Dnyansagar Arts & Commerce College

(BBA(CA) 2019 Pattern Sem I)

Database Management System

(104)

By

Prof Gayatri A Amate

Unit 1: Data Base Management System

Index

Database

A database is a collection of related data which represents some aspect of the real world.

DBMS

Database Management System (DBMS)

- It is a software for storing and retrieving user's data with security.

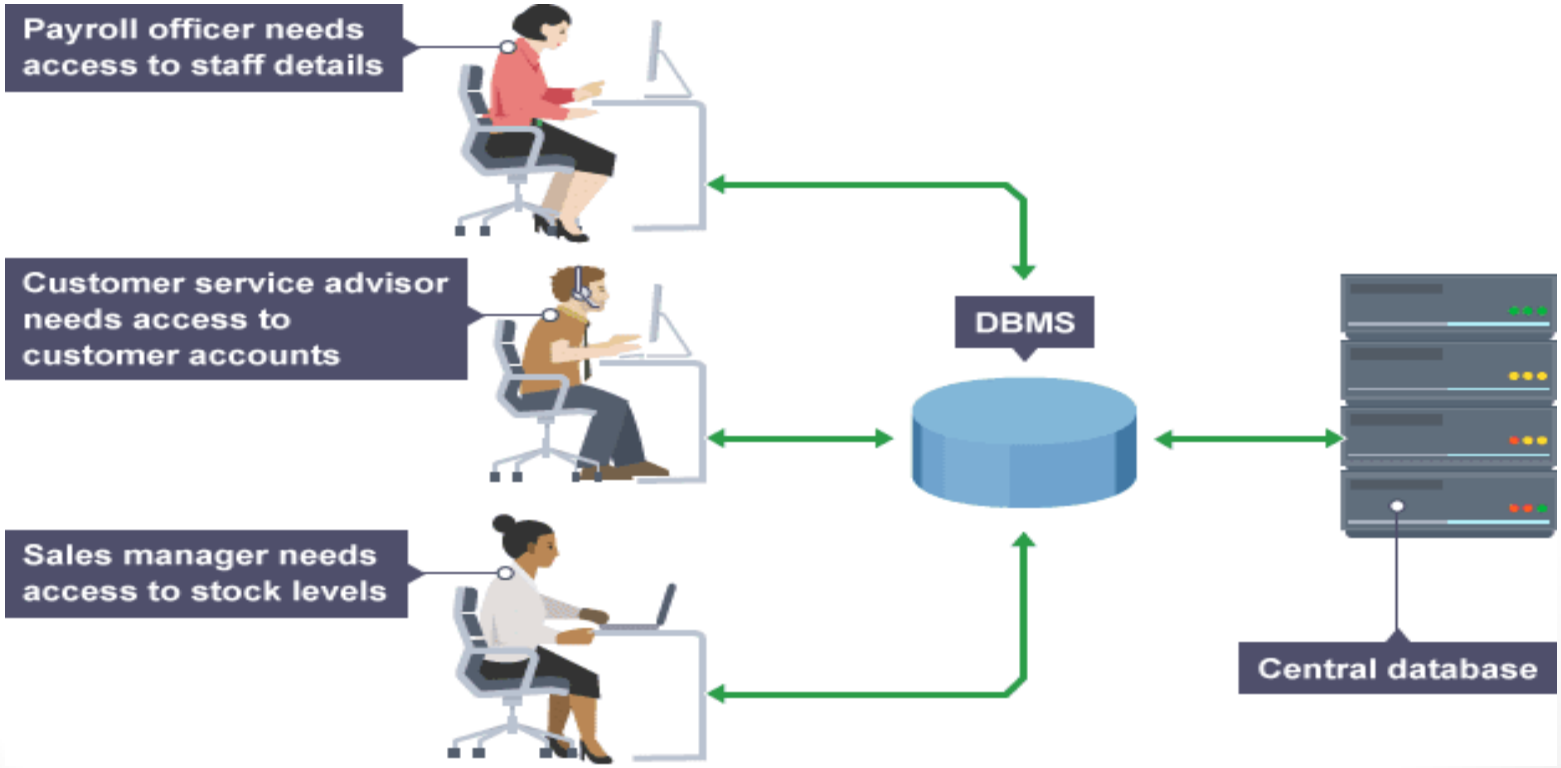
- The DBMS accepts the request for data from user and instructs the operating system to provide the specific data

What is DBMS?

- DBMS allows users to create their own databases as per their requirement.

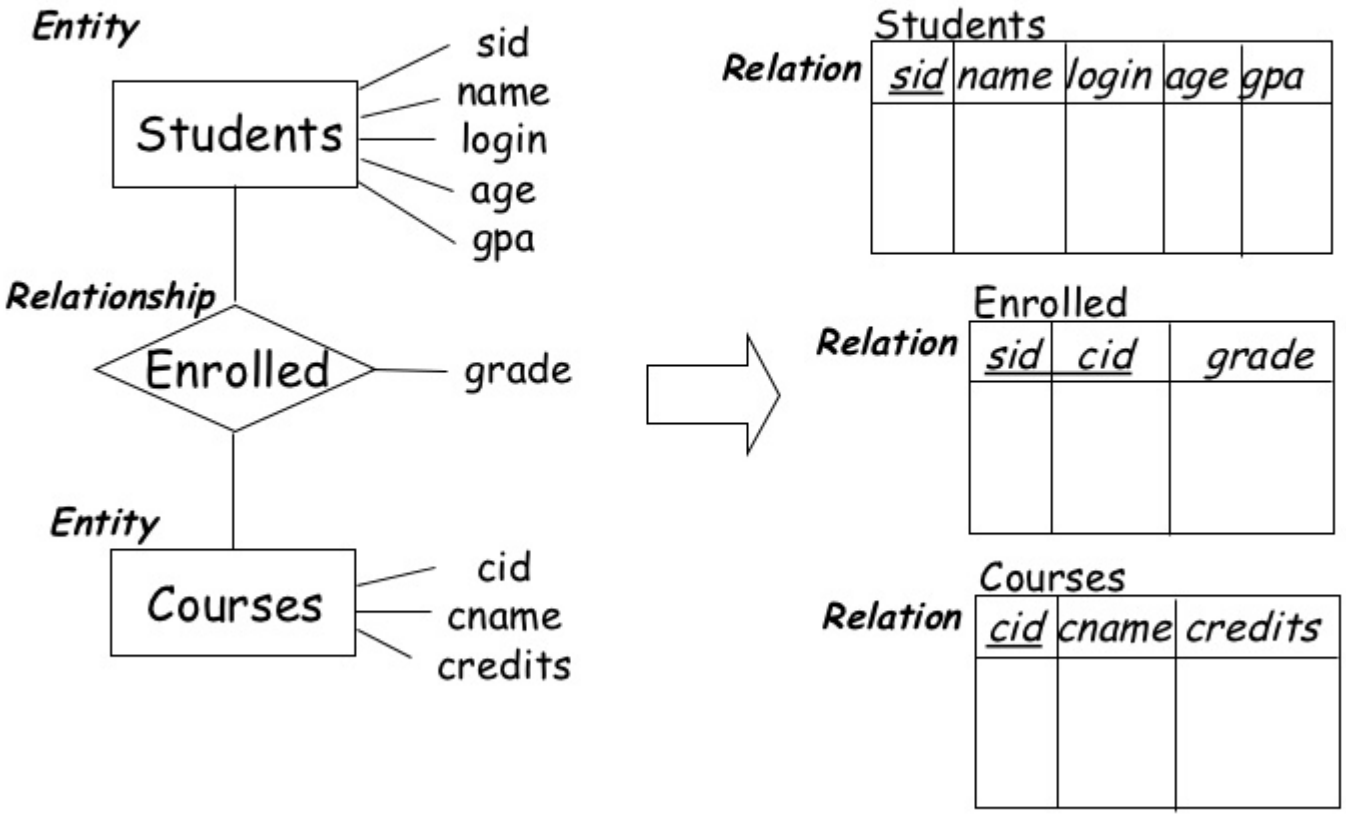
DBMS

- It provides an interface between the data and the software application



Example

Example: University Database



Application of DBMS

★ Banking: all transactions

★ Airlines: reservations, schedules

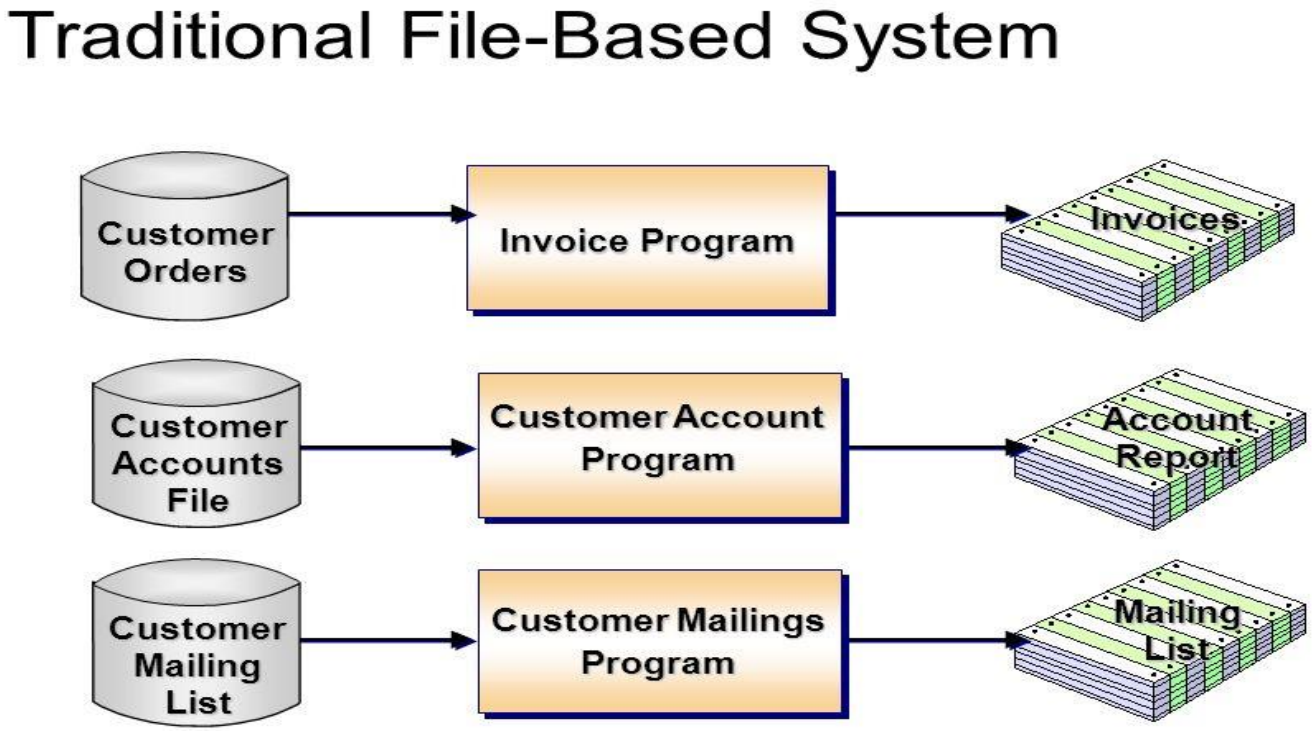
★ Universities: registration, grades

★ Sales: customers, products, purchases

★ Manufacturing: production, inventory, orders, supply chain

★ Human resources: employee records, salaries, tax deductions

Traditional File Based system



Purpose of Database System

In the early days, database applications were built on top of file systems

■ Drawbacks of using file systems to store data:

★ **Data redundancy and inconsistency**

✔ Multiple file formats, duplication of information in different files

★ **Difficulty in accessing data**

✔ Need to write a new program to carry out each new task

★ **Integrity problems**

✔ Integrity constraints (e.g. account balance > 0) become part of program code

✔ Hard to add new constraints or change existing ones

Purpose of Database Systems (Cont.)

★ **Atomicity of updates**

✔ Failures may leave database in an inconsistent state with partial updates carried out

✔ E.g. transfer of funds from one account to another should either complete or not happen at all

★ **Concurrent access by multiple users**

✔ Concurrent accessed needed for performance

✔ Uncontrolled concurrent accesses can lead to inconsistencies – E.g. two people reading a balance and updating it at the same time

★ Security problems

■ Database systems offer solutions to all the above problems

Database Administrator

- Coordinates all the activities of the database system
- The database administrator has a good understanding of the enterprise's information resources and needs.

Database administrator's duties include

★ Granting user authority to access the database

★ Specifying integrity constraints

★ Monitoring performance and **responding** to changes in requirements

Advantages of DBMS

- Compared to the File Based Data Management System, Database Management System has many advantages. Some of these advantages are given below –

Reducing Data Redundancy

- The file based data management systems contained multiple files that were stored in many different locations in a system or even across multiple systems. Because of this, there were sometimes multiple copies of the same file which lead to data redundancy.
- This is prevented in a database as there is a single database and any change in it is reflected immediately. Because of this, there is no chance of encountering duplicate data.

Sharing of Data

- In a database, the users of the database can share the data among themselves. There are various levels of authorization to access the data, and consequently the data can only be shared based on the correct authorization protocols being followed.
- Many remote users can also access the database simultaneously and share the data between themselves.

Data Integrity

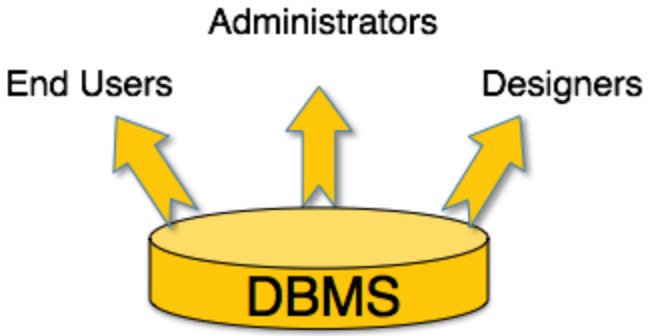
Data integrity means that the data is accurate and consistent in the database. Data Integrity is very important as there are multiple databases in a DBMS. All of these databases contain data that is visible to multiple users. So it is necessary to ensure that the data is correct and consistent in all the databases and for all the users.

Data Security

- Data Security is vital concept in a database. Only authorized users should be allowed to access the database and their identity should be authenticated using a username and password.
- Unauthorized users should not be allowed to access the database under any circumstances as it violates the integrity constraints

Users

A typical DBMS has users with different rights and permissions who use it for different purposes. Some users retrieve data and some back it up. The users of a DBMS can be broadly categorized as follows



Different Users

- Administrators − Administrators maintain the DBMS and are responsible for administrating the database. They are responsible to look after its usage and by whom it should be used. They create access profiles for users and apply limitations to maintain isolation and force security. Administrators also look after DBMS resources like system license, required tools, and other software and hardware related maintenance.

- Designers − Designers are the group of people who actually work on the designing part of the database. They keep a close watch on what data should be kept and in what format. They identify and design the whole set of entities, relations, constraints, and views.

- End Users − End users are those who actually reap the benefits of having a DBMS. End users can range from simple viewers who pay attention to the logs or market rates to sophisticated users such as business analysts.

Different Users

- **Sophisticated Users :**
  Sophisticated users can be engineers, scientists, business analyst, who are familiar with the database. They can develop their own data base applications according to their requirement. They don't write the program code but they interact the data base by writing SQL queries directly through the query processor.
-

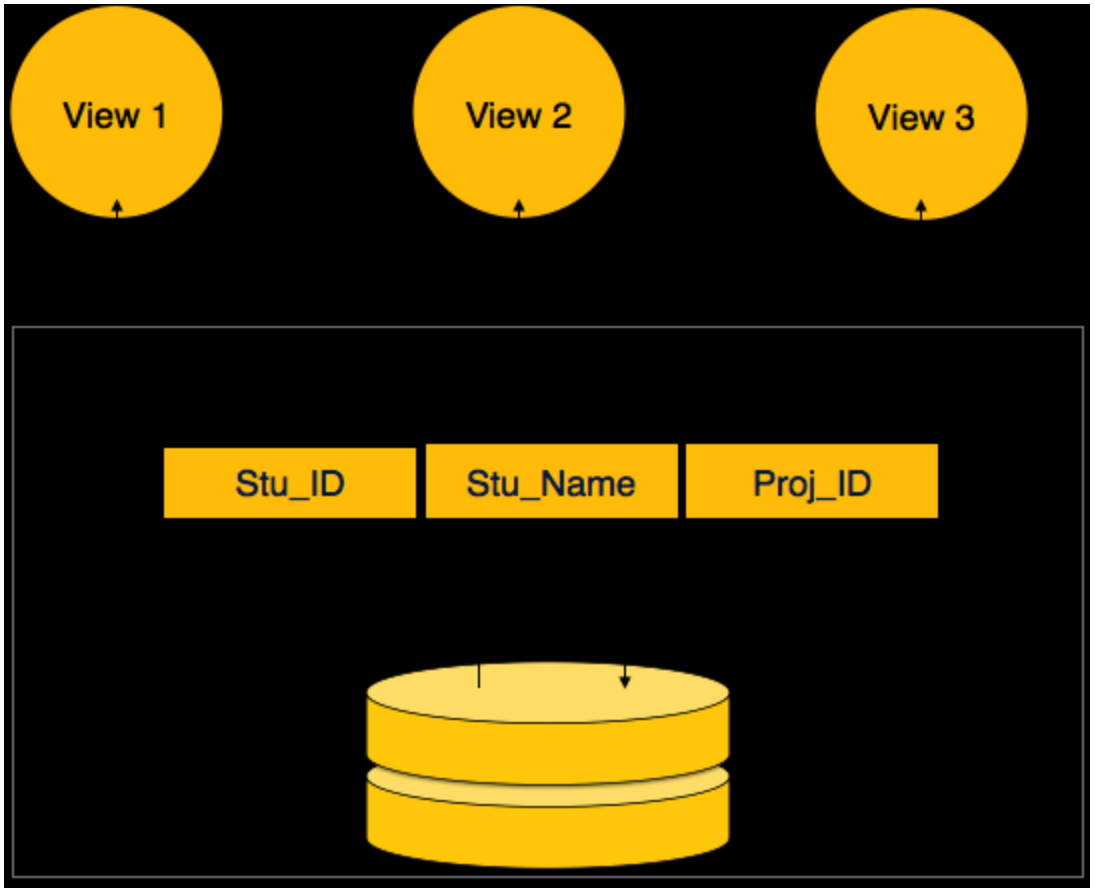
- **Data Base Designers :**
  Data Base Designers are the users who design the structure of data base which includes tables, indexes, views, constraints, triggers, stored procedures. He/she controls what data must be stored and how the data items to be related.

Database Schema

- A database schema is the skeleton structure that represents the logical view of the entire database. It defines how the data is organized and how the relations among them are associated. It formulates all the constraints that are to be applied on the data.

- A database schema defines its entities and the relationship among them. It contains a descriptive detail of the database, which can be depicted by means of schema diagrams. It’s the database designers who design the schema to help programmers understand the database and make it useful.

Database Schema



Database Schema

- A database schema can be divided broadly into two categories –

- Physical Database Schema – This schema pertains to the actual storage of data and its form of storage like files, indices, etc. It defines how the data will be stored in a secondary storage.

- Logical Database Schema – This schema defines all the logical constraints that need to be applied on the data stored. It defines tables, views, and integrity constraints.

Database Schema

- Database Instance
- It is important that we distinguish these two terms individually. Database schema is the skeleton of database. It is designed when the database doesn't exist at all. Once the database is operational, it is very difficult to make any changes to it. A database schema does not contain any data or information.

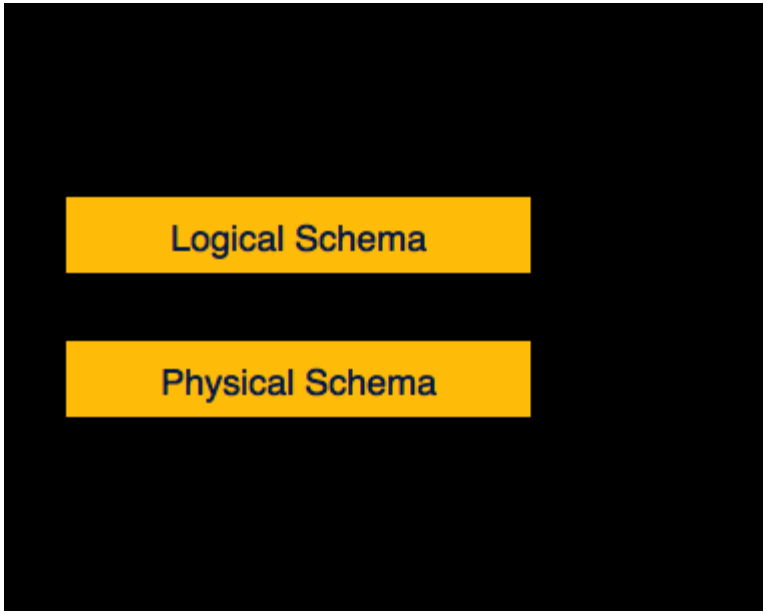
- A database instance is a state of operational database with data at any given time. It contains a snapshot of the database. Database instances tend to change with time. A DBMS ensures that its every instance (state) is in a valid state, by diligently following all the validations, constraints, and conditions that the database designers have imposed.

Database Schema

- If a database system is not multi-layered, then it becomes difficult to make any changes in the database system. Database systems are designed in multi-layers as we learnt earlier.

- Data Independence
- A database system normally contains a lot of data in addition to users' data. For example, it stores data about data, known as metadata, to locate and retrieve data easily. It is rather difficult to modify or update a set of metadata once it is stored in the database. But as a DBMS expands, it needs to change over time to satisfy the requirements of the users. If the entire data is dependent, it would become a tedious and highly complex job.

Database Schema



Database Schema

- Metadata itself follows a layered architecture, so that when we change data at one layer, it does not affect the data at another level. This data is independent but mapped to each other.

- Logical Data Independence
- Logical data is data about database, that is, it stores information about how data is managed inside. For example, a table (relation) stored in the database and all its constraints, applied on that relation.

- Logical data independence is a kind of mechanism, which liberalizes itself from actual data stored on the disk. If we do some changes on table format, it should not change the data residing on the disk.

Physical Data Independence

- All the schemas are logical, and the actual data is stored in bit format on the disk. Physical data independence is the power to change the physical data without impacting the schema or logical data.

- For example, in case we want to change or upgrade the storage system itself − suppose we want to replace hard-disks with SSD − it should not have any impact on the logical data or schemas

ER-Model

- The ER model defines the conceptual view of a database. It works around real-world entities and the associations among them. At view level, the ER model is considered a good option for designing databases.
- Entity
- An entity can be a real-world object, either animate or inanimate, that can be easily identifiable. For example, in a school database, students, teachers, classes, and courses offered can be considered as entities. All these entities have some attributes or properties that give them their identity.

ER-Model

- An entity set is a collection of similar types of entities. An entity set may contain entities with attribute sharing similar values. For example, a Students set may contain all the students of a school; likewise a Teachers set may contain all the teachers of a school from all faculties. Entity sets need not be disjoint.

Attributes

- Entities are represented by means of their properties, called attributes. All attributes have values. For example, a student entity may have name, class, and age as attributes.

- There exists a domain or range of values that can be assigned to attributes. For example, a student's name cannot be a numeric value. It has to be alphabetic. A student's age cannot be negative, etc.

Types of Attributes

- Simple attribute – Simple attributes are atomic values, which cannot be divided further. For example, a student's phone number is an atomic value of 10 digits.

- Composite attribute – Composite attributes are made of more than one simple attribute. For example, a student's complete name may have first_name and last_name.

- Derived attribute – Derived attributes are the attributes that do not exist in the physical database, but their values are derived from other attributes present in the database. For example, average_salary in a department should not be saved directly in the database, instead it can be derived. For another example, age can be derived from data_of_birth.

ER-Model

- Single-value attribute − Single-value attributes contain single value. For example − Social_Security_Number.

- Multi-value attribute − Multi-value attributes may contain more than one values. For example, a person can have more than one phone number, email_address, etc.

ER-Model

- These attribute types can come together in a way like −

- simple single-valued attributes
- simple multi-valued attributes
- composite single-valued attributes
- composite multi-valued attributes

- Entity-Set and Keys
- Key is an attribute or collection of attributes that uniquely identifies an entity among entity set.
- For example, the roll number of a student makes him/her identifiable among students.
- **Super Key** − A set of attributes (one or more) that collectively identifies an entity in an entity set.
- **Candidate Key** − A minimal super key is called a candidate key. An entity set may have more than one candidate key.
- **Primary Key** − A primary key is one of the candidate keys chosen by the database designer to uniquely identify the entity set.

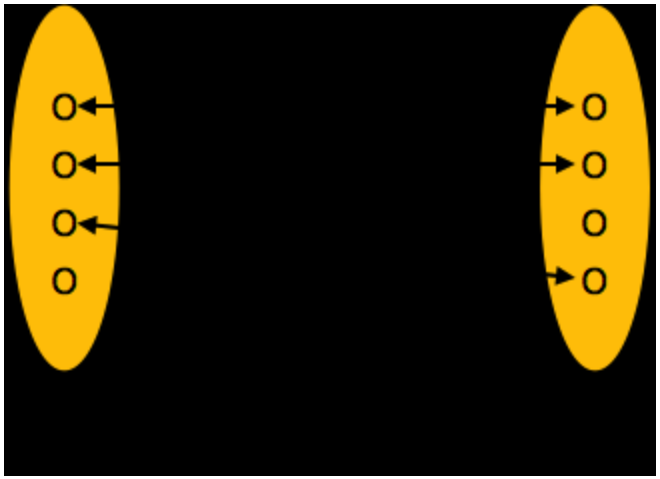
Relationship

- The association among entities is called a relationship. For example, an employee **works_at** a department, a student **enrolls** in a course. Here, Works_at and Enrolls are called relationships.
- Relationship Set
- A set of relationships of similar type is called a relationship set. Like entities, a relationship too can have attributes. These attributes are called **descriptive attributes**.
- Degree of Relationship
- The number of participating entities in a relationship defines the degree of the relationship.
- Binary = degree 2
- Ternary = degree 3
- n-ary = degree

Mapping Cardinalities

- Cardinality defines the number of entities in one entity set, which can be associated with the number of entities of other set via relationship set.

- One-to-one − One entity from entity set A can be associated with at most one entity of entity set B and vice versa.

- 

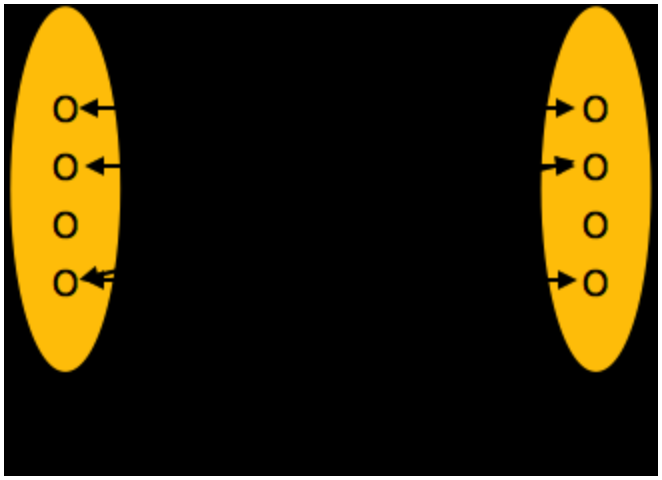
- One-to-many − One entity from entity set A can be associated with more than one entities of entity set B however an entity from entity set B, can be associated with at most one entity.

Many-to-one

- Many-to-one − More than one entities from entity set A can be associated with at most one entity of entity set B, however an entity from entity set B can be associated with more than one entity from entity set A.

Many-to-many

- Many-to-many – One entity from A can be associated with more than one entity from B and vice versa.

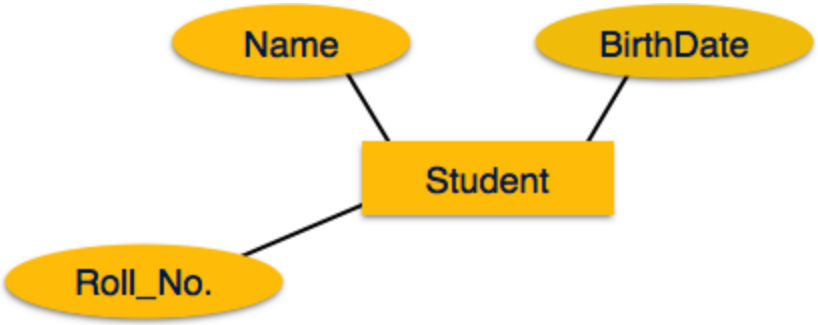


ER Diagram Representation

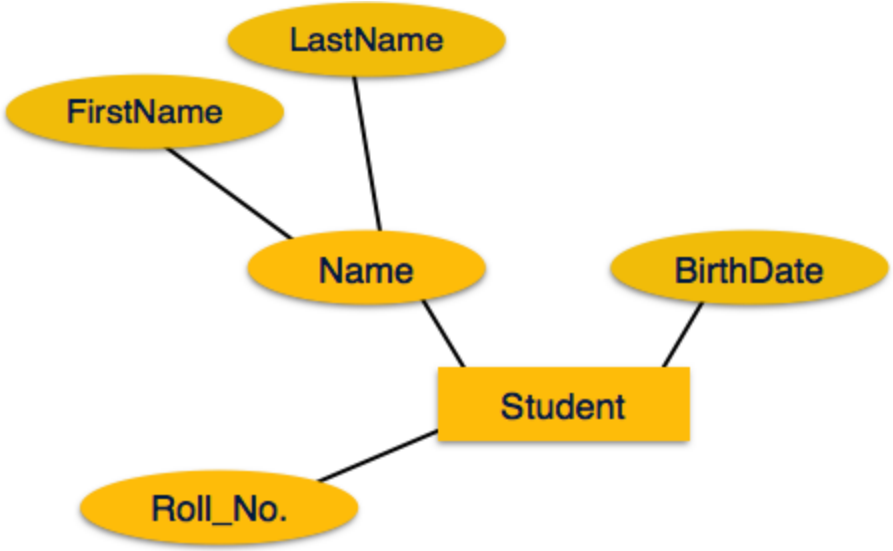
- Entity
- Entities are represented by means of rectangles. Rectangles are named with the entity set they represent.
-

Attributes

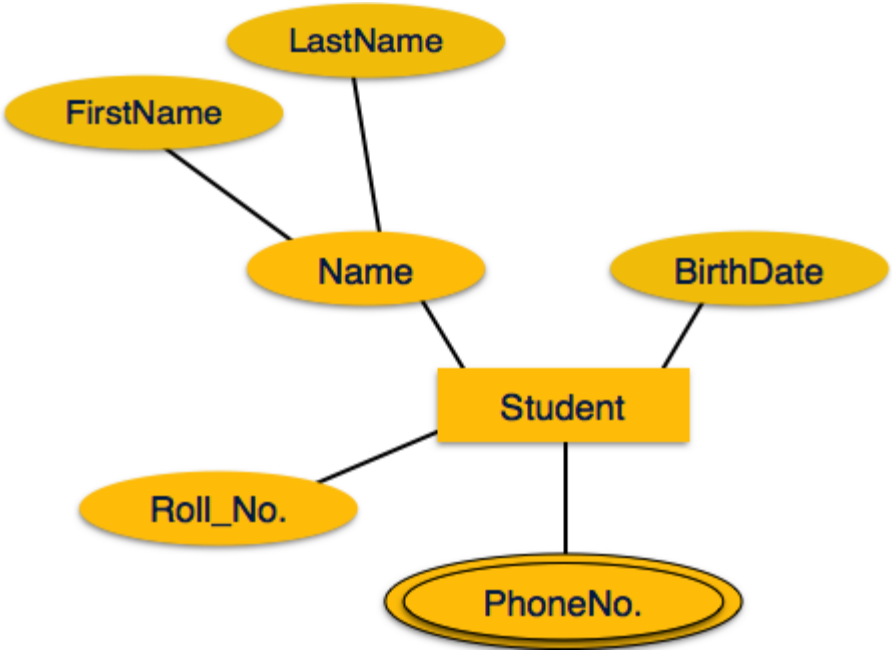
- Attributes are the properties of entities. Attributes are represented by means of ellipses. Every ellipse represents one attribute and is directly connected to its entity (rectangle).



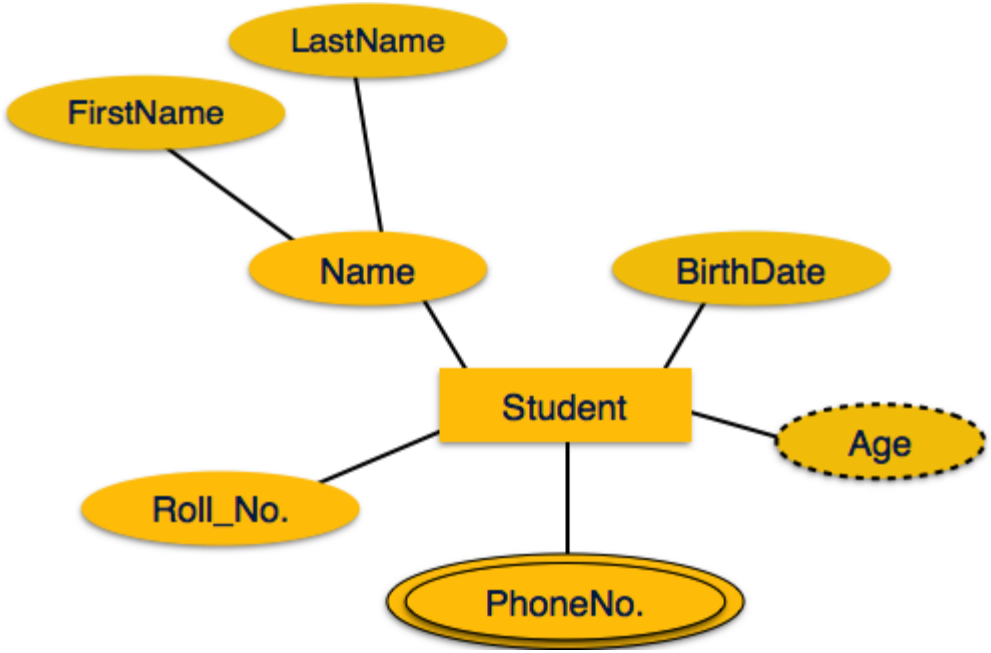
- If the attributes are composite, they are further divided in a tree like structure. Every node is then connected to its attribute. That is, composite attributes are represented by ellipses that are connected with an ellipse.



Multivalued attributes are depicted by double ellipse.



Derived attributes are depicted by dashed ellipse.



- Relationship
- Relationships are represented by diamond-shaped box. Name of the relationship is written inside the diamond-box. All the entities (rectangles) participating in a relationship, are connected to it by a line.

- Binary Relationship and Cardinality
- A relationship where two entities are participating is called a binary relationship. Cardinality is the number of instance of an entity from a relation that can be associated with the relation.

- One-to-one − When only one instance of an entity is associated with the relationship, it is marked as '1:1'. The following image reflects that only one instance of each entity should be associated with the relationship. It depicts one-to-one relationship.

- One-to-one

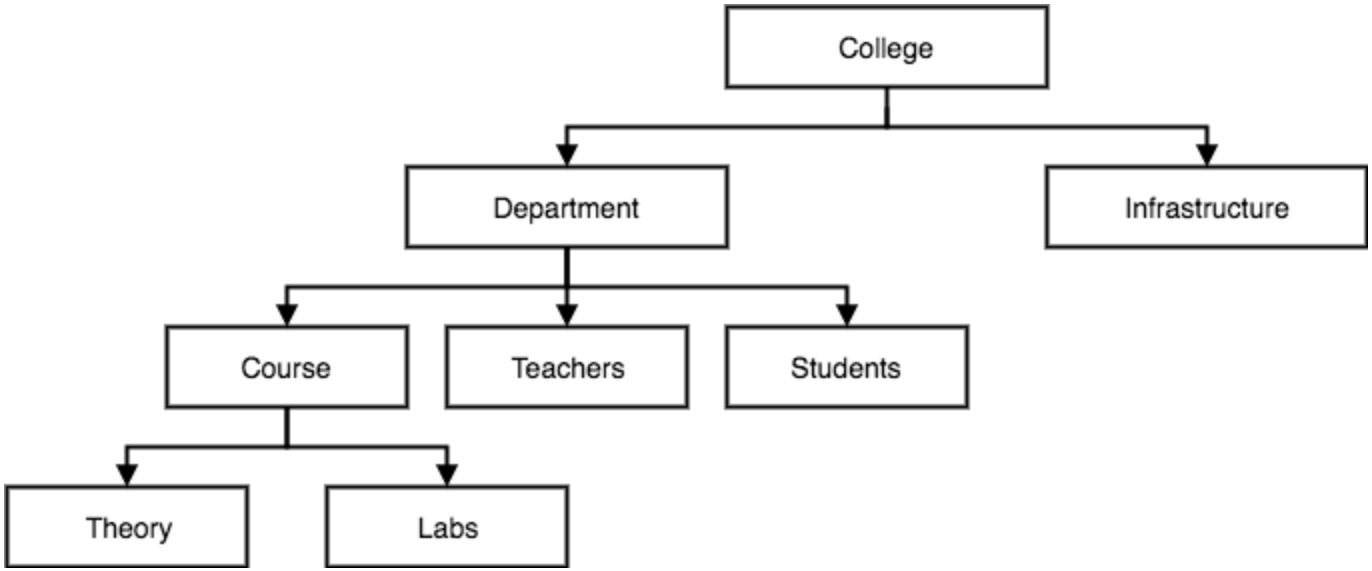
DBMS Database Models

- A Database model defines the logical design and structure of a database and defines how data will be stored, accessed and updated in a database management system. While the Relational Model is the most widely used database model, there are other models too:

- Hierarchical Model
- Network Model
- Entity-relationship Model
- Relational Model

Hierarchical Model

- This database model organizes data into a tree-like-structure, with a single root, to which all the other data is linked. The hierarchy starts from the Root data, and expands like a tree, adding child nodes to the parent nodes.
- In this model, a child node will only have a single parent node.
- This model efficiently describes many real-world relationships like index of a book, recipes etc.
- In hierarchical model, data is organised into tree-like structure with one one-to-many relationship between two different types of data, for example, one department can have many courses, many professors and of-course many students.


College
Department
Infrastructure
Course
Teachers
Students
Theory
Labs

Network Model

- This is an extension of the Hierarchical model. In this model data is organized more like a graph, and are allowed to have more than one parent node.

- In this database model data is more related as more relationships are established in this database model. Also, as the data is more related, hence accessing the data is also easier and fast. This database model was used to map many-to-many data relationships.

- This was the most widely used database model, before Relational Model was introduced.

Unit 2: File Structure and Organization

- What are files?
- File related keywords.
- File organization.
- File organizing methods.
- *heap file organization.
- *sequential files organization.
- *indexed file organization.
- *inverted file organization.
- *direct file organization.
- Comparison
- Quiz

FILES

- A file is a collection of data that is treated as a single unit on a peripheral device.
- TYPES OF FILES- MASTER FILE
- It contains record of permanent data types.

- They are created when you install your business.
- Work files
- a program can work efficiently if a work file is used.-a program can work efficiently if a
- work file is used

BASIC FILE

- Database: - It is a set of interrelated files. The files in combination tend to link to a common solution.
- For example,
- A student attendance file, a student result file, a student
- admission file, etc. are related to academic software pertaining to students.

- Record: - The elements related to are combined into a record. An employee has a record with his name, designation, basic pay, allowances, deductions etc. as its fields. A record may have a unique key to identify a record e.g. employee number. Records are represented as logical & physical records. A logical record maintains a logical
- relationship among all the data items in the record. It is the way the program or user sees the data. In contrast a physical record is the way data are recorded on a storage medium.

- Byte:- It is the smallest addressable unit in computer. A byte is a set of 8 bits and
- represents a character.

- Element:- It is a combination of one or more bytes. It is referred to as a field. A field is actually a physical space on tape or disk. A roll number, age, name of employee etc. are examples of it.

- File: - It is a collection of similar records. The records will have the same fields but
- different values in each record. The size of a file is limited by the size of memory
- available.

FILE ORGANIZATION

- It is the methodology which is applied to structured computer files. Files contain computer records which can be documents or information which is stored in a certain way for later retrieval.

- File organization refers primarily to the logical arrangement of data in a file system. It should not be confused with the physical storage of the file in some types of storage media.
- There are certain basic types of computer file, which can include files stored as blocks of data and streams of data, where the information streams out of the file while it is being read until the end of the file is encountered.

Methods of organizing files

Different methods of organizing files-

1.Heap
2.Sequential
3.Indexed-sequential
4.Inverted list
5.Direct access

- Choosing a file organization is a design decision, hence it must be done having in mind the achievement of good performance with respect to the most likely usage of the file. The criteria usually considered important are:

- Fast access to single record or collection of related records.
- Easy record adding/update/removal, without disrupting .
- Storage efficiency.
- Redundancy as a warranty against data corruption.

Heap files(unordered)

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

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

Advantages and disadvantages

Advantages

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

Disadvantages

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

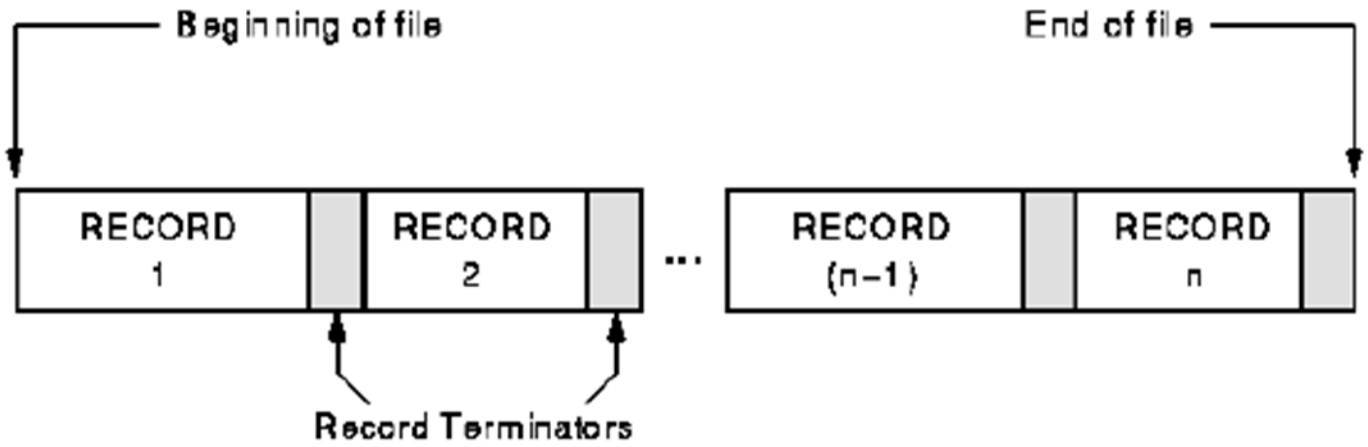
Heap file organization

- In the below figure, we can see a sample of heap file organization for EMPLOYEE relation which consists of 8 records stored in 3 contiguous blocks, each blocks can contains at most 3 records.

	Name	EID	Address	Birthdate	Salary
block 1	Adams John	….	…	…	…
	Melinda, Perkin				
	Raymond, Wong				
block 2	Alan, Delon				
	Bill, Clinton				
	Nancy, Davies				
block 3	Jay, Shana				
	Son, Nguyen				

Sequential file organization

- Sequential file organization
- Stored in key sequence.
- Adding/deleting requires making new file.
- Used as master file.
- Records in these files can only be read or written sequentially.



Sequential file organization

- Records are also in sequence within each block. To access a record, previous records within the block are scanned. Thus sequential record design is best suited for "get next" activities, reading one record after another without a search delay.

- records can be added only at the end of the file.

Advantages and disadvantages

- **ADVANTAGES**

- **Simple file design**
- **Very efficient when most of the records must be processed**
- **e.g. Payroll**
- **Very efficient if the data has a natural order**
- **Can be stored on inexpensive devices like magnetic tape.**

- **DISADVANTAGES**

- **Entire file must be processed even if a single record is to be searched.**
- **Transactions have to be sorted before processing**
- **Overall processing is slow.**

Indexed-sequential organization

- Each record of a file has a key field which uniquely identifies that record.
- An index consists of keys and addresses.
- An indexed sequential file is a sequential file (i.e. sorted into order of a key field) which has an index.
- A full index to a file is one in which there is an entry for every record.

- When a record is inserted or deleted in a file the data can be added at any location in the data file. Each index must also be updated to reflect the change. For a simple sequential index this may mean rewriting the
- index for each insertion.

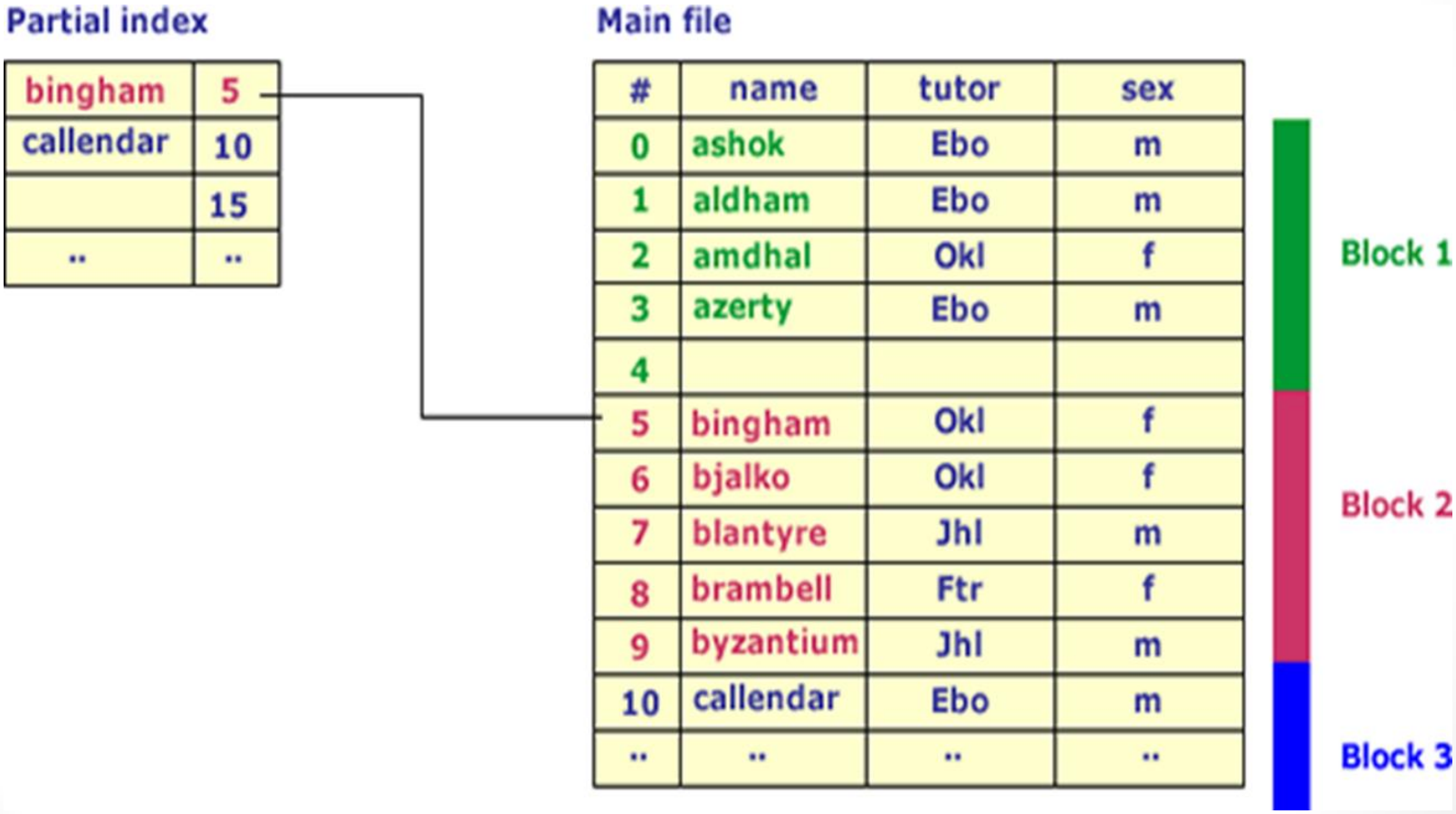
Indexed-sequential organization

Index to access data by department abbreviation.

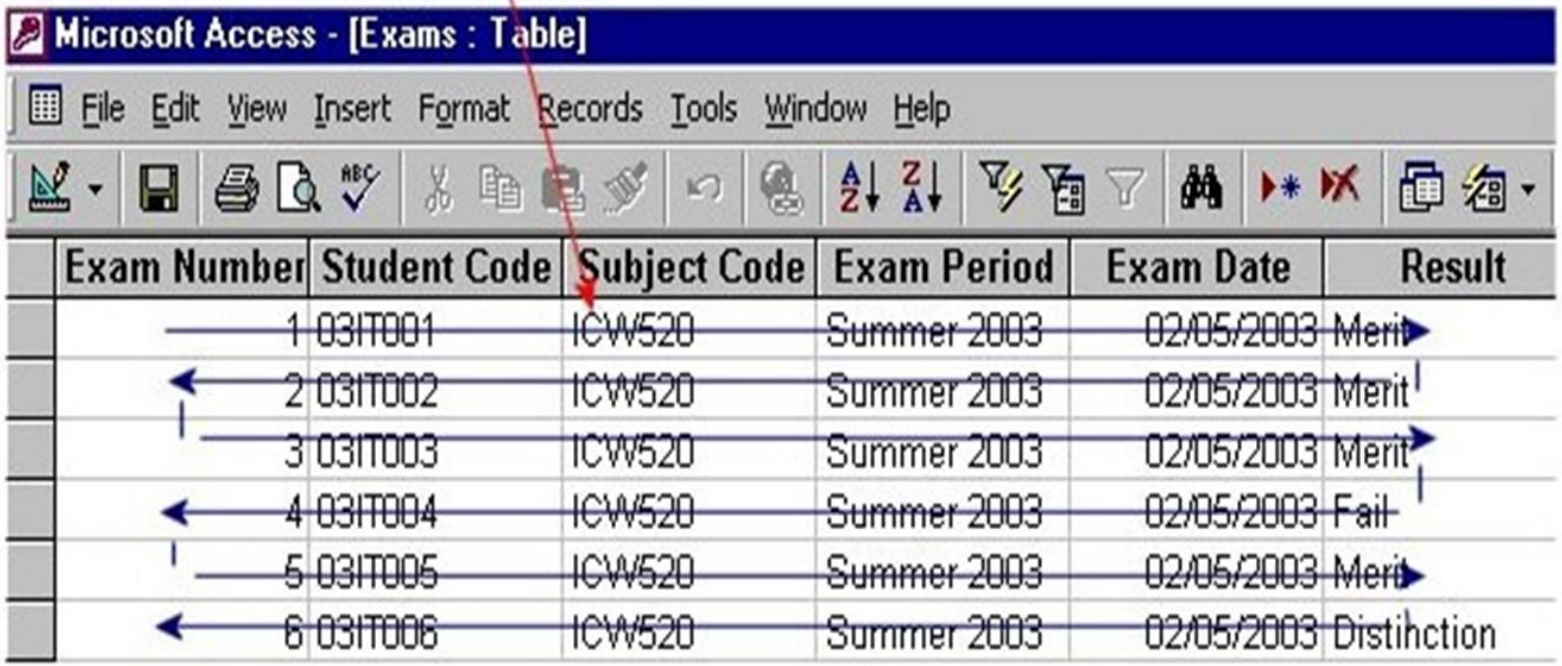
ACCT	00
ACCT	01
FIN	02
MGT	00
MGT	01
MGT	04
MKT	03

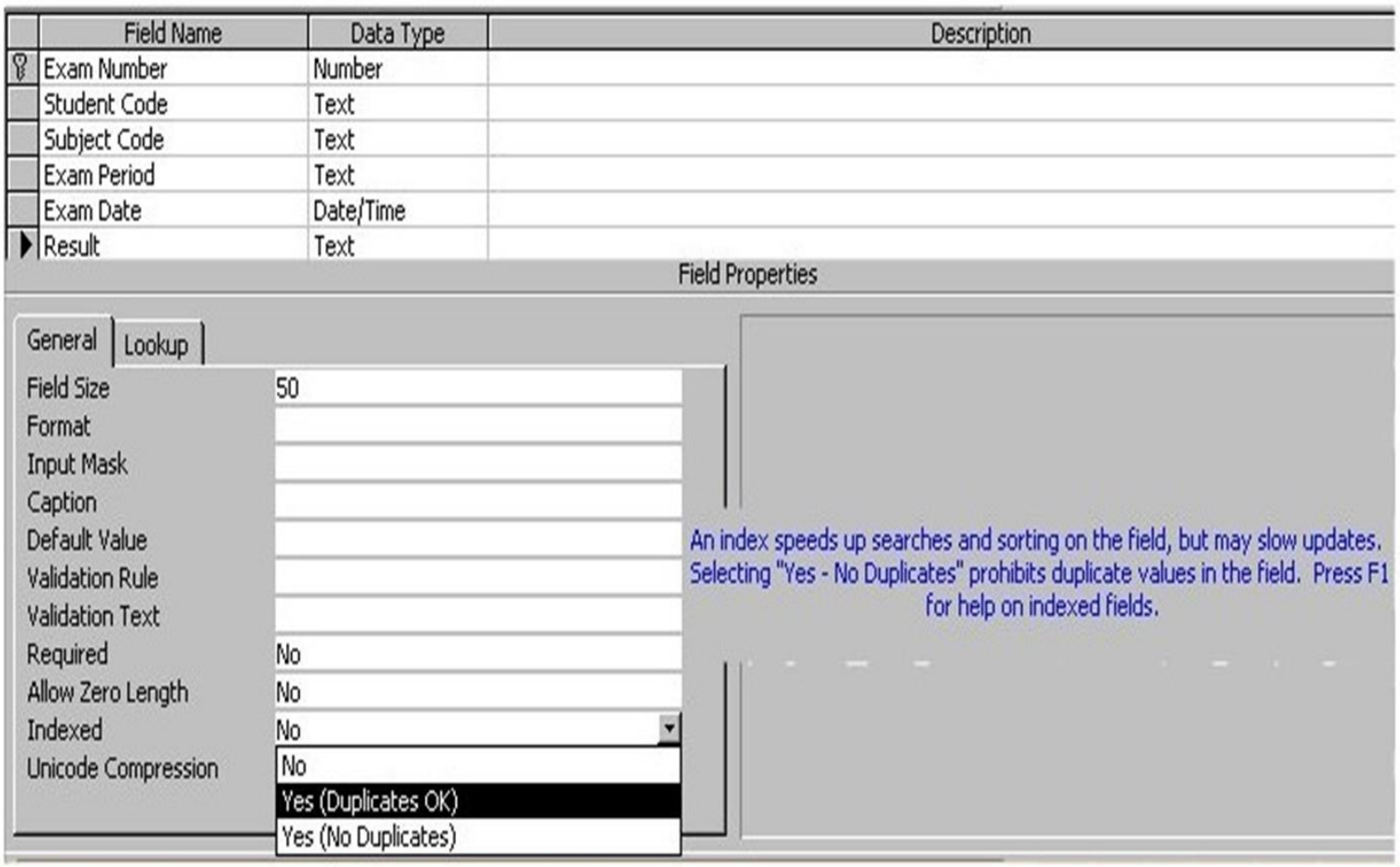
addr				
00	Price	MGT	Ayers	ACCT
01	Schwarzkopf	MGT	Daley	ACCT
02	Kenderdine	MKT	Linn	FIN
03	Lusch	MKT	Razook	MKT
04	Buckley	MGT	Dejoie	MGT

Indexed-sequential organization



When there is no index, a query searches through every field in the table record by record. This can be very slow when there are a lot of records stored in the table




Field Name
Data Type
Description
Exam Number
Number
Student Code
Text
Subject Code
Text
Exam Period
Text
Exam Date
Date/Time
Result
Text
Field Properties
General
Lookup
Field Size
50
Format
Input Mask
Caption
Default Value
Validation Rule
Validation Text
Required
No
Allow Zero Length
No
Indexed
No
Unicode Compression
No
Yes (Duplicates OK)
Yes (No Duplicates)
An index speeds up searches and sorting on the field, but may slow updates. Selecting "Yes - No Duplicates" prohibits duplicate values in the field. Press F1 for help on indexed fields.

Indexed-sequential organization

Indexed sequential files are important for applications where data needs to be accessed.....
Sequentially
randomly using the index.

An indexed sequential file can only be stored on a random access device
e.g. magnetic disc, CD.

ADVANTAGES AND DISADVANTAGES

Advantages

- **Provides flexibility for users who need both type of accesses with the same file.**
- **Faster than sequential.**

Disadvantages

- **Extra storage space for the index is required**

Inverted list organization

- Like the indexed-sequential storage method, the inverted list organization maintains an index. The two methods differ, however, in the index level and record storage. The indexed-sequential method has a multiple index for a given key, whereas
- the inverted list method has a single index for each key type.
- The records are not necessarily stored in a sequence. They are placed in the are data storage area, but indexes are updated for the record keys and location.

ADVANTAGES AND DISADVANTAGES

- Advantages
- The benefits are apparent immediately because searching is fast

- disadvantages
- inverted list files use more media space and the storage devices get full quickly with this type of organization.
- updating is much slower.

Direct/random file organization

- **Records are read directly from or written on to the file.**
- **The records are stored at known address.**
- **Address is calculated by applying a mathematical function to the key field.**
- **A random file would have to be stored on a direct access backing storage medium e.g. magnetic disc, CD, DVD**

- **Example : Any information retrieval system. Eg Train timetable system.**

Advantages and disadvantages

Advantages

- Any record can be directly accessed.
- Speed of record processing is very fast.
- Up-to-date file because of online updating.
- Concurrent processing is possible.
- Transactions need not be sorted.

Disadvantages

- More complex than sequential.
- Does not fully use memory locations.
- More security and backup problems.
- Expensive hardware and software are required.
- System design is complex and costly.
- File updation is more difficult as compared to sequential files.

Comparison

Comparison of File Designs			
	Sequential	**Direct-Access**	**Indexed-Sequential**
Types of Access	batch	online	batch or online
Data Organization	sequentially by key value	no particular order	sequentially and by index
Flexibility in Handling Inquiries	low	high	very high
Availability of Up-to-Date Data	no	yes	yes
Speed of Retrieval	slow	very fast	fast
Activity	high	low	high
Volatility	low	high	high
Examples	payroll processing and billing operations	airline reservations and banking transactions	customer ordering and billing

Quiz

- What is file organization?
- What are advantages of sequential file organization?
- True or false (indexed sequential file)
- The data can be added at any location in the file.
- Give an example of direct file organization?
- Give one advantage and disadvantage of direct file organization?

Basic Concepts Basic Concepts

- Indexing mechanisms used to speed up access to desired data.
- E.g., author catalog in library
- Search Key - attribute to set of attributes used to look up records in a file.
- An index file consists of records (called index entries) of the form
- Index files are typically much smaller than the original file

Two basic kinds of indices:

- Ordered indices: search keys are stored in sorted order
- Hash indices: search keys are distributed uniformly across "buckets" using a "hash function".

- Indexing mechanisms used to speed up access to desired data.

E.g., author catalog in library

- Search Key - attribute to set of attributes used to look up
- records in a file.
- An index file consists of records (called index entries) of the
- form
- Index files are typically much smaller than the original file
- Two basic kinds of indices:
- ¾ Ordered indices: search keys are stored in sorted order
- ¾ Hash indices: search keys are distributed uniformly across
- “buckets” using a “hash function”.

Ordered Indices Ordered Indices

- In an ordered index, index entries are stored sorted on the search key value. E.g., author catalog in library.
- Primary index: in a sequentially ordered file, the index whose
- search key specifies the sequential order of the file. Also called clustering index
- The search key of a primary index is usually but not necessarily the primary key.
- Secondary index: an index whose search key specifies an order
- different from the sequential order of the file. Also called non-clustering index.
- Index-sequential file: ordered sequential file with a primary index

Index Evaluation Metrics

- Access types supported efficiently. E.g.,
- records with a specified value in the attribute
- or records with an attribute value falling in a specified range of values.
- Access time
- Insertion time
- Deletion time
- Space overhead

- Ordered Indices Ordered Indices
- In an ordered index, index entries are stored sorted on the
- search key value. E.g., author catalog in library.
- Primary index: in a sequentially ordered file, the index whose
- search key specifies the sequential order of the file.
- ¾ Also called clustering index
- ¾ The search key of a primary index is usually but not necessarily the
- primary key.
- Secondary index: an index whose search key specifies an order
- different from the sequential order of the file. Also called
- non-clustering index.
- Index-sequential file: ordered sequential file with a primary index.

Dense Index Files

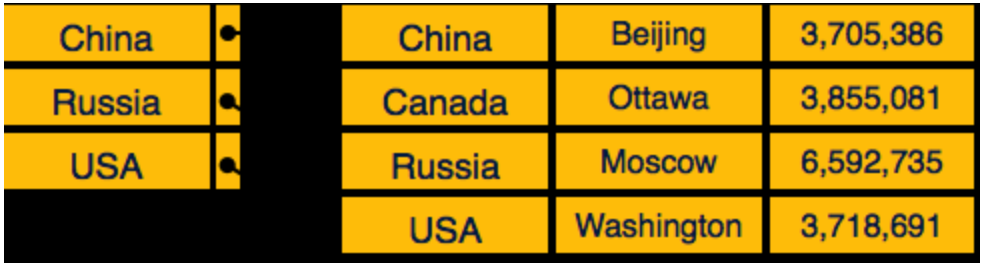
- Dense index — Index record appears for every search-key value in the file

Dense Index

In dense index, there is an index record for every search key value in the database. This makes searching faster but requires more space to store index records itself. Index records contain search key value and a pointer to the actual record on the disk.

Sparse Index

- Sparse Index
- In sparse index, index records are not created for every search key. An index record here contains a search key and an actual pointer to the data on the disk. To search a record, we first proceed by index record and reach at the actual location of the data. If the data we are looking for is not where we directly reach by following the index, then the system starts sequential search until the desired data is found.



Clustered index

- What is a Clustered index?
- Cluster index is a type of index which sorts the data rows in the table on their key values. In the Database, there is only one clustered index per table.
- A clustered index defines the order in which data is stored in the table which can be sorted in only one way. So, there can be an only a single clustered index for every table. In an RDBMS, usually, the primary key allows you to create a clustered index based on that specific column.

What is Non-clustered index?

- A Non-clustered index stores the data at one location and indices at another location. The index contains pointers to the location of that data. A single table can have many non-clustered indexes as an index in the non-clustered index is stored in different places.
- For example, a book can have more than one index, one at the beginning which displays the contents of a book unit wise while the second index shows the index of terms in alphabetical order.

- A non-clustering index is defined in the non-ordering field of the table. This type of indexing method helps you to improve the performance of queries that use keys which are not assigned as a primary key. A non-clustered index allows you to add a unique key for a table.

Characteristic of Clustered Index

- Default and sorted data storage
- Use just one or more than one columns for an index
- Helps you to store Data and index together
- Fragmentation
- Operations
- Clustered index scan and index seek
- Key Lookup

Multilevel Index

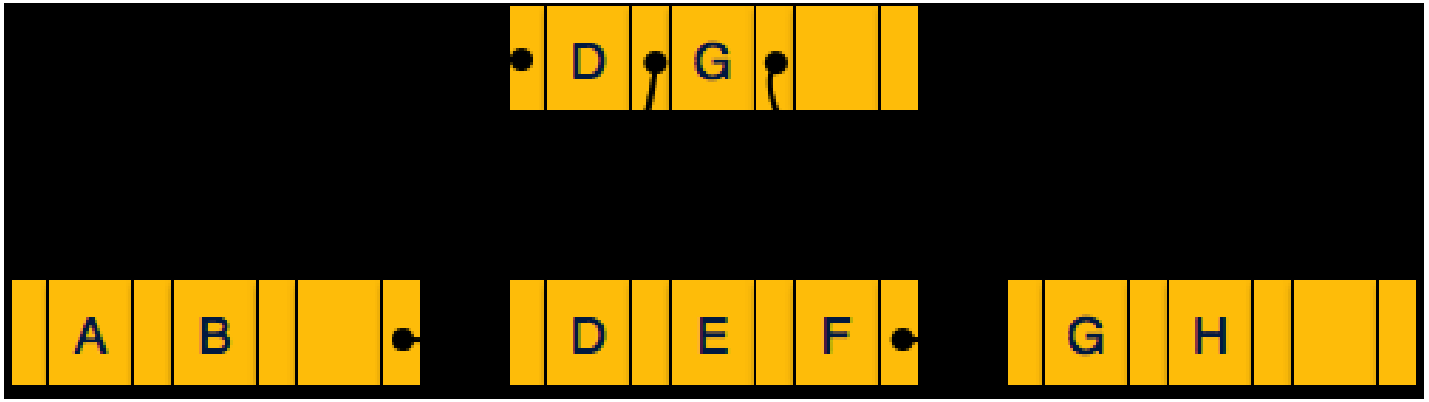
- Index records comprise search-key values and data pointers. Multilevel index is stored on the disk along with the actual database files. As the size of the database grows, so does the size of the indices. There is an immense need to keep the index records in the main memory so as to speed up the search operations. If single-level index is used, then a large size index cannot be kept in memory which leads to multiple disk accesses.

- Multi-level Index helps in breaking down the index into several smaller indices in order to make the outermost level so small that it can be saved in a single disk block, which can easily be accommodated anywhere in the main memory.

B+ Tree

- A B+ tree is a balanced binary search tree that follows a multi-level index format. The leaf nodes of a B+ tree denote actual data pointers. B+ tree ensures that all leaf nodes remain at the same height, thus balanced. Additionally, the leaf nodes are linked using a link list; therefore, a B+ tree can support random access as well as sequential access.

- Structure of B+ Tree
- Every leaf node is at equal distance from the root node. A B+ tree is of the order n where n is fixed for every B+ tree.


D
G
A
B
D
E
F
G
H

- Internal nodes −

- Internal (non-leaf) nodes contain at least [n/2] pointers, except the root node.
- At most, an internal node can contain n pointers.
- Leaf nodes −

- Leaf nodes contain at least [n/2] record pointers and [n/2] key values.
- At most, a leaf node can contain n record pointers and n key values.
- Every leaf node contains one block pointer P to point to next leaf node and forms a linked list.

B+ Tree Insertion

- B+ trees are filled from bottom and each entry is done at the leaf node.
- If a leaf node overflows −
- Split node into two parts.
- Partition at i = ⌊(m+1)/2⌋.
- First i entries are stored in one node.
- Rest of the entries (i+1 onwards) are moved to a new node.
- ith key is duplicated at the parent of the leaf.
- If a non-leaf node overflows −
- Split node into two parts.
- Partition the node at i = ⌈(m+1)/2⌉.
- Entries up to i are kept in one node.
- Rest of the entries are moved to a new node.

B+ Tree Insertion

- B+ Tree Deletion
- B+ tree entries are deleted at the leaf nodes.
- The target entry is searched and deleted.
- If it is an internal node, delete and replace with the entry from the left position.
- After deletion, underflow is tested,
- If underflow occurs, distribute the entries from the nodes left to it.
- If distribution is not possible from left, then
- Distribute from the nodes right to it.
- If distribution is not possible from left or from right, then
- Merge the node with left and right to it.

- Static Hashing
- In static hashing, when a search-key value is provided, the hash function always computes the same address. For example, if mod-4 hash function is used, then it shall generate only 5 values. The output address shall always be same for that function. The number of buckets provided remains unchanged at all times.

- Operation
- Insertion − When a record is required to be entered using static hash, the hash function h computes the bucket address for search key K, where the record will be stored.
- Bucket address = h(K)
- Search − When a record needs to be retrieved, the same hash function can be used to retrieve the address of the bucket where the data is stored.
- Delete − This is simply a search followed by a deletion operation.
- Bucket Overflow

- The condition of bucket-overflow is known as collision. This is a fatal state for any static hash function. In this case, overflow chaining can be used.

- Overflow Chaining − When buckets are full, a new bucket is allocated for the same hash result and is linked after the previous one. This mechanism is called Closed Hashing.

- Overflow chaining
- Linear Probing − When a hash function generates an address at which data is already stored, the next free bucket is allocated to it. This mechanism is called Open Hashing.

- Linear Probing
- Dynamic Hashing
- The problem with static hashing is that it does not expand or shrink dynamically as the size of the database grows or shrinks. Dynamic hashing provides a mechanism in which data buckets are added and removed dynamically and on-demand. Dynamic hashing is also known as extended hashing.

- Hash function, in dynamic hashing, is made to produce a large number of values and only a few are used initially.

- Dynamic Hashing

- Hash Organization
- **Bucket** − A hash file stores data in bucket format. Bucket is considered a unit of storage. A bucket typically stores one complete disk block, which in turn can store one or more records.
- **Hash Function** − A hash function, **h,** is a mapping function that maps all the set of search-keys **K** to the address where actual records are placed. It is a function from search keys to bucket addresses.

- Static Hashing
- In static hashing, when a search-key value is provided, the hash function always computes the same address. For example, if mod-4 hash function is used, then it shall generate only 5 values. The output address shall always be same for that function. The number of buckets provided remains unchanged at all times.

- Organization
- The prefix of an entire hash value is taken as a hash index. Only a portion of the hash value is used for computing bucket addresses. Every hash index has a depth value to signify how many bits are used for computing a hash function. These bits can address 2n buckets. When all these bits are consumed – that is, when all the buckets are full – then the depth value is increased linearly and twice the buckets are allocated.

Secondary Index

- In the sparse indexing, as the size of the table grows, the size of mapping also grows. These mappings are usually kept in the primary memory so that address fetch should be faster. Then the secondary memory searches the actual data based on the address got from mapping. If the mapping size grows then fetching the address itself becomes slower. In this case, the sparse index will not be efficient. To overcome this problem, secondary indexing is introduced.

- For example:

- If you want to find the record of roll 111 in the diagram, then it will search the highest entry which is smaller than or equal to 111 in the first level index. It will get 100 at this level.
- Then in the second index level, again it does max (111) <= 111 and gets 110. Now using the address 110, it goes to the data block and starts searching each record till it gets 111.
- This is how a search is performed in this method. Inserting, updating or deleting is also done in the same manner.

Unit 3:Relational Model

Introduction

- Relational Model was proposed by E.F. Codd to model data in the form of relations or tables. After designing the conceptual model of Database using ER diagram, we need to convert the conceptual model in the relational model which can be implemented using any RDMBS languages like Oracle SQL, MySQL etc. So we will see what Relational Model is.
- **What is Relational Model?**
- Relational Model represents how data is stored in Relational Databases. A relational database stores data in the form of relations (tables). Consider a relation STUDENT with attributes ROLL_NO, NAME, ADDRESS, PHONE and AGE shown in Table 1.

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
2	RAMESH	GURGAON	9652431543	18
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	DELHI		18

Attribute: Attributes are the properties that define a relation. e.g.; ROLL_NO, NAME

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. If a schema has more than 1 relation, it is called Relational Schema.

Tuple: Each row in the relation is known as tuple. The above relation contains 4 tuples, one of which is shown as:

1	RAM	DELHI	9455123451	18

4. **Relation Instance:** The set of tuples of a relation at a particular instance of time is called as relation instance. Table 1 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 **STUDENT** relation defined above has degree 5.
6. **Cardinality:** The number of tuples in a relation is known as cardinality. The **STUDENT** relation defined above has cardinality 4.
7. **Column:** Column represents the set of values for a particular attribute. The column **ROLL_NO** is extracted from relation STUDENT.

- ROLL_NO
- 1
- 2
- 3
- 4
- NULL Values: 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 4 is NULL.

Constraints in Relational Model

- While designing Relational Model, we define some conditions which must hold for data present in database are called Constraints. These constraints are checked before performing any operation (insertion, deletion and updation) in database. If there is a violation in any of constrains, operation will fail.
- **Domain Constraints:** These are attribute level constraints. An attribute can only take values which lie inside the domain range. e.g,; If a constrains AGE>0 is applied on STUDENT relation, inserting negative value of AGE will result in failure.

- **Key Integrity:** Every relation in the database should have atleast one set of attributes which defines a tuple uniquely. Those set of attributes is called key. e.g.; ROLL_NO in STUDENT is a key. No two students can have same roll number. So a key has two properties:
- It should be unique for all tuples.
- It can’t have NULL values.
- **Referential Integrity:** When one attribute of a relation can only take values from other attribute of same relation or any other relation, it is called referential integrity. Let us suppose we have 2 relations

STUDENT

ROLL_NO	NAME	ADDRESS	PHONE	AGE	BRANCH_CODE
1	RAM	DELHI	9455123451	18	CS
2	RAMESH	GURGAON	9652431543	18	CS
3	SUJIT	ROHTAK	9156253131	20	ECE
4	SURESH	DELHI		18	IT

BRANCH

BRANCH_CODE	BRANCH_NAME
CS	COMPUTER SCIENCE
IT	INFORMATION TECHNOLOGY
ECE	ELECTRONICS AND COMMUNICATION ENGINEERING
CV	CIVIL ENGINEERING

- BRANCH_CODE of STUDENT can only take the values which are present in BRANCH_CODE of BRANCH which is called referential integrity constraint. The relation which is referencing to other relation is called REFERENCING RELATION (STUDENT in this case) and the relation to which other relations refer is called REFERENCED RELATION (BRANCH in this case).

Relational Algebra

- Relational algebra is a procedural query language, which takes instances of relations as input and yields instances of relations as output. It uses operators to perform queries. An operator can be either **unary** or **binary**. They accept relations as their input and yield relations as their output. Relational algebra is performed recursively on a relation and intermediate results are also considered relations.

The fundamental operations

- The fundamental operations of relational algebra are as follows −
- Select
- Project
- Union
- Set different
- Cartesian product
- Rename

Select Operation (σ)

- It selects tuples that satisfy the given predicate from a relation.
- **Notation** − $\sigma_p(r)$
- Where **σ** stands for selection predicate and **r** stands for relation. *p* is prepositional logic formula which may use connectors like **and, or,** and **not**. These terms may use relational operators like − =, ≠, ≥, < , >, ≤.
- **For example** −
- $\sigma_{subject = "database"}(Books)$ **Output** − Selects tuples from books where subject is 'database'.
- $\sigma_{subject = "database" \text{ and } price = "450"}(Books)$ **Output** − Selects tuples from books where subject is 'database' and 'price' is 450.
- $\sigma_{subject = "database" \text{ and } price = "450" \text{ or } year > "2010"}(Books)$ **Output** − Selects tuples from books where subject is 'database' and 'price' is 450 or those books published after 2010.

Project Operation (∏)

- It projects column(s) that satisfy a given predicate.
- Notation − ∏A1, A2, An (r)
- Where A1, A2 , An are attribute names of relation r.
- Duplicate rows are automatically eliminated, as relation is a set.
- For example −
- ∏subject, author (Books)
- Selects and projects columns named as subject and author from the relation Books.

Union Operation (∪)

- It performs binary union between two given relations and is defined as −

- r ∪ s = { t | t ∈ r or t ∈ s}
- Notation − r U s

- Where r and s are either database relations or relation result set (temporary relation).

- For a union operation to be valid, the following conditions must hold −

- r, and s must have the same number of attributes.
- Attribute domains must be compatible.
- Duplicate tuples are automatically eliminated.
- ∏ author (Books) ∪ ∏ author (Articles)
- Output − Projects the names of the authors who have either written a book or an article or both.

Set Difference (–)

- The result of set difference operation is tuples, which are present in one relation but are not in the second relation.
- Notation – r – s
- Finds all the tuples that are present in r but not in s.
- ∏ author (Books) – ∏ author (Articles)
- Output – Provides the name of authors who have written books but not articles.

Cartesian Product (X)

- Combines information of two different relations into one.
- Notation – r X s
- Where r and s are relations and their output will be defined as –
- r X s = { q t | q ∈ r and t ∈ s}
- σauthor = 'tutorialspoint'(Books X Articles)
- Output – Yields a relation, which shows all the books and articles written by tutorialspoint.

Rename Operation (ρ)

- The results of relational algebra are also relations but without any name. The rename operation allows us to rename the output relation. 'rename' operation is denoted with small Greek letter rho ρ.
- Notation – ρ x (E)
- Where the result of expression E is saved with name of x.
- Additional operations are –
- Set intersection
- Assignment
- Natural join

Types of Keys

Different Types of Keys in Relational Model

STUDENT

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY	STUD_AGE
1	RAM	9716271721	Haryana	India	20
2	RAM	9898291281	Punjab	India	19
3	SUJIT	7898291981	Rajsthan	India	18
4	SURESH		Punjab	India	21

Table 1

STUDENT_COURSE

STUD_NO	COURSE_NO	COURSE_NAME
1	C1	DBMS
2	C2	Computer Networks
1	C2	Computer Networks

Table 2

Candidate Key:

- The minimal set of attribute which can uniquely identify a tuple is known as candidate key. For Example, STUD_NO in STUDENT relation.

- The value of Candidate Key is unique and non-null for every tuple.
- There can be more than one candidate key in a relation. For Example, STUD_NO is candidate key for relation STUDENT.
- The candidate key can be simple (having only one attribute) or composite as well. For Example, {STUD_NO, COURSE_NO} is a composite candidate key for relation STUDENT_COURSE.

Super Key

- Super Key: The set of attributes which can uniquely identify a tuple is known as Super Key. For Example, STUD_NO, (STUD_NO, STUD_NAME) etc.

- Adding zero or more attributes to candidate key generates super key.
- A candidate key is a super key but vice versa is not true.

Primary Key

- Primary Key: There can be more than one candidate key in relation out of which one can be chosen as the primary key. For Example, STUD_NO, as well as STUD_PHONE both, are candidate keys for relation STUDENT but STUD_NO can be chosen as the primary key (only one out of many candidate keys).

Alternate Key

- **Alternate Key:** The candidate key other than the primary key is called an alternate key. For Example, STUD_NO, as well as STUD_PHONE both, are candidate keys for relation STUDENT but STUD_PHONE will be alternate key (only one out of many candidate keys).

Foreign Key

- Foreign Key: If an attribute can only take the values which are present as values of some other attribute, it will be a foreign key to the attribute to which it refers. The relation which is being referenced is called referenced relation and the corresponding attribute is called referenced attribute and the relation which refers to the referenced relation is called referencing relation and the corresponding attribute is called referencing attribute. The referenced attribute of the referenced relation should be the primary key for it. For Example, STUD_NO in STUDENT_COURSE is a foreign key to STUD_NO in STUDENT relation.

- For Example, STUD_NO in STUDENT_COURSE relation is not unique. It has been repeated for the first and third tuple. However, the STUD_NO in STUDENT relation is a primary key and it needs to be always unique and it cannot be null.

Super Key:

- 1. Super Key is an attribute (or set of attributes) that is used to uniquely identifies all attributes in a relation. All super keys can't be candidate keys but its reverse is true. In a relation, number of super keys are more than number of candidate keys.

- Example:
- We have a given relation R(A, B, C, D, E, F) and we shall check for being super keys by following given dependencies:

- Functional dependencies Super key
- AB->CDEF YES
- CD->ABEF YES
- CB->DF NO
- D->BC NO
- By Using key AB we can identify rest of the attributes (CDEF) of the table. Similarly Key CD. But, by using key CB we can only identifies D and F not A and E. Similarly key D.

SQL JOIN

- A JOIN clause is used to combine rows from two or more tables, based on a related column between them.
- Let's look at a selection from the "Orders" table:

OrderID	CustomerID	OrderDate
10308	2	1996-09-18
10309	37	1996-09-19
10310	77	1996-09-20

Then, look at a selection from the "Customers" table:
Notice that the "CustomerID" column in the "Orders" table refers to the "CustomerID" in the "Customers" table. The relationship between the two tables above is the "CustomerID" column.
Then, we can create the following SQL statement (that contains an INNER JOIN), that selects records that have matching values in both tables:

CustomerID	CustomerName	ContactName	Country
1	Alfreds Futterkiste	Maria Anders	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mexico

- Example
- ```
  SELECT Orders.OrderID, Customers.CustomerName,
  Orders.OrderDate
  FROM Orders
  INNER JOIN Customers ON Orders.CustomerID=Cust
  omers.CustomerID;
  ```
-
  ```

- Different Types of SQL JOINs
- Here are the different types of the JOINs in SQL:
- **(INNER) JOIN**: Returns records that have matching values in both tables
- **LEFT (OUTER) JOIN**: Returns all records from the left table, and the matched records from the right table
- **RIGHT (OUTER) JOIN**: Returns all records from the right table, and the matched records from the left table
- **FULL (OUTER) JOIN**: Returns all records when there is a match in either left or right table
-

INNER JOIN Syntax

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

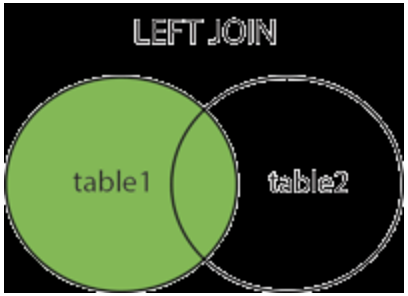
SQL LEFT JOIN Keyword

- The LEFT JOIN keyword returns all records from the left table (table1), and the matched records from the right table (table2). The result is NULL from the right side, if there is no match.

- LEFT JOIN Syntax
- ```
  SELECT column_name(s)
  FROM table1
  LEFT JOIN table2
  ON table1.column_name = table2.column_name;
  ```
  ```

SQL LEFT JOIN Keyword

- **Note:** In some databases LEFT JOIN is called LEFT OUTER JOIN.



SQL LEFT JOIN Example

- SQL LEFT JOIN Example
- The following SQL statement will select all customers, and any orders they might have:
- Example
- ```
  SELECT Customers.CustomerName, Orders.OrderID
  FROM Customers
  LEFT JOIN Orders ON Customers.CustomerID =
  Orders.CustomerID
  ORDER BY Customers.CustomerName;
  ```
  ```

SQL RIGHT JOIN Keyword

- SQL RIGHT JOIN Keyword
- The RIGHT JOIN keyword returns all records from the right table (table2), and the matched records from the left table (table1). The result is NULL from the left side, when there is no match.
- RIGHT JOIN Syntax
- ```
  SELECT column_name(s)
  FROM table1
  RIGHT JOIN table2
  ON table1.column_name = table2.column_name;
  ```
- **Note:** In some databases RIGHT JOIN is called RIGHT OUTER JOIN.
  ```

- SQL RIGHT JOIN Example
- The following SQL statement will return all employees, and any orders they might have placed:
- Example
- ```
  SELECT Orders.OrderID, Employees.LastName,
  Employees.FirstName
  FROM Orders
  RIGHT JOIN Employees ON Orders.EmployeeID =
  Employees.EmployeeID
  ORDER BY Orders.OrderID;
  ```
-
  ```

SQL FULL OUTER JOIN Keyword

- The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.
- **Note:** FULL OUTER JOIN can potentially return very large result-sets!
- **Tip:** FULL OUTER JOIN and FULL JOIN are the same.
- FULL OUTER JOIN Syntax
- ```
  SELECT column_name(s)
  FROM table1
  FULL OUTER JOIN table2
  ON table1.column_name = table2.column_name
  WHERE condition;
  ```
  ```

SQL FULL OUTER JOIN Example

- The following SQL statement selects all customers, and all orders:
- ```
  SELECT Customers.CustomerName, Orders.OrderID
  FROM Customers
  FULL OUTER JOIN Orders ON Customers.CustomerID
  =Orders.CustomerID
  ORDER BY Customers.CustomerName;
  ```
  ```

SQL Self JOIN

- A self JOIN is a regular join, but the table is joined with itself.
- Self JOIN Syntax
- ```
  SELECT column_name(s)
  FROM table1 T1, table1 T2
  WHERE condition;
  ```
  ```

SQL Self JOIN Example

- The following SQL statement matches customers that are from the same city:
- Example
- ```
  SELECT A.CustomerName AS CustomerName1,
  B.CustomerName AS CustomerName2, A.City
  FROM Customers A, Customers B
  WHERE A.CustomerID <> B.CustomerID
  AND A.City = B.City
  ORDER BY A.City;
  ```
  ```

- Concurrency Control deals with **interleaved execution** of more than one transaction. In the next article, we will see what is serializability and how to find whether a schedule is serializable or not.
- **What is Transaction?**
- A set of logically related operations is known as transaction. The main operations of a transaction are:
- **Read(A):** Read operations Read(A) or R(A) reads the value of A from the database and stores it in a buffer in main memory.
- **Write (A):** Write operation Write(A) or W(A) writes the value back to the database from buffer.

Unit 4 : SQL

Introduction

- What is SQL?
- SQL stands for Structured Query Language
- SQL lets you access and manipulate databases
- SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987

What Can SQL do?

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database
- SQL can create views in a database
- SQL can set permissions on tables, procedures, and views

A Brief History of SQL

- 1970 – Dr. Edgar F. "Ted" Codd of IBM is known as the father of relational databases. He described a relational model for databases.
- 1974 – Structured Query Language appeared.
- 1978 – IBM worked to develop Codd's ideas and released a product named System/R.
- 1986 – IBM developed the first prototype of relational database and standardized by ANSI. The first relational database was released by Relational Software which later came to be known as Oracle.

SQL Process

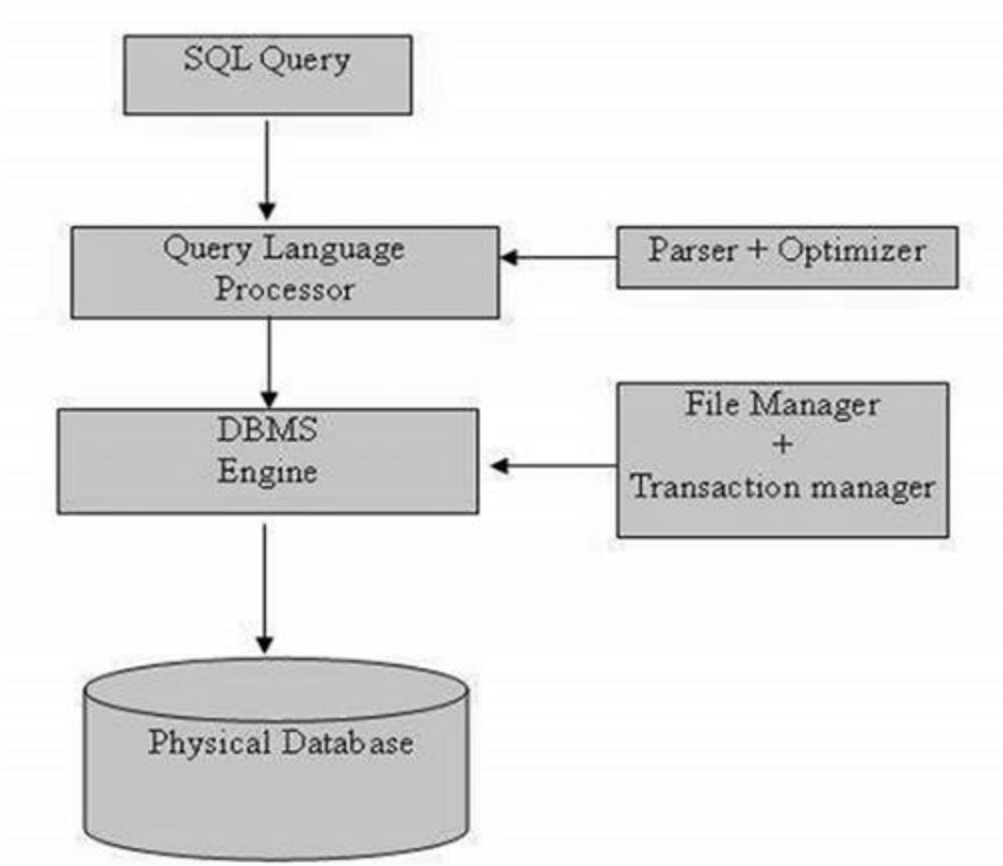
- When you are executing an SQL command for any RDBMS, the system determines the best way to carry out your request and SQL engine figures out how to interpret the task.

- There are various components included in this process.

- These components are −

- Query Dispatcher
- Optimization Engines
- Classic Query Engine
- SQL Query Engine, etc.
- A classic query engine handles all the non-SQL queries, but a SQL query engine won't handle logical files.

Following is a simple diagram showing the SQL Architecture –



SQL Commands

- The standard SQL commands to interact with relational databases are CREATE, SELECT, INSERT, UPDATE, DELETE and DROP. These commands can be classified into the following groups based on their nature −
- DDL - Data Definition Language

DDL - Data Definition Language

Sr.No.	Command & Description
1	**CREATE** Creates a new table, a view of a table, or other object in the database.
2	**ALTER** Modifies an existing database object, such as a table.
3	**DROP** Deletes an entire table, a view of a table or other objects in the database.

DML - Data Manipulation Language

Sr.No.	Command & Description
1	**SELECT** Retrieves certain records from one or more tables.
2	**INSERT** Creates a record.
3	**UPDATE** Modifies records.
4	**DELETE** Deletes records.

DCL - Data Control Language

Sr.No.	Command & Description
1	**GRANT** Gives a privilege to user.
2	**REVOKE** Takes back privileges granted from user.

- The SQL CREATE TABLE statement is used to create a new table.

- CREATE TABLE table_name(column1 datatype, column2 datatype, column3 datatype, ... columnN datatype, PRIMARY KEY(one or more columns));

- Example
- The following code block is an example, which creates a CUSTOMERS table with an ID as a primary key and NOT NULL are the constraints showing that these fields cannot be NULL while creating records in this table −

```
SQL> CREATE TABLE CUSTOMERS(
   ID   INT              NOT NULL,
   NAME VARCHAR (20)     NOT NULL,
   AGE  INT              NOT NULL,
   ADDRESS  CHAR (25) ,
   SALARY   DECIMAL (18, 2),
   PRIMARY KEY (ID)
);
```

SQL DROP TABLE

- The SQL DROP TABLE statement is used to remove a table definition and all the data, indexes, triggers, constraints and permission specifications for that table.

- DROP TABLE table_name;
- SQL> DROP TABLE CUSTOMERS;

INSERT INTO Statement

- The SQL **INSERT INTO** Statement is used to add new rows of data to a table in the database.

- INSERT INTO TABLE_NAME (column1, column2, column3,...columnN)
- VALUES (value1, value2, value3,...valueN);
- INSERT INTO TABLE_NAME VALUES (value1,value2,value3,...valueN);
- INSERT INTO CUSTOMERS
- VALUES (7, 'Muffy', 24, 'Indore', 10000.00);

- +----+----------+-----+-----------+----------+
- | ID | NAME | AGE | ADDRESS | SALARY |
- +----+----------+-----+-----------+----------+
- | 1 | Ramesh | 32 | Ahmedabad | 2000.00 |
- | 2 | Khilan | 25 | Delhi | 1500.00 |
- | 3 | kaushik | 23 | Kota | 2000.00 |
- | 4 | Chaitali | 25 | Mumbai | 6500.00 |
- | 5 | Hardik | 27 | Bhopal | 8500.00 |
- | 6 | Komal | 22 | MP | 4500.00 |
- | 7 | Muffy | 24 | Indore | 10000.00 |
- +----+----------+-----+-----------+----------+

- The following code is an example, which would fetch the ID, Name and Salary fields of the customers available in CUSTOMERS table.

- SQL> SELECT ID, NAME, SALARY FROM CUSTOMERS;

- ----+----------+----------+
- | ID | NAME | SALARY |
- +----+----------+----------+
- | 1 | Ramesh | 2000.00 |
- | 2 | Khilan | 1500.00 |
- | 3 | kaushik | 2000.00 |
- | 4 | Chaitali | 6500.00 |
- | 5 | Hardik | 8500.00 |
- | 6 | Komal | 4500.00 |
- | 7 | Muffy | 10000.00 |
- +----+----------+----------+

WHERE clause

- The SQL WHERE clause is used to specify a condition while fetching the data from a single table or by joining with multiple tables. If the given condition is satisfied, then only it returns a specific value from the table. You should use the WHERE clause to filter the records and fetching only the necessary records.

- The WHERE clause is not only used in the SELECT statement, but it is also used in the UPDATE, DELETE statement, etc., which we would examine in the subsequent chapters.

- Syntax
- The basic syntax of the SELECT statement with the WHERE clause is as shown below.
- SELECT column1, column2, columnN FROM table_name WHERE [condition]

The AND Operator

- The basic syntax of the AND operator with a WHERE clause is as follows −

- SELECT column1, column2, columnN
- FROM table_name
- WHERE [condition1] AND [condition2]...AND [conditionN];

- Consider the CUSTOMERS table having the following records −

- +----+----------+-----+-----------+----------+
- | ID | NAME | AGE | ADDRESS | SALARY |
- +----+----------+-----+-----------+----------+
- | 1 | Ramesh | 32 | Ahmedabad | 2000.00 |
- | 2 | Khilan | 25 | Delhi | 1500.00 |
- | 3 | kaushik | 23 | Kota | 2000.00 |
- | 4 | Chaitali | 25 | Mumbai | 6500.00 |
- | 5 | Hardik | 27 | Bhopal | 8500.00 |
- | 6 | Komal | 22 | MP | 4500.00 |
- | 7 | Muffy | 24 | Indore | 10000.00 |
- +----+----------+-----+-----------+----------+

- SQL> SELECT ID, NAME, SALARY
- FROM CUSTOMERS
- WHERE SALARY > 2000 AND age < 25;

- +----+-------+----------+
- | ID | NAME | SALARY |
- +----+-------+----------+
- | 6 | Komal | 4500.00 |
- | 7 | Muffy | 10000.00 |
- +----+-------+----------+

The OR Operator

- The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause
- Syntax
- The basic syntax of the OR operator with a WHERE clause is as follows −
- SELECT column1, column2, columnN FROM table_name WHERE [condition1] OR [condition2]...OR [conditionN]

- Consider the CUSTOMERS table having the following records −

- +----+----------+-----+-----------+----------+
- | ID | NAME | AGE | ADDRESS | SALARY |
- +----+----------+-----+-----------+----------+
- | 1 | Ramesh | 32 | Ahmedabad | 2000.00 |
- | 2 | Khilan | 25 | Delhi | 1500.00 |
- | 3 | kaushik | 23 | Kota | 2000.00 |
- | 4 | Chaitali | 25 | Mumbai | 6500.00 |
- | 5 | Hardik | 27 | Bhopal | 8500.00 |
- | 6 | Komal | 22 | MP | 4500.00 |
- | 7 | Muffy | 24 | Indore | 10000.00 |
- +----+----------+-----+-----------+----------+

- The following code block hasa query, which would fetch the ID, Name and Salary fields from the CUSTOMERS table, where the salary is greater than 2000 OR the age is less than 25 years.
- SQL> SELECT ID, NAME, SALARY FROM CUSTOMERS WHERE SALARY > 2000 OR age < 25;
- This would produce the following result −

- +----+----------+----------+
- | ID | NAME | SALARY |
- +----+----------+----------+
- | 3 | kaushik | 2000.00 |
- | 4 | Chaitali | 6500.00 |
- | 5 | Hardik | 8500.00 |
- | 6 | Komal | 4500.00 |
- | 7 | Muffy | 10000.00 |
- +----+----------+----------+

- You can combine N number of conditions using the AND or the OR operators.

- Example
- Consider the CUSTOMERS table having the following records −

- +----+----------+-----+-----------+----------+
- | ID | NAME | AGE | ADDRESS | SALARY |
- +----+----------+-----+-----------+----------+
- | 1 | Ramesh | 32 | Ahmedabad | 2000.00 |
- | 2 | Khilan | 25 | Delhi | 1500.00 |
- | 3 | kaushik | 23 | Kota | 2000.00 |
- | 4 | Chaitali | 25 | Mumbai | 6500.00 |
- | 5 | Hardik | 27 | Bhopal | 8500.00 |
- | 6 | Komal | 22 | MP | 4500.00 |
- | 7 | Muffy | 24 | Indore | 10000.00 |
- +----+----------+-----+-----------+----------+

SQL UPDATE

- The SQL UPDATE Query is used to modify the existing records in a table. You can use the WHERE clause with the UPDATE query to update the selected rows, otherwise all the rows would be affected.

- Syntax
- The basic syntax of the UPDATE query with a WHERE clause is as follows −
- UPDATE table_name SET column1 = value1, column2 = value2...., columnN = valueN WHERE [condition];

UPDATE

- SQL> UPDATE CUSTOMERS
- SET ADDRESS = 'Pune', SALARY = 1000.00;
- Now, CUSTOMERS table would have the following records –

- +----+----------+-----+---------+---------+
- | ID | NAME | AGE | ADDRESS | SALARY |
- +----+----------+-----+---------+---------+
- | 1 | Ramesh | 32 | Pune | 1000.00 |
- | 2 | Khilan | 25 | Pune | 1000.00 |
- | 3 | kaushik | 23 | Pune | 1000.00 |
- | 4 | Chaitali | 25 | Pune | 1000.00 |
- | 5 | Hardik | 27 | Pune | 1000.00 |
- | 6 | Komal | 22 | Pune | 1000.00 |
- | 7 | Muffy | 24 | Pune | 1000.00 |
- +----+----------+-----+---------+---------+

SQL DELETE Query

- The SQL DELETE Query is used to delete the existing records from a table.

- You can use the WHERE clause with a DELETE query to delete the selected rows, otherwise all the records would be deleted.
- Syntax
- The basic syntax of the DELETE query with the WHERE clause is as follows −

- DELETE FROM table_name
- WHERE [condition];
- You can combine N number of conditions using AND or OR operators.

- Example
- Consider the CUSTOMERS table having the following records −

- +----+----------+-----+-----------+----------+
- | ID | NAME | AGE | ADDRESS | SALARY |
- +----+----------+-----+-----------+----------+
- | 1 | Ramesh | 32 | Ahmedabad | 2000.00 |
- | 2 | Khilan | 25 | Delhi | 1500.00 |
- | 3 | kaushik | 23 | Kota | 2000.00 |
- | 4 | Chaitali | 25 | Mumbai | 6500.00 |
- | 5 | Hardik | 27 | Bhopal | 8500.00 |
- | 6 | Komal | 22 | MP | 4500.00 |
- | 7 | Muffy | 24 | Indore | 10000.00 |
- +----+----------+-----+-----------+----------+

- The following code has a query, which will DELETE a customer, whose ID is 6.

- SQL> DELETE FROM CUSTOMERS
- WHERE ID = 6;
- Now, the CUSTOMERS table would have the following records.

- +----+----------+-----+-----------+----------+
- | ID | NAME | AGE | ADDRESS | SALARY |
- +----+----------+-----+-----------+----------+
- | 1 | Ramesh | 32 | Ahmedabad | 2000.00 |
- | 2 | Khilan | 25 | Delhi | 1500.00 |
- | 3 | kaushik | 23 | Kota | 2000.00 |
- | 4 | Chaitali | 25 | Mumbai | 6500.00 |
- | 5 | Hardik | 27 | Bhopal | 8500.00 |
- | 7 | Muffy | 24 | Indore | 10000.00 |
- +----+----------+-----+-----------+----------+

- If you want to DELETE all the records from the CUSTOMERS table, you do not need to use the WHERE clause and the DELETE query would be as follows −

- SQL> DELETE FROM CUSTOMERS;
- Now, the CUSTOMERS table would not have any record.

SQL LIKE

- The SQL LIKE clause is used to compare a value to similar values using wildcard operators. There are two wildcards used in conjunction with the LIKE operator.

- The percent sign (%)
- The underscore (_)
- The percent sign represents zero, one or multiple characters. The underscore represents a single number or character. These symbols can be used in combinations.

- Syntax
- The basic syntax of % and _ is as follows −

- SELECT FROM table_name
- WHERE column LIKE 'XXXX%'

–

Sr.No.	Statement & Description
1	**WHERE SALARY LIKE '200%'** Finds any values that start with 200.
2	**WHERE SALARY LIKE '%200%'** Finds any values that have 200 in any position.
3	**WHERE SALARY LIKE '_00%'** Finds any values that have 00 in the second and third positions.
4	**WHERE SALARY LIKE '2_%_%'** Finds any values that start with 2 and are at least 3 characters in length.
5	**WHERE SALARY LIKE '%2'** Finds any values that end with 2.
6	**WHERE SALARY LIKE '_2%3'** Finds any values that have a 2 in the second position and end with a 3.
7	**WHERE SALARY LIKE '2___3'** Finds any values in a five-digit number that start with 2 and end with 3.

SQL TOP clause

- The SQL **TOP** clause is used to fetch a TOP N number or X percent records from a table.
- **Note** – All the databases do not support the TOP clause. For example MySQL supports the **LIMIT** clause to fetch limited number of records while Oracle uses the **ROWNUM** command to fetch a limited number of records.
- Syntax
- The basic syntax of the TOP clause with a SELECT statement would be as follows.
- SELECT TOP number|percent column_name(s) FROM table_name WHERE [condition]

- The SQL ORDER BY clause is used to sort the data in ascending or descending order, based on one or more columns. Some databases sort the query results in an ascending order by default.

- Syntax
- The basic syntax of the ORDER BY clause is as follows −

- SELECT column-list
- FROM table_name
- [WHERE condition]
- [ORDER BY column1, column2, .. columnN] [ASC | DESC];
- You can use more than one column in the ORDER BY clause. Make sure whatever column you are using to sort that column should be in the column-list.

SQL GROUP BY

- The SQL GROUP BY clause is used in collaboration with the SELECT statement to arrange identical data into groups. This GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.

- Syntax
- Syntax
- The basic syntax of a GROUP BY clause is shown in the following code block. The GROUP BY clause must follow the conditions in the WHERE clause and must precede the ORDER BY clause if one is used.
- SELECT column1, column2 FROM table_name WHERE [conditions] GROUP BY column1, column2 ORDER BY column1, column2

SQL SELECT

- The SQL SELECT statement is used to fetch the data from a database table which returns this data in the form of a result table. These result tables are called result-sets.

- SELECT column1, column2, columnN FROM table_name;

- SELECT * FROM table_name;

Normal Forms in DBMS

- Normalization is the process of minimizing redundancy from a relation or set of relations. Redundancy in relation may cause insertion, deletion and updation anomalies. So, it helps to minimize the redundancy in relations. Normal forms are used to eliminate or reduce redundancy in database tables.

- 1. First Normal Form –
- If a relation contain composite or multi-valued attribute, it violates first normal form or a relation is in first normal form if it does not contain any composite or multi-valued attribute. A relation is in first normal form if every attribute in that relation is singled valued attribute.

Normal Forms in DBMS

- Example 1 – Relation STUDENT in table 1 is not in 1NF because of multi-valued attribute STUD_PHONE. Its decomposition into 1NF has been shown in table 2.

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY
1	RAM	9716271721, 9871717178	HARYANA	INDIA
2	RAM	9898297281	PUNJAB	INDIA
3	SURESH		PUNJAB	INDIA

Table 1

Conversion to first normal form

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY
1	RAM	9716271721	HARYANA	
1	RAM	9871717178	HARYANA	INDIA
2	RAM	9898297281	PUNJAB	INDIA
3	SURESH		PUNJAB	INDIA

Table 2

Example 2 –

-
- ID Name Courses
- ----------
- 1 A c1, c2
- 2 E c3
- 3 M C2, c3
- In the above table Course is a multi valued attribute so it is not in 1NF.

1NF

- Below Table is in 1NF as there is no multi valued attribute

- ID Name Course
- ----------
- 1 A c1
- 1 A c2
- 2 E c3
- 3 M c2
- 3 M c3

2. Second Normal Form –

- To be in second normal form, a relation must be in first normal form and relation must not contain any partial dependency. A relation is in 2NF if it has **No Partial Dependency,** i.e.**,** no non-prime attribute (attributes which are not part of any candidate key) is dependent on any proper subset of any candidate key of the table.
- **Partial Dependency –** If the proper subset of candidate key determines non-prime attribute, it is called partial dependency.

Example 1

- Example 1 – Consider table-3 as following below.
- STUD_NO COURSE_NO COURSE_FEE
- 1 C1 1000
- 2 C2 1500
- 1 C4 2000
- 4 C3 1000
- 4 C1 1000
- 2 C5 2000
- {Note that, there are many courses having the same course fee. }

- Here,
- COURSE_FEE cannot alone decide the value of COURSE_NO or STUD_NO;
- COURSE_FEE together with STUD_NO cannot decide the value of COURSE_NO;
- COURSE_FEE together with COURSE_NO cannot decide the value of STUD_NO;
- Hence,
- COURSE_FEE would be a non-prime attribute, as it does not belong to the one only candidate key {STUD_NO, COURSE_NO} ;
- But, COURSE_NO -> COURSE_FEE , i.e., COURSE_FEE is dependent on COURSE_NO, which is a proper subset of the candidate key. Non-prime attribute COURSE_FEE is dependent on a proper subset of the candidate key, which is a partial dependency and so this relation is not in 2NF.

- To convert the above relation to 2NF,
- we need to split the table into two tables such as :
- Table 1: STUD_NO, COURSE_NO
- Table 2: COURSE_NO, COURSE_FEE

Table 1		Table 2	
STUD_NO	COURSE_NO	COURSE_NO	COURSE_FEE
1	C1	C1	1000
2	C2	C2	1500
1	C4	C3	1000
4	C3	C4	2000
4	C1	C5	2000

3. Third Normal Form –

- A relation is in third normal form, if there is **no transitive dependency** for non-prime attributes as well as it is in second normal form.
  A relation is in 3NF if **at least one of the following condition holds** in every non-trivial function dependency X –> Y
  - X is a super key.
  - Y is a prime attribute (each element of Y is part of some candidate key).

STUD_NO	STUD_NAME	STUD_STATE	STUD_COUNTRY	STUD_AGE
1	RAM	HARYANA	INDIA	20
2	RAM	PUNJAB	INDIA	19
3	SURESH	PUNJAB	INDIA	21

Table 4

Boyce-Codd Normal Form (BCNF) –

- A relation R is in BCNF if R is in Third Normal Form and for every FD, LHS is super key. A relation is in BCNF iff in every non-trivial functional dependency X –> Y, X is a super key.

Example 2 –

- ID Name Courses
- ----------
- 1 A c1, c2
- 2 E c3
- 3 M C2, c3
- In the above table Course is a multi valued attribute so it is not in 1NF.

Third Normal Form –

- A relation is in third normal form, if there is no transitive dependency for non-prime attributes as well as it is in second normal form.
- A relation is in 3NF if at least one of the following condition holds in every non-trivial function dependency X –> Y

- X is a super key.
- Y is a prime attribute (each element of Y is part of some candidate key).
- image5

Third Normal Form –

- **Example 2 –** Consider relation R(A, B, C, D, E)
  A -> BC,
  CD -> E,
  B -> D,
  E -> A
  All possible candidate keys in above relation are {A, E, CD, BC} All attribute are on right sides of all functional dependencies are prime.

- B^+ Tree Deletion
- B^+ tree entries are deleted at the leaf nodes.
- The target entry is searched and deleted.
  - If it is an internal node, delete and replace with the entry from the left position.
- After deletion, underflow is tested,
  - If underflow occurs, distribute the entries from the nodes left to it.
- If distribution is not possible from left, then
  - Distribute from the nodes right to it.
- If distribution is not possible from left or from right, then
  - Merge the node with left and right to it.

- Serializability
- When multiple transactions are being executed by the operating system in a multiprogramming environment, there are possibilities that instructions of one transactions are interleaved with some other transaction.
- **Schedule** − A chronological execution sequence of a transaction is called a schedule. A schedule can have many transactions in it, each comprising of a number of instructions/tasks.
- **Serial Schedule** − It is a schedule in which transactions are aligned in such a way that one transaction is executed first. When the first transaction completes its cycle, then the next transaction is executed. Transactions are ordered one after the other. This type of schedule is called a serial schedule, as transactions are executed in a serial manner.

Boyce Codd normal form (BCNF)

- BCNF is the advance version of 3NF. It is stricter than 3NF.
- A table is in BCNF if every functional dependency X → Y, X is the super key of the table.
- For BCNF, the table should be in 3NF, and for every FD, LHS is super key.
- Example: Let's assume there is a company where employees work in more than one department.

EMPLOYEE table:

EMP_ID	EMP_COUNTRY	EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
264	India	Designing	D394	283
264	India	Testing	D394	300
364	UK	Stores	D283	232
364	UK	Developing	D283	549

- In the above table Functional dependencies are as follows:
- EMP_ID → EMP_COUNTRY
- EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate key: {EMP-ID, EMP-DEPT}
The table is not in BCNF because neither EMP_DEPT nor EMP_ID alone are keys.
To convert the given table into BCNF, we decompose it into three tables:
EMP_COUNTRY table:

EMP_ID	EMP_COUNTRY
264	India
264	India

EMP_DEPT table:

EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
Designing	D394	283
Testing	D394	300
Stores	D283	232
Developing	D283	549

EMP_DEPT_MAPPING table:

EMP_ID	EMP_DEPT
D394	283
D394	300
D283	232
D283	549

- **Functional dependencies:**

1. EMP_ID → EMP_COUNTRY
2. EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

- **Candidate keys:**
- **For the first table:** EMP_ID
  For the second table: EMP_DEPT
  For the third table: {EMP_ID, EMP_DEPT}
- Now, this is in BCNF because left side part of both the functional dependencies is a key.