RELATIONAL DATA BASE MANAGEMENT SYSTEM (R.D.B.M.S)

Digital Notes By

BIGHNARAJ NAIK

Assistant Professor

Department of Master in Computer Application

VSSUT, Burla

Syllabus

4th SEMESTER MCA

MCA-208   RELATIONAL DATABASE MANAGEMENT SYSTEM (3-1-0)Cr.-4

Module I (10 hrs)

Database System Architecture – Data Abstraction, Data Independence, Data Definitions and Data Manipulation Languages.

Data models – Entity Relationship (ER), Mapping ER Model to Relational Mode, Network. Relational and Object Oriented Data Models, Integrity Constraints and Data Manipulation Operations.

Module II (10 hrs)

Relation Query Languages, Relational Algebra, Tuple and Domain Relational Calculus, SQL and QBE.
Relational Database Design: Domain and Data dependency, Armstrong's Axioms, Normal Forms, Dependency Preservation, Lossless design, Comparison of Oracle & DB2.

Module III (8 hrs)

Query Processing and Optimization: Evaluation of Relational Algebra Expressions, Query Equivalence, Join strategies, Query Optimization Algorithms.

Module IV (12 hrs)

Storage Strategies: Indices, B-Trees, Hashing, Transaction processing: Recovery and Concurrency Control, Locking and Timestamp based Schedulers, Multiversion and Optimistic Concurrency Control Schemes.
Advanced Topics: Object-Oriented and Object Relational databases. Logical Databases, Web Databases, Distributed Databases, Data Warehouse and Data Mining.

Text Books:

1. Database System Concepts by Sudarshan, Korth (McGraw-Hill Education)
2. Fundamentals of Database System By Elmasari &Navathe- Pearson Education

References Books:

(1) An introduction to Database System – Bipin Desai, Galgotia Publications
(2) Database System: concept, Design & Application by S.K.Singh (Pearson Education)
(3) Database management system by leon &leon (Vikas publishing House).
(4) Database Modeling and Design: Logical Design by Toby J. Teorey, Sam S. Lightstone, and Tom Nadeau, "", 4thEdition, 2005, Elsevier India Publications, New Delhi
(5) Fundamentals of Database Management System – Gillenson, Wiley India

Module1

File Management System

✓ INTRODUCTION

A **file system** (or **filesystem**) is an abstraction to store, retrieve and update a set of files. The term also identifies the data structures specified by some of those abstractions, which are designed to organize multiple files as a single stream of bytes, and the network protocols specified by some other of those abstractions, which are designed to allow files on a remote machine to be accessed.

The file system manages access to the data and the metadata of the files, and manages the available space of the device(s) which contain it. Ensuring reliability is a major responsibility of a file system. A file system organizes data in an efficient manner, and may be tuned to the characteristics of the backing device.

✓ FILENAMES

A **filename** (or **file name**) is used to identify a storage location in the file system. Most file systems have restrictions on the length of filenames. In some file systems, filenames are case-insensitive (i.e., filenames such as FOO and foo refer to the same file); in others, filenames are case-sensitive (i.e., the names FOO and foo refer to two separate files).

Most modern file systems allow filenames to contain a wide range of characters from the Unicode character set. Most file system interface utilities, however, have restrictions on the use of certain special characters, disallowing them within filenames (the file system may use these special characters to indicate a device, device type, directory prefix, or file type).

✓ DIRECTORIES

File systems typically have **directories** (also called **folders**) which allow the user to group files into separate collections. This may be implemented by associating the file name with an index in a table of contents or an inode in a Unix-like file system. Directory structures may be flat (i.e. linear), or allow hierarchies where directories may contain subdirectories. The first file system to support arbitrary hierarchies of directories was used in the Multics operating system.

✓ METADATA

The length of the data contained in a file may be stored as the number of blocks allocated for the file or as a byte count. The time that the file was last modified may be stored as the file's timestamp. File

systems might store the file creation time, the time it was last accessed, the time the file's meta-data was changed, or the time the file was last backed up. Other information can include the file's device type (e.g. block, character, socket, subdirectory, etc.), its owner user ID and group ID, its access permissions and other file attributes (e.g. whether the file is read-only, executable, etc.).

✓ PROS AND CONS OF CONVENTIONAL SYSTEM

Pros
• Easy to design because of their single-application.
• Excellent performance due to optimized organization for a single application.

Cons
• Harder to adapt to sharing across applications focus.
• Harder to adapt to new requirements.
• Need to duplicate attributes in several files.

DBMS Functions

A DBMS performs several important functions that guarantee the integrity and consistency of the data in the database. Most of those functions are transparent to end users, and most can be achieved only through the use of a DBMS. They include data dictionary management, data storage management, data transformation and presentation, security management, multiuser access control, backup and recovery management, data integrity management, database access languages and application programming interfaces and database communication interfaces. Each of these functions is explained below.

1. Data dictionary management.
The DBMS stores definitions of the data elements and their relationships (metadata) in a data dictionary. In turn, all programs that access the data in the database work through the DBMS. The DBMS uses the data dictionary to look up the required data component structures and relationships, thus relieving you from having to code such complex relationships in each program. Additionally, any changes made in a database structure are automatically recorded in the data dictionary, thereby freeing you from having to modify all of the programs that access the changed structure. In other words, the DBMS provides data abstraction, and it removes structural and data dependence from the system.

2. Data storage management.
The DBMS creates and manages the complex structures required for data storage, thus relieving you from the difficult task of defining and programming the physical data characteristics. A modern DBMS provides storage not only for the data, but also for related data entry forms or screen definitions, report definitions, data validation rules, procedural code, structures to handle video and picture formats, and so on. Data storage management is also important for database performance

tuning. Performance tuning relates to the activities that make the database perform more efficiently in terms of storage and access speed.

3. Data transformation and presentation.

The DBMS transforms entered data to conform to required data structures. The DBMS relieves you of the chore of making a distinction between the logical data format and the physical data format. That is, the DBMS formats the physically retrieved data to make it conform to the user's logical expectations. For example, imagine an enterprise database used by a multinational company. An end user in England would expect to enter data such as July 11, 2010, as "11/07/2010." In contrast, the same date would be entered in the United States as "07/11/2010." Regardless of the data presentation format, the DBMS must manage the date in the proper format for each country.

4. Security management.

The DBMS creates a security system that enforces user security and data privacy. Security rules determine which users can access the database, which data items each user can access, and which data operations (read, add, delete, or modify) the user can perform. This is especially important in multiuser database systems.

5. Multiuser access control.

To provide data integrity and data consistency, the DBMS uses sophisticated algorithms to ensure that multiple users can access the database concurrently without compromising the integrity of the database.

6. Backup and recovery management.

The DBMS provides backup and data recovery to ensure data safety and integrity. Current DBMS systems provide special utilities that allow the DBA to perform routine and special backup and restore procedures. Recovery management deals with the recovery of the database after a failure, such as a bad sector in the disk or a power failure. Such capability is critical to preserving the database's integrity.

7. Data integrity management.

The DBMS promotes and enforces integrity rules, thus minimizing data redundancy and maximizing data consistency. The data relationships stored in the data dictionary are used to enforce data integrity. Ensuring data integrity is especially important in transaction-oriented database systems.

8. Database access languages and application programming interfaces.

The DBMS provides data access through a query language. A query language is a nonprocedural language—one that lets the user specify what must be done without having to specify how it is to be done. Structured Query Language (SQL) is the de facto query language and data access standard supported by the majority of DBMS vendors.

9. Database communication interfaces.

Current-generation DBMSs accept end-user requests via multiple, different network environments. For example, the DBMS might provide access to the database via the Internet through the use of Web browsers such as Mozilla Firefox or Microsoft Internet Explorer. In this environment, communications can be accomplished in several ways:

- End users can generate answers to queries by filling in screen forms through their preferred Web

6

browser.
- The DBMS can automatically publish predefined reports on a Website.
- The DBMS can connect to third-party systems to distribute information via e-mail or other productivity applications.

✓ TYPES OF FILE SYSTEMS

File system types can be classified into disk/tape file systems, network file systems and special-purpose file systems.

- ### Disk file systems

A *disk file system* takes advantages of the ability of disk storage media to randomly address data in a short amount of time. Additional considerations include the speed of accessing data following that initially requested and the anticipation that the following data may also be requested. This permits multiple users (or processes) access to various data on the disk without regard to the sequential location of the data. Examples include FAT (FAT12, FAT16, FAT32), exFAT, NTFS, HFS and HFS+, HPFS, UFS, ext2, ext3, ext4, btrfs, ISO 9660, Files-11, Veritas File System, VMFS, ZFS, ReiserFS and UDF.

- ### Optical discs

ISO 9660 and Universal Disk Format (UDF) are two common formats that target Compact Discs, DVDs and Blu-ray discs. Mount Rainier is an extension to UDF supported by Linux 2.6 series and Windows Vista that facilitates rewriting to DVDs.

- ### Flash file systems

A *flash file system* considers the special abilities, performance and restrictions of flash memory devices. Frequently a disk file system can use a flash memory device as the underlying storage media but it is much better to use a file system specifically designed for a flash device.

- ### Tape file systems

- A *tape file system* is a file system and tape format designed to store files on tape in a self-describing form. Magnetic tapes are sequential storage media with significantly longer random data access times than disks, posing challenges to the creation and efficient management of a general-purpose file system.

- In a disk file system there is typically a master file directory, and a map of used and free data regions. Any file additions, changes, or removals require updating the directory and the used/free maps. Random access to data regions is measured in milliseconds so this system works well for disks.
- Tape requires linear motion to wind and unwind potentially very long reels of media. This tape motion may take several seconds to several minutes to move the read/write head from one end of the tape to the other.
- Consequently, a master file directory and usage map can be extremely slow and inefficient with tape. Writing typically involves reading the block usage map to find free blocks for writing, updating the usage map and directory to add the data, and then advancing the tape to write the data in the correct spot. Each additional file write requires updating the map and directory and writing the data, which may take several seconds to occur for each file.
- Tape file systems instead typically allow for the file directory to be spread across the tape intermixed with the data, referred to as *streaming*, so that time-consuming and repeated tape motions are not required to write new data.

✓ IMPORTANCE OF FILE ORGANISATION IN DATABASE

To implement a database efficiently, there are several design tradeoffs needed. One of the most important ones is the file Organisation. For example, if there were to be an application that required only sequential batch processing, then the use of indexing techniques would be pointless and wasteful.

There are several important consequences of an inappropriate file Organisation being used in a database. Thus using replication would be wasteful of space besides posing the problem of inconsistency in the data. The wrong file Organisation can also–

• Mean much larger processing time for retrieving or modifying the required record
• Require undue disk access that could stress the hardware

✓ FILE MANAGEMENT SYSTEM PROBLEMS
•Data redundancy
•Data Access: New request-new program
•Data is not isolated from the access implementation
•Concurrent program execution on the same file
•Difficulties with security enforcement
•Integrity issues .

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

- **Integrity problems**. The data values stored in the database must satisfy certain types of **consistency constraints**. For example, the balance of a bank account may never fall below a prescribed amount (say, $25). Developers enforce these constraints in the system by adding appropriate code in the various application programs. However, when new constraints are added, it is difficult to change the programs to enforce them. The problem is compounded when constraints involve several data items from different files.
- **Atomicity problems**. A computer system, like any other mechanical or electrical device, is subject to failure. In many applications, it is crucial that, if a failure occurs, the data be restored to the consistent state that existed prior to the failure. Consider a program to transfer $50 from account *A* to account *B*. If a system failure occurs during the execution of the program, it is possible that the $50 was removed from account *A*but was not credited to account *B*, resulting in an inconsistent database state. Clearly, it is essential to database consistency that either both the credit and debit occur, or that neither occur. That is, the funds transfer must be *atomic—*it must happen in its entirety or not at all. It is difficult to ensure atomicity in a conventional file-processing system.
- **Concurrent-access anomalies**. For the sake of overall performance of the system and faster response, many systems allow multiple users to update the data simultaneously. In such an environment, interaction of concurrent updates may result in inconsistent data. Consider bank account *A*, containing $500. If two customers withdraw funds (say $50 and $100 respectively) from account *A* at about the same time, the result of the concurrent executions may leave the account in an incorrect (or inconsistent) state. Suppose that the programs executing on behalf of each withdrawal read the old balance, reduce that value by the amount being withdrawn, and write the result back. If the two programs run concurrently, they may both read the value $500, and write back $450 and $400, respectively. Depending on which one writes the value last, the account may contain either $450 or $400, rather than the correct value of $350. To guard against this possibility, the system must maintain some form of supervision. But supervision is difficult to provide because data may be accessed by many different application programs that have not been coordinated previously.
- **Security problems**. Not every user of the database system should be able to access all the data. For example, in a banking system, payroll personnel need to see only that part of the database that has information about the various bank employees. They do not need access to information about customer accounts. But, since application programs are added to the system in an ad hoc manner, enforcing such security constraints is difficult.

These difficulties, among others, prompted the development of database systems. In what follows, we shall see the concepts and algorithms that enable database systems to solve the problems with file-processing systems. In most of this book, we use a bank enterprise as a running example of a typical data-processing application found in a corporation.

✓ HIERARCHY OF DATA

Data are the principal resources of an organization. Data stored in computer systems form a hierarchy extending from a single bit to a database, the major record-keeping entity of a firm. Each higher rung of this hierarchy is organized from the components below it.

Data are logically organized into:

1. Bits (characters)

2. Fields

3. Records

4. Files

5. Databases

- *Bit* (Character) - a bit is the smallest unit of data representation (value of a bit may be a 0 or 1). Eight bits make a byte which can represent a character or a special symbol in a character code.
- *Field* - a field consists of a grouping of characters. A data field represents an attribute (a characteristic or quality) of some entity (object, person, place, or event).
- *Record* - a record represents a collection of attributes that describe a real-world entity. A record consists of fields, with each field describing an attribute of the entity.
- *File* - a group of related records. Files are frequently classified by the application for which they are primarily used (employee file). A *primary key* in a file is the field (or fields) whose value identifies a record among others in a data file.

Magnetic disk

- The primary computer storage device. Like tape, it is magnetically recorded and can be re-recorded over and over. Disks are rotating platters with a mechanical arm that moves a read/write head between the outer and inner edges of the platter's surface. It can take as long as one second to find a location on a floppy disk to as little as a couple of milliseconds on a fast hard disk. See hard disk for more details.

Tracks and Spots
The disk surface is divided into concentric tracks (circles within circles). The thinner the tracks, the more storage. The data bits are recorded as tiny magnetic spots on the tracks. The smaller the spot, the more bits per inch and the greater the storage.

Sectors

Tracks are further divided into sectors, which hold a block of data that is read or written at one time; for example, READ SECTOR 782, WRITE SECTOR 5448. In order to update the disk, one or more sectors are read into the computer, changed and written back to disk. The operating system figures out how to fit data into these fixed spaces.

Modern disks have more sectors in the outer tracks than the inner ones because the outer radius of the platter is greater than the inner radius
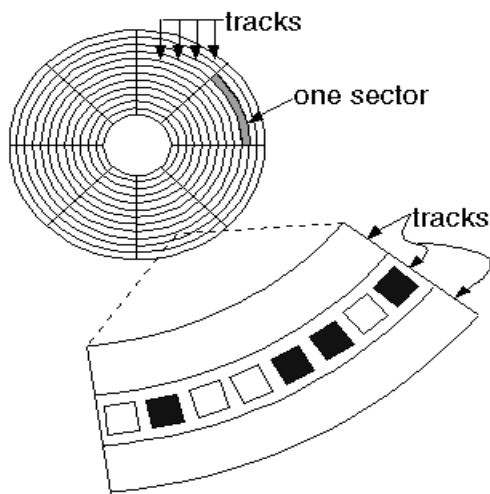
Magnetic tape

A sequential storage medium used for data collection, backup and archiving. Like videotape, computer tape is made of flexible plastic with one side coated with a ferromagnetic material. Tapes were originally open reels, but were superseded by cartridges and cassettes of many sizes and shapes.

Tape has been more economical than disks for archival data, but that is changing as disk capacities have increased enormously. If tapes are stored for the duration, they must be periodically recopied or the tightly coiled magnetic surfaces may contaminate each other.

Sequential Medium

The major drawback of tape is its sequential format. Locating a specific record requires reading every record in front of it or searching for markers that identify predefined partitions. Although most tapes are used for archiving rather than routine updating, some drives allow rewriting in place if the byte count does not change. Otherwise, updating requires copying files from the original tape to a

blank tape (scratch tape) and adding the new data in between.

Track Formats
Tracks run parallel to the edge of the tape (linear recording) or diagonally (helical scan). A linear variation is serpentine recording, in which the tracks "snake" back and forth from the end of the tape to the beginning.
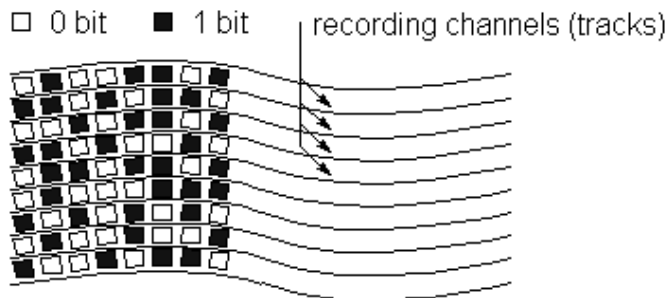
Legacy open reel tapes used nine linear tracks (8 bits plus parity), while modern cartridges use 128 or more tracks. Data are recorded in blocks of contiguous bytes, separated by a space called an "interrecord gap" or "interblock gap." Tape drive speed is measured in inches per second (ips). Over the years, storage density has increased from 200 to 38,000 bpi.

✓ FILE ORGANIZATION

Data files are organized so as to facilitate access to records and to ensure their efficient storage. A tradeoff between these two requirements generally exists: if rapid access is required, more storage is required to make it possible.

Access to a record for reading it is the essential operation on data. There are two types of access:

1. *Sequential access* - is performed when records are accessed in the order they are stored. Sequential access is the main access mode only in batch systems, where files are used and updated at regular intervals.

2. *Direct access* - on-line processing requires direct access, whereby a record can be accessed without accessing the records between it and the beginning of the file. The primary key serves to identify the needed record.

There are three methods of file organization:

1. Sequential organization

2. Indexed-sequential organization

3. Direct organization

RAID :

RAID is short for *redundant array of independent (or inexpensive) disks.*It is a category of disk drives that employ two or more drives in combination for fault tolerance and performance. RAID disk drives are used frequently on servers but aren't generally necessary for personal computers. RAID allows you to store the same data redundantly (in multiple paces) in a balanced way to improve overall storage performance.

Different RAID Levels

Different architectures are named RAID followed by a number and each architecture provides a different balance between performance, capacity and tolerance. There are number of different RAID levels including the following;

Level 0: Striped Disk Array without Fault Tolerance

Provides *data striping*(spreading out blocks of each file across multiple disk drives) but no redundancy. This improves performance but does not deliver fault tolerance. If one drive fails then all data in the array is lost.
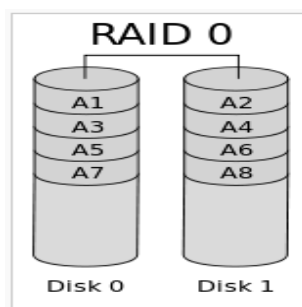
Level 1: Mirroring and Duplexing

Provides disk mirroring. Level 1 provides twice the read transaction rate of single disks and the same write transaction rate as single disks. The traditional solution, called mirroring or shadowing, uses

twice as many disks as a non-redundant disk array. whenever data is written to a disk the same data is also written to a redundant disk, so that there are always two copies of the information.
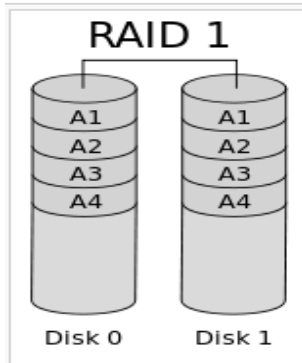
When data is read, it can be retrieved from the disk with the shorter queuing, seek and rotational delays. If a disk fails, the other copy is used to service requests. Mirroring is frequently used in database applications where availability and transaction time are more important than storage efficiency.

Level 2: Error-Correcting Coding

Not a typical implementation and rarely used, Level 2 stripes data at the bit level rather than the block level. Memory systems have provided recovery from failed components with much less cost than mirroring by using Hamming codes. Hamming codes contain parity for distinct overlapping subsets of components. In one version of this scheme, four disks require three redundant disks, one less than mirroring. Since the number of redundant disks is proportional to the log of the total number of the disks on the system, storage efficiency increases as the number of data disks increases.
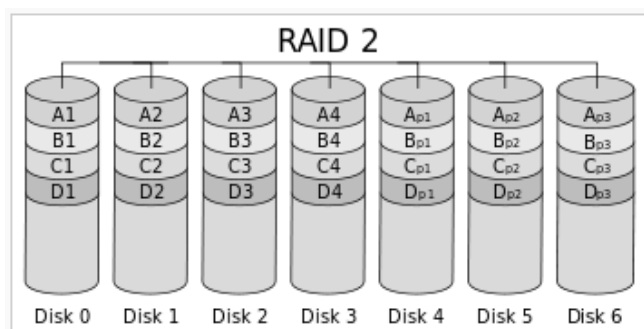

If a single component fails, several of the parity components will have inconsistent values, and the failed component is the one held in common by each incorrect subset. The lost information is recovered by reading the other components in a subset, including the parity component, and setting

the missing bit to 0 or 1 to create proper parity value for that subset. Thus, multiple redundant disks are needed to identify the failed disk, but only one is needed to recover the lost information.

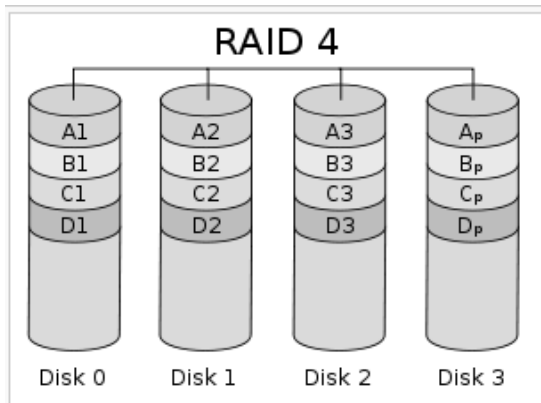## Level 3: Bit-Interleaved Parity

Provides byte-level striping with a dedicated parity disk. Level 3, which cannot service simultaneous multiple requests, also is rarely used. In a bit-interleaved, parity disk array, data is conceptually interleaved bit-wise over the data disks, and a single parity disk is added to tolerate any single disk failure. Each read request accesses all data disks and each write request accesses all data disks and the parity disk.
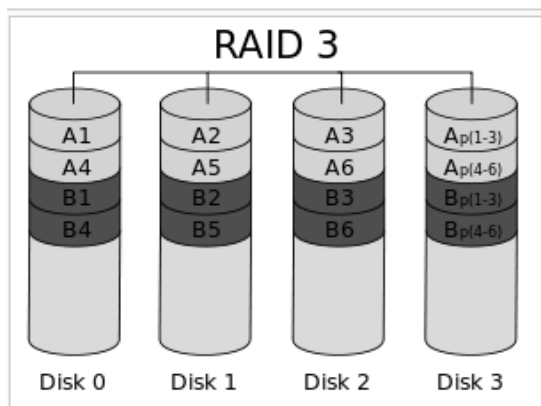
Thus, only one request can be serviced at a time. Because the parity disk contains only parity and no data, the parity disk cannot participate on reads, resulting in slightly lower read performance than for redundancy schemes that distribute the parity and data over all disks. Bit-interleaved, parity disk arrays are frequently used in applications that require high bandwidth but not high I/O rates.

Level 4: Dedicated Parity Drive

A commonly used implementation of RAID, Level 4 provides block-level striping (like Level 0) with a parity disk. If a data disk fails, the parity data is used to create a replacement disk. A disadvantage to Level 4 is that the parity disk can create write bottlenecks.

RAID 4

Level 5: Block Interleaved Distributed Parity

Provides data striping at the byte level and also stripe error correction information. This results in excellent performance and good fault tolerance. Level 5 is one of the most popular implementations of RAID.
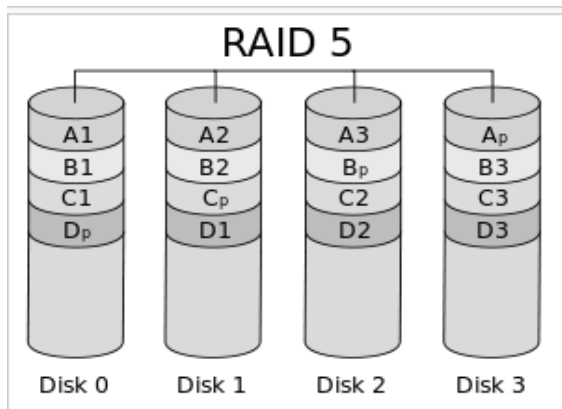
RAID 5

Level 6: Independent Data Disks with Double Parity

RAID Level 6 is similiar to RAID 5 (striped parity) except instead of one parity block per stripe there are two. With two independent parity blocks, RAID 6 can survive the loss of two disks in the group.
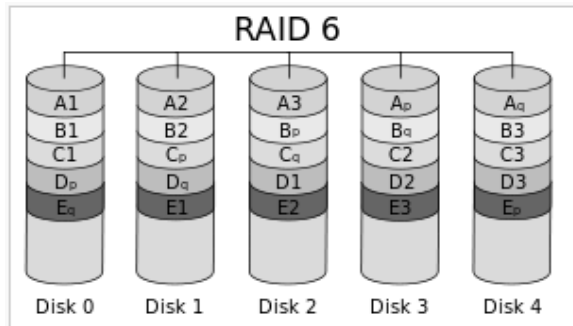
Level 0+1: A Mirror of Stripes

Not one of the original RAID levels, two RAID 0 stripes are created, and a RAID 1 mirror is created over them. Used for both replicating and sharing data among disks.

Level 10: A Stripe of Mirrors

Not one of the original RAID levels, multiple RAID 1 mirrors are created, and a RAID 0 stripe is created over these. RAID 10 uses more disk space to provide redundant data than RAID 5. However, it also provides a performance advantage by reading from all disks in parallel while eliminating the write penalty 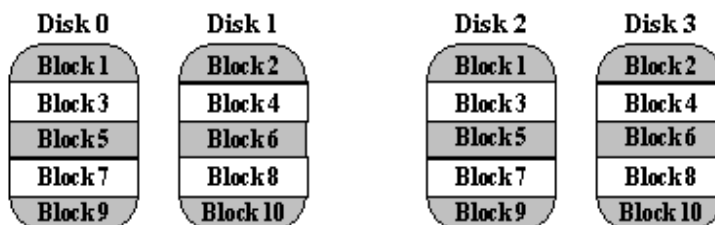of RAID 5. In addition, RAID 10 gives better performance than RAID 5 while a failed drive remains unreplaced. Under RAID 5, each attempted read of the failed drive can be performed only by reading all of the other disks.

On RAID 10, a failed disk can be recovered by a single read of its mirrored pair.

✓ SEQUENTIAL ORGANIZATION

In sequential organization records are physically stored in a specified order according to a key field in each record.The most basic way to organize the collection of records that from a file is to use sequential organization. In a sequentially organized file records are written consecutively when the file is created and must be accessed consecutively when the file is later used for input (figure 2).

```
Begnning of ——————▶  Record 1
File
                       Record 2
                          ⋮

                          ⋮

                       Record n-1
                       Record N
End of file  ——————▶
```
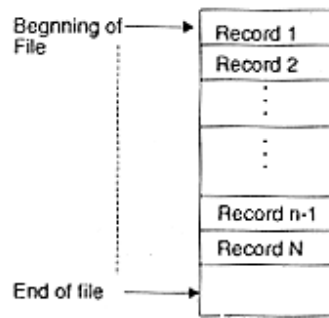
2. Structure of sequential file

In a sequential file, records are maintained in the logical sequence of their primary key values. The processing of a sequential file is conceptually simple but inefficient for random access. However, if access to the file is strictly sequential, a sequential file is suitable. A sequential file could be stored on a sequential storage device such as a magnetic tape.

Search for a given record in a sequential file requires, on average, access to half the records in the file. Consider a system where the file is stored on a direct access device such as a disk. Suppose the key value is separated from the rest of the record and a pointer is used to indicate the location of the record. In such a system, the device may scan over the key values at rotation speeds and only read in the desired record. A binary or logarithmic search technique may also be used to search for a record. In this method, the cylinder on which the required record is stored is located by a series of decreasing head movements. The search having been localized to a cylinder may require the reading of half the tracks, on average, in the case where keys are embedded in the physical records, or require only a scan over the tracks in the case where keys are also stored separately.

Updating usually requires the creation of a new file. To maintain file sequence, records are copied to the point where amendment is required. The changes are then made and copied into the new file. Following this, the remaining records in the original file are copied to the new file. This method of updating a sequential file creates an automatic backup copy. It permits updates of the type U1 through U4.

Addition can be handled in a manner similar to updating. Adding a record necessitates the shifting of all records from the appropriate point to the end of file to create space for the new record. Inversely, deletion of a record requires a compression of the file space, achieved by the shifting of records. Changes to an existing record may also require shifting if the record size expands or shrinks.

The basic advantage offered by a sequential file is the ease of access to the next record, the simplicity of organization and the absence of auxiliary data structures. However, replies to simple queries are time consuming for large files. Updates, as seen above, usually require the creation of a

18

new file. A single update is an expensive proposition if a new file must be created. To reduce the cost per update, all such requests are hatched, sorted in the order of the sequen1tial file, and then used to update the sequential file in a single pass. Such a file, containing the updates to be made to a sequential file, is sometimes referred to a transaction file.

In the batched mode of updating, a transaction file of update records is made and then sorted in the sequence of the sequential file. Ale update process requires the examination of each individual record in the original sequential file (the old master rile). Records requiring no changes are copied directly to a new file (the new master rile); records requiring one or Wore changes are written into the new master file only after all necessary changes have been made. Insertions of new records are made in the proper sequence. They are written into the new master file at the appropriate place. Records to be deleted are not copied to the new master file. A big advantage of this method of update is the creation of an automatic backup copy. The new master file can always be recreated by processing the old master file and the transaction file.
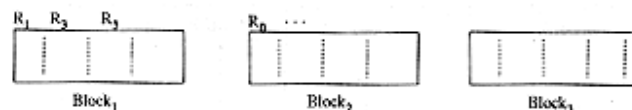
3 : A file with empty spaces for record insertions

A possible method of reducing the creation of a new file at each update run is to create the original file with "holes" (space left for the addition of new records, as shown in the last figure). As such, if a block could hold K records, then at initial creation it is made to contain only L * K records, where $0 < L < 1$ is known as the loading factor. Additional space may also be earmarked for records that may "overflow" their blocks, e.g., if the record ri logically belongs to block Bi but the physical block Bi does not contain the requisite free space. This additional free space is known as the overflow area. A similar technique is employed in index-sequential files.

Advantages of sequential access:

1. It is fast and efficient when dealing with large volumes of data that need to be processed periodically (batch system).

Disadvantages of sequential access:

1. Requires that all new transactions be sorted into the proper sequence for sequential access processing.

2. Locating, storing, modifying, deleting, or adding records in the file requires rearranging the file.

3. This method is too slow to handle applications requiring immediate updating or responses.

✓ INDEXED-SEQUENTIAL ORGANIZATION

In the indexed-sequential files method, records are physically stored in sequential order on a magnetic disk or other direct access storage device based on the key field of each record. Each file contains an index that references one or more key fields of each data record to its storage location address.

The retrieval of a record from a sequential file, on average, requires access to half the records in the file, making such inquiries not only inefficient but very time consuming for large files. To improve the query response time of a sequential file, a type of indexing technique can be added.

An index is a set of y, address pairs. Indexing associates a set of objects to a set of orderable quantities, which are usually smaller in number or their properties provide a mechanism for faster search. The purpose of indexing is to expedite the search process. Indexes created from a sequential (or sorted) set of primary keys are referred to as index sequential. Although the indices and the data blocks are held together physically, we distinguish between them logically. We shall use the term index file to describe the indexes and data file to refer to the data records. The index is usually small enough to be read into the processor memory.

✓ TYPES OF INDEXES

The idea behind an index access structure is similar to that behind the indexes used commonly in textbooks. A textbook index lists important terms at the end of the book in alphabetic order. Along with each term, a list of page numbers where the term appears is given. We can search the index to find a list of addresses - page numbers in this case - and use these addresses to locate the term in the textbook by searching the specified pages. The alternative, if no other guidance is given, is to sift slowly through the whole textbooks word by word to find the term we are interested in, which corresponds to doing a linear search on a file. Of course, most books do have additional information, such as chapter and section titles, which can help us find a term without having to search through the whole book. However, the index is the only exact indication of where each term occurs in the book.

An index is usually defined on a single field of a file, called an indexing Field. The index typically stores each value of the index field along with a list of pointers to all disk blocks that contain a record with that field value. The values in the index are ordered so that we can do a binary search on the index. The index file is much smaller than the data file, so searching the index using binary

search is reasonably efficient. Multilevel indexing does away with the need for binary search at the expense of creating indexes to the index itself!

There are several types of indexes. A primary index is an index specified on the ordering key field of an ordered file of records. Recall that an ordering key field is used to physically order the file records on disk, and every record has a unique value for that field. If the ordering field is not a key field that is, several records in the file can have the same value for the ordering field another type of index, called a clustering index, can be used. Notice that a file can have at most one physical ordering field, so it can have at most one primary index or one clustering index, but not both. A third type of index, called a secondary index, can be specified on any non-ordering field of a file. A file can have several secondary indexes in addition to its primary access method. In the next three subsections we discuss these three types of indexes.

✓ PRIMARY INDEXES

A **primary index** is an ordered file whose records are of fixed length with two fields. The first field is of the same data types as the ordering key field of the data file, and the second field is a pointer to a disk block - a block address. The ordering key field is called the primary key of the data file there is one index entry (or index record) in the index file for each block in the data file. Each index entry has the value of the primary key field for the first record in a block and a pointer to other block as its two field values. We will refer to the two field values of index entry i as K (i), P (i).

To create a primary index on the ordered file shown in figure 4, we use the Name field as the primary key, because that is the ordering key field of the file (assuming that each value of NAME is unique).

Figure 4 : Some blocks on an ordered (sequential) file of EMPLOYEE records with NAME as the ordering field

Each entry in the index will have a NAME value and a pointer. The first three index entries would be:

<K(1) = (Aaron, Ed), P(I)= address of block 1 >

<K(2) = (Adams, John), P(I) = address of block 2 >

<K(3) = (Alexander, Fd), P(3) = address of block 3 >
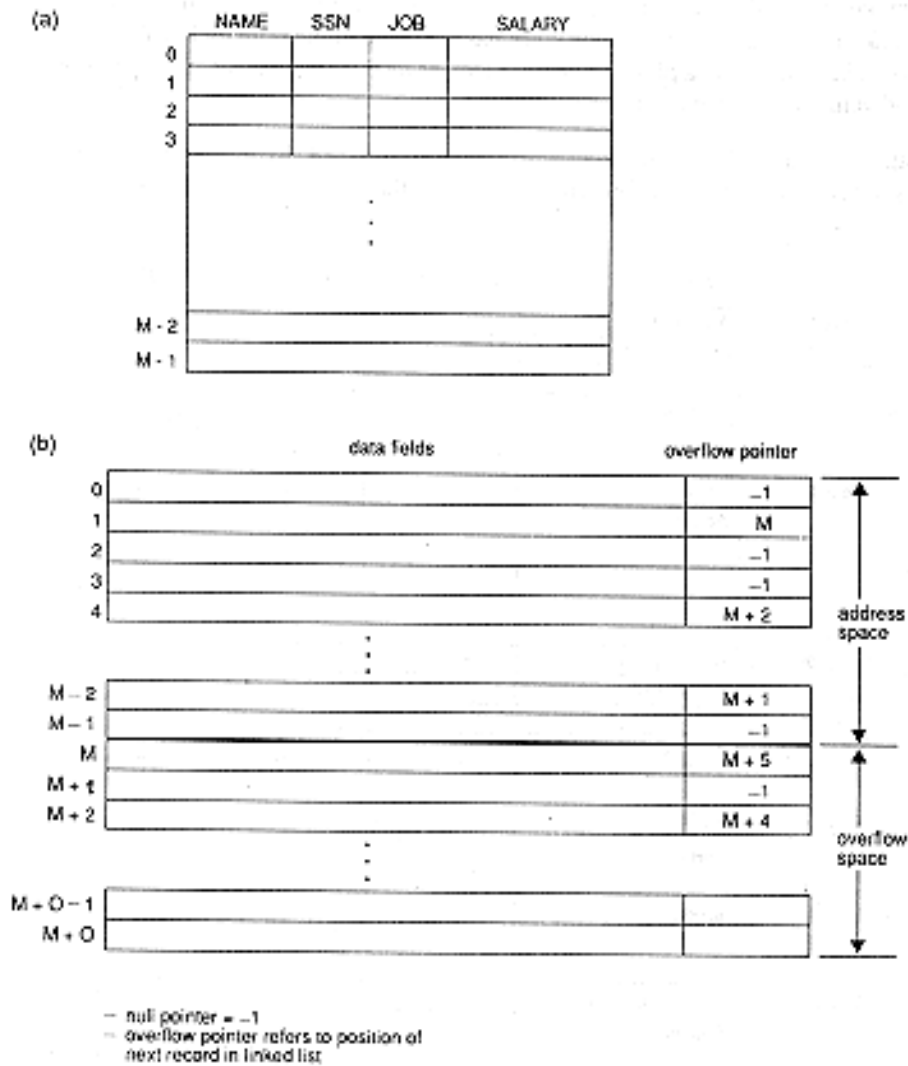
Figure 5 : Illustrating internal hashing data Structures. (a) Array of M Positions for use in hashing. (b) Collision resolution by chaining of records.

Figure 6 illustrates this Primary index. The total number of entries in the index will be the same as the number of disk block in the ordered data file.The first record in each block of the data file. The first record in each block of the data file is called the anchor record of the block, or simple the block

anchor (a scheme called the anchor record of similar to the one described here can be used, with the last record in each block, rather than the first as the block anchor. A primary index is an example of what is called a non-dense index because it includes an entry for each disk block of the data file rather than for every record in the data file. A dense index, on the other hand, contains an entry for every record in the file.

The index file for a primary index needs substantially fewer blocks than the data file for two reasons. First, there are fewer index entries than there are records in the data file because an entry exists for each whole block of the data file rather than for each record. Second, each index entry is typically smaller in size than a data record because it has only two fields, so more index entries than data records will fit in one block. A binary search on the index file will hence require fewer block accesses than a binary search on the data file.

A record whose primary key value is K will be in the block whose address is P(i), where Ki < K< (i + 1). The ith block in the data file contains all such records because of the physical ordering of the file records on the primary key field, we do a binary search on the index file to find the appropriate index entry i, then retrieve the data file block whose address is P(i).
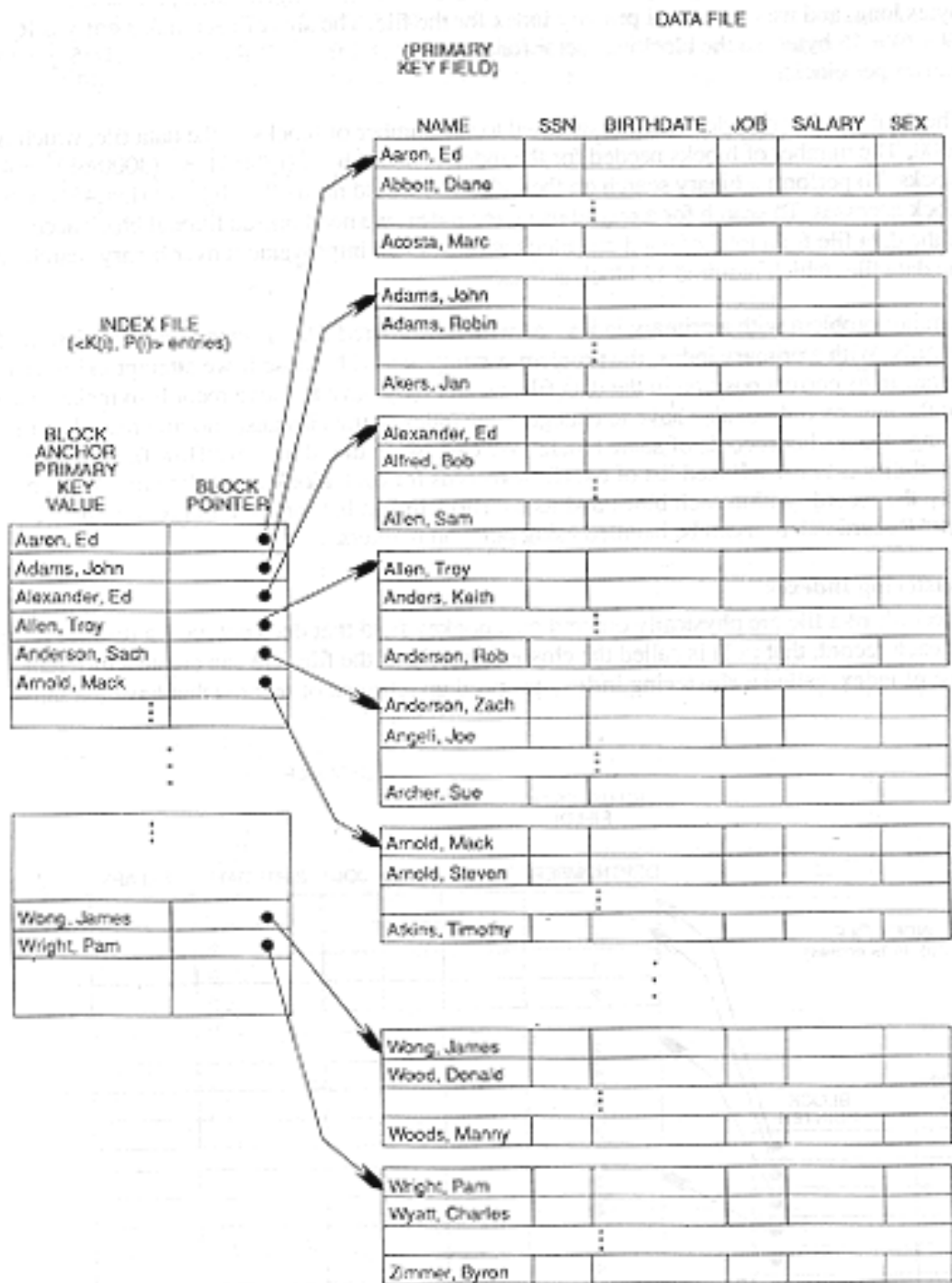
Figure 6 : Primary index on the ordering key field of the file shown in figure 5

✓ CLUSTERING INDEXES

If records of a file are physically ordered on a non-key field that does not have a distinct value, for each record, that field is called the clustering field of the file. We can create a different type of index, called a clustering index, to speed up retrieval of records that have the same value for the clustering field.
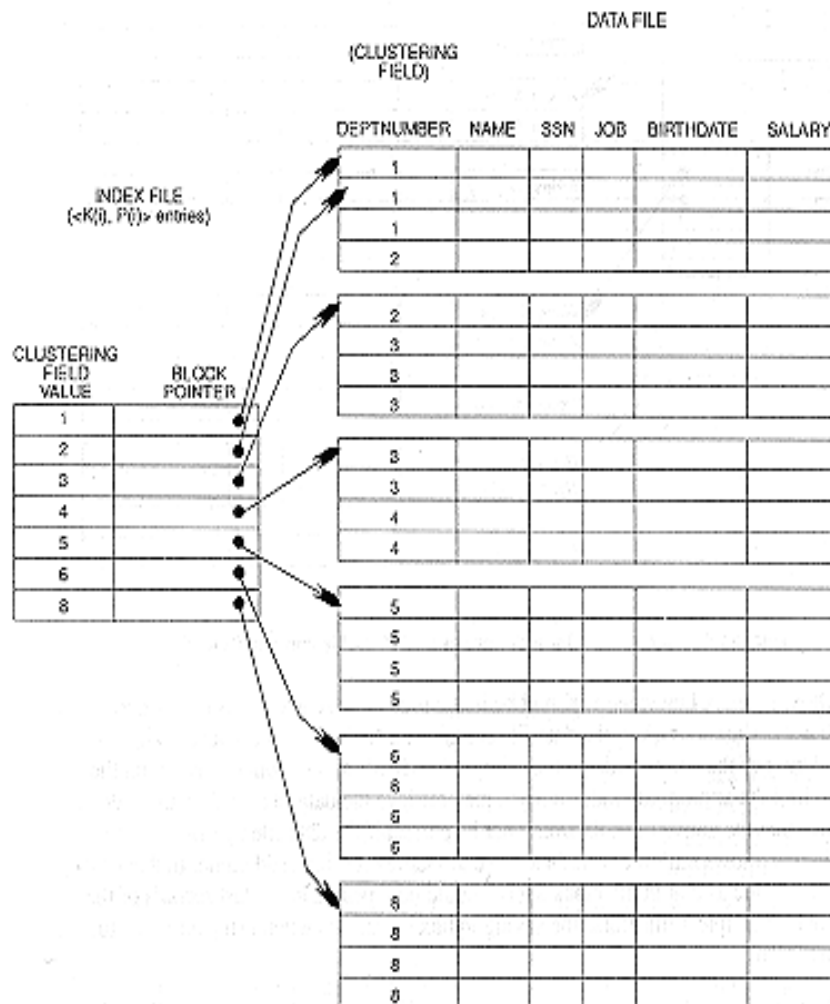
Figure 7 : A clustering Index on the DEPTNUMBER ordering field of an EMPLOYEE file

This differs from a primary index, which requires that the ordering field of the data file have a distinct value for each record.

A clustering index is also an ordered file with two fields; the first field is of the same type as the clustering field of the data file and the second field is a block pointer. There is one entry in the clustering index for each distinct value of the clustering field, containing that value and a pointer to the first block in the data file that has a record with that value for its clustering field.
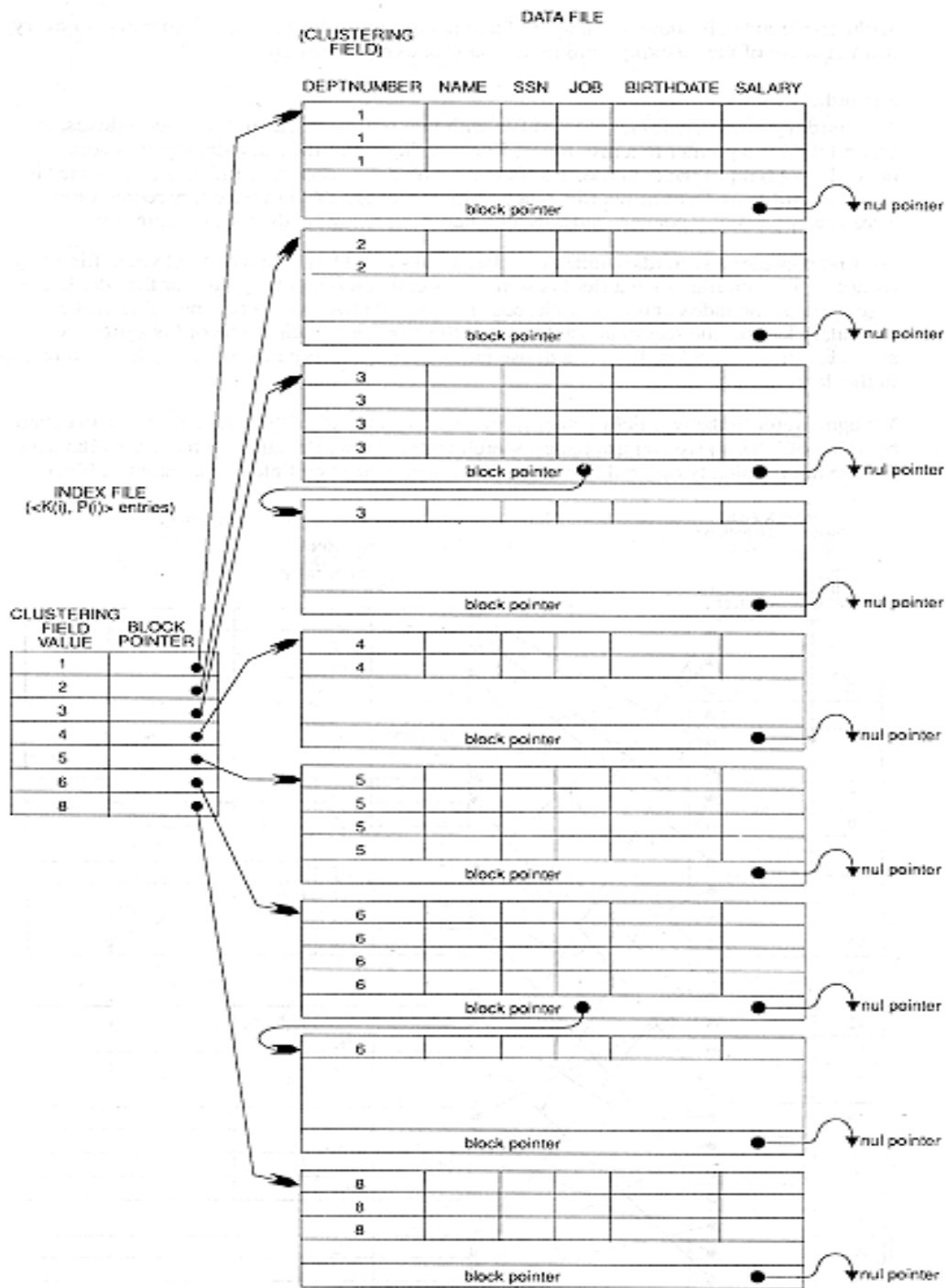
Figure 8 : Clustering index with separate blocks for each group of records with the same value for the clustering field

Figure 7 shows an exam le of a data file with a clustering index. Note the record and record deletion still cause considerable problems because the data records are physically ordered. To alleviate the problem of insertion, it is common to reserve a whole block for each value of the clustering field; all records with that value are placed in the block. If more than one block is needed to store the records for a particular value, additional blocks are allocated and linked together. This makes insertion and deletion relatively straightforward.

✓ SECONDARY INDEXES

A secondary index also is an ordered file with two fields, and, as in the other indexes, the second field is a pointer to a disk block. The first field is of the same data type as some non-ordering field of the data file. The field on which the secondary index is constructed is called an indexing field of the file, whether its values are distinct for every record or not. There can be many secondary indexes, and hence indexing fields, for the same file.

We first consider a secondary index on a key field - a field having a distinct value for every record in the data file. Such a field is sometimes called a secondary key for the file. In this case there is one index entry for each record in the data file, which has the value of the secondary key for the record and a pointer to the block in which the record is stored. A secondary index on a key field is a dense index because it contains one entry for each record in the data file.

We again refer to the two field values of index entry i as $K(i)$, $P(i)$. The entries are ordered by value of $K(i)$, so we can use binary search on the index. Because the records of the data file are not physically ordered by values of the secondary key field, we cannot use block
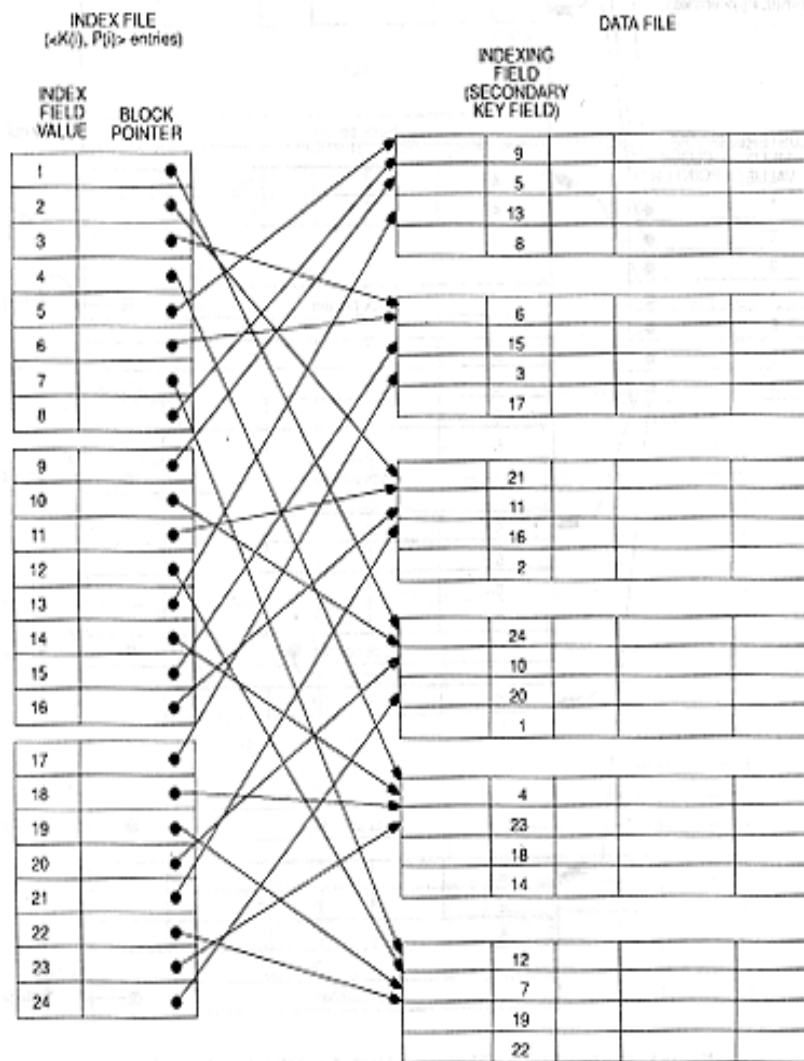
Figure 9 : A dense secondary Index on a non ordering key field of a file

anchors. That is why an index entry is created for each record in the data file rather than for each block as in the case of a primary index. Figure 9 illustrates a secondary index on a key attribute of a data file. Notice that in figure 9 the pointers P(i) in the index entries are block pointers, not record pointers. Once the appropriate block is transferred to main memory, a search for the desired record within the block can be carried out.

A secondary index will usually need substantially more storage space than a primary index because of its larger number of entries. However, the improvement in search time for an arbitrary record is much greater for a secondary index than it is for a primary index, because we would have to do a linear search on the data file if the secondary index did not exist.

30

✓ STRUCTURE OF INDEX SEQUENTIAL FILES

An index-sequential file consists of the data plus one or more levels of indexes. When inserting a record, we have to maintain the sequence of records and this may necessitate shifting subsequent records. For a large file this is a costly and inefficient process. Instead, the records that overflow their logical area are shifted into a designated overflow area and a pointer is provided in the logical area or associated index entry points to the overflow location. This is illustrated below (figure 10). Record 165 is inserted in the original logical block causing a record to be moved to an overflow block.
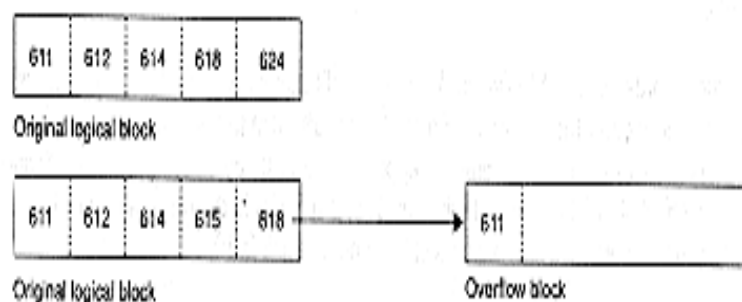
Figure 10 : Overflow of record

Multiple records belonging to the same logical area may be chained to maintained logical sequencing. When records are forced into the overflow areas as a result of insertion, the insertion process is simplified, but the search time is increased. Deletion of records from index-sequential flies creates logical gaps; the records are not physically removed but only flagged as having been deleted. If there were a number of deletions, we may have a great amount of unused space.

An index-sequential file is therefore made up of the following components:

1. A primary data storage area. In certain systems this area may have unused spaces embedded within it to permit addition of records. It may also include records that have been marked as having been deleted.

2. Overflow area(s). This permits the addition of records to the files. A number of schemes exist for the incorporation of records in these areas into the expected logical sequence.

3. A hierarchy of indices. In a random inquiry or update, the physical location of the desired record is obtained by accessing these indices.

The primary data area contains the records written by the users' programs. The records are written in data blocks in ascending key sequence. These data blocks are in turn stored in ascending sequence in

the primary data area. The highest key of the logical records contained in them sequences the data blocks.

✓ DIRECT FILE ORGANISATION ✓

In the index-sequential file organization considered in the previous sections, the mapping from the search-key value to the storage location is via index entries. In direct file

Key value ──────────▶ | Hash function | ──────────▶ Address

Figure 11 : Mapping from a key value to an address value

organization, the key value is mapped directly to the storage location. The usual method of direct mapping is by performing some arithmetic manipulation of the key value. This process is called hashing. Let us consider a hash function h that maps the key value k to the value h(k). The value h(k) is used as an address and for our application we require that this value be in some range. If our address area for the records lies between S1 and S2, the requirement for the hash function h(k) is that for all values of k it should generate values between S1 and S2.

It is obvious that a hash function that maps many different key values to a single address or one that does not map the key values uniformly is a bad hash function. A collision is said to occur when two distinct key values are mapped to the same storage location. Collision is handled in a number of ways. The colliding records may be assigned to the next available space, or they may be assigned to an overflow area. We can immediately see that with hashing schemes there are no indexes to traverse. With well-designed hashing functions where collisions are few, this is a great advantage.

Another problem that we have to resolve is to decide what address is represented by h(k). Let the addresses generated by the hash function the addresses of buckets in which the y, address pair values of records are stored. Figure shows the buckets containing the y, address pairs that allow a reorganization of the actual data file and actual record address without affecting the hash functions. A limited number of collisions could be handled automatically by the use of a bucket of sufficient capacity. Obviously the space required for the buckets will be, in general, much smaller than the actual data file. Consequently, its reorganization will not be that expensive. Once the bucket address is generated from the key by the hash function, asearch in the bucket is also required to locate the address of the required record. However, since the bucket size is small, this overhead is small.

The use of the bucket reduces the problem associated with collisions. In spite of this, a bucket may become full and the resulting overflow could be handled by providing overflow buckets and using a

pointer from the normal bucket to an entry in the overflow bucket. All such overflow entries are linked. Multiple overflows from the same bucket results in a long list and slows down the retrieval of these records. In an alternate scheme, the address generated by the hash function is a bucket address and the bucket is used to store the records directly instead of using a pointer to the block containing the record.
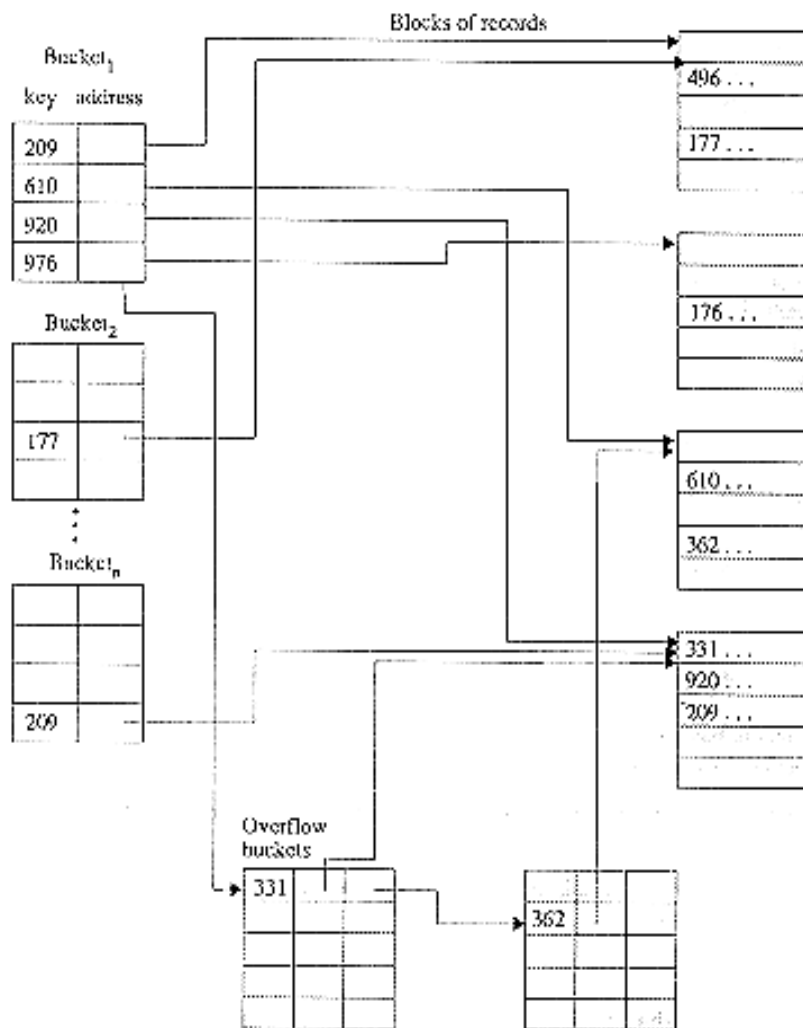
Figure 12 : Bucket and block organization for hashing

Let s represent the value:

s = upper bucket address value - lower bucket address value + 1

Here, s gives the number of buckets. Assume that we have some mechanism to convert key values to numeric ones. Then a simple hashing function is:

h(k) = k mod S

Where k is the numeric representation of the key and h(k) produces a bucket address. A moment's thought tells us that this method would perform well in some cases and not in others.

It has been shown, however, that the choice of a prime number for s is usually satisfactory. A combination of multiplicative and divisive methods can be used to advantage in many practical situations.

There are innumerable ways of converting a key to a numeric value. Most keys are numeric; others may be either alphabetic or alphanumeric. In the latter two cases, we can use the bit representation of the alphabet to generate the numeric equivalent key. A number of simple hashing methods are given below. Many hashing functions can be devised from these and other ways.

1. Use the low order part of the key. For keys that are consecutive integers with few gaps, this method can be used to map the keys to the available range.

2. End folding. For long keys, we identify start, middle, and end regions, such that the sum of the lengths of the start and end regions equals the length of the middle region. The start and end digits are concatenated and the concatenated string of drifts is added to the middle region digits. This new number, mod s where s is the upper limit of the hash function, gives the bucket address:

123456 123456789012 654321

For the above key (converted to integer value if required) the end folding gives the two values to be added as: 123456654321 and 123456789012.

3. Square all or part of the key and take a part from the result. The whole or some defined part of the key is squared and a number of digits are selected from the square as being part of the hash result. A variation is the multiplicative scheme where one part of the key is multiplied by the remaining part and a number of digits are selected from the result.

4. Division. As stated in the beginning of this section, a number, usually a prime, can divide the key and the remainder is taken as the bucket address. A simple check with, for instance, a divisor of 100 tells us that the last two digits of any key will remain unchanged. In applications where keys may be in some multiples, this would produce, a poor result. Therefore, division by a prime number is recommended. For many applications, division by odd numbers that have no divisors less than about 19 gives satisfactory results.

We can conclude from the above discussion that a number of possible methods for generating a hash function exist. In general it has been found that hash functions using division or multiplication performs quite well under most conditions.

✓ HASH FILES ORGANIZATION

Hashing (hash addressing) is a technique for providing fast direct access to a specific record on the basis of a given value of some field. If two or more key values hash to the same disk address, we have a collision.
The hash function should distribute the domain of the key possibly evenly among the address space of the file to minimize the chance of collision. The collisions may cause a page to overflow.

✓ HASH FUNCTIONS

A good hash function gives an average-case lookup that is a small constant, independent of the number of search keys.

- We hope records are distributed uniformly among the buckets.
- The worst hash function maps all keys to the same bucket.
- The best hash function maps all keys to distinct addresses.
- Ideally, distribution of keys to addresses is uniform and random

To summarize the advantages and disadvantages of this approach:

Advantages of hashing:

1. Exact key matches are extremely quick.

2. Hashing is very good for long keys, or those with multiple columns, provided the complete key value is provided for the query.

3. This organization usually allows for the allocation of disk space so a good deal of disk management is possible.

4. No disk space is used by this indexing method.

Disadvantages of hashing:
1. It becomes difficult to predict overflow because the workings of the hashing algorithm will not be visible to the DBA.

2. No sorting of data occurs either physically or logically so sequential access is poor.

3. This organization usually takes a lot of disk space to ensure that no overflow occurs there is a plus side to this though. no space is wasted on index structures because they simply don't exist.

✓ DIRECT ORGANIZATION

Direct file organization provides the fastest direct access to records. When using direct access methods, records do not have to be arranged in any particular sequence on storage media. Characteristics of the direct access method include:

1. Computers must keep track of the storage location of each record using a variety of direct organization methods so that data can be retrieved when needed.

2. New transactions' data do not have to be sorted.

3. Processing that requires immediate responses or updating is easily performed.

✓ DATA ACCESS

Database access and manipulation is performed using the data manipulation statements. These statements, which are specifically designed to interact with an Eloquence database, are invoked through Eloquence language programs. These statements are structured so that each one suggests its function (for example, DBGET gets data from a data set). All data access is carried out at the data entry level (this is known as the "full record mode"). Data entries may be accessed in one of five modes: *serial, directed, chained, indexed* or *calculated*.

✓ SERIAL ACCESS

When accessing a data set in serial mode, Eloquence DBMS starts at the most recently accessed record (data entry), called the *current record* and sequentially examines records until the next, non-empty record is located. This record is then transferred to the data buffer and becomes the new current record. Serial access is often used to examine or list all entries in a data set.

The following example shows entries in the PRODUCT master data set. The record numbers are shown to the left of each entry. The arrows to the left of the record number show how entries will be retrieved in serial mode. If the current record is 4, for example, the next record accessed in serial mode will be record number 5.

RECORD NUMBER	SEARCH ITEM	OTHER DATA
1	100	Standard Bicycle
2	50	Tricycle
3	1000	10-Speed Bicycle
4	500	5-Speed Bicycle
5	300	3-Speed Bicycle

Figure 13 A Serial Access of the PRODUCT Master Data Set

✓ DIRECTED ACCESS

A second method of accessing a data entry is directed access. With this method, Eloquence DBMS returns the record specified by a record number supplied by a program. If the specified record is non-empty the record is transferred to the data buffer. If the record is empty a status error is returned. In either case, the current record is set to the record specified. Directed access is used to read entries following a SORT or FIND operation.

The following example shows the retrieval of an entry using directed access. The record number 5, supplied by an application program, instructs Eloquence DBMS to retrieve record 5. Eloquence DBMS then copies the record into the data buffer and resets the current record to 5.

RECORD NUMBER	SEARCH ITEM	OTHER DATA
1	100	Standard Bicycle
2	50	Tricycle
3	1000	10-Speed Bicycle
4	500	5-Speed Bicycle
5	300	3-Speed Bicycle

Figure 14 Directed Access of the PRODUCT Master Data Set

✓ CHAINED ACCESS

Chained access is used to retrieve detail data entries with common search item values. Eloquence DBMS supports chained access in a forward direction. Entries along a data chain may be accessed in a reverse direction, however, by using directed access and the status information returned by Eloquence DBMS. Chained access of detail data sets is often used for retrieving information about related events.

The following example shows the retrieval of detail entries using chained access. The corresponding chain pointer information, maintained by Eloquence DBMS, is shown along with the record number for the data set. Eloquence DBMS uses this pointer information to retrieve the next entry along the chain. The arrows to the left of the record numbers show how entries will be retrieved in chained mode. If the current record is 5, for examplethe next record accessed in chained mode will be 7.

CUSTOMER (detail) Data Set

RECORD NUMBER	FORWARD CHAIN POINTER	BACKWARD CHAIN POINTER	SEARCH ITEM (PRODUCT-NO)	OTHER DATA (NAME)
1	0	0	100	Jimmy Dailing
2	5	0	50	Malcomb Gissing
3	0	0	500	Barton Decker
4	6	0	300	Sean Houseman
5	7	2	50	Sam Johnson
6	0	4	300	Bart Bekker
7	0	5	50	Thomas Smith

Figure 5 Chained Access of the CUSTOMER Detail Data Set

INTRODUCTION TO DATA BASE MANAGEMENT

As the name suggests, the database management system consists of two parts. They are:
1. Database and
2. Management System

✓ WHAT IS A DATABASE?

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

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

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

Roll	Name	Age
1	ABC	19

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

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

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

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

T1

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

T2

Roll	Address
1	KOL
2	DEL
3	MUM

T3

Roll	Year
1	I
2	II
3	I

T4

Year	Hostel
I	H1
II	H2

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

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

✓ WHAT IS MANAGEMENT SYSTEM?

A management system is a set of rules and procedures which help us to create organize and manipulate the database. It also helps us to add, modify delete data items in the database. The management system can be either manual or computerized.

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

✓ DBMS

A **database-management system** (DBMS) is a collection of interrelated data and a set of programs to access those data. This is a collection of related data with an implicit meaning and hence is a

40

database. The collection of data, usually referred to as the **database**, contains information relevant to an enterprise. The primary goal of a DBMS is to provide a way to store and retrieve database information that is both *convenient* and *efficient*. By **data,** we mean known facts that can be recorded and that have implicit meaning. For example, consider the names, telephone numbers, and addresses of the people you know. You may have recorded this data in an indexed address book, or you may have stored it on a diskette, using a personal computer and software such as DBASE IV or V, Microsoft ACCESS, or EXCEL. A **datum** – a unit of data – is a symbol or a set of symbols which is used to represent something. This relationship between symbols and what they represent is the essence of what we mean by **information**. Hence, information is interpreted data – data supplied with semantics. **Knowledge** refers to the practical use of information. While information can be transported, stored or shared without many difficulties the same can not be said about knowledge. Knowledge necessarily involves a personal experience. Referring back to the scientific experiment, a third person reading the results will have information about it, while the person who conducted the experiment personally will have knowledge about it.

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

✓ DATA PROCESSING VS. DATA MANAGEMENT SYSTEMS

Although Data Processing and Data Management Systems both refer to functions that take raw data and transform it into usable information, the usage of the terms is very different. **Data Processing** is the term generally used to describe what was done by large mainframe computers from the late 1940's until the early 1980's (and which continues to be done in most large organizations to a greater or lesser extent even today): large volumes of raw transaction data fed into programs that update a master file, with fixed-format reports written to paper.

The term **Data Management Systems** refers to an expansion of this concept, where the raw data, previously copied manually from paper to punched cards, and later into data-entry terminals, is now fed into the system from a variety of sources, including ATMs, EFT, and direct customer entry through the Internet. The master file concept has been largely displaced by database management systems, and static reporting replaced or augmented by ad-hoc reporting and direct inquiry, including downloading of data by customers. The ubiquity of the Internet and the Personal Computer have been the driving force in the transformation of Data Processing to the more global concept of Data Management Systems.

✓ CHARACTERISTICS OF DATABASE
- **Concurrent Use**

- A database system allows several users to access the database concurrently. Answering different questions from different users with the same (base) data is a central aspect of an information system. Such concurrent use of data increases the economy of a system.

An example for concurrent use is the travel database of a bigger travel agency. The employees of different branches can access the database concurrently and book journeys for their clients. Each travel agent sees on his interface if there are still seats available for a specific journey or if it is already fully booked.

- **Structured and Described Data**

A fundamental feature of the database approach is that the database systems does not only contain the data but also the complete definition and description of these data. These descriptions are basically details about the extent, the structure, the type and the format of all data and, additionally, the relationship between the data. This kind of stored data is called metadata ("data about data").

- **Separation of Data and Applications**

As described in the feature structured data the structure of a database is described through *metadata* which is also stored in the database. An application software does not need any knowledge about the physical data storage like encoding, format, storage place, etc. It only communicates with the management system f a database (DBMS) via a standardised interface with the help of a standardised language like SQL. The access to the data and the metadata is entirely done by the DBMS. In this way all the applications can be totally seperated from the data. Therefore database internal reorganisations or improvement of efficiency do not have any influence on the application software.

- **Data Integrity**

Data integrity is a byword for the quality and the reliability of the data of a database system. In a broader sense data integrity includes also the protection of the database from unauthorised access (confidentiality) and unauthorised changes. Data reflect facts of the real world. database.

- **Transactions**

A transaction is a bundle of actions which are done within a database to bring it from one consistent state to a new consistent state. In between the data are inevitable inconsistent. A transaction is atomic what means that it cannot be divided up any further. Within a transaction all or none of the actions need to be carried out. Doing only a part of the actions would lead to an inconsistent database state. One example of a transaction is the transfer of an amount of money from one bank account to another. The debit of the money from one account and the credit of it to another account makes together a consistent transaction. This transaction is also atomic. The debit or credit alone would both lead to an inconsistent state. After finishing the transaction (debit and credit) the changes to both accounts become persistent and the one who gave the money has now less money on his account while the receiver has now a higher balance.

- **Data Persistence**

Data persistence means that in a DBMS all data is maintained as long as it is not deleted explicitly. The life span of data needs to be determined directly or indirectly be the user and must not be dependent on system features. Additionally data once stored in a database must not be lost. Changes

of a database which are done by a transaction are persistent. When a transaction is finished even a system crash cannot put the data in danger.

✓ ADVANTAGES AND DISADVANTAGES OF A DBMS

Using a DBMS to manage data has many advantages:

- **Reduction of Redundancy:** This is perhaps the most significant advantage of using DBMS. Redundancy is the problem of storing the same data item in more one place. Redundancy creates several problems like requiring extra storage space, entering same data more than once during data insertion, and deleting data from more than one place during deletion. Anomalies may occur in the database if insertion, deletion etc are not done properly.

- **Sharing of Data:** In a paper-based record keeping, data cannot be shared among many users. But in computerized DBMS, many users can share the same database if they are connected via a network.

- **Data Integrity:** We can maintain data integrity by specifying integrity constrains, which are rules and restrictions about what kind of data may be entered or manipulated within the database. This increases the reliability of the database as it can be guaranteed that no wrong data can exist within the database at any point of time.

- **Data independence:** Application programs should be as independent as possible from details of data representation and storage. The DBMS can provide an abstract view of the data to insulate application code from such details.

- **Efficient data access:** A DBMS utilizes a variety of sophisticated techniques to store and retrieve data efficiently. This feature is especially important if the data is stored on external storage devices.

- **Data integrity and security:** If data is always accessed through the DBMS, the DBMS can enforce integrity constraints on the data. For example, before inserting salary information for an employee, the DBMS can check that the department budget is not exceeded. Also, the DBMS can enforce *access controls* that govern what data is visible to different classes of users.

- **Data administration:** When several users share the data, centralizing the administration of data can offer significant improvements. Experienced professionals who understand the nature of the data being managed, and how different groups of users use it, can be responsible for organizing the data representation to minimize redundancy and fine-tuning the storage of the data to make retrieval efficient.

Concurrent access and crash recovery: A DBMS schedules concurrent accesses to the data in such a manner that users can think of the data as being accessed by only one user at a time. Further, the DBMS protects users from the effects of system failures.

- **Reduced application development time:** Clearly, the DBMS supports many important functions that are common to many applications accessing data stored in the DBMS. This, in conjunction with the high-level interface to the data, facilitates quick development of

applications. Such applications are also likely to be more robust than applications developed from scratch because many important

tasks are handled by the DBMS instead of being implemented by the application.

✓ DISADVANTAGES OF A DBMS

- **Danger of a Overkill**: For small and simple applications for single users a database system is often not advisable.
- **Complexity**: A database system creates additional complexity and requirements. The supply and operation of a database management system with several users and databases is quite costly and demanding.
- **Qualified Personnel**: The professional operation of a database system requires appropriately trained staff. Without a qualified database administrator nothing will work for long.
- **Costs**: Through the use of a database system new costs are generated for the system itselfs but also for additional hardware and the more complex handling of the system.
- **Lower Efficiency**: A database system is a multi-use software which is often less efficient than specialised software which is produced and optimised exactly for one problem.
- **Instances and Schemas**

Databases change over time as information is inserted and deleted. The collection of information stored in the database at a particular moment is called an **instance** of the database. The overall design of the database is called the database **schema**. Schemas are changed infrequently, if at all.

The concept of database schemas and instances can be understood by analogy to a program written in a programming language. A database schema corresponds to the variable declarations (along with associated type definitions) in a program. Each variable has a particular value at a given instant. The values of the variables in a program at a point in time correspond to an *instance* of a database schema.

Database systems have several schemas, partitioned according to the levels of abstraction.

The **physical schema** describes the database design at the physical level, while the **logical schema** describes the database design at the logical level.Adatabase may also have several schemas at the view level, sometimes called **subschemas**, that describe different views of the database.

Of these, the logical schema is by far the most important, in terms of its effect on application programs, since programmers construct applications by using the logical schema. The physical schema is hidden beneath the logical schema, and can usually be changed easily without affecting application programs. Application programs are said to exhibit **physical data independence** if they do not depend on the physical schema, and thus need not be rewritten if the physical schema changes.

✓ DATABASE LANGUAGES

A database system provides a **data definition language** to specify the database schema and a **data manipulation language** to express database queries and updates. In practice, the data definition and

data manipulation languages are not two separate languages; instead they simply form parts of a single database language, such as the widely used SQL language.

✓ DATA-DEFINITION LANGUAGE

We specify a database schema by a set of definitions expressed by a special language called a **data-definition language** (**DDL**).

For instance, the following statement in the SQL language defines the *account* table:

***create table** account (account-number **char(10), balance integer)*

Execution of the above DDL statement creates the *account* table. In addition, it updates a special set of tables called the **data dictionary** or **data directory**.

A data dictionary contains **metadata**—that is, data about data. The schema of a table is an example of metadata. A database system consults the data dictionary before reading or modifying actual data.

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

The data values stored in the database must satisfy certain **consistency constraints**. For example, suppose the balance on an account should not fall below $100. The DDL provides facilities to specify such constraints. The database systems check these constraints every time the database is updated.

✓ DATA-MANIPULATION LANGUAGE

Data manipulation is

- The retrieval of information stored in the database
- The insertion of new information into the database
- The deletion of information from the database
- The modification of information stored in the database

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

Procedural DMLs require a user to specify *what* data are needed and *how* to get those data.

Declarative DMLs (also referred to as **nonprocedural** DMLs) require a user to specify*what* data are needed *without* specifying how to get those data.

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

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

This query in the SQL language finds the name of the customer whose customer-id is 192-83-7465:

45

> **select** *customer.customer-name*
> **from** *customer*

where *customer.customer-id* = 192-83-7465

The query specifies that those rows *from* the table *customer where* the *customer-id* is 192-83-7465 must be retrieved, and the *customer-name* attribute of these rows must be displayed.

Queries may involve information from more than one table. For instance, the following query finds the balance of all accounts owned by the customer with customerid 192-83-7465.

> **select** *account.balance*
> **from** *depositor*, *account*
> **where** *depositor.customer-id* = 192-83-7465 **and**
> *depositor.account-number= account.account-number*

There are a number of database query languages in use, either commercially or experimentally.

The levels of abstraction apply not only to defining or structuring data, but also to manipulating data. At the physical level, we must define algorithms that allow efficient access to data. At higher levels of abstraction, we emphasize ease of use. The goal is to allow humans to interact efficiently with the system. The query processor component of the database system translates DML queries into sequences of actions at the physical level of the database system.

✓ DATA DICTIONARY

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

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

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

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

ability to communicate. In addition, the data dictionary is the tool that helps the DBA to resolve data conflicts.

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

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

If the data dictionary can be organized to include data external to the DBMS itself, it becomes an especially flexible to for more general corporate resource management. The management of such an extensive data dictionary, thus, makes it possible to manage the use and allocation of all of the organization information regardless whether it has its roots in the database data.

✓ DATABASE USERS AND USER INTERFACES

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

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

As another example, consider a user who wishes to find her account balance over the World Wide Web. Such a user may access a form, where she enters her account number. An application program at the Web server then retrieves the account balance, using the given account number, and passes this information back to the user. The typical user interface for naive users is a forms interface,

where the user can fill in appropriate fields of the form. Naive users may also simply read *reports* generated from the database.

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

include special features to facilitate the generation of forms and the display of data on the screen. Most major commercial database systems include a fourth generation language.

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

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

Another class of tools for analysts is **data mining** tools, which help them find certain kinds of patterns in data.

Specialized users are sophisticated users who write specialized database applications that do not fit into the traditional data-processing framework.

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

✓ DATABASE ADMINISTRATOR

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

- **Schema definition**. The DBA creates the original database schema by executing a set of data definition statements in the DDL.
- **Schema and physical-organization modification**. The DBA carries out changes to the schema and physical organization to reflect the changing needs of the organization, or to alter the physical organization to improve performance.
- **Granting of authorization for data access**. By granting different types of authorization, the database administrator can regulate which parts of the database various users can access. The

authorization information is kept in a special system structure that the database system consults whenever someone attempts to access the data in the system.

- **Routine maintenance**. Examples of the database administrator's routine maintenance activities are:
1. Periodically backing up the database, either onto tapes or onto remote servers, to prevent loss of data in case of disasters such as flooding.
2. Ensuring that enough free disk space is available for normal operations, and upgrading disk space as required.
3. Monitoring jobs running on the database and ensuring that performance is not degraded by very expensive tasks submitted by some users.

✓ DBMS ARCHITECTURE

Three important characteristics of the database approach are (1) insulation of programs and data (program-data and program-operation independence); (2) support of multiple user views; and (3) use of a catalog to store the database description (schema). In this section we specify an architecture for database systems, called the **three-schema architecture**, which was proposed to help achieve and visualize these characteristics.

The goal of the three-schema architecture, illustrated in Figure 1.1, is to separate the user applications and the physical database. In this architecture, schemas can be defined at the following three levels:

1. The **internal level** has an **internal schema,** which describes the physical storage structure of the database. The internal schema uses a physical data model and describes the complete details of data storage and access paths for the database.
2. The **conceptual level** has a **conceptual schema,** which describes the structure of the whole database for a community of users. The conceptual schema hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations, and constraints. A high-level data model or an implementation data model can be used at this level.
3. The **external** or **view level** includes a number of **external schemas** or **user views.** Each external schema describes the part of the database that a particular user group is interested in and hides the rest of the database from that user group. A high-level data model or an implementation data model can be used at this level.
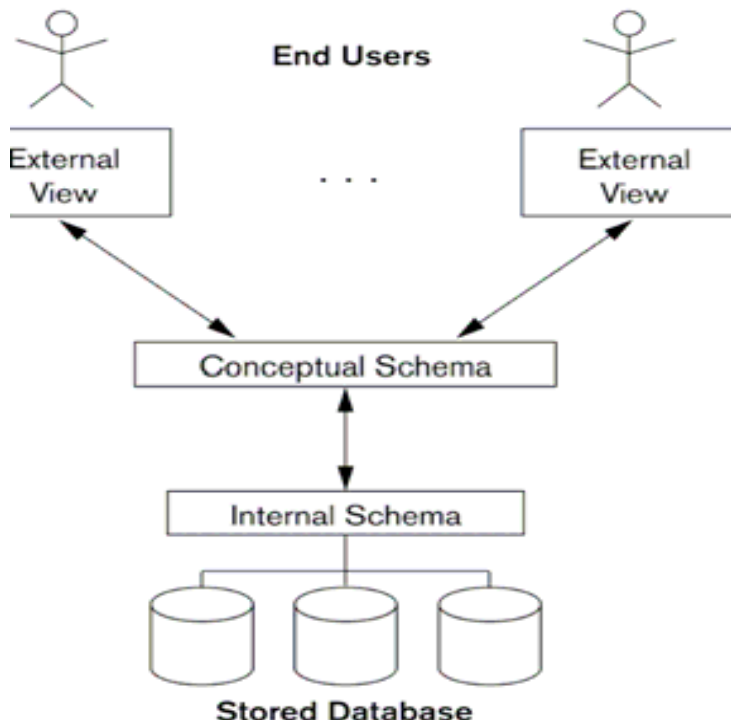
Figure 1.1 The Three Schema Architecture

The three-schema architecture is a convenient tool for the user to visualize the schema levels in a database system. Most DBMSs do not separate the three levels completely, but support the three-schema architecture to some extent. Some DBMSs may include physical-level details in the conceptual schema. In most DBMSs that support user views, external schemas are specified in the same data model that describes the conceptual-level information. Some DBMSs allow different data models to be used at the conceptual and external levels.

Notice that the three schemas are only *descriptions* of data; the only data that *actually* exists is at the physical level. In a DBMS based on the three-schema architecture, each user group refers only to its own external schema. Hence, the DBMS must transform a request specified on an external schema into a request against the conceptual schema, and then into a request on the internal schema for processing over the stored database. If the request is a database retrieval, the data extracted from the stored database must be reformatted to match the user's external view. The processes of transforming requests and results between levels are called **mappings.** These mappings may be time-consuming, so some DBMSs—especially those that are meant to support small databases—do not support external views. Even in such systems, however, a certain amount of mapping is necessary to transform requests between the conceptual and internal levels.

DATA INDEPENDENCE

The three-schema architecture can be used to explain the concept of **data independence,** which can be defined as the capacity to change the schema at one level of a database system without having to change the schema at the next higher level. We can define two types of data independence:

1. **Logical data independence** is the capacity to change the conceptual schema without having to change external schemas or application programs. We may change the conceptual schema to expand the database (by adding a record type or data item), or to reduce the database (by removing a record type or data item). In the latter case, external schemas that refer only to the remaining data should not be affected. Only the view definition and the mappings need be changed in a DBMS that supports logical data independence. Application programs that reference the external schema constructs must work as before, after the conceptual schema undergoes a logical reorganization. Changes to constraints can be applied also to the conceptual schema without affecting the external schemas or application programs.

2. **Physical data independence** is the capacity to change the internal schema without having to change the conceptual (or external) schemas. Changes to the internal schema may be needed because some physical files had to be reorganized—for example, by creating additional access structures—to improve the performance of retrieval or update. If the same data as before remains in the database, we should not have to change the conceptual schema.

Whenever we have a multiple-level DBMS, its catalog must be expanded to include information on how to map requests and data among the various levels. The DBMS uses additional software to accomplish these mappings by referring to the mapping information in the catalog. Data independence is accomplished because, when the schema is changed at some level, the schema at the next higher level remains unchanged; only the *mapping* between the two levels is changed. Hence, application programs referring to the higher-level schema need not be changed.

TYPES OF DATABASE SYSTEM

Several criteria are normally used to classify DBMSs. The *first* is the data model on which the DBMS is based. The main data model used in many current commercial DBMSs is the relational data model. The object data model was implemented in some commercial systems but has not had widespread use. Many legacy (older) applications still run on database systems based on the hierarchical and network data models. The relational DBMSs are evolving continuously, and, in particular, have been incorporating many of the concepts that were developed in object databases. This has led to a new class of DBMSs called object-relational DBMSs. We can hence categorize DBMSs based on the *data model*: **relational, object, object-relational, hierarchical, network, and other**. The *second* criterion used to classify DBMSs is the number of users supported by the system. **Single-user systems** support only one user at a time and are mostly used with personal computers. **Multiuser systems**, which include the majority of DBMSs, support multiple users concurrently. A *third* criterion is the number of sites over which the database is distributed. A DBMS is centralized if the data is stored at a single computer site. A **centralized DBMS** can support multiple users, but the DBMS and the database themselves reside totally at a single computer site. A **distributed DBMS**

(DDBMS) can have the actual database and DBMS software distributed over many sites, connected by a computer network. Homogeneous DDBMSs use the same DBMS software at multiple sites.

Types of Database Management Systems

There are four structural types of database management systems:

- Hierarchical databases.
- Network databases.
- Relational databases.
- Object-oriented databases

Hierarchical Databases (DBMS) :

In the Hierarchical Database Model we have to learn about the databases. It is very fast and simple. In a hierarchical database, records contain information about there groups of parent/child relationships, just like as a tree structure. The structure implies that a record can have also a repeating information. In this structure Data follows a series of records, It is a set of field values attached to it. It collects all records together as a record type. These record types are the equivalent of tables in the relational model, and with the individual records being the equivalent of rows. To create links between these record types, the hierarchical model uses these type Relationships.
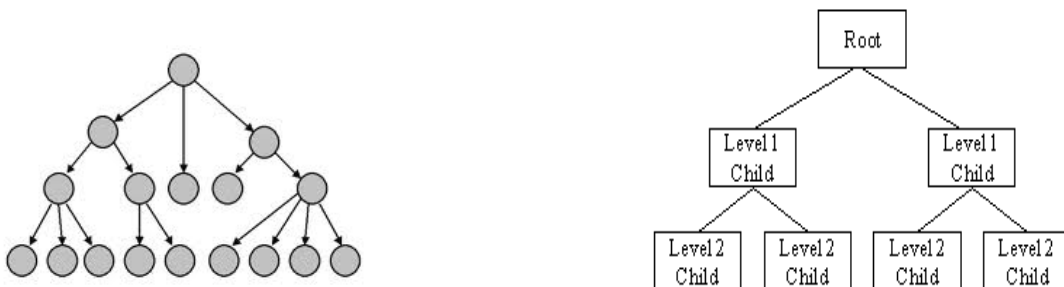
Advantage : Hierarchical database can be accessed and updated rapidly because in this model structure is like as a tree and the relationships between records are defined in advance. This feature is a two-edged.

Disadvantage : This type of database structure is that each child in the tree may have only one parent, and relationships or linkages between children are not permitted, even if they make sense

from a logical standpoint. Hierarchical databases are so in their design. it can adding a new field or record requires that the entire database be redefined.

Network Database: A network databases are mainly used on a large digital computers. It more connections can be made between different types of data, network databases are considered more efficiency It contains limitations must be considered when we have to use this kind of database. It is Similar to the hierarchical databases, network databases .Network databases are similar to hierarchical databases by also having a hierarchical structure. A network database looks more like a cobweb or interconnected network of records.

In network databases, children are called members and parents are called occupier. The difference between each child or member can have more than one parent.
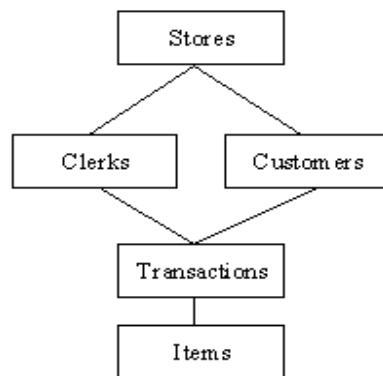
The Approval of the network data model similar with the esteem of the hierarchical data model. Some data were more naturally modeled with more than one parent per child. The network model authorized the modeling of many-to-many relationships in data.

The network model is very similar to the hierarchical model really. Actually the hierarchical model is a subset of the network model. However, instead of using a single-parent tree hierarchy, the network model uses set theory to provide a tree-like hierarchy with the exception that child tables were allowed to have more than one parent. It supports many-to-many relationships.

Relational Databases :

In relational databases, the relationship between data files is relational. Hierarchical and network

databases require the user to pass a hierarchy in order to access needed data. These databases connect to the data in different files by using common data numbers or a key field. Data in relational databases is stored in different access control tables, each having a key field that mainly identifies each row. In the relational databases are more reliable than either the hierarchical or network database structures. In relational databases, tables or files filled up with data are called relations (tuples) designates a row or record, and columns are referred to as attributes or fields.

Relational databases work on each table has a key field that uniquely indicates each row, and that these key fields can be used to connect one table of data to another.
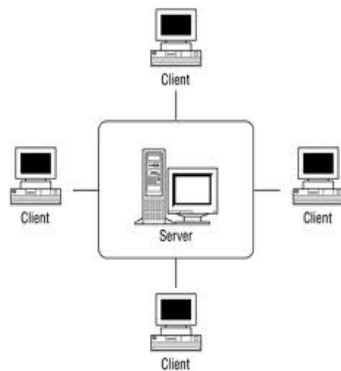
The relational database has two major reasons:

1. Relational databases can be used with little or no training.
2. Database entries can be modified without specify the entire body.

Properties of Relational Tables:

In the relational database we have to follow some properties which are given below.

- It's Values are Atomic
- In Each Row is alone.
- Column Values are of the Same thing.
- Columns is undistinguished.
- Sequence of Rows is Insignificant.
- Each Column has a common Name.

Object-Oriented Model :

In this Model we have to discuss the functionality of the object oriented Programming .It takes more than storage of programming language objects. Object DBMS's increase the semantics of the C++ and Java .It provides full-featured database programming capability, while containing native language compatibility. It adds the database functionality to object programming languages.This approach is the analogical of the application and database development into a constant data model and language environment. Applications require less code, use more natural data modeling, and code bases are easier to maintain. Object developers can write complete database applications with a decent amount of additional effort.

The object-oriented database derivation is the integrity of object-oriented programming language systems and consistent systems. The power of the object-oriented databases comes from the cyclical treatment of both consistent data, as found in databases, and transient data, as found in executing programs.
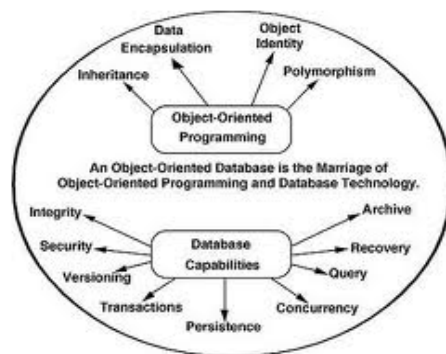
Object-oriented databases use small, recyclable separated of software called objects. The objects themselves are stored in the object-oriented database. Each object contains of two elements:

1. Piece of data (e.g., sound, video, text, or graphics).
2. Instructions, or software programs called methods, for what to do with the data.

Disadvantage of Object-oriented databases

1. Object-oriented databases have these disadvantages.
2. Object-oriented database are more expensive to develop.

55

3. In the Most organizations are unwilling to abandon and convert from those databases.

DATA MODEL

A data model is a collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints.

- **Entity:**An entity is a "thing" or "object" in the real world that is distinguishable from all other objects. For example, each person in an enterprise is an entity.
- **Entity Set:**An entity set is a set of entities of the same type that share the same properties, orattributes. The set of all persons who are customers at a given bank, for example, can be defined as the entity set customer. Similarly, the entity set loan might represent the set of all loans awarded by a particular bank.
  An entity is represented by a set of attributes. Attributes are descriptive properties possessed by each member of an entity set. The designation of an attribute for an entity set expresses that the database stores similar information concerning each entity in the entity set; however, each entity may have its own value for each attribute.
- **Simple and composite attributes:**The attributes havebeen simple; that is, they are not divided into subparts is called as "*simple attributes*". on the other hand, can be divided into subparts is called as "*composite attributes"*.For example, an attribute name could be structured as a composite attribute consisting of first-name, middle-initial, and last-name.
- **Single-valued and multivalued attributes:**For instance, the loan-number attribute for a specific loan entity refers to only one loan number. Such attributes are said to be single valued. There may be instances where an attribute has a set of values for a specific entity. Consider an employee entity set with the attribute phone-number. An employee may have zero, one, or several phone numbers, and different employees may have different numbers of phones.
  This type of attribute is said to be multivalued.
- **Derived attribute:**The value for this type of attribute can be derived from the values of other related attributes or entities. For instance, let us say that the customer entity set has an attribute loans-held, which represents how many loans a customer has from the bank.We can derive the value for this attribute by counting the number of loan entities associated with that customer.
- **Relationship Sets:**A relationship is an association among several entities. A relationship set is a set of relationships of the same type.
- **Mapping Cardinalities:**Mapping cardinalities, or cardinality ratios, express the number of entities to which another entity can be associated via a relationship set. Mapping cardinalities are most useful in describing binary relationship sets, although they can contribute to the description of relationship sets that involve more than two entity sets.

56

- *One to one*. An entity in A is associated with at most one entity in B, and an entity in B is associated with at most one entity in A.
- One to many. An entity in A is associated with any number (zero or more) of entities in B. An entity in B, however, can be associated with at most one entity in A.
- Many to one. An entity in A is associated with at most one entity in B. An entity in B, however, can be associated with any number (zero or more) of entities in A.
- Many to many. An entity in A is associated with any number (zero or more) of entities in B, and an entity in B is associated with any number (zero or more) of entities in A.

  - **Keys:** A key allows us to identify a set of attributes that suffice to distinguish entities from each other. Keys also help uniquely identify relationships, and thus distinguish relationships from each other.
1. **Superkey:**A superkey is a set of one or more attributes that, taken collectively, allow us to identify uniquely an entity in the entity set. For example, the customer-id attribute of the entity set customer is sufficient to distinguish one customer entity from another. Thus, customer-id is a superkey. Similarly, the combination of customer-name and customer-id is a superkey for the entity set customer. The customer-name attribute of customer is not a superkey, because several people might have the same name.
2. **Candidate key:**Minimal superkeys are called candidate keys.If K is a superkey, then so is any superset of K. We are often interested in superkeys for which no proper subset is a superkey.It is possible that several distinct sets of attributes could serve as a candidate key.Suppose that a combination of customer-name and customer-street is sufficient to distinguish among members of the customer entity set. Then, both {customer-id} and {customer-name, customer-street} are candidate keys. Although the attributes customerid and customer-name together can distinguish customer entities, their combination does not form a candidate key, since the attribute customer-id alone is a candidate key.
3. **Primary key:**which denotes the unique identity is called as primary key.primary key to denote a candidate key that is chosen by the database designer as the principal means of identifying entities within an entity set. A key (primary, candidate, and super) is a property of the entity set, rather than of the individual entities. Any two individual entities in the set are prohibited from having the same value on the key attributes at the same time. The designation of a key represents a constraint in the real-world enterprise being modeled.
4. **Weak Entity Sets:**An entity set may not have sufficient attributes to form a primary key. Such an entity set is termed a weak entity set. An entity set that has a primary key is termed a strong entity set. For a weak entity set to be meaningful, it must be associated with another entityset, called the identifying or owner entity set. Every weak entity must be associated with an identifying entity; that is, the weak entity set is said to be existence dependent on the identifying entity set. The identifying entity set is said to own the weak entity set that it identifies. The relationship associating the weak entity set with the identifying entity set is called the identifying relationship. The identifying relationship is many to one from the weak

entity set to the identifying entity set, and the participation of the weak entity set in the relationship is total.

SPECIALIZATION

An entity set may include subgroupings of entities that are distinct in some wayfrom other entities in the set. For instance, a subset of entities within an entity set may have attributes that are not shared by all the entities in the entity set. The E-R model provides a means for representing these distinctive entity groupings. Consider an entity set person, with attributes name, street, and city. A personmay be further classified as one of the following:
• customer
• employee

Each of these person types is described by a set of attributes that includes all the attributes of entity set person plus possibly additional attributes. For example, customer entities may be described further by the attribute customer-id, whereas employee entities may be described further by the attributes employee-id and salary. The process of designating subgroupings within an entity set is called specialization. The specialization of person allows us to distinguish among persons according to whether they are employees or customers.

GENERALIZATION

The design process may also proceed in a bottom-up manner, in which multiple entity sets are synthesized into a higher-level entity set on the basis of common features. The database designer may have first identified a customer entity set with the attributes name, street, city, and customer-id, and an employee entity set with the attributes name, street, city, employee-id, and salary. There are similarities between the customer entity set and the employee entity set in the sense that they have several attributes in common. This commonality can be expressed by generalization, which is a containment relationship that exists between a higher-level entity set and one or more lower-level entity sets. In our example, person is the higher-level entity set and customer and employee are lower-level entity sets.
Higher- and lower-level entity sets also may be designated by the terms superclass and subclass, respectively. The person entity set is the superclass of the customer and employee subclasses.For all practical purposes, generalization is a simple inversion of specialization. We will apply both processes, in combination, in the course of designing the E-R schema for an enterprise. In terms of the E-R diagram itself, we do not distinguish between specialization and generalization. New levels of entity representation will be distinguished (specialization) or synthesized (generalization) as the design schema comes to express fully the database application and the user requirements of the database. Differences in the two approaches may be characterized by their starting point and overall goal.Generalization proceeds from the recognition that a number of entity sets share some common

features (namely, they are described by the same attributes and participatein the same relationship sets).
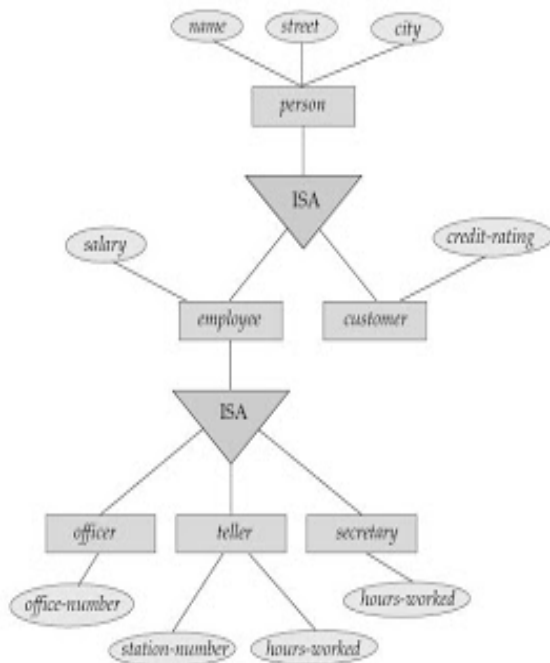
✓ **DATA MODELS**

Underlying the structure of a database is the **data model**: a collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints.

To illustrate the concept of a data model, we outline two data models in this section: the entity-relationship model and the relational model. Both provide a way to describe the design of a database at the logical level.

✓ **RELATIONAL MODEL**

The relational model uses a collection of tables to represent both data and the relationships among those data. Each table has multiple columns, and each column has a unique name.

The data is arranged in a relation which is visually represented in a two dimensional table. The data is inserted into the table in the form of tuples (which are nothing but rows). A tuple is formed by one or more than one attributes, which are used as basic building blocks in the formation of various expressions that are used to derive a meaningful information. There can be any number of tuples in

the table, but all the tuple contain fixed and same attributes with varying values. The relational model is implemented in database where a relation is represented by a table, a tuple is represented by a row, an attribute is represented by a column of the table, attribute name is the name of the column such as 'identifier', 'name', 'city' etc., attribute value contains the value for column in the row. Constraints are applied to the table and form the logical schema. In order to facilitate the selection of a particular row/tuple from the table, the attributes i.e. column names are used, and to expedite the selection of the rows some fields are defined uniquely to use them as indexes, this helps in searching the required data as fast as possible. All the relational algebra operations, such as Select, Intersection, Product, Union, Difference, Project, Join, Division, Merge etc. can also be performed on the Relational Database Model. Operations on the Relational Database Model are facilitated with the help of different conditional expressions, various key attributes, pre-defined constraints etc.

✓ THE ENTITY-RELATIONSHIP MODEL

- The entity-relationship (E-R) data model is based on a perception of a real world that consists of a collection of basic objects, called *entities*, and of *relationships* among these objects. An entity is a "thing" or "object" in the real world that is distinguishable from other objects. For example, each person is an entity, and bank accounts can be considered as entities.

- Entities are described in a database by a set of **attributes**. For example, the attributes *account-number* and *balance* may describe one particular account in a bank, and they form attributes of the *account* entity set. Similarly, attributes *customer-name*, *customer-street* address and *customer-city* may describe a *customer* entity.

- An extra attribute *customer-id* is used to uniquely identify customers (since it may be possible to have two customers with the same name, street address, and city).

- A unique customer identifier must be assigned to each customer. In the United States, many enterprises use the social-security number of a person (a unique number the U.S. government assigns to every person in the United States) as a customer identifier.

- A **relationship** is an association among several entities. For example, a *depositor* relationship associates a customer with each account that she has. The set of all entities of the same type and the set of all relationships of the same type are termed an **entity set** and **relationship set**, respectively.

- The overall logical structure (schema) of a database can be expressed graphically by an *E-R diagram*.

Advantages and Disadvantages of E-R Data Model

Following are advantages of an E-R Model:

• **Straightforward relation representation:** Having designed an E-R diagram for a database application, the relational representation of the database model becomes relatively straightforward.

• **Easy conversion for E-R to other data model:** Conversion from E-R diagram to a network or hierarchical data model can· easily be accomplished.

• Graphical representation for better understanding: An E-R model gives graphical and diagrammatical representation of various entities, its attributes and relationships between entities. This is turn helps in the clear understanding of the data structure and in minimizing redundancy and other problems.

Disadvantages of E-R Data Model

Following are disadvantages of an E-R Model:

• **No industry standard for notation:** There is no industry standard notation for developing an E-R diagram.

• **Popular for high-level design**: The E-R data model is especially popular for high level

Symbols used in the E-R notation:
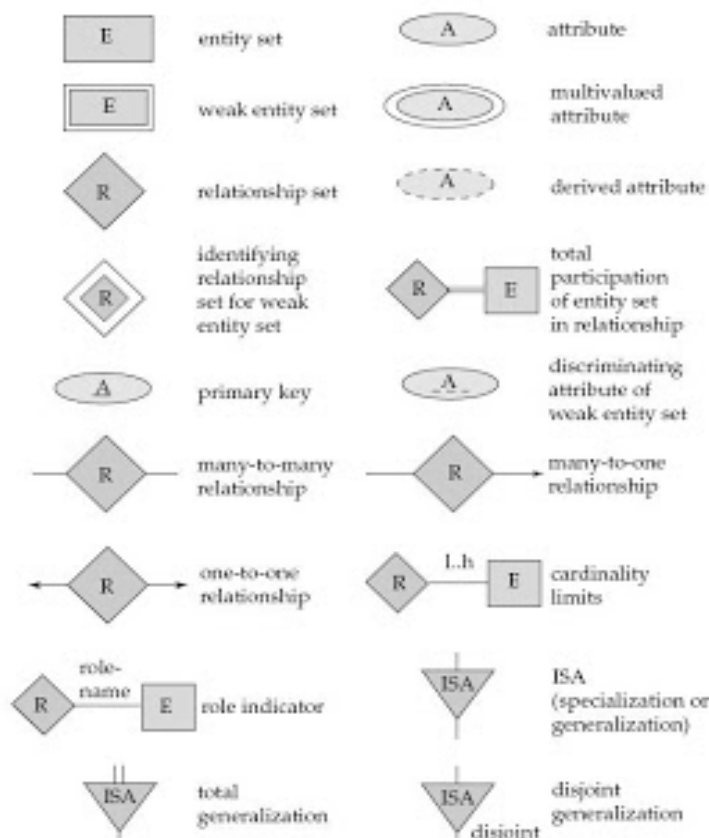
ER MODEL FOR A COLLEGE DB

Assumptions :

- A college contains many departments
- Each department can offer any number of courses
- Many instructors can work in a department
- An instructor can work only in one department
- For each department there is a Head
- An instructor can be head of only one department
- Each instructor can take any number of courses
- A course can be taken by only one instructor
- A student can enroll for any number of courses
- Each course can have any number of students

Steps in ER Modeling:

- Identify the Entities
- Find relationships
- Identify the key attributes for every Entity
- Identify other relevant attributes
- Draw complete E-R diagram with all attributes including Primary Key

Step 1: Identify the Entities:

- DEPARTMENT
- STUDENT
- COURSE
- INSTRUCTOR

Step 2: Find the relationships:

- One course is enrolled by multiple students and one student enrolls for multiple courses, hence the cardinality between course and student is Many to Many.
- The department offers many courses and each course belongs to only one department, hence the cardinality between department and course is One to Many.
- One department has multiple instructors and one instructor belongs to one and only one department , hence the cardinality between department and instructor is one to Many.

- Each department there is a "Head of department" and one instructor is "Head of department ",hence the cardinality is one to one .
- One course is taught by only one instructor, but the instructor teaches many courses, hence the cardinality between course and instructor is many to one.

Step 3: Identify the key attributes

- Deptname is the key attribute for the Entity "Department", as it identifies the Department uniquely.
- Course# (CourseId) is the key attribute for "Course" Entity.
- Student# (Student Number) is the key attribute for "Student" Entity.
- Instructor Name is the key attribute for "Instructor" Entity.

Step 4: Identify other relevant attributes

For the department entity, the relevant attribute is location

- For course entity, course name,duration,prerequisite
- For instructor entity, room#, telephone#
- For student entity, student name, date of birth

ER MODEL FOR BANKING BUSINESS

Assumptions :

- There are multiple banks and each bank has many branches. Each branch has multiple customers
- Customers have various types of accounts
- Some Customers also had taken different types of loans from these bank branches
- One customer can have multiple accounts and Loans

Step 1: Identify the Entities
• BANK
• BRANCH
• LOAN
• ACCOUNT
• CUSTOMER
Step 2: Find the relationships
• One Bank has many branches and each branch belongs to only one bank, hence the cardinality between Bank and Branch is One to Many.

• One Branch offers many loans and each loan is associated with one branch, hence the cardinality between Branch and Loan is One to Many.

• One Branch maintains multiple accounts and each account is associated to one and only one Branch, hence the cardinality between Branch and Account is One to Many

• One Loan can be availed by multiple customers, and each Customer can avail multiple loans, hence the cardinality between Loan and Customer is Many to Many.

• One Customer can hold multiple accounts, and each Account can be held by multiple Customers, hence the cardinality between Customer and Account is Many to Many

Step 3: Identify the key attributes

• BankCode (Bank Code) is the key attribute for the Entity "Bank", as it identifies the bank uniquely.

• Branch# (Branch Number) is the key attribute for "Branch" Entity.

• Customer# (Customer Number) is the key attribute for "Customer" Entity.

• Loan# (Loan Number) is the key attribute for "Loan" Entity.

• Account No (Account Number) is the key attribute for "Account" Entity.

Step 4: Identify other relevant attributes

• For the "Bank" Entity, the relevant attributes other than "BankCode" would be "Name" and "Address".

• For the "Branch" Entity, the relevant attributes other than "Branch#" would be "Name" and "Address".

• For the "Loan" Entity, the relevant attribute other than "Loan#" would be "Loan Type".

• For the "Account" Entity, the relevant attribute other than "Account No" would be "Account Type".

• For the "Customer" Entity, the relevant attributes other than "Customer#" would be "Name", "Telephone#" and "Address".
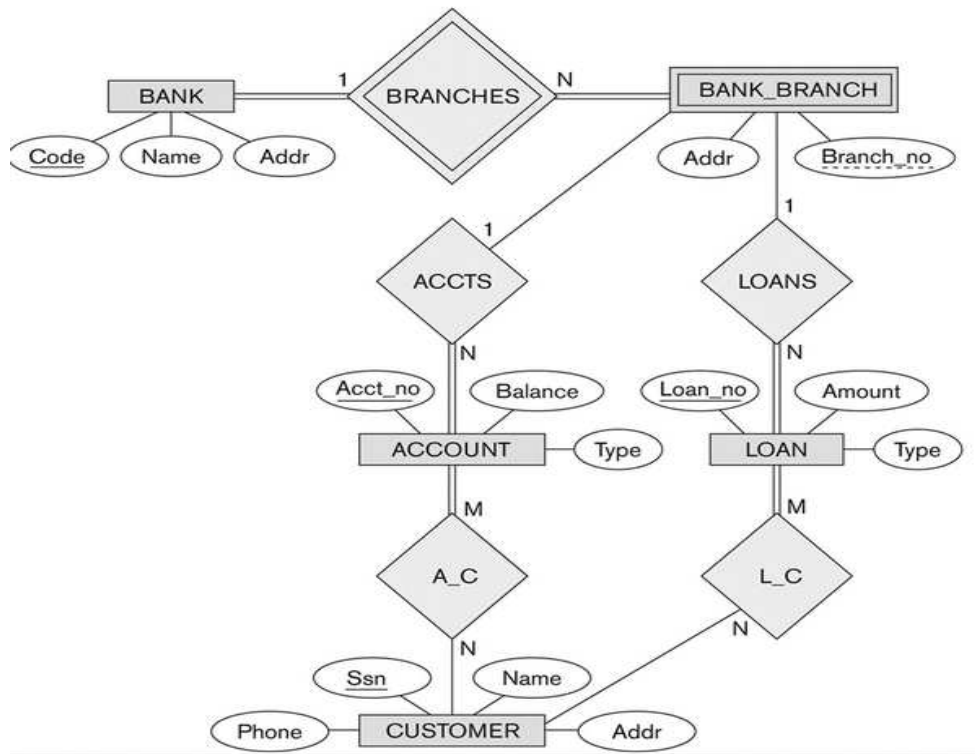
ER DIAGRAM FOR A MANAGEMENT SYSTEM
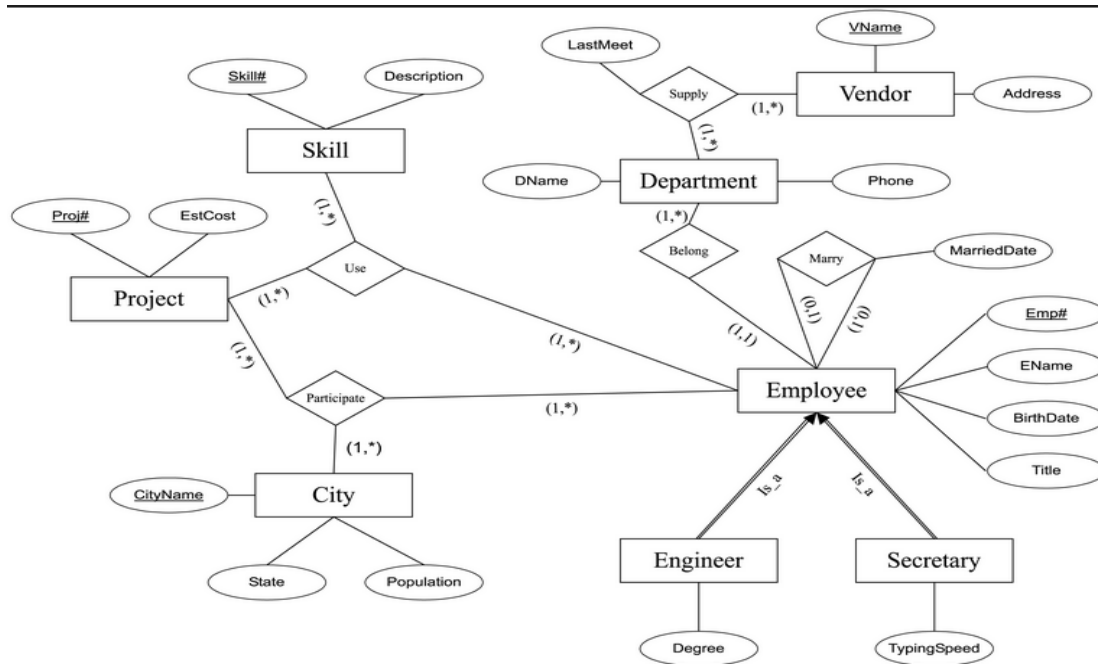
ER DIAGRAM FOR COLLEGE LIBRARY
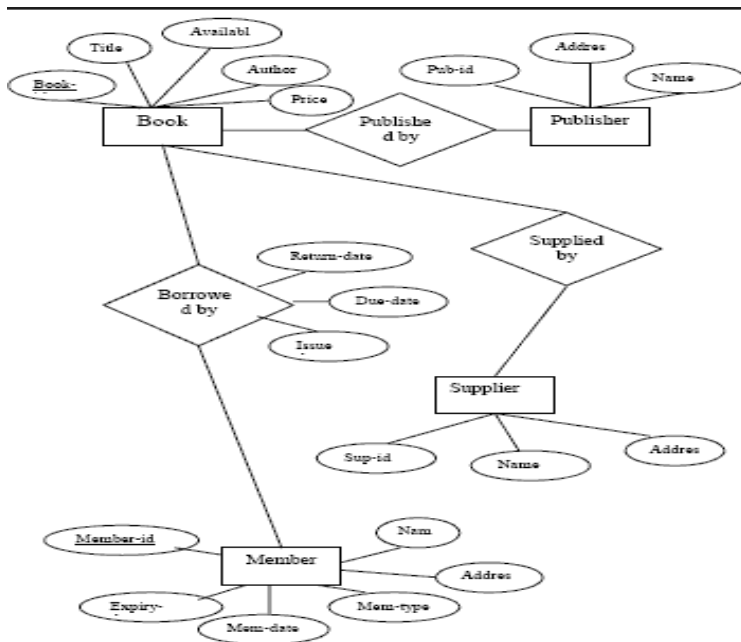
ER DIAGRAM FOR A CAR STORE DATABASE
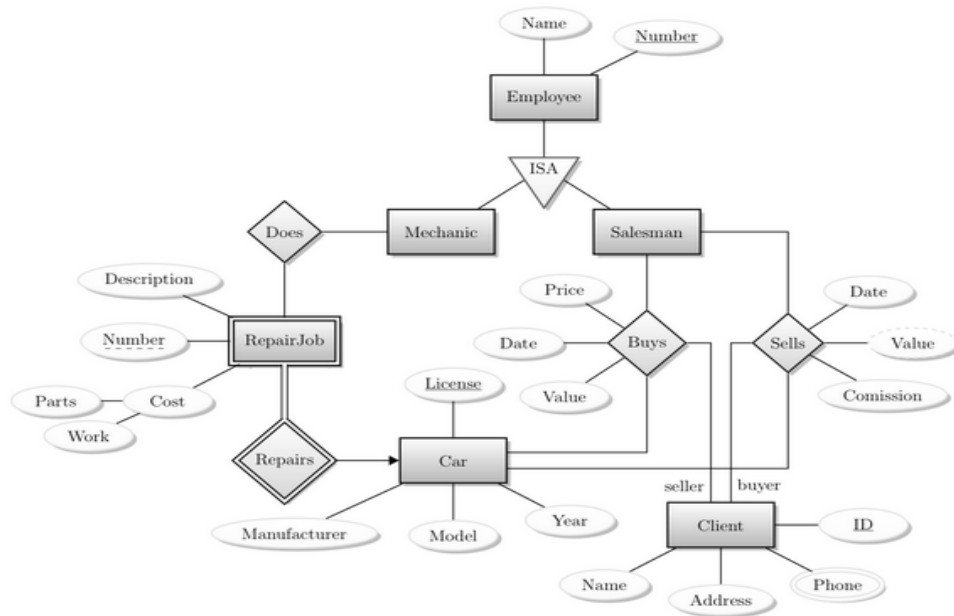
ER DIAGRAM FOR A MOVIE SHOW
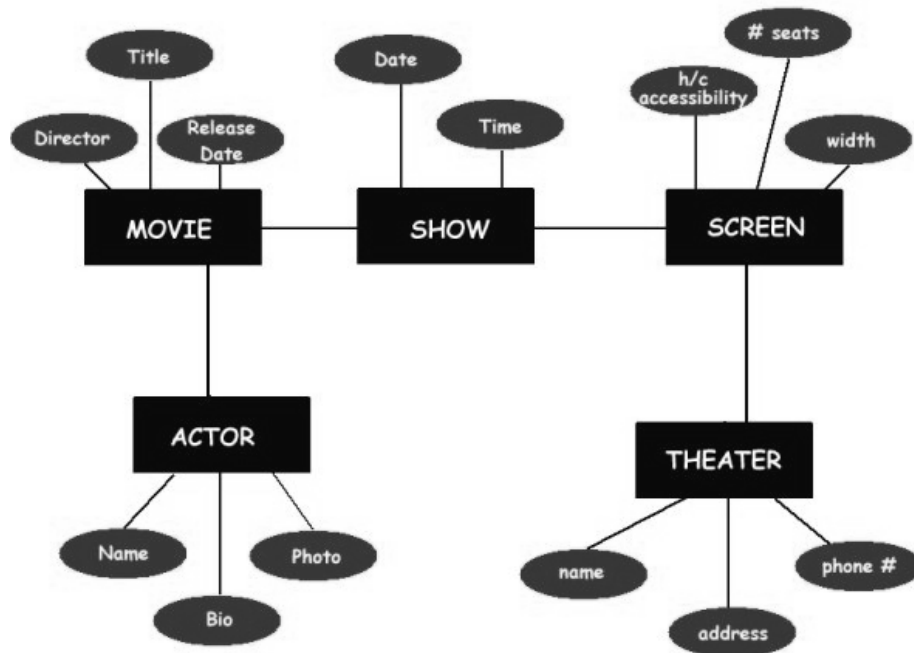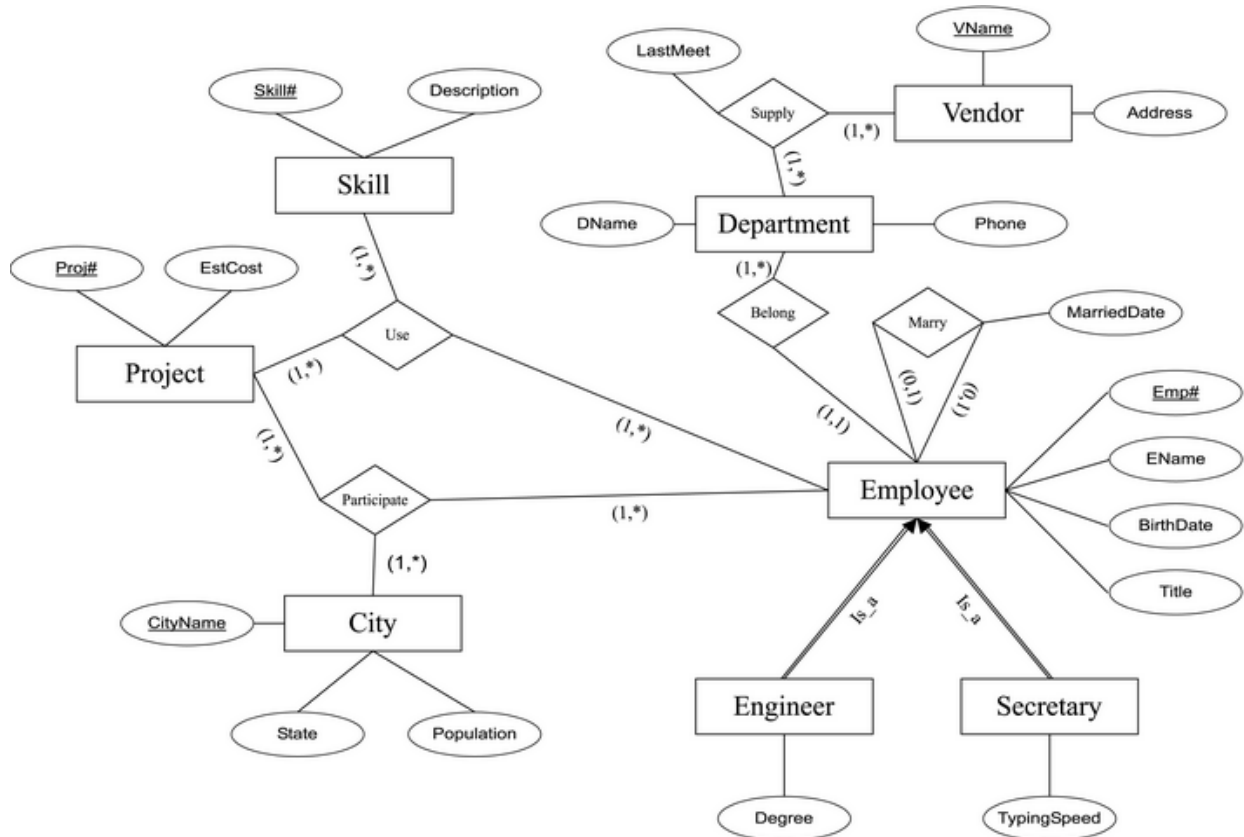
ER DIAGRAM FOR A COMPANY DATABASE
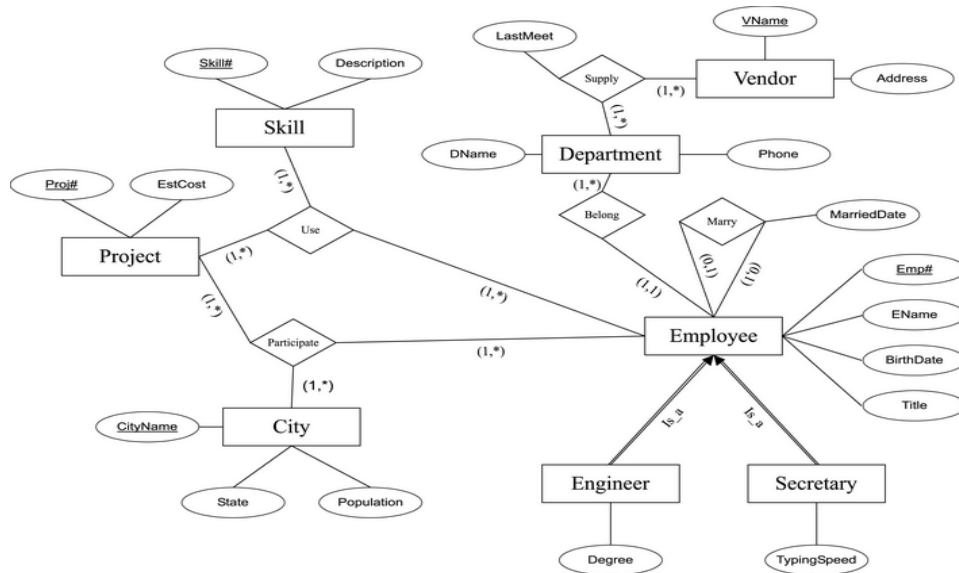
ER DIAGRAM FOR A TEACHING MEYHODOLOGY
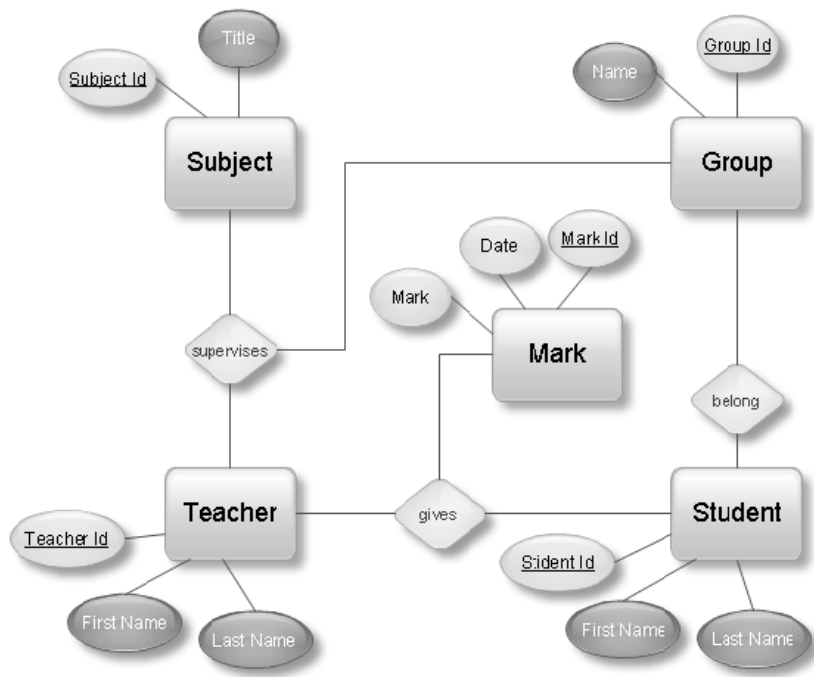
ER DIAGRAM FOR A ORDERING RELATIONSHIP
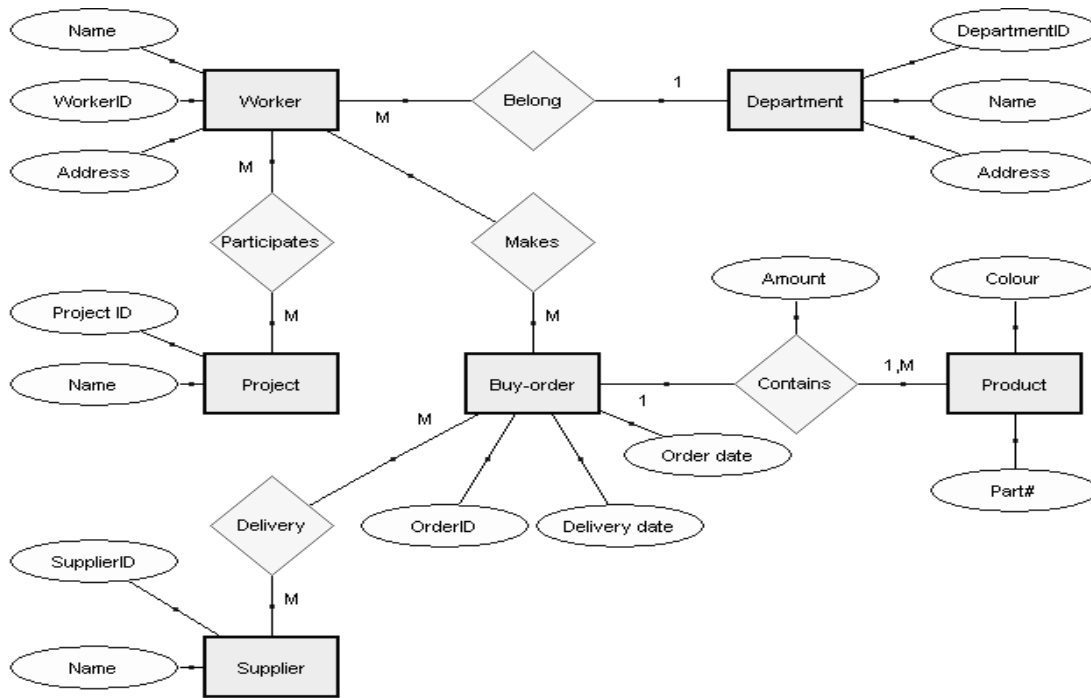
ER DIAGRAM FOR A INSURANCE COMPANY
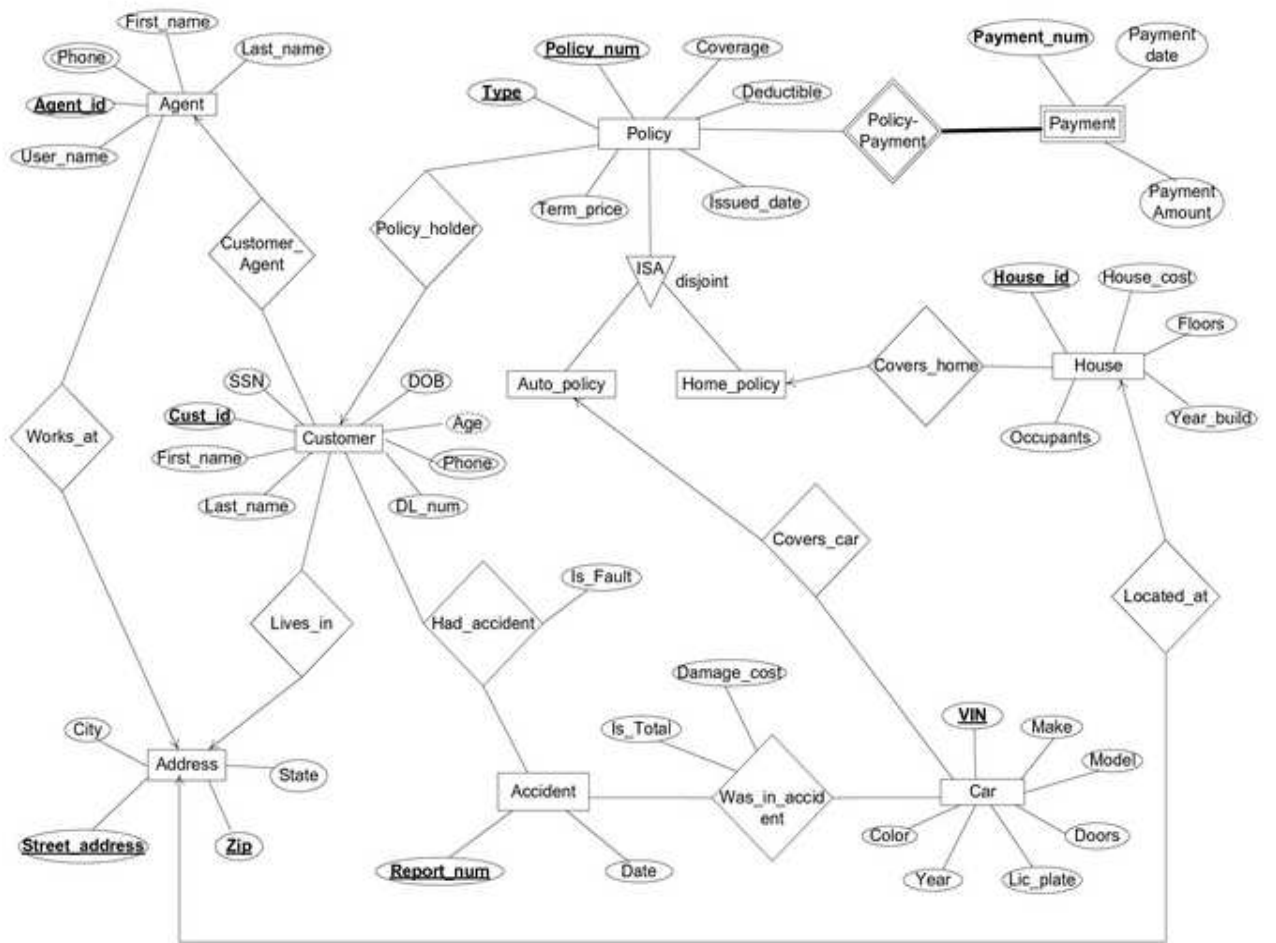
ER DIAGRAM FOR A MUSIC STORE SPECIFICATION
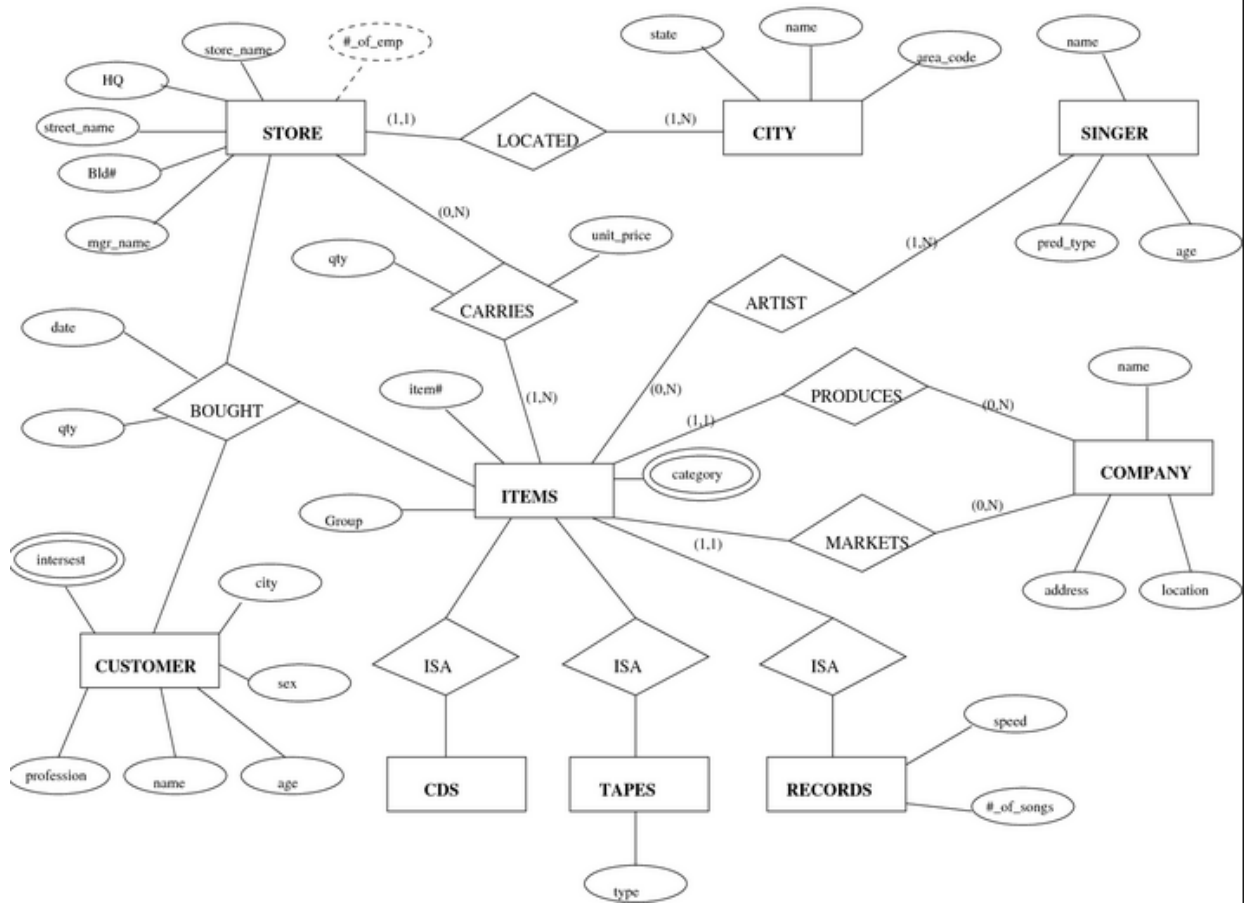
ER DIAGRAM FOR A BANK DATABASE
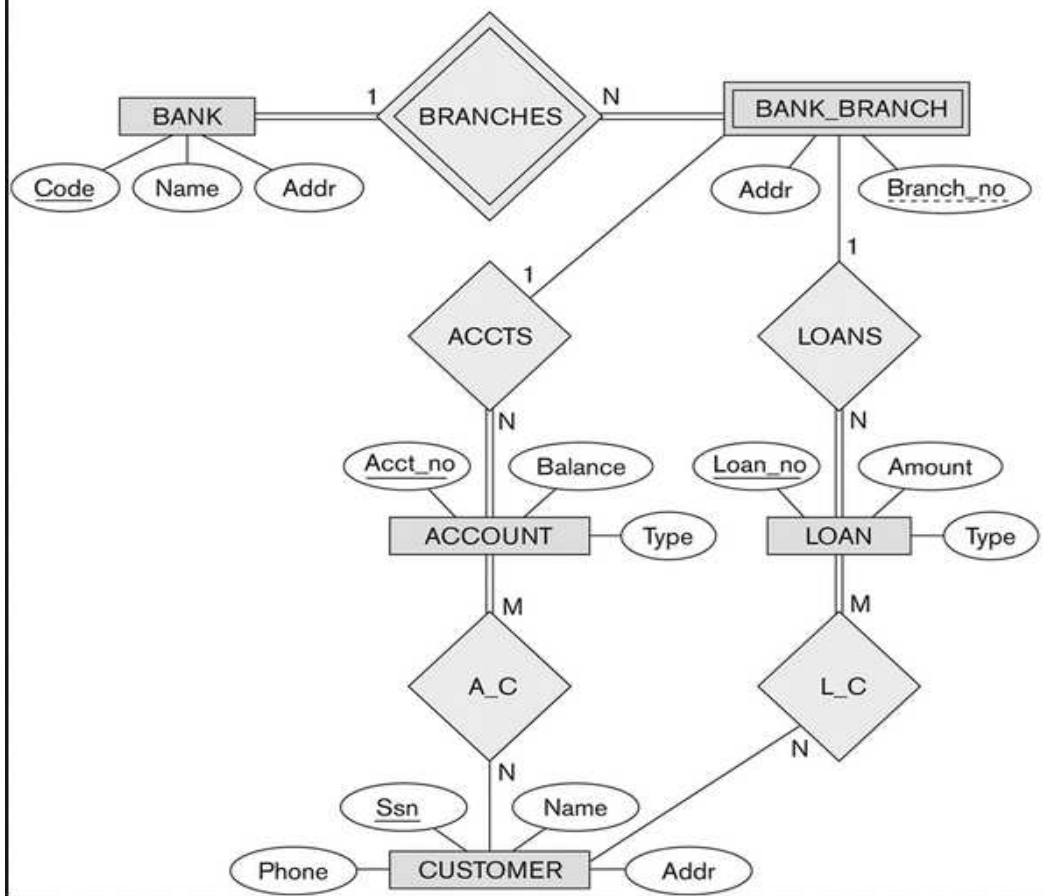
ER DIAGRAM FOR A HOSPITAL MANAGEMENT SYSTEM
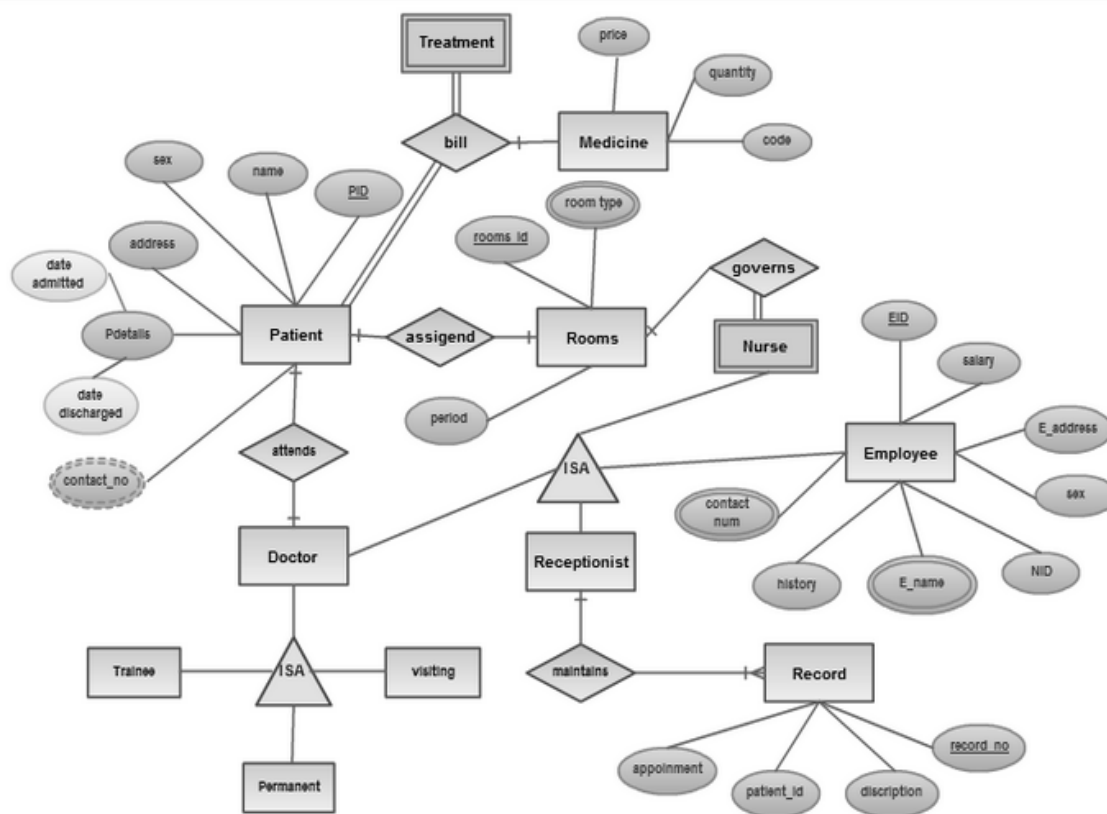
B+ Tree Index Files:

A **B+ tree** is an n-ary tree with a variable but often large number of children per node. A **B+ tree** consists of a root, internal nodes and leaves. The root may be either leaf or node with two or more children.

A **B+ tree** can be viewed as a B-tree in which each node contains only keys (not pairs), and to which an additional level is added at the bottom with linked leaves.
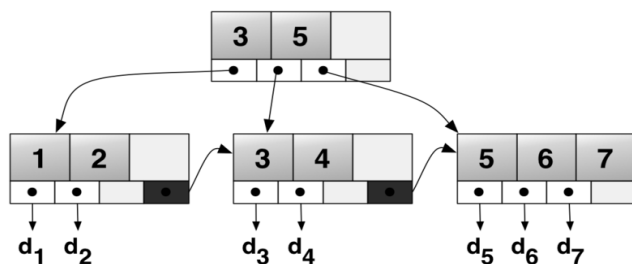
A simple B+ tree example linking the keys 1–7 to data values d_1-d_7

The primary value of a **B+ tree** is in storing data for efficient retrieval in a block-oriented storage context — in particular, file systems. This is primarily because unlike binary search trees, B+ trees have very high fan out (number of pointers to child nodes in a node, typically on the order of 100 or more), which reduces the number of I/O operations required to find an element in the tree.

Insertion

Perform a search to determine what bucket the new record should go into.

- If the bucket is not full (at most b - 1 entries after the insertion), add the record.
- Otherwise, split the bucket.
    - Allocate new leaf and move half the bucket's elements to the new bucket.
    - Insert the new leaf's smallest key and address into the parent.
    - If the parent is full, split it too.
        - Add the middle key to the parent node.
    - Repeat until a parent is found that need not split.
- If the root splits, create a new root which has one key and two pointers. (That is, the value that gets pushed to the new root gets removed from the original node)

B-trees grow at the root and not at the leaves.

Deletion

- Start at root, find leaf L where entry belongs.
- Remove the entry.
    - If L is at least half-full, done!
    - If L has fewer entries than it should,
        - Try to re-distribute, borrowing from sibling (adjacent node with same parent as L).
        - If re-distribution fails, merge L and sibling.
- If merge occurred, must delete entry (pointing to L or sibling) from parent of L.
- Merge could propagate to root, decreasing height.

Module 2
RELATIONAL ALGEBRA

Relational Algebra Operators are mathematical functions used to retrieve queries by describing a sequence operations on tables or even databases(schema) involved. With relational algebra

74

operators, a query is always composed of a number of operators, which each in turn are composed of relations as variables and return an individual abstraction as the end product.

The following are the main relational algebra operators as applied to SQL:

The SELECT Operator

The SELECT operator is used to choose a subset of the tuples(rows) from a relation that satisfies a selection condition, acting as a filter to retain only tuples that fulfills a qualifying requirement.

- The SELECT operator is relational algebra is denoted by the symbol σ (sigma).
- The syntax for the SELECT statement is then as follows:

$\sigma_{<\text{Selection condition}>}(R)$

- The σ would represent the SELECT command
- The <selection condition> would represent the condition for selection.
- The (R) would represent the Relation or the Table from which we are making a selection of the tuples.

To implement the SELECT statement in SQL, we take a look at an example in which we would like to select the EMPLOYEE tuples whose employee number is 7, or those whose date of birth is before 1980…

$\sigma_{\text{empno}=7}(\text{EMPLOYEE})$

$\sigma_{\text{dob}<'01\text{-Jan-}1980'}(\text{EMPLOYEE})$

The SQL implementation would translate into:

SELECT empno

FROM EMPLOYEE

WHERE empno=7

SELECT dob

FROM EMPLOYEE

WHERE DOB < '01-Jan-1980′

75

The PROJECT Operator

This operator is used to reorder, select and get rid of attributes from a table. At some point we might want only certain attributes in a relation and eliminate others from our query result. Therefore the PROJECT operator would be used in such operations.

- The symbol used for the PROJECT operation is $\prod$ (pi).
- The general syntax for the PROJECT operator is:

$\prod_{<attribute\ list>}(R)$

- $\prod$ would represent the ROJECT.
- <attribute list> would represent the attributes(columns) we want from a relational.
- (R) would represent the relation or table we want to choose the attributes from.

To implement the PROJECT statement in SQL, we take a look at an example in which we would like to choose the Date of Birth (dob) and Employee Number (empno) from the relation EMPLOYE…

- $\prod_{dob,\ empno}(\textbf{EMPLOYEE})$

In SQL this would translate to:

SELECT dob, empno

FROM EMPLOYEE

The RENAME Operator

The RENAME operator is used to give a name to results or output of queries, returns of selection statements, and views of queries that we would like to view at some other point in time:

- The RENAME operator is symbolized by ρ (rho).
- The general syntax for RENAME operator is: $\rho_{s(B1,\ B2,\ B3,\ldots Bn)}(R)$
- ρ is the RENAME operation.
- S is the new relation name.
- $B_1, B_2, B_3, \ldots B_n$ are the new renamed attributes (columns).
- R is the relation or table from which the attributes are chosen.

To implement the RENAME statement in SQL, we take a look at an example in which we would like to choose the Date of Birth and Employee Number attributes and RENAME them as 'Birth_Date' and 'Employee_Number' from the EMPLOYEE relation…

$$\rho_{s(Birth\_Date, Employee\_Number)}(EMPLOYEE) \leftarrow \prod_{dob, empno}(EMPLOYEE)$$

- The arrow symbol ← means that we first get the PROJECT operation results on the right side of the arrow then apply the RENAME operation on the results on the left side of the arrow.

In SQL we would translate the RENAME operator using the SQL 'AS' statement:

SELECT dob AS 'Birth_Date', empno AS 'Employee_Number'

FROM EMPLOYEE

The UNION, INTERSECTION, and MINUS Operators

UNION: the UNION operation on relation A UNION relation B designated as **A ∪ B**, joins or includes all tuples that are in A or in B, eliminating duplicate tuples. The SQL implementation of the UNION operations would be as follows:

UNION

RESULT ← **A ∪ B**

SQL Statement:

SELECT * From A

UNION

SELECT * From B

INTERSECTION: the INTERSECTION operation on a relation A INTERSECTION relation B, designated by **A ∩ B**, includes tuples that are only in A and B. In other words only tuples belonging to A and B, or shared by both A and B are included in the result. The SQL implementation of the INTERSECTION operations would be as follows:

INTERSECTION

RESULT ← **A ∩ B**

SQL Statement:

SELECT dob From A

INTERSECT

SELECT dob from B

MINUS Operations: the MINUS operation includes tuples from one Relation that are not in another Relation. Let the Relations be A and B, the MINUS operation A MINUS B is denoted by **A – B**, that results in tuples that are A and not in B. The SQL implementation of the MINUS operations would be as follows:

MINUS

RESULT ← **A – B**

SQL Statement

SELECT dob From A

MINUS

SELECT dob from B

CARTESIAN PRODUCT Operator

The CARTERSIAN PRODUCT operator, also referred to as the cross product or cross join, creates a relation that has all the attributes of A and B, allowing all the attainable combinations of tuples from A and B in the result. The CARTERSIAN PRODUCT A and B is symbolized by X as in A X B.

Let there be Relation $A(A_1, A_2)$ and Relation $B(B_1, B_2)$

The CARTERSIAN PRODUCT C of A and B which is A X B is

C = A X B

$C = (A_1B_1, A_1B_2, A_2B_1, A_2B_2)$

The SQL implementation would be something like:

SELECT A.dob, B.empno

from A, B

JOIN Operator

The JOIN operation is denoted by the ⬚ symbol and is used to compound similar tuples from two Relations into single longer tuples. Every row of the first table is joined to every row of the second table. The result is tuples taken from both tables.

- The general syntax would be A ⬚ $_{<join\ condition>}$ B

SQL translation example where attribute dob is Date of Birth and empno is Employee Number:

SELECT A.dob, A.empno

from employee

JOIN B on B.empno=A.empno

THETA JOIN Operator

This operation results in all combinations of tuples from Relation A and Relation B satisfying a join requirement. The THETA JOIN is designated by: The SQL implementation would be the same as for the JOIN example above.

- A ⬚ $_{<join\ condition>}$ B

EQUIJOIN Operator

The EQUIJOIN operation returns all combinations of tuples from Relation A and Relation B satisfying a join requirement with only equality comparisons. The EQUIJOIN operation is symbolized by :

A ⬚ $_{<join\ condition>}$ B, OR

-
- A ⬚ $_{(<join\ attributes\ 1>),}$

$_{(<join\ attributes\ 2>)}$ B

SQL translation example where attribute dob is Date of Birth and empno is Employee Number:

SELECT * from A

INNER JOIN B

on A.empno=B.empno

NATURAL JOIN Operator

The NATURAL JOIN operation returns results that does not include the JOIN attributes of the second Relation B. It is not required that attributes with the same name be mentioned. The NATURAL JOIN operator is symbolized by:

- A * ⍰ <join condition> B,

OR A * ⍰ (<join attributes 1>),

(<join attributes 2>) B

OR A * B

SQL translation example where attribute dob is Date of Birth and empno is Employee Number:

SELECT A.dob, B.empno

FROM A

NATURAL JOIN B

//where depno =5

We can always use the 'where' clause to further restrict our output and stop a Cartesian product output.

DIVISION Operator

The DIVISION operation will return a Relation R(X) that includes all tuples t[X] in R(Z) that appear in R1 in combination with every tuple from R2(Y), where $Z = X \cup Y$. The DIVISION operator is symbolized by:

- $R1(Z) \div R2(Y)$

The DIVISION operator is the most difficult to implement in SQL as no SQL command is given for DIVISION operation. The DIVISION operator would be seen as the opposite of the CARTERSIAN PRODUCT operator; just as in standard math, the relation between division and multiplication. Therefore a series of current SQL commands have to be utilized in implementation of the DIVISION operator. An example of the SQL implementation of DIVISION operator:

SELECT surname, forenames

FROM employee X

WHERE NOT EXISTS

(SELECT 'X'

FROM employee y

WHERE NOT EXISTS

(SELECT 'X'

FROM employee z

WHERE x.empno = z.empno

AND y.surname = z.surname))

ORDER BY empno

RELATIONAL CALCULUS

In relational calculus, a query is expressed as a formula consisting of a number of variables and an expression involving these variables. It is up to the DBMS to transform these nonprocedural queries into equivalent, efficient, procedural queries. The concept of relational calculus was first proposed by Codd. The relational calculus is used to measure the selective power of relational languages. A language that can be used to produce any relation that can be derived using the relational calculus is said to be relationally complete.

Relation calculus, which in effect means calculating with relations, is based on predicate calculus, which is calculating with predicates. It is a formal language used to symbolize logical arguments in mathematics. Propositions specifying a property consist of an expression that names an individual object, and another expression, called the predicate, that stands for the property that the individual object possesses. If for instance, *p* and *q* are propositions, we can build other propositions *"not p", "p or q", "p and q"* and so on. In predicate calculus, propositions may be built not only out of other propositions but also out of elements that are not themselves propositions. In this manner we can build a proposition that specifies a certain property or characteristic of an object.

TUPLE RELATIONAL CALCULUS

The tuple relational calculus is based on specifying a number of tuple variables. Each such tuple variable normally ranges over a particular database relation. This means that the variable may take any individual tuple from that relation as its value. A simple tuple relational calculus query is of the form { t I COND(t)·}, where '1' is a tuple variable and COND(t) is a conditional expression involving '1'. The result of such a query is a relation that contains all the tuples (rows) that satisfy COND(t).

For example, the relational calculus query {t I BOOK(t) and t.PRICE>lOO} will get you all the books whose price is greater than 100. In the above example, the condition 'BOOK(t)' specifies that the range relation of the tuple variable '1' is BOOK. Each BOOK tuple 't' that satisfies the condition 't.PRICE> 100' will be retrieved. Note that 't.PRICE' references the attribute PRICE of the tuple variable '1'.

The query {t IBOOK (t) and t.PRICE>100} retrieves all attribute values for each selected

BOOK tuple. To retrieve only some of the attributes (say TITLE, AUTHOR and PRICE) we can modify the query as follows:

{t.TITLE, t.AUTHOR, t.PRICE I BOOK(t) and t.PRICE>200}

Thus, in a tuple calculus expression we need to specify the following information:

For each tuple variable the range relation 'R' of 'to This value is specified by a condition of the form R(t) .

• A condition to select the required tuples from the relation.

• A set of attributes to be retrieved. This set is called the requested attributes. The values of these attributes for each selected combination of tuples. If the requested attribute list is not specified, then all the attributes of the selected tuples are retrieved.

Thus, to retrieve the details of all books (Title and Author name) which were published by 'Kalyani' and whose price is greater than 100 we will write the query as follows:

{t.TITLE, t.AUTHOR I BOOK (t) and t.PUBLISHER='xyz' and t.PRICE>100}

Expressions and Formulas

A general expression of the.tuple relational calculus is of the following form:

{tl·Aj,t2·A2, tn·An, I COND(tl,t2· .. tn, tn+j,1n+2.... tn+m)}

Where tj,t2 1n,1n+j,1n+2 1n+mare tuple variables, each Ai is an attribute of the relation on which ranges and COND is a condition or formula of the tuple relational calculus.

A formula is defined as follows:

Every condition is a WFF (Well Formed Formula). Here, a well-formed formula is constructed from conditions, Boolean operations (AND, OR, NOT) and quantifiers like for all values (V) or there exists (3).

There are following rules which are applicable on WFF:

• Every condition is WFF.

• If F is a WFF the (F) and NOT(F) are also WFF.

• If Fj and F2 are WFFs, then (Fj AND F2), (Fj OR F2) are also WFFs .

• If F is a WFF in which T occurs as a free variable (free variables are those range

variables when, the meaning of the formula changed if all the occurrences of range

variable say 'x' were replaced by some other variables say 'y') then 3 T(t) and

V T(F) are WFFs .

• Nothing else is WFF.

Bound variables: Bound variables are those range variables when the meaning of the formula would remain unchanged if all the occurrence of range variable say 'x' were replaced by some other variable say 'y'. Then range variable 'x' is called as the Bound variable.

For example: x (x>3) means

EXISTS x (x>3)

Here, WFF simply states that there exists some integer x that is greater than 3. Note, that the meaning of this WFF would remain totally unchanged if all references of x were replaced by references to some other variable y. In other words the WFF EXISTS y(y>3) is semantically same.

Free Variables: Free variables are those range variables when the meaning of the formula changed, if all the occurrences of range variable say 'x' were replaced by some other variables say 'y'. Then range variable 'x' is called as the Free variable.

For example: x (x>3) and x<O means

EXISTS x (x>3) and x <0

Here, there are three references to x, denoting two different variables. The first two references are bound and could be replaced by references to some other variable y without changing the overall meaning. The third reference is free, and cannot be replaced without changing the meaning of the formula. Thus, of the two WFFs shown below, the first is equivalent to the one just given and the second is not: -

EXIST Y (y>3) and x<O

EXITS y (y>3) and y<O

Closed and Open WFF: A WFF in which all variables references are bound is called Closed WFF. e.g. EXISTS x (x>3) is a closed WFF.

An open WFF is a WFF that is not closed i.e. one that consists of at least one free variable reference. e.g. EXISTS y (y>3) and x<O

DOMAIN RELATIONAL CALCULUS

The domain calculus differs from the tuple calculus in the type of variables used in formulas. In domain calculus the variables range over single values from domains of attributes rather than ranging over tuples. To form a relation of degree 'n' for a query result, we must have 'n' of these domain variables-one for each attribute.

An expression of the domain calculus is of the following form:

{Xl, X2, ... , Xn I COND(XI, X2, .. ·, Xn, Xn+b Xn+2, , Xn+m)}

In the above expression Xl, X2, ... , Xn, Xn+b Xn+2, , Xn+m are domain variables that range over domains of attributes and COND is a condition or formula of the domain relational calculus.

Expression of the domain calculus are constructed from the following elements:

• Domain variables Xl, X2, ... , Xn, Xn+b Xn+2, ... , Xn+m each domain variable is to range over some specified domain .

• Conditions, which can take two forms:

• Simple comparisons of the form x * y, as for the tuple calculus, except that x and yare now domain variables.

• Membership conditions, of the form R (term,

term ...).

Here, R is a relation, and each "term" is a pair AV, where A in turn is an attribute

Of R and V is either a domain variable or a constant. For example EMP (empno: 100, ename: 'Ajay') is a membership condition (which evaluates to true if and only if there exists an EMP tuple having empno=100 and ename = 'Ajay') .

• Well Formed Formulates (WFFs), formed in accordance with rules of tuple calculus (but with the revised definition of "condition").

Free and Bound Variables

The rules concerning free and bound variables given for the tuple calculus are also applicable similarly on the domain calculus.

SQL

What is SQL?

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

SQL is the standard language for Relation Database System. All relational database management systems like MySQL, MS Access, Oracle, Sybase, Informix, postgres and SQL Server uses SQL as 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?

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

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 the process. These components are Query Dispatcher, Optimization engines, Classic Query Engine and SQL query engine etc. Classic query engine handles all non-SQL queries but SQL query engine won't handle logical files.

Following is a simple diagram showing SQL Architecture:
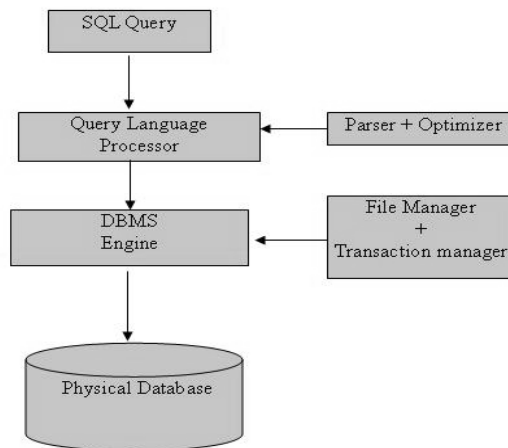
SQL DATABASE :

MySQL

MySQL is open source SQL database, which is developed by Swedish company MySQL AB. MySQL is pronounced "my ess-que-ell," in contrast with SQL, pronounced "sequel."

MySQL is supporting many different platforms including Microsoft Windows, the major Linux distributions, UNIX, and Mac OS X.

MySQL has free and paid versions, depending on its usage (non-commercial/commercial) and features. MySQL comes with a very fast, multi-threaded, multi-user, and robust SQL database server.

Features:

- High Performance.
- High Availability.
- Scalability and Flexibility Run anything.
- Robust Transactional Support.
- Web and Data Warehouse Strengths.
- Strong Data Protection.
- Comprehensive Application Development.
- Management Ease.
- Open Source Freedom and 24 x 7 Support.
- Lowest Total Cost of Ownership.

MS SQL Server

MS SQL Server is a Relational Database Management System developed by Microsoft Inc. Its primary query languages are:

- T-SQL.
- ANSI SQL.

Features:

- High Performance.
- High Availability.
- Database mirroring.
- Database snapshots.
- CLR integration.
- Service Broker.
- DDL triggers.
- Ranking functions.
- Row version-based isolation levels.
- XML integration.
- TRY...CATCH.
- Database Mail.

ORACLE

It is very large and multi-user database management system. Oracle is a relational database management system developed by 'Oracle Corporation'.

Oracle works to efficiently manage its resource, a database of information, among the multiple clients requesting and sending data in the network.

It is an excellent database server choice for client/server computing. Oracle supports all major operating systems for both clients and servers, including MSDOS, NetWare, UnixWare, OS/2 and most UNIX flavors.

Features:

- Concurrency
- Concurrency
- Read Consistency

- Locking Mechanisms
- Quiesce Database
- Portability
- Self managing database
- SQL*Plus
- ASM
- Scheduler
- Resource Manager
- Data Warehousing
- Materialized views
- Bitmap indexes
- Table compression
- Parallel Execution
- Analytic SQL
- Data mining
- Partitioning

MS- ACCESS

This is one of the most popular Microsoft products. Microsoft Access is entry-level database management software. MS Access database is not only an inexpensive but also powerful database for small-scale projects.

MS Access uses the Jet database engine which utilizes a specific SQL language dialect

Features:

- Users can create tables, queries, forms and reports, and connect them together with macros.
- The import and export of data to many formats including Excel, Outlook, ASCII, dBase, Paradox, FoxPro, SQL Server, Oracle, ODBC, etc.
- There is also the Jet Database format (MDB or ACCDB in Access 2007) which can contain the application and data in one file. This makes it very convenient to distribute the entire application to another user, who can run it in disconnected environments.
- Microsoft Access offers parameterized queries. These queries and Access tables can be referenced from other programs like VB6 and .NET through DAO or ADO.
- The desktop editions of Microsoft SQL Server can be used with Access as an alternative to the Jet Database Engine.
- Microsoft Access is a file server-based database. Unlike client-server relational database management systems (RDBMS), Microsoft Access does not implement database triggers, stored procedures, or transaction logging.

- SQL data type is an attribute that specifies type of data of any object. Each column, variable and expression has related data type in SQL.
- You would use these data types while creating your tables. You would choose a particular data type for a table column based on your requirement.
- SQL Server offers six categories of data types for your use:

SQL DATA TYPE :

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.

Character Strings Data Types:

DATA TYPE	FROM	TO
char	char	Maximum length of 8,000 characters.(Fixed length non-Unicode characters)
varchar	varchar	Maximum of 8,000 characters.(Variable-length non-Unicode data).
varchar(max)	varchar(max)	Maximum length of 231characters, Variable-length non-Unicode data (SQL Server 2005 only).
text	text	Variable-length non-Unicode data with a maximum length of 2,147,483,647 characters.

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.

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:

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:

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

Operator	Description
ALL	The ALL operator is used to compare a value to all values in another value set.
AND	The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.
ANY	The ANY operator is used to compare a value to any applicable value in the list according to the condition.
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.

EXISTS	The EXISTS operator is used to search for the presence of a row in a specified table that meets certain criteria.
IN	The IN operator is used to compare a value to a list of literal values that have been specified.
LIKE	The LIKE operator is used to compare a value to similar values using wildcard operators.
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 negate operator.**
OR	The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.
IS NULL	The NULL operator is used to compare a value with a NULL value.
UNIQUE	The UNIQUE operator searches every row of a specified table for uniqueness (no duplicates).

COMMANDS IN SQL :

1. **CREATE DATABASE**

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

Syntax:

Basic syntax of CREATE DATABASE statement is as follows:

CREATE DATABASE DatabaseName;

Always database name should be unique within the RDBMS.

Example:

If you want to create new database <testDB>, then CREATE DATABASE statement would be as follows:

SQL> CREATE DATABASE testDB;

2. **DROP DATABASE**

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

Syntax:

Basic syntax of DROP DATABASE statement is as follows:

DROP DATABASE DatabaseName;

Always database name should be unique within the RDBMS.

Example:

If you want to delete an existing database <testDB>, then DROP DATABASE statement would be as follows:

SQL> DROP DATABASE testDB;

3. **USE**

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

Syntax:

Basic syntax of USE statement is as follows:

USE DatabaseName;

4. **CREATE TABLE**

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

Syntax:

Basic syntax of CREATE TABLE statement is as follows:

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

95

CREATE TABLE is the keyword telling the database system what you want to do.in this case, you want to create a new table. The unique name or identifier for the table follows the CREATE TABLE statement.

Then in brackets comes the list defining each column in the table and what sort of data type it is. The syntax becomes clearer with an example below.

A copy of an existing table can be created using a combination of the CREATE TABLE statement and the SELECT statement.

5. **DROP TABLE**

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

Syntax:

Basic syntax of DROP TABLE statement is as follows:

DROP TABLE table_name;

6. **INSERT INTO**

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

Syntax:

There are two basic syntax of INSERT INTO statement is as follows:

INSERT INTO TABLE_NAME (column1, column2, column3,...columnN)]
VALUES (value1, value2, value3,...valueN);

Here column1, column2,...columnN are the names of the columns in the table into which you want to insert data.

You may not need to specify the column(s) name in the SQL query if you are adding values for all the columns of the table. But make sure the order of the values is in the same order as the columns in the table. The SQL INSERT INTO syntax would be as follows:

INSERT INTO TABLE_NAME VALUES

Example:

Following statements would create six records in CUSTOMERS table:

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (1, 'Ramesh', 32, 'Ahmedabad', 2000.00);

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (2, 'Khilan', 25, 'Delhi', 1500.00);

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (3, 'kaushik', 23, 'Kota', 2000.00);

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (4, 'Chaitali', 25, 'Mumbai', 6500.00);

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (5, 'Hardik', 27, 'Bhopal', 8500.00);

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (6, 'Komal', 22, 'MP', 4500.00);

You can create a record in CUSTOMERS table using second syntax as follows:

INSERT INTO CUSTOMERS
VALUES (7, 'Muffy', 24, 'Indore', 10000.00);

All the above statement would product following records in CUSTOMERS table:

```
+----+----------+-----+-----------+----------+
| 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. **SELECT**

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

Syntax:

The basic syntax of SELECT statement is as follows:

SELECT column1, column2, columnN FROM table_name;

Here column1, column2...are the fields of a table whose values you want to fetch. If you want to fetch all the fields available in the field then you can use following syntax:

SELECT * FROM table_name;

Example:

Consider CUSTOMERS table is having following records:

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

Following is an example which would fetch ID, Name and Salary fields of the customers available in CUSTOMERS table:

SQL> SELECT ID, NAME, SALARY FROM CUSTOMERS;

This would produce following result:

```
+----+----------+----------+
| ID | NAME     | SALARY   |
+----+----------+----------+
|  1 | Ramesh   |  2000.00 |
|  2 | Khilan   |  1500.00 |
```

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

8. WHERE CLAUSE

The SQL **WHERE** clause is used to specify a condition while fetching the data from single table or joining with multiple table.

If the given condition is satisfied then only it returns specific value from the table. You would use WHERE clause to filter the records and fetching only necessary records.

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

Syntax:

The basic syntax of SELECT statement with WHERE clause is as follows:

```
SELECT column1, column2, columnN
FROM table_name
WHERE [condition]
```

You can specify a condition using comparision or logical operators like >, <, =, LIKE, NOT etc. Below examples would make this concept clear.

Example:

Consider CUSTOMERS table is having following records:

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

Following is an example which would fetch ID, Name and Salary fields from the CUSTOMERS table where salary is greater than 2000:

SQL> SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE SALARY > 2000;

This would produce following result:

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

9. AND and OR OPERATORS

The SQL **AND** and **OR** operators are used to combile multiple conditions to narrow data in an SQL statement. These two operators are called conjunctive operators.

These operators provide a means to make multiple comparisons with different operators in the same SQL statement.

The AND Operator:

The **AND** operator allows the existence of multiple conditions in an SQL statement's WHERE clause.

Syntax:

The basic syntax of AND operator with WHERE clause is as follows:

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

You can combine N number of conditions using AND operator. For an action to be taken by the SQL statement, whether it be a transaction or query, all conditions separated by the AND must be TRUE.

100

Example:

Consider CUSTOMERS table is having following records:

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

Following is an example which would fetch ID, Name and Salary fields from the CUSTOMERS table where salary is greater than 2000 AND age is less tan 25 years:

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

This would produce following result:

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

10. UPDATE

The SQL **UPDATE** Query is used to modify the existing records in a table.

You can use WHERE clause with UPDATE query to update selected rows otherwise all the rows would be effected.

Syntax:

The basic syntax of UPDATE query with WHERE clause is as follows:

UPDATE table_name

```
SET column1 = value1, column2 = value2...., columnN = valueN
WHERE [condition];
```

11. **DELETE**

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

You can use WHERE clause with DELETE query to delete selected rows, otherwise all the records would be deleted.

Syntax:

The basic syntax of DELETE query with WHERE clause is as follows:

```
DELETE FROM table_name
WHERE [condition];
```

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

Example:

Consider CUSTOMERS table is having following records:

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

Following is an example which would DELETE a customer whose ID is 6:

```
SQL> DELETE FROM CUSTOMERS
WHERE ID = 6;
```

Now CUSTOMERS table would have following records:

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
```

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

12. **LIKE**

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

- The percent sign (%)
- The underscore (_)

The percent sign represents zero, one, or multiple characters. The underscore represents a single number or character. The symbols can be used in combinations.

Syntax:

The basic syntax of % and _ is as follows:

SELECT FROM table_name
WHERE column LIKE 'XXXX%'

or

SELECT FROM table_name
WHERE column LIKE '%XXXX%'

or

SELECT FROM table_name
WHERE column LIKE 'XXXX_'

consider CUSTOMERS table is having following records:

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
| 1 | Ramesh   | 32 | Ahmedabad | 2000.00 |
| 2 | Khilan   | 25 | Delhi     | 1500.00 |
| 3 | kaushik  | 23 | Kota      | 2000.00 |
```

```
| 4 | Chaitali |  25 | Mumbai    |  6500.00 |
| 5 | Hardik   |  27 | Bhopal    |  8500.00 |
| 6 | Komal    |  22 | MP        |  4500.00 |
| 7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Following is an example which would display all the records from CUSTOMERS table where SALARY starts with 200:

```
SQL> SELECT * FROM CUSTOMERS
WHERE SALARY LIKE '200%';
```

This would produce following result:

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
+----+----------+-----+-----------+----------+
```

13. **TOP**

The SQL **TOP** clause is used to fetch a TOP N number or X percent records from a table.

Syntax:

The basic syntax of TOP clause with SELECT statement would be as follows:

```
SELECT TOP number|percent column_name(s)
FROM table_name
WHERE [condition]
```

Example:

Consider CUSTOMERS table is having following records:

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
```

```
| 4 | Chaitali |  25 | Mumbai    |  6500.00 |
| 5 | Hardik   |  27 | Bhopal    |  8500.00 |
| 6 | Komal    |  22 | MP        |  4500.00 |
| 7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Following is an example on SQL server which would fetch top 3 records from CUSTOMERS table:

SQL> SELECT TOP 3 * FROM CUSTOMERS;

This would produce following result:

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad | 2000.00 |
|  2 | Khilan   |  25 | Delhi     | 1500.00 |
|  3 | kaushik  |  23 | Kota      | 2000.00 |
+----+----------+-----+-----------+----------+
```

14. ORDER BY

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

Syntax:

The basic syntax of ORDER BY clause is as follows:

SELECT column-list
FROM table_name
[WHERE condition]
[ORDER BY column1, column2, .. columnN] [ASC | DESC];

You can use more than one column in the ORDER BY clause. Make sure whatever column you are using to sort, that column should be in column-list.

Example:

Consider CUSTOMERS table is having following records:

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
```

105

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

Following is an example which would sort the result in ascending order by NAME and SALARY:

```
SQL> SELECT * FROM CUSTOMERS
    ORDER BY NAME, SALARY;
```

This would produce following result:

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

15. GROUP BY

The SQL **GROUP BY** clause is used in collaboration with the SELECT statement to arrange identical data into groups.

The GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.

Syntax:

The basic syntax of GROUP BY clause is given below. The GROUP BY clause must follow the conditions in the WHERE clause and must precede the ORDER BY clause if one is used.

```
SELECT column1, column2
FROM table_name
WHERE [ conditions ]
GROUP BY column1, column2
```

ORDER BY column1, column2

Example:

Consider CUSTOMERS table is having following records:

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

If you want to know the total amount of salary on each customer, then GROUP BY query would be as follows:

```
SQL> SELECT NAME, SUM(SALARY) FROM CUSTOMERS
   GROUP BY NAME;
```

This would produce following result:

```
+----------+-------------+
| NAME     | SUM(SALARY) |
+----------+-------------+
| Chaitali |     6500.00 |
| Hardik   |     8500.00 |
| kaushik  |     2000.00 |
| Khilan   |     1500.00 |
| Komal    |     4500.00 |
| Muffy    |    10000.00 |
| Ramesh   |     2000.00 |
+----------+-------------+
```

16. DISTINCT KEYWORD

The SQL **DISTINCT** keyword is used in conjunction with SELECT statement to eliminate all the duplicate records and fetching only unique records.

There may be a situation when you have multiple duplicate records in a table. While fetching such records, it makes more sense to fetch only unique records instead of fetching duplicate records.

107

Syntax:

The basic syntax of DISTINCT keyword to eliminate duplicate records is as follows:

```
SELECT DISTINCT column1, column2,.....columnN
FROM table_name
WHERE [condition]
```

Example:

Consider CUSTOMERS table is having following records:

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

First let us see how the following SELECT query returns duplicate salary records:

```
SQL> SELECT SALARY FROM CUSTOMERS
   ORDER BY SALARY;
```

This would produce following result where salary 2000 is coming twice which is a duplicate record from the original table.

```
+----------+
| SALARY   |
+----------+
|  1500.00 |
|  2000.00 |
|  2000.00 |
|  4500.00 |
|  6500.00 |
|  8500.00 |
| 10000.00 |
+----------+
```

Now let us use DISTINCT keyword with the above SELECT query and see the result:

```
SQL> SELECT DISTINCT SALARY FROM CUSTOMERS
   ORDER BY SALARY;
```

This would produce following result where we do not have any duplicate entry:

```
+----------+
| SALARY   |
+----------+
|  1500.00 |
|  2000.00 |
|  4500.00 |
|  6500.00 |
|  8500.00 |
| 10000.00 |
+----------+
```

17. CONSTRAINTS

Constraints are the rules enforced on data columns on table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database.

Contraints could be column level or table level. Column level constraints are applied only to one column where as table level constraints are applied to the whole table.

Following are commonly used constraints available in SQL. These constraints have already been discussed in SQL - RDBMS Concepts chapter but its worth to revise them at this point.

- NOT NULL Constraint: Ensures that a column cannot have NULL value.
- DEFAULT Constraint : Provides a default value for a column when none is specified.
- UNIQUE Constraint: Ensures that all values in a column are different.
- PRIMARY Key: Uniquely identified each rows/records in a database table.
- FOREIGN Key: Uniquely identified a rows/records in any another database table.
- CHECK Constraint: The CHECK constraint ensures that all values in a column satisfy certain conditions.
- INDEX: Use to create and retrieve data from the database very quickly.

Constraints can be specified when a table is created with the CREATE TABLE statement or you can use ALTER TABLE statment to create constraints even after the table is created.

Dropping Constraints:

Any constraint that you have defined can be dropped using the ALTER TABLE command with the DROP CONSTRAINT option.

For example, to drop the primary key constraint in the EMPLOYEES table, you can use the following command:

ALTER TABLE EMPLOYEES DROP CONSTRAINT EMPLOYEES_PK;

Some implementations may provide shortcuts for dropping certain constraints. For example, to drop the primary key constraint for a table in Oracle, you can use the following command:

ALTER TABLE EMPLOYEES DROP PRIMARY KEY;

Some implementations allow you to disable constraints. Instead of permanently dropping a constraint from the database, you may want to temporarily disable the constraint, and then enable it later.

Integrity Constraints:

Integrity constraints are used to ensure accuracy and consistency of data in a relational database. Data integrity is handled in a relational database through the concept of referential integrity.

There are many types of integrity constraints that play a role in referential integrity (RI). These constraints include Primary Key, Foreign Key, Unique Constraints and other constraints mentioned above.

The SQL **Joins** clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each.

Consider following two tables, (a) CUSTOMERS table is as follows:

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

(b) Another table is ORDERS as follows:

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

```
|OID  | DATE            | CUSTOMER_ID | AMOUNT |
+-----+--------------------+------------+--------+
| 102 | 2009-10-08 00:00:00 |        3 |  3000 |
| 100 | 2009-10-08 00:00:00 |        3 |  1500 |
| 101 | 2009-11-20 00:00:00 |        2 |  1560 |
| 103 | 2008-05-20 00:00:00 |        4 |  2060 |
+-----+--------------------+------------+--------+
```

Now let us join these two tables in our SELECT statement as follows:

```
SQL> SELECT ID, NAME, AGE, AMOUNT
      FROM CUSTOMERS, ORDERS
      WHERE  CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

This would produce following result:

```
+----+----------+-----+--------+
| ID | NAME     | AGE | AMOUNT |
+----+----------+-----+--------+
|  3 | kaushik  | 23 |  3000 |
|  3 | kaushik  | 23 |  1500 |
|  2 | Khilan   | 25 |  1560 |
|  4 | Chaitali | 25 |  2060 |
+----+----------+-----+--------+
```

Here it is noteable that the join is performed in the WHERE clause. Several operators can be used to join tables, such as =, <, >, <>, <=, >=, !=, BETWEEN, LIKE, and NOT; they can all be used to join tables. However, the most common operator is the equal symbol.

18. SQL JOIN TYPES

There are different type of joins available in SQL:

- INNER JOIN: returns rows when there is a match in both tables.
- LEFT JOIN: returns all rows from the left table, even if there are no matches in the right table.
- RIGHT JOIN: returns all rows from the right table, even if there are no matches in the left table.
- FULL JOIN: returns rows when there is a match in one of the tables.
- SELF JOIN: is used to join a table to itself, as if the table were two tables, temporarily renaming at least one table in the SQL statement.
- CARTESIAN JOIN: returns the cartesian product of the sets of records from the two or more joined tables.

19. **UNION**

The SQL **UNION** clause/operator is used to combine the results of two or more SELECT statements without returning any duplicate rows.

To use UNION, each SELECT must have the same number of columns selected, the same number of column expressions, the same data type, and have them in the same order but they do not have to be the same length.

Syntax:

The basic syntax of **UNION** is as follows:

SELECT column1 [, column2]
FROM table1 [, table2]
[WHERE condition]

UNION

SELECT column1 [, column2]
FROM table1 [, table2]
[WHERE condition]

Here given condition could be any given expression based on your requirement.

Example:

Consider following two tables, (a) CUSTOMERS table is as follows:

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

(b) Another table is ORDERS as follows:

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

```
|OID  | DATE              | CUSTOMER_ID | AMOUNT |
+-----+-------------------+-------------+--------+
| 102 | 2009-10-08 00:00:00 |       3 |  3000 |
| 100 | 2009-10-08 00:00:00 |       3 |  1500 |
| 101 | 2009-11-20 00:00:00 |       2 |  1560 |
| 103 | 2008-05-20 00:00:00 |       4 |  2060 |
+-----+-------------------+-------------+--------+
```

Now let us join these two tables in our SELECT statement as follows:

```
SQL> SELECT  ID, NAME, AMOUNT, DATE
     FROM CUSTOMERS
     LEFT JOIN ORDERS
     ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID
UNION
     SELECT  ID, NAME, AMOUNT, DATE
     FROM CUSTOMERS
     RIGHT JOIN ORDERS
     ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

This would produce following result:

```
+------+----------+--------+--------------------+
| ID   | NAME     | AMOUNT | DATE               |
+------+----------+--------+--------------------+
|  1 | Ramesh   |   NULL | NULL               |
|  2 | Khilan   |   1560 | 2009-11-20 00:00:00 |
|  3 | kaushik  |   3000 | 2009-10-08 00:00:00 |
|  3 | kaushik  |   1500 | 2009-10-08 00:00:00 |
|  4 | Chaitali |   2060 | 2008-05-20 00:00:00 |
|  5 | Hardik   |   NULL | NULL               |
|  6 | Komal    |   NULL | NULL               |
|  7 | Muffy    |   NULL | NULL               |
+------+----------+--------+--------------------+
```

20. **NULL**

The SQL **NULL** is the term used to represent a missing value. A NULL value in a table is a value in a field that appears to be blank.

A field with a NULL value is a field with no value. It is very important to understand that a NULL value is different than a zero value or a field that contains spaces.

Syntax:

The basic syntax of **NULL** while creating a table:

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

Here **NOT NULL** signifies that column should always accept an explicit value of the given data type. There are two column where we did not use NOT NULL which means these column could be NULL.

A field with a NULL value is one that has been left blank during record creation.

Example:

The NULL value can cause problems when selecting data, however, because when comparing an unknown value to any other value, the result is always unknown and not included in the final results.

You must use the **IS NULL** or **IS NOT NULL** operators in order to check for a NULL value.

Consider following table, CUSTOMERS having following records:

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

Now following is the usage of **IS NOT NULL** operator:

```
SQL> SELECT  ID, NAME, AGE, ADDRESS, SALARY
   FROM CUSTOMERS
```

```
   WHERE SALARY IS NOT NULL;
```

This would produce following result:

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

21.ALIAS

You can rename a table or a column temporarily by giving another name known as alias.

The use of table aliases means to rename a table in a particular SQL statement. The renaming is a temporary change and the actual table name does not change in the database.

The column aliases are used to rename a table's columns for the purpose of a particular SQL query.

Syntax:

The basic syntax of **table** alias is as follows:

```
SELECT column1, column2....
FROM table_name AS alias_name
WHERE [condition];
```

The basic syntax of **column** alias is as follows:

```
SELECT column_name AS alias_name
FROM table_name
WHERE [condition];
```

Example:

Consider following two tables, (a) CUSTOMERS table is as follows:

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

(b) Another table is ORDERS as follows:

```
+-----+--------------------+-------------+--------+
|OID  | DATE               | CUSTOMER_ID | AMOUNT |
+-----+--------------------+-------------+--------+
| 102 | 2009-10-08 00:00:00 |         3 |  3000 |
| 100 | 2009-10-08 00:00:00 |         3 |  1500 |
| 101 | 2009-11-20 00:00:00 |         2 |  1560 |
| 103 | 2008-05-20 00:00:00 |         4 |  2060 |
+-----+--------------------+-------------+--------+
```

Now following is the usage of **table alias**:

```
SQL> SELECT C.ID, C.NAME, C.AGE, O.AMOUNT
     FROM CUSTOMERS AS C, ORDERS AS O
     WHERE  C.ID = O.CUSTOMER_ID;
```

This would produce following result:

```
+----+---------+-----+--------+
| ID | NAME    | AGE | AMOUNT |
+----+---------+-----+--------+
| 3 | kaushik  | 23 |  3000 |
| 3 | kaushik  | 23 |  1500 |
| 2 | Khilan   | 25 |  1560 |
| 4 | Chaitali | 25 |  2060 |
+----+---------+-----+--------+
```

Following is the usage of **column alias**:

```
SQL> SELECT  ID AS CUSTOMER_ID, NAME AS CUSTOMER_NAME
   FROM CUSTOMERS
   WHERE SALARY IS NOT NULL;
```

This would produce following result:

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

116

```
| CUSTOMER_ID | CUSTOMER_NAME |
+-------------+---------------+
|           1 | Ramesh        |
|           2 | Khilan        |
|           3 | kaushik       |
|           4 | Chaitali      |
|           5 | Hardik        |
|           6 | Komal         |
|           7 | Muffy         |
+-------------+---------------+
```

22. ALTER TABLE

The SQL **ALTER TABLE** command is used to add, delete, or modify columns in an existing table.

You would also use ALTER TABLE command to add and drop various constraints on a an existing table.

Syntax:

The basic syntax of **ALTER TABLE** to add a new column in an existing table is as follows:

ALTER TABLE table_name ADD column_name datatype;

The basic syntax of ALTER TABLE to **DROP COLUMN** in an existing table is as follows:

ALTER TABLE table_name DROP COLUMN column_name;

The basic syntax of ALTER TABLE to change the **DATA TYPE** of a column in a table is as follows:

ALTER TABLE table_name MODIFY COLUMN column_name datatype;

The basic syntax of ALTER TABLE to add a **NOT NULL** constraint to a column in a table is as follows:

ALTER TABLE table_name MODIFY column_name datatype NOT NULL;

The basic syntax of ALTER TABLE to **ADD UNIQUE CONSTRAINT** to a table is as follows:

ALTER TABLE table_name
ADD CONSTRAINT MyUniqueConstraint UNIQUE(column1, column2...);

The basic syntax of ALTER TABLE to **ADD CHECK CONSTRAINT** to a table is as follows:

ALTER TABLE table_name
ADD CONSTRAINT MyUniqueConstraint CHECK (CONDITION);

The basic syntax of ALTER TABLE to **ADD PRIMARY KEY** constraint to a table is as follows:

ALTER TABLE table_name
ADD CONSTRAINT MyPrimaryKey PRIMARY KEY (column1, column2...);

The basic syntax of ALTER TABLE to **DROP CONSTRAINT** from a table is as follows:

ALTER TABLE table_name
DROP CONSTRAINT MyUniqueConstraint;

If you're using MySQL, the code is as follows:

ALTER TABLE table_name
DROP INDEX MyUniqueConstraint;

The basic syntax of ALTER TABLE to **DROP PRIMARY KEY** constraint from a table is as follows:

ALTER TABLE table_name
DROP CONSTRAINT MyPrimaryKey;

If you're using MySQL, the code is as follows:

ALTER TABLE table_name
DROP PRIMARY KEY;

Example:

Consider CUSTOMERS table is having following records:

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

118

Following is the example to ADD a new column in an existing table:

ALTER TABLE CUSTOMERS ADD SEX char(1);

Now CUSTOMERS table is changed and following would be output from SELECT statement:

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

Following is the example to DROP sex column from existing table:

ALTER TABLE CUSTOMERS DROP SEX;

Now CUSTOMERS table is changed and following would be output from SELECT statement:

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

23. TRUNCATE TABLE

The SQL **TRUNCATE TABLE** command is used to delete complete data from an existing table.

You can also use DROP TABLE command to delete complete table but it would remove complete table structure form the database and you would need to re-create this table once again if you wish you store some data.

Syntax:

The basic syntax of **TRUNCATE TABLE** is as follows:

TRUNCATE TABLE table_name;

Example:

Consider CUSTOMERS table is having following records:

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

Following is the example to turncate:

SQL > TRUNCATE TABLE CUSTOMERS;

Now CUSTOMERS table is truncated and following would be output from SELECT statement:

SQL> SELECT * FROM CUSTOMERS;
Empty set (0.00 sec)

The HAVING clause enables you to specify conditions that filter which group results appear in the final results.

The WHERE clause places conditions on the selected columns, whereas the HAVING clause places conditions on groups created by the GROUP BY clause.

Syntax:

The following is the position of the HAVING clause in a query:

SELECT
FROM
WHERE

GROUP BY
HAVING
ORDER BY

The HAVING clause must follow the GROUP BY clause in a query and must also precede the ORDER BY clause if used. The following is the syntax of the SELECT statement, including the HAVING clause:

SELECT column1, column2
FROM table1, table2
WHERE [conditions]
GROUP BY column1, column2
HAVING [conditions]
ORDER BY column1, column2

Example:

Consider CUSTOMERS table is having following records:

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

Following is the example which would display record for which similar age count would be more than or equal to 2:

SQL > SELECT *
FROM CUSTOMERS
GROUP BY age
HAVING COUNT(age) >= 2;

This would produce following result:

```
+----+--------+-----+---------+---------+
| ID | NAME   | AGE | ADDRESS | SALARY  |
+----+--------+-----+---------+---------+
| 2  | Khilan | 25  | Delhi   | 1500.00 |
+----+--------+-----+---------+---------+
```

24. DATE FUNCTIONS

Following is the list of all important Date and Time related functions available through SQL. There are various other functions supported by your RDBMS. Given list is based on MySQL RDBMS.

Name	Description
ADDDATE()	Add dates
ADDTIME()	Add time
CONVERT_TZ()	Convert from one timezone to another
CURDATE()	Return the current date
CURRENT_DATE(), CURRENT_DATE	Synonyms for CURDATE()
CURRENT_TIME(), CURRENT_TIME	Synonyms for CURTIME()
CURRENT_TIMESTAMP(), CURRENT_TIMESTAMP	Synonyms for NOW()
CURTIME()	Return the current time
DATE_ADD()	Add two dates
DATE_FORMAT()	Format date as specified
DATE_SUB()	Subtract two dates
DATE()	Extract the date part of a date or datetime expression
DATEDIFF()	Subtract two dates
DAY()	Synonym for DAYOFMONTH()
DAYNAME()	Return the name of the weekday
DAYOFMONTH()	Return the day of the month (1-31)
DAYOFWEEK()	Return the weekday index of the argument

DAYOFYEAR()	Return the day of the year (1-366)
EXTRACT	Extract part of a date
FROM_DAYS()	Convert a day number to a date
FROM_UNIXTIME()	Format date as a UNIX timestamp
HOUR()	Extract the hour
LAST_DAY	Return the last day of the month for the argument
LOCALTIME(), LOCALTIME	Synonym for NOW()
LOCALTIMESTAMP, LOCALTIMESTAMP()	Synonym for NOW()
MAKEDATE()	Create a date from the year and day of year
MAKETIME	MAKETIME()
MICROSECOND()	Return the microseconds from argument
MINUTE()	Return the minute from the argument
MONTH()	Return the month from the date passed
MONTHNAME()	Return the name of the month
NOW()	Return the current date and time
PERIOD_ADD()	Add a period to a year-month
PERIOD_DIFF()	Return the number of months between periods
QUARTER()	Return the quarter from a date argument
SEC_TO_TIME()	Converts seconds to 'HH:MM:SS' format
SECOND()	Return the second (0-59)
STR_TO_DATE()	Convert a string to a date
SUBDATE()	When invoked with three arguments a synonym for DATE_SUB()

SUBTIME()	Subtract times
SYSDATE()	Return the time at which the function executes
TIME_FORMAT()	Format as time
TIME_TO_SEC()	Return the argument converted to seconds
TIME()	Extract the time portion of the expression passed
TIMEDIFF()	Subtract time
TIMESTAMP()	With a single argument, this function returns the date or datetime expression. With two arguments, the sum of the arguments
TIMESTAMPADD()	Add an interval to a datetime expression
TIMESTAMPDIFF()	Subtract an interval from a datetime expression
TO_DAYS()	Return the date argument converted to days
UNIX_TIMESTAMP()	Return a UNIX timestamp
UTC_DATE()	Return the current UTC date
UTC_TIME()	Return the current UTC time
UTC_TIMESTAMP()	Return the current UTC date and time
WEEK()	Return the week number
WEEKDAY()	Return the weekday index
WEEKOFYEAR()	Return the calendar week of the date (1-53)
YEAR()	Return the year
YEARWEEK()	Return the year and week

Examples of SQL Commands in Different Tables:

ITEMS table

This table stores information about all the items that are offered by compnay. The structure of the table is as follows:

Column	Datatype	Meaning
Itemno	Number(5)	A unique number assigned to each item.
ItemName	Varchar2(20)	Name of the item.
Rate	Number(8,2)	Rate of the item.
taxrate	Number(4,2)	Sales tax rate for this item.

The following are the constraints related to ITEMS table:

- ITEMNO is primary key
- RATE and TAXRATE must be >= 0
- Default value for TAXRATE is 0

```
create table ITEMS
(
 itemno   number(5)  constraint items_pk  primary key,
 itemname varchar2(20),
 rate     number(8,2) constraint items_rate_chk check( rate >= 0),
 taxrate  number(4,2) default 0 constraint items_rate_chk check( rate >= 0)
);


insert into items values(1,'Samsung 14" monitor',7000,10.5);
insert into items values(2,'TVS Gold Keyboard',1000,10);
insert into items values(3,'Segate HDD 20GB',6500,12.5);
insert into items values(4,'PIII processor',8000,8);
insert into items values(5,'Logitech Mouse',500,5);
insert into items values(6,'Creative MMK',4500,11.5);
```

CUSTOMERS Table

This table contains information about customers who have placed one or more orders. The following is the structure of the table.

Column	Datatype	Meaning

Custno	Number(5)	A unique number assigned to each customer.
CustName	Varchar2(20)	Complete name of the customer.
Address1	varchar2(50)	First line of address.
Address2	varchar2(50)	Second line of address.
City	varchar2(30)	Name of the city where customer lives.
state	varchar2(30)	Name of the state where customer lives.
PinCode	varchar2(10)	Pincode of the city.
Phone	varchar2(30)	One or more phone numbers separated using comma(,).

The following are the constraint related to CUSTOMERS table.

- CUSTNO is primary key
- CUSTNAME is not null column

```
create table CUSTOMERS
(
 custno    number(5)    constraint customers_pk  primary key,
 custname  varchar2(20) constraint customers_custname_nn not null,
 address1  varchar2(50),
 address2  varchar2(50),
 city      varchar2(30),
 state     varchar2(30),
 pin       varchar2(10),
 phone     varchar2(30)
);
```

```
insert into customers values(101,'Raul','12-22-29','Dwarakanagar',
       'Vizag','AP','530016','453343,634333');
insert into customers values(102,'Denilson','43-22-22','CBM Compound',
       'Vizag','AP','530012','744545');
insert into customers values(103,'Mendiator','45-45-52','Abid Nagar',
       'Vizag','AP','530016','567434');
insert into customers values(104,'Figo','33-34-56','Muralinagar',
       'Vizag','AP','530021','875655,876563,872222');
insert into customers values(105,'Zidane','23-22-56','LB Colony',
       'Vizag','AP','530013','765533');
```

ORDERS Table

Contains information about all orders placed by customers. Contains one row for each order. The details of items ordered in an order will be found in LINEITEMS table. The following is the structure of the table.

Column	Datatype	Meaning
OrdNo	Number(5)	A unique number assigned to each order.
OrdDate	Date	Date on which order is placed.
ShipDate	Date	Date on which goods are to be shipped to customer.
Address1	varchar2(50)	First line of shipping address.
Address2	varchar2(50)	Second line of shipping address.
City	varchar2(30)	City name in shipping address.
state	varchar2(30)	State name in shipping address.
PinCode	varchar2(10)	Pincode of the city in shipping address.
Phone	varchar2(30)	One or more phone numbers separated using comma(,) of shipping place.

The following are the constraint related to ORDERS table.

- ORDNO is primary key
- CUSTNO is foreign key referencing CUSTNO of CUSTOMERS table.
- SHIPDATE must be >= ORDDATE.

```
create table ORDERS
(
 ordno    number(5) constraint orders_pk  primary key,
 orddate  date,
 shipdate date,
 custno   number(5) constraint orders_custno_pk references customers,
 address1 varchar2(50),
 address2 varchar2(50),
 city    varchar2(30),
 state   varchar2(30),
 pin     varchar2(10),
```

```
 phone     varchar2(30),
 constraint order_dates_chk  check( orddate <= shipdate)
);


insert into orders values(1001,'15-May-2001','10-jun-2001',102,
          '43-22-22','CBM Compound','Vizag','AP','530012','744545');

insert into orders values(1002,'15-May-2001','5-jun-2001',101,
  '12-22-29','Dwarakanagar','Vizag','AP','530016','453343,634333');

insert into orders values(1003,'17-May-2001','7-jun-2001',101,
  '12-22-29','Dwarakanagar','Vizag','AP','530016','453343,634333');

insert into orders values(1004,'18-May-2001','17-jun-2001',103,
   '45-45-52','Abid Nagar', 'Vizag','AP','530016','567434');

insert into orders values(1005,'20-May-2001','3-jun-2001',104,
'33-34-56','Muralinagar','Vizag','AP','530021','875655,876563,872222');

insert into orders values(1006,'23-May-2001','11-jun-2001',104,
'54-22-12','MVP Colony','Vizag','AP','530024',null);
```

LINEITEMS Table

Contains details of items ordered in each order. For each item in each order this table contains one row. The following is the structure of the table.

Column	Datatype	Meaning
OrdNo	Number(5)	Refers to the order number of the order.
Itemno	Number(5)	Refers to the item number of the item.
qty	number(3)	Howmany units of this item arerequired in this order.
price	Number(8,2)	Selling price of the item for this order.
DisRate	Number(4,2)	Discount Rate for this item in this order.

The following are the constraint related to ORDERS table.

- Primary key is ORDNO and ITEMNO.
- ORDNO is a foreign key referencing ORDNO of ORDERS table.
- ITEMNO is a foreign key referencing ITEMNO of ITEMS table.

- Default DISRATE is 0
- QTY must be >= 1
- DISRATE must be >= 0

```
create table LINEITEMS
(
 ordno   number(5)  constraint LINEITEMS_ORDNO_FK references ORDERS,
 itemno  number(5)  constraint LINEITEMS_itemno_FK references ITEMS,
 qty     number(3)  constraint LINEITEMS_qty_CHK CHECK( qty >= 1),
 price   number(8,2),
 disrate number(4,2) default 0
                 constraint LINEITEMS_DISRATE_CHK CHECK( disrate >= 0),
 constraint lineitems_pk primary key (ordno,itemno)
);


insert into lineitems values(1001,2,3,1000,10.0);
insert into lineitems values(1001,1,3,7000,15.0);
insert into lineitems values(1001,4,2,8000,10.0);
insert into lineitems values(1001,6,1,4500,10.0);

insert into lineitems values(1002,6,4,4500,20.0);
insert into lineitems values(1002,4,2,8000,15.0);
insert into lineitems values(1002,5,2,600,10.0);

insert into lineitems values(1003,5,10,500,0.0);
insert into lineitems values(1003,6,2,4750,5.0);

insert into lineitems values(1004,1,1,7000,10.0);
insert into lineitems values(1004,3,2,6500,10.0);
insert into lineitems values(1004,4,1,8000,20.0);

insert into lineitems values(1005,6,1,4600,10.0);
insert into lineitems values(1005,2,2,900,10.0);

insert into lineitems values(1006,2,10,950,20.0);
insert into lineitems values(1006,4,5,7800,10.0);
insert into lineitems values(1006,3,5,6600,15.0);
```

Procedure to create tables and data

You can download and run sql scripts used to create these tables and data using the following procedure.

1. First download tables.sql.
2. Get into SQL*PLUS and run it by using START filename. Where *filename* is complete path of the tables.sql file in your system. For example, if you downloaded the file into c:\downloads then the command will be
   SQL>START C:\DOWNLOADS\TABLES.SQL
   This will create all four tables with required constraints.
3. Download data.sql
4. Follow the same procedure as above to run it using START command in SQL*PLUS.

Queries

DISPLAY DETAILS OF ITEMS WHERE ITEMNAME CONTAINS LETTER 'O' TWICE

SELECT * FROM ITEMS
WHERE ITEMNAME LIKE '%O%O%';

DISPLAY ITEMNO,NAME,PRICE AND SELLING PRICE(PRICE+TAX) ROUND SELLING PRICE TO 100

SELECT ITEMNO, ITEMNAME, RATE, ROUND(RATE + RATE * TAXRATE /100) "SPRICE"
FROM ITEMS;

DISPLAY DETAILS OF ITEMS BY PADDING ITEMNAME TO 20 CHARACTERS WITH '.' AND IN UPPERCASE

SELECT ITEMNO, UPPER(RPAD(ITEMNAME,20,'.')) ITEMNAME, RATE, TAXRATE
FROM ITEMS;

DISPLAY CUSTNO,NAME AND ADDRESS

COLUMN ADDRESS FORMAT A40
COLUMN PHONE FORMAT A15
SELECT CUSTNO, CUSTNAME, TRIM(ADDRESS1 || ',' || ADDRESS2 || ',' || CITY ||
   ',' || STATE || ',' || PIN) ADDRESS , PHONE
FROM CUSTOMERS;

DISPLAY ORDERDATE,APPROXIMATE SHIPDATE, WHICH WILL BE COMMING MONDAY AFTER 7 DAYS FROM ORDERDATE

SELECT ORDNO,ORDDATE, NEXT_DAY(ORDDATE+7,'MON') SHIPDATE
FROM ORDERS;

DISPLAY ALL THE ORDERS THAT ARE PLACED IN THE CURRENT MONTH

```
SELECT * FROM ORDERS
WHERE TO_CHAR(ORDDATE,'MMYY') = TO_CHAR(SYSDATE,'MMYY');
```

DISPLAY THE ORDERS THAT WERE PLACED IN THE LASTWEEK OF PREVIOUS MONTH

```
SELECT * FROM ORDERS
WHERE  ORDDATE BETWEEN  LAST_DAY( ADD_MONTHS(SYSDATE,-1)) - 7
    AND  LAST_DAY( ADD_MONTHS(SYSDATE,-1));
```

DISPLAY ORDERNO,ORDERDATE IN DD-MM HH24:MI FORMAT,SHIPDATE IF NOT AVAILABLE TAKE IT AS 15 DAYS FROM THE DAY OF ORDER

```
SELECT ORDNO, TO_CHAR(ORDDATE,'DD-MM HH24:MI') ORDDATE,
NVL(SHIPDATE,ORDDATE + 15)  SHIPDATE
FROM  ORDERS;
```

DISPALY TOTAL NO OF ORDERS

```
SELECT COUNT(*) "TOTAL NO. ORDERS"
FROM  ORDERS;
```

DISPLY ORDERNO,NO.OF ITEMS IN AN ORDER AND AVG RATE OF ORDERS

```
SELECT ORDNO, COUNT(*) "NO ITEMS", ROUND(AVG(PRICE),2) "AVERAGE RATE"
FROM LINEITEMS
GROUP BY ORDNO;
```

DISPLAY ORDERNO FOR ORDERS WHERE ATLEAST ONE PRODUCT IS HAVING RATE MORE THAN 5000 AND TOTAL NO.OF UNITS IS MORE THAN 10

```
SELECT ORDNO
FROM   LINEITEMS
GROUP  BY ORDNO
HAVING MAX(PRICE) > 5000 AND SUM(QTY) > 10;
```

DISPLAY MONTH NAME AND NO.OF ORDERS RECEIVED IN THE MONTH

```
SELECT  TO_CHAR(ORDDATE,'MONTH') MONTH, COUNT(*) "NO. ORDERS"
FROM  ORDERS
GROUP BY TO_CHAR(ORDDATE,'MONTH');
```

DISPLAY CUSTNO WHO HAVE PLACED MORE THAN 2 ORDERS IN THE LAST 3 MONTHS

```
SELECT CUSTNO
FROM   ORDERS
WHERE  ORDDATE >  ADD_MONTHS(SYSDATE,-3)
GROUP  BY CUSTNO
HAVING COUNT(*) > 2;
```

DISPLAY CUSTNO,NO.OF ORDERS ,DATE OF MOST RECENT ORDER

```
SELECT  CUSTNO, COUNT(*) "NO. ORDERS", MAX(ORDDATE) "RECENT ORDER ON"
FROM  ORDERS
GROUP BY CUSTNO;
```

DISPLAY CUSTNO,DATE ON WHICH FIRST ORDER WAS PLACED AND THE GAP
BETWEEN FIRST ORDER AND LAST ORDER IN DAYS

```
SELECT CUSTNO, MIN(ORDDATE) "FIRST ORDER", MAX(ORDDATE)  - MIN(ORDDATE)
"GAP IN DAYS"
FROM   ORDERS
GROUP BY CUSTNO;
```

DISPLAY ORDERNO,MAX PRICE IN THE ORDER FOR THE ORDERS WHERE THE
AMOUNT OF ITEMS IS MORE THAN 10000

```
SELECT  ORDNO, MAX(PRICE) "MAX PRICE"
FROM    LINEITEMS
GROUP BY ORDNO
HAVING SUM(PRICE * QTY) > 10000;
```

DISPLAY ITEMNO,TOTAL NO.OF UNITS SOLD,MAXPRICE,MINPRICE

```
SELECT ITEMNO, SUM(QTY) "TOTAL NO. UNITS", MAX(PRICE), MIN(PRICE)
FROM   LINEITEMS
GROUP  BY  ITEMNO;
```

DISPLAY CUSTNO,DATE,NO.OF ORDERS PLACED

```
SELECT CUSTNO, ORDDATE, COUNT(*) "NO. ORDRES"
FROM   ORDERS
GROUP BY CUSTNO, ORDDATE;
```

DISPLAY ORDERNO,CUSTNAME,ORDERDATE,NO.OF DATE BETWEEN SHIPDATE AND
ORDERDATE FOR ORDERS THAT HAVE BEEN SHIPPED

```
SELECT ORDNO, CUSTNAME, ORDDATE, SHIPDATE - ORDDATE "DAYS"
```

FROM ORDERS O, CUSTOMERS C
WHERE SHIPDATE IS NOT NULL AND O.CUSTNO = C.CUSTNO;

DISPLAY ORDERNO,ORDERDATE,CUSTNO,NAME FOR ALL THE ORDERS WHERE THE
ORDER CONTAINS ORDER FOR ITEMNO 5.

SELECT O.ORDNO, ORDDATE, O.CUSTNO, CUSTNAME
FROM   ORDERS O, CUSTOMERS C, LINEITEMS L
WHERE ITEMNO = 5 AND L.ORDNO = O.ORDNO AND O.CUSTNO = C.CUSTNO;

The above query can also be written as follows.

SELECT ORDNO, ORDDATE, O.CUSTNO, CUSTNAME

FROM ORDERS O, CUSTOMERS C

WHERE O.CUSTNO = C.CUSTNO

   AND ORDNO IN

      (SELECT ORDNO FROM LINEITEMS WHERE ITEMNO = 5);

DISPLAY ITEMNO,NAME,ORDERNO,CUSTNAME AND AMOUNT.

SELECT I.ITEMNO, ITEMNAME, O.ORDNO, CUSTNAME, PRICE * QTY "AMOUNT"
FROM   CUSTOMERS C, ORDERS O, LINEITEMS L, ITEMS I
WHERE O.CUSTNO = C.CUSTNO AND O.ORDNO = L.ORDNO
    AND I.ITEMNO = L.ITEMNO

DISPLAY DETAILS OF ORDEERS IN WHICH ORDERDATE IS AS MONDAY AND
CUSTOMER RESIDES IN VSP

SELECT * FROM ORDERS
WHERE TO_CHAR(ORDDATE,'fmDAY') = 'MONDAY'
 AND CUSTNO IN (SELECT CUSTNO FROM CUSTOMERS WHERE CITY LIKE '%VIS%');

DISPLAY DETAILS OF CUSTOMERS WHO PLACED ANY ORDERS WORTH MORE THAN
30000

```
SELECT * FROM CUSTOMERS
WHERE  CUSTNO IN
 ( SELECT CUSTNO
   FROM ORDERS
   WHERE  ORDNO IN
     ( SELECT  ORDNO
       FROM   LINEITEMS
       GROUP BY ORDNO
       HAVING  SUM(QTY*PRICE) > 30000)
 );
```

DISPLAY DETAILS OF ITEMS FOR WHICH THERE IS AN ORDER IN THE CURRENT
MONTH

```
SELECT * FROM ITEMS
WHERE ITEMNO IN
  ( SELECT ITEMNO
    FROM LINEITEMS
    WHERE ORDNO IN
     ( SELECT ORDNO
       FROM   ORDERS
       WHERE  TO_CHAR(ORDDATE,'MMYY') = TO_CHAR(SYSDATE,'MMYY')
     )
  );
```

DISPLAY DETAILS OF ORDER IN WHICH WE SOLD ITEM 3 FOR MAX PRICE

```
SELECT * FROM ORDERS
WHERE ORDNO IN
  (
    SELECT ORDNO
    FROM   LINEITEMS
    WHERE  PRICE =
     ( SELECT MAX(PRICE) FROM  LINEITEMS
       WHERE  ITEMNO = 3)
    AND  ITEMNO = 3
  );
```

DISPLAY DETAILS OF ITEMS FOR WHICH THERE IS AN ORDER IN THE LAST 7 DAYS
OR TOTAL NO.OF UNITS ORDERED IS MORE THAN 10.

```
SELECT * FROM ITEMS
WHERE ITEMNO IN
  (SELECT ITEMNO
```

```
   FROM   LINEITEMS
   WHERE  ORDNO IN
     (SELECT ORDNO FROM ORDERS WHERE SYSDATE-ORDDATE <= 7)
  )
  OR ITEMNO IN
   ( SELECT ITEMNO
     FROM   LINEITEMS
     GROUP BY ITEMNO
     HAVING SUM(QTY) > 10
   );
```

DISPLAY ALL THE LINEITEMS IN WHICH THE RATE OF THE ITEM IS MORE THAN AVG
RATE OF THE ITEMS

```
SELECT * FROM LINEITEMS L
WHERE  PRICE >
 (SELECT  AVG(PRICE)
  FROM LINEITEMS
  WHERE  ITEMNO = L.ITEMNO);
```

DISPLAY DETAILS OF CUSTOMER WHO HAS PLACED MAX NO OF ORDERS

```
SELECT * FROM CUSTOMERS
WHERE CUSTNO IN
 ( SELECT  CUSTNO
   FROM    ORDERS
   GROUP  BY CUSTNO
   HAVING  COUNT(*) =
    (
     SELECT MAX(COUNT(*))
     FROM   ORDERS
     GROUP BY CUSTNO
    )
  );
```

DISPLAY DETAILS OF ORDERS IN WHICH ATLEAST ONE ITEM IS SOLD FOR HIGHER
RATE THAN ACTUAL RATE

```
SELECT * FROM ORDERS
WHERE  ORDNO  IN
 ( SELECT  ORDNO
   FROM    LINEITEMS L, ITEMS I
```

```
    WHERE   L.ITEMNO = I.ITEMNO
        AND PRICE > RATE );
```

DETAILS OF CUSTOMERS WHO HAVE NOT PLACED ANY ORDER FOR THE LAST 15 DAYS

```
SELECT * FROM CUSTOMERS
WHERE  CUSTNO NOT IN
 (SELECT CUSTNO
  FROM   ORDERS
  WHERE  SYSDATE - ORDDATE <= 15);
```

DISPLAY DETAILS OF ITEMS FOR WHICH THERE WAS NO ORDER IN THE PREVIOUS MONTH

```
SELECT * FROM ITEMS
WHERE ITEMNO NOT IN
 (
   SELECT ITEMNO
   FROM   LINEITEMS
   WHERE  ORDNO IN
     ( SELECT ORDNO
      FROM   ORDERS
      WHERE  TO_CHAR(ORDDATE,'MMYY') = TO_CHAR( ADD_MONTHS(SYSDATE,-
1),'MMYY')
      )
 );
```

DISPLAY ORDERS WHERE ORDDATE IS IN THE CURRENT MONTH OR AFTER ORDER 1004.

```
SELECT O1.*
FROM ORDERS O1, ORDERS O2
WHERE TO_CHAR( O1.ORDDATE,'MMYY') = TO_CHAR(SYSDATE,'MMYY')
   OR (O2.ORDNO = 1004 AND O1.ORDDATE > O2.ORDDATE);
```

DISPLAY DETAILS OF ITEMS THAT ARE PURCHASED BY CUSTOMER 102

```
SELECT * FROM ITEMS
WHERE ITEMNO IN
    ( SELECT ITEMNO
     FROM   LINEITEMS
     WHERE  ORDNO IN
       ( SELECT ORDNO
```

```
         FROM ORDERS
         WHERE  CUSTNO = 102
       )
   );


DISPLAY DETAILS OF ITEMS THAT ARE PURCHASED BY CUSTOMER 102

SELECT * FROM ITEMS
WHERE ITEMNO IN
    ( SELECT ITEMNO
      FROM   LINEITEMS
      WHERE  ORDNO IN
       ( SELECT ORDNO
         FROM ORDERS
         WHERE  CUSTNO = 102
       )
    );
```

CHANGE SHIPDATE OF ORDER 1004 TO THE ORDER DATE OF MOST RECENT ORDER

```
UPDATE ORDERS
   SET SHIPDATE = ( SELECT  MAX(ORDDATE)
            FROM ORDERS)
WHERE  ORDNO = 1004;
```

DISPLAY THE DETAILS OF ITEMS WHERE ITEMNAME CONTAINS LETTER O OR M

```
SELECT * FROM ITEMS
WHERE ITEMNAME LIKE '%O%' OR ITEMNAME LIKE '%M%';
```

DISPLAY DETAILS OF ORDERS THAT WERE PLACED IN THE MONTH OF JUNE 2000.

```
SELECT * FROM ORDERS
WHERE ORDDATE BETWEEN '01-JUN-2000' AND '30-JUN-2000';
```

DISPLAY ORDERNO,ORDERDATE AND APPROXIMATE SHIPDATE(15 DAYS FROM ORDDATE) FOR ALL ORDERS THAT ARE NOT SHIPPED.

SELECT ORDNO, ORDDATE, ORDDATE + 15 "SHIPDATE"
FROM ORDER WHERE SHIPDATE IS NULL;

DISPLAY ITEMNO,ORDERNO AND TOTAL AMOUNT AFTER ROUNDING THE VALUE TO 100'S FOR ALL THE ITEMS WHERE THE QUANTITY IS MORE THAN 5 UNITS OR PRICE IS LESS THAN 5000.

SELECT ITEMNO, ORDNO, ROUND(QTY*PRICE,-2) "TOTAL"
FROM LINEITEMS
WHERE QTY > 5 OR PRICE < 5000;

DISPLAY ITEMNO,ITEMNAME,PRICE AND TAX FOR ITEMS THAT ARE TAXABLE.

SELECT ITEMNO, ITEMNAME, PRICE , PRICE * TAX /100 "TAX"
FROM ITEMS
WHERE TAXRATE IS NOT NULL;

DISPLAY ORDERNO,CUSTMERNO,ORDERDATE,NO. OF DAYS BETWEEN DAYS ORDERDATE AND SYSTEM DATE AND DATE ON WHICH THE AMOUNT SHOULD BE COLLECTED, WHICH IS 5TH OF NEXT MONTH OF THE MONTH IN WHICH ITEMS ARE DELIVERED.
  SELECT ORDNO, CUSTNO, ORDDATE, SYSDATE - ORDDATE "NODAYS" ,
LAST_DAY(SHIPDATE) + 5 "COLLDATE"
FROM ORDERS
WHERE SHIPDATE IS NOT NULL;

DISPLAY THE DETAILS OF ORDERS THAT PLACED IN THE LAST 20 DAYS AND DELIVERED.

SELECT * FROM ORDERS
WHERE SYSDATE - ORDDATE <= 20 AND SHIPDATE IS NOT NULL;

CHANGE THE RATE OF ITEMS IN ORDER 1003 SO THAT 10% DISCOUNT IS GIVEN TO ALL ITEMS.

UPDATE LINEITEMS SET PRICE = PRICE * 0.90
WHERE ORDNO = 1003;

DISPLAY THE ITEMS WHERE ITEMNAME CONTAINS MORE THAN 10 CHARACTERS.

SELECT * FROM ITEMS

WHERE LENGTH(ITEMNAME) > 10;

DISPLAY ITEMS WHERE ITEMNAME CONTAINS LETTER 'O' AFTER 5TH POSITION.

SELECT * FROM ITEMS
WHERE INSTR(ITEMNAME,'0') > 5;

DISPLAY FIRST NAME OF THE CUSTOMER.

SELECT SUBSTR(ITEMNAME,1, INSTR(ITEMNAME,' ') -1) "FIRST NAME"
FROM CUSTOMERS;

DISPLAY ITEMNO,ITEMNAME IN UPPER CASE FOR ALL ITEMS WHERE THE LETTER
'M' IS EXISTING IN ANY CASE.

SELECT ITEMNO, UPPER(ITEMNAME)
FROM ITEMS
WHERE UPPER(ITEMNAME) LIKE '%M%';

DISPLAY THE ORDERS THAT ARE PLACED IN THE CURRENT MONTH.

SELECT * FROM ORDERS
WHERE TO_CHAR(ORDDATE,'YYMM') = TO_CHAR(SYSDATE,'YYMM');

INSERT INTO A NEW ORDER WITH THE FOLLOWING: ORDERNO-1010,CUSTOMERNO-
105,ORDERDATE-13-JULY-2001 AT 4:45 PM,SHIPDATE-NULL, SHIPADDRESS-NULL.

INSERT INTO ORDERS VALUES(1010,TO_DATE('13-07-2001 16:45','DD-MM-YYYY
HH24:MI'),NULL,105,
                NULL,NULL,NULL,NULL,NULL,NULL);

DISPLAY ORDERNO,CUSTOMERNO,THE NO. OF DAYS BETWEEN SHIPDATE AND
ORDERDATE.IF SHIPDATE IS-NOT AVAILABLE, TAKE IT AS SYSTEM DATE.

SELECT ORDNO,CUSTNO, NVL(SHIPDATE,SYSDATE)-ORDDATE
FROM ORDERS;

DISPLAY ITEMNO,PRICE,QUANTITY,DISCOUNT RATE FOR ITEMS WHERE THE
DISCOUNT RATE IS NON-ZERO. DISCOUNT-RATE IS CALUCULATED AS 10% FOR ITEM
1,7% FOR ITEM 6 AND 8% FOR REMAINING.

```
SELECT ITEMNO, PRICE, QTY, DECODE(ITEMNO,1,10,6,7,10) "DISRATE"
FROM LINEITEMS
WHERE DISRATE <> 0
```

DISPLAY TOTAL AMOUNT OF ORDERS WE RECEIVED SO FAR.

```
SELECT  SUM(QTY*PRICE)
FROM LINEITEMS;
```

DISPLAY CUSTOMERNO,MONTH-NAME,NO. OF ORDERS OF THE CURRENT YEAR.

```
SELECT  CUSTNO, TO_CHAR(ORDDATE,'MONTH'), COUNT(*)
FROM ORDERS
GROUP BY CUSTNO, TO_CHAR(ORDDATE,'MONTH');
```

DISPLAY DIFFERENCE BETWEEN HIGHEST PRICE AND LOWEST PRICE AT WHICH THE
ITEM WAS SOLD.

```
SELECT MAX(PRICE) - MIN(PRICE)
FROM LINEITEMS
GROUP BY ITEMNO;
```

DISPLAY HOW MANY ORDERS ARE STILL PENDING.

```
SELECT COUNT(*)
FROM ORDERS
WHERE SHIPDATE IS NULL;
```

DISPLAY ORDERNO,AVERAGE OF PRICE BY TAKING INTO ORDERS THAT WERE
PLACED IN THE LAST 15 DAYS.

```
SELECT O.ORDNO, AVG(PRICE)
FROM  ORDERS O, LINEITEMS L
WHERE  O.ORDNO = L.ORDNO AND  SYSDATE - ORDDATE <= 15
GROUP BY O.ORDNO;
```

DISPLAY YEAR,NO.OF ORDERS IN WHICH THE DIFFERENCE BETWEEN SHIPDATE
AND ORDERDATE IS LESS THAN 10 DAYS.

```
SELECT  TO_CHAR(ORDDATE,'YYYY'), COUNT(*)
FROM  ORDERS
WHERE  SHIPDATE  - ORDDATE <=10
GROUP BY TO_CHAR(ORDDATE,'YYYY');
```

DISPLAY STATE,NO.OF CUSTOMERS IN THE STATE WHERE THE CUSTOMER NAME
CONTAINS THE WORD 'NIKE'.

```
SELECT STATE, COUNT(*)
FROM  CUSTOMERS
WHERE  CUSTNAME LIKE '%NIKE%'
GROUP BY STATE;
```

DISPLAY CUSTOMER WHO HAS PLACED MORE THAN 2 ORDERS IN A SINGLE MONTH.

```
SELECT CUSTNO
FROM ORDERS
GROUP BY  CUSTNO, TO_CHAR(ORDDATE,'MMYY')
HAVING COUNT(*) > 2;
```

DISPLAY HIGHEST NO.OF ORDERS PLACED BY A SINGLE CUSTOMER.

```
SELECT  MAX( COUNT(*))
FROM  ORDERS
GROUP BY  CUSTNO;
```

DISPLAY CUSTOMERNO,NO.OF COMPLETED ORDERS AND NO.OF INCOMPLETE
ORDERS.

```
SELECT CUSTNO, SUM( DECODE(SHIPDATE,NULL,1,0) ) "INCOMP ORDERS", SUM(
DECODE(SHIPDATE,NULL,0,1)) "COMP ORDERS"
FROM  ORDERS
GROUP BY CUSTNO;
```

DISPLAY ORDERNO,ITEMNO,ITEMNAME,PRICE AT WHICH ITEM IS SOLD AND
CURRENT PRICE OF THE ITEM.

```
SELECT  ORDNO, L.ITEMNO, ITEMNAME, PRICE,RATE
FROM  LINEITEMS L , ITEMS I
WHERE  L.ITEMNO  = I.ITEMNO;
```

DISPLAY ORDERNO,ITEMNO,AMOUNT FOR ITEMS WHERE THE PRICE OF THE ITEM IS
MORE THAN THE CURRENT PRICE OF THE ITEM.

SELECT ORDNO, L.ITEMNO, QTY * PRICE
FROM LINEITEMS L, ITEMS I
WHERE PRICE > RATE
  AND L.ITEMNO = I.ITEMNO;

DISPLAY ITEMNO,ITEMNAME,ORDERNO,DIFFERENCE BETWEEN CURRENT PRICE
AND SELLING PRICE FOR THE ITEMS WHERE THERE IS A DIFFERENCE BETWEEN
CURRENT PRICE AND SELLING PRICE.

SELECT L.ITEMNO, ITEMNAME, ORDNO, RATE- PRICE
FROM   ITEMS I, LINEITEMS L
WHERE I.ITEMNO = L.ITEMNO AND RATE <>PRICE;

DISPLAY CUSTOMERNO,CUTOMER NAME,ORDERNO, ORDERDATE FOR ORDERS
WHERE THE SHIPADDRESS AND CUSTOMER ADDRESS ARE SAME.

SELECT O.CUSTNO, CUSTNAME, ORDNO, ORDDATE
FROM ORDERS O, CUSTOMERS C
WHERE O.ADDRESS1 = C.ADDRESS1 AND O.ADDRESS2= C.ADDRESS2 AND C.CITY =
O.CITY
AND C.STATE = O.STATE AND C.PIN = O.PIN;

DISPLAY ITEMNO,ITEMNAME,ORDERNO,QUANTITY REQUIRED FOR ALL ITEMS
(THAT ARE NOT EVEN ORDERED FOR).

SELECT I.ITEMNO, ITEMNAME, ORDNO, QTY
FROM   LINEITEMS L , ITEMS I
WHERE   I.ITEMNO = L.ITEMNO(+);

DISPLAY NO.OF ORDERS PLACED BY CUSTOMERS RESIDING IN VIZAG.

SELECT O.CUSTNO, COUNT(*)
FROM ORDERS O, CUSTOMERS C
WHERE O.CUSTNO = C.CUSTNO AND C.CITY = 'VIZAG'
GROUP BY O.CUSTNO;

DISPLAY ORDERNO,CUSTOMER NAME,DIFFERENCE BETWEEN SYSTEM DATE AND
ORDERDATE FOR ORDERS THAT HAVE NOT BEEN SHIPPED AND OLDER THAN 10
DAYS.

SELECT ORDNO, CUSTNAME, SYSDATE - ORDDATE
FROM   ORDERS O, CUSTOMERS C
WHERE O.CUSTNO = C.CUSTNO AND SYSDATE - ORDDATE > 10 AND SHIPDATE IS
NULL;

DISPLAY CUSTOMER NAME AND TOTAL AMOUNT OF ITEMS PURCHASED BY
CUSTOMER.

SELECT CUSTNAME, SUM(QTY * PRICE)
FROM   LINEITEMS L, ORDERS O, CUSTOMERS C
WHERE   L.ORDNO = O.ORDNO AND O.CUSTNO = C.CUSTNO
GROUP BY CUSTNAME;

DISPLAY THE DETAILS OF ITEM THAT HAS HIGHEST PRICE.

  SELECT * FROM ITEMS
  WHERE RATE = (SELECT MAX(RATE) FROM ITEMS);

DISPLAY DETAILS OF CUSTOMERS WHO PLACED MORE THAN 5 ORDERS.

SELECT * FROM CUSTOMERS
WHERE CUSTNO IN (SELECT CUSTNO FROM ORDERS GROUP BY CUSTNO HAVING
COUNT(*) > 5);

DISPLAY DETAILS OF CUTOMERS WHO HAVE NOT PLACED ANY ORDER.

SELECT * FROM CUSTOMERS
WHERE CUSTNO NOT IN (SELECT CUSTNO FROM ORDERS);

DISPLAY DETAILS OF CUTOMERS WHO HAVE PLACED AN ORDER IN THE LAST 6
MONTHS.

SELECT * FROM CUSTOMERS
WHERE CUSTNO IN (SELECT CUSTNO FROM ORDERS WHERE
MONTHS_BETWEEN(SYSDATE,ORDDATE) <= 6);

DISPLAY THE ITEMS FOR WHICH WE HAVE SOLD MORE THAN 50 UNITS BY TAKING INTO ORDERS WHERE THE PRICE IS MORE THAN 5000.

SELECT * FROM ITEMS
WHERE ITEMNO IN (SELECT ITEMNO FROM LINEITEMS WHERE PRICE > 5000 GROUP BY ITEMNO
          HAVING SUM(QTY) > 50);

DISPLAY THE DETAILS OF ORDERS THAT WERE PLACED BY A CUSTOMER WITH PHONE NUMBER STARTING WITH 541 OR THE ORDERS IN WHICH WE HAVE MORE THAN 5 ITEMS.

SELECT * FROM ORDERS
WHERE CUSTNO IN (SELECT CUSTNO FROM CUSTOMERS WHERE PHONE LIKE '541%')
 OR ORDNO IN (SELECT ORDNO FROM LINEITEMS GROUP BY ORDNO HAVING COUNT(*) > 5);

CHANGE THE RATE OF ITEMNO 1 IN ITEMS TABLE TO THE HIGHEST RATE OF LINEITEMS TABLE OF THAT ITEM.

UPDATE ITEMS SET RATE = (SELECT MAX(PRICE) FROM LINEITEMS WHERE ITEMNO = 1)
WHERE ITEMNO = 1;

DELETE CUSTOMERS WHO HAVE NOT PLACED ANY ORDER.

DELETE FROM CUSTOMERS WHERE CUSTNO NOT IN (SELECT CUSTNO FROM ORDERS);

RENAME COLUMN RATE IN ITEMS TO PRICE

STEP1: CREATE TABLE NEWITEMS AS SELECT ITEMNO, ITEMNAME, RATE PRICE, TAXRATE
    FROM ITEMS;

STEP2: DROP TABLE ITEMS;

STEP3: RENAME NEWITEMS TO ITEMS;

DISPLAY DETAILS OF CUSTOMERS WHO HAVE PLACED MAXIMUM NUMBER OF ORDERS.

SELECT * FROM CUSTOMERS
WHERE CUSTNO IN (SELECT CUSTNO FROM ORDERS

144

```
            GROUP BY CUSTNO HAVING COUNT(*) =
               ( SELECT  MAX(COUNT(*))
                 FROM   ORDERS
                 GROUP BY  CUSTNO));
```

DISPLAY DETAILS OF CUSTOMERS WHO HAVEN'T PLACED ANY ORDER IN THAT
CURRENT MONTH.

```
SELECT * FROM CUSTOMERS
WHERE CUSTNO NOT IN ( SELECT  CUSTNO FROM ORDERS WHERE
TO_CHAR(ORDDATE,'MMYY') =
                            TO_CHAR(SYSDATE,'MMYY'));
```

DISPLAY DETAILS OF ITEMS FOR WHICH THERE WAS NO ORDER IN THE CURRENT
MONTH BUT THERE WAS AN ORDER IN THE PREVIOUS MONTH.

```
SELECT  *  FROM ITEMS
WHERE  ITEMNO IN ( SELECT ITEMNO FROM LINEITEMS L, ORDERS O
         WHERE  L.ORDNO = O.ORDNO  AND
         TO_CHAR( ADD_MONTHS(SYSDATE,-1),'MMYY') =
TO_CHAR(ORDDATE,'MMYY'))
AND  ITEMNO NOT IN (SELECT ITEMNO FROM LINEITEMS L, ORDERS O
         WHERE  L.ORDNO = O.ORDNO  AND
         TO_CHAR(SYSDATE,'MMYY') = TO_CHAR(ORDDATE,'MMYY'));
```

DISPLAY DETAILS OF ITEMS THAT WERE PURCHASED BY CUSTOMER WHO HAS
PLACED MORE THAN 3 ORDERS.

```
SELECT * FROM ITEMS
WHERE ITEMNO IN ( SELECT ITEMNO FROM LINEITEMS
        WHERE  ORDNO IN ( SELECT  ORDNO FROM ORDERS
               WHERE  CUSTNO IN (
                        SELECT  CUSTNO
                        FROM ORDERS
                                         GROUP BY  CUSTNO
                                         HAVING COUNT(*) > 1
                             )
                   )
        );
```

DISPLAY THE ORDERS IN WHICH THE GAP BETWEEN SHIPDATE AND ORDERDATE IS MORE THAN THE AVERAGE GAP FOR INDIVIDUAL CUSTOMERS.

```
SELECT  * FROM ORDERS  O
WHERE  SHIPDATE - ORDDATE >
    (SELECT  AVG(SHIPDATE - ORDDATE)
     FROM ORDERS
     WHERE CUSTNO = O.CUSTNO);
```

DISPLAY THE DETAILS OF ITEMS IN WHICH THE CURRENT PRICE IS MORE THAN THE MAXIMUM PRICE AT WHICH WE SOLD IT.

```
SELECT * FROM ITEMS I
WHERE  RATE >
   ( SELECT  MAX(PRICE)
     FROM LINEITEMS
     WHERE  ITEMNO = I.ITEMNO);
```

CREATE A NEW TABLE 'COMPORDERS' WITH ORDNO, CUSTOMER-NAME,ORDERDATE,SHIPDATE,DIFFERENCE BETWEEN SHIPDATE AND ORDERDATE.

```
CREATE TABLE COMPORDERS AS SELECT  ORDNO, CUSTNAME,ORDDATE,
SHIPDATE, SHIPDATE-ORDDATE "NODAYS"
FROM  ORDERS O, CUSTOMERS C
WHERE  O.CUSTNO= C.CUSTNO AND SHIPDATE IS NOT NULL;
```

DISPLAY THE ITEMS THAT HAVE TOP 3 HIGHEST PRICES.

```
SELECT  * FROM ITEMS I
WHERE  2 >= ( SELECT COUNT(*) FROM ITEMS WHERE RATE > I.RATE)
ORDER BY RATE DESC;
```

DISPLAY DETAILS OF ITEM THAT HAS SECOND LOWEST PRICE.

```
SELECT  * FROM ITEMS I
WHERE  1 = ( SELECT  COUNT(*) FROM ITEMS WHERE RATE < I.RATE)
```

<
ADD A NEW ITEM TO THE LAST ORDER PLACED BY CUSTOMER 106 WITH THE FOLLOWING DETAILS- ITEMNO-3,QUANTITY-2,PRICE AS THE CURRENT RATE OF THE ITEM,DISCOUNT-8%.

146

```
DECLARE
  V_ORDNO  ORDERS.ORDNO%TYPE;
  V_RATE   ITEMS.RATE%TYPE;

BEGIN
  SELECT  MAX(ORDNO) INTO V_ORDNO
  FROM  ORDERS WHERE CUSTNO = 106;

  SELECT RATE INTO V_RATE
  FROM   ITEMS WHERE  ITEMNO = 3;

  INSERT INTO LINEITEMS VALUES (V_ORDNO,3,2,V_RATE,8);

END;
```

CHANGE RATE OF ITEM 5 TO EITHER AVERAGE RATE OF ITEM 5 OR CURRENT RATE
WHICHEVER IS HIGHER.

```
DECLARE
  V_APRICE LINEITEMS.PRICE%TYPE;
  V_RATE   ITEMS.RATE%TYPE;

BEGIN
  SELECT  AVG(PRICE) INTO V_APRICE
  FROM   LINEITEMS WHERE ITEMNO = 5;

  SELECT  RATE INTO V_RATE
  FROM  ITEMS WHERE ITEMNO  = 5;

  UPDATE  ITEMS SET RATE = GREATEST( V_APRICE, V_RATE)
  WHERE ITEMNO = 5;
END;
```

INSERT A NEW ROW INTO LINEITEMS WITH THE FOLLOWING DETAILS. ORDERNO IS
THE LAST ORDER PLACED BY CUSTOMERNO 102,ITEMNO IS THE ITEM OF P3
PROCESSOR, RATE IS LOWEST RATE OF THAT ITEM,QUANTITY IS 2,DISCOUNT IS 10%
IF ITEM'S CURRENT RATE IS MORE THAN THE LEAST RATE OTHERWISE NO
DISCOUNT.

```
DECLARE
  V_ORDNO  ORDERS.ORDNO%TYPE;
  V_PRICE  LINEITEMS.PRICE%TYPE;
```

147

```
    V_DIS    NUMBER(2);
    V_RATE   ITEMS.RATE%TYPE;
    V_ITEMNO ITEMS.ITEMNO%TYPE;

BEGIN
  SELECT  MAX(ORDNO) INTO V_ORDNO
  FROM    ORDERS WHERE CUSTNO = 102;

  SELECT ITEMNO, RATE INTO V_ITEMNO , V_RATE
  FROM  ITEMS WHERE  UPPER(ITEMNAME) = 'PIII PROCESSOR';

  -- GET LOWEST RATE OF THE ITEM

  SELECT MIN(PRICE) INTO V_PRICE
  FROM   LINEITEMS
  WHERE  ITEMNO = V_ITEMNO;

  IF  V_RATE > V_PRICE THEN
      V_DIS := 10;
  ELSE
      V_DIS := 0;
  END IF;

  INSERT INTO LINEITEMS VALUES  ( V_ORDNO, V_ITEMNO, 2, V_PRICE, V_DIS);

END;


DISPLAY THE HIGHEST OF THE MISSING ORDERNOS.

DECLARE

   V_MAXORDNO ORDERS.ORDNO%TYPE;
   V_MINORDNO ORDERS.ORDNO%TYPE;
   V_CNT    NUMBER(2);

BEGIN
  SELECT  MAX(ORDNO), MIN(ORDNO) INTO  V_MAXORDNO, V_MINORDNO
  FROM    ORDERS;

  FOR I IN REVERSE V_MINORDNO..V_MAXORDNO
  LOOP
    SELECT COUNT(*) INTO V_CNT
    FROM   ORDERS WHERE  ORDNO = I;
```

```
    IF  V_CNT = 0 THEN
       DBMS_OUTPUT.PUT_LINE(I);
       EXIT;
     END IF;
  END LOOP;

END;
```

DISPLAY CUSTOMER NAMES OF THE CUSTOMERS WHO HAVE PLACED MORE THAN 3
ORDERS WHERE THE TOTAL AMOUNT OF THE ORDER IS MORE THAN 10,000.

```
 SELECT  CUSTNAME
 FROM  CUSTOMERS
 WHERE CUSTNO IN ( SELECT  CUSTNO
          FROM   ORDERS
          WHERE  ORDNO IN ( SELECT ORDNO
                 FROM   LINEITEMS
                 GROUP BY ORDNO
                            HAVING  SUM(QTY*PRICE) > 10000)
          GROUP BY  CUSTNO
          HAVING  COUNT(*) > 1 );
```

CHANGE THE RATE OF EACH ITEM AS FOLLOWS (1) INCREASE THE RATE BY 10% IF
THE ITEM WAS SOLD IN MORE THAN 5 ITEMS. (2) INCREASE THE RATE BY 2% IF
AVERAGE PRICE IS GREATER THAN CURRENT PRICE, OTHERWISE DECREASE THE
PRICE BY 3%.

```
DECLARE
  CURSOR  CITEMS IS
   SELECT  ITEMNO,COUNT(*) CNT, AVG(PRICE) APRICE  FROM LINEITEMS
   GROUP BY ITEMNO;

  V_PER NUMBER(5,2);
  V_RATE ITEMS.RATE%TYPE;

BEGIN
  FOR REC  IN  CITEMS
  LOOP


    IF  REC.CNT > 5 THEN
       V_PER := 0.90;
    ELSE
```

```
       -- GET CURRENT RATE
       SELECT  RATE INTO V_RATE
       FROM  ITEMS WHERE ITEMNO =  REC.ITEMNO;

       IF REC.APRICE > V_RATE THEN
          V_PER := 1.02;
       ELSE
          V_PER := 0.97;
       END IF;
     END IF;

     UPDATE ITEMS SET RATE = RATE * V_PER
     WHERE ITEMNO = REC.ITEMNO;

  END LOOP;

END;
```

CREATE A NEW TABLE CALLED CUSTSUM AND STORE THE FOLLOWING DATA INTO
THE TABLE - CUSTOMERNO,CUSTOMER NAME,NO.OF ORDERS PLACED, DATE OF
MOST RECENT ORDER AND TOTAL AMOUNT OF ALL THE ORDERS.

BEFORE THIS PROGRAM IS RUN, YOU HAVE TO CREATE TABLE AS FOLLOWS:

```
CREATE TABLE CUSTSUM
(  CUSTNO NUMBER(5),
   CUSTNAME VARCHAR2(20),
   NOORD    NUMBER(5),
   RORDDATE DATE,
   TOTAMT   NUMBER(10)
);


DECLARE
  CURSOR CUSTCUR IS
     SELECT  CUSTNO, CUSTNAME FROM CUSTOMERS;
  V_ORDCNT NUMBER(5);
  V_MORDDATE DATE;
  V_TOTAMT  NUMBER(10);


BEGIN
```

```
    FOR REC IN  CUSTCUR
    LOOP
       -- GET DETAILS OF CUSTOMER
       SELECT COUNT(*), MAX(ORDDATE), SUM(QTY*PRICE) INTO V_ORDCNT,
V_MORDDATE, V_TOTAMT
       FROM  ORDERS O, LINEITEMS L
       WHERE  O.ORDNO = L.ORDNO AND  CUSTNO = REC.CUSTNO;

       INSERT INTO CUSTSUM VALUES ( REC.CUSTNO, REC.CUSTNAME, V_ORDCNT,
V_MORDDATE,V_TOTAMT);
     END LOOP;

END;
```

DISPLAY ITEMNAMES OF ITEMS FOR WHICH THE CURRENT PRICE IS LESS THAN THE
AVERAGE PRICE OR TOTAL QUANTITY SOLD IS LESS THAN 10 UNITS.

```
SELECT ITEMNAME
FROM  ITEMS
WHERE ITEMNO  IN
  ( SELECT  ITEMNO
    FROM ITEMS I
    WHERE  RATE < ( SELECT AVG(PRICE) FROM LINEITEMS WHERE ITEMNO =
I.ITEMNO)
  )
OR  ITEMNO IN
  ( SELECT ITEMNO
    FROM LINEITEMS
    GROUP BY ITEMNO
    HAVING SUM(QTY) > 10 );
```

CREATE A PROCEDURE THAT TAKES ORDERNO,ITEMNO AND INSERTS A ROW INTO
LINEITEMS, PRICE-RATE OF THE ITEM, QTY-1,DISCOUNT-10%.

```
CREATE OR REPLACE PROCEDURE NEWITEM(P_ORDNO NUMBER, P_ITEMNO
NUMBER)
AS
  V_RATE ORDERS.ORDNO%TYPE;
BEGIN
  SELECT RATE INTO  V_RATE
  FROM  ITEMS WHERE ITEMNO = P_ITEMNO;
```

```
    INSERT INTO LINEITEMS VALUES ( P_ORDNO, P_ITEMNO, V_RATE, 1, 10);

END;


CREATE A FUNCTION THAT RETURNSTHE FIRST MISSING ORDERNO.


CREATE OR REPLACE FUNCTION  FIRSTMISORDNO RETURN NUMBER
AS
   V_MAXORDNO ORDERS.ORDNO%TYPE;
   V_MINORDNO ORDERS.ORDNO%TYPE;
   V_CNT      NUMBER(2);

BEGIN
   SELECT  MAX(ORDNO), MIN(ORDNO) INTO  V_MAXORDNO, V_MINORDNO
   FROM    ORDERS;

   FOR I IN V_MINORDNO..V_MAXORDNO
   LOOP
      SELECT COUNT(*) INTO V_CNT
      FROM   ORDERS WHERE  ORDNO = I;

      IF  V_CNT = 0 THEN
        RETURN  I;
      END IF;
   END LOOP;

   -- NO MISSING ORDNO
   RETURN NULL;
END;


CREATE A FUNCTION THAT TAKES ORDERNO AND RETURNS CUSTOMER NAME OF
THAT ORDER.

CREATE OR REPLACE FUNCTION  GETCUSTNAME ( P_ORDNO NUMBER) RETURN
VARCHAR2
IS
   V_CUSTNAME VARCHAR2(30);
BEGIN
    SELECT  CUSTNAME  INTO V_CUSTNAME
    FROM CUSTOMERS
    WHERE CUSTNO = ( SELECT CUSTNO FROM ORDERS WHERE ORDNO = P_ORDNO);
```

```
    RETURN  V_CUSTNAME;
END;


CREATE A PROCEDURE THAT INSERTS A NEW ROW INTO LINEITEMS WITH GIVEN
ITEMNO,PRICE,QUANTITY, ORDERNO IS THE MOST RECENT ORDER. CHECK
WHETHER PRICE IS MORE THAN THE CURRENT RATE OF THE ITEM, CHECK
WHETHER ITEM IS ALREADY EXISTING IN THE ORDER AND CHECK WHETHER THE
TOTAL AMOUNT OF THE ORDER INCLUDING THE NEW ITEM HAS EXCEEDED 50,000.


CREATE OR REPLACE PROCEDURE  NEWITEMS(P_ITEMNO NUMBER, P_PRICE
NUMBER, P_QTY NUMBER)
IS
  V_CNT NUMBER(2);
  V_RATE ITEMS.RATE%TYPE;
  V_TOTAMT NUMBER(10);
  V_ORDNO ORDERS.ORDNO%TYPE;
BEGIN
  SELECT MAX(ORDNO) INTO V_ORDNO FROM ORDERS;

  -- CHECK CONDITIONS

  SELECT RATE INTO V_RATE  FROM ITEMS WHERE ITEMNO = P_ITEMNO;

  IF P_PRICE > V_RATE THEN
     RAISE_APPLICATION_ERROR(-20001,'PRICE IS MORE THAN CURRENT PRICE');
  END IF;

  SELECT COUNT(*) INTO V_CNT
  FROM  LINEITEMS
  WHERE  ORDNO = V_ORDNO AND ITEMNO = P_ITEMNO;

  IF  V_CNT = 1 THEN
     RAISE_APPLICATION_ERROR(-20002,'ITEM IS ALREADY EXISTING');
  END IF;


  -- GET TOTAL AMOUNT

  SELECT  SUM(QTY * PRICE) INTO V_TOTAMT
  FROM LINEITEMS WHERE ORDNO = V_ORDNO;

  IF   V_TOTAMT + P_PRICE * P_QTY > 50000 THEN
```

```
    RAISE_APPLICATION_ERROR(-20003,'TOTAL AMOUNT EXCEEDED 50000');
  END IF;

  INSERT INTO LINEITEMS VALUES (V_ORDNO, P_ITEMNO, P_PRICE,P_QTY,0);

END;
```

MAKE SURE AN ORDER IS NOT CONTAINING MORE THAN 5 ITEMS.

```
CREATE OR REPLACE TRIGGER  CHECKITEMCOUNT
BEFORE INSERT
ON  LINEITEMS
FOR EACH ROW
DECLARE
  V_CNT NUMBER(5);
BEGIN

  SELECT COUNT(*) INTO V_CNT
  FROM  LINEITEMS WHERE  ORDNO = :NEW.ORDNO;

  IF  V_CNT >= 5  THEN
     RAISE_APPLICATION_ERROR(-20010,'CANNOT HAVE MORE THAN 5 ITEMS IN AN
ORDER');
  END IF;
END;
```

DO NOT ALLOW ANY CHANGES TO ITEMS TABLE AFTER 9PM BEFORE 9AM.

```
CREATE OR REPLACE TRIGGER CHECKTIME
BEFORE INSERT OR DELETE OR UPDATE
ON  ITEMS
BEGIN
   IF  TO_CHAR(SYSDATE,'HH24') < 9  OR  TO_CHAR(SYSDATE,'HH24') > 21 THEN
      RAISE_APPLICATION_ERROR(-200011,'NO CHANGES CAN BE MADE BEFORE 9
A.M AND AFTER 9 P.M');
   END IF;
END;
```

DO NOT ALLOW ANY CHANGE TO ITEM RATE IN SUCH A WAY DIFFERENCE IS MORE
THAN 25% OF THE EXISTING RATE.

CREATE OR REPLACE TRIGGER TRGDIFFRATE

```
BEFORE UPDATE
ON ITEMS
FOR EACH ROW
DECLARE
   V_DIFF NUMBER(5);
BEGIN
     V_DIFF := ABS(:NEW.RATE - :OLD.RATE);

   IF  V_DIFF > :OLD.RATE * 0.25 THEN
       RAISE_APPLICATION_ERROR(-20014,'INVALID RATE FOR AMOUNT. CHANGE IS
TOO BIG');
     END IF;

END;
```

DATABASE DEVELOPMENT LIFE CYCLE (DDLC)

The database development life cycle (DDLC) is a process of designing, implementing and maintaining a database system to meet strategic or operational information needs of an organization or enterprise such as:

- Improved customer support and customer satisfaction.
- Better production management.
- Better inventory management.
- More accurate sales forecasting.

PHASES OF DDLC

The software development is the group of actions needed to transform the user's need into an effectual software solution. Software development procedure consist the activities needed for building the software systems and integrating the techniques and practices to be accepted. It also includes the planning of project, tracking development and managing the complications of building software.

This different database related activities can be grouped into below phases (more commonly known as DDLC – Database Development Life Cycle):

Requirements Analysis Database Design Evaluation and Selection Logical Database Design Physical Database Design Implementation Data Loading Testing and Performance Tuning Operation Maintenance

- **Requirements Analysis**

The most important step n implementing a database system is to find out what is needed — What type of a database is required for the business organization, daily volume of the data, how much data needs to be stored in the master files etc. In order to collect all this required information, a database analyst need to spend a lot of time within the business organization talking to people, end users and get acquainted with day-to-day process.

- **Database Design**

In this stage the database designers will make a decision on the database model that is perfectly suited for the organization's requirements. The database designers will study the documents prepared by the analysts in the requirements analysis stage and then start developing a system that fulfills the needs.

- **Evaluation and Selection**

Once the data model is designed, tested and demonstrated, the next phase is to evaluate the diverse database management systems and choose the one that is perfectly suited for the requirements of the organization. In order to identify best performing database for the organization, end user should be involved in this phase.

- **Logical Database Design**

Once the evaluation and selection phase is completed successfully, the next step n the database development life cycle is logical database design. The conceptual design is translated into internal model in the logical design phase. This includes the mapping of all objects i.e. tables design, indexes, views, transactions, access privileges etc.

- **Physical Database Design**

Physical database design is the procedure of selecting and characterizing the data storage and data access of the database. The data storage depends on the type of devices supported by the hardware, the data access methods and the DBMS.

Physical design is mainly significant for older database models like hierarchical and network models. Physical design is very vital in database development life cycle and has great significance as a bad design can result in deprived performance.

- **Implementation**

In most databases a new database implementation needs the formation of special storage related constructs to house the end user tables. These constructs typically comprise storage group, table spaces, data files, tables etc.

- **Data Loading**

Once the database has been created, the data must be loaded into the database. The data required to be converting and migrating to the new database, if the loaded data is currently stored n a different system or in a different format.

- **Testing and Performance Tuning**

The next phase is testing and performance tuning, this phase starts soon the data is loaded into the database. In this phase, database is tested and fine-tuned for performance, integrity, access and security constraints. It is very important that the database administrators and application programmers work together during this phase, because testing and performance tuning happens in parallel.

- **Operation**

Once the data is loaded into the database and it s fully tested, the database is than released into production.

In operation phase, the database is accessed by the end users and application programs. This stage includes adding of new data, modifying existing data and deletion of obsolete data. The database administrators perform the administrative tasks periodically such as performance tuning, expanding storage space, database backup etc. This is the crucial phase as it provides useful information and helps management to make a business decision, thus making the smooth and well-organized functioning of the organization.

- **Maintenance**

Database maintenance phase is very important and it is one of the ongoing phases in DDLC. Factors such as new business needs, new information requirements, acquisition of new data etc will make it essential to formulate ongoing changes and improvements to the existing design. The major tasks in this phase include: database backup and recovery, performance tuning, design modifications, access management and audits, usage monitoring, hardware maintenance, upgradation etc.

Functional Dependencies
FD's are constraints on well-formed relations and represent a formalism on the infrastructure of relation.

Definition: A *functional dependency* (FD) on a relation schema **R** is a <u>constraint</u> *X → Y*, where *X* and *Y* are subsets of attributes of **R.** An FD is a relationship between an attribute "Y" and a determinant (1 or more other attributes) "X" such that for a given value of a determinant the value of the attribute is uniquely defined.

- X is a determinant
- X determines Y
- Y is functionally dependent on X
- X → Y
- X →Y is trivial if Y Í X

Example:

Let R be
NewStudent(*stuId, lastName, major, credits, status, socSecNo*)

FDs in R include

- *{stuId}→{lastName}*, but not the reverse
- *{stuId} →{lastName, major, credits, status, socSecNo, stuId}*
- *{socSecNo} →{stuId, lastName, major, credits, status, socSecNo}*
- *{credits}→{status},* but not *{status}→{credits}*

ZipCode→AddressCity

ArtistName→BirthYear

Autobrand→Manufacturer, Engine type

Author, Title→PublDate

TRIVIAL FUNCTIONAL DEPENDENCY

A functional dependency is trivial if Y is a subset of X. In a table with attributes of employee name and Social Security number (SSN), employee name is functionally dependant on SSN because the SSN is unique for individual names. An SSN identifies the employee specifically, but an employee name cannot distinguish the SSN because more than one employee could have the same name. Functional dependency defines Boyce-Codd normal form and third normal form. This preserves dependency between attributes, eliminating the repetition of information. Functional dependency is related to a candidate key, which uniquely identifies a tuple and determines the value of all other attributes in the relation. In some cases, functionally dependant sets are irreducible if: The right-hand

set of functional dependency holds only one attribute. The left-hand set of functional dependency cannot be reduced, since this may change the entire content of the set. Reducing any of the existing functional dependency might change the content of the set. An important property of a functional dependency is Armstrong's axiom, which is used in database normalization. In a relation, R, with three attributes (X, Y, Z) Armstrong's axiom holds strong if the following conditions are satisfied: Axiom of Transivity: If X->Y and Y->Z then, X->Z. Axiom of Reflexivity (Subset Property): If Y is a subset of X then X->Y. Axiom of Augmentation: If X->Y then XZ->YZ.

Dependency Preservation

1. Another desirable property in database design is **dependency preservation**.
   o We would like to check easily that updates to the database do not result in illegal relations being created.
   o It would be nice if our design allowed us to check updates without having to compute natural joins.
   o To know whether joins must be computed, we need to determine what functional dependencies may be tested by checking each relation individually.
   o Let *F* be a set of functional dependencies on schema *R*.
   o Let $\{R_1, R_2, \ldots, R_n\}$ be a decomposition of *R*.
   o The **restriction** of *F* to R_i is the set of all functional dependencies in F^+ that include only attributes of R_i.
   o Functional dependencies in a restriction can be tested in one relation, as they involve attributes in one relation schema.
   o The set of restrictions $F_1, F_2, \ldots, F_n$ is the set of dependencies that can be checked efficiently.
   o We need to know whether testing only the restrictions is sufficient.
   o Let $F' = F_1, F_2, \ldots, F_n$.
   o *F*' is a set of functional dependencies on schema *R*, but in general, $F' \neq F$.
   o However, it may be that $F'^+ = F^+$.
   o If this is so, then every functional dependency in *F* is implied by *F*', and if *F*' is satisfied, then *F* must also be satisfied.
   o A decomposition having the property that $F'^+ = F^+$ is a **dependency-preserving** decomposition.

MULTIVALUED DEPENDENCIES

Multivalued dependencies occur when the presence of one or more rows in a table implies the presence of one or more other rows in that same table.

Examples:

For example, imagine a car company that manufactures many models of car, but always makes both red and blue colors of each model. If you have a table that contains the model name, color and year of each car the company manufactures, there is a multivalued dependency in that table. If there is a row for a certain model name and year in blue, there must also be a similar row corresponding to the red version of that same car.

NORMALIZATION

Normalization is a process of reducing redundancies of data in a database. Quite often we come across tables having a lot of bulk data with many columns. All these data might not be necessary all the time whenever we use those tables. So, a better option is to split up the bulk table into small parts and use only those tables which suit the actual purpose at a given instance of time. In this way, redundancy is reduced. To make the long story short, we can simply say that normalization is a process of dividing a big table into smaller ones in order to reduce redundancy.

ANAMOLIES IN DBMS:

Insertion Anomaly
It is a failure to place information about a new database entry into all the places in the database where information about the new entry needs to be stored. In a properly normalized database, information about a new entry needs to be inserted into only one place in the database, in an inadequatly normalized database, information about a new entry may need to be inserted into more than one place, and human fallibility being what it is, some of the needed additional insertions may be missed.

Deletion anomaly
It is a failure to remove information about an existing database entry when it is time to remove that entry. In a properly normalized database, information about an old, to-be-gotten-rid-of entry needs to be deleted from only one place in the database, in an inadequatly normalized database, information about that old entry may need to be deleted from more than one place.

Update Anomaly
An update of a database involves modifications that may be additions, deletions, or both. Thus "update anomalies" can be either of the kinds discussed above.

All three kinds of anomalies are highly undesirable, since thier occurence constitutes corruption of the database. Properly normalized database are much less susceptible to corruption than are un-normalized databases.

Normalization Avoids

- Duplication of Data – The same data is listed in multiple lines of the database
- Insert Anomaly – A record about an entity cannot be inserted into the table without first inserting information about another entity – Cannot enter a customer without a sales order

- Delete Anomaly – A record cannot be deleted without deleting a record about a related entity. Cannot delete a sales order without deleting all of the customer's information.
- Update Anomaly – Cannot update information without changing information in many places. To update customer information, it must be updated for each sales order the customer has placed

Process of normalization:

Before getting to know the normalization techniques in detail, let us define a few building blocks which are used to define normal form.

1. **Determinant** : Attribute X can be defined as determinant if it uniquely defines the value Y in a given relationship or entity .To qualify as determinant attribute need NOT be a key attribute .Usually dependency of attribute is represented as X->Y ,which means attribute X decides attribute Y.

Example: In RESULT relation, Marks attribute may decide the grade attribute .This is represented as Marks->grade and read as Marks decides Grade.

Marks -> Grade

In the result relation, Marks attribute is not a key attribute .Hence it can be concluded that key attributes are determinants but not all the determinants are key attributes.

2. **Functional Dependency**: Yes functional dependency has definition but let's not care about that. Let's try to understand the concept by example. Consider the following relation :

REPORT(Student#,Course#,CourseName,IName,Room#,Marks,Grade)
Where:

- Student#-Student Number
- Course#-Course Number
- CourseName -CourseName
- IName- Name of the instructor who delivered the course
- Room#-Room number which is assigned to respective instructor
- Marks- Scored in Course Course# by student Student #
- Grade –Obtained by student Student# in course Course #
- Student#,Course# together (called composite attribute) defines EXACTLY ONE value of marks .This can be symbolically represented as

Student#Course# Marks

This type of dependency is called **functional dependency**. In above example Marks is functionally dependent on Student#Course#.

Other Functional dependencies in above examples are:

- Course# -> CourseName
- Course#-> IName(Assuming one course is taught by one and only one instructor)
- IName -> Room# (Assuming each instructor has his /her own and non shared room)
- Marks ->Grade

161

Formally we can define functional dependency as: In a given relation R, X and Y are attributes. Attribute Y is functional dependent on attribute X if each value of X determines exactly one value of Y. This is represented as :

X->Y

However X may be composite in nature.

3. **Full functional dependency:** In above example Marks is fully functional dependent on *student#Course#* and not on the sub set of *Student#Course#* .This means marks cannot be determined either by student # or Course# alone .It can be determined by using Student# and Course# together. Hence Marks is fully functional dependent on student#course#.

CourseName is not fully functionally dependent on student#course# because one of the subset course# determines the course name and Student# does not having role in deciding Course name .Hence CourseName is not fully functional dependent on student #Course#.

Student#

Marks

Course#

Formal Definition of full functional dependency : In a given relation R ,X and Y are attributes. Y is fully functionally dependent on attribute X only if it is not functionally dependent on sub-set of X.However X may be composite in nature.

4. **Partial Dependency**: In the above relationship CourseName,IName,Room# are partially dependent on composite attribute Student#Course# because Course# alone can defines the coursename, IName,Room#.

Room#
IName
CourseName
Course#
Student#

Formal Definition of Partial dependency: In a given relation R, X and Y are attributes .Attribute Y is partially dependent on the attribute X only if it is dependent on subset attribute X .However X may be composite in nature.

5. **Transitive Dependency**: In above example , Room# depends on IName and in turn depends on Course# .Here Room# transitively depends on Course#.

IName
Room#
Course#

Similarly Grade depends on Marks,in turn Marks depends on Student# Course# hence Grade

Fully transitively depends on Student# Course#.

6. **Key attributes :** In a given relationship R ,if the attribute X uniquely defines all other attributes ,then the attribute X is a key attribute which is nothing but the candidate key.

Ex: Student#Course# together is a composite key attribute which determines all attributes in relationship REPORT(student#,Course#,CourseName,IName,Room#,Marks,Grade)uniquely.Hence Student# and Course# are key attributes.

Un-Normalized Form (UNF)

If a table contains non-atomic values at each row, it is said to be in UNF. An **atomic value** is something that can not be further decomposed. A **non-atomic value**, as the name suggests, can be further decomposed and simplified. Consider the following table:

Emp-Id	Emp-Name	Month	Sales	Bank-Id	Bank-Name
E01	AA	Jan	1000	B01	SBI
		Feb	1200		
		Mar	850		
E02	BB	Jan	2200	B02	UTI
		Feb	2500		
E03	CC	Jan	1700	B01	SBI
		Feb	1800		
		Mar	1850		
		Apr	1725		

In the sample table above, there are multiple occurrences of rows under each key Emp-Id. Although considered to be the primary key, Emp-Id cannot give us the unique identification facility for any single row. Further, each primary key points to a variable length record (3 for E01, 2 for E02 and 4 for E03).

First Normal Form (1NF)

A relation is said to be in 1NF if it contains no non-atomic values and each row can provide a unique combination of values. The above table in UNF can be processed to create the following table in 1NF.

Emp-Id	Emp-Name	Month	Sales	Bank-Id	Bank-Name
E01	AA	Jan	1000	B01	SBI

E01	AA	Feb	1200	B01	SBI
E01	AA	Mar	850	B01	SBI
E02	BB	Jan	2200	B02	UTI
E02	BB	Feb	2500	B02	UTI
E03	CC	Jan	1700	B01	SBI
E03	CC	Feb	1800	B01	SBI
E03	CC	Mar	1850	B01	SBI
E03	CC	Apr	1725	B01	SBI

As you can see now, each row contains unique combination of values. Unlike in UNF, this relation contains only atomic values, i.e. the rows can not be further decomposed, so the relation is now in 1NF.

Second Normal Form (2NF)

A relation is said to be in 2NF f if it is already in 1NF and each and every attribute fully depends on the primary key of the relation. Speaking inversely, if a table has some attributes which is not dependant on the primary key of that table, then it is not in 2NF.

Let us explain. Emp-Id is the primary key of the above relation. Emp-Name, Month, Sales and Bank-Name all depend upon Emp-Id. But the attribute Bank-Name depends on Bank-Id, which is not the primary key of the table. So the table is in 1NF, but not in 2NF. If this position can be removed into another related relation, it would come to 2NF.

Emp-Id	Emp-Name	Month	Sales	Bank-Id
E01	AA	JAN	1000	B01
E01	AA	FEB	1200	B01
E01	AA	MAR	850	B01
E02	BB	JAN	2200	B02
E02	BB	FEB	2500	B02
E03	CC	JAN	1700	B01
E03	CC	FEB	1800	B01
E03	CC	MAR	1850	B01
E03	CC	APR	1726	B01

Bank-Id	Bank-Name
B01	SBI
B02	UTI

After removing the portion into another relation we store lesser amount of data in two relations without any loss information. There is also a significant reduction in redundancy.

Third Normal Form (3NF)

A relation is said to be in 3NF, if it is already in 2NF and there exists no **transitive dependency** in that relation. Speaking inversely, if a table contains transitive dependency, then it is not in 3NF, and the table must be split to bring it into 3NF.

What is a transitive dependency? Within a relation if we see
A → B [B depends on A]
And
B → C [C depends on B]
Then we may derive
A → C[C depends on A]

Such derived dependencies hold well in most of the situations. For example if we have
Roll → Marks
And
Marks → Grade
Then we may safely derive
Roll → Grade.

This third dependency was not originally specified but we have derived it.

The derived dependency is called a transitive dependency when such dependency becomes improbable. For example we have been given
Roll → City
And
City → STDCode

If we try to derive Roll → STDCode it becomes a transitive dependency, because obviously the STDCode of a city cannot depend on the roll number issued by a school or college. In such a case the relation should be broken into two, each containing one of these two dependencies:
Roll → City
And
City → STD code

Boyce-Code Normal Form (BCNF)

A relationship is said to be in BCNF if it is already in 3NF and the left hand side of every dependency is a candidate key. A relation which is in 3NF is almost always in BCNF. These could be same situation when a 3NF relation may not be in BCNF the following conditions are found true.

1. The candidate keys are composite.
2. There are more than one candidate keys in the relation.
3. There are some common attributes in the relation.

Professor Code	Department	Head of Dept.	Percent Time
P1	Physics	Ghosh	50
P1	Mathematics	Krishnan	50
P2	Chemistry	Rao	25
P2	Physics	Ghosh	75
P3	Mathematics	Krishnan	100

Consider, as an example, the above relation. It is assumed that:

1. A professor can work in more than one department
2. The percentage of the time he spends in each department is given.
3. Each department has only one Head of Department.

The relation diagram for the above relation is given as the following:
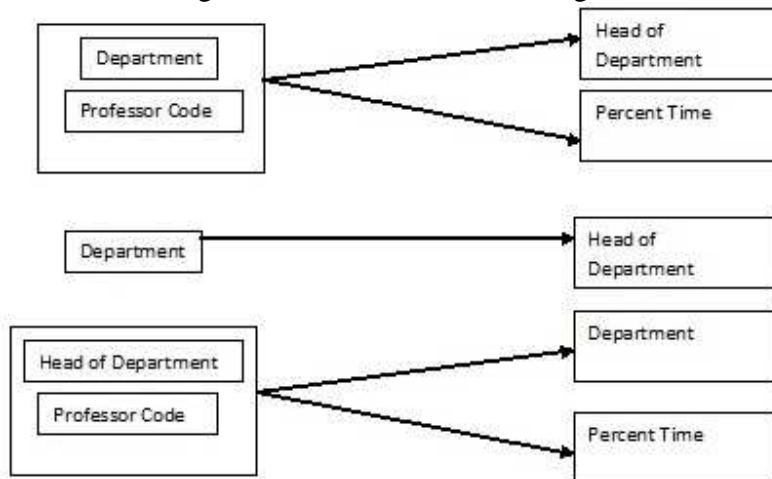
The given relation is in 3NF. Observe, however, that the names of Dept. and Head of Dept. are duplicated. Further, if Professor P2 resigns, rows 3 and 4 are deleted. We lose the information that Rao is the Head of Department of Chemistry.

The normalization of the relation is done by creating a new relation for Dept. and Head of Dept. and deleting Head of Dept. form the given relation. The normalized relations are shown in the following.

Professor Code	Department	Percent Time
P1	Physics	50
P1	Mathematics	50
P2	Chemistry	25

P2	Physics	75
P3	Mathematics	100

Department	Head of Dept.
Physics	Ghosh
Mathematics	Krishnan
Chemistry	Rao

See the dependency diagrams for these new relations.
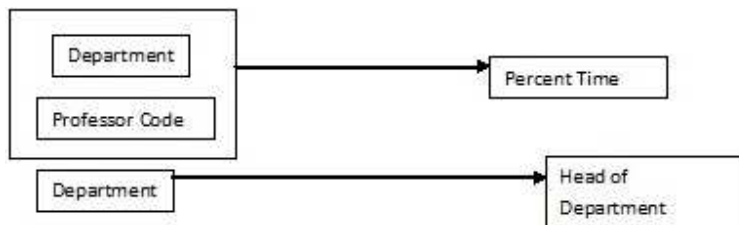
Fourth Normal Form (4NF)

When attributes in a relation have multi-valued dependency, further Normalization to 4NF and 5NF are required. Let us first find out what multi-valued dependency is.

A **multi-valued dependency** is a typical kind of dependency in which each and every attribute within a relation depends upon the other, yet none of them is a unique primary key.

We will illustrate this with an example. Consider a vendor supplying many items to many projects in an organization. The following are the assumptions:

1. A vendor is capable of supplying many items.
2. A project uses many items.
3. A vendor supplies to many projects.
4. An item may be supplied by many vendors.

A multi valued dependency exists here because all the attributes depend upon the other and yet none of them is a primary key having unique value.

Vendor Code	Item Code	Project No.
V1	I1	P1
V1	I2	P1
V1	I1	P3

V1	I2	P3
V2	I2	P1
V2	I3	P1
V3	I1	P2
V3	I1	P3

The given relation has a number of problems. For example:

1. If vendor V1 has to supply to project P2, but the item is not yet decided, then a row with a blank for item code has to be introduced.
2. The information about item I1 is stored twice for vendor V3.

Observe that the relation given is in 3NF and also in BCNF. It still has the problem mentioned above. The problem is reduced by expressing this relation as two relations in the Fourth Normal Form (4NF). A relation is in 4NF if it has no more than one independent multi valued dependency or one independent multi valued dependency with a functional dependency.

The table can be expressed as the two 4NF relations given as following. The fact that vendors are capable of supplying certain items and that they are assigned to supply for some projects in independently specified in the 4NF relation.

Vendor-Supply

Vendor Code	Item Code
V1	I1
V1	I2
V2	I2
V2	I3
V3	I1

Vendor-Project

| Vendor Code	Project No.
V1	P1
V1	P3
V2	P1
V3	P2

Fifth Normal Form (5NF)

168

These relations still have a problem. While defining the 4NF we mentioned that all the attributes depend upon each other. While creating the two tables in the 4NF, although we have preserved the dependencies between Vendor Code and Item code in the first table and Vendor Code and Item code in the second table, we have lost the relationship between Item Code and Project No. If there were a primary key then this loss of dependency would not have occurred. In order to revive this relationship we must add a new table like the following. Please note that during the entire process of normalization, this is the only step where a new table is created by joining two attributes, rather than splitting them into separate tables.

Project No.	Item Code
P1	11
P1	12
P2	11
P3	11
P3	13

QUERY PROCESSING

Query processing includes translation of high-level queries into low-level expressions that can be used at the physical level of the file system, query optimization and actual execution of the query to get the result. It is a three-step process that consists of parsing and translation, optimization and execution of the query submitted by the user.

Query-processing Steps
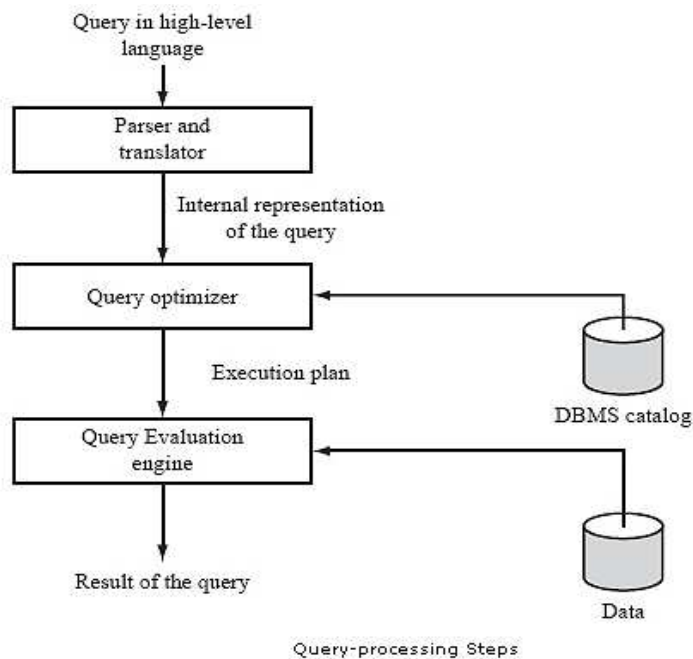
A query is processed in four general steps:

1. Scanning and Parsing
2. Query Optimization or planning the execution strategy
3. Query Code Generator (interpreted or compiled)
4. Execution in the runtime database processor

1. Scanning and Parsing

- When a query is first submitted (via an applications program), it must be scanned and parsed to determine if the query consists of appropriate syntax.
- **Scanning** is the process of converting the query text into a tokenized representation.
- The tokenized representation is more compact and is suitable for processing by the parser.
- This representation may be in a tree form.
- The **Parser** checks the tokenized representation for correct syntax.
- In this stage, checks are made to determine if columns and tables identified in the query exist in the database and if the query has been formed correctly with the appropriate keywords and structure.
- If the query passes the parsing checks, then it is passed on to the Query Optimizer.

2. Query Optimization or Planning the Execution Strategy

- For any given query, there may be a number of different ways to execute it.
- Each operation in the query (SELECT, JOIN, etc.) can be implemented using one or more different *Access Routines*.
- For example, an access routine that employs an index to retrieve some rows would be more efficient that an access routine that performs a full table scan.
- The goal of the **query optimizer** is to find a *reasonably efficient* strategy for executing the query (not quite what the name implies) using the access routines.
- Optimization typically takes one of two forms: *Heuristic Optimization* or *Cost Based Optimization*
- In **Heuristic Optimization**, the query execution is refined based on *heuristic rules* for reordering the individual operations.
- With **Cost Based Optimization**, the overall cost of executing the query is systematically reduced by estimating the costs of executing several different execution plans.

3. Query Code Generator (interpreted or compiled)

- Once the query optimizer has determined the execution plan (the specific ordering of access routines), the code generator writes out the actual access routines to be executed.
- With an interactive session, the query code is interpreted and passed directly to the runtime database processor for execution.
- It is also possible to *compile* the access routines and store them for later execution.

4. Execution in the runtime database processor

- At this point, the query has been scanned, parsed, planned and (possibly) compiled.
- The runtime database processor then executes the access routines against the database.
- The results are returned to the application that made the query in the first place.
- Any runtime errors are also returned.

Query Optimization

- To enable the system to achieve (or improve) acceptable performance by choosing a better (if not the best) strategy during the process of a query. One of the great strengths to the relational database.

Automatic Optimization vs. Human Programmer

1. A good automatic optimizer will have a wealth of information available to it that human programmers typically do not have.
2. An automatic optimizer can easily reprocess the original relational request when the organization of the database is changed. For a human programmer, reorganization would involve rewriting the program.
3. The optimizer is a program, and therefore is capable of considering literally hundreds of different implementation strategies for a given request, which is much more than a human programmer can.
4. The optimizer is available to a wide range of users, in an efficient and cost-effective manner.

The Optimization Process

1. Cast the query into some internal representation, such as a query tree structure.
2. Convert the internal representation to canonical form.

*A subset (say C) of a set of queries (say Q) is said to be a set of canonical forms for Q if and only if every query Q is equivalent to just one query in C.

During this step, some optimization is already achieved by transforming the internal representation to a better canonical form.

Possible improvements

    a. Doing the restrictions (selects) before the join.
    b. Reduce the amount of comparisons by converting a restriction condition to an equivalent condition in **conjunctive normal form**- that is, a condition consisting of a set of restrictions that are ANDed together, where each restriction in turn consists of a set of simple comparisons connected only by OR's.
    c. A sequence of restrictions (selects) before the join.
    d. In a sequence of projections, all but the last can be ignored.
    e. A restriction of projection is equivalent to a projection of a restriction.
    f. Others
3. Choose candidate low-level procedures by evaluate the transformed query.

*Access path selection: Consider the query expression as a series of basic operations (join, restriction, etc.), then the optimizer choose from a set of pre-defined, low-level implementation procedures. These procedures may involve the user of primary key, foreign key or indexes and other information about the database.

4. Generate query plans and choose the cheapest by constructing a set of candidate query plans first, then choose the best plan. To pick the best plan can be achieved by assigning cost to each given plan. The costs is computed according to the number of disk I/O's involved.

MODULE 3

Database security

Database security concerns the use of a broad range of information security controls to protect databases (potentially including the data, the database applications or stored functions, the database systems, the database servers and the associated network links) against compromises of their confidentiality, integrity and availability. It involves various types or categories of controls, such as technical, procedural/administrative and physical. *Database security* is a specialist topic within the broader realms of computer security, information security and risk management.

Security risks to database systems include, for example:

- Unauthorized or unintended activity or misuse by authorized database users, database administrators, or network/systems managers, or by unauthorized users or hackers (e.g. inappropriate access to sensitive data, metadata or functions within databases, or inappropriate changes to the database programs, structures or security configurations);
- Malware infections causing incidents such as unauthorized access, leakage or disclosure of personal or proprietary data, deletion of or damage to the data or programs, interruption or denial of authorized access to the database, attacks on other systems and the unanticipated failure of database services;
- Overloads, performance constraints and capacity issues resulting in the inability of authorized users to use databases as intended;
- Physical damage to database servers caused by computer room fires or floods, overheating, lightning, accidental liquid spills, static discharge, electronic breakdowns/equipment failures and obsolescence;
- Design flaws and programming bugs in databases and the associated programs and systems, creating various security vulnerabilities (e.g. unauthorized privilege escalation), data loss/corruption, performance degradation etc.;
- Data corruption and/or loss caused by the entry of invalid data or commands, mistakes in database or system administration processes, sabotage/criminal damage etc.

Many layers and types of information security control are appropriate to databases, including:

- Access control
- Auditing
- Authentication
- Encryption
- Integrity controls
- Backups
- Application security

Traditionally databases have been largely secured against hackers through network security measures such as firewalls, and network-based intrusion detection systems. While network security controls remain valuable in this regard, securing the database systems themselves, and the programs/functions and data within them, has arguably become more critical as networks are increasingly opened to wider access, in particular access from the Internet. Furthermore, system, program, function and data access controls, along with the associated user identification, authentication and rights management functions, have always been important to limit and in some cases log the activities of authorized users and administrators. In other words, these are complementary approaches to database security, working from both the outside-in and the inside-out as it were.

Many organizations develop their own "baseline" security standards and designs detailing basic security control measures for their database systems. These may reflect general information security requirements or obligations imposed by corporate information security policies and applicable laws and regulations (e.g. concerning privacy, financial management and reporting systems), along with generally-accepted good database security practices (such as appropriate hardening of the underlying systems) and perhaps security recommendations from the relevant database system and software vendors. The security designs for specific database systems typically specify further security administration and management functions (such as administration and reporting of user access rights, log management and analysis, database replication/synchronization and backups) along with various business-driven information security controls within the database programs and functions (e.g. data entry validation and audit trails). Furthermore, various security-related activities (manual controls) are normally incorporated into the procedures, guidelines etc. relating to the design, development, configuration, use, management and maintenance of databases.

Database security cannot be seen as an isolated problem because it is effected by other components of a computerized system as well. The security requirements of a system are specified by means of a security policy which is then enforced by various security mechanisms. For databases, requirements on the security can be classified into the following categories:

Identification, Authentication
Usually before getting access to a database each user has to identify himself to the computer system. Authentication is the way to verify the identity of a user at log-on time. Most common authentication methods are passwords but more advanced techniques like badge readers, biometric recognition techniques, or signature analysis devices are also available.

Authorization, Access Controls
Authorization is the specification of a set of rules that specify who has which type of access to what information. Authorization policies therefore govern the disclosure and modification of information. Access controls are procedures that are designed to control authorizations. They are responsible to limit access to stored data to authorized users only.

Integrity, Consistency
An integrity policy states a set of rules (i. e. semantic integrity constraints) that define the correct states of the database during database operation and therefore can protect against malicious or accidental modification of information. Closely related issues to integrity and consistency are concurrency control and recovery. Concurrency control policies protect the integrity of the database in the presence of concurrent transactions. If these transactions do not terminate normally due to

system crashes or security violations recovery techniques are used to reconstruct correct or valid database states.

☐Auditing

The requirement to keep records of all security relevant actions issued by a user is called auditing. Resulting audit records are the basis for further reviews and examinations in order to test the adequacy of system controls and to recommend any changes in the security policy.

Locking

Locking is a mechanism commonly used by systems to control access to shared resources by concurrently running users. In the context of a DBMS, these shared resources are data objects, and the users are transactions.

Locking is typically implemented using a lock manager, which records which objects are locked, by whom, and in what mode. When a transaction wishes to use a particular

object (to read or write), it must request a lock from the lock manager. After it is done with the object, it releases the lock by again notifying the lock manager. In certain cases, the lock manager is not able to immediately grant a lock when it is requested (e.g., if it is held by another transaction). In this case, the lock manager maintains a queue of transactions waiting for the lock.

It is important to also recognize that some data items can be shared simultaneously between transactions (e.g., transactions T1 and T2 both want to read object X), but in other cases it is necessary for a transaction to have an exclusive lock (e.g., T1 wants to write to X). This motivates the need for multiple lock modes. In this case, the idea is that, if a transaction requests a lock on an object in a mode that is incompatible with an existing lock on that object, then it must wait on the lock queue until the existing lock is released.

In a DBMS, the goal is to develop a locking protocol that guarantees a schedule with desirable properties (e.g., serializability, recoverability, avoid cascading aborts). Two common protocols are two-phase locking (2PL) and strict two-phase locking (Strict 2PL).

Strict Two-Phase Locking (Strict 2PL)

1. If a transaction T wants to read object X, it requests a shared lock on X. If it wants to write X, it requests an exclusive lock.

2. All locks requested by a transaction are held until the transaction is completed (commits or aborts), at which point the locks are released. It can be shown that Strict 2PL guarantees schedules that are serializable, recoverable, and that avoid cascading aborts.

Two-Phase Locking (2PL)

2PL relaxes Strict 2PL slightly. A transaction need not hold all locks until completion, but once it has released a lock, it may not request any more locks. 2PL is guaranteed to produce schedules that are serializable.

Shared and Exclusive Locks

A lock is a system object associated with a shared resource such as a data item of an elementary type, a row in a database, or a page of memory. In a database, a lock on a database object (a data-access lock) may need to be acquired by a transaction before accessing the object. Correct use of locks prevents undesired, incorrect or inconsistent operations on shared resources by other concurrent transactions. When a database object with an existing lock acquired by one transaction needs to be accessed by another transaction, the existing lock for the object and the type of the

intended access are checked by the system. If the existing lock type does not allow this specific attempted concurrent access type, the transaction attempting access is blocked (according to a predefined agreement/scheme). In practice a lock on an object does not directly block a transaction's operation upon the object, but rather blocks that transaction from acquiring another lock on the same object, needed to be held/owned by the transaction before performing this operation. Thus, with a locking mechanism, needed operation blocking is controlled by a proper lock blocking scheme, which indicates which lock type blocks which lock type.

Two major types of locks are utilized:

- **Write-lock** (**exclusive lock**) is associated with a database object by a transaction (Terminology: "the transaction locks the object," or "acquires lock for it") before *writing* (inserting/modifying/deleting) this object.
- **Read-lock** (**shared lock**) is associated with a database object by a transaction before *reading* (retrieving the state of) this object.

The common interactions between these lock types are defined by blocking behavior as follows:

- An existing *write-lock* on a database object blocks an intended *write* upon the same object (already requested/issued) by another transaction by blocking a respective *write-lock* from being acquired by the other transaction. The second write-lock will be acquired and the requested write of the object will take place (materialize) after the existing write-lock is released.
- A *write-lock* blocks an intended (already requested/issued) *read* by another transaction by blocking the respective *read-lock* .
- A *read-lock* blocks an intended *write* by another transaction by blocking the respective *write-lock* .
- A *read-lock* does not block an intended *read* by another transaction. The respective *read-lock* for the intended read is acquired (shared with the previous read) immediately after the intended read is requested, and then the intended read itself takes place.

TRANSCATION

A transaction is a set of changes that must all be made together. It is a program unit whose execution mayor may not change the contents of a database. Transaction is executed as a single unit. If the database was in consistent state before a transaction, then after execution of the transaction also, the database must be in a consistent state. For example, a transfer of money from one bank account to another requires two changes to the database both must succeed or fail together.

Example:

You are working on a system for a bank. A customer goes to the ATM and instructs it to transfer Rs. 1000 from savings to a checking account. This simple transaction requires two steps:

• Subtracting the money from the savings account balance.

• Adding the money to the checking account balance.

The code to create this transaction will require two updates to the database. For example, there will be two SQL statements: one UPDATE command to decrease the balance in savings and a second UPDATE command to increase the balance in the checking account.

You have to consider what would happen if a machine crashed between these two operations. The money has already been subtracted from the savings account will not be added to the checking account. It is lost. You might consider performing the addition to checking first, but then the customer ends up with extra money, and the bank loses. The point is that both changes must be made successfully. Thus, a transaction is defined as a set of changes that must be made together

States of Transaction

A transaction must be in one of the following states:

- **Active**: the initial state, the transaction stays in this state while it is executing.
- **Partially committed**: after the final statement has been executed.
- **Failed**: when the normal execution can no longer proceed.
- **Aborted**: after the transaction has been rolled back and the database has been restored to its state prior to the start of the transaction.
- **Committed**: after successful completion.

States of transaction
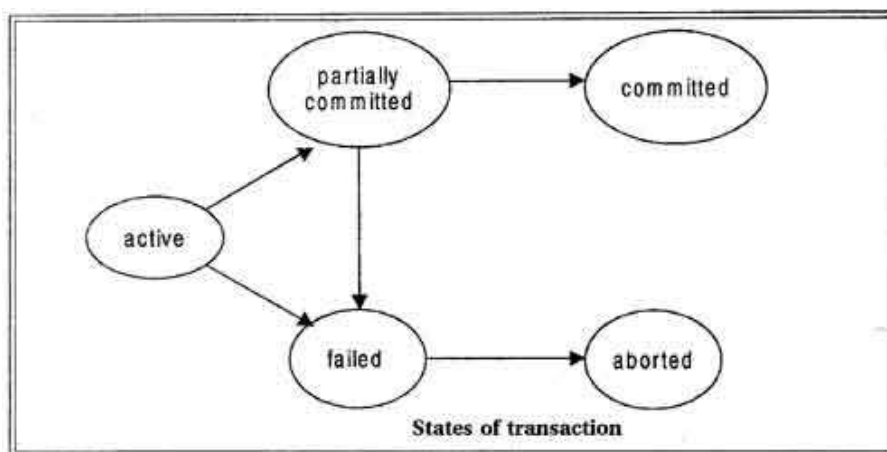
We say that a transaction has committed only if it has entered the committed state. Similarly, we say that a transaction has aborted only if it has entered the aborted state. A transaction is said to have terminated if has either committed or aborted.

A transaction starts in the active state. When it finishes its final statement, it enters the partially committed state. At this point, the transaction has completed its execution, but it is still possible that

it may have to be aborted, since the actual output may still be temporarily hiding in main memory and thus a hardware failure may preclude its successful completion

The database system then writes out enough information to disk that, even in the event of a failure, the updates performed by the transaction can be recreated when the system restarts after the failure. When the last of this information is written out, the transaction enters the committed state.

ACID Properties of Transactions

Most of what we're calling transactional locking relates to the ability of a database management system (DBMS) to ensure reliable transactions that adhere to these ACID properties. ACID is an acronym that stands for Atomicity, Consistency, Isolation, and Durability. Each of these properties is described in more detail below. However, all of these properties are related and must be considered together. They are more like different views of the same object than independent things.

Atomicity
Atomicity means all or nothing. Transactions often contain multiple separate actions. For example, a transaction may insert data into one table, delete from another table, and update a third table. Atomicity ensures that either all of these actions occur or none at all.

Consistency
Consistency means that transactions always take the database from one consistent state to another. So, if a transaction violates the databases consistency rules, then the entire transaction will be rolled back.

Isolation
Isolation means that concurrent transactions, and the changes made within them, are not visible to each other until they complete. This avoids many problems, including those that could lead to violation of other properties. The implementation of isolation is quite different in different DBMS'. This is also the property most often related to locking problems.

Durability
Durability means that committed transactions will not be lost, even in the event of abnormal termination. That is, once a user or program has been notified that a transaction was committed, they can be certain that the data will not be lost.

Timestamp Ordering Protocol

A **timestamp** is a tag that can be attached to any transaction or any data item, which denotes a specific time on which the transaction or data item had been activated in any way. We, who use computers, must all be familiar with the concepts of "Date Created" or "Last Modified" properties of files and folders. Well, timestamps are things like that.

A timestamp can be implemented in two ways. The simplest one is to directly assign the current value of the clock to the transaction or the data item. The other policy is to attach the value of a logical counter that keeps incrementing as new timestamps are required. The timestamp of a transaction denotes the time when it was first activated. The timestamp of a data item can be of the following two types:

W-timestamp (Q): This means the latest time when the data item Q has been written into.
R-timestamp (Q): This means the latest time when the data item Q has been read from.

These two timestamps are updated each time a successful read/write operation is performed on the data item Q.

How should timestamps be used?

The timestamp ordering protocol ensures that any pair of conflicting read/write operations will be executed in their respective timestamp order. This is an alternative solution to using locks.

For Read operations:

1. If TS (T) < W-timestamp (Q), then the transaction T is trying to read a value of data item Q which has already been overwritten by some other transaction. Hence the value which T wanted to read from Q does not exist there anymore, and T would be rolled back.
2. If TS (T) >= W-timestamp (Q), then the transaction T is trying to read a value of data item Q which has been written and committed by some other transaction earlier. Hence T will be allowed to read the value of Q, and the R-timestamp of Q should be updated to TS (T).

For Write operations:

1. If TS (T) < R-timestamp (Q), then it means that the system has waited too long for transaction T to write its value, and the delay has become so great that it has allowed another transaction to read the old value of data item Q. In such a case T has lost its relevance and will be rolled back.
2. Else if TS (T) < W-timestamp (Q), then transaction T has delayed so much that the system has allowed another transaction to write into the data item Q. in such a case too, T has lost its relevance and will be rolled back.
3. Otherwise the system executes transaction T and updates the W-timestamp of Q to TS (T).

Causes of DBMS Failure
There are many causes of DBMS failure. When a DBMS fails, it falls into an incorrect state and will likely contain erroneous data. Typical causes of DBMS failures include errors in the application program, an error by the terminal user, an operator error, loss of data validity and consistency, a hardware error, media failures, an error introduced by the environment, and errors caused by mischief or catastrophe.

Typically, the three major types of failure that result from a major hardware or software malfunction are transaction, system, and media. These failures may be caused by a natural disaster, computer crime, or user, designer, developer, or operator error. Each type of failure is described in the following paragraphs.

Transaction Failure.
Transaction failures occur when the transaction is not processed and the processing steps are rolled back to a specific point in the processing cycle. In a distributed data base environment, a single logical data base may be spread across several physical data bases.

Transaction failure can occur when some, but not all, physical data bases are updated at the same time.

System Failure.

System failure can be caused by bugs in the data base, operating system, or hardware. In each case, the Transaction processing is terminated without control of the application. Data in the memory is lost; however, disk storage remains stable. The system must recover in the amount of time it takes to complete all interrupted transactions. At one transaction per second, the system should recover in a few seconds. System failures may occur as often as several times a week.

Media Failure.

Disk crashes or controller failures can occur because of disk-write bugs in the operating system release, hardware errors in the channel or controller, head crashes, or media degradation. These failures are rare but costly.

By identifying the type of DBMS failure, an organization can define the state of activity to return to after recovery. To design the data base recovery procedures, the potential failures must be identified and the reliability of the hardware and software must be determined. the following is a summary of four such recovery actions:

· **TRANSACTION UNDO.** a transaction that aborts itself or must be aborted by the system during routine execution.

· **GLOBAL REDO.** When recovering from a system failure, the effects of all incomplete transaction must be rolled back.

· **PARTIAL UNDO.** While a system is recovering from failure, the results of completed transactions may not yet be reflected in the data base because execution has been terminated in an uncontrolled manner. Therefore, they must be repeated, if necessary, by the recovery component.

· **GLOBAL UNDO.** If the data base is totally destroyed, a copy of the entire data base must be reloaded from a backup source. A supplemental copy of the transaction is necessary to roll up the state of the data base to the present.

Techniques for Reviewing DBMS Recovery

The review of a DBMS recovery must ensure that employees with specific responsibilities perform their functions in accordance with operational policy and procedure. There are several useful DBMS recovery review techniques.

There are two ways to make the system operate again. First, all transactions that have occurred since the last backup can be reapplied, which would bring the data base up to date.

Second, the current contents of the data base can be taken and all transactions can be backed out until the integrity and validity of the data are restored. Whichever method is selected, it should be documented and a checklist of specific tasks and responsibilities identified.

The DBMS typically provides exhaustive review trails so that the system can know its exact state at any time. These review tails should be complete enough to reconstruct transactions and aid in recovery procedures. A data base administrator should know how to use these review trails in recovery to fully understand the inner workings of the DBMS.

A data base that has been backed up regularly helps the system recover from a failure and begin operating again as soon as possible. Daily backups are sufficient in most organizations. Those organizations that must always have current data must sometimes perform hourly backups. Each backup should be well documented to provide further insight into the review process.

Review techniques should examine application design, security procedures, and personnel control to ensure that managers can meet emergencies and have effective contingencies in place. These three areas are extremely critical review points for the auditor, management, users, and IS personnel.

Application Design

It is important to build sound recovery procedures and processes into an application during the design phase. The design of an application should take into consideration the data base control issues that affect backup and recovery processes. Possible weaknesses in controls include:

· Inaccurate or incomplete data in the data base.
· An inadequate audit trail.
· An inadequate service level.
· Failure of the DBMS to function as specified.
· Inadequate documentation.
· Lack of processing continuity.
· Lack of management support.
· Fraud or embezzlement.

The data base administrator should be responsible for examining the backup and recovery controls being considered by the user and developer when reviewing application design. The user and the developer of the application must assess the risks of not having appropriate controls in place to aid in recovery. Some key controls that should be adopted are:

· **Review trails.** A method of chronologically recording system activities that allows the reconstruction, review, and examination of each event in a transaction from inception to the final results.

· **Recovery procedures.** Automated or manual tools and techniques for recovering the integrity of a data base.

· **Application system failure procedures.** Procedures for users to follow in the event that their applications cannot operate.

· **Checkpoint data bases.** Copies of the data base and transaction files that are made at specific point in time for recovery purposes.

At a minimum, these controls should be tested during the module and integration testing phases of development. In terms of a new system review before implementation, these controls are most effective if thoroughly validated and approved by the user and developer before the system is placed into operation. One important issue to be considered in application design is data integrity.

Maintaining Data Integrity.

Data integrity concerns the accuracy of the contents of the data base. The integrity of the data can be compromised because of failures(i.e., events at which the system fails to provide normal operation or correct data). Failures are caused primarily by errors, which may originate in programs, interactions between these programs, or the system. A transaction is a sequence of actions. It should be designed and executed so that it either is successfully completed or has no effect on the data base. A transaction can fail to be completed for the following reasons:

· An action violates a security or integrity constraint.
· The user cancels the transaction.
· An unrecoverable I/O error occurs.

· The system backs out the transaction to resolve a deadlock.

· The application program fails.

· The system crashes.

Semantic Integrity.

This refers to the accuracy of the data base despite the fact that users or applications programs try to modify it incorrectly. Assuming that the data base security system prevents unauthorized access, and hence malicious attempts to corrupt data, most potential errors will be caused by incorrect input, incorrect programs, or lack of user understanding.

Traditionally, most integrity checking has been performed by the applications programs and by periodic auditing of the data base. The following are some problems that occur when relying on application programs for integrity checking:

· Checking is likely to be incomplete because the applications programmer may not be aware of the semantics of the complete data base.

· Each application program relies on other programs that can modify the data base, and a problem in one program could corrupt the whole data base.

· Code that enforces the same integrity constraints occurs in several programs. This leads to unnecessary duplication of the programming effort and exposes the system to potential inconsistencies.

· The criteria for integrity are buried within procedures and are therefore difficult to understand and control.

· Maintenance operations performed by users of high-level query language cannot be controlled.

Most of these errors could be detected through auditing, although the time lag in detecting errors by auditing can cause problems, such as difficulty in tracing the source of an error and hence correcting it as well as incorrect data used in various ways, causing errors to propagate through the data base and into the environment.

The semantics, or meaning, of a data base is partly drawn from a shared understanding among the users, partly implied by the data structures used, and partly expressed as integrity constraints. These constraints are explicitly stated by the individuals responsible for data control. Data bases can also be classified as:

· A single record or set.

· Static or transitional.

· General or selective.

· Immediate or deferred.

· Unconditional or conditional.

A system of concurrent transactions must be correctly synchronized—that is, the processing of these transactions must reach the same final state and produce the same output. Three forms of inconsistency result from concurrence: lost updates, an incorrect read, and an unrepeatable read. Lost updates can also result from backing up or undoing a transaction.

Correcting Inconsistency Problems.

The most commonly used approach to eliminate consistency problems is locking. The DBMS can use the locking facilities that the operating system provides so that multiple processes can synchronize their concurrent access of shared resources. A lock can be granted to multiple processes, but a given object cannot be locked in shared and exclusive mode at the same time. Shared and exclusive modes conflict because they are incompatible. The operating system usually provides lock and unlock commands for requesting and releasing locks. If a lock request cannot be granted, the

process is suspended until the request can be granted. If transactions do not follow restrictive locking rules, Deadlock can occur. Deadlock can cause the loss of an entire file; therefore, it is critical to have a recovery system in place to alleviate this problem.

The Deadlock problem can be solved either by preventing Deadlock or by detecting them after they occur and taking steps to resolve them. Deadlock can be prevented by placing restrictions on the way locks are requested. They can be detected by examining the status of locks. After they are detected, the Deadlock can be resolved by aborting a transaction and rescheduling it. Methods for selecting the best transaction to abort have also been developed.

A synchronization problem can occur in a distributed data base environment, such as a client/server network. Data bases can become out of sync when data from one data base fails to be updated on other data bases. When updates fail to occur, users at some locations may use data that is not current with data at other locations. Distributed data bases provide different types of updating mechanisms. In a two-phase commit update process, network nodes must be online and receive data simultaneously before updates can occur. A newer update method called *data replication* enables updates to be stored until nodes are online and ready to receive. Update methods must ensure currency in all network data bases.

Security Procedures

A data base usually contains information that is vital to an organization's survival. A secure data base environment, with physical and logical security controls, is essential during recover procedures.

Physical Security.

In some distributed environments, many physical security controls, such as the use of security badges and cipher locks, are not feasible and the organization must rely more heavily on logical security measures. In these cases, many organizational members may have data processing needs that do not involve a data base but require the use of computer peripherals.

Logical Security.

Logical security prevents unauthorized users from invoking DBMS functions. The primary means of implementing this type of security is the use of passwords to prevent access to files, records, data elements, and DBMS utilities. Passwords should be checked to ensure that they are designated in an intelligent, logical manner.

Security Logs.

Each time an unauthorized user attempts to access the data base, it should be recorded in a security log. Entries in this log should consist of user ID, terminal or port number, time, date, and type of infraction. With this information, it is possible to investigate any serious breaches of security. From the data base administrator's standpoint, evidence that the DBMS is detecting security violations and that a consistent procedure is used to follow them up should be sufficient.

Personnel Control

Data base recovery involves ensuring that only authorized users are allowed access and that no subsequent misuse of information occurs. These controls are usually reestablished when a system becomes operational. When operations cease or problems occur, however, controls often become inoperative.

The three primary classes of data base users are data base administrator, applications and systems programmers, and end users--and each has a unique view of the data. The DBMS must be flexible enough to present data appropriately to each class of user and maintain the proper controls to inhibit abuse of the system, especially during recovery, when controls may not be fully operational.

Data Base Administrator.

The data base administrator is responsible for ensuring that the data base retains its integrity and is accountable if the data base becomes compromised, no matter what circumstances arise. This individual has ultimate power over the schema that the organization has implemented. Any modifications or additions to this schema must be approved by the data base administrator. Permission to use subschema (i.e., logical views) is given to end users and programmers only after their intentions are fully known and are consistent with organizational goals.

Because the data base administrator has immediate and unrestricted access to almost every piece of valuable organizational information, an incompetent employee in this position can expose the organization to enormous risk, especially during DBMS recovery. Therefore, an organization should have controls in place to ensure the appointment of a qualified data base administrator.

The data base administrator must ensure that appropriate procedures are followed during DBMS recovery. The data base administrator should also validate and verify the system once it has been recovered before allowing user access so that if controls are not functioning or accessing problem continue, users will not be affected.

Applications and Systems Programmers.

After recovery, programmers must access the data base to manipulate and report on data according to some predetermined specification or to access whether data loss has occurred. Each application should have a unique subschemas with which to work. After recovery, the data base administrator validates the subschemas organization to ensure that it is operating properly and allowing the application to receive only the data necessary to perform its tasks. Systems programmers must be controlled in a slightly different manner than applications programmers. They must have the freedom to perform their tasks but be constrained from altering production programs or system utility programs in a fraudulent
manner.

End Users.

End users are defined as all organizational members not included in the previous categories who need to interact with the data base through DBMS utilities or application programs. Data elements of the data base generally originate from end users. Each data element should be assigned to an end user. The end user is then responsible for defining the element's access and security rules. Every other user who wishes to use this data element must confer with the responsible end user. If access is granted, the data base administrator must implement any restrictions placed on the request through the DBMS.

Assigning ownership of specific data elements to end users discourages the corruption of data elements, thereby enhancing data base integrity. Reviewers should ensure that this process exists and is appropriately reinstituted after the recovery process has been completed and operational approval has been provided by the data base administrator.

After recovery, the data base administrator should ensure that all forms of security practices and procedures are reinstated. These processes are a part of data base security.

Object Oriented Database (OODB)

Object Oriented Database (**OODB**) provides all the facilities associated with object oriented paradigm. It enables us to create classes, organize objects, structure an inheritance hierarchy and call methods of other classes. Besides these, it also provides the facilities associated with standard database systems. However, object oriented database systems have not yet replaced the RDBMS in commercial business applications. Following are the two different approaches for designing an object-oriented database:

• Designed to store, retrieve and manage objects created by programs written in some object oriented languages (OOL) such as C++ or java.

Although a relational database can be used to store and manage objects, it does not understand objects as such. Therefore, a middle layer called object manager or object-oriented layer software is required to translate objects into tuples of a relation .

• Designed to provide object-oriented facilities to users of non object-oriented programming languages (OOPLs) such as C or Pascal.

The user will create classes, objects, inheritance and so on and the database system will store and manage these objects and classes. This second approach, thus, turns non-OOPLs into OOPLs. A translation layer is required to map the objects created by user into objects of the database system.

Advantages of OODBMS

Enriched modeling capabilities

The object-oriented data model allows the 'real world' to be modeled more closely. The object, which encapsulates both state and behavior, is a more natural and realistic representation of real-world objects. An object can store all the relationships it has with other objects, including many-to-many relationships, and objects can be formed into complex objects that the traditional data models cannot cope with easily.

Extensibility

OODBMSs allow new data types to be built from existing types. The ability to factor out common properties of several classes and form them into a superclass that can be shared with subclasses can greatly reduce redundancy within system and, as we stated· at the start of this chapter, is regarded as one of the main advantages of object orientation. Further, the reusability of classes promotes faster development and easier maintenance of the database and its applications.

Capable of handling a large variety of data types

Unlike traditional databases (such as hierarchical, network or relational), the object oriented database are capable of storing different types of data, for example, pictures, voice video, including text, numbers and so on.

Removal of impedance mismatch

A single language interface between the Data Manipulation Language (DML) and the programming language overcomes the impedance mismatch. This eliminates many of the efficiencies that occur in mapping a declarative language such as SQL to an imperative 'language such as 'C'. Most OODBMSs provide a DML that is computationally complete compared with SQL, the 'standard language of RDBMSs.

More expressive query language

Navigational access from the object is the most common form of data access in an OODBMS. This is in contrast to the associative access of SQL (that is, declarative statements with selection based on one or more predicates). Navigational access is more suitable for handling parts explosion, recursive queries, and so on.

Support for schema evolution

The tight coupling between data and applications in an OODBMS makes schema evolution more feasible.

Support for long-duration, transactions

Current relational DBMSs enforce serializability on concurrent transactions to maintain database consistency. OODBMSs use a different protocol to handle the types of long-duration transaction that are common in many advanced database application.

Applicability to advanced database applications

There are many areas where traditional DBMSs have not been particularly successful, such as, Computer-Aided Design (CAD), Computer-Aided Software Engineering (CASE), Office Information System(OIS), and Multimedia Systems. The enriched modeling capabilities of OODBMSs have made them suitable for these applications.

Improved performance

There have been a number of benchmarks that have suggested OODBMSs provide significant performance improvements over relational DBMSs. The results showed an average 30-fold performance improvement for the OODBMS over the RDBMS.

Disadvantages of OODBMSs

There are following disadvantages of OODBMSs:

Lack of universal data model: There is no universally agreed data model for an OODBMS, and most models lack a theoretical foundation. This .disadvantage is seen as a significant drawback, and is comparable to pre-relational systems.

Lack of experience: In comparison to RDBMSs the use of OODBMS is still relatively limited. This means that we do not yet have the level of experience that we have with traditional systems. OODBMSs are still very much geared towards the programmer, rather than the naïve end-user. Also there is a resistance to the acceptance of the technology. While the OODBMS is limited to a small niche market, this problem will continue to exist

Lack of standards: There is a general lack of standards of OODBMSs. We have already mentioned that there is not universally agreed data model. Similarly, there is no standard object-oriented query language.

Competition: Perhaps one of the most significant issues that face OODBMS vendors is the competition posed by the RDBMS and the emerging ORDBMS products. These products have an established user base with significant experience available. SQL is an approved standard and the relational data model has a solid theoretical formation and relational products have many supporting tools to help .both end-users and developers.

Query optimization compromises encapsulations: Query optimization requires. An understanding of the underlying implementation to access the database efficiently. However, this compromises the concept of incrassation.

Locking at object level may impact performance Many OODBMSs use locking as the basis for concurrency control protocol. However, if locking is applied at the object level, locking of an inheritance hierarchy may be problematic, as well as impacting performance.

Complexity: The increased functionality provided by the OODBMS (such as the illusion of a single-level storage model, pointer sizzling, long-duration transactions, version management, and schema evolution--makes the system more complex than that of traditional DBMSs. In complexity leads to products that are more expensive and more difficult to use.

Lack of support for views: Currently, most OODBMSs do not provide a view mechanism, which, as we have seen previously, provides many advantages such as data independence, security, reduced complexity, and customization.

Lack of support for security: Currently, OODBMSs do not provide adequate security mechanisms. The user cannot grant access rights on individual objects or classes.

If OODBMSs are to expand fully into the business field, these deficiencies must be rectified.

OBJECT RELATIONAL DBMS

Relational DBMSs are currently the dominant database technology. The OODBMS has also become the favored system for financial and telecommunications applications. Although the OODBMS market is still same. The OODBMS continues to find new application areas, such as the World Wide Web. Some industry analysts expect the market for the OODBMSs to grow at over 50% per year, a rate faster than the total database market. However, their sales are unlikely to overtake those of relational systems because of the wealth of businesses that find RDBMSs acceptable, and because businesses have invested to much money and resources in their development that change is prohibitive. This is the approach that has been taken by many extended relational DBMSs, although each has implemented different combinations of features. Thus there is no single extended relational model rather, there are a variety of these models, whose characteristics depends upon the way and the degree to which extensions were made. However, all the models do share the same basic relational tables and query language, all incorporate some concept of 'object, and some have the ability to store methods (or procedures or triggers), as well as data in the database.

Four-Quadrant View of Database World
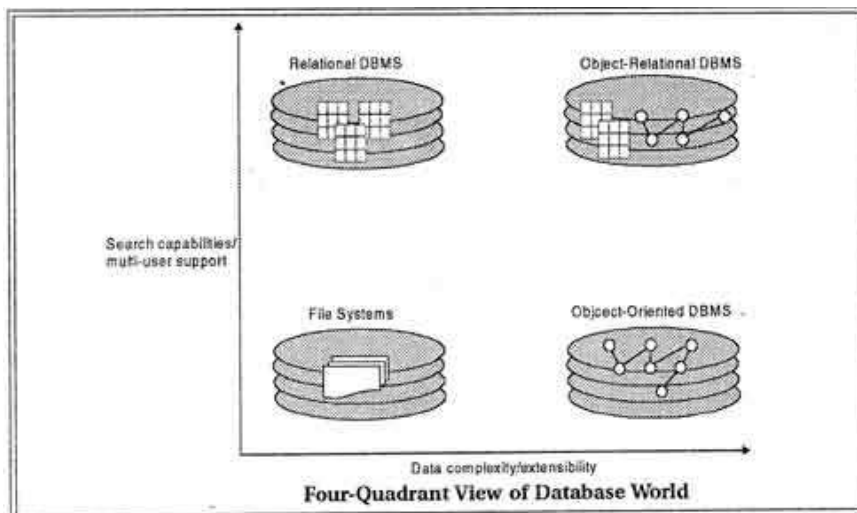
In a four-quadrant view of the database world, as illustrated in the figure, the lower-left quadrant are those applications that process simple data and have no requirements for querying the data.

These types of application, for example standard text processing packages such as Word,

WordPerfect, and Frame maker, can use the underlying operating system to obtain the essential DBMS functionality of persistence. In the lower-right quadrant are those applications that process complex data but again have no significant requirements for querying the data. For these types of application, for example computer-aided design packages, an OODBMS may be an appropriate choice of DBMS.

In the top-left quadrant are those applications that process simple data and also have requirements for complex querying. Many traditional business applications fall into this quadrant and an RDBMS may be the most appropriate DBMS.

Finally, in the top-right quadrant are those applications that process completed data and have complex querying requirements. This represents many of the advanced database applications and for these applications an ORDBMS may be the appropriate choice of DBMS.

Advantages and Disadvantages of ORDBMSS

ORDBMSs can provide appropriate solutions for many types of advanced database applications. However, there are also disadvantages.

Advantages of ORDBMSs

There are following advantages of ORDBMSs:

Reuse and Sharing: The main advantages of extending the Relational data model come from reuse and sharing. Reuse comes from the ability to extend the DBMS server to perform standard functionality centrally, rather than have it coded in each application.

Increased Productivity: ORDBMS provides increased productivity both for the developer and for the, end user

Use of experience in developing RDBMS: Another obvious advantage is that .the extended relational approach preserves the significant body of knowledge and experience that has gone into developing relational applications. This is a significant advantage, as many organizations would find it prohibitively expensive to change. If the new functionality is designed appropriately, this approach should allow organizations to take advantage of the new extensions in an evolutionary way without losing the benefits of current database features and functions.

Disadvantages of ORDBMSs

The ORDBMS approach has the obvious disadvantages of complexity and associated increased costs. Further, there are the proponents of the relational approach that believe the· essential simplicity' and purity of the .relational model are lost with these types of extension.

ORDBMS vendors are attempting to portray object models as extensions to the relational model with some additional complexities. This potentially misses the point of object orientation, highlighting the large semantic gap between these two technologies. Object applications are simply not as data-centric as relational-based ones.

PARALLEL DATABASE

• A parallel database system seeks to improve performance through parallelization of various operations, such as loading data, building indexes and evaluating queries. Although data may be stored in a distributed fashion, the distribution is governed solely by performance considerations. Parallel database improves processing and input/output speeds by using multiple CPUs and disks in

parallel. Centralized and client-server database systems are not powerful enough to handle such applications. In parallel processing, many operations are performed simultaneously, as opposed to serial processing, in which the computational steps are performed sequentially.

- A parallel database system seeks to improve performance through parallelization of various operations, such as loading data, building indexes and evaluating queries.
- Although data may be stored in a distributed fashion, the distribution is governed solely by performance considerations. Parallel databases improve processing and input/output speeds by using multiple CPUs and disks in parallel.
- Centralized and client–server database systems are not powerful enough to handle such applications.
- In parallel processing, many operations are performed simultaneously, as opposed to serial processing, in which the computational steps are performed sequentially.
- Parallel databases can be roughly divided into two groups, the first group of architecture is the multiprocessor architecture, the alternatives of which are the followings :

- ✓ **Shared memory architecture,** where multiple processors share the main memory space.
- ✓ **Shared disk architecture**, where each node has its own main memory, but all nodes share mass storage, usually a storage area network. In practice, each node usually also has multiple processors.
- ✓ **Shared nothing architecture**, where each node has its own mass storage as well as main memory.

Distributed Database Architecture

A **distributed database system** allows applications to access data from local and remote databases. In a **homogenous distributed database system**, each database is an Oracle Database. In a **heterogeneous distributed database system**, at least one of the databases is not an Oracle Database. Distributed databases use client**/server** architecture to process information requests.

It contains the following database systems:

- Homogenous Distributed Database Systems
- Heterogeneous Distributed Database Systems
- Client/Server Database Architecture

Homogenous Distributed Database Systems

A homogenous distributed database system is a network of two or more Oracle Databases that reside on one or more machines. Below Figure illustrates a distributed system that connects three databases: hq, mfg, and sales. An application can simultaneously access or modify the data in several databases in a single distributed environment. For example, a single query from a Manufacturing client on local database mfg can retrieve joined data from the products table on the local database and the dept table on the remote hq database.

For a client application, the location and platform of the databases are transparent. You can also create **synonyms** for remote objects in the distributed system so that users can access them with the same syntax as local objects. For example, if you are connected to database mfg but want to access data on database hq, creating a synonym on mfg for the remote dept table enables you to issue this query:

SELECT * FROM dept;

Homogeneous Distributed Database
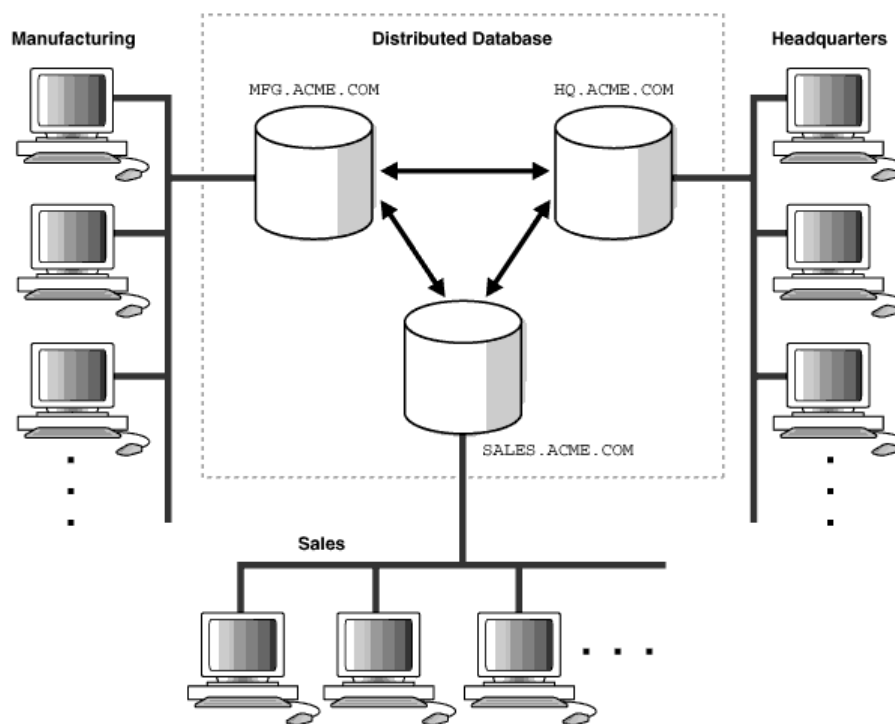
An Oracle Database distributed database system can incorporate Oracle Databases of different versions. All supported releases of Oracle Database can participate in a distributed database system. Nevertheless, the applications that work with the distributed database must understand the functionality that is available at each node in the system. A distributed database application cannot expect an Oracle7 database to understand the SQL extensions that are only available with Oracle Database.

Heterogeneous Distributed Database Systems

In a heterogeneous distributed database system, at least one of the databases is a non-Oracle Database system. To the application, the heterogeneous distributed database system appears as a single, local, Oracle Database. The local Oracle Database server hides the distribution and heterogeneity of the data.

The Oracle Database server accesses the non-Oracle Database system using Oracle Heterogeneous Services in conjunction with an **agent**. If you access the non-Oracle Database data store using an Oracle Transparent Gateway, then the agent is a system-specific application. For example, if you include a Sybase database in an Oracle Database distributed system, then you need to obtain a Sybase-specific transparent gateway so that the Oracle Database in the system can communicate with it.

Alternatively, you can use **generic connectivity** to access non-Oracle Database data stores so long as the non-Oracle Database system supports the ODBC or OLE DB protocols.

Heterogeneous Services

Heterogeneous Services (HS) is an integrated component within the Oracle Database server and the enabling technology for the current suite of Oracle Transparent Gateway products. HS provides the common architecture and administration mechanisms for Oracle Database gateway products and other heterogeneous access facilities. Also, it provides upwardly compatible functionality for users of most of the earlier Oracle Transparent Gateway releases.
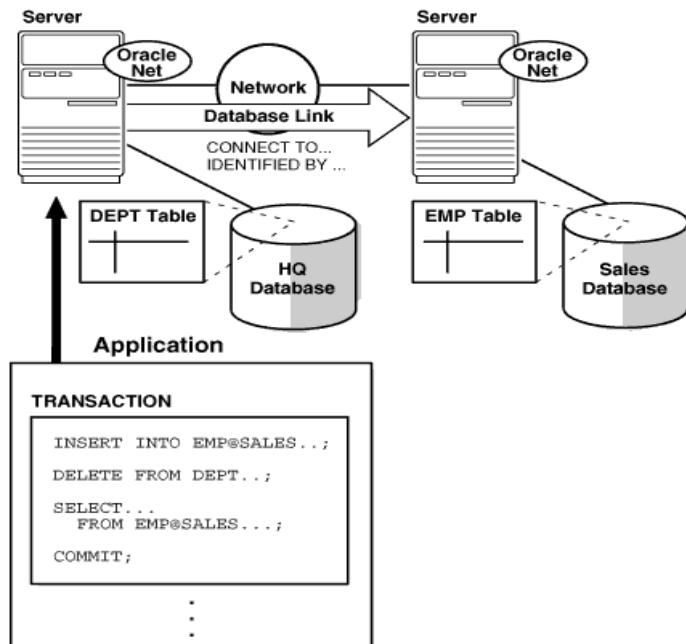
Transparent Gateway Agents

For each non-Oracle Database system that you access, Heterogeneous Services can use a transparent gateway agent to interface with the specified non-Oracle Database system. The agent is specific to the non-Oracle Database system, so each type of system requires a different agent.

The transparent gateway agent facilitates communication between Oracle Database and non-Oracle Database systems and uses the Heterogeneous Services component in the Oracle Database server. The agent executes SQL and transactional requests at the non-Oracle Database system on behalf of the Oracle Database server.

Client/Server Database Architecture

A database server is the Oracle software managing a database, and a client is an application that requests information from a server. Each computer in a network is a node that can host one or more databases. Each node in a distributed database system can act as a client, a server, or both, depending on the situation.

An Oracle Database Distributed Database System

```
Server                          Server
  Oracle                          Oracle
   Net                             Net
          Network
       Database Link
       CONNECT TO...
       IDENTIFIED BY ...

DEPT Table                      EMP Table

            HQ                            Sales
          Database                       Database

Application

TRANSACTION

   INSERT INTO EMP@SALES..;

   DELETE FROM DEPT..;

   SELECT...
      FROM EMP@SALES...;

   COMMIT;

            .
            .
            .
```

A client can connect **directly** or **indirectly** to a database server. A direct connection occurs when a client connects to a server and accesses information from a database contained on that server. For example, if you connect to the hq database and access the dept table on this database as in below Figure, you can issue the following:

SELECT * FROM dept;

This query is direct because you are not accessing an object on a remote database.

In contrast, an indirect connection occurs when a client connects to a server and then accesses information contained in a database on a different server. For example, if you connect to the hq database but access the emp table on the remote sales database as in above Figure_you can issue the following:

SELECT * FROM emp@sales;

PARALLEL VS. DISTRIBUTED DATABASE :

Parallel Database System seeks to improve performance through parallelization of various operations, such as data loading, index building and query evaluating. Although data may be stored in a distributed fashion in such a system, the distribution is governed solely by performance considerations.

In **Distributed Database System**, data is physically stored across several sites, and each site is

typically managed by a DBMS capable of running independent of the other sites. In contrast to parallel databases, the distribution of data is governed by factors such as local ownership and increased availability.

PDB & DDB Comparison:

1. System Components
- Distributed DBMS consists of many **Geo-distributed**, **low-bandwidth link connected**, **autonomic** sites.
- Parallel DBMS consists of **tightly coupled**, **high-bandwidth link connected**, **non-autonomic** nodes.

2. Component Role
- Sites in Distributed DBMS can work independently to handle **local** transactions or work together to handle **global** transactions.
- Nodes in Parallel DBMS can only work together to handle **global** transactions.

3. Design Purposes
= Distributed DBMS is for:
   - **Sharing Data**
   - **Local Autonomy**
   - **High Availability**
= Parallel DBMS is for:
   - **High Performance**
   - **High Availability**

   But both PDB&DDB need to consider the following problems:

 1. Data Distribution (Placement & Replicatioin);

2. Query Parallelization(Distributed Evaluation). And also, many parallel system consists of network of workstation, the difference between Parallel DB & Distributed DB is becoming smaller.

DATA WAREHOUSING

A data warehouse is a collection of data marts representing historical data from different operations in the company. This data is stored in a structure optimized for querying and data analysis as a data warehouse. Table design, dimensions and organization should be consistent throughout a data warehouse so that reports or queries across the data warehouse are consistent.

A data warehouse can also be viewed as a database for historical data from different functions within a company. The term Data Warehouse was coined by Bill Inmon in 1990, which he defined in the following way: "A warehouse is a subject-oriented, integrated, time-variant and non-volatile collection of data in support of management's decision making process". He defined the terms in the sentence as follows: Subject Oriented: Data that gives information about a particular subject instead of about a company's ongoing operations.

Integrated: Data that is gathered into the data warehouse from a variety of sources and merged into a coherent whole.

Time-variant: All data in the data warehouse is identified with a particular time period.

Non-volatile: Data is stable in a data warehouse. More data is added but data is never removed. This enables management to gain a consistent picture of the business. It is a single, complete and consistent store of data obtained from a variety of different sources made available to end users in what they can understand and use in a business context. It can be

• Used for decision Support
• Used to manage and control business
• Used by managers and end-users to understand the business and make judgments

Benefits of data warehousing

• Data warehouses are designed to perform well with aggregate queries running on large amounts of data.
• The structure of data warehouses is easier for end users to navigate, understand and query against unlike the relational databases primarily designed to handle lots of transactions.
• Data warehouses enable queries that cut across different segments of a company's operation. E.g. production data could be compared against inventory data even if they were originally stored in different databases with different structures.
• Queries that would be complex in very normalized databases could be easier to build and maintain in data warehouses, decreasing the workload on transaction systems.
• Data warehousing is an efficient way to manage and report on data that is from a variety of sources, non uniform and scattered throughout a company.
• Data warehousing is an efficient way to manage demand for lots of information from lots of users.
• Data warehousing provides the capability to analyze large amounts of historical data for nuggets of wisdom that can provide an organization with competitive advantage.

Data Warehouse Characteristics

• A data warehouse can be viewed as an information system with the following attributes:

– It is a database designed for analytical tasks

– It's content is periodically updated

– It contains current and historical data to provide a historical perspective of information

Data warehouse admin and management

The management of data warehouse includes,

• Security and priority management

• Monitoring updates from multiple sources

• Data quality checks

• Managing and updating meta data

• Auditing and reporting data warehouse usage and status

• Purging data

• Replicating, sub setting and distributing data

• Backup and recovery

• Data warehouse storage management which includes capacity planning, hierarchical storage management and purging of aged data etc..

DESIGN OF DATA WAREHOUSE

The following nine-step method is followed in the design of a data warehouse:

1. Choosing the subject matter

2. Deciding what a fact table represents

3. Identifying and conforming the dimensions

4. Choosing the facts

5. Storing pre calculations in the fact table

6. Rounding out the dimension table

7. Choosing the duration of the db

8. The need to track slowly changing dimensions

9. Deciding the query priorities and query models

Technical considerations

A number of technical issues are to be considered when designing a data warehouse environment. These issues include:

• The hardware platform that would house the data warehouse

• The dbms that supports the warehouse data

• The communication infrastructure that connects data marts, operational systems and end users

• The hardware and software to support meta data repository

• The systems management framework that enables admin of the entire environment

Implementation considerations

The following logical steps needed to implement a data warehouse:

• Collect and analyze business requirements

• Create a data model and a physical design

• Define data sources

• Choose the db tech and platform

• Extract the data from operational db, transform it, clean it up and load it into the warehouse

• Choose db access and reporting tools

• Choose db connectivity software

• Choose data analysis and presentation s/w

• Update the data warehouse

Access tools

Data warehouse implementation relies on selecting suitable data access tools. The best way to choose this is based on the type of data can be selected using this tool and the kind of access it permits for a particular user. The following lists the various type of data that can be accessed:

• Simple tabular form data

• Ranking data

• Multivariable data

• Time series data

• Graphing, charting and pivoting data

• Complex textual search data

• Statistical analysis data

• Data for testing of hypothesis, trends and patterns

• Predefined repeatable queries

• Ad hoc user specified queries

• Reporting and analysis data

• Complex queries with multiple joins, multi level sub queries and sophisticated search criteria

Data extraction, clean up, transformation and migration

A proper attention must be paid to data extraction which represents a success factor for a data warehouse architecture. When implementing data warehouse several the following selection criteria that affect the ability to transform, consolidate, integrate and repair the data should be considered:

• Timeliness of data delivery to the warehouse

• The tool must have the ability to identify the particular data and that can be read by conversion tool

• The tool must support flat files, indexed files since corporate data is still in this type

• The tool must have the capability to merge data from multiple data stores

197

• The tool should have specification interface to indicate the data to be extracted

• The tool should have the ability to read data from data dictionary

• The code generated by the tool should be completely maintainable

• The tool should permit the user to extract the required data

• The tool must have the facility to perform data type and character set translation

• The tool must have the capability to create summarization, aggregation and derivation of records

• The data warehouse database system must be able to perform loading data directly from

these tools

Data placement strategies

– As a data warehouse grows, there are at least two options for data placement. One is to put some of the data in the data warehouse into another storage media.

– The second option is to distribute the data in the data warehouse across multiple servers.

User levels

The users of data warehouse data can be classified on the basis of their skill level in accessing the warehouse. There are three classes of users:Casual users: are most comfortable in retrieving info from warehouse in pre defined formats and running pre existing queries and reports. These users do not need tools that allow for building standard and ad hoc reports

Power Users: can use pre defined as well as user defined queries to create simple and ad hoc reports. These users can engage in drill down operations. These users may have the experience of using reporting and query tools.

Expert users: These users tend to create their own complex queries and perform standard analysis on the info they retrieve. These users have the knowledge about the use of query and report tools

Benefits of data warehousing

Data warehouse usage includes,

– Locating the right info

198

– Presentation of info

– Testing of hypothesis

– Discovery of info

– Sharing the analysis

The benefits can be classified into two:

• Tangible benefits (quantified / measureable):It includes,

– Improvement in product inventory

– Decrement in production cost

– Improvement in selection of target markets

– Enhancement in asset and liability management

• Intangible benefits (not easy to quantified): It includes,

– Improvement in productivity by keeping all data in single location and eliminating

– Reduced redundant processing

– Enhanced customer relation rekeying of data

ARCHITECTURE OF DATA WAREHOUSING

The data in a data warehouse comes from operational systems of the organization as well as from other external sources. These are collectively referred to as *source systems*. The data *extracted* from source systems is stored in a area called *data staging area*, where the data is cleaned, *transformed*, combined, duplicated to prepare the data for us in the data warehouse. The data staging area is generally a collection of machines where simple activities like sorting and sequential processing takes place. The data staging area does not provide any query or presentation services. As soon as a system provides query or presentation services, it is categorized as a *presentation server*. A presentation server is the target machine on which the data is *loaded* from the data staging area organized and stored for direct querying by end users, report writers and other applications. The three different kinds of systems that are required for a data warehouse are:

1. Source Systems
2. Data Staging Area
3. Presentation servers

The data travels from source systems to presentation servers via the data staging area. The entire process is popularly known as ETL (extract, transform, and load) or ETT (extract, transform, and transfer). Oracle's ETL tool is called Oracle Warehouse Builder (OWB) and MS SQL Server's ETL tool is called Data Transformation Services (DTS).

A typical architecture of a data warehouse is shown below:
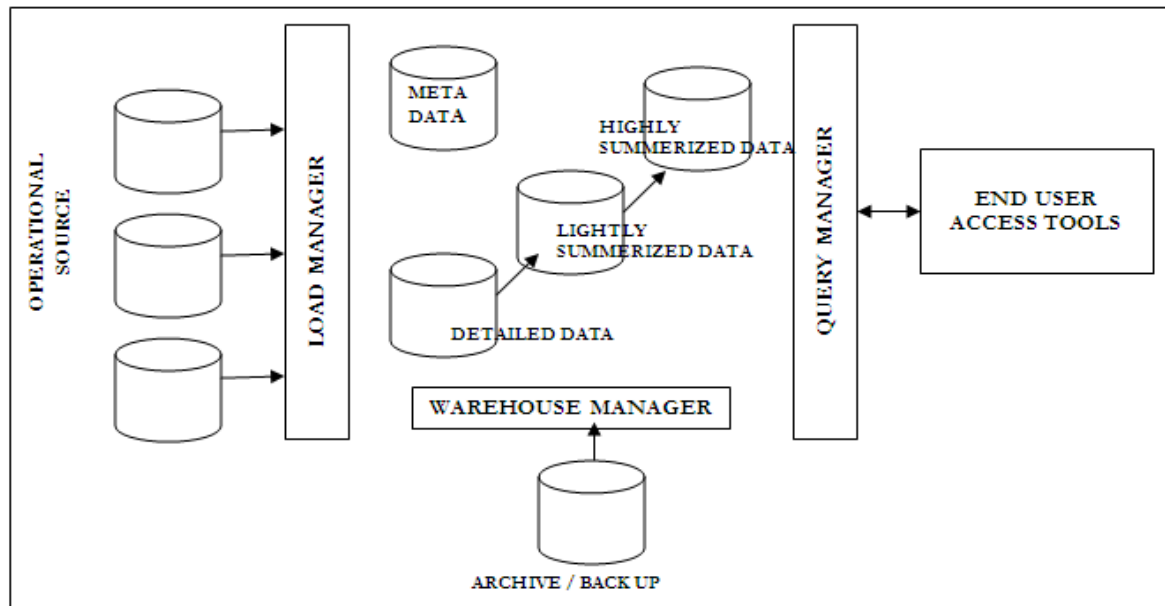
Each component and the tasks performed by them are explained below:

1. OPERATIONAL DATA
The sources of data for the data warehouse is supplied from:

(i)   The data from the mainframe systems in the traditional network and hierarchical format.
(ii)   Data can also come from the relational DBMS like Oracle, Informix.
(iii)   In addition to these internal data, operational data also includes external data obtained from commercial databases and databases associated with supplier and customers.

2. LOAD MANAGER
The load manager performs all the operations associated with extraction and loading data into the data warehouse. These operations include simple transformations of the data to prepare the data for entry into the warehouse. The size and complexity of this component will vary between data warehouses and may be constructed using a combination of vendor data loading tools and custom built programs.

3. WAREHOUSE MANAGER

The warehouse manager performs all the operations associated with the management of data in the warehouse. This component is built using vendor data management tools and custom built programs. The operations performed by warehouse manager include:

(i)   Analysis of data to ensure consistency
(ii)   Transformation and merging the source data from temporary storage into data warehouse tables
(iii)   Create indexes and views on the base table.
(iv)   Denormalization
(v)   Generation of aggregation
(vi)   Backing up and archiving of data

In certain situations, the warehouse manager also generates query profiles to determine which indexes ands aggregations are appropriate.

4. QUERY MANAGER

The query manager performs all operations associated with management of user queries. This component is usually constructed using vendor end-user access tools, data warehousing monitoring tools, database facilities and custom built programs. The complexity of a query manager is determined by facilities provided by the end-user access tools and database.

5. DETAILED DATA

This area of the warehouse stores all the detailed data in the database schema. In most cases detailed data is not stored online but aggregated to the next level of details. However the detailed data is added regularly to the warehouse to supplement the aggregated data.

6. LIGHTLY AND HIGHLY SUMMERIZED DATA

The area of the data warehouse stores all the predefined lightly and highly summarized (aggregated) data generated by the warehouse manager. This area of the warehouse is transient as it will be subject to change on an ongoing basis in order to respond to the changing query profiles. The purpose of the summarized information is to speed up the query performance. The summarized data is updated continuously as new data is loaded into the warehouse.

7. ARCHIVE AND BACK UP DATA

This area of the warehouse stores detailed and summarized data for the purpose of archiving and back up. The data is transferred to storage archives such as magnetic tapes or optical disks.

8. META DATA

The data warehouse also stores all the Meta data (data about data) definitions used by all processes in the warehouse. It is used for variety of purposed including:

(i)   The extraction and loading process – Meta data is used to map data sources to a common view of information within the warehouse.
(ii)   The warehouse management process – Meta data is used to automate the production of summary tables.
(iii)   As part of Query Management process Meta data is used to direct a query to the most appropriate data source.

The structure of Meta data will differ in each process, because the purpose is different. More about Meta data will be discussed in the later Lecture Notes.

9. END-USER ACCESS TOOLS
The principal purpose of data warehouse is to provide information to the business managers for strategic decision-making. These users interact with the warehouse using end user access tools. The examples of some of the end user access tools can be:

      (i)   Reporting and Query Tools
      (ii)   Application Development Tools
      (iii)   Executive Information Systems Tools
      (iv)   Online Analytical Processing Tools
      (v)   Data Mining Tools

THE E T L (EXTRACT TRANSFORMATION LOAD) PROCESS
In this section we will discussed about the 4 major process of the data warehouse. They are **extract** (data from the operational systems and bring it to the data warehouse), **transform** (the data into internal format and structure of the data warehouse), **cleanse** (to make sure it is of sufficient quality to be used for decision making) and **load** (cleanse data is put into the data warehouse).
The four processes from extraction through loading often referred collectively as **Data Staging**.

EXTRACT
Some of the data elements in the operational database can be reasonably be expected to be useful in the decision making, but others are of less value for that purpose. For this reason, it is necessary to extract the relevant data from the operational database before bringing into the data warehouse. Many commercial tools are available to help with the extraction process. **Data Junction** is one of the commercial products. The user of one of these tools typically has an easy-to-use windowed interface by which to specify the following:

      (i)   Which files and tables are to be accessed in the source database?
      (ii)   Which fields are to be extracted from them? This is often done internally by SQL Select statement.
      (iii)   What are those to be called in the resulting database?
      (iv)   What is the target machine and database format of the output?
      (v)   On what schedule should the extraction process be repeated?

TRANSFORM

The operational databases developed can be based on any set of priorities, which keeps changing with the requirements. Therefore those who develop data warehouse based on these databases are typically faced with inconsistency among their data sources. Transformation process deals with rectifying any inconsistency (if any).

One of the most common transformation issues is 'Attribute Naming Inconsistency'. It is common for the given data element to be referred to by different data names in different databases. Employee Name may be EMP_NAME in one database, ENAME in the other. Thus one set of Data Names are picked and used consistently in the data warehouse. Once all the data elements have right names, they must be converted to common formats. The conversion may encompass the following:

(i)   Characters must be converted ASCII to EBCDIC or vise versa.
(ii)   Mixed Text may be converted to all uppercase for consistency.
(iii)   Numerical data must be converted in to a common format.
(iv)   Data Format has to be standardized.
(v)   Measurement may have to convert. (Rs/ $)
(vi)   Coded data (Male/ Female, M/F) must be converted into a common format.

All these transformation activities are automated and many commercial products are available to perform the tasks. **DataMAPPER** from Applied Database Technologies is one such comprehensive tool.

CLEANSING

Information quality is the key consideration in determining the value of the information. The developer of the data warehouse is not usually in a position to change the quality of its underlying historic data, though a data warehousing project can put spotlight on the data quality issues and lead to improvements for the future. It is, therefore, usually necessary to go through the data entered into the data warehouse and make it as error free as possible. This process is known as **Data Cleansing**.

Data Cleansing must deal with many types of possible errors. These include missing data and incorrect data at one source; inconsistent data and conflicting data when two or more source are involved. There are several algorithms followed to clean the data, which will be discussed in the coming lecture notes.

LOADING

Loading often implies physical movement of the data from the computer(s) storing the source database(s) to that which will store the data warehouse database, assuming it is different. This takes place immediately after the extraction phase. The most common channel for data movement is a high-speed communication link. Ex: Oracle Warehouse Builder is the API from Oracle, which provides the features to perform the ETL task on Oracle Data Warehouse.

DATA MINING

Architecture of a Data Mining System
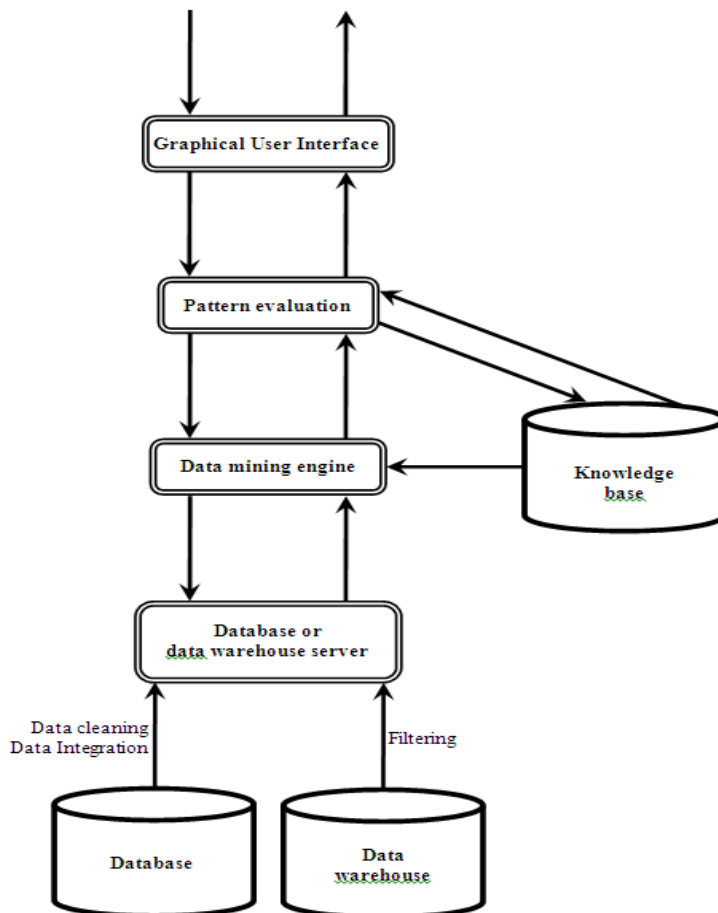
The architecture of a typical data mining system may have the following major components .

- **Database, data warehouse, or other information repository:** This is one or a set of databases, data warehouses, spreadsheets, or other kinds of information repositories. Data cleaning and data integration techniques may be performed on the data.

- **Database or data warehouse server:** The database or data warehouse server is responsible for fetching the relevant data, based on the user's data mining request.
- **Knowledge base:** This is the domain knowledge that is used to guide the search, or evaluate the interestingness of resulting patterns. Such knowledge can include concept hierarchies, used to organize

204

attributes or attribute values into different levels of abstraction. Knowledge such as user beliefs, which can be used to assess a pattern's interestingness based on its unexpectedness, may also be included. Other examples of domain knowledge are additional interestingness constraints or thresholds, and metadata (e.g., describing data from multiple heterogeneous sources).

- **Data mining engine:** This is essential to the data mining system and ideally consists of a set of functional modules for tasks such as characterization, association, classification, cluster analysis, and evolution and deviation analysis.

- **Pattern evaluation module:** This component typically employs interestingness measures and interacts with the data mining modules so as to focus the search towards interesting patterns. It may use interestingness thresholds to filter out discovered patterns. Alternatively, the pattern evaluation module may be integrated with the mining module, depending on the implementation of the data mining method used. For efficient data mining, it is highly recommended to push the evaluation of pattern interestingness as deep as possible into the mining process so as to confine the search to only the interesting patterns.

- **Graphical user interface:** This module communicates between users and the data mining system, allowing the user to interact with the system by specifying a data mining query or task, providing information to help focus the search, and performing exploratory data mining based on the intermediate data mining results. In addition, this component allows the user to browse database and data warehouse schemas or data structures, evaluate mined patterns, and visualize the patterns in different forms.

Functions of Data Mining

Data mining identifies facts or suggests conclusions based on sifting through the data to discover either patterns or anomalies. Data mining has five main functions:

- **Classification:** infers the defining characteristics of a certain group (such as customers who have been lost to competitors).

- **Clustering:** identifies groups of items that share a particular characteristic. (Clustering differs from classification in that no predefining characteristic is given in classification.)

- **Association:** identifies relationships between events that occur at one time (such as the contents of a shopping basket).

- **Sequencing:** similar to association, except that the relationship exists over a period of time (such as repeat visits to a supermarket or use of a financial planning product).

- **Forecasting:** estimates future values based on patterns within large sets of data (such as demand forecasting).

Data Mining Applications

The areas where data mining has been applied recently include:

- Science
    - astronomy,
    - bioinformatics,
    - drug discovery, ...
- Business
    - advertising,
    - Customer modeling and CRM (Customer Relationship management)
    - e-Commerce,
    - fraud detection
    - health care, ...
    - investments,
    - manufacturing,
    - sports/entertainment,
    - telecom (telephone and communications),
    - targeted marketing,
- Web:
    - search engines, bots, ...
- Government
    - anti-terrorism efforts
    - law enforcement,
    - profiling tax cheaters

One of the most important and widespread business applications of data mining is Customer Modeling, also called Predictive Analytics. This includes tasks such as

- predicting attrition or churn, i.e. find which customers are likely to terminate service
- targeted marketing:
    - customer acquisition - find which prospects are likely to become customers
    - cross-sell - for given customer and product, find which other product(s) they are likely to buy
- credit-risk - identify the risk that this customer will not pay back the loan or credit card
- fraud detection - is this transaction fraudulent?

The largest users of Customer Analytics are industries such as banking, telecom, retailers, where businesses with large numbers of customers are making extensive use of these technologies.