Unit 1 : File Structure and Organization

File Organization

The File is a collection of records. Using the primary key, we can access the records. The type and frequency of access can be determined by the type of file organization which was used for a given set of records.

File organization is a logical relationship among various records. This method defines how file records are mapped onto disk blocks.

File organization is used to describe the way in which the records are stored in terms of blocks, and the blocks are placed on the storage medium.

The first approach to map the database to the file is to use the several files and store only one fixed length record in any given file. An alternative approach is to structure our files so that we can contain multiple lengths for records.

Files of fixed length records are easier to implement than the files of variable length records.

Physical and Logical File Systems

1. Physical files :

Physical files contain the actual data that is stored on an iSeries system, and a description of how data is to be presented to or received from a program. They contain only one record format, and one or more members. Records in database files can be described using either a field level description or record level description.

A field-level description describes the fields in the record to the system. Database files that are created with field level descriptions are referred to as externally described files. A record-level description describes only the length of the record, and not the contents of the record. Database files that are created with record level descriptions are referred to as program-described files. This means that your ILE C/C++ program must describe the fields in the record.

2. Logical files :

Logical files do not contain data. They contain a description of records that are found in one or more physical files. A logical file is a view or representation of one or more physical files. Logical files that contain more than one format are referred to as multi-format logical files.

If your program processes a logical file which contains more than one record format, you can use the _Rformat() function to set the format you wish to use. Some operations cannot be performed on logical files. If you open a logical file for stream file processing with open modes W, W+, WB or WB+, the file is opened but not cleared. If you open a logical file for record file processing with open modes WR or WR+, the file is opened but not cleared. Records in iSeries database files can be described using either a field level description or record level description.

Physical versus Logical Files :

- **Physical File –**
  A collection of bytes stored on a disk or tape.
- **Logical File –**
  A "Channel" (like a telephone line) that hides the details of the file's location and physical format to the program.

When a program wants to use a particular file, "data", the operating system must find the physical file called "data" and make logical name by assigning a logical file to it. This logical file has a logical name which is what is used inside the program.

PHYSICAL FILE	LOGICAL FILE
It occupies the portion of memory. It contains the original data.	It does not occupy memory space. It does not contain data.
A physical file contains one record format.	It can contain upto 32 record formats.
It can exist without logical file.	It cannot exist without physical file.
If there is a logical file for physical file, the physical file cannot be deleted until and unless we delete the logical file.	If there is a logical file for a physical file, the logical file can be deleted without deleting the physical file.
CRTPF command is used to make such object.	CRTLF command is used to make such object.

Logical Storage Views – viewed by users are a collection of files organized within directories and storage volumes.

- Logical file structure is independent of its physical implementation.
- Logical file structure "ignores".

Physical storage allocations – records can be stored in separate file locations. Data access methods and Data encoding methods.

Physical Storage Views – a collection of physical storage locations organized as a linear address space.

- File is subdivided into records.
- Record usually contains information about a single customer, thing such as a product in inventory, or an event.
- Records are divided into fields.
- Fields are individual units of data.

File Operations

Operations on database files can be broadly classified into two categories −

- **Update Operations**
- **Retrieval Operations**

Update operations change the data values by insertion, deletion, or update. Retrieval operations, on the other hand, do not alter the data but retrieve them after optional conditional filtering. In both types of operations, selection plays a significant role. Other than creation and deletion of a file, there could be several operations, which can be done on files.

- **Open** − A file can be opened in one of the two modes, **read mode** or **write mode**. In read mode, the operating system does not allow anyone to alter data. In other words, data is read only. Files opened in read mode can be shared among several entities. Write mode allows data modification. Files opened in write mode can be read but cannot be shared.
- **Locate** − Every file has a file pointer, which tells the current position where the data is to be read or written. This pointer can be adjusted accordingly. Using find (seek) operation, it can be moved forward or backward.
- **Read** − By default, when files are opened in read mode, the file pointer points to the beginning of the file. There are options where the user can tell the operating system where to locate the file pointer at the time of opening a file. The very next data to the file pointer is read.
- **Write** − User can select to open a file in write mode, which enables them to edit its contents. It can be deletion, insertion, or modification. The file pointer can be located at the time of opening or can be dynamically changed if the operating system allows to do so.
- **Close** − This is the most important operation from the operating system's point of view. When a request to close a file is generated, the operating system
    - removes all the locks (if in shared mode),
    - saves the data (if altered) to the secondary storage media, and
    - releases all the buffers and file handlers associated with the file.

The organization of data inside a file plays a major role here. The process to locate the file pointer to a desired record inside a file various based on whether the records are arranged sequentially or clustered.

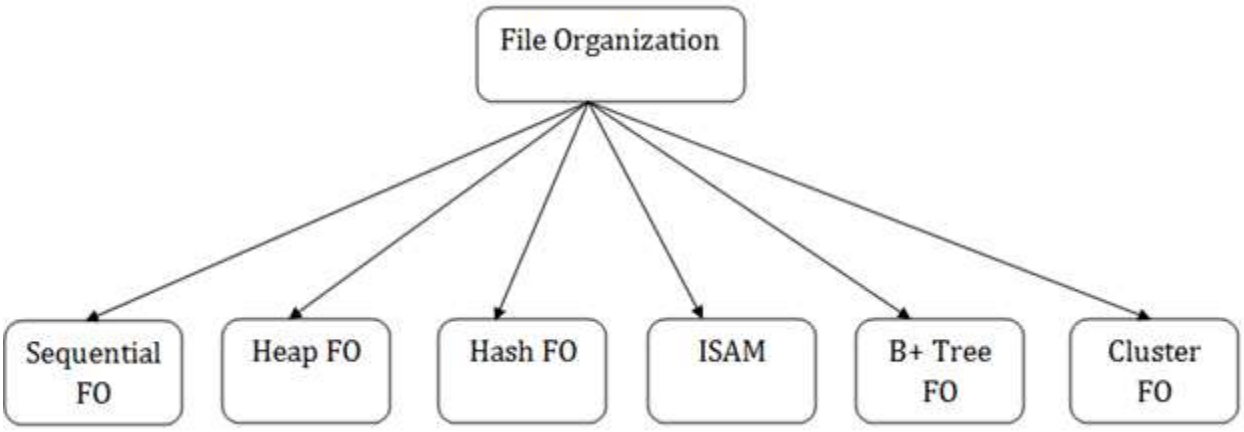
Objective of file organization

- It contains an optimal selection of records, i.e., records can be selected as fast as possible.
- To perform insert, delete or update transaction on the records should be quick and easy.
- The duplicate records cannot be induced as a result of insert, update or delete.
- For the minimal cost of storage, records should be stored efficiently.

Types of file organization:

File organization contains various methods. These particular methods have pros and cons on the basis of access or selection. In the file organization, the programmer decides the best-suited file organization method according to his requirement.

Types of file organization are as follows:



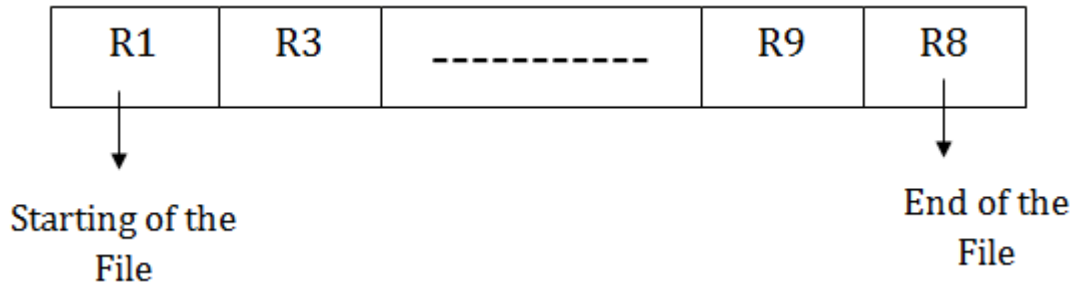
Sequential File Organization

This method is the easiest method for file organization. In this method, files are stored sequentially. This method can be implemented in two ways:

1. Pile File Method:

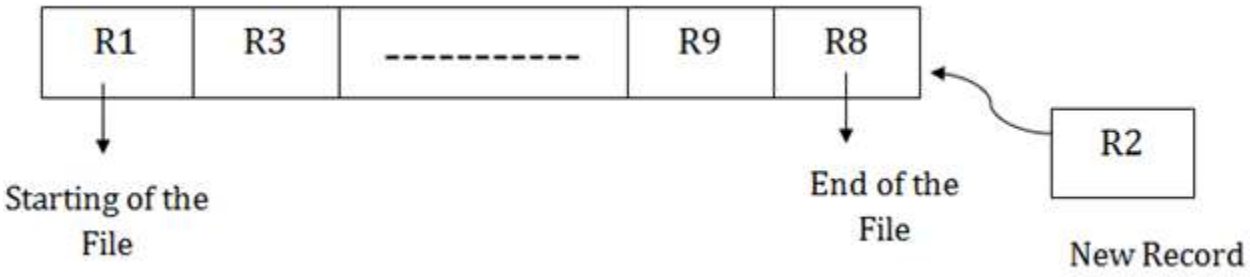
It is a quite simple method. In this method, we store the record in a sequence, i.e., one after another. Here, the record will be inserted in the order in which they are inserted into tables.

In case of updating or deleting of any record, the record will be searched in the memory blocks. When it is found, then it will be marked for deleting, and the new record is inserted.



1.Insertion of the new record:

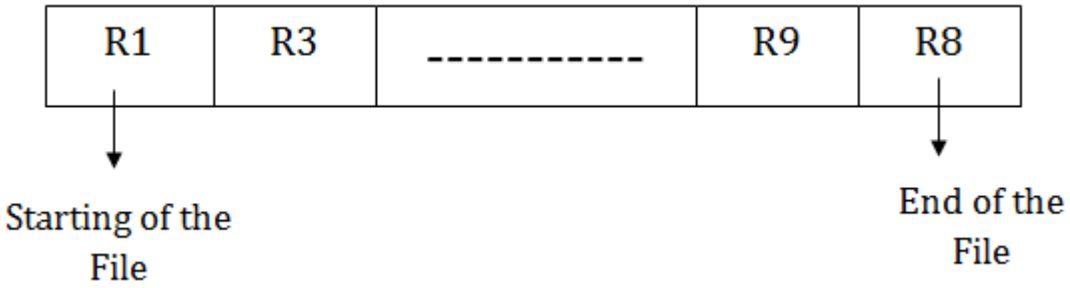
Suppose we have four records R1, R3 and so on upto R9 and R8 in a sequence. Hence, records are nothing but a row in the table. Suppose we want to insert a new record R2 in the sequence, then it will be placed at the end of the file. Here, records are nothing but a row in any table.



2. Sorted File Method:

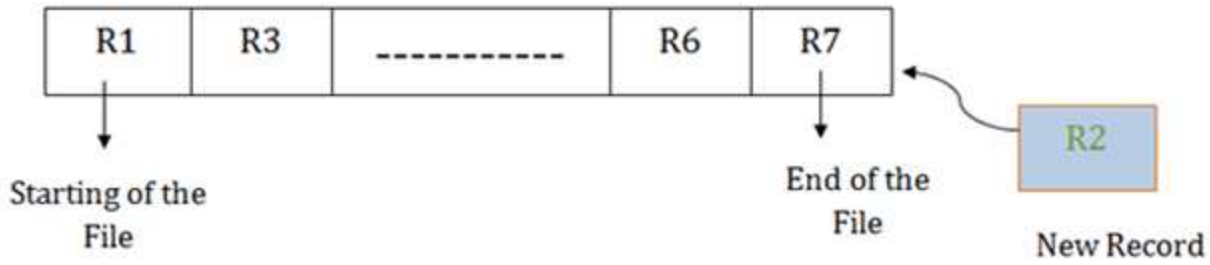
In this method, the new record is always inserted at the file's end, and then it will sort the sequence in ascending or descending order. Sorting of records is based on any primary key or any other key.

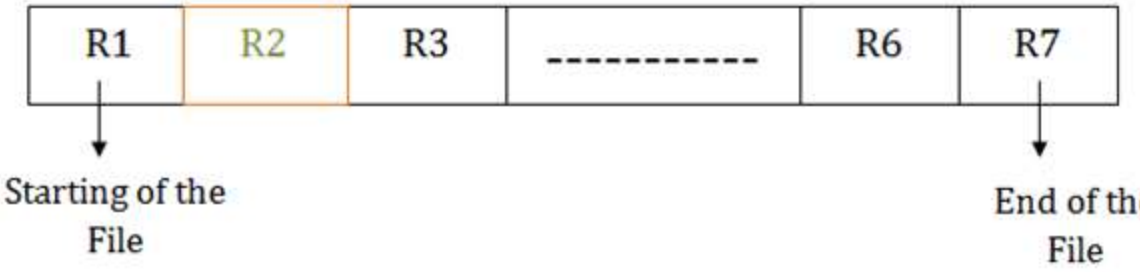
In the case of modification of any record, it will update the record and then sort the file, and lastly, the updated record is placed in the right place.



Insertion of the new record:

Suppose there is a preexisting sorted sequence of four records R1, R3 and so on upto R6 and R7. Suppose a new record R2 has to be inserted in the sequence, then it will be inserted at the end of the file, and then it will sort the sequence.





Pros of sequential file organization

It contains a fast and efficient method for the huge amount of data.

In this method, files can be easily stored in cheaper storage mechanism like magnetic tapes.

It is simple in design. It requires no much effort to store the data.

This method is used when most of the records have to be accessed like grade calculation of a student, generating the salary slip, etc.

This method is used for report generation or statistical calculations.

Cons of sequential file organization

It will waste time as we cannot jump on a particular record that is required but we have to move sequentially which takes our time.

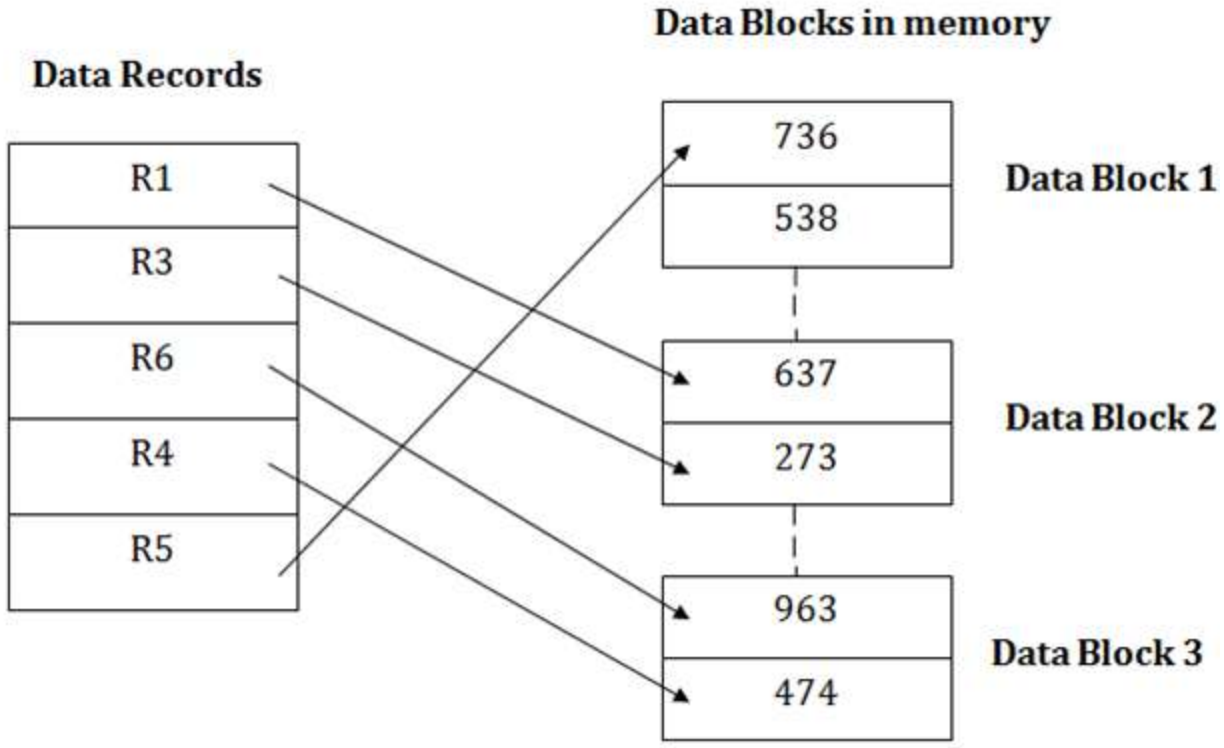
Sorted file method takes more time and space for sorting the records.

Heap file organization

It is the simplest and most basic type of organization. It works with data blocks. In heap file organization, the records are inserted at the file's end. When the records are inserted, it doesn't require the sorting and ordering of records.

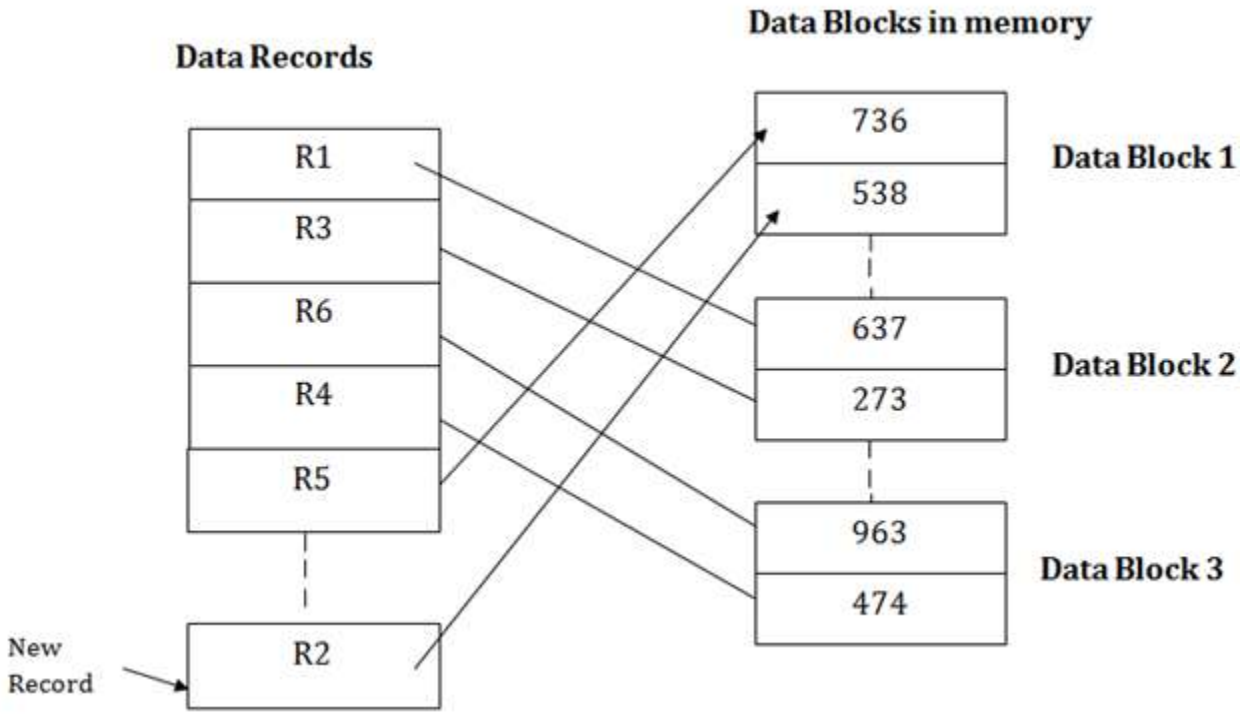
When the data block is full, the new record is stored in some other block. This new data block need not to be the very next data block, but it can select any data block in the memory to store new records. The heap file is also known as an unordered file.

In the file, every record has a unique id, and every page in a file is of the same size. It is the DBMS responsibility to store and manage the new records.



Insertion of a new record

Suppose we have five records R1, R3, R6, R4 and R5 in a heap and suppose we want to insert a new record R2 in a heap. If the data block 3 is full then it will be inserted in any of the database selected by the DBMS, let's say data block 1.



If we want to search, update or delete the data in heap file organization, then we need to traverse the data from staring of the file till we get the requested record.

If the database is very large then searching, updating or deleting of record will be time-consuming because there is no sorting or ordering of records. In the heap file organization, we need to check all the data until we get the requested record.

Pros of Heap file organization

It is a very good method of file organization for bulk insertion. If there is a large number of data which needs to load into the database at a time, then this method is best suited.

In case of a small database, fetching and retrieving of records is faster than the sequential record.

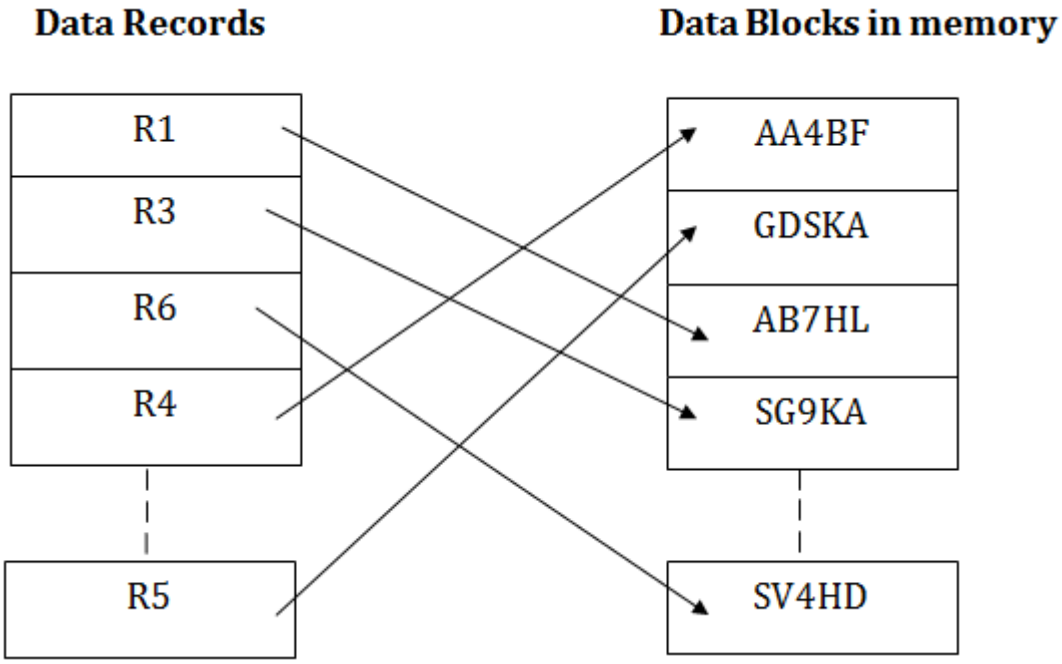
Cons of Heap file organization

This method is inefficient for the large database because it takes time to search or modify the record.

This method is inefficient for large databases.

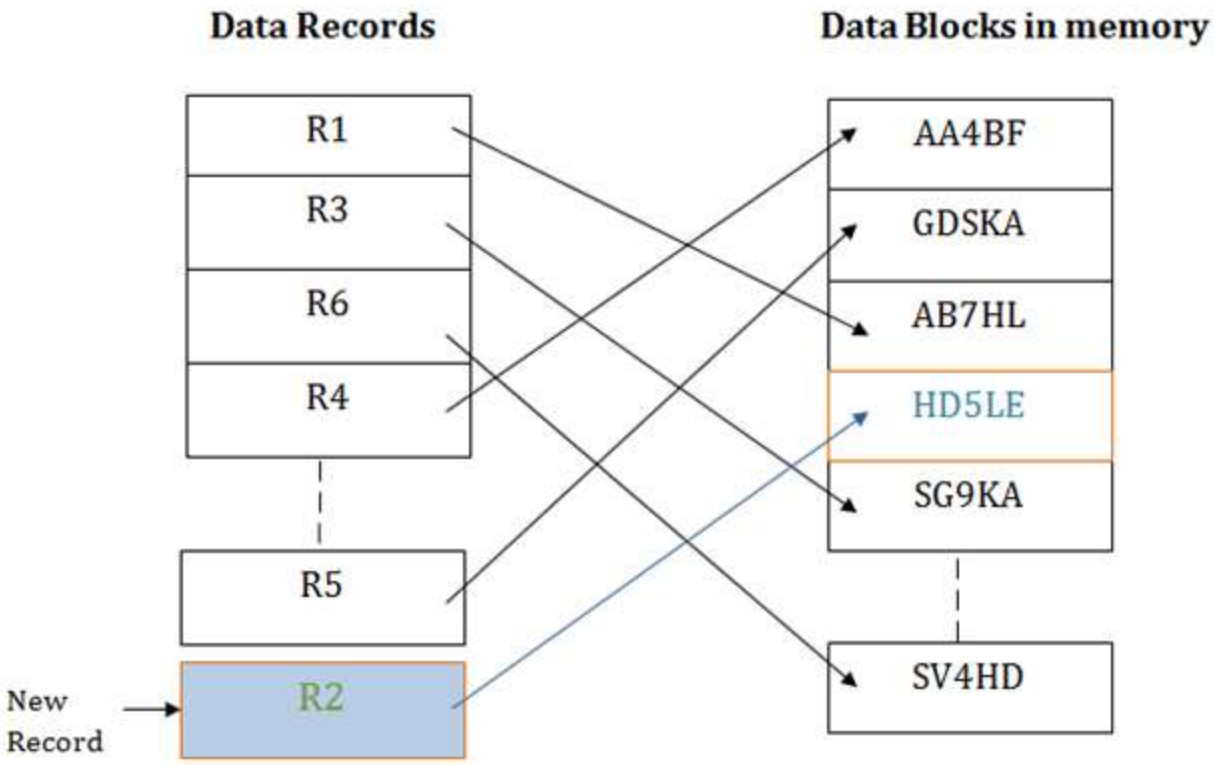
Hash File Organization

Hash File Organization uses the computation of hash function on some fields of the records. The hash function's output determines the location of disk block where the records are to be placed.



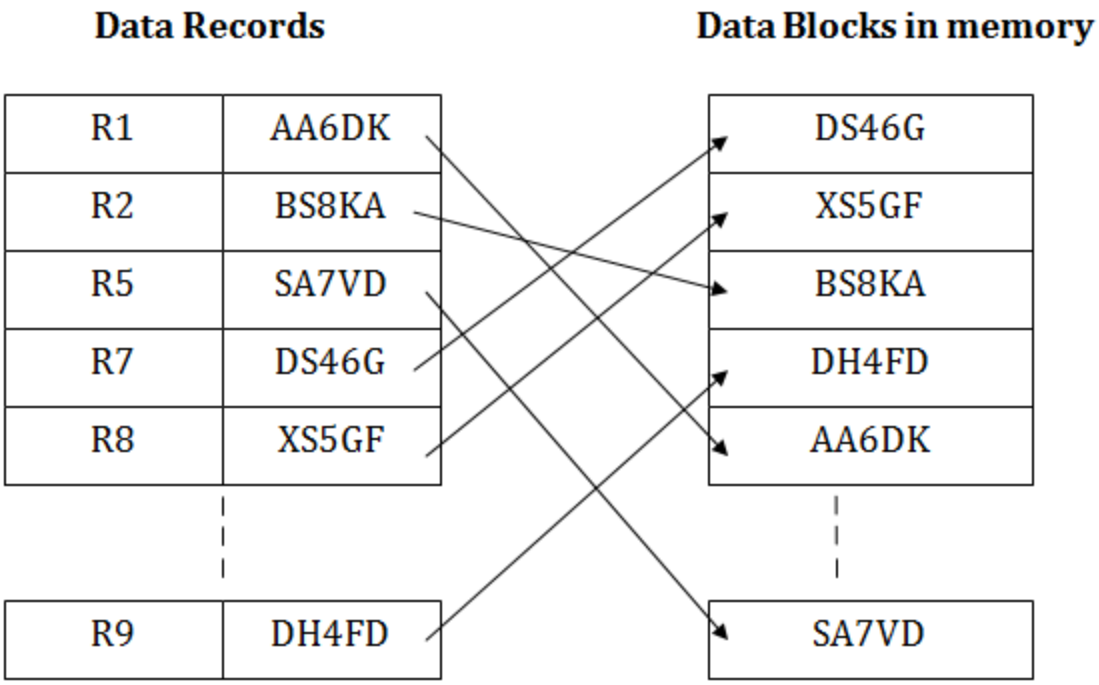
When a record has to be received using the hash key columns, then the address is generated, and the whole record is retrieved using that address. In the same way, when a new record has to be inserted, then the address is generated using the hash key and record is directly inserted. The same process is applied in the case of delete and update.

In this method, there is no effort for searching and sorting the entire file. In this method, each record will be stored randomly in the memory.



Indexed sequential access method (ISAM)

ISAM method is an advanced sequential file organization. In this method, records are stored in the file using the primary key. An index value is generated for each primary key and mapped with the record. This index contains the address of the record in the file.



If any record has to be retrieved based on its index value, then the address of the data block is fetched and the record is retrieved from the memory.

Pros of ISAM:

In this method, each record has the address of its data block, searching a record in a huge database is quick and easy.

This method supports range retrieval and partial retrieval of records. Since the index is based on the primary key values, we can retrieve the data for the given range of value. In the same way, the partial value can also be easily searched, i.e., the student name starting with 'JA' can be easily searched.

Cons of ISAM

This method requires extra space in the disk to store the index value.

When the new records are inserted, then these files have to be reconstructed to maintain the sequence.

When the record is deleted, then the space used by it needs to be released. Otherwise, the performance of the database will slow down.

Cluster file organization

When the two or more records are stored in the same file, it is known as clusters. These files will have two or more tables in the same data block, and key attributes which are used to map these tables together are stored only once.

This method reduces the cost of searching for various records in different files.

The cluster file organization is used when there is a frequent need for joining the tables with the same condition. These joins will give only a few records from both tables. In the given example, we are retrieving the record for only particular departments. This method can't be used to retrieve the record for the entire department.

EMPLOYEE

EMP_ID	EMP_NAME	ADDRESS	DEP_ID
1	John	Delhi	14
2	Robert	Gujarat	12
3	David	Mumbai	15
4	Amelia	Meerut	11
5	Kristen	Noida	14
6	Jackson	Delhi	13
7	Amy	Bihar	10
8	Sonoo	UP	12

DEPARTMENT

DEP_ID	DEP_NAME
10	Math
11	English
12	Java
13	Physics
14	Civil
15	Chemistry

Cluster Key

DEP_ID	DEP_NAME	EMP_ID	EMP_NAME	ADDRESS
10	Math	7	Amy	Bihar
11	English	4	Amelia	Meerut
12	Java	2	Robert	Gujarat
12		8	Sonoo	UP
13	Physics	6	Jackson	Delhi
14	Civil	1	John	Delhi
14		5	Kristen	Noida
15	Chemistry	3	David	Mumbai

In this method, we can directly insert, update or delete any record. Data is sorted based on the key with which searching is done. Cluster key is a type of key with which joining of the table is performed.

Types of Cluster file organization:

Cluster file organization is of two types:

1. Indexed Clusters:
   In indexed cluster, records are grouped based on the cluster key and stored together. The above EMPLOYEE and DEPARTMENT relationship is an example of an indexed cluster. Here, all the records are grouped based on the cluster key- DEP_ID and all the records are grouped.

   2. Hash Clusters:
   It is similar to the indexed cluster. In hash cluster, instead of storing the records based on the cluster key, we generate the value of the hash key for the cluster key and store the records with the same hash key value.

Pros of Cluster file organization

The cluster file organization is used when there is a frequent request for joining the tables with same joining condition.

It provides the efficient result when there is a 1:M mapping between the tables.

Cons of Cluster file organization

This method has the low performance for the very large database.

If there is any change in joining condition, then this method cannot use. If we change the condition of joining then traversing the file takes a lot of time.

This method is not suitable for a table with a 1:1 condition.

Indexing in DBMS

- Indexing is used to optimize the performance of a database by minimizing the number of disk accesses required when a query is processed.
- The index is a type of data structure. It is used to locate and access the data in a database table quickly.

Index structure:

Indexes can be created using some database columns.

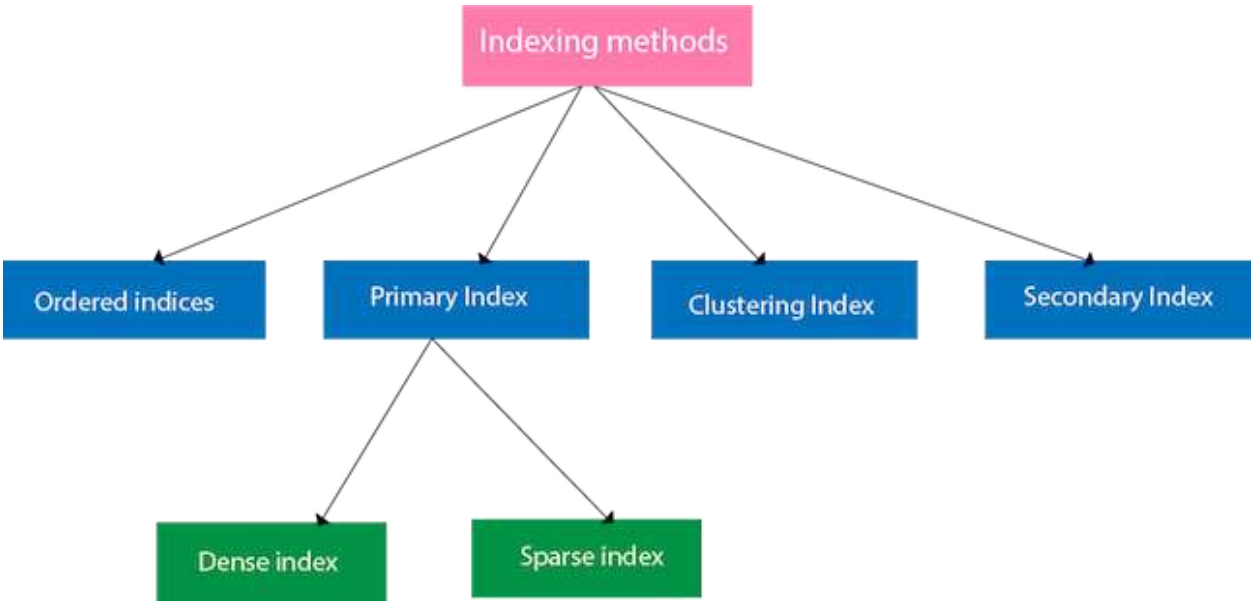
Search key	Data Reference

Fig: Structure of Index

The first column of the database is the search key that contains a copy of the primary key or candidate key of the table. The values of the primary key are stored in sorted order so that the corresponding data can be accessed easily.

The second column of the database is the data reference. It contains a set of pointers holding the address of the disk block where the value of the particular key can be found.

Indexing Methods



Ordered indices

The indices are usually sorted to make searching faster. The indices which are sorted are known as ordered indices.

Example: Suppose we have an employee table with thousands of record and each of which is 10 bytes long. If their IDs start with 1, 2, 3....and so on and we have to search student with ID-543.

In the case of a database with no index, we have to search the disk block from starting till it reaches 543. The DBMS will read the record after reading 543*10=5430 bytes.

In the case of an index, we will search using indexes and the DBMS will read the record after reading 542*2= 1084 bytes which are very less compared to the previous case.

Primary Index

If the index is created on the basis of the primary key of the table, then it is known as primary indexing. These primary keys are unique to each record and contain 1:1 relation between the records.

As primary keys are stored in sorted order, the performance of the searching operation is quite efficient.

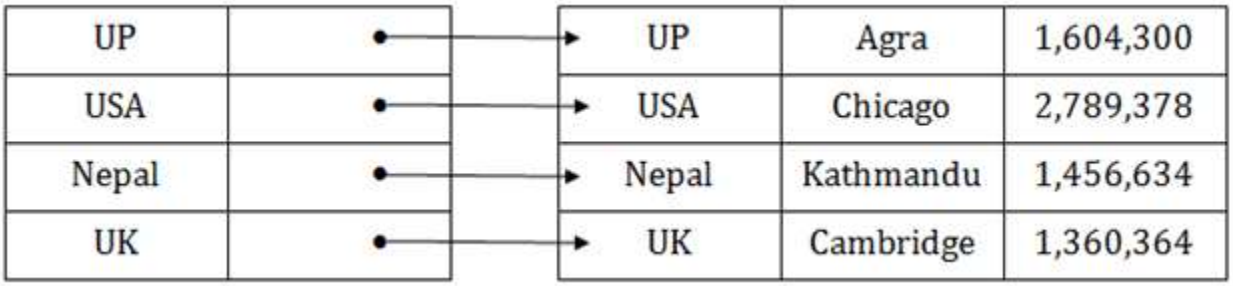
The primary index can be classified into two types: Dense index and Sparse index.

Dense index

The dense index contains an index record for every search key value in the data file. It makes searching faster.

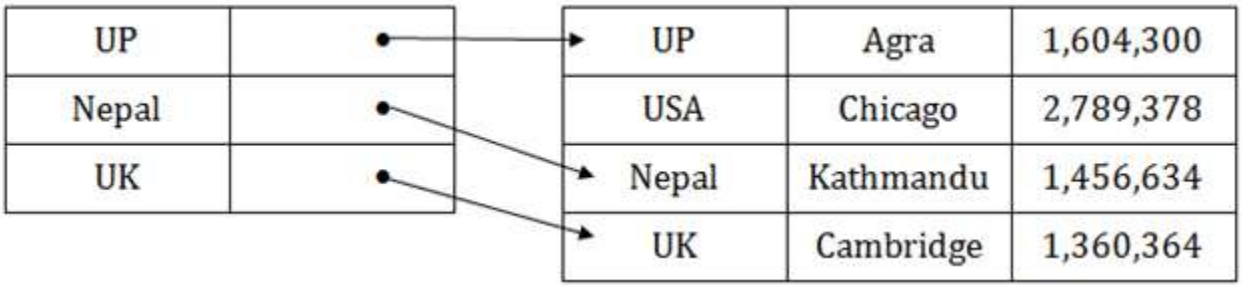
In this, the number of records in the index table is same as the number of records in the main table.

It needs more space to store index record itself. The index records have the search key and a pointer to the actual record on the disk.



Sparse index

- In the data file, index record appears only for a few items. Each item points to a block.
- In this, instead of pointing to each record in the main table, the index points to the records in the main table in a gap.



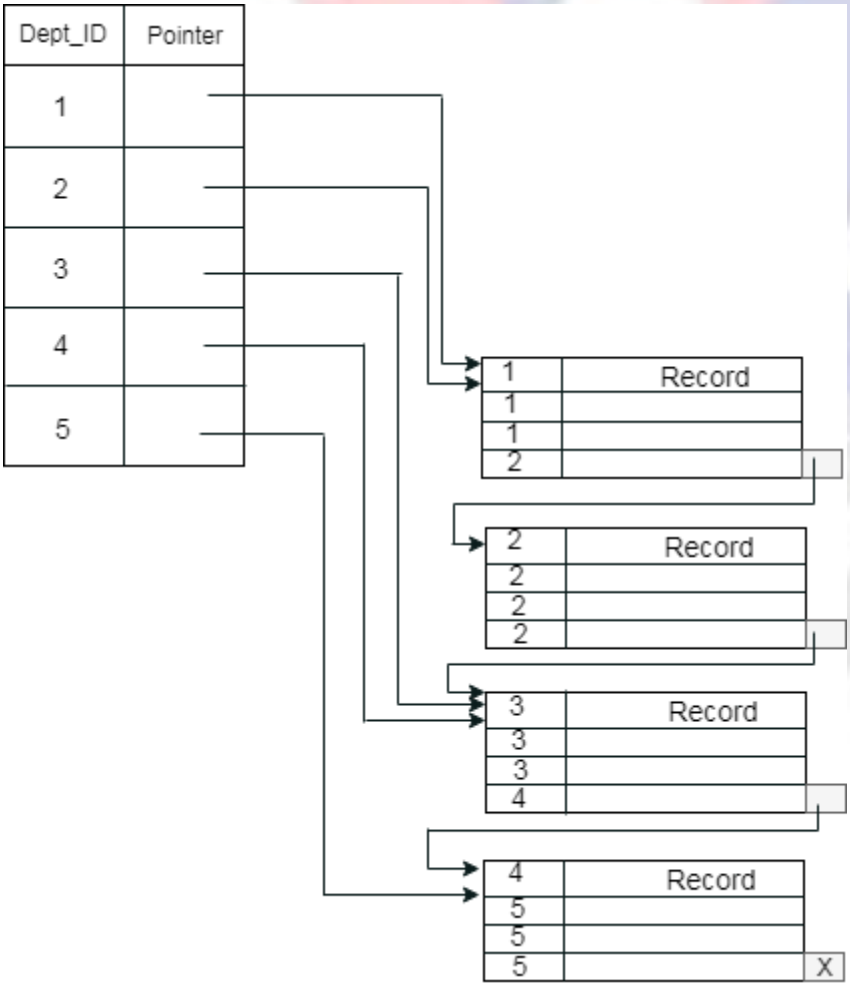
Clustering Index

A clustered index can be defined as an ordered data file. Sometimes the index is created on non-primary key columns which may not be unique for each record.

In this case, to identify the record faster, we will group two or more columns to get the unique value and create index out of them. This method is called a clustering index.

The records which have similar characteristics are grouped, and indexes are created for these group.

Example: suppose a company contains several employees in each department. Suppose we use a clustering index, where all employees which belong to the same Dept_ID are considered within a single cluster, and index pointers point to the cluster as a whole. Here Dept_Id is a non-unique key.

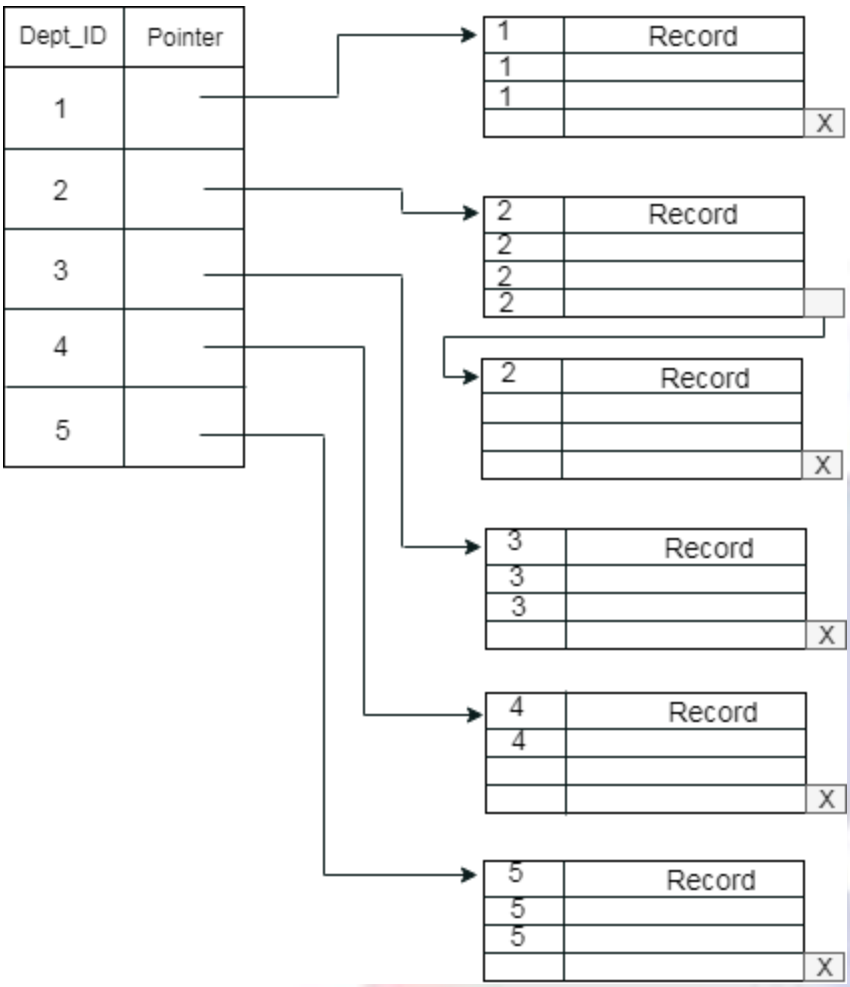


The previous schema is little confusing because one disk block is shared by records which belong to the different cluster. If we use separate disk block for separate clusters, then it is called better technique.

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.

In secondary indexing, to reduce the size of mapping, another level of indexing is introduced. In this method, the huge range for the columns is selected initially so that the mapping size of the first level becomes small. Then each range is further divided into smaller ranges. The mapping of the first level is stored in the primary memory, so that address fetch is faster. The mapping of the second level and actual data are stored in the secondary memory (hard disk).



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.

Indexing in Databases

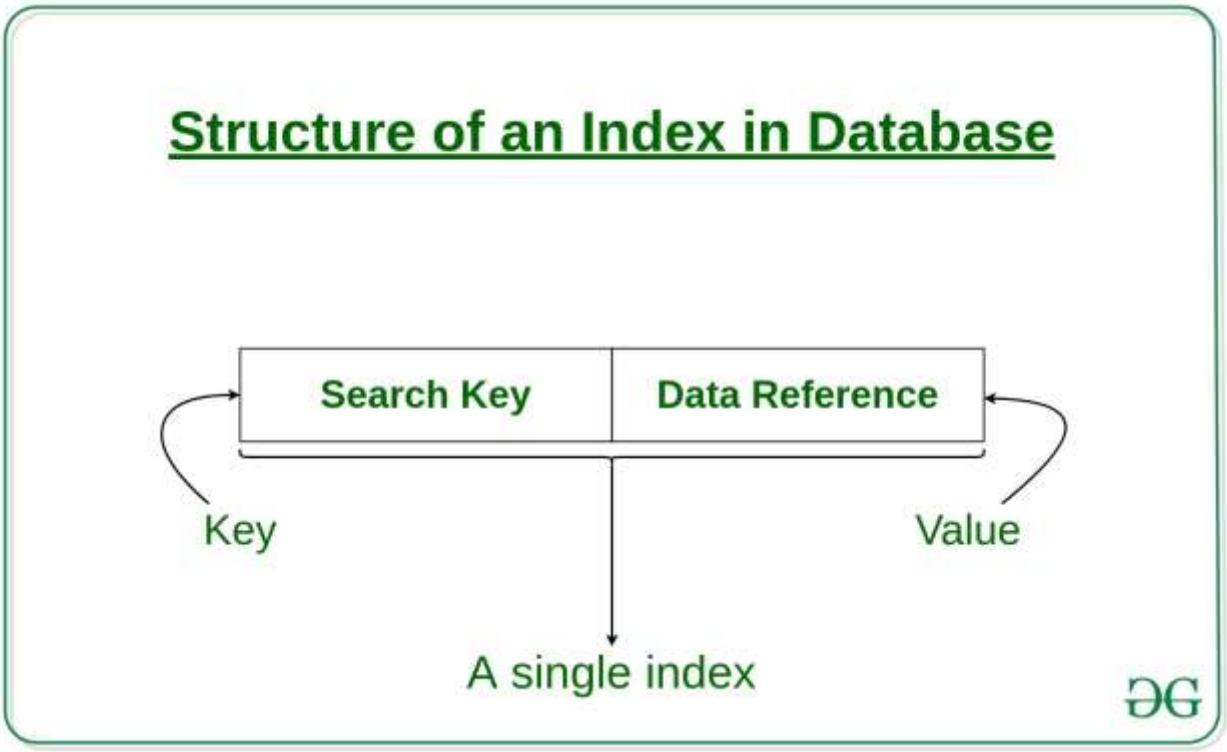
Indexing is a way to optimize the performance of a database by minimizing the number of disk accesses required when a query is processed. It is a data structure technique which is used to quickly locate and access the data in a database.

Indexes are created using a few database columns.

The first column is the Search key that contains a copy of the primary key or candidate key of the table. These values are stored in sorted order so that the corresponding data can be accessed quickly.

Note: The data may or may not be stored in sorted order.

The second column is the Data Reference or Pointer which contains a set of pointers holding the address of the disk block where that particular key value can be found.



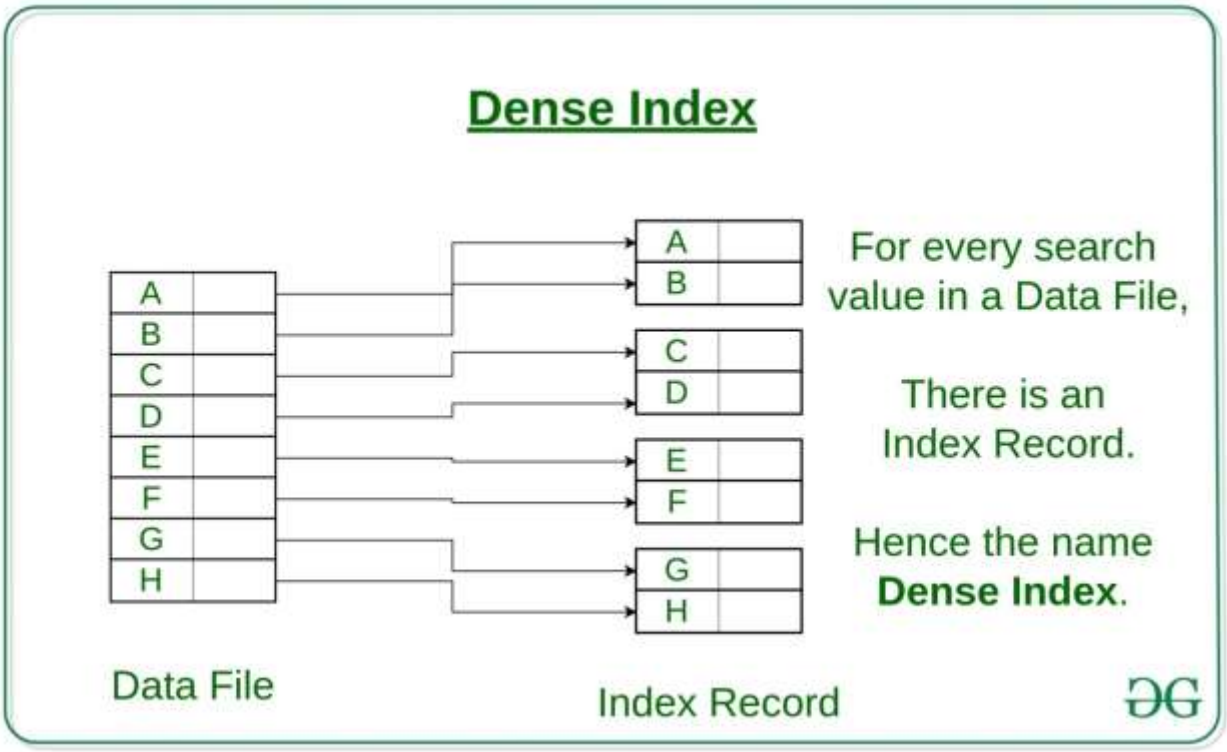
The indexing has various attributes:

- **Access Types**: This refers to the type of access such as value based search, range access, etc.
- **Access Time**: It refers to the time needed to find particular data element or set of elements.
- **Insertion Time**: It refers to the time taken to find the appropriate space and insert a new data.
- **Deletion Time**: Time taken to find an item and delete it as well as update the index structure.
- **Space Overhead**: It refers to the additional space required by the index.

In general, there are two types of file organization mechanism which are followed by the indexing methods to store the data:

Sequential File Organization or Ordered Index File: In this, the indices are based on a sorted ordering of the values. These are generally fast and a more traditional type of storing mechanism. These Ordered or Sequential file organization might store the data in a dense or sparse format:

- **Dense Index:**
    - For every search key value in the data file, there is an index record.
    - This record contains the search key and also a reference to the first data record with that search key value.

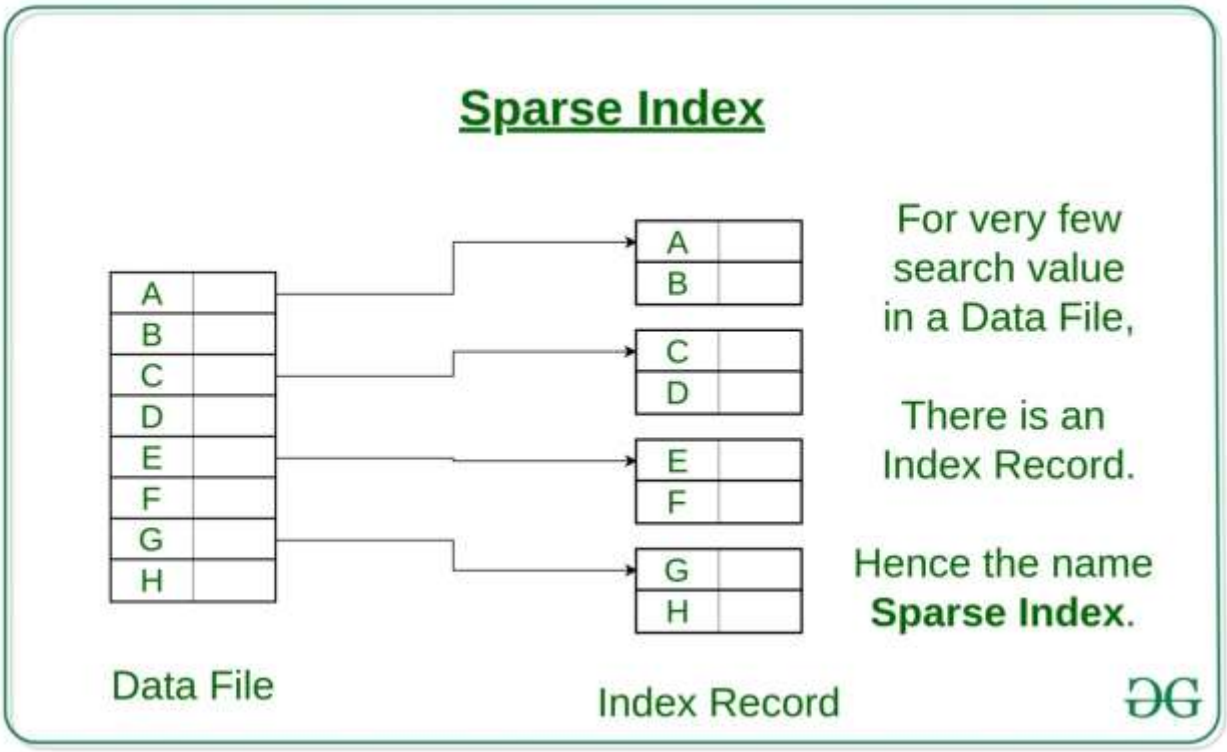


Sparse Index:

The index record appears only for a few items in the data file. Each item points to a block as shown.

To locate a record, we find the index record with the largest search key value less than or equal to the search key value we are looking for.

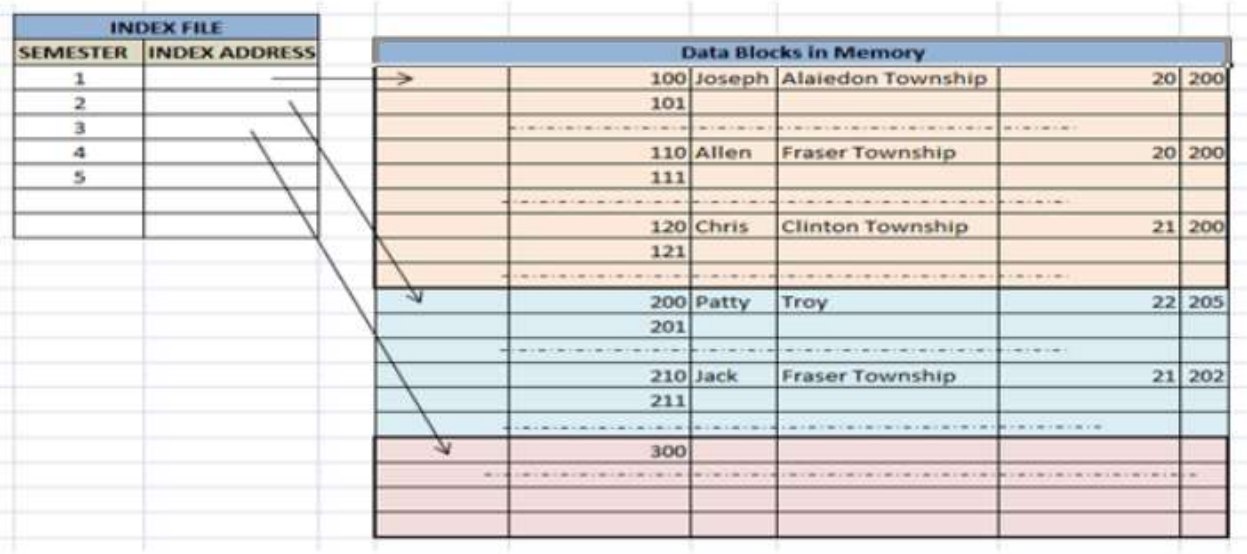
We start at that record pointed to by the index record, and proceed along with the pointers in the file (that is, sequentially) until we find the desired record.



There are primarily three methods of indexing:

- Clustered Indexing
- Non-Clustered or Secondary Indexing
- Multilevel Indexing

1. **Clustered Indexing**
   When more than two records are stored in the same file these types of storing known as cluster indexing. By using the cluster indexing we can reduce the cost of searching reason being multiple records related to the same thing are stored at one place and it also gives the frequent joing of more than two tables(records).
   Clustering index is defined on an ordered data file. The data file is ordered on a non-key field. In some cases, the index is created on non-primary key columns which may not be unique for each record. In such cases, in order to identify the records faster, we will group two or more columns together to get the unique values and create index out of them. This method is known as the clustering index. Basically, records with similar characteristics are grouped together and indexes are created for these groups.
   For example, students studying in each semester are grouped together. i.e. 1st Semester students, 2nd semester students, 3rd semester students etc are grouped.



1. Clustered index sorted according to first name (Search key)

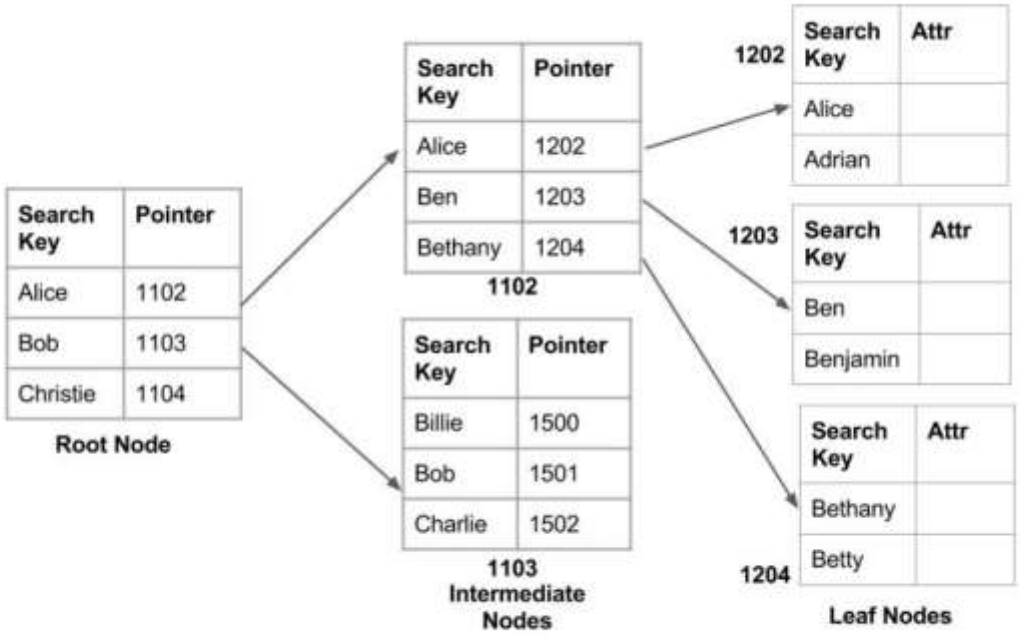
Primary Indexing:

This is a type of Clustered Indexing wherein the data is sorted according to the search key and the primary key of the database table is used to create the index. It is a default format of indexing where it induces sequential file organization. As primary keys are unique and are stored in a sorted manner, the performance of the searching operation is quite efficient.

2. **Non-clustered or Secondary Indexing**

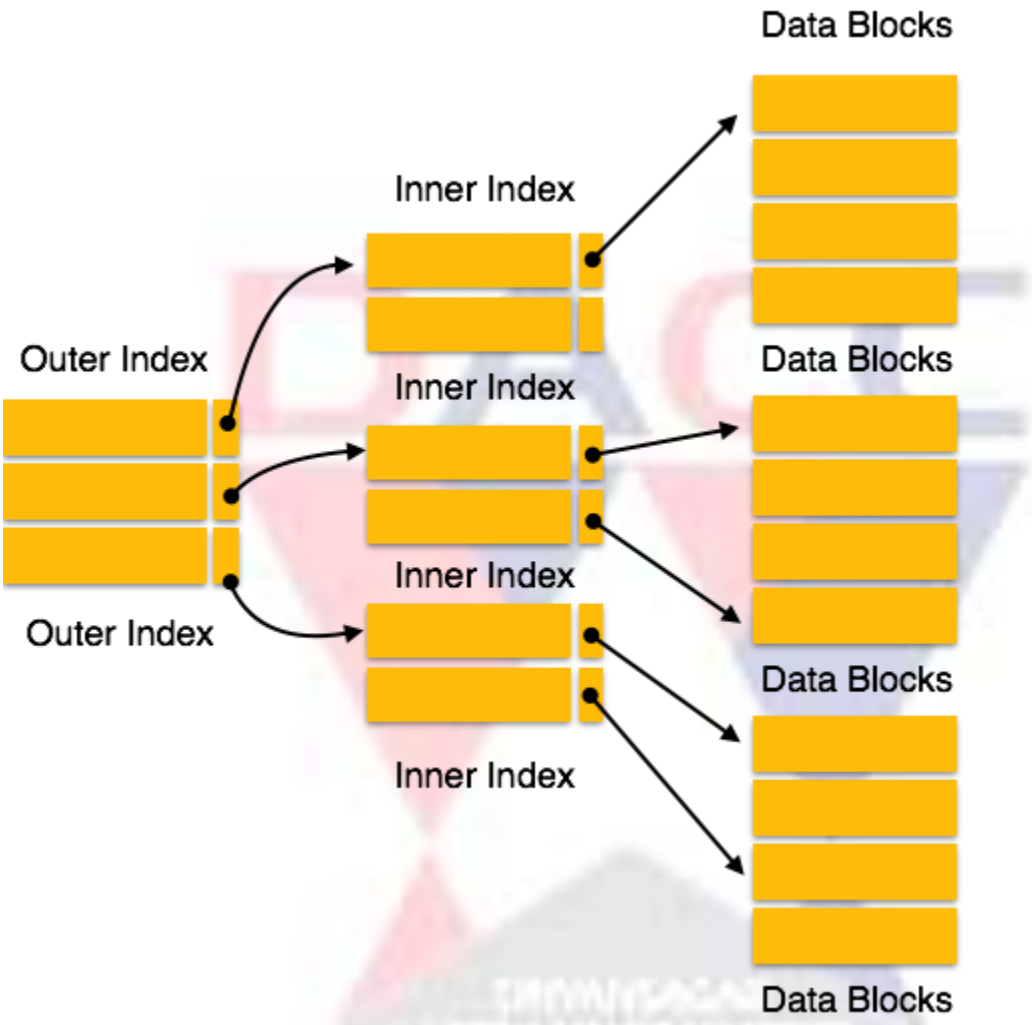
   A non clustered index just tells us where the data lies, i.e. it gives us a list of virtual pointers or references to the location where the data is actually stored. Data is not physically stored in the order of the index. Instead, data is present in leaf nodes. For eg. the contents page of a book. Each entry gives us the page number or location of the information stored. The actual data here(information on each page of the book) is not organized but we have an ordered reference(contents page) to where the data points actually lie. We can have only dense ordering in the non-clustered index as sparse ordering is not possible because data is not physically organized accordingly.

   It requires more time as compared to the clustered index because some amount of extra work is done in order to extract the data by further following the pointer. In the case of a clustered index, data is directly present in front of the index.



Non clustered index

1. **Multilevel Indexing**
   With the growth of the size of the database, indices also grow. As the index is stored in the main memory, a single-level index might become too large a size to store with multiple disk accesses. The multilevel indexing segregates the main block into various smaller blocks so that the same can stored in a single block. The outer blocks are divided into inner blocks which in turn are pointed to the data blocks. This can be easily stored in the main memory with fewer overheads.



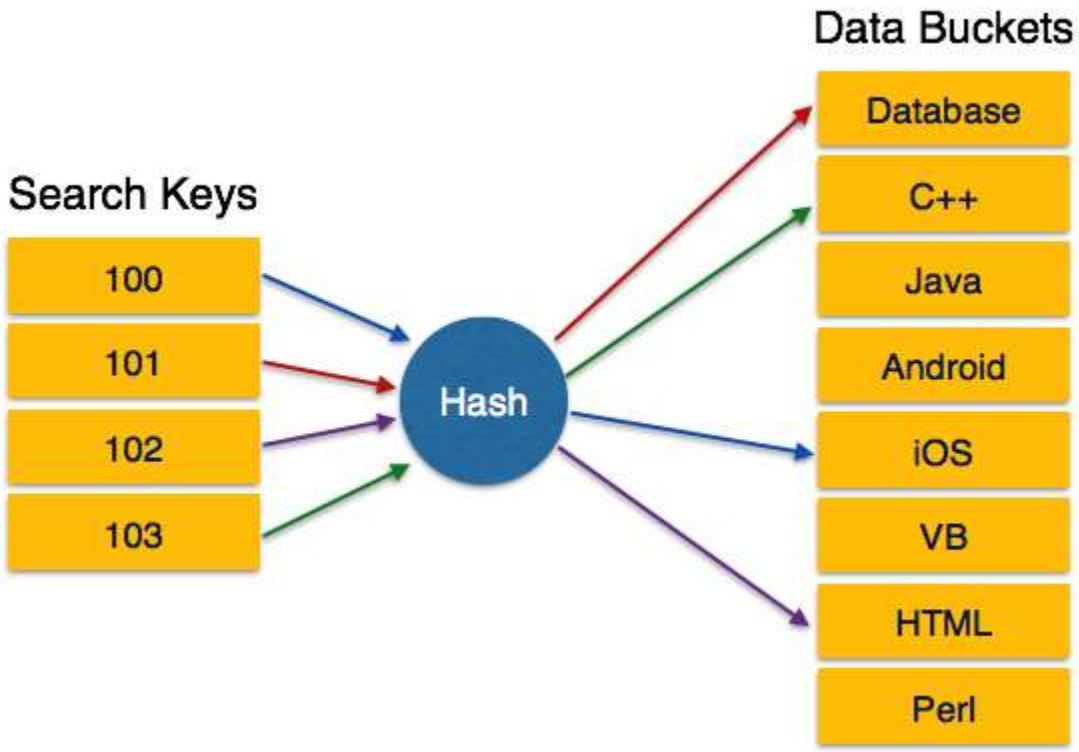
Multi-level Index helps in breaking down the index into several smaller indices in order to make the outermost level so small that it can be saved in a single disk block, which can easily be accommodated anywhere in the main memory.

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.



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.

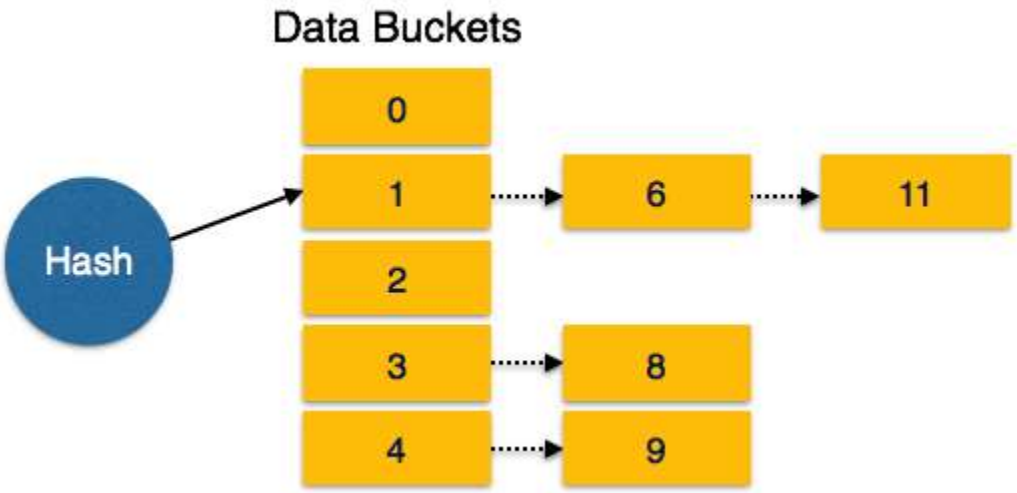
  Bucket address = h(K)

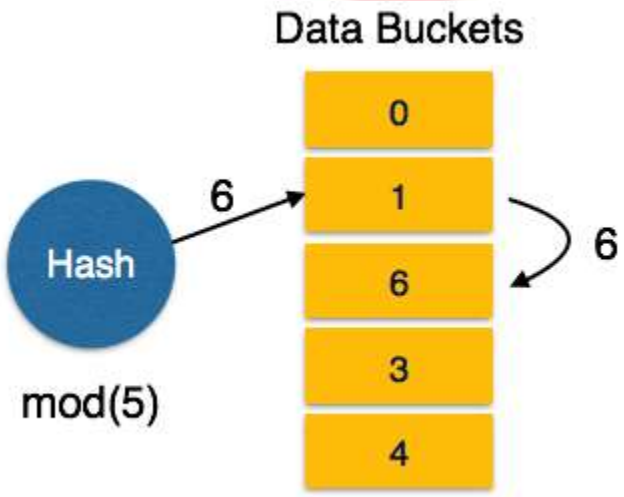
- **Search** − When a record needs to be retrieved, the same hash function can be used to retrieve the address of the bucket where the data is stored.
- **Delete** − This is simply a search followed by a deletion operation.

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**.

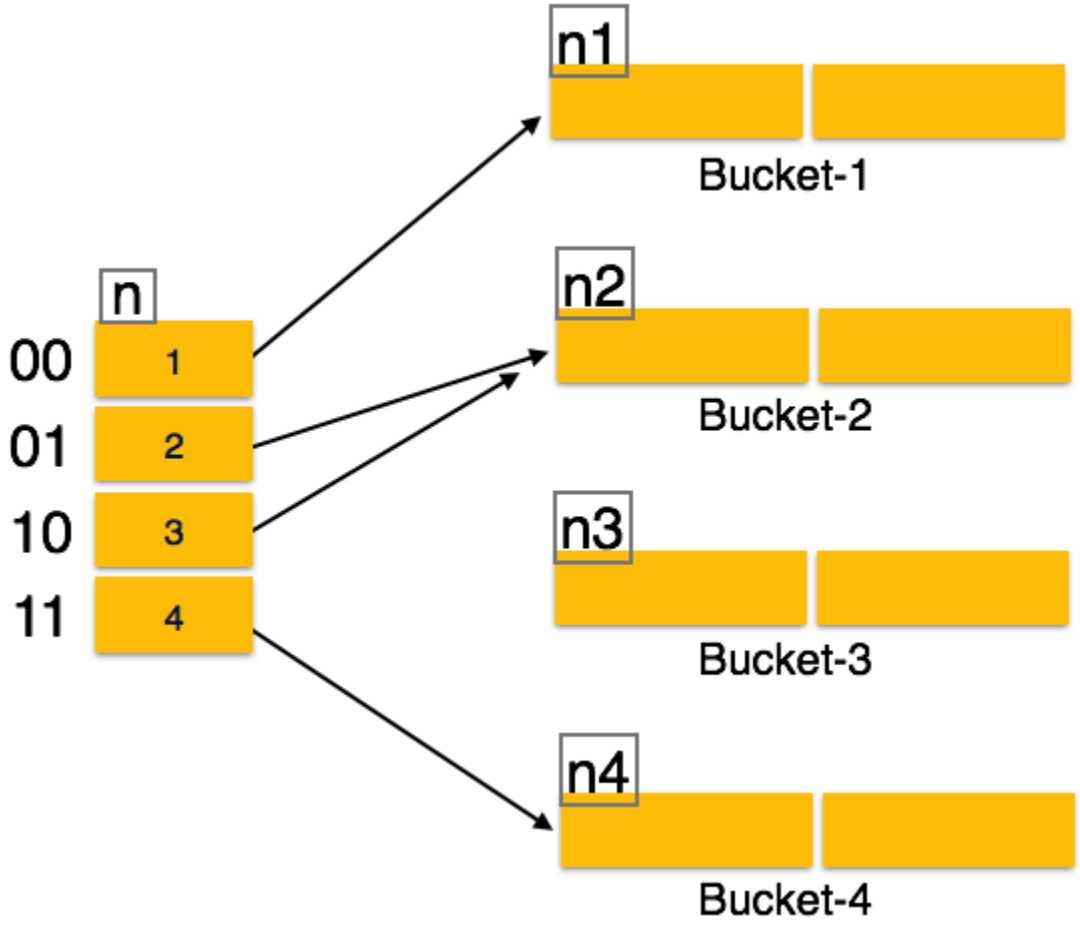




Dynamic Hashing

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.



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.

Operation

- **Querying** − Look at the depth value of the hash index and use those bits to compute the bucket address.
- **Update** − Perform a query as above and update the data.
- **Deletion** − Perform a query to locate the desired data and delete the same.
- **Insertion** − Compute the address of the bucket
    - If the bucket is already full.
        - Add more buckets.
        - Add additional bits to the hash value.
        - Re-compute the hash function.
    - Else
        - Add data to the bucket,
    - If all the buckets are full, perform the remedies of static hashing.

Hashing is not favorable when the data is organized in some ordering and the queries require a range of data. When data is discrete and random, hash performs the best.

Hashing algorithms have high complexity than indexing. All hash operations are done in constant time.

UNIT : 2 –DBMS

What is Database

The database is a collection of inter-related data which is used to retrieve, insert and delete the data efficiently. It is also used to organize the data in the form of a table, schema, views, and reports, etc.

For example: The college Database organizes the data about the admin, staff, students and faculty etc.

Using the database, you can easily retrieve, insert, and delete the information.

Database Management System

- Database management system is a software which is used to manage the database. For example: MySQL, Oracle, etc are a very popular commercial database which is used in different applications.
- DBMS provides an interface to perform various operations like database creation, storing data in it, updating data, creating a table in the database and a lot more.
- It provides protection and security to the database. In the case of multiple users, it also maintains data consistency.

DBMS allows users the following tasks:

- **Data Definition:** It is used for creation, modification, and removal of definition that defines the organization of data in the database.
- **Data Updation:** It is used for the insertion, modification, and deletion of the actual data in the database.
- **Data Retrieval:** It is used to retrieve the data from the database which can be used by applications for various purposes.

- **User Administration:** It is used for registering and monitoring users, maintain data integrity, enforcing data security, dealing with concurrency control, monitoring performance and recovering information corrupted by unexpected failure.

Characteristics of DBMS

- It uses a digital repository established on a server to store and manage the information.
- It can provide a clear and logical view of the process that manipulates data.
- DBMS contains automatic backup and recovery procedures.
- It contains ACID properties which maintain data in a healthy state in case of failure.
- It can reduce the complex relationship between data.
- It is used to support manipulation and processing of data.
- It is used to provide security of data.
- It can view the database from different viewpoints according to the requirements of the user.

Advantages of DBMS

- **Controls database redundancy:** It can control data redundancy because it stores all the data in one single database file and that recorded data is placed in the database.
- **Data sharing:** In DBMS, the authorized users of an organization can share the data among multiple users.
- **Easily Maintenance:** It can be easily maintainable due to the centralized nature of the database system.
- **Reduce time:** It reduces development time and maintenance need.
- **Backup:** It provides backup and recovery subsystems which create automatic backup of data from hardware and software failures and restores the data if required.
- **multiple user interface:** It provides different types of user interfaces like graphical user interfaces, application program interfaces

Disadvantages of DBMS

- **Cost of Hardware and Software:** It requires a high speed of data processor and large memory size to run DBMS software.
- **Size:** It occupies a large space of disks and large memory to run them efficiently.
- **Complexity:** Database system creates additional complexity and requirements.

- **Higher impact of failure:** Failure is highly impacted the database because in most of the organization, all the data stored in a single database and if the database is damaged due to electric failure or database corruption then the data may be lost forever.

DBMS vs. File System

There are following differences between DBMS and File system:

DBMS	File System
DBMS is a collection of data. In DBMS, the user is not required to write the procedures.	**File system is a collection of data. In this system, the user has to write the procedures for managing the database.**
DBMS gives an abstract view of data that hides the details.	**File system provides the detail of the data representation and storage of data.**
DBMS provides a crash recovery mechanism, i.e., DBMS protects the user from the system failure.	**File system doesn't have a crash mechanism, i.e., if the system crashes while entering some data, then the content of the file will lost.**
DBMS provides a good protection mechanism.	**It is very difficult to protect a file under the file system.**
DBMS contains a wide variety of sophisticated techniques to store and retrieve the data.	**File system can't efficiently store and retrieve the data.**
DBMS takes care of Concurrent access of	**In the File system, concurrent access**

data using some form of locking.	**has many problems like redirecting the file while other deleting some information or updating some information.**

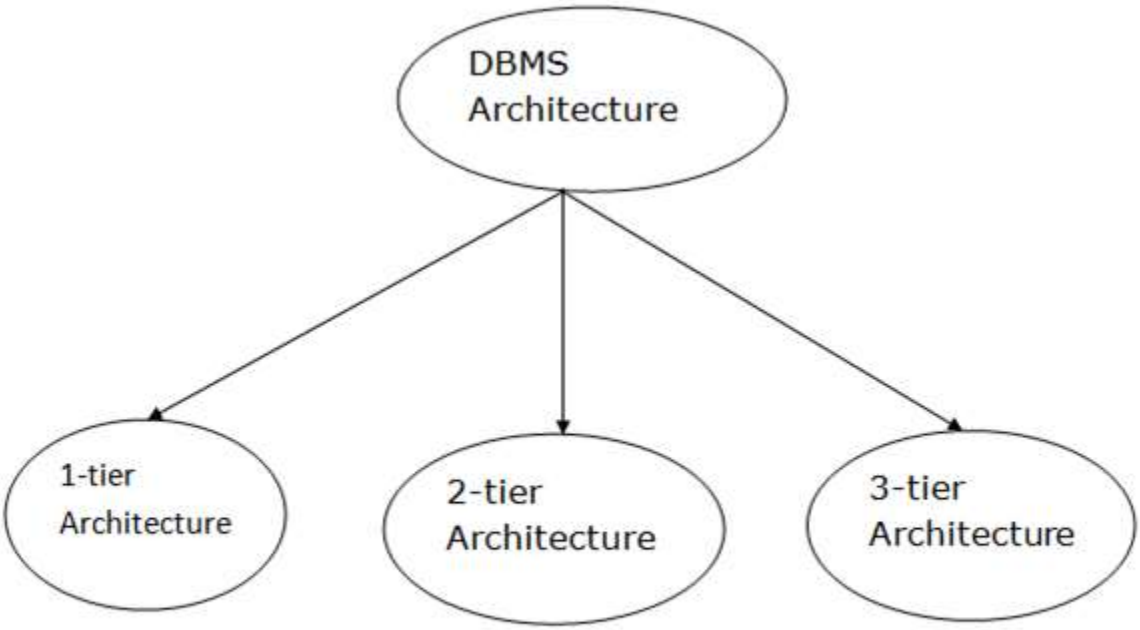
DBMS Architecture

The DBMS design depends upon its architecture. The basic client/server architecture is used to deal with a large number of PCs, web servers, database servers and other components that are connected with networks.

The client/server architecture consists of many PCs and a workstation which are connected via the network.

DBMS architecture depends upon how users are connected to the database to get their request done.

Types of DBMS Architecture



Database architecture can be seen as a single tier or multi-tier. But logically, database architecture is of two types like: 2-tier architecture and 3-tier architecture.

1-Tier Architecture

In this architecture, the database is directly available to the user. It means the user can directly sit on the DBMS and uses it.

Any changes done here will directly be done on the database itself. It doesn't provide a handy tool for end users.

The 1-Tier architecture is used for development of the local application, where programmers can directly communicate with the database for the quick response.

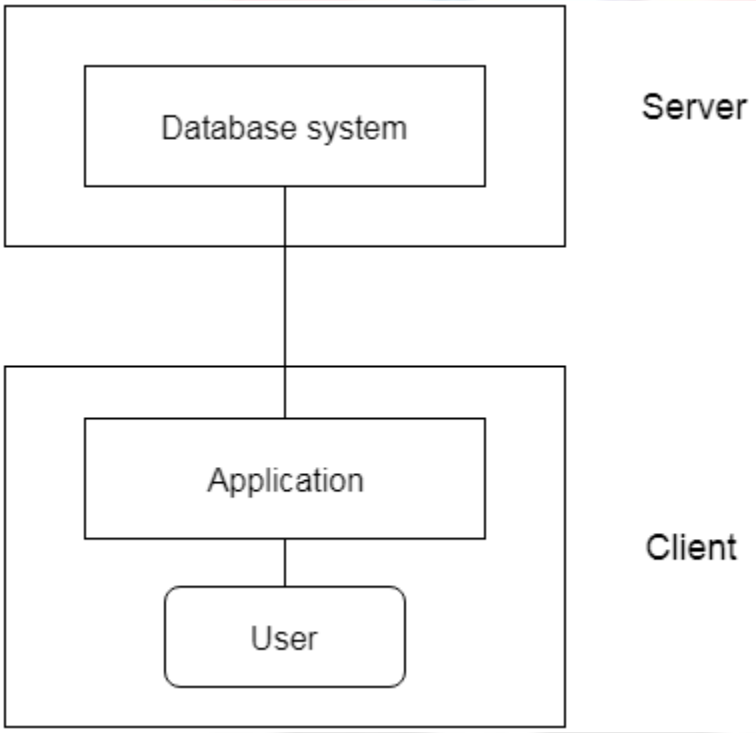
2-Tier Architecture

The 2-Tier architecture is same as basic client-server. In the two-tier architecture, applications on the client end can directly communicate with the database at the server side. For this interaction, API's like: ODBC, JDBC are used.

The user interfaces and application programs are run on the client-side.

The server side is responsible to provide the functionalities like: query processing and transaction management.

To communicate with the DBMS, client-side application establishes a connection with the server side.



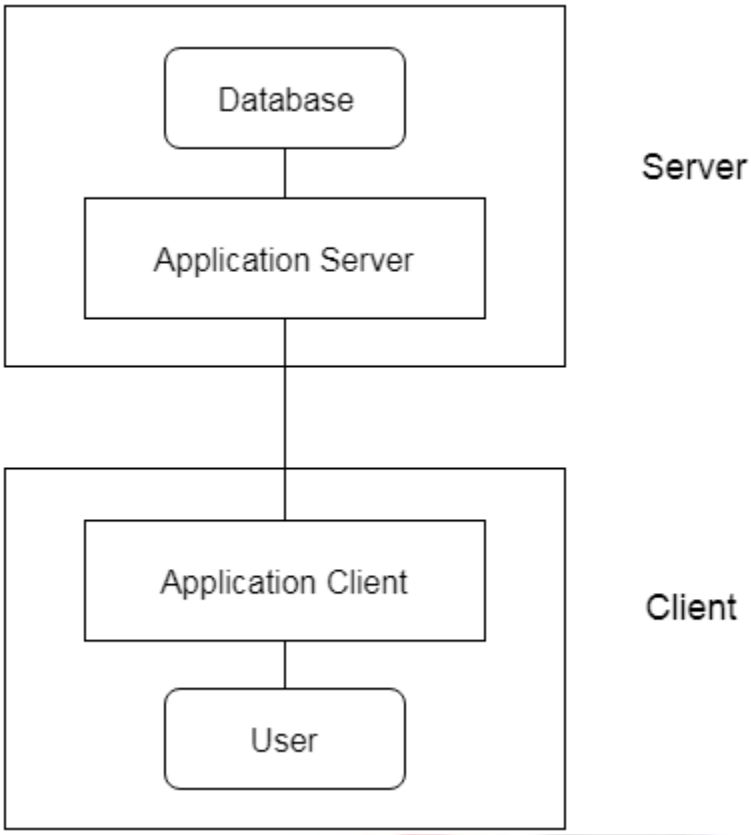
3-Tier Architecture

The 3-Tier architecture contains another layer between the client and server. In this architecture, client can't directly communicate with the server.

The application on the client-end interacts with an application server which further communicates with the database system.

End user has no idea about the existence of the database beyond the application server. The database also has no idea about any other user beyond the application.

The 3-Tier architecture is used in case of large web application



Different types of Database Users

Database users are categorized based up on their interaction with the data base.

These are seven types of data base users in DBMS.

Database Administrator (DBA) :

Database Administrator (DBA) is a person/team who defines the schema and also controls the 3 levels of database.

The DBA will then create a new account id and password for the user if he/she need to access the data base.

DBA is also responsible for providing security to the data base and he allows only the authorized users to access/modify the data base.

DBA also monitors the recovery and back up and provide technical support.

The DBA has a DBA account in the DBMS which called a system or superuser account.

DBA repairs damage caused due to hardware and/or software failures**.**

Naive / Parametric End Users :

Parametric End Users are the unsophisticated who don't have any DBMS knowledge but they frequently use the data base applications in their daily life to get the desired results.

For examples, Railway's ticket booking users are naive users. Clerks in any bank is a naive user because they don't have any DBMS knowledge but they still use the database and perform their given task.

System Analyst :

System Analyst is a user who analyzes the requirements of parametric end users. They check whether all the requirements of end users are satisfied.

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.

Application Program :

Application Program are the back end programmers who writes the code for the application programs.They are the computer professionals. These programs could be written in Programming languages such as Visual Basic, Developer, C, FORTRAN, COBOL etc.

Casual Users / Temporary Users :

Casual Users are the users who occasionally use/access the data base but each time when they access the data base they require the new information, for example, Middle or higher level manager.

Various Views Of Data

It refers that how database is actually stored in database, what data and structure of data used by database for data. So describe all this database provides user with views and these are

- **Data abstraction**
- **Instances and schemas.**

Data abstraction

As a data in database are stored with very complex data structure so when user come and want to access any data, he will not be able to acess data if he has go through this data structure. So to simplify the interaction of user and database, dBMS hides some informations which is not of user interest, a this is called data abstraction:- **So developer hides complexity from user and store abstract view of data.**

Data abstraction has three level of abstractions

- **level / internal level**
- **Logical level / conceptual level**
- **view level / external level**

Physical level:- this is the lowest level of data abstraction which describe How data is actual stored in database. This level basically describe the data structure and access path /indexing use for accessing file.

Logical level:- The next level of abstraction describe **what data are stored in the database** and what are the relationship existed among those of data.

View level:- In this level user only interact with database and the complexity remain unview . user see data and there may be many views of one data like chart and graph.

- **Let suppose we have customer information so at physical level this record[customer information] can be described as block of storage.**
- **At the logical level these record can be described as fields and attributes along with their data type and relationship among each other.**
- **At view level user just interact with system with the help of GUI and enter the detail at the screen. User not aware of what and how data is stored.**

What Are Data Models -

Definition 1-

Data Models define how the **logical structure** of Database in Modelled. Database are the fundamental entities to introduce **Abstraction** in DBMS. Data models define the **Data Types, the Constraint and the Relationships** for the description or storage of data respectively.

Definition 2: It is a collection of ' concepts and rules' for implementing " abstraction , instance and schemas ".

Various data models are -

- E-R model- Entity Relationship Model
- Relational Model
- Object Oriented Data Model
- Hierarchical Data Model
- Network Data Model

Data models define how the logical structure of a database is modeled. Data Models are fundamental entities to introduce abstraction in a DBMS. Data models define how data is connected to each other and how they are processed and stored inside the system.

The very first data model could be flat data-models, where all the data used are to be kept in the same plane. Earlier data models were not so scientific, hence they were prone to introduce lots of duplication and update anomalies.

Entity-Relationship Model

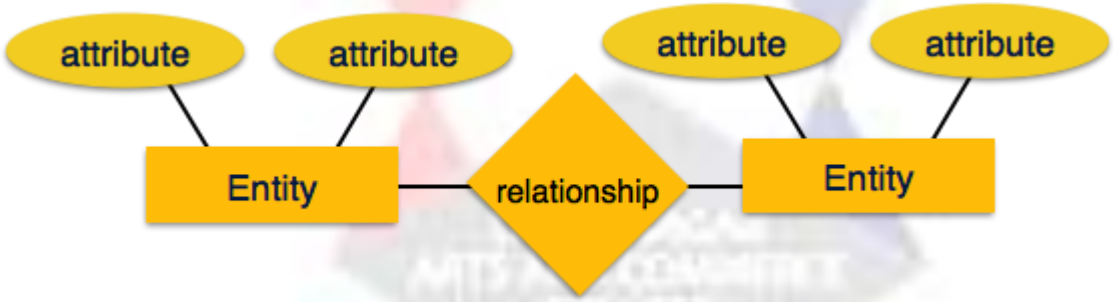
Entity-Relationship (ER) Model is based on the notion of real-world entities and relationships among them. While formulating real-world scenario into the database model, the ER Model creates entity set, relationship set, general attributes and constraints.

ER Model is best used for the conceptual design of a database.

ER Model is based on −

- **Entities** and their *attributes.*
- **Relationships** among entities.

These concepts are explained below.



- **Entity** − An entity in an ER Model is a real-world entity having properties called **attributes**. Every **attribute** is defined by its set of values called **domain**. For example, in a school database, a student is considered as an entity. Student has various attributes like name, age, class, etc.
- **Relationship** − The logical association among entities is called ***relationship***. Relationships are mapped with entities in various ways. Mapping cardinalities define the number of association between two entities.

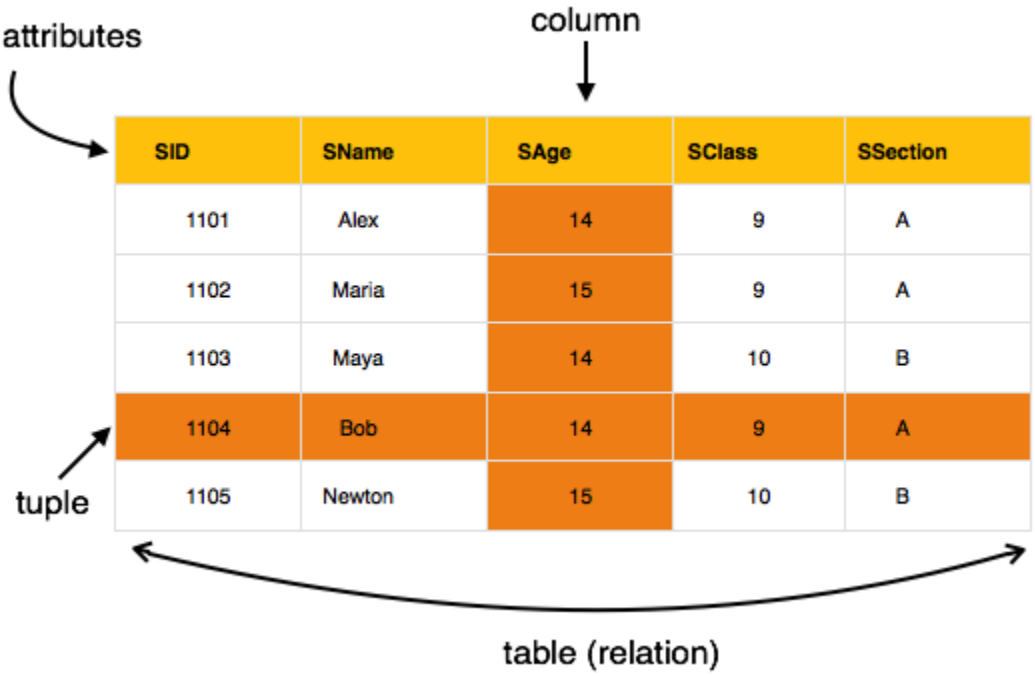
  Mapping cardinalities −

  - one to one
  - one to many
  - many to one

- many to many

Relational Model

The most popular data model in DBMS is the Relational Model. It is more scientific a model than others. This model is based on first-order predicate logic and defines a table as an **n-ary relation**.



SID	SName	SAge	SClass	SSection
1101	Alex	14	9	A
1102	Maria	15	9	A
1103	Maya	14	10	B
1104	Bob	14	9	A
1105	Newton	15	10	B

The main highlights of this model are −

Data is stored in tables called relations.

Relations can be normalized.

In normalized relations, values saved are atomic values.

Each row in a relation contains a unique value.

Each column in a relation contains values from a same domain.

Relational Model concept

Relational model can represent as a table with columns and rows. Each row is known as a tuple. Each table of the column has a name or attribute.

Domain: It contains a set of atomic values that an attribute can take.

Attribute: It contains the name of a column in a particular table. Each attribute Ai must have a domain, dom(Ai)

Relational instance: In the relational database system, the relational instance is represented by a finite set of tuples. Relation instances do not have duplicate tuples.

Relational schema: A relational schema contains the name of the relation and name of all columns or attributes.

Relational key: In the relational key, each row has one or more attributes. It can identify the row in the relation uniquely.

Example: STUDENT Relation

NAME	ROLL_NO	PHONE_NO	ADDRESS	AGE
Ram	14795	7305758992	Noida	24
Shyam	12839	9026288936	Delhi	35
Laxman	33289	8583287182	Gurugram	20
Mahesh	27857	7086819134	Ghaziabad	27
Ganesh	17282	9028 9i3988	Delhi	40

- In the given table, NAME, ROLL_NO, PHONE_NO, ADDRESS, and AGE are the attributes.
- The instance of schema STUDENT has 5 tuples.

- t3 = <Laxman, 33289, 8583287182, Gurugram, 20>

Properties of Relations

- Name of the relation is distinct from all other relations.
- Each relation cell contains exactly one atomic (single) value
- Each attribute contains a distinct name
- Attribute domain has no significance
- tuple has no duplicate value
- Order of tuple can have a different sequence

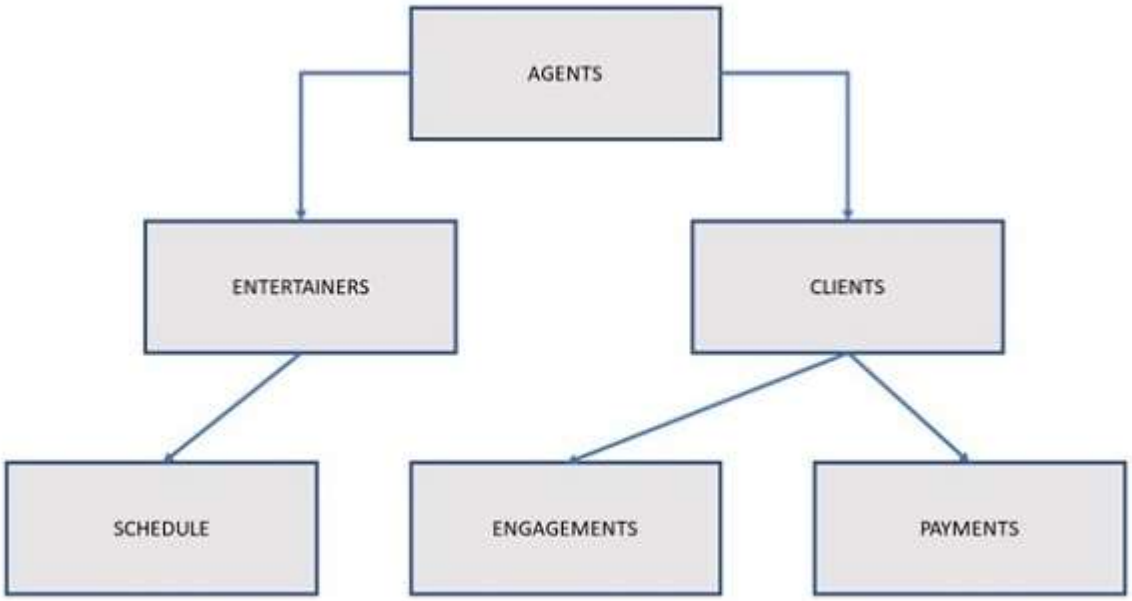
Hierarchical Database Model

A hierarchical model represents the data in a tree-like structure in which there is a single parent for each record. To maintain order there is a sort field which keeps sibling nodes into a recorded manner. These types of models are designed basically for the early mainframe database management systems, like the Information Management System (IMS) by IBM.

This model structure allows the one-to-one and a one-to-many relationship between two/ various types of data. This structure is very helpful in describing many relationships in the real world; table of contents, any nested and sorted information.

The hierarchical structure is used as the physical order of records in storage. One can access the records by navigating down through the data structure using pointers which are combined with sequential accessing. Therefore, the hierarchical structure is not suitable for certain database operations when a full path is not also included for each record.

Data in this type of database is structured hierarchically and is typically developed as an inverted tree. The "root" in the structure is a single table in the database and other tables act as the branches flowing from the root. The diagram below shows a typical hierarchical database structure.



Agents Database

In the above diagram, an agent books several entertainers, and each entertainer, in return has his/her own schedule. It is the duty of an agent to maintain several clients whose entertainment needs are to be met. A client books engagement through the agent and makes payments to the agent for his services.

A relationship in this database model is represented by the term parent/child. A parent table can be linked with one or more child tables in this type of relationship, but a single child table can be linked with only one parent table. The tables are explicitly linked via a pointer/index or by the physical arrangement of the records within the tables.

A user can access the data by starting at the root table and working down through the tree to the target data. the user must be familiar with the structure of the database to access the data without any complexity.

Advantages

A user can retrieve data very quickly due to the presence of explicit links between the table structures.

The referential integrity is built in and automatically enforced due to which a record in a child table must be linked to an existing record in a parent table, along with that if a record deleted in the parent table then that will cause all associated records in the child table to be deleted as well.

Disadvantages

When a user needs to store a record in a child table that is currently unrelated to any record in a parent table, it gets difficulty in recording and user must record an additional entry in the parent table.

This type of database cannot support complex relationships, and there is also a problem of redundancy, which can result in producing inaccurate information due to the inconsistent recording of data at various sites.

Consider an example using the database diagram shown in the previous diagram. A user cannot enter a new record for the entertainer in the Entertainers table until the entertainer is assigned to a specific agent in the Agents table since a record in a child table (Entertainers) must be related to a record in the parent table (Agents). Therefore, this type of database suffers from the problem of redundant data. For example, if there is a many-to-many relationship between clients and entertainers; an entertainer will perform for many clients, and a client will hire many entertainers. This type of relationship in a hierarchical database cannot easily

model, so developers must introduce redundant data into both the Schedule and Engagements tables.

The Schedule table will now have client data which contains information such as client name, address, and phone number to show for whom and where each entertainer is performing. This data is redundant because it is currently stored also in the Clients table.

The Engagements table will now contain data on entertainers which contains information such as entertainer name, phone number, and type of entertainer to indicate which entertainers are performing for a given client. This data is also redundant because it is currently stored in the Entertainers table.

The problem with this redundancy is that it can result in producing inaccurate information because it opens the possibility of allowing a user to enter a single piece of data inconsistently.

This problem can be solved by creating one hierarchical database specifically for entertainers and another one specifically for agents. The Entertainers database will contain only the data recorded in the Entertainers table, and the revised Agents database will contain the data recorded in Agents, Clients, Payments, and Engagements tables. there is no need of as you can define a logical child relationship between the Engagements table in the Agents database and the Entertainers table in the Entertainers database. With this relationship in place, you can retrieve a variety of information, such as a list of booked entertainers for a given client or a performance schedule for a given entertainer. The below diagram describes the whole picture.

Network Model

The network model is the extension of the hierarchical structure because it allows many-to-many relationships to be managed in a tree-like structure that allows multiple parents.

There are two fundamental concepts of a network model −

Records contain fields which need hierarchical organization.

Sets are used to define one-to-many relationships between records that contain one owner, many members.

A record may act as an owner in any number of sets, and a member in any number of sets.

P.S. Set must not be confused with the mathematical set.

A set is designed with the help of circular linked lists where one record type, the owner of the set also called as a parent, appears once in each circle, and a second record type, also known as the subordinate or child, may appear multiple times in each circle.

A hierarchy is established between any two record types where one type (A) is the owner of another type (B). At the same time, another set can be developed where the latter set (B) is the owner of the former set (A). In this model, ownership is defined by the direction, thus all the sets comprise a general directed graph. Access to records is developed by the indexing structure of circular linked lists.

The network model has the following major features −

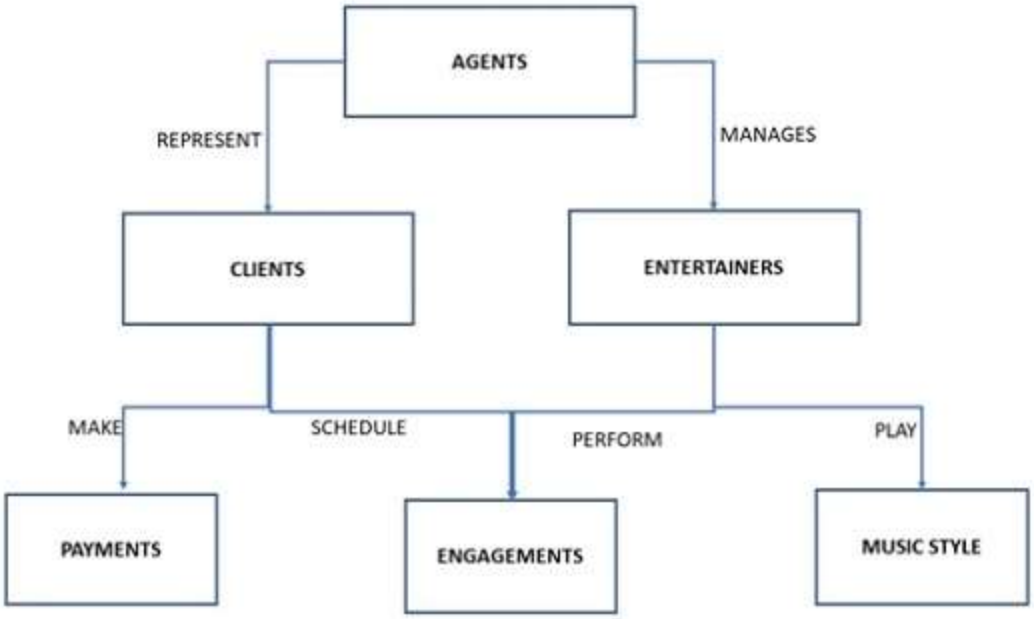
It can represent redundancy in data more efficiently than that in the hierarchical model.

There can be more than one path from a previous node to successor node/s.

The operations of the network model are maintained by indexing structure of linked list (circular) where a program maintains a current position and navigates from one record to another by following the relationships in which the record participates.

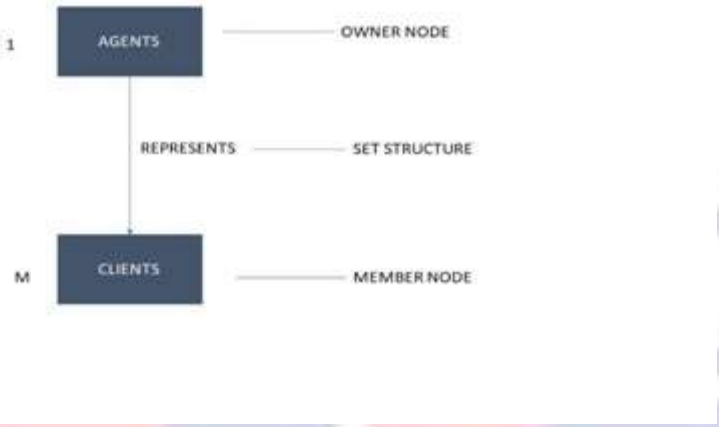
Records can also be located by supplying key values.

The following diagram depicts a network model. An agent represents several clients and manages several entertainers. Each client schedules any number of engagements and makes payments to the agent for his or her services. Each entertainer performs several engagements and may play a variety of musical styles.



A collection of records is represented by a node, and a set structure helps to establish a relationship in a network helps to This development helps to relate a pair of nodes together by using one node as an owner and the other node as a member. A one-to-many relationship is managed by set structure, which means that a record in the owner node can be related to one or more records in the member node, but a single record in the member node is related to only one record in the owner node.

Additionally, a record in the member node cannot exist without being related to an existing record in the owner node. For example, a client must be assigned to an agent, but an agent with no clients can still be listed in the database.



The above diagram shows a diagram of a basic set structure. One or more sets (connections) can be defined between a specific pair of nodes, and a single node can also be involved in other sets with other nodes in the database.

The data can be easily accessed inside a network model with the help of an appropriate set structure. there are no restrictions on choosing the root node, the data can be accessed via any node and running backward or forward with the help of related sets.

For example, when a user wants to find the agent who booked a specific engagement. He/she begins by locating the appropriate engagement record in the ENGAGEMENTS node, and then determines which client "owns" that engagement record via the Schedule set structure. Finally, he/she identifies the agent that "owns" the client record via the Represent set structure.

Advantages

fast data access.

It also allows users to create queries that are more complex than those they created using a hierarchical database. So, a variety of queries can be run over this model.

Disadvantages

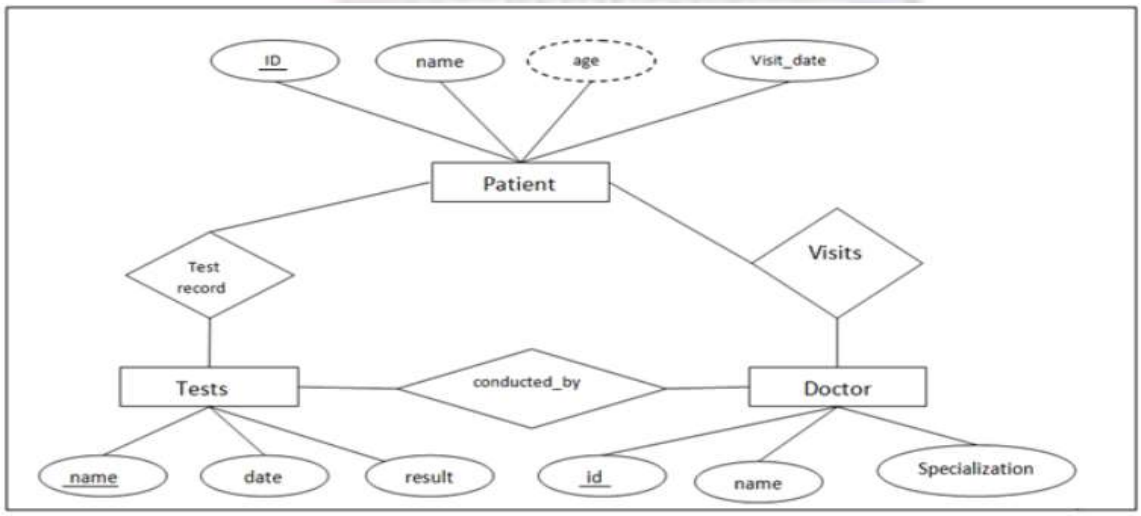
A user must be very familiar with the structure of the database to work through the set structures.

Updating inside this database is a tedious task. One cannot change a set structure without affecting the application programs that use this structure to navigate through the data. If you change a set structure, you must also modify all references made from within the application program to that structure.

Object Oriented Data Model

Object oriented data model is also based on using real life scenarios. In this model, the scenarios are represented as objects. The objects with similar functionalities are grouped together and linked to different other objects.

An Example of the Object Oriented data model is −



- PERSON and EMPLOYEE are 2 objects in this model.
- PERSON has the attributes Name, Address, Age and Phone number.
- EMPLOYEE has the attributes Employee ID, Employee Type and Department ID

The object EMPLOYEE inherits data from the object PERSON i.e the attributes for PERSON would also be available for EMPLOYEE.

An Object relational model is a combination of a Object oriented database model and a Relational database model. So, it supports objects, classes, inheritance etc. just like Object Oriented models and has support for data types, tabular structures etc. like Relational data model.

One of the major goals of Object relational data model is to close the gap between relational databases and the object oriented practises frequently used in many programming languages such as C++, C#, Java etc.

History of Object Relational Data Model

Both Relational data models and Object oriented data models are very useful. But it was felt that they both were lacking in some characteristics and so work was started to build a model that was a combination of them both. Hence, Object relational data model was created as a result of research that was carried out in the 1990's.

Advantages of Object Relational model

The advantages of the Object Relational model are −

Inheritance

The Object Relational data model allows its users to inherit objects, tables etc. so that they can extend their functionality. Inherited objects contains new attributes as well as the attributes that were inherited.

Complex Data Types

Complex data types can be formed using existing data types. This is useful in Object relational data model as complex data types allow better manipulation of the data.

Extensibility

The functionality of the system can be extended in Object relational data model. This can be achieved using complex data types as well as advanced concepts of object oriented model such as inheritance.

Disadvantages of Object Relational model

The object relational data model can get quite complicated and difficult to handle at times as it is a combination of the Object oriented data model and Relational data model and utilizes the functionalities of both of them.

Unit 3 :Relational Model

Relational data model is the primary data model, which is used widely around the world for data storage and processing. This model is simple and it has all the properties and capabilities required to process data with storage efficiency.

Concepts

Tables − In relational data model, relations are saved in the format of Tables. This format stores the relation among entities. A table has rows and columns, where rows represents records and columns represent the attributes.

Tuple − A single row of a table, which contains a single record for that relation is called a tuple.

Relation instance − A finite set of tuples in the relational database system represents relation instance. Relation instances do not have duplicate tuples.

Relation schema − A relation schema describes the relation name (table name), attributes, and their names.

Relation key − Each row has one or more attributes, known as relation key, which can identify the row in the relation (table) uniquely.

Attribute domain − Every attribute has some pre-defined value scope, known as attribute domain.

Constraints

Every relation has some conditions that must hold for it to be a valid relation. These conditions are called Relational Integrity Constraints. There are three main integrity constraints −

Key constraints

Domain constraints

Referential integrity constraints

Key Constraints

There must be at least one minimal subset of attributes in the relation, which can identify a tuple uniquely. This minimal subset of attributes is called key for that relation. If there are more than one such minimal subsets, these are called candidate keys.

Key constraints force that −

in a relation with a key attribute, no two tuples can have identical values for key attributes.

a key attribute can not have NULL values.

Key constraints are also referred to as Entity Constraints.

Domain Constraints

Attributes have specific values in real-world scenario. For example, age can only be a positive integer. The same constraints have been tried to employ on the attributes of a relation. Every attribute is bound to have a specific range of values. For example, age cannot be less than zero and telephone numbers cannot contain a digit outside 0-9.

Referential integrity Constraints

Referential integrity constraints work on the concept of Foreign Keys. A foreign key is a key attribute of a relation that can be referred in other relation.

Referential integrity constraint states that if a relation refers to a key attribute of a different or same relation, then that key element must exist.

Relational database systems are expected to be equipped with a query language that can assist its users to query the database instances. There are two kinds of query languages − relational algebra and relational calculus.

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 of relational algebra are as follows −

- Select
- Project
- Union
- Set different
- Cartesian product
- Rename

We will discuss all these operations in the following sections.

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 − $\prod_{A1}, A_2, A_n (r)$

Where A_1, A_2 , A_n are attribute names of relation **r**.

Duplicate rows are automatically eliminated, as relation is a set.

For example −

$\prod_{subject, author}$ (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.

$\prod_{author}$ (Books) ∪ $\prod_{author}$ (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**.

$\prod_{author}$ (Books) − $\prod_{author}$ (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}

$\sigma_{author = 'tutorialspoint'}$(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

Relational Calculus

In contrast to Relational Algebra, Relational Calculus is a non-procedural query language, that is, it tells what to do but never explains how to do it.

Relational calculus exists in two forms −

Tuple Relational Calculus (TRC)

Filtering variable ranges over tuples

Notation − {T | Condition}

Returns all tuples T that satisfies a condition.

For example −

{ T.name | Author(T) AND T.article = 'database' }

Output − Returns tuples with 'name' from Author who has written article on 'database'.

TRC can be quantified. We can use Existential (∃) and Universal Quantifiers (∀).

For example −

{ R| ∃T ∈ Authors(T.article='database' AND R.name=T.name)}

Output − The above query will yield the same result as the previous one.

Domain Relational Calculus (DRC)

In DRC, the filtering variable uses the domain of attributes instead of entire tuple values (as done in TRC, mentioned above).

Notation −

{ a_1, a_2, a_3, ..., a_n | P (a_1, a_2, a_3, ... ,a_n)}

Where a1, a2 are attributes and **P** stands for formulae built by inner attributes.

For example −

{< article, page, subject > | ∈ TutorialsPoint ∧ subject = 'database'}

Output − Yields Article, Page, and Subject from the relation TutorialsPoint, where subject is database.

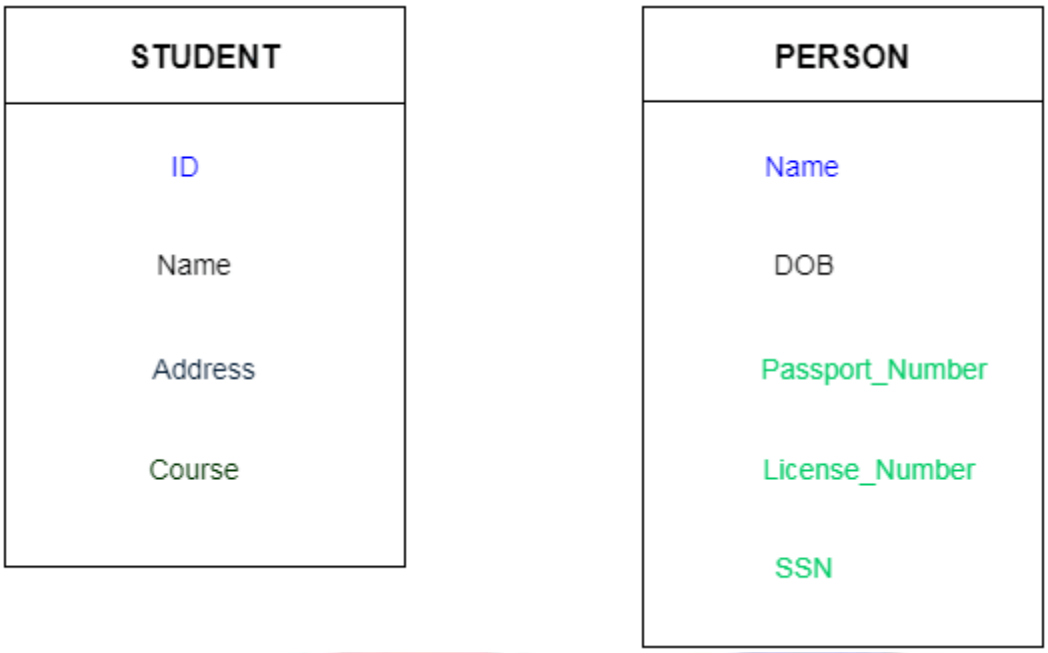
Just like TRC, DRC can also be written using existential and universal quantifiers. DRC also involves relational operators.

The expression power of Tuple Relation Calculus and Domain Relation Calculus is equivalent to Relational Algebra.

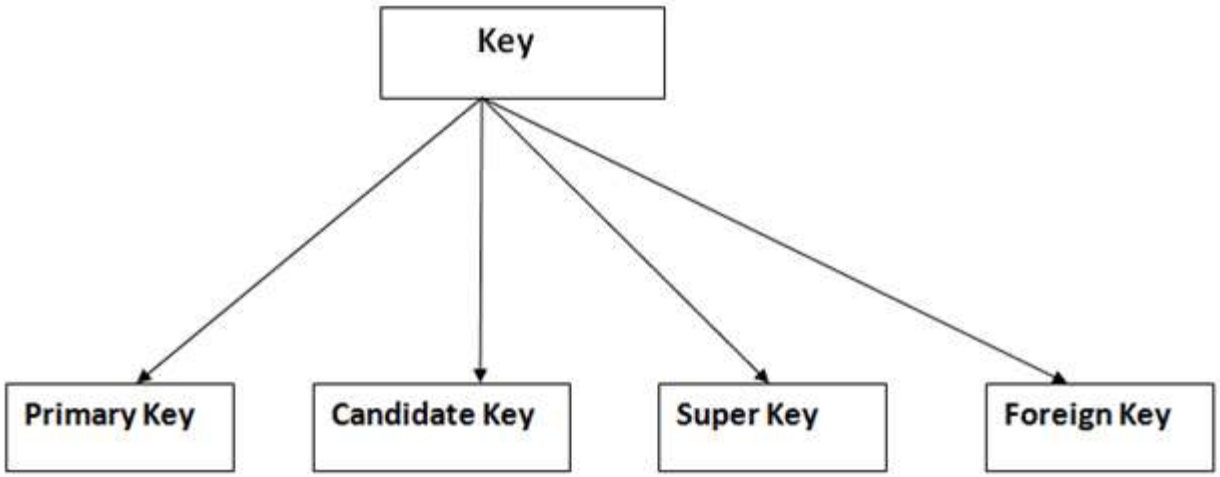
Keys

- Keys play an important role in the relational database.
- It is used to uniquely identify any record or row of data from the table. It is also used to establish and identify relationships between tables.

For example: In Student table, ID is used as a key because it is unique for each student. In PERSON table, passport_number, license_number, SSN are keys since they are unique for each person.

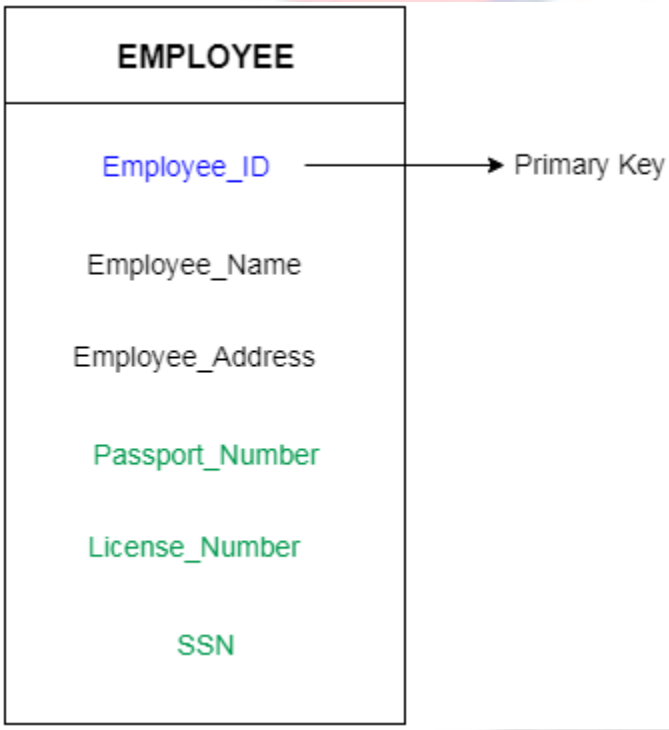

STUDENT
ID
Name
Address
Course
PERSON
Name
DOB
Passport_Number
License_Number
SSN

Types of key:


Key
Primary Key
Candidate Key
Super Key
Foreign Key

1. Primary key

- It is the first key which is used to identify one and only one instance of an entity uniquely. An entity can contain multiple keys as we saw in PERSON table. The key which is most suitable from those lists become a primary key.
- In the EMPLOYEE table, ID can be primary key since it is unique for each employee. In the EMPLOYEE table, we can even select License_Number and Passport_Number as primary key since they are also unique.
- For each entity, selection of the primary key is based on requirement and developers.

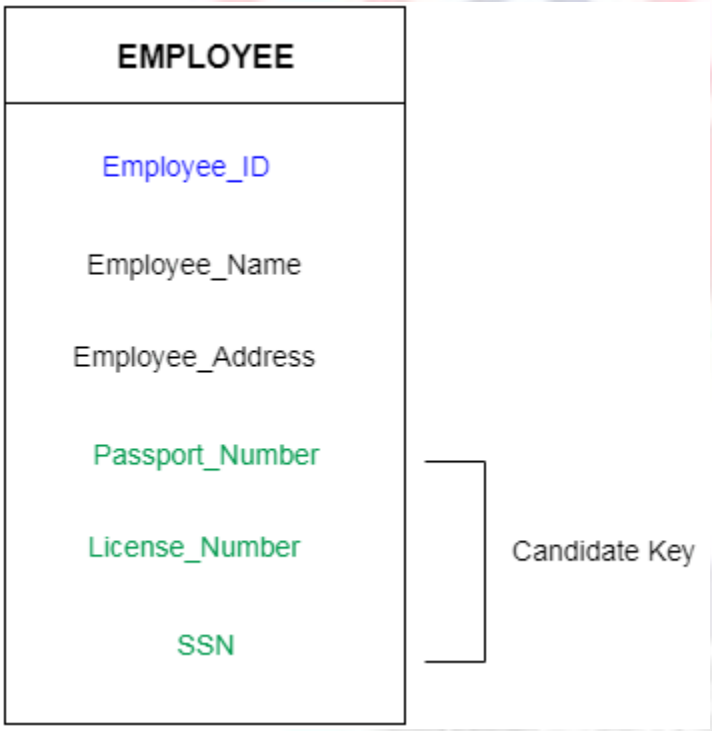


2. Candidate key

- A candidate key is an attribute or set of an attribute which can uniquely identify a tuple.

- The remaining attributes except for primary key are considered as a candidate key. The candidate keys are as strong as the primary key.

For example: In the EMPLOYEE table, id is best suited for the primary key. Rest of the attributes like SSN, Passport_Number, and License_Number, etc. are considered as a candidate key.



3. Super Key

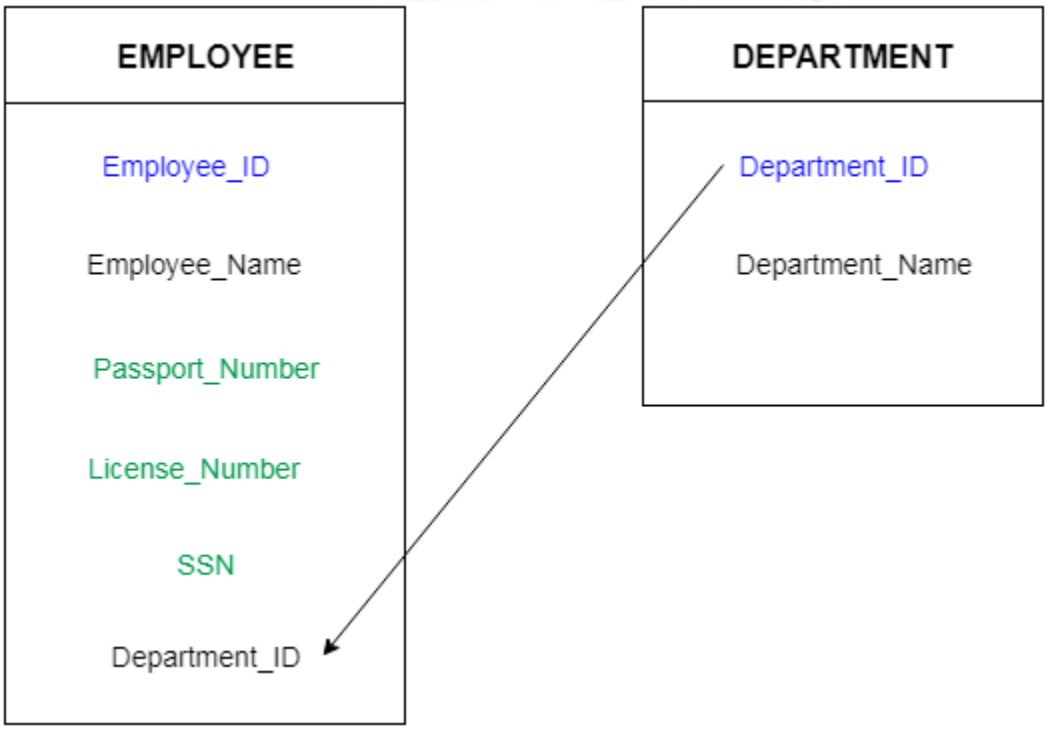
Super key is a set of an attribute which can uniquely identify a tuple. Super key is a superset of a candidate key.

For example: In the above EMPLOYEE table, for(EMPLOEE_ID, EMPLOYEE_NAME) the name of two employees can be the same, but their EMPLYEE_ID can't be the same. Hence, this combination can also be a key.

The super key would be EMPLOYEE-ID, (EMPLOYEE_ID, EMPLOYEE-NAME), etc.

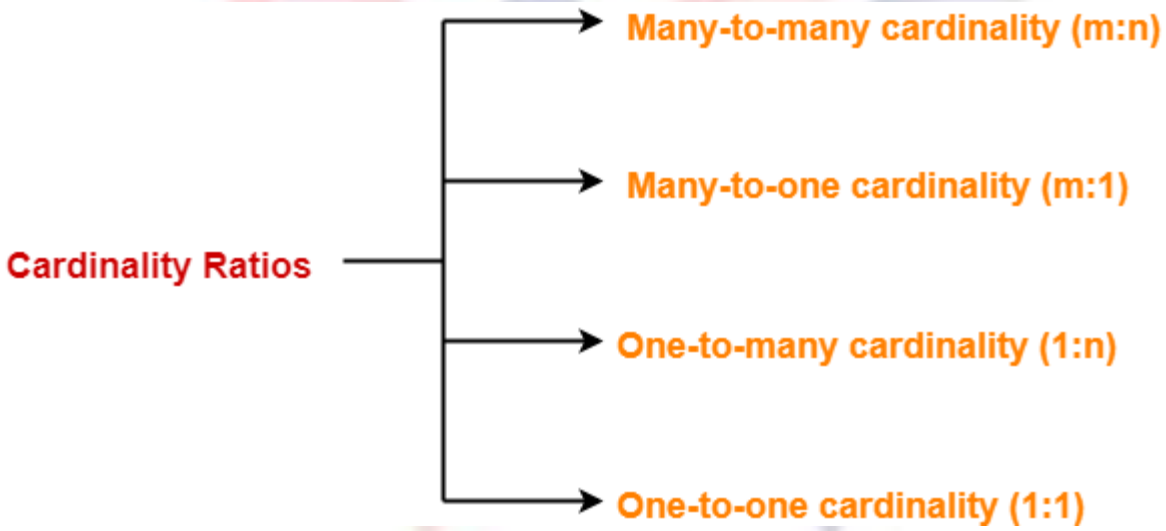
4. Foreign key

- Foreign keys are the column of the table which is used to point to the primary key of another table.
- In a company, every employee works in a specific department, and employee and department are two different entities. So we can't store the information of the department in the employee table. That's why we link these two tables through the primary key of one table.
- We add the primary key of the DEPARTMENT table, Department_Id as a new attribute in the EMPLOYEE table.
- Now in the EMPLOYEE table, Department_Id is the foreign key, and both the tables are related.



Types of Cardinality Ratios-

There are 4 types of cardinality ratios-



1. Many-to-Many cardinality (m:n)
2. Many-to-One cardinality (m:1)
3. One-to-Many cardinality (1:n)
4. One-to-One cardinality (1:1)

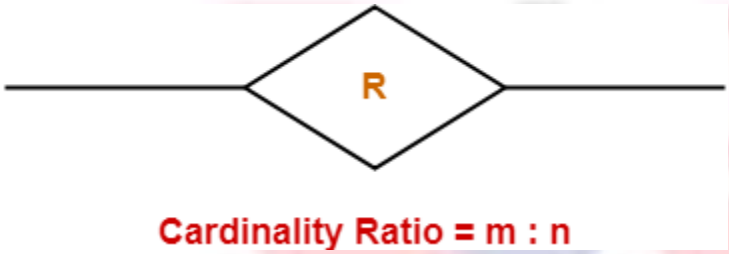
1. Many-to-Many Cardinality-

By this cardinality constraint,

An entity in set A can be associated with any number (zero or more) of entities in set B.

An entity in set B can be associated with any number (zero or more) of entities in set A.

Symbol Used-



2. Many-to-One Cardinality-

By this cardinality constraint,

An entity in set A can be associated with at most one entity in set B.

An entity in set B can be associated with any number (zero or more) of entities in set A.

Symbol Used-

OR

Cardinality Ratio = m : 1

Example-

Consider the following ER diagram-

Many to One Relationship

Here,

One student can enroll in at most one course.

One course can be enrolled by any number (zero or more) of students.

3. One-to-Many Cardinality-

By this cardinality constraint,

An entity in set A can be associated with any number (zero or more) of entities in set B.

An entity in set B can be associated with at most one entity in set A.

Symbol Used-

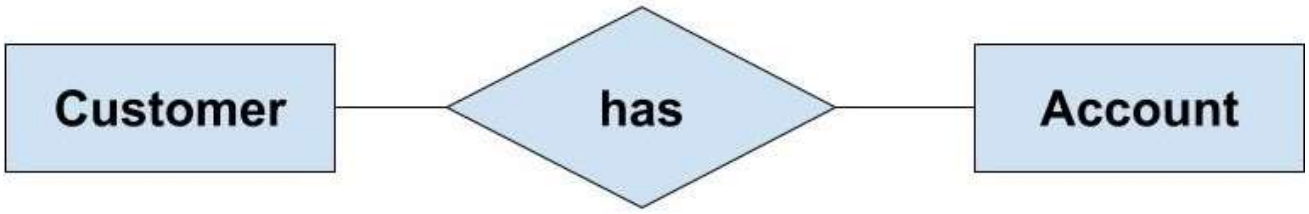
OR

Cardinality Ratio = 1 : n

Degree of Relationship

The degree of a relationship is the number of entity types that participate(associate) in a relationship. By seeing an E-R diagram, we can simply tell the degree of a relationship i.e **the number of an entity type that is connected to a relationship is the degree of that relationship**.

For example, If we have two entity type 'Customer' and 'Account' and they are linked using the primary key and foreign key. We can say that the degree of relationship is 2 because here two entities are taking part in the relationship.



degree 2

Based on the number of entity types that are connected we have the following degree of relationships:

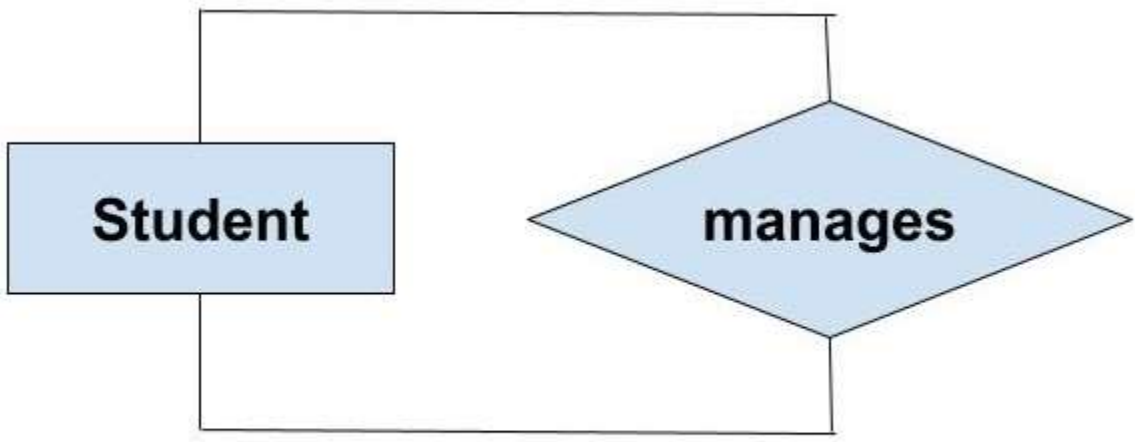
- Unary

- Binary
- Ternary
- N-ary

Unary (degree 1)

A unary relationship exists when both the participating entity type are the same. When such a relationship is present we say that the degree of relationship is 1.

For example, Suppose in a classroom, we have many students who belong to a particular club-like dance club, basketball club etc. and some of them are club leads. So, a particular group of student is managed by their respective club lead. Here, the group is formed from students and also, the club leads are chosen from students. So, the 'Student' is the only entity participating here. We can represent this relationship using the E-R diagram as follows:



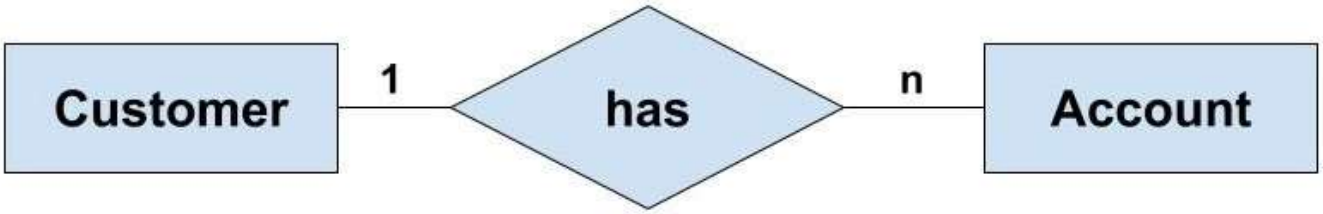
Unary Relationship
(degree 1)

So, here we get the answer to our first question which was asked in the introduction section. Yes, there can be only one entity type in a relationship and the minimum degree of a relationship can be one.

Binary (degree 2)

A binary relationship exists when exactly two entity type participates. When such a relationship is present we say that the degree is 2. This is the most common degree of relationship. It is easy to deal with such relationship as these can be easily converted into relational tables.

For example, We have two entity type 'Customer' and 'Account' where each 'Customer' has an 'Account' which stores the account details of the 'Customer'. Since we have two entity types participating we call it a binary relationship. Also, one 'Customer' can have many 'Account' but each 'Account' should belong to only one 'Customer'. We can say that it is a one-to-many binary relationship. (Learn more about types of relationships in DBMS from here)

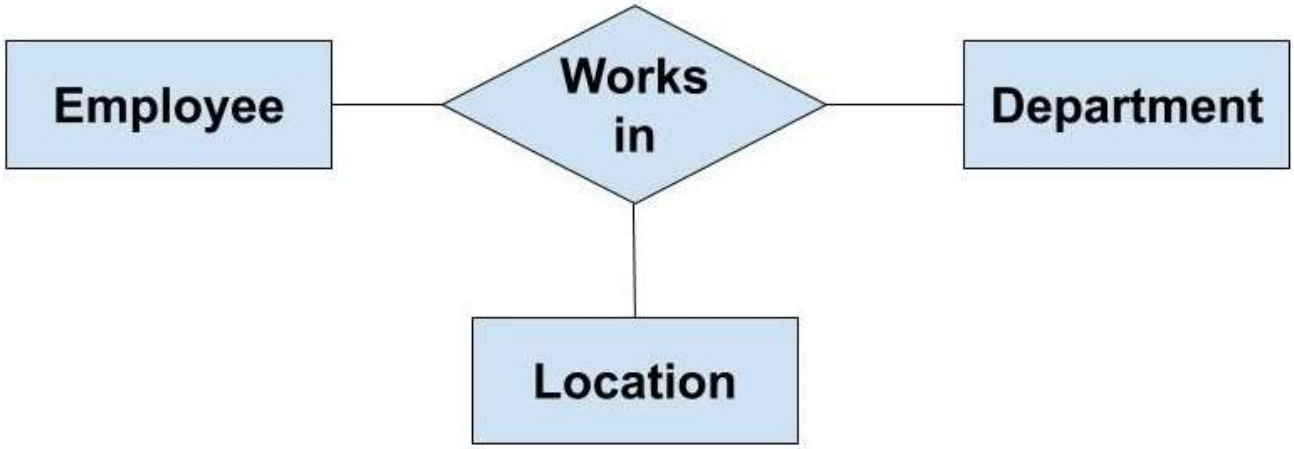


Binary Relationship (degree 2)

Ternary(degree 3)

A ternary relationship exists when exactly three entity type participates. When such a relationship is present we say that the degree is 3. As the number of entity increases in the relationship, it becomes complex to convert them into relational tables.

For example, We have three entity type 'Employee', 'Department' and 'Location'. The relationship between these entities are defined as an employee works in a department, an employee works at a particular location. So, we can see we have three entities participating in a relationship so it is a ternary relationship. The degree of this relation is 3.



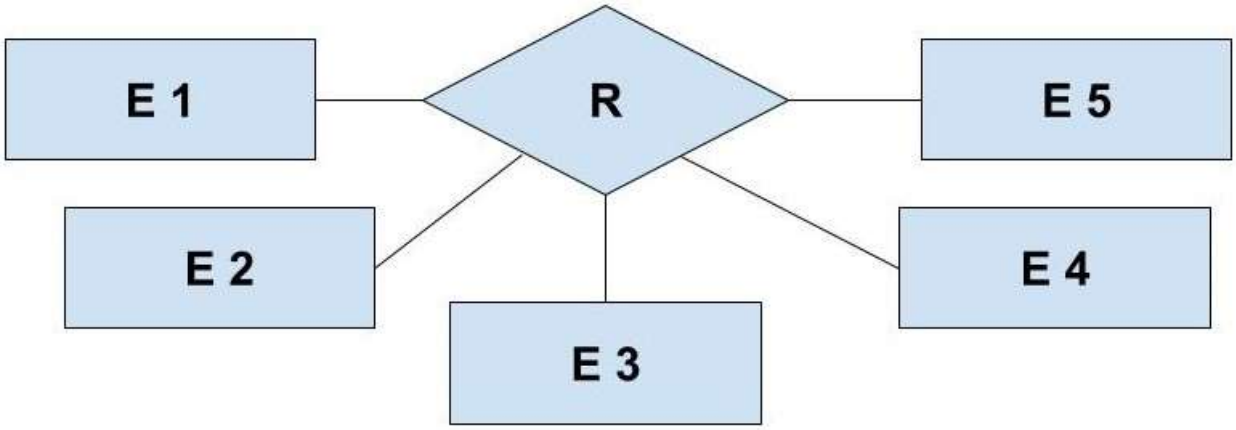
Ternary Relationship
(degree 3)

N-ary (n degree)

An N-ary relationship exists when ‘n’ number of entities are participating. So, any number of entities can participate in a relationship. There is no limitation to the maximum number of entities that can participate. But, relations with a higher degree are not common. This is because the conversion of higher degree relations to relational tables gets complex. We are making an E-R model because it can be easily be converted into any other model for implementing the database. But, this benefit is not available if we use higher degree relations. So, binary relations are more popular and widely used. Though we can make a relationship with any number of entity types but we don't do that.

We represent an N-ary relationship as follows:

Cartesian Product Operation in Relational Algebra



N-ary Relationship

Cartesian Product Operation in Relational Algebra

We already are aware of the fact that relations are nothing but a set of tuples, and here we will have 2 sets of tuples.

On applying CARTESIAN PRODUCT on two relations that is on two sets of tuples, it will take every tuple one by one from the left set(relation) and will pair it up with all the tuples in the right set(relation).

So, the CROSS PRODUCT of two relation A(R1, R2, R3, …, Rp) with degree p, and B(S1, S2, S3, …, Sn) with degree n, is a relation C(R1, R2, R3, …, Rp, S1, S2, S3, …, Sn) with degree p + n attributes.

CROSS PRODUCT is a binary set operation means, at a time we can apply the operation on two relations. But the two relations on which we are performing the operations do not have the same type of tuples, which means Union compatibility (or Type compatibility) of the two relations is not necessary.

Notation:

A × S

where A and S are the relations,
the symbol '×' is used to denote the CROSS PRODUCT operator.

Example:
Consider two relations STUDENT(SNO, FNAME, LNAME) and DETAIL(ROLLNO, AGE) below:

SNO	FNAME	LNAME
1	Albert	Singh
2	Nora	Fatehi

ROLLNO	AGE
5	18
9	21

On applying CROSS PRODUCT on STUDENT and DETAIL:

STUDENT × DETAILS

SNO	FNAME	LNAME	ROLLNO	AGE
1	Albert	Singh	5	18
1	Albert	Singh	9	21

2	Nora	Fatehi	5	18
2	Nora	Fatehi	9	21

We can observe that the number of tuples in STUDENT relation is 2, and the number of tuples in DETAIL is 2. So the number of tuples in the resulting relation on performing CROSS PRODUCT is 2*2 = 4.

Important points on CARTESIAN PRODUCT(CROSS PRODUCT) Operation:

1. The cardinality (number of tuples) of resulting relation from a Cross Product operation is equal to the number of attributes(say m) in the first relation multiplied by the number of attributes in the second relation(say n).

Cardinality = m*n

2. The Cross Product of two relation A(R1, R2, R3, …, Rp) with degree p, and B(S1, S2, S3, …, Sn) with degree n, is a relation C(R1, R2, R3, …, Rp, S1, S2, S3, …, Sn) with degree p + n attributes.

Degree = p+n

3. In SQL, CARTESIAN PRODUCT(CROSS PRODUCT) can be applied using CROSS JOIN.

4. In general, we don't use cartesian Product unnecessarily, which means without proper meaning we don't use Cartesian Product. Generally, we use Cartesian Product followed by a Selection operation and comparison on the operators as shown below :

$\sigma_{A=D}(A \times B)$

The above query gives meaningful results.

And this combination of Select and Cross Product operation is so popular that JOIN operation is inspired by this combination.

5. CROSS PRODUCT is a binary set operation means, at a time we can apply the operation on two relations.

What is Natural Join in SQL?

We have already learned that an EQUI JOIN performs a JOIN against equality or matching column(s) values of the associated tables and an equal sign (=) is used as comparison operator in the where clause to refer equality.

The SQL NATURAL JOIN is a type of EQUI JOIN and is structured in such a way that, columns with the same name of associated tables will appear once only.

Natural Join: Guidelines

- The associated tables have one or more pairs of identically named columns.
- The columns must be the same data type.
- Don't use ON clause in a natural join.

Syntax:

```
SELECT *
FROM table1
NATURAL JOIN table2;
```

Example:

Here is an example of SQL natural join between tow tables:

Sample table: foods

Sample table: company

To get all the unique columns from foods and company tables, the following SQL statement can be used:

SQL Code:

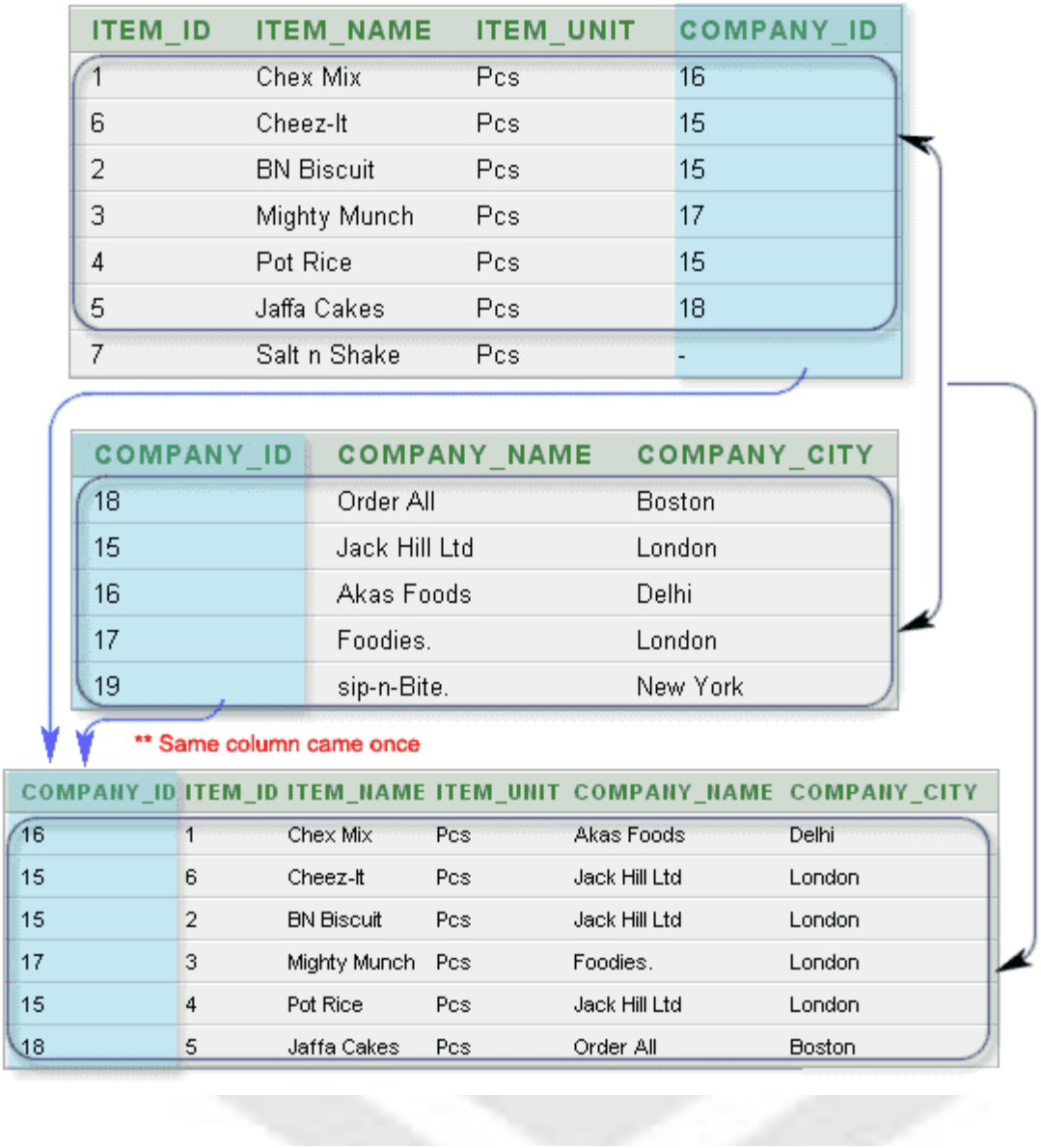
```
SELECT *

FROM foods

NATURAL JOIN company;
```

Copy

Output:

```
COMPANY_ID ITEM_ID    ITEM_NAME                 ITEM_UNIT
COMPANY_NAME             COMPANY_CITY
---------- ---------- ------------------------- ---------- -----
-------------------- --------------
16         1           Chex Mix                 Pcs        Akas
Foods                Delhi
15         6           Cheez-It                 Pcs        Jack
Hill Ltd             London
15         2           BN Biscuit               Pcs        Jack
Hill Ltd             London
17         3           Mighty Munch             Pcs
Foodies.                 London
15         4           Pot Rice                 Pcs        Jack
Hill Ltd             London
18         5           Jaffa Cakes              Pcs        Order
All                 Boston
```

Pictorial presentation of the above Natural Join:



ITEM_ID	ITEM_NAME	ITEM_UNIT	COMPANY_ID
1	Chex Mix	Pcs	16
6	Cheez-It	Pcs	15
2	BN Biscuit	Pcs	15
3	Mighty Munch	Pcs	17
4	Pot Rice	Pcs	15
5	Jaffa Cakes	Pcs	18
7	Salt n Shake	Pcs	-

COMPANY_ID	COMPANY_NAME	COMPANY_CITY
18	Order All	Boston
15	Jack Hill Ltd	London
16	Akas Foods	Delhi
17	Foodies.	London
19	sip-n-Bite.	New York

COMPANY_ID	ITEM_ID	ITEM_NAME	ITEM_UNIT	COMPANY_NAME	COMPANY_CITY
16	1	Chex Mix	Pcs	Akas Foods	Delhi
15	6	Cheez-It	Pcs	Jack Hill Ltd	London
15	2	BN Biscuit	Pcs	Jack Hill Ltd	London
17	3	Mighty Munch	Pcs	Foodies.	London
15	4	Pot Rice	Pcs	Jack Hill Ltd	London
18	5	Jaffa Cakes	Pcs	Order All	Boston

Difference between natural join and inner join

There is one significant difference between INNER JOIN and NATURAL JOIN is the number of columns returned. See the following example on company table and foods table :

SQL Code:

```
SELECT *

FROM company;
```

Copy

Output:

```
COMPANY_ID COMPANY_NAME               COMPANY_CITY
---------- -------------------------- ---------------
18         Order All                  Boston
15         Jack Hill Ltd              London
16         Akas Foods                 Delhi
17         Foodies.                   London
19         sip-n-Bite.                New York
```

SQL Code:

```
SELECT *

FROM foods;
```

Copy

Output:

```
ITEM_ID    ITEM_NAME                  ITEM_UNIT  COMPANY_ID
---------- -------------------------- ---------- ----------
1          Chex Mix                   Pcs        16
6          Cheez-It                   Pcs        15
2          BN Biscuit                 Pcs        15
3          Mighty Munch               Pcs        17
4          Pot Rice                   Pcs        15
5          Jaffa Cakes                Pcs        18
7          Salt n Shake               Pcs
```

The INNER JOIN of company and foods on company_id will return :

SQL Code:

```
SELECT *

FROM company

INNER JOIN foods
```

```
ON company.company_id = foods.company_id;
```

Copy

Output:

```
COMPANY_ID COMPANY_NAME    COMPANY_CITY    ITEM_ID    ITEM_NAME
ITEM_UNIT  COMPANY_ID
---------- --------------- --------------- ---------- ----------
----- ---------- ----------
16         Akas Foods      Delhi           1          Chex Mix
Pcs        16
15         Jack Hill Ltd   London          6          Cheez-It
Pcs        15
15         Jack Hill Ltd   London          2          BN Biscuit
Pcs        15
17         Foodies.        London          3          Mighty
Munch    Pcs        17
15         Jack Hill Ltd   London          4          Pot Rice
Pcs        15
18         Order All       Boston          5          Jaffa
Cakes     Pcs        18
```

SQL Code:

```
SELECT *

FROM company

NATURAL JOIN foods;
```

Copy

Output:

```
COMPANY_ID COMPANY_NAME    COMPANY_CITY    ITEM_ID    ITEM_NAME
ITEM_UNIT
---------- --------------- --------------- ---------- ----------
----- ----------
16         Akas Foods      Delhi           1          Chex Mix
Pcs
15         Jack Hill Ltd   London          6          Cheez-It
Pcs
15         Jack Hill Ltd   London          2          BN Biscuit
Pcs
```

```
17          Foodies.        London          3          Mighty
Munch     Pcs
15          Jack Hill Ltd   London          4          Pot Rice
Pcs
18          Order All       Boston          5          Jaffa
Cakes      Pcs
```

What is Equi Join in SQL?

SQL EQUI JOIN performs a JOIN against equality or matching column(s) values of the associated tables. An equal sign (=) is used as comparison operator in the where clause to refer equality.

You may also perform EQUI JOIN by using JOIN keyword followed by ON keyword and then specifying names of the columns along with their associated tables to check equality.

Syntax:

SELECT column_list

FROM table1, table2....

WHERE table1.column_name =

table2.column_name

or

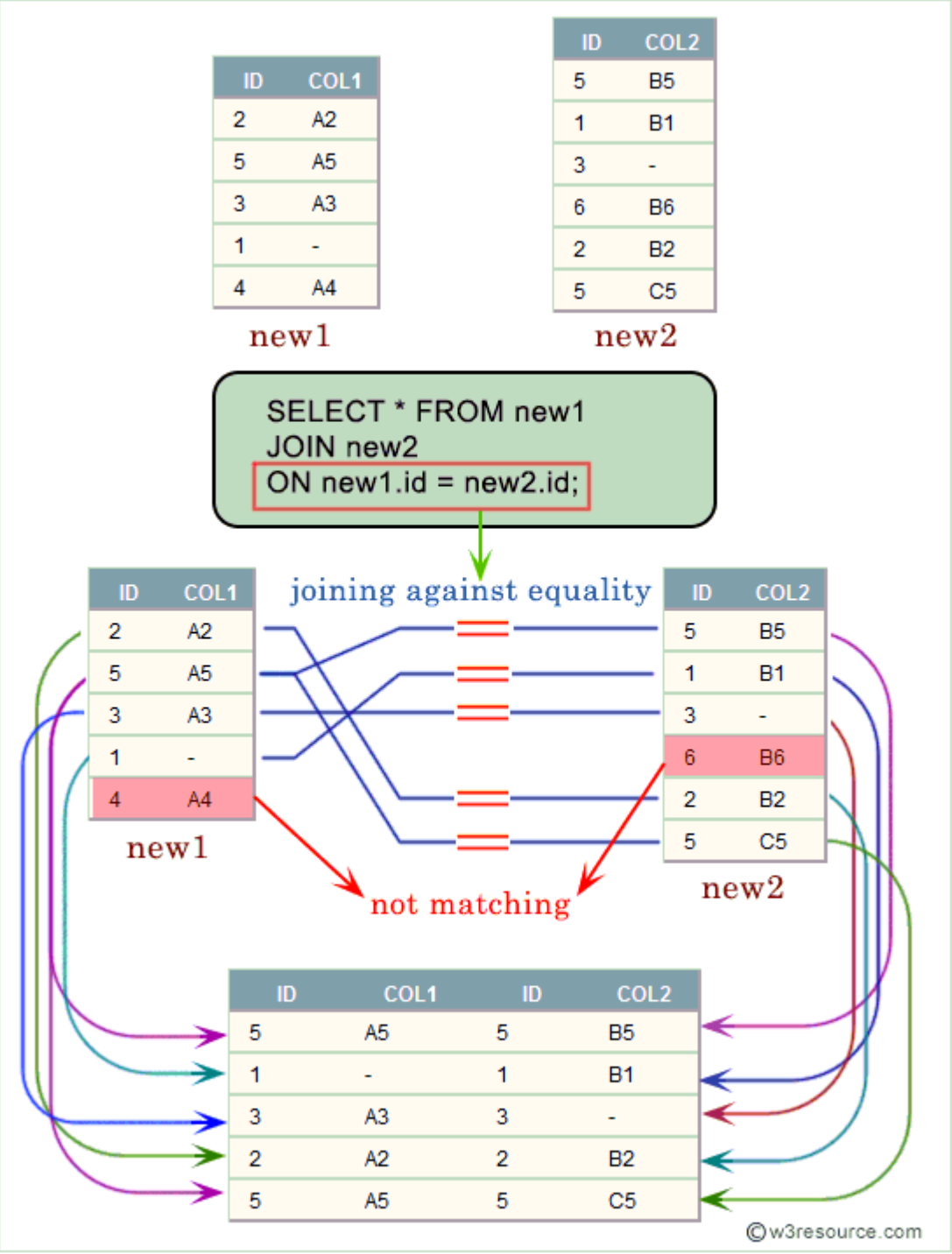
SELECT *

FROM table1

JOIN table2

[ON (join_condition)]

Pictorial representation:



Example:

Here is an example of Equi Join in SQL.

Sample table: agents

Sample table: customer

To get agent name column from agents table and cust name and cust city columns from customer table after joining said two tables with the following condition -

1. working area of agents and customer city of customer table must be same,

the following SQL statement can be used:

SQL Code:

```
SELECT agents.agent_name,customer.cust_name,
customer.cust_city
FROM agents,customer
WHERE agents.working_area=customer.cust_city;
```

Copy

Output:

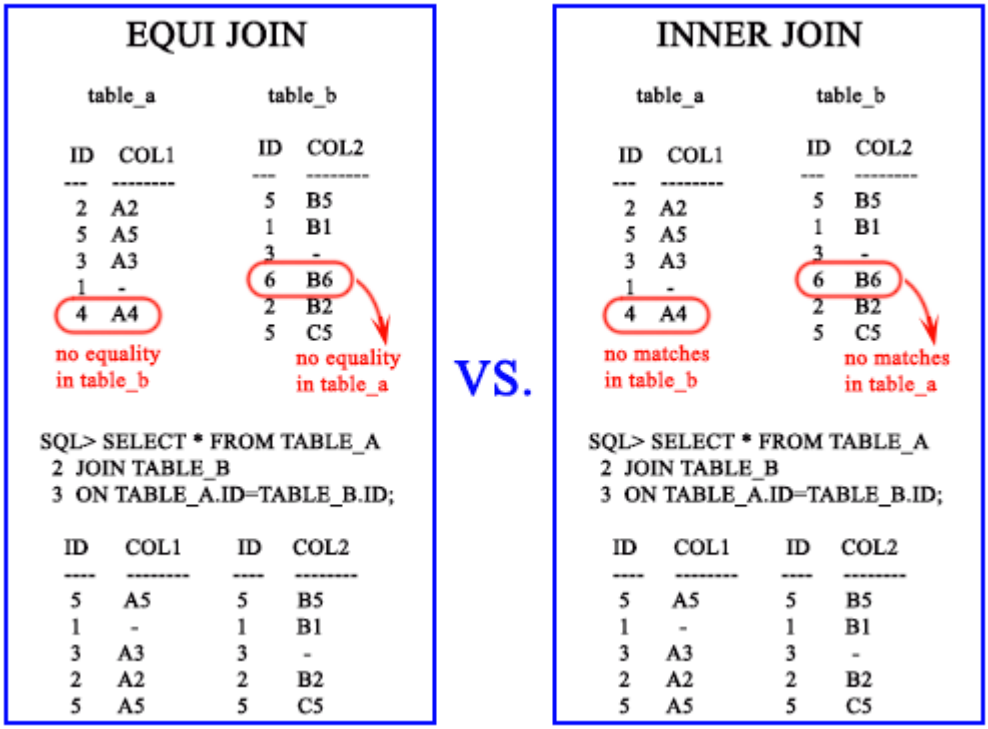
```
AGENT_NAME                                         CUST_NAME
CUST_CITY
---------------------------------------- ----------------------
---------------- ------------
Ravi Kumar                                         Ravindran
Bangalore
Ramasundar                                         Ravindran
Bangalore
```

```
Subbarao                                Ravindran
Bangalore
Ravi Kumar                              Srinivas
Bangalore
Ramasundar                              Srinivas
Bangalore
Subbarao                                Srinivas
Bangalore
Ravi Kumar                              Rangarappa
Bangalore
Ramasundar                              Rangarappa
Bangalore
Subbarao                                Rangarappa
Bangalore
Ravi Kumar                              Venkatpati
Bangalore
Ramasundar                              Venkatpati
Bangalore
Subbarao                                Venkatpati
Bangalore
Anderson                                Fleming
Brisban
Anderson                                Jacks
Brisban
Anderson                                Winston
Brisban
Santakumar                              Yearannaidu
Chennai
...........
...........
```

What is the difference between Equi Join and Inner Join in SQL?

An equijoin is a join with a join condition containing an equality operator. An equijoin returns only the rows that have equivalent values for the specified columns.

An inner join is a join of two or more tables that returns only those rows (compared using a comparison operator) that satisfy the join condition.



Unit 4 : SQL (Structured Query Language)

Introduction to SQL

SQL is a standard language for accessing and manipulating databases.

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

SQL is a Standard - BUT....

Although SQL is an ANSI/ISO standard, there are different versions of the SQL language.

However, to be compliant with the ANSI standard, they all support at least the major commands (such as SELECT, UPDATE, DELETE, INSERT, WHERE) in a similar manner.

SQL is a language to operate databases; it includes database creation, deletion, fetching rows, modifying rows, etc. SQL is an **ANSI** (American National Standards Institute) standard language, but there are many different versions of the SQL language.

What is SQL?

SQL is Structured Query Language, which is a computer language for storing, manipulating and retrieving data stored in a relational database.

SQL is the standard language for Relational Database System. All the Relational Database Management Systems (RDMS) like MySQL, MS Access, Oracle, Sybase, Informix, Postgres and SQL Server use SQL as their standard database language.

Also, they are using different dialects, such as −

- MS SQL Server using T-SQL,
- Oracle using PL/SQL,
- MS Access version of SQL is called JET SQL (native format) etc.

Why SQL?

SQL is widely popular because it offers the following advantages −

- Allows users to access data in the relational database management systems.
- Allows users to describe the data.
- Allows users to define the data in a database and manipulate that data.
- Allows to embed within other languages using SQL modules, libraries & pre-compilers.
- Allows users to create and drop databases and tables.
- Allows users to create view, stored procedure, functions in a database.
- Allows users to 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.

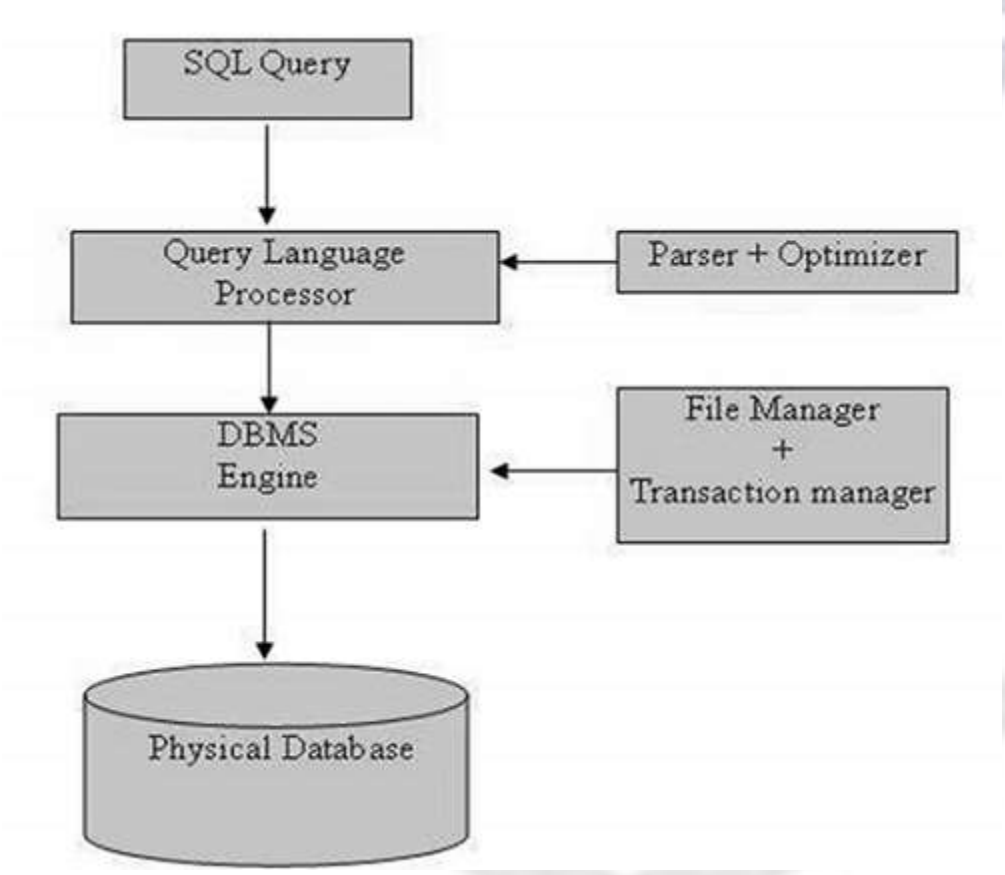
These components are −

- 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

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.

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.

DML - Data Manipulation Language

DCL - Data Control Language

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

SQL is followed by a unique set of rules and guidelines called Syntax. This tutorial gives you a quick start with SQL by listing all the basic SQL Syntax.

All the SQL statements start with any of the keywords like SELECT, INSERT, UPDATE, DELETE, ALTER, DROP, CREATE, USE, SHOW and all the statements end with a semicolon (;).

The most important point to be noted here is that SQL is case insensitive, which means SELECT and select have same meaning in SQL statements. Whereas, MySQL makes difference in table names. So, if you are working with MySQL, then you need to give table names as they exist in the database.

Various Syntax in SQL

All the examples given in this tutorial have been tested with a MySQL server.

SQL SELECT Statement

```
SELECT column1, column2....columnN
FROM   table_name;
```

SQL DISTINCT Clause

```
SELECT DISTINCT column1, column2....columnN
FROM   table_name;
```

SQL WHERE Clause

```
SELECT column1, column2....columnN
FROM   table_name
WHERE  CONDITION;
```

SQL AND/OR Clause

```
SELECT column1, column2....columnN
FROM   table_name
WHERE  CONDITION-1 {AND|OR} CONDITION-2;
```

SQL IN Clause

```
SELECT column1, column2....columnN
FROM   table_name
WHERE  column_name IN (val-1, val-2,...val-N);
```

SQL BETWEEN Clause

```
SELECT column1, column2....columnN
FROM   table_name
WHERE  column_name BETWEEN val-1 AND val-2;
```

SQL LIKE Clause

```
SELECT column1, column2....columnN
FROM   table_name
WHERE  column_name LIKE { PATTERN };
```

SQL ORDER BY Clause

```
SELECT column1, column2....columnN
FROM   table_name
WHERE  CONDITION
ORDER BY column_name {ASC|DESC};
```

SQL GROUP BY Clause

```
SELECT SUM(column_name)
FROM   table_name
WHERE  CONDITION
GROUP BY column_name;
```

SQL COUNT Clause

```
SELECT COUNT(column_name)
FROM   table_name
WHERE  CONDITION;
```

SQL HAVING Clause

```
SELECT SUM(column_name)
FROM   table_name
WHERE  CONDITION
GROUP BY column_name
HAVING (arithematic function condition);
```

SQL CREATE TABLE Statement

```
CREATE TABLE table_name(
column1 datatype,
```

```
column2 datatype,
column3 datatype,
.....
columnN datatype,
PRIMARY KEY( one or more columns )
);
```

SQL DROP TABLE Statement

```
DROP TABLE table_name;
```

SQL CREATE INDEX Statement

```
CREATE UNIQUE INDEX index_name
ON table_name ( column1, column2,...columnN);
```

SQL DROP INDEX Statement

```
ALTER TABLE table_name
DROP INDEX index_name;
```

SQL DESC Statement

```
DESC table_name;
```

SQL TRUNCATE TABLE Statement

```
TRUNCATE TABLE table_name;
```

SQL ALTER TABLE Statement

```
ALTER TABLE table_name {ADD|DROP|MODIFY} column_name {data_ype};
```

SQL ALTER TABLE Statement (Rename)

```
ALTER TABLE table_name RENAME TO new_table_name;
```

SQL INSERT INTO Statement

```
INSERT INTO table_name( column1, column2....columnN)
VALUES ( value1, value2....valueN);
```

SQL UPDATE Statement

```
UPDATE table_name
SET column1 = value1, column2 = value2....columnN=valueN
[ WHERE  CONDITION ];
```

SQL DELETE Statement

```
DELETE FROM table_name
WHERE  {CONDITION};
```

SQL CREATE DATABASE Statement

```
CREATE DATABASE database_name;
```

SQL DROP DATABASE Statement

```
DROP DATABASE database_name;
```

SQL USE Statement

```
USE database_name;
```

SQL COMMIT Statement

```
COMMIT;
```

SQL ROLLBACK Statement

SQL Data Type is an attribute that specifies the type of data of any object. Each column, variable and expression has a related data type in SQL. You can use these data types while creating your tables. You can choose a data type for a table column based on your requirement.

SQL Server offers six categories of data types for your use which are listed below –

Exact Numeric Data Types

DATA TYPE	FROM	TO

bigint	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
int	-2,147,483,648	2,147,483,647
smallint	-32,768	32,767
tinyint	0	255
bit	0	1
decimal	-10^38 +1	10^38 -1
numeric	-10^38 +1	10^38 -1
money	-922,337,203,685,477.5808	+922,337,203,685,477.5807
smallmoney	-214,748.3648	+214,748.3647

Approximate Numeric Data Types

DATA TYPE	FROM	TO
float	-1.79E + 308	1.79E + 308
real	-3.40E + 38	3.40E + 38

Date and Time Data Types

DATA TYPE	FROM	TO

datetime	Jan 1, 1753	Dec 31, 9999
smalldatetime	Jan 1, 1900	Jun 6, 2079
date	Stores a date like June 30, 1991	
time	Stores a time of day like 12:30 P.M.	

Note − Here, datetime has 3.33 milliseconds accuracy where as smalldatetime has 1 minute accuracy.

Character Strings Data Types

Sr.No.	DATA TYPE & Description
1	**char** Maximum length of 8,000 characters.(Fixed length non-Unicode characters)
2	**varchar** Maximum of 8,000 characters.(Variable-length non-Unicode data).
3	**varchar(max)** Maximum length of 2E + 31 characters, Variable-length non-Unicode data (SQL Server 2005 only).
4	**text** Variable-length non-Unicode data with a maximum length of 2,147,483,647 characters.

Unicode Character Strings Data Types

Sr.No.	DATA TYPE & Description
1	**nchar** Maximum length of 4,000 characters.(Fixed length Unicode)
2	**nvarchar** Maximum length of 4,000 characters.(Variable length Unicode)
3	**nvarchar(max)** Maximum length of 2E + 31 characters (SQL Server 2005 only).(Variable length Unicode)
4	**ntext** Maximum length of 1,073,741,823 characters. (Variable length Unicode)

Binary Data Types

Sr.No.	DATA TYPE & Description
1	**binary** Maximum length of 8,000 bytes(Fixed-length binary data)
2	**varbinary** Maximum length of 8,000 bytes.(Variable length binary data)

3	**varbinary(max)** Maximum length of 2E + 31 bytes (SQL Server 2005 only). (Variable length Binary data)
4	**image** Maximum length of 2,147,483,647 bytes. (Variable length Binary Data)

Misc Data Types

Sr.No.	DATA TYPE & Description
1	**sql_variant** Stores values of various SQL Server-supported data types, except text, ntext, and timestamp.
2	**timestamp** Stores a database-wide unique number that gets updated every time a row gets updated
3	**uniqueidentifier** Stores a globally unique identifier (GUID)
4	**xml** Stores XML data. You can store xml instances in a column or a variable (SQL Server 2005 only).

5	**cursor** Reference to a cursor object
6	**table** Stores a result set for later processing

What is an Operator in SQL?

An operator is a reserved word or a character used primarily in an SQL statement's WHERE clause to perform operation(s), such as comparisons and arithmetic operations. These Operators are used to specify conditions in an SQL statement and to serve as conjunctions for multiple conditions in a statement.

- Arithmetic operators
- Comparison operators
- Logical operators
- Operators used to negate conditions

SQL Arithmetic Operators

Assume '**variable a**' holds 10 and '**variable b**' holds 20, then −

Show Examples

Operator	**Description**	**Example**
+ (Addition)	Adds values on either side of the operator.	a + b will give 30

- (Subtraction)	Subtracts right hand operand from left hand operand.	a - b will give -10
* (Multiplication)	Multiplies values on either side of the operator.	a * b will give 200
/ (Division)	Divides left hand operand by right hand operand.	b / a will give 2
% (Modulus)	Divides left hand operand by right hand operand and returns remainder.	b % a will give 0

SQL Comparison Operators

Assume **'variable a'** holds 10 and **'variable b'** holds 20, then −

Show Examples

Operator	Description	Example
=	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(a = b) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(a != b) is true.
<>	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(a <> b) is true.

>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(a > b) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(a < b) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(a >= b) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(a <= b) is true.
!<	Checks if the value of left operand is not less than the value of right operand, if yes then condition becomes true.	(a !< b) is false.
!>	Checks if the value of left operand is not greater than the value of right operand, if yes then condition becomes true.	(a !> b) is true.

SQL Logical Operators

Here is a list of all the logical operators available in SQL.

Show Examples

Sr.No.	Operator & Description
1	**ALL** The ALL operator is used to compare a value to all values in another value set.

2	**AND** The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.
3	**ANY** The ANY operator is used to compare a value to any applicable value in the list as per the condition.
4	**BETWEEN** The BETWEEN operator is used to search for values that are within a set of values, given the minimum value and the maximum value.
5	**EXISTS** The EXISTS operator is used to search for the presence of a row in a specified table that meets a certain criterion.
6	**IN** The IN operator is used to compare a value to a list of literal values that have been specified.
7	**LIKE** The LIKE operator is used to compare a value to similar values using wildcard operators.
8	**NOT** The NOT operator reverses the meaning of the logical operator with which it is

	used. Eg: NOT EXISTS, NOT BETWEEN, NOT IN, etc. **This is a negate operator.**
9	**OR** The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.
10	**IS NULL** The NULL operator is used to compare a value with a NULL value.
11	**UNIQUE** The UNIQUE operator searches every row of a specified table for uniqueness (no duplicates).

An expression is a combination of one or more values, operators and SQL functions that evaluate to a value. These SQL EXPRESSIONs are like formulae and they are written in query language. You can also use them to query the database for a specific set of data.

Syntax

Consider the basic syntax of the SELECT statement as follows −

SELECT column1, column2, columnN

FROM table_name

WHERE [CONDITION|EXPRESSION];

There are different types of SQL expressions, which are mentioned below −

- Boolean
- Numeric
- Date

Let us now discuss each of these in detail.

Boolean Expressions

SQL Boolean Expressions fetch the data based on matching a single value. Following is the syntax −

SELECT column1, column2, columnN

FROM table_name

WHERE SINGLE VALUE MATCHING EXPRESSION;

Consider the CUSTOMERS table having the following records −

SQL> SELECT * FROM CUSTOMERS;

+----+----------+-----+-----------+----------+

| 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 |

+----+----------+-----+-----------+----------+

7 rows in set (0.00 sec)

The following table is a simple example showing the usage of various SQL Boolean Expressions −

SQL> SELECT * FROM CUSTOMERS WHERE SALARY = 10000;

+----+-------+-----+---------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+-------+-----+---------+----------+

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+-------+-----+---------+----------+

1 row in set (0.00 sec)

Numeric Expression

These expressions are used to perform any mathematical operation in any query. Following is the syntax −

SELECT numerical_expression as OPERATION_NAME

[FROM table_name

WHERE CONDITION] ;

Here, the numerical_expression is used for a mathematical expression or any formula. Following is a simple example showing the usage of SQL Numeric Expressions −

SQL> SELECT (15 + 6) AS ADDITION

+----------+

| ADDITION |

```
+----------+
|       21 |
+----------+
```

1 row in set (0.00 sec)

There are several built-in functions like avg(), sum(), count(), etc., to perform what is known as the aggregate data calculations against a table or a specific table column.

```
SQL> SELECT COUNT(*) AS "RECORDS" FROM CUSTOMERS;
+---------+
| RECORDS |
+---------+
|       7 |
+---------+
```

1 row in set (0.00 sec)

Date Expressions

Date Expressions return current system date and time values −

```
SQL>  SELECT CURRENT_TIMESTAMP;
+---------------------+
| Current_Timestamp   |
+---------------------+
| 2009-11-12 06:40:23 |
+---------------------+
```

1 row in set (0.00 sec)

Another date expression is as shown below −

```
SQL>  SELECT  GETDATE();;
+-------------------------+
| GETDATE                 |
+-------------------------+
| 2009-10-22 12:07:18.140 |
+-------------------------+
```

1 row in set (0.00 sec)

The SQL **CREATE DATABASE** statement is used to create a new SQL database.

Syntax

The basic syntax of this CREATE DATABASE statement is as follows −

```
CREATE DATABASE DatabaseName;
```

Always the database name should be unique within the RDBMS.

Example

If you want to create a new database <testDB>, then the CREATE DATABASE statement would be as shown below −

```
SQL> CREATE DATABASE testDB;
```

Make sure you have the admin privilege before creating any database. Once a database is created, you can check it in the list of databases as follows −

```
SQL> SHOW DATABASES;
+--------------------+
```

| Database |

+--------------------+

| information_schema |

| AMROOD |

| TUTORIALSPOINT |

| mysql |

| orig |

| test |

| testDB |

+--------------------+

7 rows in set (0.00 sec)

The SQL **DROP DATABASE** statement is used to drop an existing database in SQL schema.

Syntax

The basic syntax of DROP DATABASE statement is as follows −

DROP DATABASE DatabaseName;

Always the database name should be unique within the RDBMS.

Example

If you want to delete an existing database <testDB>, then the DROP DATABASE statement would be as shown below −

SQL> DROP DATABASE testDB;

NOTE − Be careful before using this operation because by deleting an existing database would result in loss of complete information stored in the database.

Make sure you have the admin privilege before dropping any database. Once a database is dropped, you can check it in the list of the databases as shown below −

```
SQL> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| AMROOD             |
| TUTORIALSPOINT     |
| mysql              |
| orig               |
| test               |
+--------------------+
```

When you have multiple databases in your SQL Schema, then before starting your operation, you would need to select a database where all the operations would be performed.

The SQL **USE** statement is used to select any existing database in the SQL schema.

Syntax

The basic syntax of the USE statement is as shown below −

```
USE DatabaseName;
```

Always the database name should be unique within the RDBMS.

Example

You can check the available databases as shown below −

```
SQL> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| AMROOD             |
| TUTORIALSPOINT     |
| mysql              |
| orig               |
| test               |
+--------------------+
6 rows in set (0.00 sec)
```

Now, if you want to work with the AMROOD database, then you can execute the following SQL command and start working with the AMROOD

A Subquery or Inner query or a Nested query is a query within another SQL query and embedded within the WHERE clause.

A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.

Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.

There are a few rules that subqueries must follow −

- Subqueries must be enclosed within parentheses.
- A subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the subquery to compare its selected columns.
- An ORDER BY command cannot be used in a subquery, although the main query can use an ORDER BY. The GROUP BY command can be used to perform the same function as the ORDER BY in a subquery.
- Subqueries that return more than one row can only be used with multiple value operators such as the IN operator.
- The SELECT list cannot include any references to values that evaluate to a BLOB, ARRAY, CLOB, or NCLOB.
- A subquery cannot be immediately enclosed in a set function.
- The BETWEEN operator cannot be used with a subquery. However, the BETWEEN operator can be used within the subquery.

Subqueries with the SELECT Statement

Subqueries are most frequently used with the SELECT statement. The basic syntax is as follows −

```
SELECT column_name [, column_name ]

FROM   table1 [, table2 ]

WHERE  column_name OPERATOR

  (SELECT column_name [, column_name ]

  FROM table1 [, table2 ]

  [WHERE])
```

Example

Consider the CUSTOMERS table having the following records −

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  35 | 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 |
+----+----------+-----+-----------+----------+
```

Now, let us check the following subquery with a SELECT statement.

```
SQL> SELECT *
  FROM CUSTOMERS
  WHERE ID IN (SELECT ID
      FROM CUSTOMERS
      WHERE SALARY > 4500) ;
```

This would produce the following result.

```
+----+----------+-----+---------+----------+
```

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+---------+----------+

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+---------+----------+

Subqueries with the INSERT Statement

Subqueries also can be used with INSERT statements. The INSERT statement uses the data returned from the subquery to insert into another table. The selected data in the subquery can be modified with any of the character, date or number functions.

The basic syntax is as follows.

INSERT INTO table_name [(column1 [, column2])]

SELECT [*|column1 [, column2]

FROM table1 [, table2]

[WHERE VALUE OPERATOR]

Example

Consider a table CUSTOMERS_BKP with similar structure as CUSTOMERS table. Now to copy the complete CUSTOMERS table into the CUSTOMERS_BKP table, you can use the following syntax.

SQL> INSERT INTO CUSTOMERS_BKP

SELECT * FROM CUSTOMERS

WHERE ID IN (SELECT ID

FROM CUSTOMERS) ;

Subqueries with the UPDATE Statement

The subquery can be used in conjunction with the UPDATE statement. Either single or multiple columns in a table can be updated when using a subquery with the UPDATE statement.

The basic syntax is as follows.

UPDATE table

SET column_name = new_value

[WHERE OPERATOR [VALUE]

(SELECT COLUMN_NAME

FROM TABLE_NAME)

[WHERE)]

Example

Assuming, we have CUSTOMERS_BKP table available which is backup of CUSTOMERS table. The following example updates SALARY by 0.25 times in the CUSTOMERS table for all the customers whose AGE is greater than or equal to 27.

SQL> UPDATE CUSTOMERS

SET SALARY = SALARY * 0.25

WHERE AGE IN (SELECT AGE FROM CUSTOMERS_BKP

WHERE AGE >= 27);

This would impact two rows and finally CUSTOMERS table would have the following records.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  35 | Ahmedabad |   125.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  2125.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Subqueries with the DELETE Statement

The subquery can be used in conjunction with the DELETE statement like with any other statements mentioned above.

The basic syntax is as follows.

```
DELETE FROM TABLE_NAME
[ WHERE OPERATOR [ VALUE ]
   (SELECT COLUMN_NAME
   FROM TABLE_NAME)
   [ WHERE) ]
```

Example

Assuming, we have a CUSTOMERS_BKP table available which is a backup of the CUSTOMERS table. The following example deletes the records from the CUSTOMERS table for all the customers whose AGE is greater than or equal to 27.

```
SQL> DELETE FROM CUSTOMERS
  WHERE AGE IN (SELECT AGE FROM CUSTOMERS_BKP
    WHERE AGE >= 27 );
```

This would impact two rows and finally the CUSTOMERS table would have the following records.

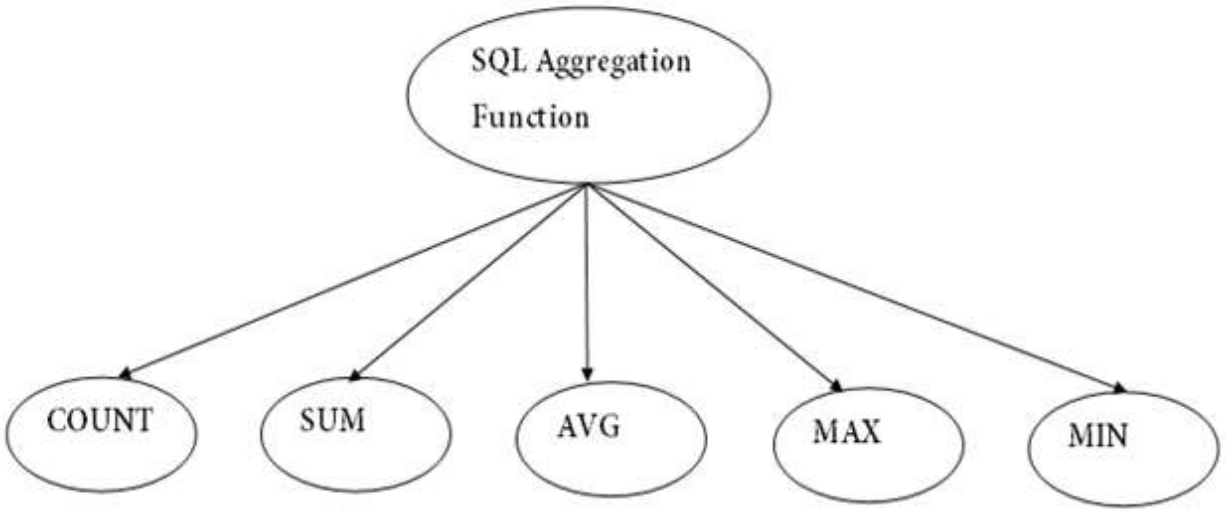
```
+----+----------+-----+---------+----------+
| ID | NAME     | AGE | ADDRESS | SALARY   |
+----+----------+-----+---------+----------+
|  2 | Khilan   |  25 | Delhi   |  1500.00 |
|  3 | kaushik  |  23 | Kota    |  2000.00 |
|  4 | Chaitali |  25 | Mumbai  |  6500.00 |
|  6 | Komal    |  22 | MP      |  4500.00 |
|  7 | Muffy    |  24 | Indore  | 10000.00 |
```

SQL Aggregate Functions

- SQL aggregation function is used to perform the calculations on multiple rows of a single column of a table. It returns a single value.
- It is also used to summarize the data.

Types of SQL Aggregation Function

+----+----------+-----+---------+----------+



1. COUNT FUNCTION

COUNT function is used to Count the number of rows in a database table. It can work on both numeric and non-numeric data types.

COUNT function uses the COUNT(*) that returns the count of all the rows in a specified table. COUNT(*) considers duplicate and Null.

Syntax

COUNT(*)

or

COUNT([ALL|DISTINCT] expression)

Sample table:

PRODUCT_MAST

PRODUCT	COMPANY	QTY	RATE	COST
Item1	Com1	2	10	20
Item2	Com2	3	25	75
Item3	Com1	2	30	60
Item4	Com3	5	10	50
Item5	Com2	2	20	40
Item6	Cpm1	3	25	75
Item7	Com1	5	30	150
Item8	Com1	3	10	30
Item9	Com2	2	25	50
Item10	Com3	4	30	120

Example: COUNT()

```
1. SELECT COUNT(*)
2. FROM PRODUCT_MAST;
```

Output:

```
10
```

Example: COUNT with WHERE

```
1. SELECT COUNT(*)
2. FROM PRODUCT_MAST;
3. WHERE RATE>=20;
```

Output:

```
7
```

Example: COUNT() with DISTINCT

```
1. SELECT COUNT(DISTINCT COMPANY)
2. FROM PRODUCT_MAST;
```

Output:

```
3
```

Example: COUNT() with GROUP BY

```
1. SELECT COMPANY, COUNT(*)
2. FROM PRODUCT_MAST
3. GROUP BY COMPANY;
```

Output:

```
Com1     5
Com2     3
Com3     2
```

Example: COUNT() with HAVING

```
1. SELECT COMPANY, COUNT(*)
2. FROM PRODUCT_MAST
3. GROUP BY COMPANY
4. HAVING COUNT(*)>2;
```

Output:

```
Com1     5
Com2     3
```

2. SUM Function

Sum function is used to calculate the sum of all selected columns. It works on numeric fields only.

Syntax

```
1. SUM()
2. or
3. SUM( [ALL|DISTINCT] expression )
```

Example: SUM()

```
1. SELECT SUM(COST)
2. FROM PRODUCT_MAST;
```

Output:

```
670
```

Example: SUM() with WHERE

```
1. SELECT SUM(COST)
2. FROM PRODUCT_MAST
3. WHERE QTY>3;
```

Output:

```
320
```

Example: SUM() with GROUP BY

```
1. SELECT SUM(COST)
2. FROM PRODUCT_MAST
3. WHERE QTY>3
4. GROUP BY COMPANY;
```

Output:

```
Com1    150
Com2    170
```

Example: SUM() with HAVING

```
1. SELECT COMPANY, SUM(COST)
2. FROM PRODUCT_MAST
3. GROUP BY COMPANY
4. HAVING SUM(COST)>=170;
```

Output:

```
Com1    335
Com3    170
```

3. AVG function

The AVG function is used to calculate the average value of the numeric type. AVG function returns the average of all non-Null values.

Syntax

1. AVG()
2. or
3. AVG([ALL|DISTINCT] expression)

Example:

```
1. SELECT AVG(COST)
2. FROM PRODUCT_MAST;
```

Output:

```
67.00
```

4. MAX Function

MAX function is used to find the maximum value of a certain column. This function determines the largest value of all selected values of a column.

Syntax

1. MAX()
2. or
3. MAX([ALL|DISTINCT] expression)

Example:

```
1. SELECT MAX(RATE)
2. FROM PRODUCT_MAST;
```

```
30
```

5. MIN Function

MIN function is used to find the minimum value of a certain column. This function determines the smallest value of all selected values of a column.

Syntax

1. MIN()
2. or
3. MIN([ALL|DISTINCT] expression)

Example:

```
1. SELECT MIN(RATE)
2. FROM PRODUCT_MAST;
```

Output:

```
10
```

next →← prev

SQL JOIN

As the name shows, JOIN means to combine something. In case of SQL, JOIN means "to combine two or more tables".

In SQL, JOIN clause is used to combine the records from two or more tables in a database.

Types of SQL JOIN

INNER JOIN

LEFT JOIN

RIGHT JOIN

FULL JOIN

Sample Table

EMP_ID	EMP_NAME	CITY	SALARY	AGE
1	Angelina	Chicago	200000	30
2	Robert	Austin	300000	26
3	Christian	Denver	100000	42
4	Kristen	Washington	500000	29
5	Russell	Los angels	200000	36
6	Marry	Canada	600000	48

PROJECT

PROJECT_NO	EMP_ID	DEPARTMENT
101	1	Testing
102	2	Development
103	3	Designing
104	4	Development

1. INNER JOIN

In SQL, INNER JOIN selects records that have matching values in both tables as long as the condition is satisfied. It returns the combination of all rows from both the tables where the condition satisfies.

Syntax

```
SELECT table1.column1, table1.column2, table2.column1,....
FROM table1
INNER JOIN table2
ON table1.matching_column = table2.matching_column;
```

Query

```
SELECT EMPLOYEE.EMP_NAME, PROJECT.DEPARTMENT
FROM EMPLOYEE
INNER JOIN PROJECT
ON PROJECT.EMP_ID = EMPLOYEE.EMP_ID;
```

Output

EMP_NAME	DEPARTMENT
Angelina	Testing
Robert	Development
Christian	Designing
Kristen	Development

LEFT JOIN

The SQL left join returns all the values from left table and the matching values from the right table. If there is no matching join value, it will return NULL.

Syntax

```
SELECT table1.column1, table1.column2, table2.column1,....
FROM table1
LEFT JOIN table2
ON table1.matching_column = table2.matching_column;
```

Query

```
SELECT EMPLOYEE.EMP_NAME, PROJECT.DEPARTMENT
FROM EMPLOYEE
LEFT JOIN PROJECT
ON PROJECT.EMP_ID = EMPLOYEE.EMP_ID;
```

Output

EMP_NAME	DEPARTMENT
Angelina	Testing
Robert	Development
Christian	Designing

Kristen Development

Russell NULL

Marry NULL

RIGHT JOIN

In SQL, RIGHT JOIN returns all the values from the values from the rows of right table and the matched values from the left table. If there is no matching in both tables, it will return NULL.

Syntax

SELECT table1.column1, table1.column2, table2.column1,....

FROM table1

RIGHT JOIN table2

ON table1.matching_column = table2.matching_column;

Query

SELECT EMPLOYEE.EMP_NAME, PROJECT.DEPARTMENT

FROM EMPLOYEE

RIGHT JOIN PROJECT

ON PROJECT.EMP_ID = EMPLOYEE.EMP_ID;

Output

EMP_NAME DEPARTMENT

Angelina Testing

Robert Development

Christian Designing

Kristen Development

FULL JOIN

In SQL, FULL JOIN is the result of a combination of both left and right outer join. Join tables have all the records from both tables. It puts NULL on the place of matches not found.

Syntax

```
SELECT table1.column1, table1.column2, table2.column1,....
FROM table1
FULL JOIN table2
ON table1.matching_column = table2.matching_column;
```

Query

```
SELECT EMPLOYEE.EMP_NAME, PROJECT.DEPARTMENT
FROM EMPLOYEE
FULL JOIN PROJECT
ON PROJECT.EMP_ID = EMPLOYEE.EMP_ID;
```

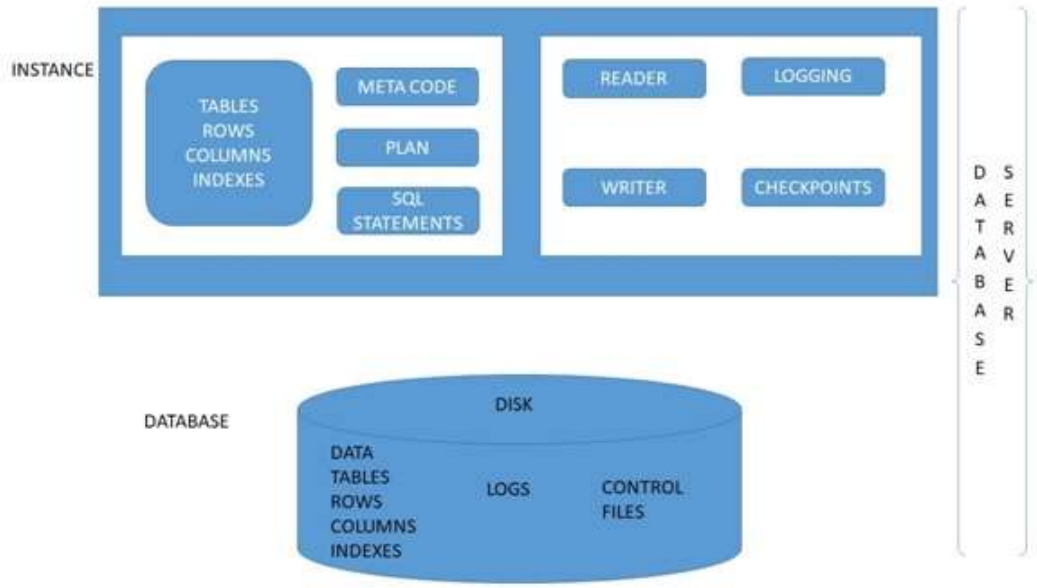
EMP_NAME	DEPARTMENT
Angelina	Testing
Robert	Development
Christian	Designing
Kristen	Development
Russell	NULL
Marry	NULL

Unit 5 :Relational Database Design

Relational database design (RDD) models' information and data into a set of tables with rows and columns. Each row of a relation/table represents a record, and each column represents an attribute of data. The Structured Query Language (SQL) is used to manipulate relational databases. The design of a relational database is composed of four stages, where the data are modeled into a set of related tables. The stages are −

- Define relations/attributes
- Define primary keys
- Define relationships
- Normalization

Relational databases differ from other databases in their approach to organizing data and performing transactions. In an RDD, the data are organized into tables and all types of data access are carried out via controlled transactions. Relational database design satisfies the ACID (atomicity, consistency, integrity, and durability) properties required from a database design. Relational database design mandates the use of a database server in applications for dealing with data management problems.



Relational Database Design Process

Database design is more art than science, as you have to make many decisions. Databases are usually customized to suit a particular application. No two customized applications are alike, and hence, no two databases are alike. Guidelines (usually in terms of what not to do instead of what to do) are provided in making these design decision, but the choices ultimately rest on the designer.

Step 1 − Define the Purpose of the Database (Requirement Analysis)

- Gather the requirements and define the objective of your database.
- Drafting out the sample input forms, queries and reports often help.

Step 2 − Gather Data, Organize in tables and Specify the Primary Keys

- Once you have decided on the purpose of the database, gather the data that are needed to be stored in the database. Divide the data into subject-based tables.
- Choose one column (or a few columns) as the so-called primary key, which uniquely identifies the each of the rows.

Step 3 − Create Relationships among Tables

A database consisting of independent and unrelated tables serves little purpose (you may consider using a spreadsheet instead). The power of a relational database lies in the relationship that can be defined between tables. The most crucial aspect in designing a relational database is to identify the relationships among tables. The types of relationship include:

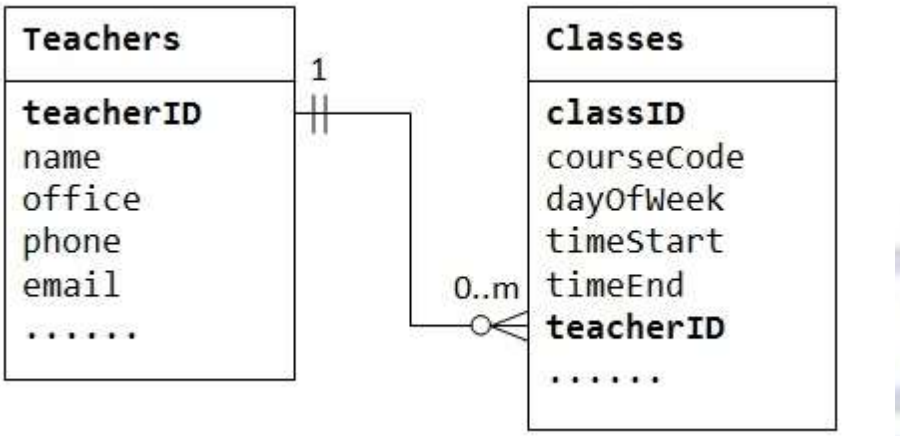
- one-to-many
- many-to-many
- one-to-one

One-to-Many

In a "class roster" database, a teacher may teach zero or more classes, while a class is taught by one (and only one) teacher. In a "company" database, a manager manages zero or more employees, while an employee is managed by one (and only one) manager. In a "product sales" database, a customer may place many orders; while an order is placed by one particular customer. This kind of relationship is known as one-to-many.

The one-to-many relationship cannot be represented in a single table. For example, in a "class roster" database, we may begin with a table called Teachers, which stores information about teachers (such as name, office, phone, and email). To store the classes taught by each teacher, we could create columns class1, class2, class3, but faces a problem immediately on how many columns to create. On the other hand, if we begin with a table called Classes, which stores information about a class, we could create additional columns to store information about the (one) teacher (such as name, office, phone, and email). However, since a teacher may teach many classes, its data would be duplicated in many rows in table Classes.

To support a one-to-many relationship, we need to design two tables: for e.g. a table Classes to store information about the classes with classID as the primary key; and a table Teachers to store information about teachers with teacherID as the primary key. We can then create the one-to-many relationship by storing the primary key of the table Teacher (i.e., teacherID) (the "one"-end or the parent table) in the table classes (the "many"-end or the child table), as illustrated below.



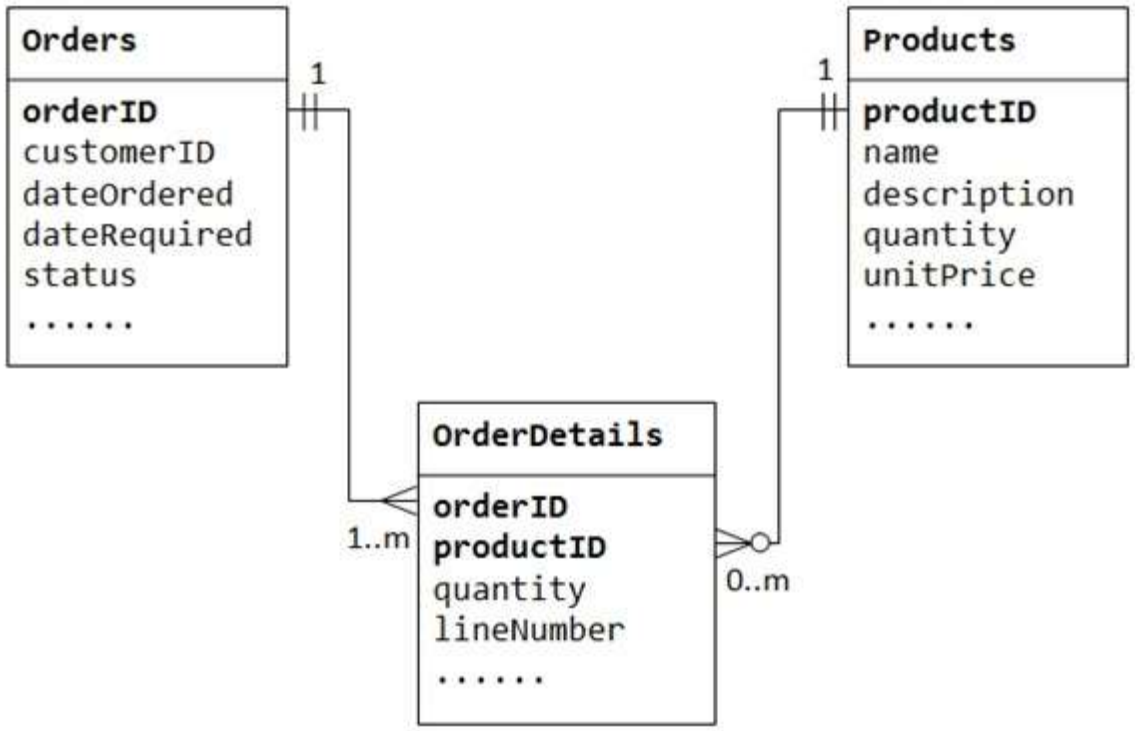
The column teacherID in the child table Classes is known as the foreign key. A foreign key of a child table is a primary key of a parent table, used to reference the parent table.

Many-to-Many

In a "product sales" database, a customer's order may contain one or more products; and a product can appear in many orders. In a "bookstore" database, a book is written by one or more authors; while an author may write zero or more books. This kind of relationship is known as many-to-many.

Let's illustrate with a "product sales" database. We begin with two tables: Products and Orders. The table products contain information about the products (such as name, description and quantityInStock) with productID as its primary key. The table orders contain customer's orders (customerID, dateOrdered, dateRequired and status). Again, we cannot store the items ordered inside the Orders table, as we do not know how many columns to reserve for the items. We also cannot store the order information in the Products table.

To support many-to-many relationship, we need to create a third table (known as a junction table), say OrderDetails (or OrderLines), where each row represents an item of a particular order. For the OrderDetails table, the primary key consists of two columns: orderID and productID, that uniquely identify each row. The columns orderID and productID in OrderDetails table are used to reference Orders and Products tables, hence, they are also the foreign keys in the OrderDetails table.



The many-to-many relationship is, in fact, implemented as two one-to-many relationships, with the introduction of the junction table.

An order has many items in OrderDetails. An OrderDetails item belongs to one particular order.

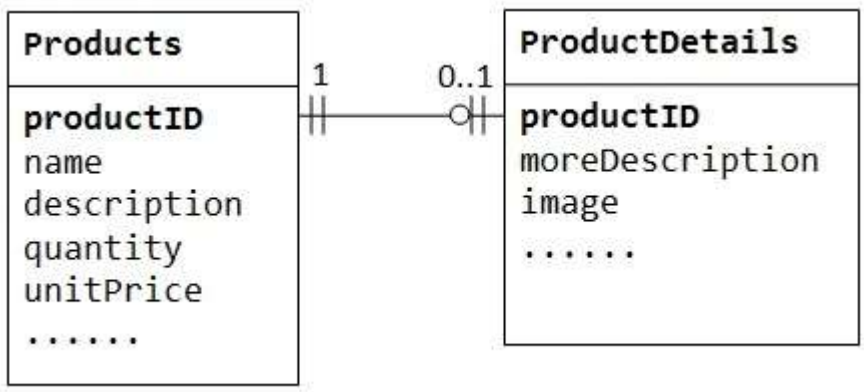
A product may appear in many OrderDetails. Each OrderDetails item specified one product.

One-to-One

In a "product sales" database, a product may have optional supplementary information such as image, more description and comment. Keeping them inside the Products table results in many empty spaces (in those records without these optional data). Furthermore, these large data may degrade the performance of the database.

Instead, we can create another table (say ProductDetails, ProductLines or ProductExtras) to store the optional data. A record will only be created for those products with optional data. The two tables, Products and ProductDetails, exhibit a one-to-one relationship. That is, for every row in the parent table, there is at most one row (possibly zero) in the child table. The same column productID should be used as the primary key for both tables.

Some databases limit the number of columns that can be created inside a table. You could use a one-to-one relationship to split the data into two tables. A one-to-one relationship is also useful for storing certain sensitive data in a secure table, while the non-sensitive ones in the main table.



Column Data Types

You need to choose an appropriate data type for each column. Commonly data types include integers, floating-point numbers, string (or text), date/time, binary, collection (such as enumeration and set).

Step 4 − Refine & Normalize the Design

For example,

- adding more columns,
- create a new table for optional data using one-to-one relationship,
- split a large table into two smaller tables,
- Other methods.

Normalization

Apply the so-called normalization rules to check whether your database is structurally correct and optimal.

First Normal Form (1NF): A table is 1NF if every cell contains a single value, not a list of values. This property is known as atomic. 1NF also prohibits a repeating group of columns such as item1, item2, itemN. Instead, you should create another table using a one-to-many relationship.

Second Normal Form (2NF) − A table is 2NF if it is 1NF and every non-key column is fully dependent on the primary key. Furthermore, if the primary key is made up of several columns, every non-key column shall depend on the entire set and not part of it.

For example, the primary key of the OrderDetails table comprising orderID and productID. If unitPrice is dependent only on productID, it shall not be kept in the OrderDetails table (but in the Products table). On the other hand, if the unit price is dependent on the product as well as the particular order, then it shall be kept in the OrderDetails table.

Third Normal Form (3NF) − A table is 3NF if it is 2NF and the non-key columns are independent of each other. In other words, the non-key columns are dependent on primary key, only on the primary key and nothing else. For example, suppose that we have a Products table with columns productID (primary key), name and unitPrice. The column discountRate shall not belong to the Products table if it is also dependent on the unitPrice, which is not part of the primary key.

Higher Normal Form: 3NF has its inadequacies, which leads to a higher Normal form, such as Boyce/Codd Normal form, Fourth Normal Form (4NF) and Fifth Normal Form (5NF), which is beyond the scope of this tutorial.

At times, you may decide to break some of the normalization rules, for performance reason (e.g., create a column called totalPrice in Orders table which can be derived from the orderDetails records); or because the end-user requested for it. Make sure that you fully aware of it, develop programming logic to handle it, and properly document the decision.

Integrity Rules

You should also apply the integrity rules to check the integrity of your design −

1. Entity Integrity Rule − The primary key cannot contain NULL. Otherwise, it cannot uniquely identify the row. For composite key made up of several columns, none of the columns can contain NULL. Most of the RDBMS check and enforce this rule.

2.Referential Integrity Rule − Each foreign key value must be matched to a primary key value in the table referenced (or parent table).

You can insert a row with a foreign key in the child table only if the value exists in the parent table.

If the value of the key changes in the parent table (e.g., the row updated or deleted), all rows with this foreign key in the child table(s) must be handled accordingly. You could either (a) disallow the changes; (b) cascade the change (or delete the records) in the child tables accordingly; (c) set the key value in the child tables to NULL.

Most RDBMS can be set up to perform the check and ensure the referential integrity, in a specified manner.

3.Business logic Integrity − Besides the above two general integrity rules, there could be integrity (validation) pertaining to the business logic, e.g., zip code shall be 5-digit within a certain ranges, delivery date and time shall fall in the business hours; quantity ordered shall be equal or less than quantity in stock, etc. These could be carried out invalidation rule (for the specific column) or programming logic.

Column Indexing

You could create an index on the selected column(s) to facilitate data searching and retrieval. An index is a structured file that speeds up data access for SELECT but may slow down INSERT, UPDATE, and DELETE. Without an index structure, to process a SELECT query with a

matching criterion (e.g., SELECT * FROM Customers WHERE name='Tan Ah Teck'), the database engine needs to compare every record in the table. A specialized index (e.g., in BTREE structure) could reach the record without comparing every record. However, the index needs to be rebuilt whenever a record is changed, which results in overhead associated with using indexes.

The index can be defined on a single column, a set of columns (called concatenated index), or part of a column (e.g., first 10 characters of a VARCHAR(100)) (called partial index). You could build more than one index in a table. For example, if you often search for a customer using either customerName or phone number, you could speed up the search by building an index on column customerName, as well as phoneNumber. Most RDBMS builds an index on the primary key automatically.

Edgar F Codd was a Computer Scientist who invented the Relational model for Database management. He is also credited with creating the foundation for relational databases as well as relational database management systems.

Codd's twelve rules define the characteristics required by a database management system to be considered relational i.e a relational database management system.

Rule 0: Foundation rule

For any system to be qualified as a relational database management system, it should manage its data using its relational capability.

Rule 1: Information rule

All information (including metadata) is represented in tables using rows and columns. The rows and columns have to be strictly unordered.

Rule 2: Guaranteed access rule

All values in a database should be accessible by using a combination of table name, primary key and column name. (Ability to access the data directly using a pointed is invalid according to this rule).

Rule 3: Systematic treatment of Null values

NULL values are fully supported in a database for representing absence of data or invalid data in a systematic way. NULL values are independent of data types and any operation on them must return NULL.

Rule 4: Dynamic Online Catalog

The catalog is the complete description of a database. It is stored online and gives additional information on the database. The query language that is used on the database is also used on the catalog.

Rule 5: Powerful and Well-Structured Language

A relational database may support multiple languages. However there should be at least one language that provides all types of data access, data manipulation etc. One such language is SQL.

Rule 6:View Updation rule

All the views that are updatable theoretically should also be updatable by the system.

Rule 7: Quality Insertion, Update, Deletion

There should be insertion, deletion and updation possible in terms of a single operand. This facility must be available at all level of relations.

Rule 8: Physical Data Independence

The physical changes to the system, such as change of storage space, change of access methods etc. should not affect the application programs and other activities of the system.

Rule 9: Logical Data Independence

The user view of the relational database should remain consistent even if there are changes to the logical structure. This rule is quite difficult to satisfy.

Rule 10: Integrity Independence

Integrity constraints that are particular to a relational database should be defined in that database's language and stored in the catalog. This rule means that relational DBMS is not reliant on front end.

Rule 11: Distribution Independence

Even if the data in the database is distributed across multiple locations, the end user should see the database as an uniform entity. Also, the database should work perfectly even if it is distributed across the network.

Rule 12: Nonsubversion Rule

If low level access is allowed for the relational database, it should not be able to subvert the integrity constraints and bypass security to change the data.

Normalization is a process of organizing the data in database to avoid data redundancy, insertion anomaly, update anomaly & deletion anomaly. Let's discuss about anomalies first then we will discuss normal forms with examples.

Anomalies in DBMS

There are three types of anomalies that occur when the database is not normalized. These are – Insertion, update and deletion anomaly. Let's take an example to understand this.

Example: Suppose a manufacturing company stores the employee details in a table named employee that has four attributes: emp_id for storing employee's id, emp_name for storing employee's name, emp_address for storing employee's address and emp_dept for storing the department details in which the employee works. At some point of time the table looks like this:

emp_id	emp_name	emp_address	emp_dept
101	Rick	Delhi	D001
101	Rick	Delhi	D002
123	Maggie	Agra	D890
166	Glenn	Chennai	D900
166	Glenn	Chennai	D004

The above table is not normalized. We will see the problems that we face when a table is not normalized.

Update anomaly: In the above table we have two rows for employee Rick as he belongs to two departments of the company. If we want to update the address of Rick then we have to update the same in two rows or the data will become inconsistent. If somehow, the correct address gets updated in one department but not in other then as per the database, Rick would be having two different addresses, which is not correct and would lead to inconsistent data.

Insert anomaly: Suppose a new employee joins the company, who is under training and currently not assigned to any department then we would not be able to insert the data into the table if emp_dept field doesn't allow nulls.

Delete anomaly: Suppose, if at a point of time the company closes the department D890 then deleting the rows that are having emp_dept as D890 would also delete the information of employee Maggie since she is assigned only to this department.

To overcome these anomalies we need to normalize the data. In the next section we will discuss about normalization.

Normalization

Here are the most commonly used normal forms:

First normal form(1NF)

Second normal form(2NF)

Third normal form(3NF)

Boyce & Codd normal form (BCNF)

First normal form (1NF)

As per the rule of first normal form, an attribute (column) of a table cannot hold multiple values. It should hold only atomic values.

Example: Suppose a company wants to store the names and contact details of its employees. It creates a table that looks like this:

emp_id	emp_name	emp_address	emp_mobile
101	Herschel	New Delhi	8912312390
102	Jon	Kanpur	8812121212
			9900012222
103	Ron	Chennai	7778881212
104	Lester	Bangalore	9990000123
			8123450987

Two employees (Jon & Lester) are having two mobile numbers so the company stored them in the same field as you can see in the table above.

This table is not in 1NF as the rule says “each attribute of a table must have atomic (single) values”, the emp_mobile values for employees Jon & Lester violates that rule.

To make the table complies with 1NF we should have the data like this:

emp_id	emp_name	emp_address	emp_mobile
101	Herschel	New Delhi	8912312390
102	Jon	Kanpur	8812121212
102	Jon	Kanpur	9900012222
103	Ron	Chennai	7778881212
104	Lester	Bangalore	9990000123
104	Lester	Bangalore	8123450987

Second normal form (2NF)

A table is said to be in 2NF if both the following conditions hold:

Table is in 1NF (First normal form)

No non-prime attribute is dependent on the proper subset of any candidate key of table.

An attribute that is not part of any candidate key is known as non-prime attribute.

Example: Suppose a school wants to store the data of teachers and the subjects they teach. They create a table that looks like this: Since a teacher can teach more than one subjects, the table can have multiple rows for a same teacher.

teacher_id	subject	teacher_age
111	Maths	38
111	Physics	38
222	Biology	38
333	Physics	40
333	Chemistry	40

Candidate Keys: {teacher_id, subject}

Non prime attribute: teacher_age

The table is in 1 NF because each attribute has atomic values. However, it is not in 2NF because non prime attribute teacher_age is dependent on teacher_id alone which is a proper subset of candidate key. This violates the rule for 2NF as the rule says “no non-prime attribute is dependent on the proper subset of any candidate key of the table”.

To make the table complies with 2NF we can break it in two tables like this:

teacher_details table:

teacher_id	teacher_age
111	38
222	38
333	40

teacher_subject table:

teacher_id	subject
111	Maths
111	Physics
222	Biology
333	Physics
333	Chemistry

Now the tables comply with Second normal form (2NF).

Third Normal form (3NF)

A table design is said to be in 3NF if both the following conditions hold:

Table must be in 2NF

Transitive functional dependency of non-prime attribute on any super key should be removed.

An attribute that is not part of any candidate key is known as non-prime attribute.

In other words 3NF can be explained like this: A table is in 3NF if it is in 2NF and for each functional dependency X-> Y at least one of the following conditions hold:

X is a super key of table

Y is a prime attribute of table

An attribute that is a part of one of the candidate keys is known as prime attribute.

Example: Suppose a company wants to store the complete address of each employee, they create a table named employee_details that looks like this:

emp_id	emp_name	emp_zip	emp_state	emp_city	emp_district
1001	John	282005	UP	Agra	Dayal Bagh
1002	Ajeet	222008	TN	Chennai	M-City
1006	Lora	282007	TN	Chennai	Urrapakkam
1101	Lilly	292008	UK	Pauri	Bhagwan
1201	Steve	222999	MP	Gwalior	Ratan

Super keys: {emp_id}, {emp_id, emp_name}, {emp_id, emp_name, emp_zip}…so on

Candidate Keys: {emp_id}

Non-prime attributes: all attributes except emp_id are non-prime as they are not part of any candidate keys.

Here, emp_state, emp_city & emp_district dependent on emp_zip. And, emp_zip is dependent on emp_id that makes non-prime attributes (emp_state, emp_city & emp_district) transitively dependent on super key (emp_id). This violates the rule of 3NF.

To make this table complies with 3NF we have to break the table into two tables to remove the transitive dependency:

employee table:

emp_id	emp_name	emp_zip
1001	John	282005
1002	Ajeet	222008
1006	Lora	282007
1101	Lilly	292008

1201	Steve	222999

employee_zip table:

emp_zip	emp_state	emp_city	emp_district
282005	UP	Agra	Dayal Bagh
222008	TN	Chennai	M-City
282007	TN	Chennai	Urrapakkam
292008	UK	Pauri	Bhagwan
222999	MP	Gwalior	Ratan

Boyce Codd normal form (BCNF)

It is an advance version of 3NF that's why it is also referred as 3.5NF. BCNF is stricter than 3NF. A table complies with BCNF if it is in 3NF and for every functional dependency X->Y, X should be the super key of the table.

Example: Suppose there is a company wherein employees work in more than one department. They store the data like this:

emp_id	emp_nationality	emp_dept	dept_type	dept_no_of_emp
1001	Austrian	Production and planning	D001	200
1001	Austrian	stores	D001	250
1002	American	design and technical support	D134	100
1002	American	Purchasing department	D134	600

Functional dependencies in the table above:

emp_id -> emp_nationality

emp_dept -> {dept_type, dept_no_of_emp}

Candidate key: {emp_id, emp_dept}

The table is not in BCNF as neither emp_id nor emp_dept alone are keys.

To make the table comply with BCNF we can break the table in three tables like this:

emp_nationality table:

emp_id	emp_nationality
1001	Austrian
1002	American

emp_dept table:

emp_dept	dept_type	dept_no_of_emp
Production and planning	D001	200
stores	D001	250
design and technical support	D134	100
Purchasing department	D134	600

emp_dept_mapping table:

emp_id	emp_dept
1001	Production and planning
1001	stores
1002	design and technical support

1002 Purchasing department

Functional dependencies:

emp_id -> emp_nationality

emp_dept -> {dept_type, dept_no_of_emp}

Candidate keys:

For first table: emp_id

For second table: emp_dept

For third table: {emp_id, emp_dept}

This is now in BCNF as in both the functional dependencies left side part is a key.