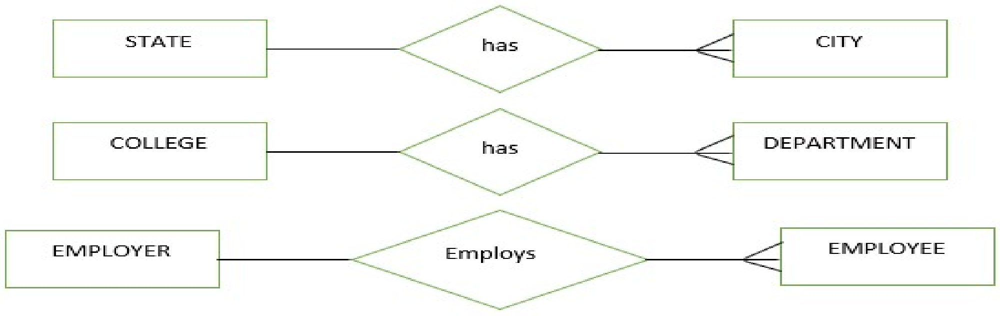


iii. One-to-Many Relationship:

A **Relationship** from **X** to **Y** is **one-to-many** if **each Entity Instance** in **X** is associated with **many Entity Instances** in **Y** but **each Entity Instance** in **Y** is associated with **one Entity Instance** in **X**.

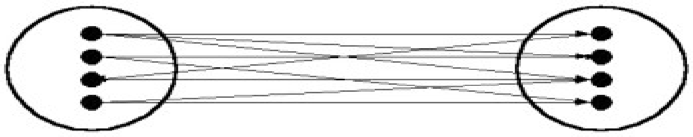


(c) One-to-Many Mapping

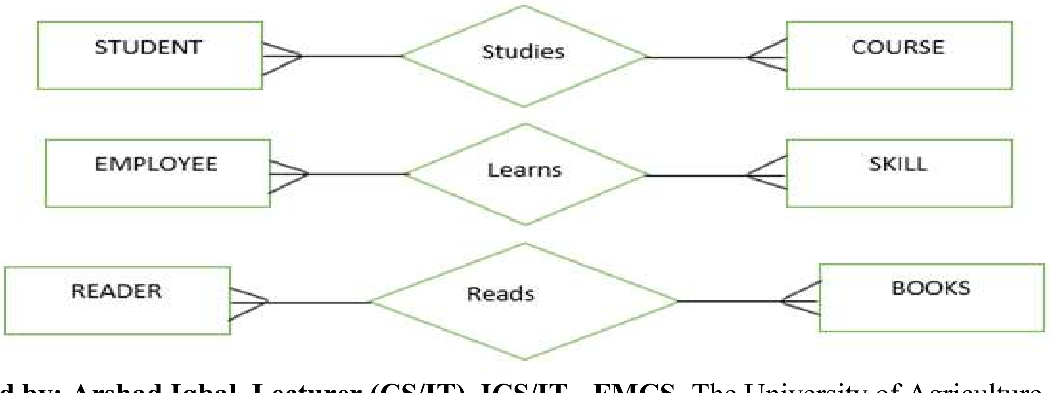


iv. Many-to-Many Relationship:

A **Relationship** from **X** to **Y** is **many-to-many** if **each Entity Instance** from **X** is associated with **many Entity Instances** in **Y** and **one Entity Instance** in **Y** is associated with **many Entity Instances** in **X**.



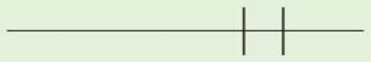
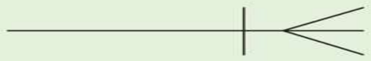
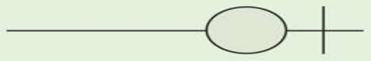
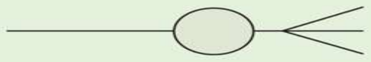
(d) Many-to-Many Mapping



Cardinality Constraints:

The Cardinality Constraint specifies the number of instances of one entity Type that can be associated with each instance of the other Entity Type. There are three symbols to show Cardinality Constraint. A circle (**O**) means **Zero**, a line (|) means **one** and crow's foot symbol ⩤ means **many**. A circle **O** indicates that relationship is **Optional**. A stroke | indicates that relationship **Mandatory**. A crows-foot ⩤ indicates that many relationships between instances.

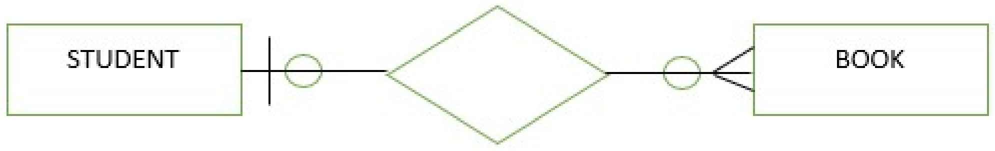
Cardinality Constraint Symbols are:

Symbol	Meaning
	Mandatory—One
	Mandatory—Many
	Optional—One
	Optional—Many

Minimum Cardinality:
The minimum number of instances of one Entity Type that may be associated with each instance of another Entity Type is known as **Minimum Cardinality**. If the **Minimum Cardinality** is **Zero**, participation is **Optional**. It the **Minimum Cardinality** is **One**, participation is **Mandatory**.

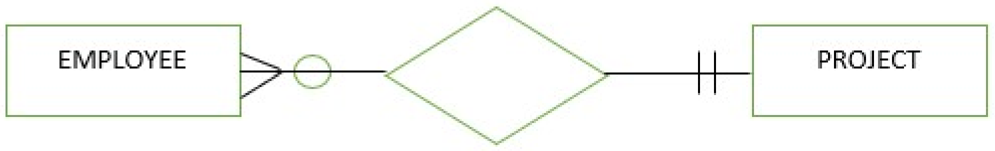
Maximum Cardinality:
The **maximum** number of Instances of one Entity Type that may be associated with each Instance of another Entity Type as **Maximum Cardinality**.

Example No. 01: **One – to – Many Relationship (Optional)**



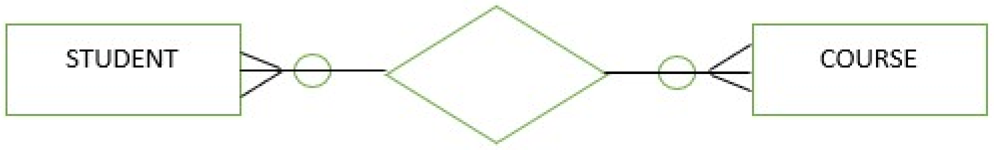
The above **ER-Diagram** shows that the **Minimum** Cardinality of **STUDENT** is **Zero** and **Maximum** Cardinality is **1**. The **Minimum** Cardinality of **BOOK** is **Zero** and **Maximum** Cardinality is **many**.

Example No. 02: Many – to – One Relationship (Mandatory)



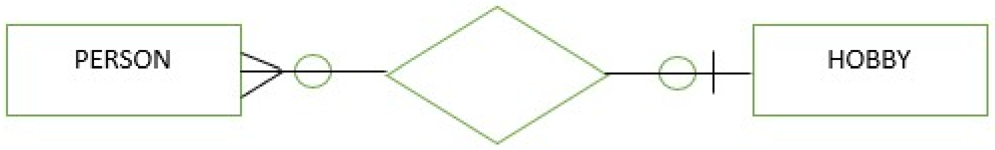
The above **ER-Diagram** shows that the **Minimum** Cardinality of **EMPLOYEE** is **Zero** and **Maximum** Cardinality is **many**. The **Minimum** Cardinality of **PROJECT** is **1** and **Maximum** Cardinality is also **1**.

Example No. 03: Many – to – Many Relationship (Optional)



The above **ER-Diagram** shows that the **Minimum** Cardinality of **STUDENT** is **Zero** and **Maximum** Cardinality is **many**. The **Minimum** Cardinality of **COURSE** is **Zero** and **Maximum** Cardinality is **many**.

Example No. 04: Many – to – One Relationship (Optional)



The above **ER-Diagram** shows that the **Minimum** Cardinality of **PERSON** is **Zero** and **Maximum** Cardinality is **many**. The **Minimum** Cardinality of **HOBBY** is **Zero** and **Maximum** Cardinality is **1**.

Week No. 06: Relational Data Model

Relational Data Model:

The **Relation Data Model** is a **simple model** because there is just **one structure** and that is a **Relation** or a **Table**. Even this **single structure** is very easy to understand, so a user of even of a moderate genius can understand it easily.

A **Relational Data Model** involves the use of **data tables** that collect groups of elements into **Relations**. These **models** work based on the idea that each table setup will include a **Primary Key** or **Identifier**. The **Primary Key** is a **fundamental tool** in creating and using **Relational Data Models**. It must be **unique** for each member of a data set.

In **E-R Data Model**, both the **Entity Types** and **Relationships** are represented using **Relations** in **Relational Data Model**.

The **Relation** in **Relational Data Model** is **similar** to the **Mathematical Relation** however **Database Relation** is also represented in a **Two-Dimensional Structure** called **Table**. A **Table** consists of **rows** and **columns**. **Rows** of a **Table** are also called **Tuples**. A **Row** or **Tuple** of a **Table** represents a **Record** or an **Entity Instance**, whereas the **columns** of the **Table** represent the **Properties** or **Attributes**.

Table No. 6.1: Relational Table

stID	stName	clName	doB	sex
S005	Ehsan M.	BBA	22/7/88	M
S004	Rubab A.	MBA	23/4/86	F
S003	Naila S.	MCS	7/8/85	F
S002	M. Shahid	BCS	3/9/86	M
S001	M. Suhail	MCS	12/6/84	M

A **Database Relation** represented in the **form** of a **Table**. In the above diagram, a **Table** is shown that consists of **five rows** and **five columns**. The top most rows contain the names of the columns or attributes whereas the rows represent the records or Entity Instances.

Relational Data Model Basic Terminologies:

Table No. 6.2: Student

Roll No.	Name	Address	Phone	Age
1	Khan	Peshawar	0332999955555	19
2	Ali	Mardan	011245745	31
3	Ahmad	Swat		20
4	Nasir	Kohat	0112335426	19

1. **Attribute:** Attributes are the **properties** that define a **Relation** e.g., **ROLL_NO**, **NAME**.

2. **Relation Schema:** A **Relation Schema** represents **name** of the **Relation** with its **attributes** e.g., STUDENT (ROLL_NO, NAME, ADDRESS, PHONE and AGE) is **Relation Schema** for STUDENT.

3. **Tuple:** Each **Row** in the **Relation** is known as **Tuple**. The above relation contains **4 tuples**, one of which is shown as:

1	Khan	Peshawar	033299995555	19

4. **Relation Instance:** The **set** of **Tuples** of a **Relation** at a **particular instance** of **time** is called **Relation Instance**. **Table No. 6.2** shows the **Relation Instance** of STUDENT at a particular time. It can change whenever there is insertion, deletion or updation in the database.

5. **Degree:** The **number** of **Attributes** in the **Relation** is known as **Degree** of the **Relation**. The above **STUDENT** Relation has defined **Degree 5**.

6. **Cardinality:** The **number** of **Tuples** in a **Relation** is known as **Cardinality**. The above **STUDENT** Relation has defined **Cardinality 4**.

7. **Column:** Column represents the **set** of **values** for a **particular Attribute**. The column **ROLL_NO** is extracted from Relation **STUDENT**.

Rollno
1
2
3
4

8. **NULL Value:** The **value** which is **not known** or **unavailable** is called **NULL Value**. It is represented by **blank space**. e.g.; PHONE of STUDENT having ROLL_NO 3 is **NULL**.

9. **Relation Key:** Every **Row** has one, two or multiple **Attributes**, which is called **Relation Key**.

10. **Attribute Domain:** Every **Attribute** has some **pre-defined value** and **scope** which is known as **Attribute Domain**

Advantages of a Relational Data Model:

1. Data Integrity
2. Data Independence
3. Structural Independence
4. Data Consistency and Accuracy
5. Easy Data Retrieval and Sharing

1. **Data Integrity:**
   Relational Data Model allows **Data Integrity** from **Field Level** to **Table Level** to avoid **duplication** of **records**. It **detects** records with missing **Primary Key values** at the **Relationship Level** to ensure **Valid Relationships** between **Relations**.

2. **Data Independence:**
   The **implementation** of **Database** will not be **affected** by **changes** made in the **Logical Design** of the **Database** or **changes** made in the **Database Software**.

3. **Structural Independence:**
   Structural Independence exists when the **Structure** of **Database** can be **changed** without **affecting DBMSs** ability to **access** the **data**. Any **change** in **Relational Database Structure** does not affect **data access** in any way. It makes **Relational Databases** as a **Model Structural Independence**.

4. **Data Consistency and Accuracy:**
   Since **multiple level check** and **constraints** are built in, that make data accurate and consistent.

5. **Easy Data Retrieval and Sharing:**
   Data can be **easily extracted** from **one** or **multiple Relations**. **Data** can also be **easily shared** among **users**.

Types of Relation:

Different **Types** of **Relations** in **Relational Data Model** are as **follows**:

1. Base Table
2. Query Result Table
3. View Table

1. **Base Table:**
   A **Base Table** is a **Table** that exists in a **Database**. It is a **Table** that is created by the **user**. It is **not derived** from **another Table**. A **Base Table** can be created, altered and removed from a **database**. These **tasks** are accomplished using **SQL** statements.

2. **Query Result Table:**
   Query is a **question** in which the **user** asks the **Database Management System** to perform different **operations** on **Tables**. For **example**, a **user** may wish to see all students who got an **A grade**. When a **question** is asked and **Query** is executed, the **resultant data** is also stored in a **Table**. Such **Tables** are called **Query Result Tables**.

3. **View Table:**
   A **View** is a **Virtual Table**. Suppose a **user** wants to **View** a few columns of a **Base Table** instead of **all columns**. A **View** can be created to fulfill this request. The **View** consists of only those **columns** of the **Base Table** that the user required. This format can be saved as a **View** by giving it a name.

Base Table:

Registration No.	Roll No.	Name	Class
96-AG-1940	1	Nadeem Khalil	MSc.
96-AG-1889	2	Ejaz Saeed	MSc.
96-AG-1991	3	Nauman Qadeer	MSc.

View Table:

Registration No.	Name	Class
96-AG-1940	Nadeem Khalil	MSc.
96-AG-1889	Ejaz Saeed	MSc.
96-AG-1991	Nauman Qadeer	MSc.

Basic Characteristics/Properties of Relational Data Model:

1. Unique Relation Name
2. Atomic/Single value in a Cell
3. Unique Attribute Name
4. Domain of an Attribute
5. Order of Attributes
6. Order of Tuples
7. Unique/Distinct Tuples

1. **Unique Relation Name:** Each **Relation** (Table) has a **Unique Name**.

2. **Atomic/Single Value in a Cell:** Each **Cell** of a **Table** contains **Atomic**/**Single** Value. A **Cell** is the **intersection** of a **Row** and a **Column**, so it represents a value of an attribute in a particular **row**. The **property** means that the **value** stored in a **single cell** is considered as a **single value**. In real life, there are many situations when a **property**/**attribute** of any **entity** contains **multiple values**, **like** degrees that a person has, hobbies of a student, the cars owned by a person, the jobs of an employee. All these **attributes** have **multiple values**, these values cannot be placed as the value of a single attribute or in a **cell** of the **table**. It does not mean that the **Relation Data Model** cannot handle such situations, **however**, there are some special means that have to adopt in these situations, and they cannot be placed as the **value** of an **attribute** because an **attribute** can contain only a **single value**. The **values** of **attributes** shown in **table no. 6.1**, are all **Atomic** or **Single**.

3. **Unique Attribute Name:** Each **Column** has a **Unique Name**. Each **Column** has a **heading** that is basically the **name** of the **attribute** which represents the **Column**. It has to be **unique**, that is, a **table** cannot have **duplicated column**/**attribute names**. In the above **table**, the **bold items** in the **first row** represent the **Column**/**attribute names**.

4. **Domain of an Attribute:** The **values** of the **Attributes** come from the **same Domain**. Each **attribute** is assigned a **domain** along with the **name** when it is defined. The **domain** represents the **set** of **possible values** that an **attribute** can have. Once the **domain** has been assigned to an **attribute**, then all the **rows** that are added into the **table** will have the **values** from the **same domain** for that **particular column**. For **example**, in the **above table**, the **attribute** doB (date of birth) is assigned the **domain** "Date", now all the **rows** have the **date value** against the **attribute** of **doB**. This attribute cannot have a text or numeric value.

5. **Order of Attributes:** The **Order** of the **Columns** is **immaterial** (unimportant), If the **Order** of the **Columns** in a **Table** is **changed**, the **table** still remains the **same**. **Order** of the **Columns** does not matter.

6. **Order of Tuples:** The **Order** of the **Rows** is **immaterial** (unimportant) as with the **Columns**, if **Rows Order** is **changed** the **Table** remains the **same**. So, the **Order** of the **Rows** doesn't matter.

7. **Unique/Distinct Tuples:** Each **Row**/**Tuple**/**Record** is **Distinct**, no two **Rows** can be **same**, means **two rows** of a **Table** cannot be **same**. The **value** of even a **single attribute** has to be different that makes the entire Row **Distinct**.