Chapter 1
File Organization

1.0 OBJECTIVES

This chapter deals with file organization. Here the various file structures used in a database environment and their relative merits for retrieval and update operations are presented.

After studying this chapter you will be able to –

- Explain storage devices characteristics

- Discuss file organization

1.1 INTRODUCTION

This chapter gives a brief discussion about different file organizations like sequential, index, hash files etc. It also introduces us to various storage devices & their characteristics.

1.2 STORAGE DEVICES CHARACTERISTICS

Presently, the common secondary storage media used to store data are disk and tape.

Tape is generally used for archival data. The storage medium used in a disk is a **disk pack.** It is made up of number of surfaces. Data is read and written from the disk pack by means of transducers called **read/write heads.** The number of read/write heads depends on the type of the disk drive. If we trace projection of one head on the surface associated with it as the disk rotates, we would create a circular figure called **track.** The tracks at the same position on every surface of the disk form the surface of an imaginary cylinder. In disk terminology, therefore, a cylinder consists of the tracks under the heads on each of its surfaces.

If one type of disk drive each track on each surface has a dedicated stationary head, which does not move. Such a disk drive is called a **fixed head drive.** The other type of disk drive is a **moving head drive,** wherein a single head is used for each surface. When data from a given track is to be read, the head is moved to the track.

The disk stores data along concentric tracks. It takes some time for the read /writes head of a moving head disk drive to move from track (or cylinder) to track. This is called the seek time. (For a fixed head disk, the seek time is 0). In the case of a moving head drive, the seek time depends on the distance between the current head and the target head positions. Typical values are from 10 to 50 msec (msec =1/1000 sec). If a file consist of c consecutive cylinders and we assume uniform and random distribution of requests for the different cylinders, we can show that the average distance (in cylinders) the head moves is c/3.Before data can be read or written the disk has to rotate so that the head is positioned at some point relative to a marked start point. The time needed for the reading and writing to start depends on the rotational delay. On the average, the rotational delay is half the rotation time, that is, the time for the disk to rotate once. The rotational delay is called **latency time.** For a drive that rotates at 3600 revolutions per minute, the average latency time is 8.33 msec. The **access time,** therefore, depends on the seek time and the latency time.

On magnetic tapes, data blocks are separated by **interblock gaps (IBG).** The IBG can be attributed to the deceleration /acceleration (stop/start) that takes place

between successive block reads. This only happens when, after a single access, time is needed to process the data before a second read. When continuously scanning over the data, there is no need to start/stop after reading each block. The IBG is also scanned at the faster rate. The typical value for the IBG is 0.6 inch. The access time, i.e. the time required to locate the target block on a magnetic tape, depends on the distance between the current and target blocks.

As we see from the above, the access time depends on the distance between the current and target positions for both types of storage devices. This time can be optimized by suitably placing records. It can also be affected by the file organization employed.

We can abstract the disk storage medium in terms of a two dimensional array and the tape as a one-dimensional array of data blocks. Note that in both cases we can specify a unique address for a block (or physical record). We will omit details of these physical mediums on which the files are stored. These details can be found in most elementary texts in computer organization. It is sufficient to say that some file organizations may be unsuitable for some mediums.

A block of data may contain one or more logical records (henceforth unless otherwise stated, a record denotes a logical record), or a single record may be split across more than one block. Therefore in addition to the block address, we require data on the start position, length, and pointer to continuation block for every record. As different blocks may be assigned to different files, a mapping must be maintained to record the particular files to which the blocks are assigned.

The importance of the study of file organization methods is that different methods have different properties and hence efficiencies for different applications. A major factor that determines the overall system performance is response time for data on secondary storage. This time depends not only on physical device characteristics, but also on the data arrangement and request sequencing. In general; the response cost has two components: access time and data transfer time. **Data transfer time** is the time needed to move data from the secondary storage device to processor memory; **access time** is the time needed to position the read/write head at the required position. The data transfer time depends on physical device characteristics and cannot be optimized. In the case of reading a 1KB (kilobyte=1024 bytes) block of data from a device that can transfer it at 100KB/sec (KB/sec =kilobyte/second), the data transfer time is 10msec. The response time, while influenced by physical characteristics, depends on the distance between current and target positions and therefore on data organization

1.2 Check Your Progress
A) Fill in the blanks
1) ……………………………. is the time needed to move data from the secondary storage device to Processor memory
2) ………………………… is the time needed to position the read/write head at the required position.

B) State true or false
1) The data transfer time depends on physical device characteristics and cannot be optimized.
2) Tape is generally used for archival data

1.3 FILE ORGANIZATION

A file is a collection of or log of records. Having stored the records in a file it is necessary to access these records using either a primary or secondary key. The type and frequency of access required determines the type of file organization to be used for a given set of records.

A File is organized logically as a sequence of records. These records are mapped onto disk blocks .Files are provided as a basic construct in operating systems, so we shall assume the existence of an underlying file system. We need to consider ways of representing logical data models in terms of files.

Although blocks are of a fixed size determined by the physical properties of the disk and by the operating system, record sizes vary. In a relational database, tuples of distinct relations are generally of different sizes.

One approach to mapping the database to files is to use several files and to store records of only one fixed length in any given file. An alternative is to structure our files so that we can accommodate multiple lengths for records; however, files of fixed length records are easier to implement than are files of variable length records. Many of the techniques used for the former can be applied to the variable length case. Thus we begin by considering a file of fixed length records.

Fixed Length Records

As an example, let us consider a file of *account* records for our *bank* database .Each record of this file is defined (in pseudo code) as:

```
type deposit     =     record
                       Account _number char (10);
                       Branch_name char (22);
                       Balance numeric (12, 2);
end
```

If we assume that each character occupies 1 byte and that numeric (12, 2) occupies 8 bytes, our account record is 40 bytes long. A simple approach is to use the first 40 bytes for the first record, the next 40 bytes for the second record and so on.

However there are two main problems with this simple approach.

1)   It is difficult to delete a record from this structure. The space occupied by the record to be deleted must be filled with some other record of the file, or we must have a way of marking deleted records so that they can be ignored.

2)   Unless the block size happens to be a multiple of 40(which is unlikely), some records will cross block boundaries. That is, part of the record will be stored in one block and part in another. It would thus require two block accesses to read or write such a record.

When a record is deleted, we could move the record that came after it into the space formerly occupied by the deleted record, and so on, until every record following the deleted record has been moved ahead. Such an approach requires moving a large number of records. It might be easier simply to move the final record of the file into the space occupied by the deleted record.

It is undesirable to move records to occupy the space freed by the deleted record, since doing so requires additional block accesses. Since insertions tend to be more frequent than deletions, it is acceptable to leave open the space occupied by the deleted record, and to wait for a subsequent insertion before reusing the space. A simple marker on the deleted record is not sufficient, since it is hard to find this available space when an insertion is being done. Thus we need to introduce an additional structure.

At the beginning of the file, we allocate a certain number of bytes as a file header. The header will contain a variety of information about the file.

For now, all we need to store there is the address of the first record whose contents are deleted. We use this first record to store the address of the second available record, and so on. Intuitively we can think of these stored addresses as pointers, since they point to the location of a record. The deleted records thus form a linked list, which is often referred to as a free list.

On insertion of a new record, we use the record pointed to by the header. We change the header pointer to point to the next available record. If no space is available, we add the new record to the end of the file.

Insertion and deletion for files of fixed length records are simple to implement, because the space made available by a deleted record is exactly the space needed to insert a record. If we allow records of variable length in a file, this match no longer holds. An inserted record may not fit in the space left free by a deleted record, or it may fill only part of that space.

Variable Length Records

Variable length records arise in the database systems in several ways.

-      Storage of multiple record types in a file
-      Record types that allow variable lengths for one or more fields.
-      Record types that allow repeating fields, such as arrays or multisets.

Different techniques for implementing variable length records exist.

The **slotted page structure** is commonly used for organizing records within a block. There is a header at the beginning of each block, containing the following information.

1)   The number of record entries in the header.
2)   The end of free space in the block
3)   An array whose entries contain the location and size of each record.

The actual records are allocated contiguously in the block, starting from the end of the block. The free space in the block is contiguous, between the final entry in the header array, and the first record. If a record is inserted, space is allocated for it at the end of free space, and an entry containing its size and location is added to the header.

If a record is deleted, the space that it occupies is freed, and its entry is set to deleted (its size is set to -1, for example). Further the records in the block before the deleted records are moved, so that the free space created by the deletion gets occupied, and all free space is again between the final entry in the header array and the first record. The end of free space pointer in the header is appropriately updated as well. Records can be grown or shrunk by similar techniques, as long as there is space in the block. The cost of moving the records is not too high, since the size of a block is limited: a typical value is 4 kilobytes.

The slotted page structure requires that there be no pointers that point directly to records. Instead, pointers must point to the entry in the header that contains the actual location of the record. This level of indirection allows records to be moved to prevent fragmentation of space inside a block, while supporting indirect pointers to the record.

Databases often store data that can be much larger than a disk block .For instance, an image or an audio recording may be multiple megabytes in size, while a video object may be multiple gigabytes in size. Recall that SQL supports the types blob and clob, which store binary and character large objects.

Most relational databases restrict the size of a record to be no larger than the size of a block, to simplify buffer management and free space management. Large objects are often stored in a special file (or collection of files) instead of being stored with the other (short) attributes of records in which they occur. Large objects are often represented using B+ - tree file organizations.

Organization of Records in a File

So far, we have studied how records are represented in a file structure. A relation is a set of records. Given a set of records; the next question is how to organize them in a file. Several of the possible ways of organizing records in files are:

Heap File Organization

Any record can be placed anywhere in the file where there is space for a record. There is no ordering of records. Typically, there is a single file for each relation.

Sequential File Organization

Records are stored in sequential order, according to the value of a "search key" of each record.

Hashing File Organization

A hash function is computed on some other attribute of each record. The result of the hash function specifies in which block of the file the record should be placed.

1.3.1 Sequential File Organization

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. 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, an average access to half the records in the file. A binary2 or logarithmic search technique may also be used to search for a record.

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 to the new file. Following this, the remaining records in the original file are copied to the new file, thus creating an automatic back-up copy.

The basic advantage offered by a sequential file is the ease of access to the next record, the simplicity of the organization and the absence of auxiliary data structures. To reduce the cost per update, all updates are batched, sorted in order of sequential file and the file is updated in single pass, such a file containing updates to be made to sequential file is sometimes referred to as 'Transaction File'

Thus a sequential file is designed for efficient processing of records in sorted order based on some search key. A search key is any attribute or set of attributes; it need not be the primary key, or even a super key. To permit fast retrieval of records in search key order, we chain together records by pointers. The pointer in each record points to the next record in the search order. Furthermore, to minimize the number of block accesses in sequential file processing, we store records physically in search key order, or as close to search key order as possible.

The sequential file organization allows records to be read in sorted order, that can be useful for display purposes as well as for certain query processing algorithms.

It is difficult, however to maintain physical sequential order as records are inserted and deleted, since it is costly to move many records as a result of a single insertion or deletion. We can manage deletion by using pointer chains, as we saw previously. For insertion, we apply the following rules.

1) Locate the record in the file that comes before the record to be inserted in search key order.
2) If there is a free record (that is, space left after a deletion) within the same block as this record, insert the new record there. Otherwise, insert the new record in an overflow block. In either case, adjust the pointers so as to chain together the records in search key order.

1.3.2 Indexing and Methods of Indexing

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

An index is a set of <key, 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.

A sequential (or sorted on primary keys) file that is indexed is called an **index sequential file.** The index provides for random access to records, while the sequential nature of the file provides easy access to the subsequent records as well as sequential processing. An additional feature of this file system is the overflow area. This feature provides additional space for record addition without necessitating the creation of a new file.

In index sequential organization, it is the usual practice to have a hierarchy of indexes with the lowest level index pointing to the records while the higher level ones point to the index below them.

Updates to an index sequential file may entail modifications to the index in addition to the file The index file can be simplified or its storage requirements reduced if only the address part of the <key, address> pair is held in the index. This however necessitates holding the address of every possible key in the key range, including addresses of records not in file. The addresses of nonexistent records can be set to an impossibly high or low value to indicate their absence from the file. If the number of such missing records in the range of stored key values is small, the saving obtained by not storing the key is considerable.

1.3.3 Hash Files

In the index sequential file organization considered, the mapping from the search key value to the storage location is via index entries. In direct file organizations, the key value is mapped directly to the storage location, avoiding the use of indices. The usual method of direct mapping is by performing some arithmetic manipulation of the key value. This process is called **hashing.**

However hashing schemes usually give rise to collisions when two or more distinct key values are mapped to the same value. Collisions are handled in a number of ways .The colliding records may be assigned to the next available free space or they may be assigned to **overflow area.**

In using the hash function to generate a value, which is the address of a **bucket** where the <key, address> pair values of records are stored, we can handle limited collisions as well as re-organizations of the file without affecting the hash function. In **extendable hashing;** the database size changes are handled by splitting or coalescing buckets.

1.3 Check your progress

A)   Fill in the blanks

1)   A file is a collection of or …………………….. of records.

2)   However hashing schemes usually give rise to …………………….. when two or more distinct key. Values are mapped to the same value.

B)   State True or false

1)   A sequential file could be stored on a sequential storage device such as a magnetic tape.

2)   In a relational database, tuples of distinct relations are generally of same sizes.

1.4 SUMMARY

A file is a collection or bag of records. Having stored the records in a file, it is necessary to access these records using either a primary or secondary key. The type and frequency of access required determines the type of file organization to be used for a given set of records. In this chapter we looked at some common file organizations: Sequential, Index sequential, direct etc.

In a sequential file, records are maintained in the logical sequence of their primary key value. The search for a given record requires, on average, access to half the records in the file. Update operations, including the appending of a new record, require creation of a new file. Updates could be batched and a transaction file of updates used to create a new master file from the existing one .This scheme automatically creates a backup copy of the file.

Access to a sequential file can be enhance by creating an index. The index provides random access to records and the sequential nature of the file provides easy access to the next record. To avoid frequent reorganization, an index sequential file uses overflow areas. This scheme provides space for the addition of records without the need for the creation of a new file. In index sequential organization, it is the usual practice to have a hierarchy of indexes with the lowest level index pointing to the records while the higher level ones point to the index below them.

In direct file organization the key value is mapped directly or indirectly to a storage location, avoiding the use of indices. The usual method of direct mapping is by some arithmetical manipulation of the key value, the process is called hashing.

1.5 CHECK YOUR PROGRESS-ANSWERS

1.2
A) 1) Data transfer time
2) Access time

B) 1) True
2) True

1.3
A) 1) Log
2) Collisions

B) 1) True
2) False

1.6 QUESTIONS FOR SELF - STUDY

1) Discuss the differences between the following file organizations
- Sequential
- Index sequential
- Hash files
2) Write a note on storage devices characteristics.

❖ ❖ ❖

NOTES

Chapter 2
Introduction to Database Systems

2.0 OBJECTIVES

In this chapter the basic concepts of the database systems are introduced. The structure of a Database Management System and its components are presented.

After studying this chapter you will be able to –

- Explain DBMS

- Discuss Advantages and drawbacks of DBMS.

- Describe Architecture of DBMS.

- Explain users of DBMS.

- Describe Components of DBMS

- State Features of RDBMS .

- Describe CODD's Rules.

2.1 INTRODUCTION

2.1.1 What is Data, Database System, DBMS?

Database:

It is fair to say that databases will play a critical role in almost all areas where computers are used including business, engineering, medicine, law, education and library science etc. The term Database is a collection of related data with an implicit meaning.

By Data, we mean known facts that can be recorded and that have implicit meaning. Consider the names, telephone numbers, addresses of the people you know. We may record this data in an indexed address book, or diskette, using a personal computer and software such as FoxPro, Excel, and Access etc.

A database has the following implicit properties

1)   A database represents some aspect of the real world, sometimes called the mini world or the Universe of Discourse (UOD). Changes to the mini world are reflected in the database.

2)   A database is logically coherent collection of data with some inherent meaning. A random assortment of data cannot correctly be referred to as a database.

3)   A database is designed, built and populated with data for a specific purpose. It has an intended group of users and some preconvinced applications in which these users are interested.

Thus Database is
- Collection of interrelated data

- Set of programs to access the data

- DBMS contains information about a particular enterprise

- DBMS provides an environment that is both *convenient* and *efficient* to use.

- Database Applications:

    - ★ Banking: all transactions

    - ★ Airlines: reservations, schedules

    - ★ Universities: registration, grades

    - ★ Sales: customers, products, purchases

    - ★ Manufacturing: production, inventory, orders, supply chain

    - ★ Human resources: employee records, salaries, tax deductions

- Databases touches all aspects of our lives

A database can be of any size and varying complexity. It may contain a few hundred records or million. This huge amount of information must be organized and managed so that users can search for retrieve and update the data as needed

A database may be generated and maintained manually or by machine. The library card catalog is an example of database that may be manually created and maintained. A computerized database may be created and maintained either by a group of application programs written specifically for that task or by a database management system.
An organization must have accurate and reliable data for effective decision making. To this end, the organization maintains records on the various facets of its operations by building appropriate models of diverse classes of object of interest. These models capture

The essential properties of the objects and records relationships among them. Such related data is called database. A database system is an integrated collection of related files, along with details of interpretation of the data contained therein. DBMS is a s/w system that allows access to data contained in a database. The objective of the DBMS is to provide a convenient and effective method of defining, storing and retrieving the information contained in the database.

The DBMS interfaces with application programs so that the data contained in the database can be used by multiple applications and users. The DBMS allows these users to access and manipulate the data contained in the database in a convenient and effective manner. In addition the DBMS exerts centralized control of the database, prevents unauthorized users from accessing the data and ensures privacy of data.

A DBMS (Database Management System) is a collection of programs that enables users to create and maintain a database. A DBMS is hence a general purpose s/w system that facilitates the process of *Defining, Constructing* and *Manipulating* databases for various applications.

Defining a database involves specifying the data types, structures and constraints for the data to be stored in the database (I. specifying different types of data elements to be stored in each record)

Constructing the database is the process of storing the data itself on some storage medium that is controlled by the DBMS.

Manipulating a database includes such functions as querying the database to retrieve the specific data, updating the database to reflect changes in the mini world and generating reports from the data .Data manipulation involves querying and updating. These informal queries and updates must be specified precisely in the database system language before they can be processed. Most medium and large size databases include many types of records and have many relationships among records.

Database system is basically a computerized record keeping system i.e. a computerized system whose overall purpose is to store information and to allow users to retrieve and update that information on demand. The data/Information in question can be anything that is of significance to the individual or organization concerned. - Anything in other words that is needed to assist in the general process of running the business of that individual or organization.

Database is a repository or container for a collection of computerized data files.

Users of the system can perform a variety of operations on such files.

1)   Adding new, empty files to the database.

2)   Inserting data into existing files.

3)   Retrieving data from existing files

4)   Changing data in existing files

5)   Deleting data from existing files

6)   Removing existing files from database

A database is a collection of **persistent data** that is used by the application systems of some given enterprise. By persistent data we mean – Data can be subsequently be removed from the database only by some explicit request to the DBMS, not as a mere side effect of some program completing execution.

Enterprise may be Manufacturing company, Bank, Hospital, University, Government Department
Eg. Of persistent data can be
        1) Product data   2) Account data
        3) Patient data   4) Student data
        5) Planning data

It is meant to show that a database system involves 4 major components: Data, Hardware, Software and users

Database systems are available on machines that range all the way from the smallest personal computers to the largest mainframes.

2.1.2 Single and Multi-user Systems
In particular, systems on large machines ("large systems") tend to be multi user, whereas those on smaller machines ("small systems") tend to be single user.

A single user system is a system in which at which at most one user can access the database at any given time.

A multi user system is a system in which many users can access the database at any given time (Same time).

A major objective of multi user system in general is precisely to allow each user to behave as if he or she were working with a single user system instead.
2.1.3 Advantages and Drawbacks Of DBMS
Advantages of DBMS

1) The organization can exert via the DBA, a centralized management and control over the data. The database administrator is the focus of the centralized control. Any application requiring a change in the structure of a data record requires an arrangement with the DBA, who makes the necessary modifications .Such modifications do not affect other applications or users of the record in question. Therefore these changes meet another requirement of the DBMS : data independence

2) **Reduction of Redundancies:**
Centralized control of data by the DBA avoids unnecessary duplication of data and effectively reduces the total amount of data storage required. It also eliminates the extra processing necessary to trace the required data in a large mass of data. Another advantage of avoiding duplication is the elimination of the inconsistencies that tend to be present in redundant data files. Any redundancies that exist in the DBMS are controlled and the system ensures that these multiple copies are consistent.

3) **Shared Data:**
A database allows the sharing of data under its control by any number of application programs or users.

Eg. The application for the public relations and payroll departments could share the data contained for the record type EMPLOYEE

4) **Integrity:**
Centralized control can also ensure that adequate checks are incorporated in the DBMS to provide data integrity. Data integrity means that the data contained in the database is both accurate and consistent. Therefore data values being entered for storage could be checked to ensure that they fall within a specified range and are of correct format. Eg. The value for the age of an employee may be in the range of 16 to 65.

Another integrity check that should be incorporated in the database is to ensure that if there is a reference to certain object, that object must exist.

In the case of an automatic teller machine, for eg. A user is not allowed to transfer funds from a nonexistent savings account to checking account.

5) **Security**
Data is of vital importance to an organization and may be confidential. Such confidential data must not be accessed by unauthorized persons.

The DBA who has the ultimate responsibility for the data in the DBMS can ensure that proper access procedures are followed, including proper authentication schemes for access to the DBMS and additional checks before permitting access to sensitive data.

Different levels of security could be implemented for various types of data and operations. The enforcement of security could be data value dependent (eg. A manager has access to the salary details of the employees in his or her department only) as well as data-type dependent (but the manager cannot access the medical history of any employee, including those in his or her department).

6) **Conflict Resolution**
Since the database is under the control of DBA, he could resolve the conflicting requirements of various users and applications. In essence, the DBA chooses the best file structure and access method to get optimal performance for the response critical applications, while permitting less critical applications to continue to use the database, with a relatively slower response.

7) **Data Independence**
Is usually considered from 2 points of view
- Physical data independence
- Logical data independence

Physical data Independence

It allows changes in the physical storage devices or organization of files to be made without requiring changes in the conceptual view or any of the external views and hence in the application programs using the database. Thus the files may migrate from one type of physical media to another or the file structure may change without any need for changes in the application programs.

Logical Data Independence

It implies that application programs need not be changed if fields are added to an existing record, nor do they have to be changed if fields not used by application programs are deleted. Logical data independence indicates that the conceptual schema can be changed without affecting the existing external schemas. Data independence is advantageous in the database environment since it allows for changes at one level of the database without affecting other levels. These changes are absorbed by the mappings between the levels.

Disadvantages of DBMS
1) Significant disadvantage of DBMS is cost.
2) In addition to cost of purchasing or developing the software, the hardware has to be upgraded to allow for the extensive programs and work spaces required for their execution and storage.
3) The processing overhead introduced by the DBMS to implement security, integrity and sharing of the data causes a degradation of the response and through put times.

4) An additional cost is that of migration from a traditionally separate application environment to an integrated one.

5) While centralization reduces duplication, the lack of duplication requires that the database be adequately backed up so that in the case of failure the data can be recovered. Backup and recovery operations are fairly complex in the DBMS environment. Furthermore a database system requires a certain amount of controlled redundancies and duplication to enable access to related data items.
6) Centralization also means that the data is accessible from a single source namely the database. This increases the potential severity of security branches and disruption of the operation of the organization because of downtimes and failures.
7) The replacement of a monolithic centralized database by a federation of independent and cooperating distributed databases resolves some of the problems resulting from failures and downtimes.

2.1.4 Architecture of DBMS
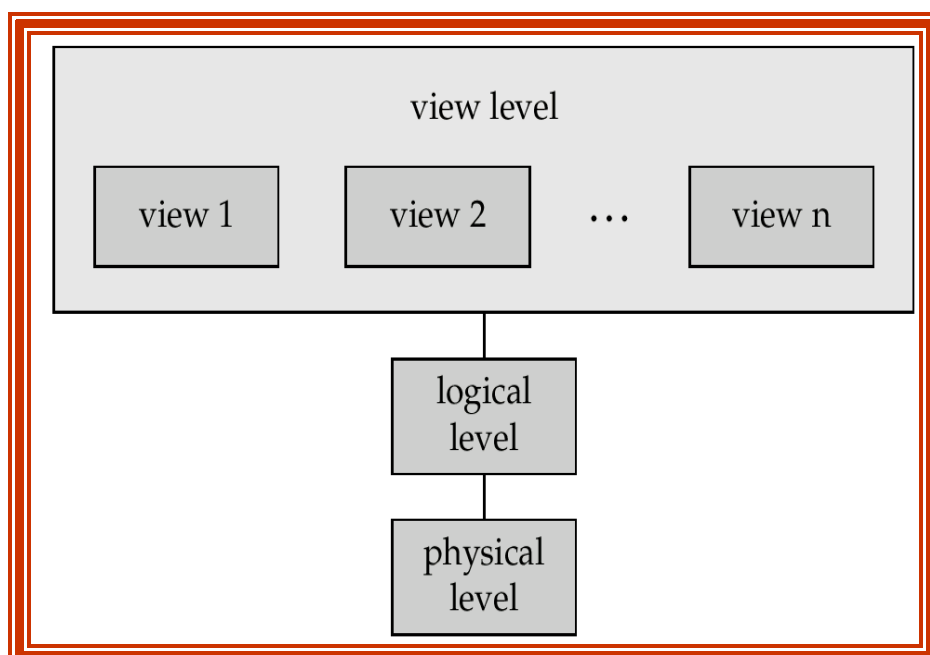
The 3 levels of architecture

1)   The Internal level (Also known as Physical Level)
It is the closest level to the physical storage (i.e. One concerned with the way the data is physically stored). It is at a low-level representation of the entire database. It consists of many occurrences of each of the many types of internal record.

The internal view is described by means of the internal schema, which not only defines the various stored record types but also specifies what indexes exists, how stored fields are represented, what physical sequence the stored records are in and so on.

2)   The Conceptual level (Also known as the community logical level, or sometimes just the Logical level)
It is the level of indirection between the other two levels. It is a representation of the entire information content of the database .Broadly speaking, the conceptual view is intended to be a view of the data "as it really is" rather than as users are forced to see it by the limitations of the particular language or hardware they might be using.

The conceptual view consists of many occurrences of each of many types of conceptual record. The conceptual view is defined by means of the conceptual schema, which includes definitions of each of the various conceptual record types. The conceptual schema is written using another data definition language, the conceptual DDL.

The conceptual view then is a view of the total database content, and the conceptual schema is a definition of that view. The definitions in the conceptual schema are intended to include a great many additional features such as security and integrity constraints.

3)   The External level (Also known as the User Logical Level)
It is the closest level to the users – It is the one concerned with the way the data is seen by individual users. In other words there will be many distinct "external views", each consisting of a more or less abstract representation of some portion of the total database. It is individual user level. A given user can be either an application programmer or an end user of any degree of sophistication.

For application programmer, that language will be either conventional programming language (E.g. c++, Java) or else a proprietary language that is specific to the system in question. Such proprietary languages are often called Fourth Generation Languages (4GL).

For the end user, the language will be either a query language or some special purpose language perhaps form or menu driven, tailored to that user's requirements and supported by some online application program. The important thing about all such languages is that they will include a data sublanguage.

I.e. A subset of the total language that is concerned specifically with database objects and operations. The data sublanguage is said to be embedded within the corresponding host language.

In principle any given data sublanguage is really a combination of at least 2 subordinate languages – Data definition language (DDL) which supports the definition or declaration of database objects, and data manipulation language which supports the manipulation or processing of some objects. The external view is defined by means of external schema. The external schema is written using the DDL portion of the user's Data sublanguage.

2.1.5 Users of DBMS

The users of a database system can be classified according to the degree of expertise or the mode of their interactions with the DBMS.

1. **Naive Users**

   Users who need not be aware of the presence of the database system or any other system supporting their usage are considered naïve users.

   A user of an automatic teller machine falls in this category. The operations that can be performed by these types of users are very limited and affect a precise portion of the database.

   Other such users are end users of the database who work through menu oriented application program where the type and range of response is always indicated to the user.

2. **Online Users**

   These are the users who may communicate with the database directly via an online terminal or indirectly via a user interface and application program.

   These users are aware of the presence of database system and may have acquired a certain amount of expertise in the limited interaction with the database through the application program.

   The more sophisticated of these users may also use a data manipulation language to manipulate the database directly. Online users can also be naïve users requiring additional help such as menus.

3. **Application Programmers**

   Professional programmers who are responsible for developing application programs or user interfaces utilized by the naïve and online users fall into this category. The application program could be written in a general purpose programming language such as C, Assembler, FORTRAN, Pascal etc. and can include commands required to manipulate the database.

4. **Database Administrator**

   Centralized control of the database is exerted by a person or group of persons under the supervision of high level administrator. This person or group is referred to as the Database Administrator(DBA). They are the users who are most familiar with the database and are responsible for creating, modifying and maintaining its 3 levels.

2.1.6 Roll Of Database Administrator

Centralized control of the database is exerted by a person or group of persons under the supervision of high level administrator. This person or group is referred to as the Database Administrator(DBA). They are the users who are most familiar with the database and are responsible for creating, modifying and maintaining its 3 levels.

1) The DBA is the custodian of data and controls the database structures. The DBA administers the 3 levels of the database and in consultation with the overall user community, sets up the definition of the global view or conceptual level of the database.

2) The DBA further specifies the external view of the various users and application and is responsible for the definition and implementation of the internal level including the storage structure and access methods to be used for the optimum performance of the DBMS.

3) Changes to any of the 3 levels necessitated by changes or growth in the organization or emerging technology are under the control of DBA

4) Mappings between the internal and conceptual levels, as well as between the conceptual and external levels are also defined by the DBA

5) Ensuring that appropriate measures are in place to maintain the integrity of the database and that the database is not accessible to unauthorized users is another responsibility.

6) The DBA is responsible for granting permission to the users of the database and stores the profile of each user in the database .This profile describes the

permissible activities of a user on that portion of the database accessible to the user via one or more user views. The user profile can be used by the database system to verify that a particular user can perform a given operation on the database.

7) The DBA is also responsible for defining procedures to recover the database from failures due to human, natural, hardware causes with minimal loss of data. This recovery procedure should enable the organization to continue to function and the intact portion of the database should continue to be available.

8) The Data Administrator (DA) is the person who makes the strategic and policy decisions regarding the data of the enterprise and the Database Administrator (DBA) is the person who provides the necessary technical support for implementing these decisions. Thus the DBA is responsible for the overall control of the system at a technical level.

2.1 Check Your Progress

A) Fill In The Blanks

1) A database is logically coherent collection of data with some

……………………. meaning.

2) DBMS is a ……………………. that allows access to data contained in a database.

3) DBMS stands for ……………………

4) A ……………………. is a system in which many users can access the database at any given time (Same time).

5) Data integrity means that the data contained in the database is both …………………… and Consistent

6) Data Independence is of 2 types, physical data independence and ……………………

7) …………………… means that the data is accessible from a single source namely the database

8) The internal level is also known as ……………………

9) DDL stands for ……………………

10) A user of an automatic teller machine are categorized as ……………………

B) State True or False

1) A single user system is a system in which many users can access the database at any given time (Same time).

2) The internal level is also known as physical level

3) Data independence is of two types, physical and logical data independence

4) DBA stands for database access

5) Conceptual level is also known as logical level.

Overall system structure
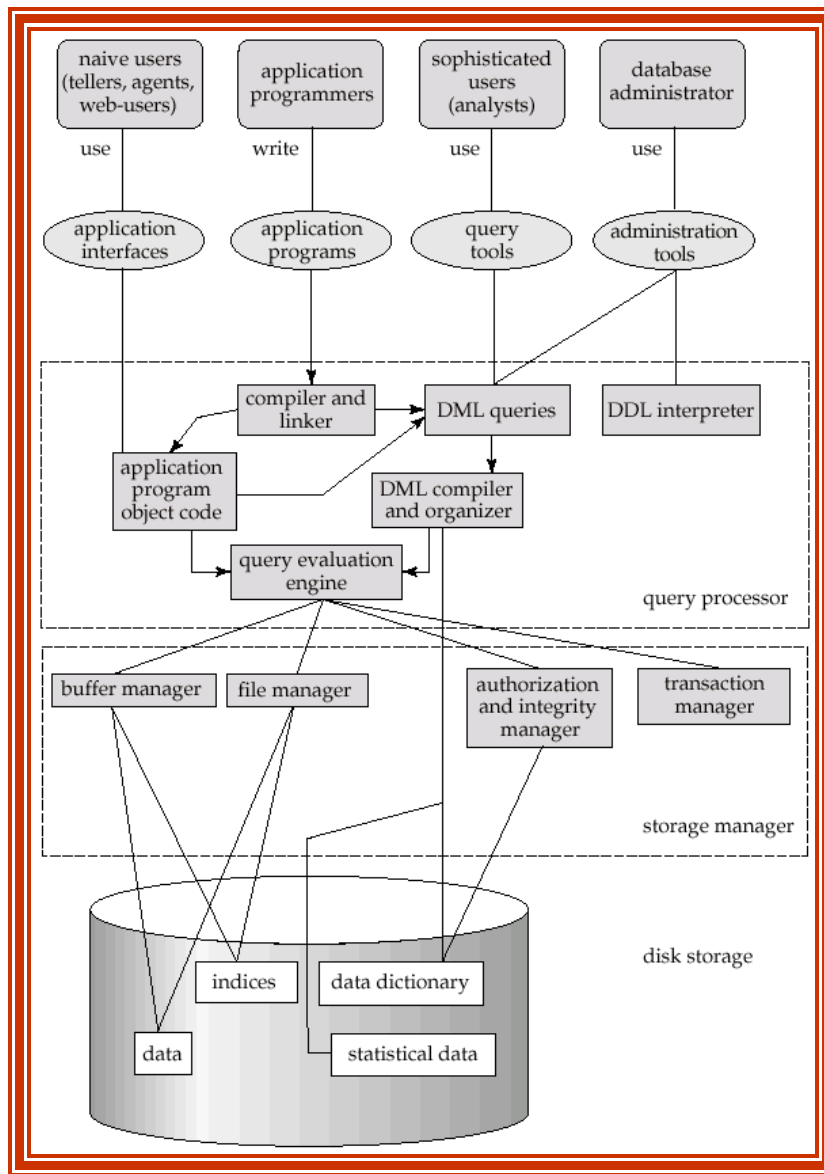
A database system is partitioned into modules that deal with each of the responsibilities of the overall system. The **<u>functional components</u>** of a database system are broadly divided into **the storage manager** and **query processor** components.

Storage Manager

The storage manager is important because databases typically require a large amount of storage space. Corporate databases range in size from hundreds of gigabytes to, for the largest databases, terabytes of data. A gigabyte is (1 billion bytes) and terabyte is 1 million megabytes (1 trillion bytes). Since the main memory of computers cannot store this much information, the information is stored on disks. Data are moved between disk storage and main memory as needed. Since the movement of data to and from disk is slow relative to the speed of the CPU. (Which minimizes need to move data from disk and main memory).

A storage manager is a program module that provides the interface between the low level data stored in the database and the application programs and queries submitted to the system. It is also responsible for interaction with the file manager. The raw data are stored on the disk using the file system, which is usually provided by a conventional operating system. The storage manager translates the various DML

statements into low-level file-system commands. Thus the storage manager is responsible for storing, retrieving and updating data in the database

Following are the responsibilities of Database Manager or Storage Manager

1) **Interaction with the file Manager**
Actual data is stored in the file system. The storage manager translates the various DML statements into low level file system commands. This database manager is responsible for actual storing, retrieving and updating of the data in the database.

2) **Integrity Enforcement**
Consistency constraints are specified by Database Administrator. But the responsibility of database manager is to enforce, implement or check these constraints.

3) **Backup and Recovery**
It is the responsibility of database manager to detect system failures and restore the database to a consistent state.

4) **Concurrency Control**
Interaction among the concurrent users is controlled by database manager.
Thus storage manager or database manager is responsible for
- Storing the data
- Retrieving the data
- Updating the data in the database

Storage Manager Components Include

1) *Authorization and Integrity Manager*
-Which tests for the satisfaction of integrity constraints and check the authority of users to access data?

2) *Transaction Manager*
-Which ensures that the database system remains in a consistent state despite system failures, and that concurrent transaction execution proceed without conflicting

3) *File Manager*
-Which manages the allocation of space on disk storage and the data structures used to represent information stored on disk.
Responsibility for the structure of the files and managing the file space rests with the file manager. It is also responsible for locating the block containing the required record, requesting this block from the disk manager, and transmitting the required record to the data manager. The file manager can be implemented using an interface to the existing file subsystem provided by the operating system of the host computer or it can include a file subsystem written especially for the DBMS.

4) *Buffer Manager*
-Which is responsible for fetching data from disk storage into main memory and deciding what data to cache in main memory? The buffer manager is a critical part of the database system, since it enables the database to handle data sizes that are much larger than the size of the main memory.
The storage manager implements several **data structures** as part of the physical system implementation.

Following Data structures are required as a part of the physical system implementation.

1) *Data Files*
-Which stores the database itself. It contains the data portion of the database.

2) *Data Dictionary*
-Which stores metadata about the structure of the database, in particular the schema of the database. Information pertaining to the structure and usage of data contained in the database, the metadata, is maintained in a data dictionary. The term system catalog also describes this metadata. The data dictionary, which is a database itself, documents the data. Each database user

can consult the data dictionary to learn what each piece of data and the various synonyms of the data field means.

In an integrated system (i.e. in a system where the data dictionary is part of the DBMS) the data dictionary stores information concerning the external, conceptual and internal levels of the database. It contains the source of each data field value, the frequency of its use, and an audit trail concerning updates, including the who and when of each update.

3)   ***Indices***
-Which provides fast access to data items that hold particular values

4)   ***Statistical Data***
It stores statistical data /Information about the data in the database. This information is used by query processor to select efficient ways to execute query.

Query Processor

The database user retrieves data by formulating a query in the data manipulation language provided with the database. The query processor is used to interpret the online user's query and convert it into an efficient series of operations in a form capable of being sent to the data manager for execution. The query processor uses the data dictionary to find the structure of the relevant portion of the database and uses this information in modifying the query and preparing an optimal plan to access the database.

Query Processor includes

1)   ***DDL Interpreter***
-Which interprets DDL statements and records the definitions in the data dictionary

2)   ***DML PreCompiler***
-Which translates DML statements in a query language into an evaluation plan consisting of low – level instructions that the query evaluation engine understands. A query can usually be translated into any number of alternative evaluation plans that all give the same result. The DML compiler also performs query optimization, that is, it picks the lowest cost evaluation plan from among the alternatives.

3)   ***Embedded DML Pre compiler***
It converts DML statements embedded in an application program to normal procedure calls in the host language. The pre compiler must interact with the DML compiler to generate the appropriate code.

4)   ***Query Evaluation Engine***
-Which executes low-level instructions generated by the DML compiler.

2.2 Check your progress

A) Fill in the blanks

1)   A gigabyte is ……………………. bytes and terabyte is …………………….. megabytes (1 trillion bytes).

2)   The functional components of a database system are broadly divided into **the Storage Manager** and ……………………. components.

3)   ……………………. means data about data.

4)   Thus the ……………………. is responsible for storing, retrieving and updating data in the database

5)   ……………………. provides fast access to data items that hold particular values

6)   ……………………. interprets DDL statements and records the definitions in the data dictionary

7)   ……………………. executes low-level instructions generated by the DML compiler.

---

8) ……………………. …………….tests for the satisfaction of integrity constraints and check the authority of users to access data.

B) State True or False

1) DDL stands for data designing language

2) Gigabyte is one billion bytes

3) Indices provides fast access to data items that hold particular values

4) Query Evaluation Engine interprets DDL statements and records the definitions in the data dictionary

2.3 TYPES OF DBMS

A model is an abstraction process that hides superfluous details while highlighting details important to the application in hand. **A data model** is a mechanism that provides this abstraction for atabase applications.

Data modeling is used for representing entities of interest and their relationships in the database. It allow the conceptualization of the association between various entities and their attributes. A number of models for data representation have been developed. Most data representation models provide mechanisms to structure data for the entities being modeled and allow a set of operations to be defined on them.

The models can also enforce a set of constraints to maintain the integrity of the data. These models differ in their method of representing the association among entities and attributes.

The main models that we will study are the hierarchical, network and relational models.

Following models are also categorized as **Traditional Data Models.**

The Hierarchical Data Model
Implementing Many to Many Relationships using HDM

(a) E-R diagram

(b) Tree-structure diagram

Implementing Many to Many Relationships using HDM

(a) E-R diagram

(b) Tree-structure diagrams
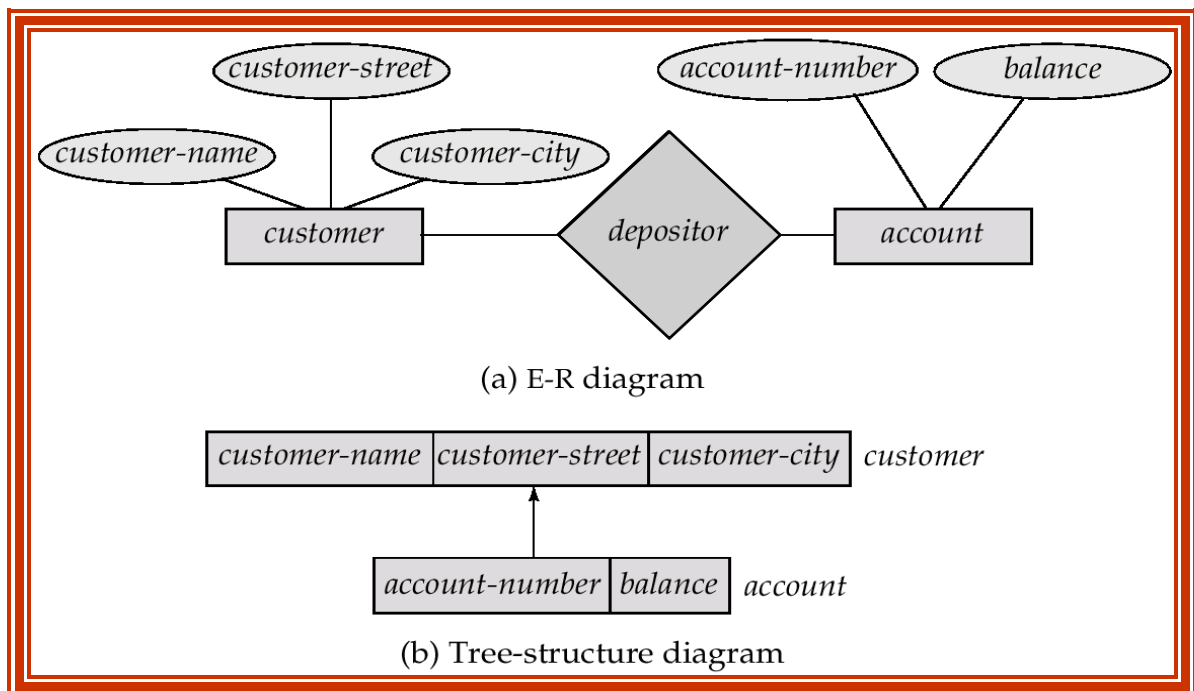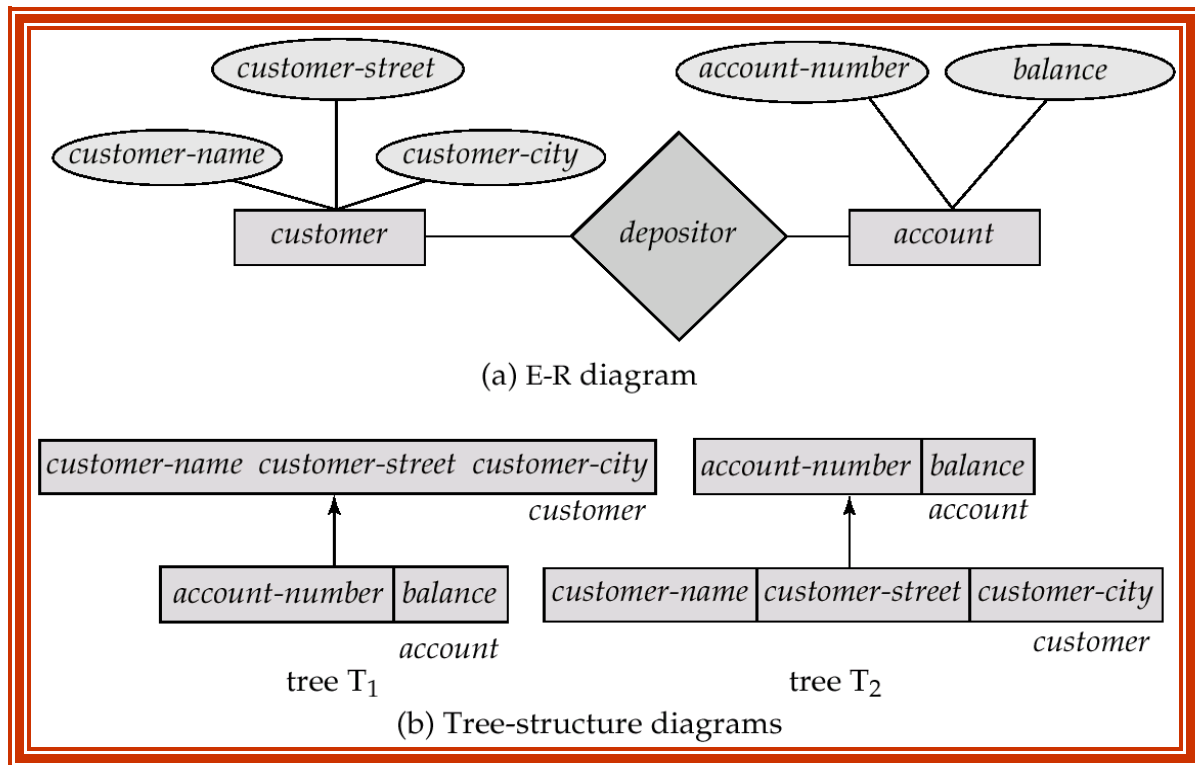
It uses records and pointers or links to represent entities and the relationships among them. Here data structure used is a rooted tree with a strict parent to child ordering (PCR).

A tree may be defined as a set of nodes such that there is one specially designated nodes called the root(node) and the remaining nodes are partitioned into disjoint sets ,each of which in turn is a tree the sub trees of a root. If relative order of the sub trees is significant, the tree is an **ordered tree.**

Like an **organization chart or a family tree** a hierarchy is an ordered tree and is easy to understand. At the root of the tree is the single parent; the parent can have none, one or more children.

In a hierarchical database the data is organized in a hierarchical or ordered tree structure and the database is a collection of such disjoint trees (sometimes referred to as a forest or spanning trees). The nodes of the trees represents record types. Each tree effectively represents a root record type and all of its dependent record types. If we define the root record type to be at level 0, then the level of its dependent record types can be defined as being at level 1. The dependents of the record types at level 1 are said to be at level 2, and so on.

An occurrence of a hierarchical tree type consists of one occurrence of the root record type along with zero or more occurrences of its dependent sub tree types. Each dependent sub tree is in turn hierarchical and consists of a record type as its root node. In a hierarchical model, no dependent record can occur without its parent record occurrence. Furthermore, no dependent record occurrence may be connected to more than one parent record occurrence.

A hierarchical model can represent one-to-many relationships between two entities where the two are respectively parent and child However, to represent a many-to-many relationship requires duplication of one of the record types corresponding to one of the entities involved in this relationship. Note that such duplication could lead to inconsistencies when only one copy of a duplicated record is updated.

Another method of representing a many – to-many relationships is by the use of the virtual record.

Thus HDM uses the tree concept to represent data and the relationship among data. The nodes of the tree are the record types representing entity sets and are connected by pointers or links.

The relationship between the entities is represented by the structure of the resulting ordered tree. A pointer or a link as in the network data model represents relationship between exactly two records .However in the hierarchical model, the relationship is parent to child relationship. .Furthermore hierarchical data model restricts each record type to only one parent record type. A parent record type can have any no of children record types. Two record types in a hierarchical tree can have at most one relationship between them and this relationship is that of one-to-one or one-to-many.

The HDM has following constraints
1) The hierarchical tree can have only one root record type and this record type does not have a parent record type.

2) The root can have any number of child record types each of which can itself be a root of hierarchical sub tree.

3) Each child record type can have only one parent record type, thus many-to-many relationship cannot be directly expressed between two record types.

4) Data in a parent record applies to all its children records.

5) Each occurrence of a record type can have any number of occurrences of each of its child record types.

6) A child record occurrence must have a parent record occurrence, deleting a parent record occurrence requires deleting all its children record occurrences.

7) A hierarchical tree can have any number of record occurrences for each record type at each level of the hierarchical tree.

In the implementation of the hierarchical data model, the pointers are normally from a parent record to child record only

The hierarchical database can be represented by a structure in which the records are represented by rectangular boxes and the relationship between records by links pointing from root towards leaf. The links does not have labels, since the relationship is always of type parent to child. Such structured diagrams are called **tree structured diagrams, definition trees or hierarchical definition trees**.

The hierarchical model provides straightforward or natural method of implementing one to many relationships, However M: N relationship can't be expressed directly in the hierarchical model. Such a relationship can be expressed by using data replication or virtual records.

The disadvantage of data replication are wastage of storage space and the problems of maintaining data consistencies. A virtual record is a mechanism to point to an occurrence of a physical record. Thus instead of replicating record occurrence, a single record occurrence is stored and a virtual record points to this record wherever this record is required. The virtual record can contain some data that is common to a relationship, such data is called as intersection data. The virtual record is the logical child of the physical record, that is it points to, which is its logical parent.

In hierarchical structure diagrams one can assume that the DBMS provides a single occurrence of a dummy record type and all the hierarchical trees can then be attached to this single dummy parent record. The roots of these trees can be treated as children of this dummy record.

The data manipulation facility of the hierarchical model provides functions similar to the network approach; however the navigation to be provided is based on the hierarchical model.

The *get* command is used to retrieve an occurrence of a specified record type that satisfies the specified conditions.

The get *next* command is used for sequential processing *Get next within parent* is used for sequential processing within a preselected hierarchy.

The database can be modified using insert ,replace and delete operations. When records to be modified are virtual records, detailed rules have to be specified so that the modifications if allowed, leaves the database in a consistent state.

Converting an ER diagram to a hierarchical model can be accomplished as follows

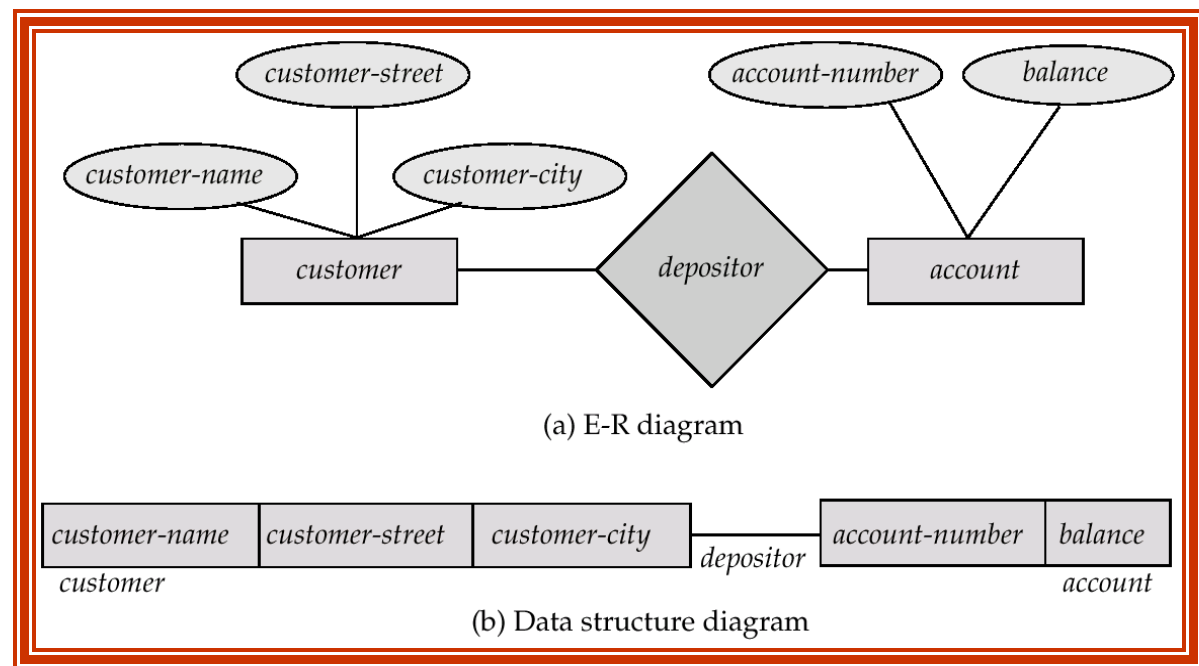Each entity type in the ER diagram is represented by a record type.

A 1: N relationship is represented as a hierarchy type where the record type corresponding to the one side of the relation is parent (1:1 relationship is a special case of 1: N relationship)

A weak entity can be represented as a separate record type. This record type becomes a dependent in a hierarchy where the record type, corresponding to the strong entity in the identifying relationship is the parent.
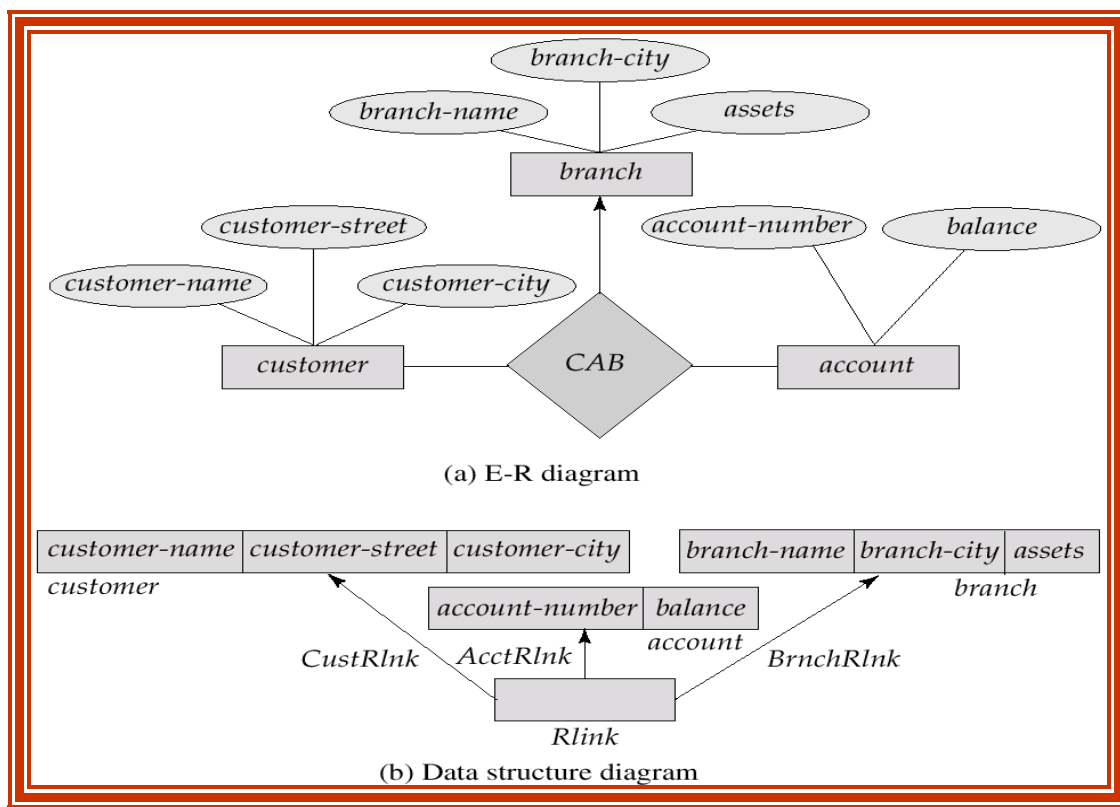
M: N relationships require introducing duplicate record types or using multiple hierarchies and introducing virtual records.

Network Data Model
Implementing one to one relationships using HDM

(a) E-R diagram

customer-name	customer-street	customer-city		account-number	balance
			depositor		

customer   account

(b) Data structure diagram

Implementing Many to Many Relationships using HDM

(a) E-R diagram

(b) Data structure diagram

Implementing Ternary relationship using HDM

The database task group (DBTG), a special group within the conference on Data Systems Language (CODASYL) was changed to define standard specifications for an environment that would facilitate the database creation and data manipulation. DBTG issued a final report in 1971.

In the network model data is represented by collection of records and relationships among data are represented by links. Thus it is called record based model.

A record is in many aspects similar to an entity in ER model. Each record is a collection of fields (attributes), each of which contains only one data value A link is an association between precisely two records. Thus a link can be viewed as a restricted (binary) form of a relationship in the sense of ER model.

Thus network database model was created to represent complex data relationships more effectively than HDM could, to improve database performance and to impose database standard.

The lack of database standards was troublesome to programmers and application designers because it made database designers and applications less portable.

Data Structure Diagrams

Is a scheme representing the design of a network database. Such a diagram consists of 2 basic components.
- Boxes, which corresponds to record types
- Lines, which corresponds to Links

A data structure diagram serves the same purpose as an ER diagram, namely it specifies the overall logical structure of database.

Using Network database terminology, a relationship is called a **set**. Each set is composed of at least two record types, an ***owner record*** that is equivalent to HDM's parent and ***member record*** that is equivalent to the HDM's child.

The difference between HDM and NDM is that the NDM might include a condition in which record can appear (as a member) in more than one set. In other words a member may have several owners.

The basic data definition structure of the DBTG proposal includes records and sets.

Record types are representations of entity types and are made up of *data items, vectors and repeating groups.*

A **set** is a means of representing a 1: N relationship between record types. A set is declared to have one record as the owner record type and one or more records as the member record type's one of the constraints imposed by DBTG proposal, for ease of implementation is that a given record occurrence could be an owner or a member in only one occurrence of a set type.

This restriction means that M: N relationship can only be represented by introducing an intermediate record type and representing M: N relationship by two 1: N relationship.

Data Item:

It is the DBTG term for field and is the smallest unit of named data. An occurrence of a data item is a representation of a value. A data item has a name and a type or format. The format could be arithmetic, string or one specified by the implementer via a TYPE clause.

Vector:

The DBTG record is made up of basic units of data representation called data items. It also allows a data item to be repeated, this is called a vector in DBTG terminology. Suppose that the record type CLIENT contain an additional data item for storing the phone number. However two or more phone numbers for instance, the home and business phone numbers may be required to be stored for some clients. The phone number could then be described as a one dimensional array of data items all of which have identical characteristics (so here phone number is an example of vector)

Repeating Group:

In an example of a relationship between Employee and Dependent, we need to store in the record for each employee, a number of dependents and relationship of the dependent to the employee. This is an example of a repeating group.

The repeating group is a collection of two or more data items, vectors or repeating groups that occurs more than once within a record. A repeating group is thus nested.

A repeating group can be considered to be a vector of records.

Advantages of network model
1) In NDM 1: N, M: M, 1:1 relationships can be implemented. Thus NDM handles more relationship types.
2) Data duplication is avoided.
3) ER to network mapping is easier than in HDM
4) Like the HDM the conceptual view of the database is simple, thus promoting design simplicity.
5) Data access flexibility: It is superior to that found in HDM in the file system. An application can access an owner record and all the member records within a set.
6) Promotes data integrity: The NDM enforces database integrity because the user must first define the owner record and then the member (A member cannot exist without owner)
7) Data Independence: Changes in the data characteristics do not require changes in the data access portions of the application programs.
8) Conformance to standards: The network database's existence is at least partially traceable to the database standards imposed in the seventies. These standards include a DDL thus greatly facilitating administration and portability.

Disadvantages of network model

System complexity -- Database integrity control and the efficiency with which the network model manages relationship are sometimes short circuited by the system complexity. Like the HDM, the NDM provides a navigational data access environment, in which data are accessed one record at a time. Consequently DBA, programmers and

end users must be very familiar with the internal structures in order to access the database.

Lacks of structural Independence -- Some structural changes are impossible to make in a network database. If changes are made to the database structure, all application programs must be revalidated before they can access the database. In short although the network model achieves data independence, it still does not produce structural independence.

The Relational Data Model

The relational model, first developed by E.F.Codd (of   IBM) in 1970 represented a major break for both designers and users. The RDM, after more than a decade has emerged from the research, development, test and trial stages as a commercial product.

Software system using this approach is available for all sizes of computer system. This model has the advantage of being simple in principle and users can express their queries in powerful query language.

2.3 Check your progress

A) Fill in the blanks

1)   PCR stands for ………………………………..

2)   The hierarchical tree can have only ………………………………..root record type

3)   ………………………………..relationship can't be expressed directly in the hierarchical model

4)   A ………………………………..is a mechanism to point to an occurrence of a physical record.

5)   ………………………………..stands for database task group

6)   Using Network database terminology, a relationship is called a …………………

7)   Each set is composed of at least two record types, an ……………………………….. record that is equivalent to HDM's parent and ………………………………..record that is equivalent to the HDM's child

8)   ………………………………..is the smallest unit of named data

9)   ………………………………..also allows a data item to be repeated

10)   ………………………………..can be considered to be a vector of records.

11)   The relational model, first developed by ---------------- (of IBM) in 1970

B) State True or False

1)   The relational model, first developed by E.F.Codd (of IBM) in 1970

2)   The hierarchical tree can have number of root record types

3)   A repeating group can be considered to be a vector of records.

4)   Data Item is the smallest unit of named data

5)   Many to Many relationships can be expressed directly in the hierarchical model

6)   ER to Network mapping is easier than in HDM

2.4 WHY RDBMS?

The relational model frees the user from the details of storage structures and access methods. It is also conceptually simple and more importantly based on sound theoretical principles that provide formal tools to tackle problems arising in database design and maintenance.

In this model, the relation is the only construct required to represent the associations among the attributes of entity as well as relationships among different entities.

One of the main reasons for introducing this model was to increase the productivity of the application programmer by eliminating the need to change application program when a change is made to the database. Users need not know exact physical structures to use the database and are protected from any changes made to these structures. They are however still required to know how the data has been positioned into the various relations.

2.5 FEATURES OF RDBMS

Relational Database Characteristics

The relational model is an example of record-based model. Record based models are so named because the database is structured in fixed format records of several types. Each table contains records of a particular type. Each record type defines a fixed number of fields, or attributes. The columns of the table correspond to the attributes of the record types. The relational data model is the most widely used data model, and a vast majority of current database systems are based on the relational model.

The relational model was designed by the IBM research scientist and mathematician, Dr. E.F.Codd. Two of Dr.Codd's main focal points when designing the relational model were to further reduce data redundancy and to improve data integrity within database systems. The relational model originated from a paper authored by Dr.codd entitled "A Relational Model of Data for Large Shared Data Banks", written in 1970. This paper included the following concepts that apply to database management systems for relational databases.

Example of a relation

account-number	branch-name	balance
A-101	Downtown	500
A-102	Perryridge	400
A-201	Brighton	900
A-215	Mianus	700
A-217	Brighton	750
A-222	Redwood	700
A-305	Round Hill	350

2.6 ATTRIBUTES, TUPLES & TABLES, CODD'S RULES

The relation is the only data structure used in the relational data model to represent both entities and relationships between them. Rows of the relation are referred to as **tuples** of the relation and columns are its **attributes**. Each attribute of the column are drawn from the set of values known as **domain**. The domain of an attribute contains the set of values that the attribute may assume.

From the historical perspective, the relational data model is relatively new The first database systems were based on either network or hierarchical models. The relational data model has established itself as the primary data model for commercial data processing applications. Its success in this domain has led to its applications outside data processing in systems for computer aided design and other environments.

In practice we can distinguish between entities and the relationships that exist between them. In modeling we represent an entity set in the database by a set of its properties. However only those properties of the entity type of interest to the application are used in to the model. A data model allows the capturing of these properties using its data structures Furthermore, we may wish to retrieve or update the stored data and for this purpose a data model supports certain operations. The data may also need to conform certain consistency and integrity rules, as in the case of bank's rule that a customer's account balance remain non – negative(i.e.>=0). These constraints are specified as integrity rules.The relational data model, like all data models consist of 3 basic components.

1. A set of domains and a set of relation
2. Operation on relations
3. Integity rules.

Structure of relational databases

A relational database consist of a collection of tables, each of which is assigned a unique name .A row in a table represents a relationship among a set of values. Since a table is a collection of such relationships, there is a close correspondence between the concept of a table and mathematical concept of relation, from which relational data model takes its name. The basic building blocks of relational systems are attributes and the domains on which they are defined.Because tables are essentially relations we shall use the mathematical terms relation and tuple in place of terms table and row.

customer
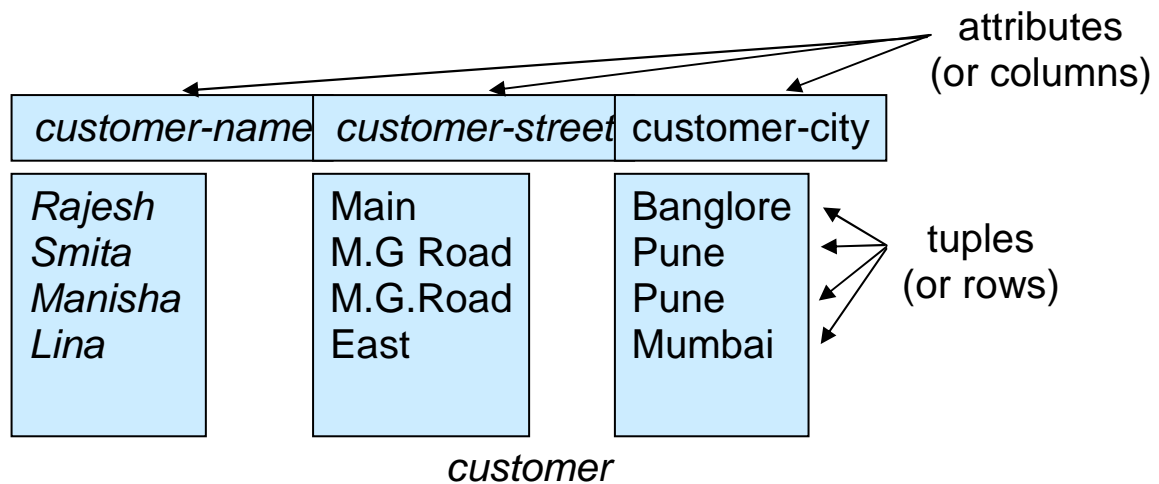
Relation Instance

Attributes and domains

An object or entity is characterized by its properties (or attributes). In conventional file system the term field refers to the smallest item of data with some practical meaning i.e. A field is used to capture some specific property of the object. In relational database systems, attributes corresponds to fields. For a given application an attribute may only be allowe3d to take a value from a set of permissible values. This set of allowable values for the attribute is the domain of the attribute. For the attribute branch name for example, the domain is the set of all branch names.

Domain

We define domain Di as a set of values of the same data type. The domain Di, a set having "homogeneous" members, is conceptually similar to the data type concept in programming languages. A domain like a data type may be unstructured (atomic) or structured.

Domain Di is said to be simple if all its elements are non decomposable (i.e. atomic).In typical DBMS systems atomic domains are general sets, such as the set of integers, real numbers, character strings and so on.

Atomic domains are sometimes referred to as application –Independent domains because these general sets are not dependent on a particular application.

We can also define application-dependent domains by specifying the values permitted in the particular database. Structured or composite domains can be specified as consisting of nonatomic values. For instance, the domain for the attribute address which specifies street number, street name, city, state, zip or postal code is considered as composite domain.

Attributes are defined on some underlying domain. That is they can assume values from the set of values in the domain. Attributes defined on the same domain are comparable as these attributes draw their values from the same set .It is meaningless to compare attributes defined on different domains. It has become traditional to denote attributes by uppercase letters from the beginning of the alphabet. Thus A, B, C with or without subscripts denote attributes.

In applications however attributes are given meaningful names. Set of attributes are denoted by uppercase letters from the end of the alphabets such as x, y, z.

Some example of domains

1) USA phone numbers
    The set of 10 digit phone numbers valid in United States.
2) Local phone numbers
    The set of 8 digit phone numbers valid within a particular area code.
3) Social security number
    The set of valid 9 digit social security number
4) Names
    The set of names of persons

Tuples

An entity type having n attributes can be represented by an ordered set of these attributes called an n-tuple.A tuple is comparable to a record in a conventional file systems and is used for handling relationship between entities.Tuples are generally denoted by lowercase letters r,s,t…of the alphabet. An n-tuple t can be specified as t = $(a_i, .a_n)$

We may write t [1] to denote the value of tuple t on the first attribute, t [2] to denote second attribute and so on.

Database scheme - Is logical design of database

Database Instance – Is the data in the database at a given instance in time.

The concept of a **relation scheme** corresponds to the programming language notion of type definition. A variable of a given type has a particular value at a given instance in time. Thus a variable in programming language corresponds to the concept of an **instance of a relation.**

It is convenient to give name to a relation, just as we give names to type definitions in programming language. We adopt the convention of using lowercase names for relations and names beginning with an uppercase letters for relation schemes. In following eg.use Deposit-Scheme to denote the relation scheme for relation deposit.

Deposit-scheme = (branch_name, account_number, customer_name, balance)

We denote the fact that deposit is a relation on the scheme Deposit by Deposit (Deposit-Scheme)

In general, relation scheme is a list of attributes and their corresponding domains. Thus relational data model is based on collection of tables. The user of the database system may query these tables, inert new tuples, delete tuples and update (modify) tuples.

There are several languages for expressing these operations. The tuple relational calculus and the domain relational calculus are non-procedural languages that represent the basic power required in a relational query language.

The relational algebra is a procedural language that is equivalent to both forms of relational calculus when they are restricted to safe expressions.

The relational algebra and the relational calculi are formal languages that are inappropriate for casual users of database system.

Databases may be modified by insertion, deletion, updation of tuples.

Different users of a shared database may benefit from individual views of the database.

CODD'S   RULES

Rule 0

For any system to be called a RDBMS, it must be able to manage databases entirely through its relational capabilities.

Rule 1: The information rule

All information in a RDBMS is represented explicitly (at the logical level) in exactly one way, by values in table

According to this rule, anything that does not exist in a table in the database does not exist at all. This means that all information including table names, view names, column names, and column data types should be available in some table within the database. Tables that hold such information constitute the data dictionary. Many products claiming to be relational fail on this count as they implement their data dictionary in structures which are not accessible like other relational tables.

Rule 2: The guaranteed access rule

Each and every datum (atomic values) is logically accessible through a combination of table name, column name and primary key value.

In any relational system, it should be possible to retrieve any item of data by supplying a table name, a column name and primary key (which identifies the row within the table) of the table. This also means that given a table name, column name and the values for the primary key, you will necessarily find one and only one value (or null) To achieve this, every table must have one and exactly one primary key (containing one or more columns) and no two rows should have the same values in all the columns of the primary key, i.e. there should be no duplicate rows. Most RDBMSs do not make the definition of the primary key mandatory, and are deficient to that extent.

Rule 3: Systematic treatment of null values

Inapplicable or missing information can be represented through null values. This rule specifies that RDBMS should support representation of null in place of column values to represent 'unknown values' and/or 'inapplicable values'. This support should be automatic and should not require any action on the part of the user. The support for 'null' is expected in all data types. Viz.character, integer, float etc. Note that in an integer column, null is different from the value zero.

RDBMS should support different kinds of 'null' to distinguish between the different kinds of 'null' to distinguish between the following values.

- Not applicable
- Unknown
- Does not exists
- Undefined
- Not valid
- Not supplied and so on

Rule 4: Active online catalog based on the relational model

The description of the database is held in the same way as ordinary data that is in tables and columns and is accessible to authorized users.

This rule states that table, view and authorization access definitions should be held in exactly one manner. i.e. tables and views. These tables should be accessible like other tables (i.e. through SQL statements)

Rule 5: Comprehensive data sublanguage rule

There must be at least one language which is comprehensive in supporting data definition, view definition, data manipulation, integrity constraints, and authorization and transaction control.

This means that there should be at least one language with a well defined syntax which can be used to manage the database completely. SQL is one such language which attempts to satisfy this rule. The SQL standard however does not cover all the requirements of such a language.

Rule 6: The view updating rule

All views that are theoretically updateable are updateable by the system. i.e.

Views are used to restrict access to data and for convenience. Data is stored only once (in base tables) but may be 'viewed' through many views. The SQL update command, may be used to update a view instead of a base table if it meets certain conditions. If it meets these conditions, it is called an 'updateable view', if not it is a 'non-updateable view'

To be updateable, a view must be
1)   Derived from a single table.
2)   Not have a GROUP BY clause or DISTINCT clause
3)   Not have an aggregate function
4)   No field of the view should use arithmetic expressions or constants

It is sufficient to say here that complex views can be built and it may be impossible to determine whether a view is theoretically updateable or not.

Rule 7: High level insert, update and delete

The capability of handling a base or derived table as a single operand applies not only to the retrieval of data but also to the insertion, updation and deletion of data. This means that all SELECT, UPDATE, DELETE, INSERT or their equivalents must be available and operate on set of rows in any relation.

Rule 8: physical data independence

Application programs and terminal activity remain logically unimpaired whenever any changes are made in the storage representation and access method.

E.g. Creation and dropping of an index to a table should not make any difference to the users and application programs accessing this table (though the time for access may vary). This rule implies that the behaviour (excluding response time) of the application programs and user terminal activity should be predictable and based on the logical definition of the database and this behaviour should remain unchanged irrespective of the changes in the physical definition as long as the logical definition of the database is not changed.

Changes to the physical definition may be required to improve performance, to assimilate the growth in size of the database, new application requirements etc.

Rule 9: Logical data independence

Application programs and terminal activities remain logically unimpaired when information persevering changes of any kind that theoretically permit unimpairement are made to the base table.

This logical data independence specifies that application programs and terminal activity should be independent of the logical structure and hence changes in the logical structure should not impair the application programs and terminal activities. E.g. it should be possible to split table TAB into tables TAB1 and TAB2 and define a view TAB (which combines TAB1 and TAB2) without impairing any application program or terminal activity.

Rule 10: Integrity independence

This specifies that RDBMS should support all integrity constraints such as

- Entity Integrity
- Referential Integrity

Rule 11: Distribution independence

The system must have a data sub-language which can support distributed databases without impairing application programs or terminal activities.

A complete distributed database support would mean that an arbitrary collection of relations, databases, running on a mix of machines and operating systems and connected by a variety of networks can function as if they were available on a single database and the application programs and user terminal activities are oblivious to this fact.

It should also be possible to move from a non-distributed to a distributed database implementation (and vice versa) without having to modify any of the existing programs or asking the users to change the method of terminal activity.

Rule 12: The nonsubversion rule

If the system has a low level (record at a time) language, this language cannot be used to bypass the integrity rules and constraints expressed in the higher level relational language.

Many products which merely build a relational front – end to their non-relational DBMS stand exposed on this count. In general, if all the database access is through SQL (or any other equivalent language), then this rule is satisfied, however many 'born again' DBMS support a row-at-a–time access mechanism below the relational integrity constraints thereby subverting the system in a variety of ways.

Relational Database Objects

Various types of objects can be found in a relational database. Some of the most common objects found in a relational database include

- **Table –** A table is the primary object used to store data in a relational database. When data is queried and accessed for modification, it is usually found in a table. A table is defined by columns. One occurrence of all columns in a table is called a row of data.

- **View -** A view is a virtual table, in that it looks like and acts like a table. A view is defined based on the structure and data of a table. A view can be queried and sometimes updated.

- **Constraint -** A constraint is an object used to place rules on data. Constraints are used to control the allowed data in a column. Constraints are created at the column level and also used to enforce referential integrity (parent and child table relationships)

- **Index -** An index is an object that is used to speed the process of data retrieval on a table. For example, an index might be created on a customer's name if users tend to search for customers by name. The customer names would be stored alphabetically in the index. The rows in the index would point to the corresponding rows in the table, much like an index in a book points to a particular page.

- **Trigger -** A trigger is a stored unit of programming code in the database that is fired based on an event that occurs in the database. When a trigger is fired, data might be modified based on other data that is accessed or modified. Triggers are useful for maintaining redundant data.

- **Procedure -** A procedure is a program that is stored in the database. A procedure is executed at the database level. Procedures are typically used to manage data and for batch processing.

The first four objects deal with the definition of the database, whereas the last two objects deal with methods for accessing database objects. Objects in a relational database provide users with a logical representation of data, such that the physical location of the data is immaterial to the user.

2.7 SUMMARY

Data are facts from which a conclusion can be drawn; for this reason, humans record data. Data is required in the operation of any organization, and the same or similar data may be required in various facets of its functioning.

A database system is an integrated collection of related files along with the details about their definition, interpretation, manipulation and maintenance. It is an attempt to satisfy the data needs of the various applications in an organization without unnecessary duplication. The DBMS not only makes the integrated collection of reliable and accurate data available to multiple applications and users, but also exerts centralized control, prevents fraud lent or unauthorized users from accessing the data and ensures privacy.

A DBMS is a complex software system consisting of a number of components. It provides the user with a data definition language and a data manipulation language. The user defines the external and conceptual views by using the DDL and manipulates the data contained in the database by using the DML.

The data manager is the component of the DBMS that provides an interface between the user (via the query processor or the compiled application program) and the file system. It is also responsible for controlling the simultaneous use of the database and maintaining its integrity and security. Responsibility for recovery of the database after any failure lies with the data manager.

Database users can be categorized into several classes, and each class of users usually uses a different type of interface to the database

DBMS Components

A database system has several subsystems

-       The storage manager subsystem provides the interface between the low level data stored in the database and the application programs and queries submitted to the system.
-       The query processor subsystem compiles and executes DDL and DML statements
-       Transaction management ensures that the database remains in a consistent (correct) state despite system failures .The transaction manager ensures that concurrent transaction executions proceeds without conflicting.
-       Database applications are typically broken up into a front-end part that runs at client machines and a part that runs at the back end. In two tier architectures, the front end directly communicates with a database running at the back end. In three tier architectures, the back end part is itself broken up into an application server and a database server.

DBMS Types

A number of data representation models have been developed over the years. As in the case of programming languages, one concludes that there is no one "best"

---

choice for all applications. These models differ in their method of representing the associations between entities and attributes. Traditional data models are hierarchical, network or relational data models. The hierarchical model evolved from the file based system; the network model is a superset of the hierarchical model. The relational data model is based on the mathematical relational concept. The data model concept evolved at about the same time as the relational data model.

RDBMS

The relational data model is the most widely developed model for storing data in databases. It is based on a collection of tables .The user of the database system may query these tables, insert new tuples, delete tuples and update (modify) tuples. There are several languages for expressing these operations.

In this chapter we studied the relational data model, consisting of the relational data structure. This model borrows heavily from set theory and is based on sound fundamental principles. Relational operations are applied to relations, and the result is relation.

Conceptually, a relation can be represented as a table, each column of the table represents an attribute of the relation and each row represents a tuple of the relation Mathematically a relation is a correspondence between a number of sets and is a subset of the Cartesian product of these sets. The sets are domains of the attributes of the relation. Duplicate tuples are not permitted in a relation. Each tuple can be identified uniquely using a subset of the attributes of the relation. Such a minimum subset is called a key (primary) of the relation. The unique identification property of the key is used to capture relationship between entities. Such a relationship is represented by a relation that contains a key for each entity involved in the relationship.

2.8 CHECK YOUR PROGRESS - ANSWERS

2.1
A)
1) Inherent
2) s/w system
3) Database Management System
4) Multi user system
5) Accurate
6) Logical data independence
7) Centralization
8) Physical level
9) Data definition language
10) Naïve users

B)
1) False
2) True
3) True
4) False
5) True

2.2
A)
1) 1 billion, 1 million
2) Query processor
3) Metadata
4) Storage manager
5) Indices
6) DDL interpreter
7) Query Evaluation Engine
8) Authorization and Integrity Manager

B)
1) False
2) True
3) True
4) False

2.3
A)
1) Parent child relationship
2) One
3) M: N
4) Virtual record
5) DBTG
6) Set
7) Owner, member
8) Data Item
9) Vector
10) A repeating group
11) E.F.Codd

B)
1) True
2) False
3) True
4) True
5) False
6) True

2.4, 2.5 2.6
A)
1) Tables
2) Tuples
3) Domain

B)
1) True
2) False

2.9 QUESTIONS FOR SELF –STUDY

1. Explain the terms Data, Database, DBMS.

2. What is DBMS, explain its characteristics?

3. Write a note on advantages and Limitations of DBMS.

4. State and explain various users of DBMS .

5. What do you mean by persistent data? Give any 4 examples.

6. Explain Physical and Logical Data Independence.

7. Explain 3-tier architecture of DBMS.

8. Explain software components of DBMS and write their functions.

9. What is data model? Describe the type and significance with proper example.

10. State and explain the responsibilities of DBA

11. Describe following terms -

    - Attribute   - Domain   - Tuple

    - Relation   - Schema   - Instance

12. Explain codd's rules.

13. State difference between single and multi user system.

14. State difference between HDM, NDM, and RDM.

15. Explain tree structure diagrams and data structure diagrams.

❖ ❖ ❖

NOTES

Chapter 3
Entity Relationship Model

3.0 OBJECTIVES

In this chapter the entity–relationship model is introduced.

After studying this chapter you will be able to –

- Discuss entity relationship model.
- Explain the distinction among the Keys in DBMS.
- Explain different convention conventions for Drawing ERD.
- Discuss what is abstraction.

3.1 INTRODUCTION

This chapter gives you clear idea about what is entity, entity sets, relationship sets etc. It also teaches you different types of entities, different keys available in DBMS and also how to draw an ERD (Entity Relationship diagram)

3.2 ENTITY-RELATIONSHIP MODEL

ER model is a popular high level conceptual data model. Earlier commercial systems were based on the hierarchical and network approach. The ER model is a generalization of these models. It allows the representation of explicit constraints as well as relationships.

Even though the ER model has some means of describing the physical database model, it is basically useful in the design and communication of the logical database model.

In this model, objects of similar structure are collected into entity set. The relationship between entity set is represented by a named ER relationship and is 1:1, 1:M or M:N., mapping from one entity set to another. The database structure, employing the ER model is usually shown pictorially using ERDs.

ER model consist of basic objects, called entities and relationship among these objects. It represents overall logical structure of a database. The database structure, employing the ER model is usually shown pictorially using ERDs. Semantic modeling is known by many names including data modeling, Entity relationship modeling, entity modeling and object modeling. ER model was introduced by Chen in 1976 and refined in various ways by Chen and numerous others since that time. It is one of the several semantic data models ER model is based on 3 basic concepts.

1) **Entity sets**
2) **Relationship sets**
3) **Attributes**

3.2.1 Entity set

An Entity is a thing or object in the real world that is distinguishable from all other objects. It is an object of interest to an organization. E.g. Each person in an enterprise

Each student in the institute
Loan in bank etc.
For instance a student has student-id which uniquely identifies one particular student in the institute. Similarly, loans can be thought of as entities and loan number at particular branch uniquely identifies a loan entity.

An entity may be concrete, such as a person or a book, or it may be abstract such as loan, holiday

Objects of similar types are characterized by the same set of attributes or properties. Such similar objects form an entity set or entity type. Two objects are mutually distinguishable and this fact is represented in the entity set by giving them unique identifiers. Objects are represented by their attributes and as objects are interdistinguishable, a subset of these attributes forms a primary key or a key for uniquely identifying an instance of an entity.

An **Entity Set** is a set of entities of the same type that share the same properties or attributes. The set of all persons who are students at a given institute can be defined as an entity set student. Similarly entity set loan might represent set of all loans awarded by a particular bank. The individual entities that constitute a set are said to be the extension of the entity set. Eg. Individual bank customers are the extension of the entity set customer.

3.2.2 Relationship sets

A relationship is an association among several entities .e.g. Relationship between loan and customer entities. A relationship set is a set of relationships of the same type. We define the relationship set borrower to denote the association between customers and bank loans that the customers have. When 2 entities are involved in relationship it is called binary relationship. When 3 entities are involved in it it is ternary relationship. When N entities are involved then it is N-ary relationship.

The association between entity sets is referred to as participation. The entities involved in a given relationship are said to be participants in that relationship. The function that an entity plays in a relationship is called that entities role. The relationship may also have attributes called **descriptive attributes.**

A binary relationship can be one-to-one, one-to-many, many-to-many.
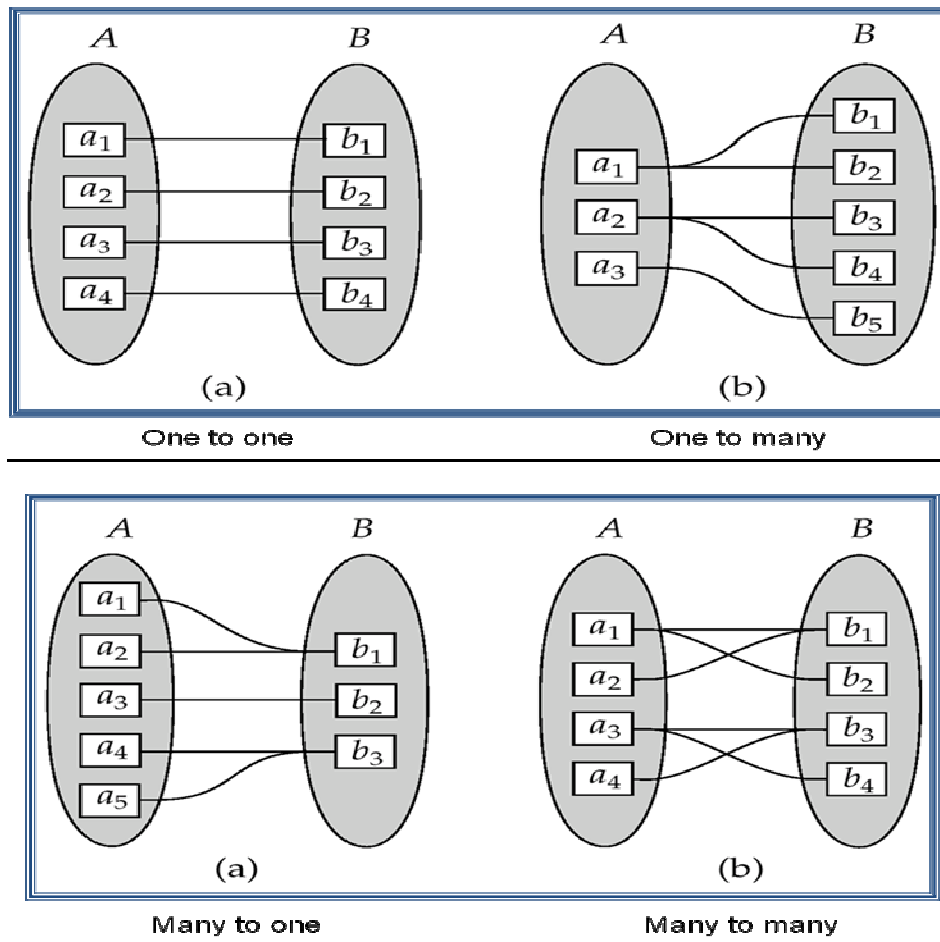E.g. of one-to-one Relationship
1) person-spouse
2) person-driving license

E.g. of Many-to-one Relationship
1) Project-Manager
2) Employee-Department
3) Dependent-Employee

E.g. of Many-to-Many Relationships
Employee - Project
Supplier – Parts

Mapping Cardinalities

(a) One to one (b) One to many

(a) Many to one (b) Many to many

3.2.3 Attributes and Values

Entities are distinguishable objects of concern and are modeled using their characteristics and attributes.

Formally, an attribute of an entity set is a function that maps from the entity set into a domain. Associations exist between different attributes of an entity. An association between 2 attributes indicates that the values of the associated attributes are interdependent. Association that exists among the attributes of an entity is called attribute association. And that exist between entities is called a relationship. Formally, the association or interdependence between attributes is called functional dependency.

An entity is represented by a set of attributes. Attributes are properties of each member of entity set. E.g. possible attributes for customer entity set are customer-id, customer-name, customer-street and customer-city. Each attribute has a value for each of its attributes.

For each attribute, there is a set of permitted values, called the **domain**, or **value set** of that attribute. Eg.Domain of attribute loan -number might be the set of all strings of the form "L-n" where n is a positive integer.

Since an entity set may have several attributes each entity can be described by a set of (attribute, data value) pairs.

An attribute values describing an entity will constitute a significant portion of the data stored in the database. An attribute as used in ER model can be characterized by the following attribute types.

1) **Simple and composite attributes**
2) **Single and multivalued attributes**
3) **Derived Attributes**

Simple and composite attributes:

Simple attributes are attributes which are not further divided into subparts. On other hand composite attributes can be divided into subparts. (I.e. other attributes).

E.g. Attribute student-name could be structured as composite attribute consisting of first-name, middle-initial and last name.

Attribute address may consist of attributes street (street number, street name, apartment number), city, state and zip code.

Thus composite attributes help us to group together related attributes.

Single valued and multivalued attributes

Attributes having single value for a particular entity are single valued. E.g. Loan-number.

Attributes that have a set of values for a specific entity are multivalued. E.g.
1) 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.
2) An attribute dependent-name of the employee set would be multivalued, since any particular employee may have zero, one or more dependents.

Derived attributes

The value for this type of attribute can be derived from the values of other related attributes or entities. E.g.
1) Customer entity has an attribute age. If the customer entity also has an attribute date-of –birth, we can calculate age from the date-of-birth and the current date. Thus age is a derived attribute. (Here date-of-birth may be referred to as a *base attribute* or a *stored* attribute). The value of a derived attribute is not stored but is computed when required.
2) If 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.

An attribute takes null value when an entity does not have a value for it. E.g. Middle name Null may indicate not applicable –that is, that the value does not exist for the entity, or the attribute value is unknown (missing or not known).
E.g.
1) A null value for apartment number attribute could mean that the address does not include an apartment number (not applicable), that an apartment number exists but we do not know what it is (missing) or it is not part of customer address (unknown)

3.2 Check your progress

A) Fill in the blanks

1) Attributes having single value for a particular entity are …………………………….

2) The database structure, employing the ER model is usually shown pictorially using ………………………….

3) An …………………………is a thing or object in the real world that is distinguishable from all other objects

4) For each attribute, there is a set of permitted values, called the ------------.

B) State true or false

1) Null value means value 0

2) ER model was introduced by codd in 1976

3) Domain is also called as a value set

4) Employee-Project is an example of Many-to-Many relationship

3.3 STRONG AND WEAK ENTITIES

An entity set that has a primary key is termed as strong entity set. E.g. Employee

An entity that does not have sufficient attributes to form a primary key is termed as a weak entity set. E.g. Dependent. Thus weak entity is an entity that is existence dependent on some other entity, in the sence that it can not exist if that other entity does not also exist. So here if a given employee is deleted, all dependents of that employee must be deleted too. Entities belonging to a weak entity type are identified by being related to specific entities from another entity type in combination with some of their attribute values. We call this other entity type the **identifying owner** (strong entity) and we call the relationship type that relates a weak entity type to its owner the **identifying relationship** of the weak entity type. A weak entity cannot be identified without an owner entity.
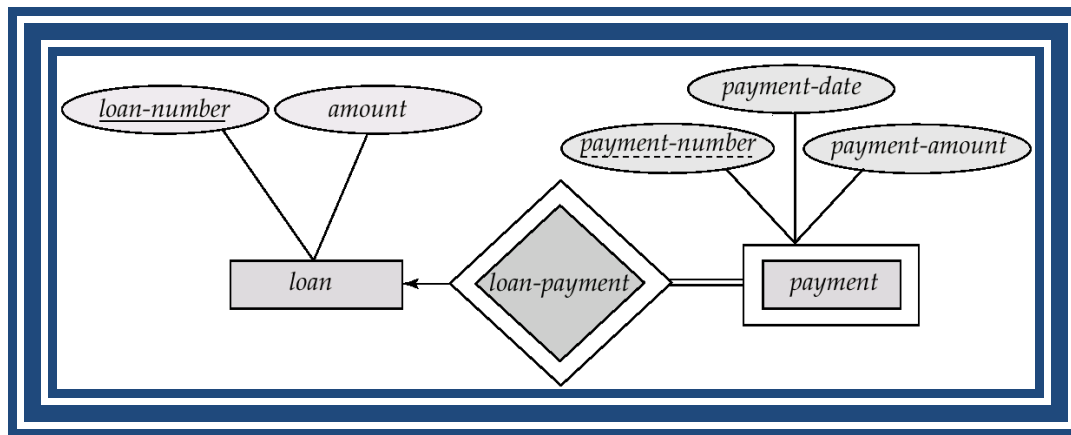
3.4 KEYS IN DBMS

A key is a single attribute or combination of two or more attributes of an entity set that is used to identify one or more instances of the set. The attribute of entity set which identifies and distinguishes instances of entity set is called primary key. If we add additional attributes to a primary key, the resulting combination would still uniquely identify an instance of the entity set. Such keys are called super keys. A primary key is therefore a minimal super key

There may be two or more attributes or combinations of attributes that uniquely identify an instance of an entity set. These attributes or combination of attributes that uniquely identify an instance of an entity set. These attributes or combinations of attributes are called **candidate keys.** In such a case we must decide which of the candidate keys will be used as the primary key. The remaining candidate keys would be considered **alternate keys.**

A **secondary key** is an attribute or combination of attributes that may not be a candidate key but that classifies the entity set on a particular characteristic.

3.3 3.4 Check your Progress

Fill in the blanks

1) An entity set that has a primary key is termed as ……………………… entity set

2) An identifying owner is nothing but a ……………………… entity

State true or false

1) An entity that does not have sufficient attributes to form a primary key is termed as a Strong entity set
2) Weak entity is an entity that is existence dependent on some other entity
3) Relationship that exists among owner to weak entity set is called as an identifying relationship

---

1) An entity is shown as a rectangle

2) A diamond represents the relationship among a number of entities, which are connected to the diamond by lines.

3) The attributes, shown as ovals, are connected to the entities or relationship by lines.

4) Diamonds, ovals and rectangles are labeled. The type of relationship existing between the entities is represented by giving the cardinality of the relationship on the line joining
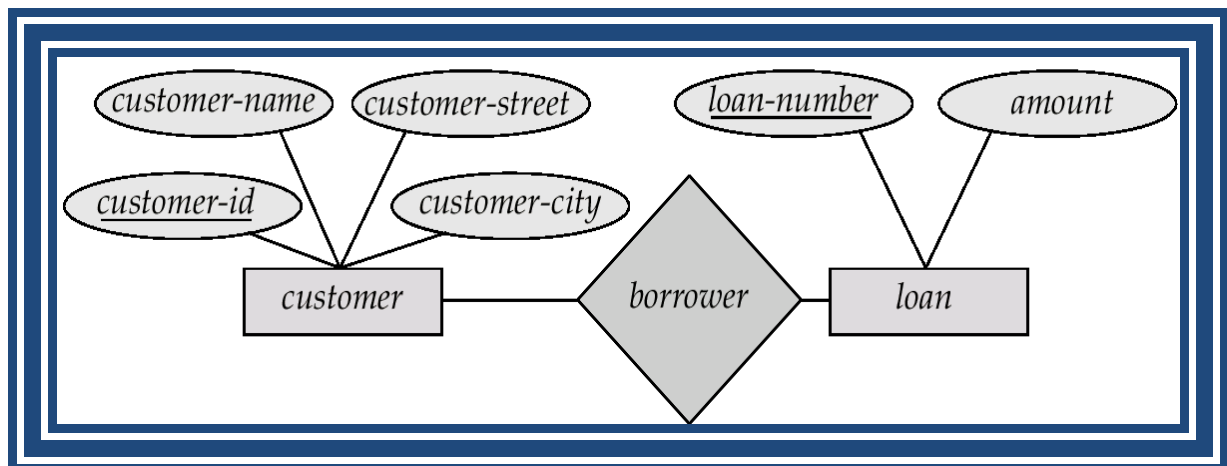
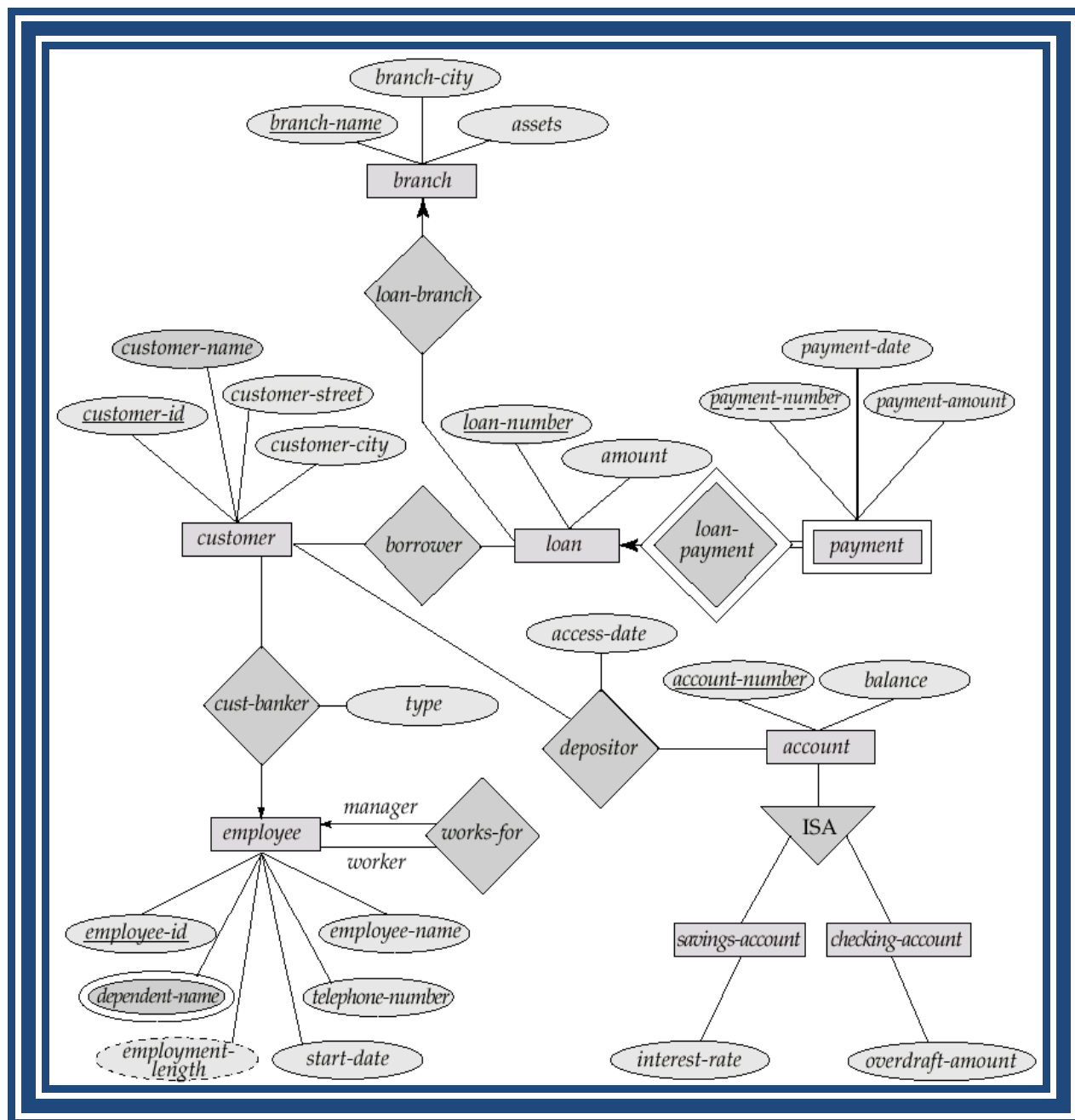The relationship to the entity.

Symbols used while drawing ERD
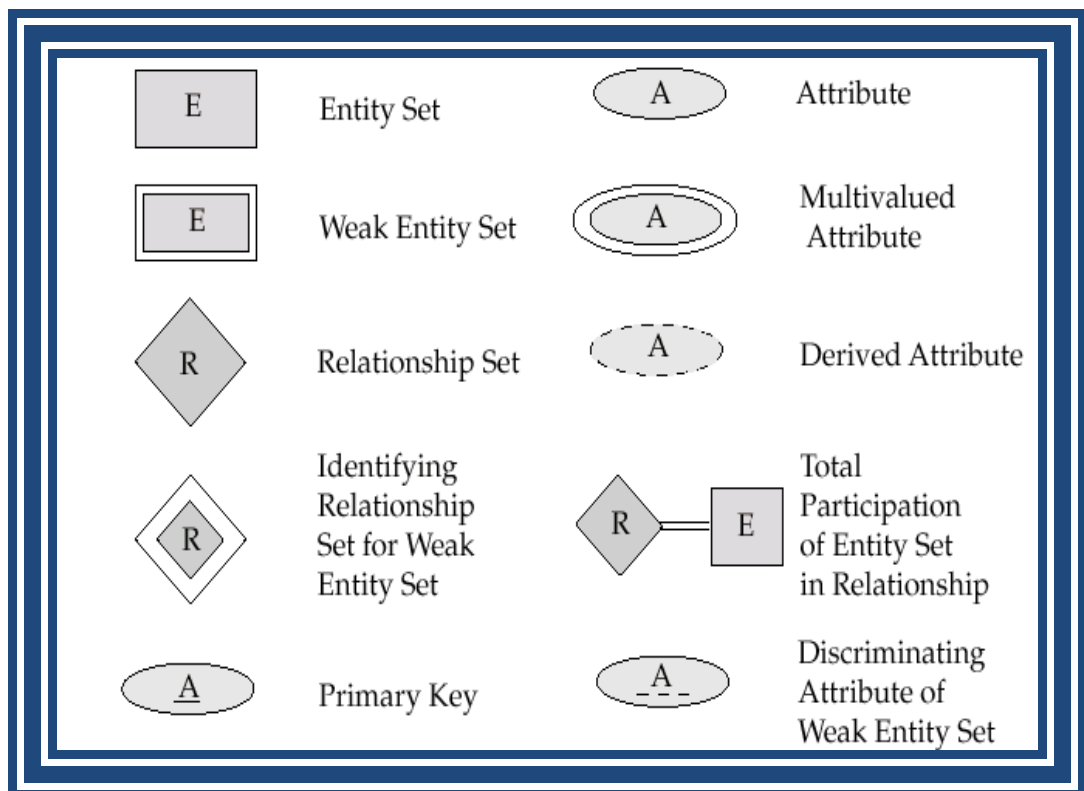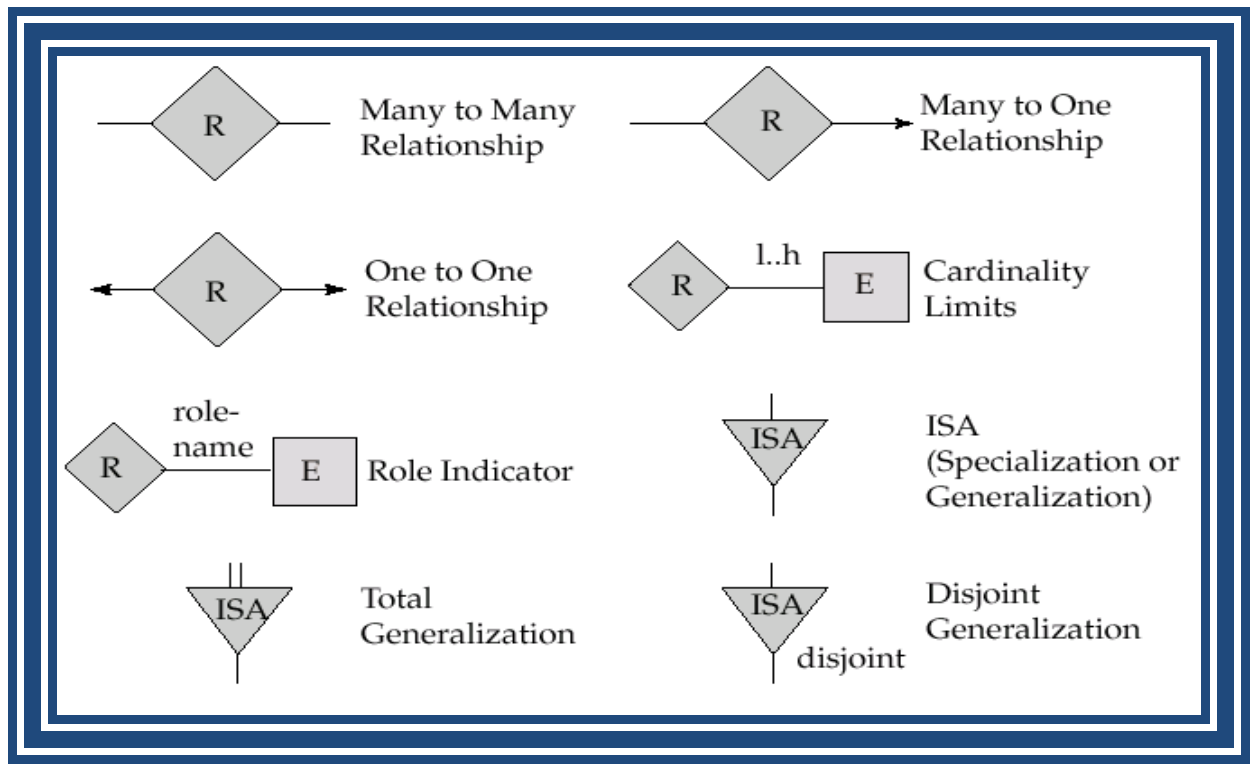
Example of Entity Relationship diagram (ERD)

Example of Entity Relationship diagram (ERD)

3.6 ABSTRACTION

Although the basic ER model can express most database features, some aspects of a database may be more cleanly expressed by certain expressions to the basic ER model. This is called enhanced or extended ER model.

Abstraction is the simplification mechanism used to hide superfluous details of a set of objects; it allows one to concentrate on the properties that are of interest to the application. Vehicle itself is an abstraction that includes the types car, truck and bus. There are 2 main abstraction mechanisms used to model information. These are also called features of extended ER model.

   1)   **Generalization**
   2)   **Specialization**
   3)   **Aggregation**

3.7 GENERALIZATION

Generalization, Specialization and Aggregation
   Generalization is the abstracting process of viewing set of objects as a single general class by concentrating on general characteristics of the constituent sets while

suppressing or ignoring their differences. It is the union of a number of lower level entity types for the purpose of producing a higher level entity types. Generalization is a containment relationship that exists between a higher level entity set and one or more low-level entity sets.

E.g. Student is a generalization of graduate or undergraduate, full-time or part-time students Employee is a generalization of the classes of objects cook, waiter, cashier etc. Generalization is an Is-A relationship, therefore manager Is-An employee, cook Is-An employee and so forth.

Specialization

The process of designating sub groupings within an entity set is specialization.

It is the abstracting process of introducing new characteristics to an existing class of lower level entities also inherits the characteristics of higher level entity. Specialization may be seen as the reverse process of generalization .Additional specific properties are introduced at a lower level in a hierarchy of objects.

Aggregation

One limitation of the ER model is that it is not possible to express relationships among relationships. Aggregation is an abstraction through which relationships are treated as higher level entities. It helps us to express relationships among relationships.

It is the process of compiling information on an object, thereby abstracting a higher-level object. In this manner the entity person is derived by aggregating the characteristics name, address and social security number. Another form of aggregation is abstracting a relationship between objects and viewing the relationship as an object .As such, the Enrollment relationship between entities student and course could be viewed as entity Registration.

<u>**Specialization Example**</u>
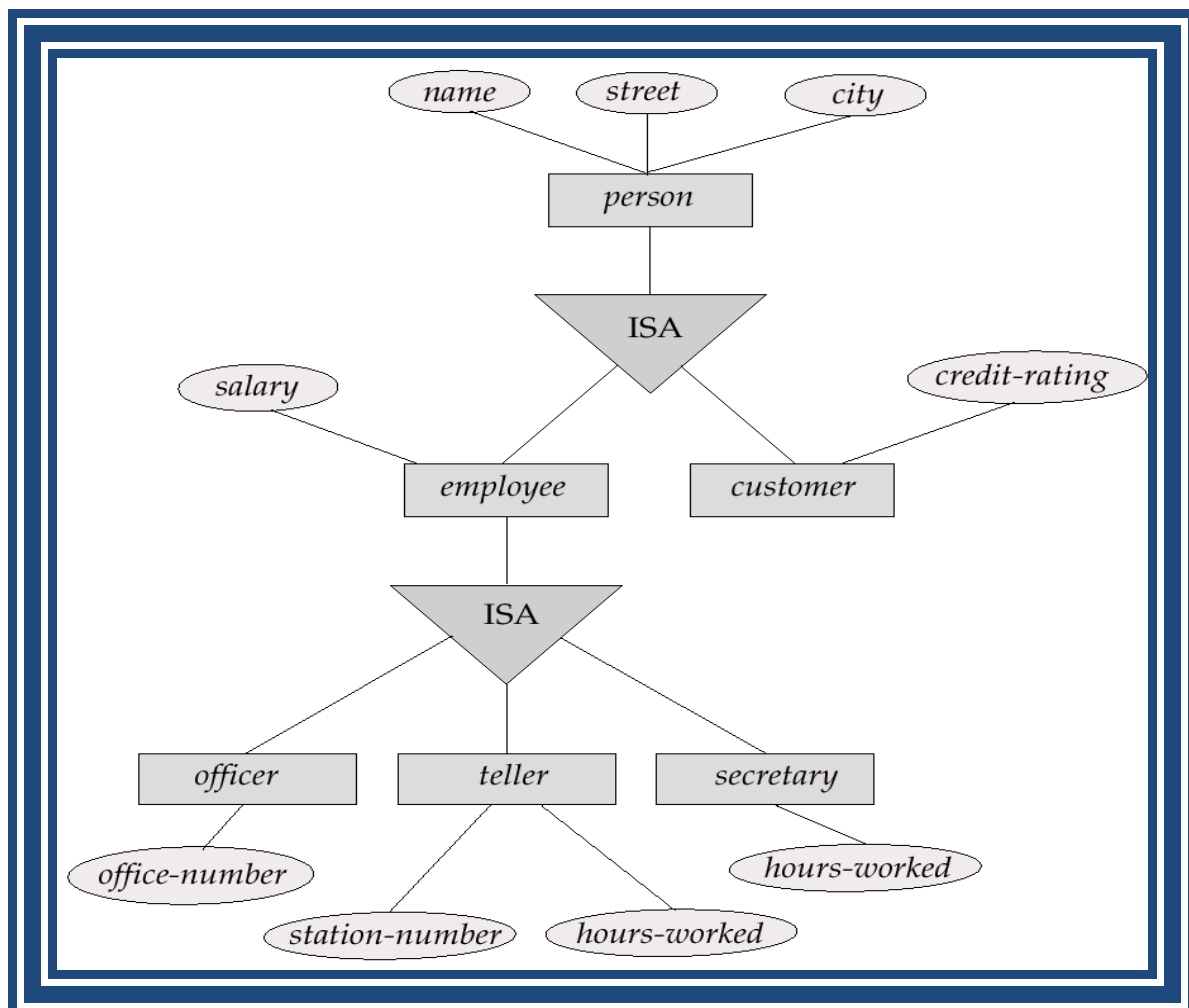
Aggregation Example
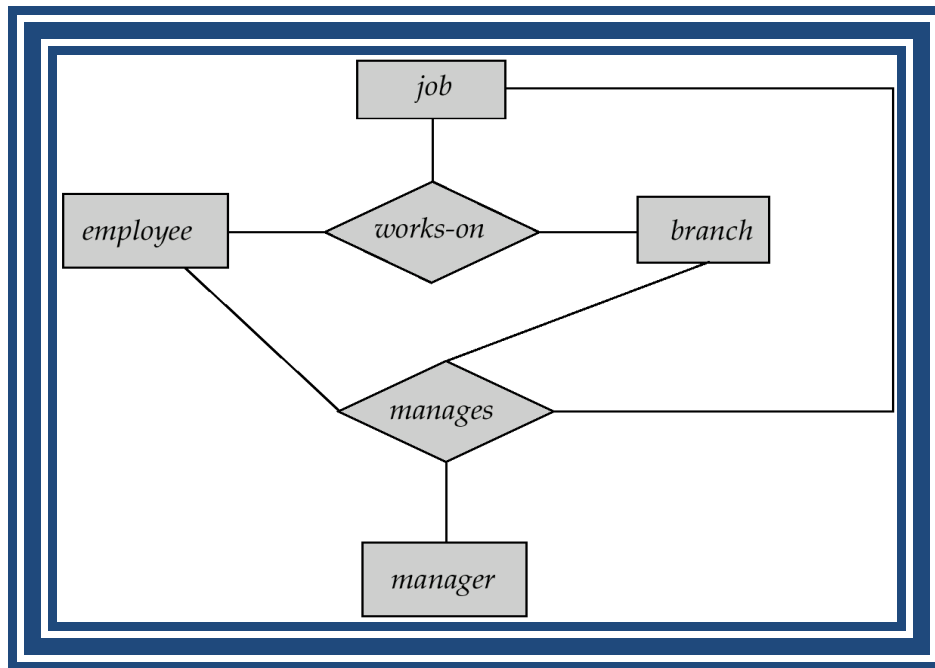
Entity Relationship Diagram with aggregation
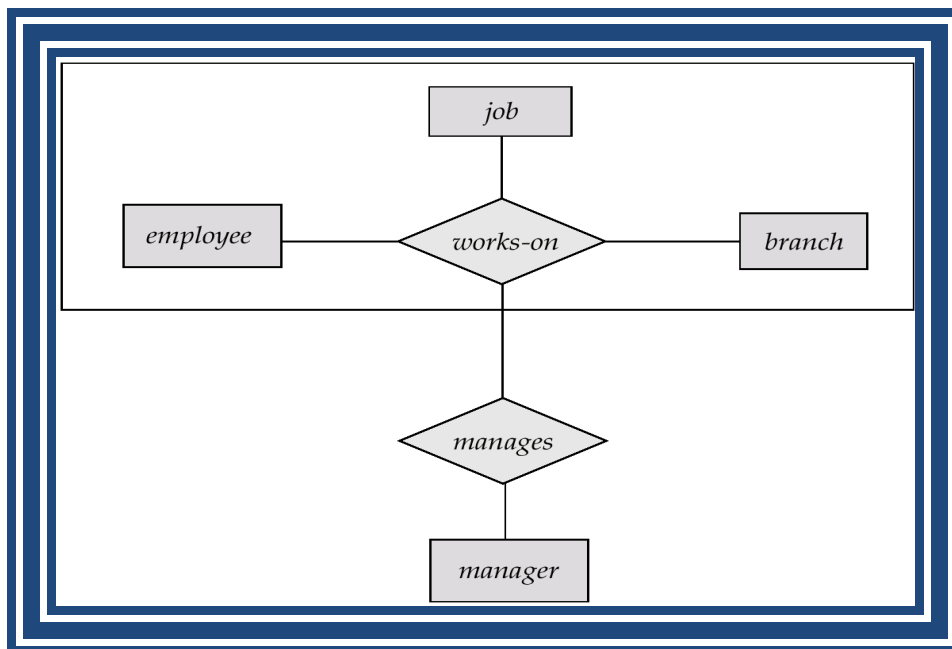
3.5, 3.6, 3.7 Check your progress

A) Fill in the blanks.

1)   -------------------------- the union of a number of lower level entity types for the purpose of producing a higher level entity types.

2)   One limitation of the ER model is that it is not possible to express relationships among -----------------------.

3)   ---------------------- is an abstraction through which relationships are treated as higher level entities.

B) State true or false.

1)   Aggregation is an abstraction through which relationships are treated as higher level entities

2)   Generalization is an Is-A relationship

3)   An entity is shown as a diamond

Example of ERD
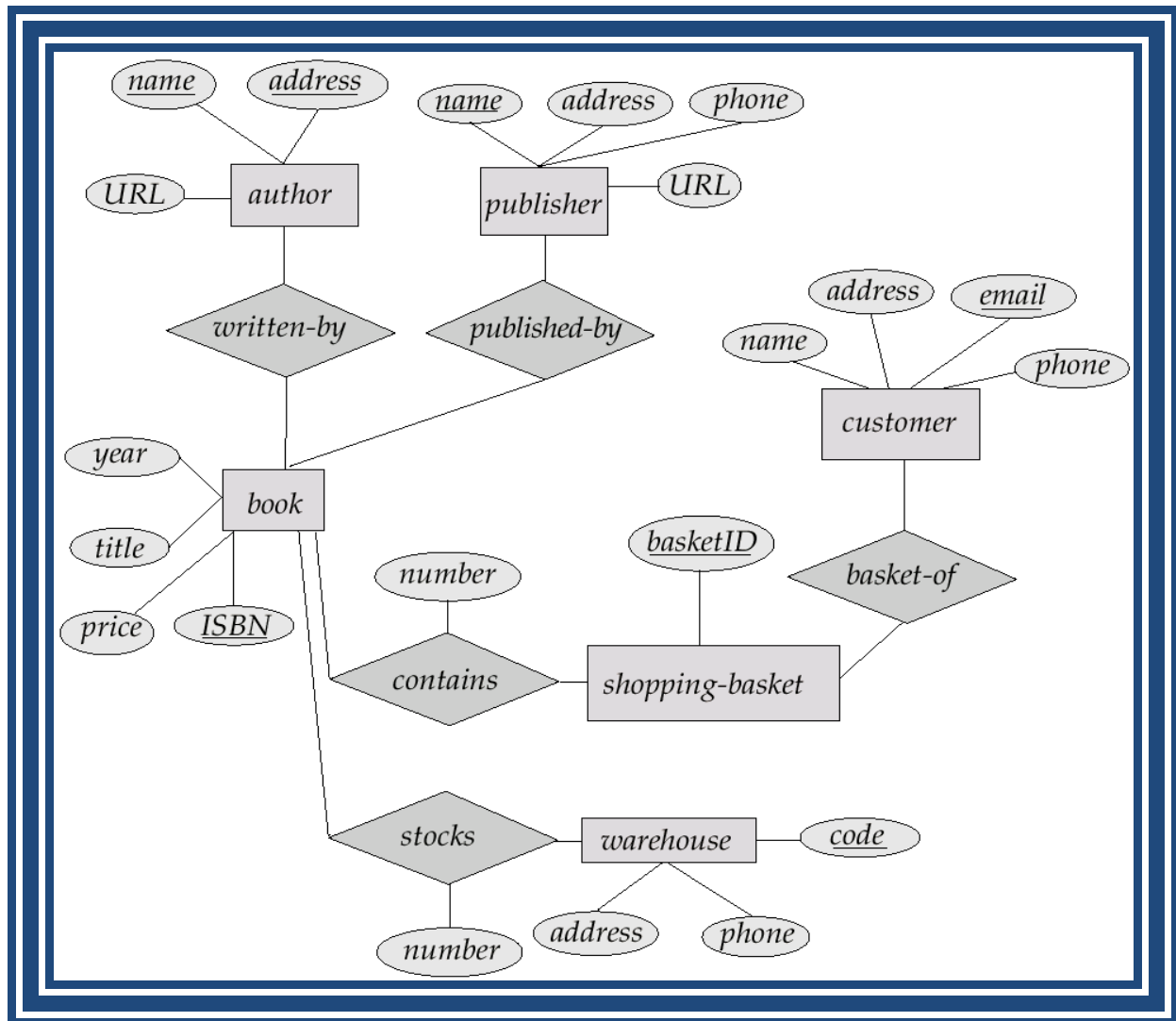
3.8 SUMMARY

Database design mainly involves the design of the database schema. The entity relationship (E-R) data model is a widely used data model for database design. It provides a convenient graphical representation to view data, relationships and constraints.

The model is intended primarily for the database design process. It was developed to facilitate database design by allowing the specifications of an enterprise schema. Such a schema represent the overall logical structure of the database. The overall structure can be expressed graphically by an E-R diagram.

An entity is an object that exists in the real world and is distinguishable from other objects. We express the distinction by associating with each entity a set of attributes that describes the object.

Mapping cardinalities express the number of entities to which another entity can be associated via a relationship set.

A relationship is an association among several entities. A relationship set is a collection of relationships of the same type, and an entity set is a collection of entities of the same type.

An entity set that does not have sufficient attributes to form a primary key is termed as a weak entity set. An entity set that has a primary key is termed a strong entity set.

3.2

A)
1) Single valued
2) ERDs
3) Entity
4) Domain

B)
1) True
2) False
3) True
4) True

3.3, 3.4

A)
1) Strong
2) Strong

B)
1) False
2) True
3) True

3.5, 3.6, 3.7

A)
1) Generalization
2) Relationships
3) Aggregation

B)
1) True
2) True
3) False

3.10 QUESTIONS FOR SELF - STUDY

1. Explain the distinctions among the terms primary key, candidate key and super key.

2. Explain the difference between a weak and strong entity set with example.

3. Define the concept of aggregation with example.

4. Design a generalization-specialization hierarchy for a motor vehicle sales company. The company sales motorcycles, passenger cars, vans and buses. Justify your placement of attributes at each level of the hierarchy Explain why they should not be placed at a higher or lower level.

5. Explain different conventions that you follow while drawing ERD

6. What do you mean by single valued and multivalued attributes Explain with example?

7. Explain following terms

   -Entity   -Entity set   -Relationship

   - Relationship set   -Attribute   -Domain

8. Identifying proper entities, d raw entity relationship diagram for hospital management system for OPD (out door patients) only.

❖ ❖ ❖

NOTES

NOTES

DBMS Concepts

4.0 OBJECTIVES

Friends,

After studying this chapter you will be able to –

- Describe various DBMS concepts.

- Explain different integrity constraints.

4.1 INTRODUCTION

This chapter introduces you with different DBMS concepts like concurrency, recovery, different integrity constraints etc.

4.2 ACID PROPERTIES:-TRANSACTION PROPERTIES

To ensure the integrity of data, database system maintains following properties of transaction. These are called ACID properties.

1) Atomicity
Atomicity property ensures that at the end of transaction, either no changes have occurred to the database or the database has been changed in a consistent manner. At the end of transaction, the updates made by the transaction will be accessible to other transactions and processes outside the transaction.

2) Consistency
Consistency property of transaction implies that if the database was in consistent state before the start of transaction, then on termination of transaction the database will also be in a consistent state

3) Isolation
This property indicates that action performed by a transaction will be hidden from outside the transaction until the transaction terminates. Thus each transaction is unaware of other transaction executing concurrently in the system.

4) Durability
Durability ensures that once a transaction completes, i.e. successfully commits, the changes it has made to the database persist, even if there are system failures

4.3 CONCURRENCY CONTROL

Concurrency occurs in a multi-user system where at a time more than one user tries to make transactions and it is called concurrency.

So when several transactions are executed concurrently, the corresponding schedule is called concurrent schedule .But due to concurrency following problems may occur.

1)   Lost Update Problem
2)   Inconsistent Read Problem
3)   The phantom phenomenon
4)   Semantics of concurrent transaction

Concurrency control techniques

1)   Time stamp ordering
2)   Locking Protocol
3)   Validation Techniques
4)   Multiversion Technique (MVT)

4.4 RECOVERY MECHANISMS

The types of failures that the computer system is likely to be subjected to include failures of components or subsystems, s/w failures, power cutages, accidents, unforeseen situations and the natural or manmade disasters. Database recovery techniques are the methods of making the database fault tolerant. The aim of the recovery scheme is to allow database operations to be resumed after a failure with minimum loss of information at an economically justifiable cost.

Recovery mechanisms in centralized DBMS

-Log Based Recovery
A log, which is usually written to the stable storage, contains the redundant data required to recover from volatile storage failures and also from errors discovered by the transaction or the database system.

-Checkpoints
In an on-line database system, for example in airline reservation system, there could be hundreds of transactions handled per minute. The log for this type of database contains a very large volume of information. A scheme called checkpoint is used to limit the volume of log information that has to be handled and processed in the event of a system failure involving loss of volatile information. The checkpoint scheme is an additional component of the logging scheme described above.

-Shadow page table
Shadow paging is an alternative to log-based recovery; this scheme is useful if transactions execute serially
Idea: maintain two page tables during the lifetime of a transaction – the current page table, and the shadow page table
Store the shadow page table in nonvolatile storage, such that state of the database prior to transaction execution may be recovered.
Shadow page table is never modified during execution

4.5 VIEWS AND SECURITY

It is not desirable for all users to see entire logical model. Security considerations may require that certain data to be hidden from users.

Any relation that is not part of the logical model, but is made visible to user as a virtual relation, is called a view. It is possible to support a large number of views on top of any given set of actual relations.

View Definition

We define a view in SQL by using the **create view** command. To define a view, we must give the view a name and must state the query that computes the view. The form of the **create view** command is **Create view v as <**query expression>

A view can hide data that a user does not need to see. The ability of views to hide data serves both to simplify usage of the system and to enhance security. Views simplify system usage because they restrict the user's attention to the data of interest. Although a user may be denied direct access to a relation, that user may be allowed to access part of that relation through a view. Thus a combination of relational-level security and view-level security limits a user's access to precisely the data that the user needs.

4.6 INTEGRITY CONSTRAINTS

Integrity constraints provides a means of ensuring that changes made to the database by authorized users do not result in a loss of data consistency. Thus integrity constraints guard against accidental damage to the database. Security constraints guard against accidental or malicious tampering with data whereas integrity constraints ensure that any properly authorized access, alteration, deletion or insertion of the data in the database does not change the consistency and validity of the data. This requires that there is a need for guarding against invalid database operations. An operation here is used to indicate any action performed on behalf of a user or application program that modifies the state of the database. Such operations are the result of actions such as update, insert or delete. In short invalid changes have to be guarded against by the integrity subsystem whereas illegal updates must be guarded against by the security subsystem. Database integrity involves the correctness of data; this correctness has to be preserved in the presence of concurrent operations, errors in the user operations and application programs and failures in H/W and S/W. The recovery system ensures that failures of various types which may cause the loss of some of the actions of one or more transactions will not cause the database to become inconsistent. The centralized integrity constraints can be maintained in a system catalog (data dictionary) and can be accessible to the database users via the query language. The relational data model includes integrity rules. These integrity rules implicitly or explicitly defines the set of consistent database states or changes of state or both.

Entity Integrity (Integrity Rule 1)

Integrity rule 1 is concerned with primary key values. Before we formally state the rule let us look at the effect of null values in prime attributes. A null value for an attribute is a value that is either not known at the time or does not apply to a given instance of the object. If any attribute of a primary key (prime attribute) were permitted to have null values, then because the attributes in key must not be redundant, the key cannot be used for unique identification of tuples. This contradicts the requirements for a primary key. Integrity rule 1 specifies that instances of the entities are distinguishable and thus no prime attribute (component of a primary key) value may be null. This rule is also referred to as the entity rule. We could state this rule formally as **If attribute A of relation R(R) is a prime attribute of R(R), then A cannot accept null values.**

Referential Integrity (Integrity rule 2)

It is concerned with foreign keys i.e. with attributes of a relation having domains that are those of the primary key of another relation. Relation (R) may contain reference to another relation (S). Relations R and S need not be distinct. Suppose the reference in R is via a set of attributes that forms a primary key of the relation S. This set of attributes in R is a foreign key. A valid relationship between a tuple R to one in S requires that the values of the attributes in the foreign key of R corresponds to the primary key of tuple in S. This ensures that the references from a tuple of the relation R is made unambiguously to an existing tuple in the S relation. The referencing attribute (s) in the R relation can have numm value(s) in this case; it is not referencing any tuple in the S relation. However if the value is not null, it must exist as the primary attribute of a tuple of the S relation. If the referencing attribute in R has a value that is nonexistent in S, R is attempting to refer a nonexistent tuple and hence nonexistent instance of the corresponding entity. This cannot be allowed. In the following example, employees has managers and as managers are also employees we may represent managers by their employee numbers, if the employee numbers, if the employee number is a key of the relation employee. Here the manager attribute represents the employee number of the manager. Manager is a foreign key, note that it is referring to the primary key of the same relation An employee can only have a manager who is also an employee. Some employees may also be temporarily without managers and this can be represented by the manager taking null values.

EMP No	Name	Manager
101	Jones	@
103	Smith	110
104	James	107
107	Evan	110
110	Drew	112
112	Smith	112

Given two relations R and S, suppose R refers to the relation S via a set of attributes that forms the primary key of S and this set of attributes forms a foreign key in R. Then the value of the foreign key in a tuple R must either be equal to the primary key of a tuple of S or be entirely null
Domain or Data Item value Integrity Rule

One aspect that has to be dealt with by the integrity subsystem is to ensure that only valid values can be assigned to each data item. This is referred to as Domain integrity. Domain integrity rules are simply the definition of the domains of the attributes or the value set for the data-items. The value that each attribute or data item can be assigned is expressed as being one or more of the following forms:

A data type. E.g. Alphanumeric string or numeric, a range of values or a value from a specified set. For instance, in the relation employee, the domain of the attribute salary may be given as being between $12000 and $300000.The final grade assigned to a student in a course can only be one of say A,B,C,D or F.

A domain can be composite, for instance the date attribute in the medical history is restricted to the form mm/dd/yyyy

Since online data entry is a common operation, validation of the entered values has to be performed by application programs. however this approach has drawbacks

1) It depends on the application programmer to include all validity checks
2) Each application program is duplicating some of these checks.
   Integrity mechanisms can only ensure that the data is in the specified domain. Some domain constraints could be conditional.
   Thus domain constraints specify the set of possible values that may be associated with an attribute. Such constraints may also prohibit the use of null values for particular attributes.

4.7 DATA SECURITY

Data Security is nothing but protection from malicious attempts to steal or modify data.

- **Database system level**
- Authentication and authorization mechanisms to allow specific
  Users access only to required data
- We concentrate on authorization in the rest of this chapter

- **Operating system level**
- Operating system super-users can do anything they want to the database! Good operating system level security is required.

- **Network level: must use encryption to prevent**
- Eavesdropping (unauthorized reading of messages)
- Masquerading (pretending to be an authorized user or sending messages supposedly from authorized users)

- **Physical level**
- Physical access to computers allows destruction of data by Intruders; traditional lock-and-key security is needed
- Computers must also be protected from floods, fire, etc.

- **Human level**
- Users must be screened to ensure that authorized users do not Give access to intruders
- Users should be trained on password selection and secrecy

Check your progress

Fill in the blanks

1) Entity Integrity rule is concerned with ……………………… key values

2) A log, which is usually written to the stable storage, contains the …………………. data

4.8 SUMMARY

In this chapter we have seen basic DBMS concepts like what is meant by transaction, which are properties of transaction. Then we have seen what concurrency is, problems associated with concurrency, and concurrency control mechanisms. We have also introduced to what is recovery of data, why it is necessary, and in case of failures which mechanisms are available to recover data back. Then we have studied views from security point of database. We have also introduced to the concept of integrity constraint and seen which all different integrity constraints are available. At end we studied the concept of data security. All these concepts are described in detail in next coming chapters.

4.9 CHECK YOUR PROGRESS - ANSWERS

1)   Primary
2)   redundant

4.10 QUESTIONS FOR SELF - STUDY

1)   Explain the ACID properties.

2)   What do you mean by concurrency? Which are the problems that arise due to concurrency? Also list the concurrency control mechanisms.

3)   Why data recovery is needed? Explain in short t the database recovery mechanisms.

4)   Write a short note on views and security.

5)   Explain different integrity constraints available.

6)   Write a note on data security.

❖ ❖ ❖

NOTES

CHAPTER - 5

Relational Database Design

5.0 OBJECTIVES

After studying this chapter you will be able to –

- Explain need for proper Database.

- Discuss functional dependencies.

- Describe properties of decomposition.

5.1 INTRODUCTION

This chapter introduces you to the concept of normalization, and different normalization roles as well as need for normalization. It also describes two approaches to normalization. - Decomposition & Synthesis.

5.2 NEED FOR PROPER DATABASE

Most of the problems faced at the time of implementation of any system are outcome of a poor database design. In many cases it happens that system has to be continuously modified in multiple respects due to changing requirements of users. It is very important that a proper planning has to be done.

A relation in a relational database is based on a relational schema, which consists of number of attributes.

A relational database is made up of a number of relations and corresponding relational database schema.

This chapter deals with issues involved in design of database schema using relational model.

The goal of a relational database design is to generate a set of relation schema that allows us to store information without unnecessary redundancy and also to retrieve information easily.

One approach to design schemas that are in an appropriate normal form. The normal forms are used to ensure that various types of anomalies and inconsistencies are not introduced into the database.

5.4 UNDESIRABLE PROPERTIES OF BAD DATABASE DESIGN

A database that is not normalized can include data that is contained in one or more different tables for no apparent reason. This is not optional design with regard to security reasons, disk space usage, query retrieval speed, efficiency of database updates and most importantly data integrity.

A database before normalization is one that has not been broken down logically into smaller, more manageable tables.

Consider following database design

EMPLOYEE

Empno	Fname	Lname	Deptname	Location
1	Chetan	Kelkar	Sales	Pune
2	Pushkar	Kate	Marketing	Mumbai
3	Mukta	Kunte	Sales	Pune
4	Deepti	Joshi	Marketing	Mumbai
5	Medha	Pawar	Purchase	Banglore
6	Amit	Sathe	Purchase	Banglore
7	Rajesh	Mohite	Accounts	Nasik

Thus the above database design has following undesirable properties

1. **Repetition of information or Redundancy**

   The aim of the database system is to reduce the redundancy i.e. the information stored must not be repeated. Repetition of information wastes space.

   In above data Deptname (i e Department name) and Location information is repeated for number of times.

2. **Various types of Anomalies**
   a) Insertion anomaly   b) Updation anomaly   c) Deletion anomaly

Insertion anomaly

Insertion of new department is not possible in above data if no corresponding employee is working. And vice versa (i.e. if new Administration Department is to be added and no single employee is assigned to it then we cannot enter information of Administration department unless and until some employee is attached to it)

Updating anomaly

If any of the department or location information is to be updated, then corresponding change has to be done for number of times. (I.e. if purchase department get shifted from Bangalore to pune, corresponding change has to be done for number of times, depending on number of records)

Deletion anomaly

Deletion of employee's record having corresponding unique department information (I.e. In above data if Rajesh's record is deleted corresponding account department's information will get lost). This leads to deletion anomaly.

5.4 FUNCTIONAL DEPENDENCIES

Functional dependencies are constraints on the set of legal relations. They allow us to express facts about the enterprise that we are modeling with our database. The notion of functional dependency generalizes the notion of super key.

Functional dependency can be described as follows. If column2 is functionally dependent on column1, it means that each value in column1 is associated with one and only one value in column2 at a particular point in time. A particular value for column1 will always have the same value in column2. The reverse is not true, however. A particular value for column2 can have multiple corresponding values in column1.

In other words functional dependency can be described as, in relation R if there is a set of attributes called x and other set called y. For each occurrence of x there will be only one occurrence of y. In such case y is called as functionally dependent on x. R=X→ Y

5.5 NORMALIZATION USING FDS

Any DBMS strives to achieve to reduce the anomalies in the database resulting in duplication or loss of data.

Thus a scientific technique called Normalization is used to design a file which will represent database design in a normal form. Thus normalization represents good database design which eliminates the anomalies and inconsistencies.

It was proposed by Dr.E.F.codd (in 1972).

Normalization of data can be looked on as a process during which unsatisfactory relation schemas are decomposed by breaking up their attributes into smaller relation schemas that possess desirable properties

Normalization is the process of reducing redundancy of data in a relational database. Redundant data refers to the data that is stored in more than one location in the database. Data should not be redundant, which means that the duplication of data should be kept to a minimum for several reasons. For example, it is unnecessary to store an employee's home address in more than one table. With duplicate data unnecessary space is used. Confusion is always a threat when, for instance, an address for an employee in one table does not match the address of the same employee in another table.

When considering normalization, some user-related design considerations include
-	What data should be stored in the database?
-	How will the user access the database?
-	What privileges does the user require?
-	How should the data be grouped in the database?
-	What data is the most commonly accessed?
-	How is all data related in the database?
-	What measures should be taken to ensure accurate data?

The Normalization was defined by E.F.Codd as a mathematical basis for deriving tables (rows and columns of data) using relational calculus and a set of transformations that could be applied to data structures to ensure that they met the requirements for a Relational DBMS.

There are two approaches to normalization.
1)	**Decomposition**
2)	**Synthesis**

The process of converting one entity into multiple entities in order to normalize the original is called **decomposition.** An unnormalized single entity will be decomposed into two or more normalized entities via the application of normalization rules.

E.g. The third level of normalization will have no repeating or redundant data attributes and will have entities with attributes fully dependent on that entity's entire primary key and not on each other.

The decomposition approach starts with one relation and associated set of constraints in the form of

1) **Functional dependencies**
2) **Multivalued dependencies**
3) **Join dependencies**

A relation that has any undesirable properties in the form of insertion, deletion or update anomalies, repetition of information or loss of information and replaced by its projections. This results in normal form.

The second approach is synthesis approach. It starts with a set of functional dependencies on a set of attributes. It then synthesizes relations of third normal form(3NF).It is also known as transitivity dependency.

Advantages of Normalization
Organization is brought about by the normalization process, making everyone's job easier – from end user who accesses tables to the database administrator (DBA) who is responsible for the overall management of every object in the database. Data redundancy is reduced, which simplifies data structures and conserves disk space .Because duplicate data is minimized or completely eliminated; the possibility of inconsistent data is greatly reduced.

If a database has been normalized and broken into smaller tables, you are provided with more flexibility as far as modifying existing structures. It is much easier to modify a small table with a small amount of data than to modify one large table that holds all the data. With smaller tables, it is much easier to logically separate and find columns, especially when coding the end user applications, or querying the database.

Additionally, security is also improved in the sense that the DBA can grant access to limited tables to certain users. Security is easier to control when normalization has been accomplished.

Normalization provides numerous benefits to the design of a database, the design and implementation of an application.

1) It gives refined file design and greater overall database organization will be gained.
2) It removes undesirable elements (fields) i.e. the amount of unnecessary redundant data is reduced.
3) It helps in understanding the system
4) It avoids anomalies
5) Data integrity is easily maintained within the database.
6) The database and application design processes are much more flexible.
7) Security is easier to manage.

Disadvantages of Normalization

1) The disadvantages of normalization are that it produces a lot of tables with a relatively small number of columns. These columns then have to be joined using their primary/foreign key relationships in order to put the information back together so we can use it.

   For example a query might require retrieval of data from multiple normalized tables. This can result in complicated table joins. The required tables for the query were probably one before decomposition (normalization).

2) Decomposition of tables has two primary impacts. The first is performance. All the joins required to merge data slow processing down and place additional stress on your hardware.

   The second impact challenges developers to code queries that return desired information, without experiencing the impact of the relational database's insistence on returning a row for every possible combination of matching values if the tables are not properly joined by the developer.

Additional rows that are returned because of tables that are not properly joined (using their key values) are not useful.

Some normalization rules are there which are used as guidelines to normalization

1)   ***First Normal Form***
The objective of the first normal form is to divide the base data into logical units called entities, or tables. When each entity has been designed, a primary key is assigned to it.

It is also called as 'Atomic value rule'. Thus 1st NF states that each attribute in a database must have an atomic value. Multivalued attributes are not allowed in 1st NF.

Thus attribute cannot be subdivided further

A repeating or multivalued attribute is an attribute or group of attributes that will have multiple values for one occurrence of the UID. You would have to repeat the attribute(s) multiple times in the entity structure in order to capture all the possible occurrences of the event being modeled. The solution is to move repeating attribute(s) to their own entity and create a relationship between the two decomposed entities. Working with tables in design, you'd move the repeating columns to their own table along with a copy of the original entity's UID as a foreign key.

"A Relation schema is said to be in 1 NF (First normal form) if the values in the domain of each attribute of the relation are atomic. In other words, only one value is associated with each attribute and the value is not a set of values or list of values."

A table is said to be in first normal form if it has no repeating groups of attributes. In other words, if there is a group of attributes that is repeated, remove it to a separate table.

A database schema is in First Normal Form if every relation schema included in the database schema is in 1st NF.

A 1st NF disallows having a set of values, a tuple of values or combination of both as an attribute value for a single tuple. In other words 1st NF disallows "relations within relations" or "relations as attributes of tuples". The only attribute values permitted by 1st NF are single atomic (or indivisible) values.

The 1st NF also disallows composite attributes that are themselves multivalued. These are called nested relations because each tuple can have a relation within it.

Note the following example

The entity FACULTY_SCHEDULE is meant to track faculties and the courses they teach by the semester. The Faculty_id is the UID for this entity. Note that the pound sign is typically used to identify the primary key for an entity.

FACULTY SCHEDULE

#Faculty_id
fname
mname
lname
year
semester
department1
course1

section_id1
department2
course2
section_id2
department3
course3
section_id3

Because the faculty teaches multiple classes each semester, the class attribute and its attendant department and section attributes would need to have more than one value (be multivalued) in the entity structure. Another way of saying this is that the class attribute would repeat as class1, class2, class3 and so on. These repeating attributes need to be moved to their own entity.

In the previous example, department (1-3), course (1-3) and section_id (1-3) are multivalued attributes or repeating columns. This is violation of FIRST NORMAL FORM normalization rules. The solution is to place department, course and section in their own entity, COURSES_TAUGHT, with a relation to the decomposed INSTRUCTOR entity.

FACULTY	COURSES_TAUGHT
#Faculty_id	#course
fname	#department
mname	#section
lname	#semester
category	#year
category_level	Faculty_id

2) *Second Normal form*

It is based on functional dependency "It says that your file design must be in 1st NF, and all non key attributes must be fully functionally dependent on prime attribute(s) or key attribute(s)"

A relation schema is in second normal form if it is in 1 NF and if all nonprime attributes are fully functionally dependent on the relation key(s).i.e. Prime attributes or key attribute.

A database schema in in 2nd NF if every relation schema included in the database schema is in second normal form.

Each non key attribute within a table with a multiple attribute key is examined to transform a set of tables into second normal form.

Thus the objective of the Second Normal Form is to take data that is only partly dependent on the primary key and enter it into another table

Let's look at our prior example. The COURSES_TAUGHT entity has a composite UID consisting of course, department, section, semester and year required to uniquely identify an instance. Now if someone wants to add an attribute called department_location to the COURSES_TAUGHT entity. Because such an attribute would be dependent only on the attribute department and not the whole composite UID, it would be a violation of SECOND NORMAL FORM. The solution to this is to move the offending attribute(s) to another entity and establish a relationship between the two entities.

FACULTY	COURSES_TAUGHT	DEPARTMENT
#Faculty_id	#course	#department_id
fname		#department
		department_name
mname	#section	department_address
lname	#semester	
category	#year	
category_level		
Faculty_id		

3) Third Normal Form (Transitivity dependency)

It says that the file design must be in 2nd NF. "It also states that non key attributes must be fully fully functionally and non transitively dependent on primary key attribute(s)"

A table is said to be in 3rd NF if it is in second normal form and every non key attribute is dependent on the entire primary key.

A relation schema in 3 NF does not allow partial or transitional dependency.

A database schema is in 3rd NF if every relational schema included in the database schema is in 3rd NF.

The 3rd NF schema, like the 2nd NF does not allow partial dependencies. Furthermore unlike the 2nd NF schema, it does not allow any transitive dependency.

Thus the THIRD NORMAL FORM's objective is to remove data in a table that is not dependent on the primary key. The entity is in SECOND NORMAL FORM and a non-UID attribute can't depend on another non-UID attribute. All non-UID attributes should depend directly on the whole UID and not on each other. Put another way, attributes don't have attributes of their own. If attributes do not have attributes, they're really entities.
For example, the EMPLOYEE entity has an attribute called category. Category initially has potential values of technical, management, administrative or professional.

Later however, we decide to add an attribute called category_level, which further qualifies category into a more detailed subgroup based on expertise level with potential values of 1 form a beginner, 2 for a middle level and 3 for an expert

Here category_level is dependent first on category. Category_level is only dependent on the employee_id entity UID through category. An attribute dependency on the UID- which is not direct but only passes through another attribute that is dependent on the UID- is called **transitive dependency.** Transitive dependencies are unacceptable in THIRD NORMAL FORM. Category and category_level need to be moved from the EMPLOYEE entity to their own EMPLOYEE_CATEGORY entity as a violator of THIRD NORMAL FORM.

FACULTY	COURSES_TAUGHT	DEPARTMENT
#Faculty_id	#course	#department_id
fname		#department_id
	department_name	
mname	#section	department_address
lname	#semester	
category	#year	
Faculty_id		

EMPLOYEE_CATEGORY
#category
category_level

4) Fourth Normal Form

A relation meets the requirements of 4th NF only if it is in 3rd NF (or BCNF) and has no multivalued dependencies. Multivalued dependencies exist only in relations containing at least 3 attributes. The problem arises when a single value of the primary key returns multiple records for 2 or more attributes that are mutually dependent.

Often this means that the relation contains multiple themes. Relations with multivalued dependencies can cause considerable data redundancy and can make data updates difficult.

To take a relation from 3rd NF to 4th NF, it must be decomposed into 2 or more relations. Each relation should address only one theme.

Decomposing the relations into 2 separate relations removes anomalies, reduces redundancies and allows easy updates.

5)   Fifth Normal Form

A relation R is in 5th NF (also called project/join normal form) if and only if every join dependency in R is a consequence of the candidate keys of R i.e. Not implied by the candidate keys of R .If R is in 5th NF it is also in 4th NF.

Discovering join dependencies is not easy as unlike functional dependencies and multivalued dependencies, they do not have a straightforward real world interpretation.

Boyce Codd Normal Form (BCNF)

BCNF was developed to overcome some problems with 3rd NF. A relation meets the requirements of BCNF if every determinant is a candidate key.

Remember a determinant is an attribute on which another attribute is fully functionally dependent.

"A relation R is in BCNF if and only if every determinant is a candidate key"

Any relation R can be non-loss decomposed into an equivalent collection of BCNF relation.

Thus Boyce-codd NORMAL FORM is in effect when an entity is in THIRD NORMAL FORM and every attribute or combination of attributes (a determinant) upon which any attribute is functionally dependent is a candidate key (that is, unique). An attribute or combination of attributes upon which any attribute is functionally dependent is also called a determinant.

If there is only one determinant upon which other attributes depend and it is a candidate key (such as a primary key), THIRD NORMAL FORM and Boyce-codd NORMAL FORM are identical. The difference between THIRD NORMAL FOEM and Boyce-codd NORMAL FORM is that Boyce-codd requires all attributes that have other attributes with functional dependencies on them(are determinants) to be candidate keys and THIRD NORMAL FORM does not. Boyce-codd, with this subtle difference, is in effect a stronger version of THIRD NORMAL FORM.

Fourth Normal Form

Fourth Normal Form is in effect when an entity is in BOYCE-CODD NORMAL FORM ,it has no multivalued dependencies, and neither of the following conditions exists.
1)   The dependent attribute(s) is not a subset of the attribute(s) upon which it depends (the determinant)
2)   The determinant in union (combination) with the dependent attribute(s) includes the entire entity.

A multivalued dependency is a situation in which two or more attributes are dependent on a determinant and each dependent attribute has a specific set of values. The values in these dependent attributes are independent of each other.

Fifth Normal Form

FIFTH NORMAL FORM (also called project-join normal form) is in effect when FOURTH NORMAL FORM is in effect and the entity has no join dependencies that are not related to that entity's candidate keys.

When we decompose (break up) entities as we apply normalization rules, we should be able to reconstruct the original entity by doing joins between the two resulting entities without losing any data and without generating extra and generally incorrect rows. This is called a lossless join.

An entity is in FIFTH NORMAL FORM when it cannot be decomposed into several smaller entities, which have different keys from the original without data losses. If data was lost after decomposition, a lossless-join would not be possible. If all the decomposed entities use one of the original entity's candidate keys as keys, joins of those entities would result in a lossless join. With respect to the original entity, it would still be considered to be in FIFTH NORMAL FORM without decomposition.

Domain Key Normal Form (DKNF)

The idea behind domain key normal form is to specify to specify the "ultimate normal form" that takes into account all possible types of dependencies and constraints. A relation is said to be in DKNF if all constraints and dependencies that should hold on the relation can be enforced simply by enforcing the domain constraints specified on the relation. For a relation in DKNF, it becomes very straightforward to enforce the constraints by simply checking that each attribute value in a tuple is of the appropriate domain and that every key constraint on the relation is enforced. However it seems unlikely that complex constraints can be included in a DKNF relation, hence its practical utility is limited.

Denormalization

As oppose to normalization, denormalization allows some redundancy in fields (for user requirements). It also allows derived fields. When we denormalize data it creates redundancy but it must be controlled redundancy.

Occasionally database designers choose a schema that has redundant information, that is it is not normalized. They use the redundancy to improve performance for specific applications.

For instance, suppose that the name of an account holder has to be displayed along with the account number and balance, every time the account is accessed, In our normalized schema, this requires a join of account with depositor.

One alternative to computing the join is to store a relation containing all the attributes of *account* and *depositor*. This makes displaying the account information faster. However the balance information for an account is repeated for every person who owns the account and all copies must be updated by the application, whenever the account balance is updated. The process of taking a normalized schema and making it nonnormaized is called denormalization.

The penalty paid for not using a normalized schema is the extra work (in terms of coding time and execution time) to keep redundant data consistent.

Thus Denormalization is the process of taking a normalized database and modifying table structures to allow controlled redundancy for increased database performance. There are costs to denormalization, however. Data redundancy is increased in a denormalized database, which might improve performance but requires more extraneous efforts in order to keep track of related data. When denormalization is employed, it's usually derivative of normalized data, so that even if an atomic unit of data exists in several places, it is derived from one source. When a database has been denormalized, you must consider how the redundant data will be managed. Denormalization has a price and should be undertaken only to the degree required to obtain necessary performance. Denormalization generally involves one of these three tasks.

-Replicating some data columns to several tables in order to have them more easily accessible without multi-table joins. This can include data columns themselves or foreign key columns. With replication of data columns, the data is often available to the application without joins. Additional foreign key columns can potentially allow direct connections between tables rather than multiple joins including tables that are only providing a connection, but carry a large time cost in the join.

-Pre-calculation and storage of derived data such as summary calculations or concatenations can speed processing. Create special purpose tables that contain precalculated information required by a commonly run report or create tables that contain data often needed for queries.

Undoing some decomposition of entities to avoid the price of multiple joins. This would primarily be undertaken for tables that have one-to-many relationships.

5.6 PROPERTIES OF DECOMPOSITION

The process of converting one entity into multiple entities in order to normalize the original is called decomposition. An unnormalized single entity will be decomposed into two or more normalized entities via the application of normalization rules.

Any relation having constraints in the form of FDs only can be decomposed into relations in third normal form; such decomposition is lossless and preserves the dependencies. Any relation can also be decomposed losslessly into relations in Boyce codd normal form (and hence into third normal form)

Loss less Join

Let R be a relation schema and let F be a set of functional and multivalued dependencies on R. Let R1 and R2 form a decomposition of R.

Let r(R) be a relation with schema R.We say that the decomposition is a lossless decomposition if for all legal database instances (that is, database instances that satisfy the specified functional dependencies and other constraints).

In other words, if we project r onto R1 and R2, and compute the natural join of the projection results, we get back exactly r. A decomposition that is not a lossless decomposition is called loosy decomposition. The terms *lossless-join decomposition* and *loosy-join decomposition* are sometimes used in place of *lossless decomposition* and *loosy decomposition.*

We can use functional dependencies to show when certain decompositions are lossless

Let R, R1, R2 and F be as above.R1 and R2 form a lossless decomposition of R if at least one of the functional dependencies is in F^+
R1 ∩ R2 →→ R1
R1 ∩ R2 →→ R2

In other words, if R1 ∩ R2 forms a superkey of either R1 or R2, the decomposition of R is a lossless decomposition.

Dependency Preserving

If the decomposition is dependency preserving, given a database update, all functional dependencies can be verifiable from individual relations, without computing a join of relations in the decompositions.
The relation design is dependency preserving if the original set of constraints can be derived from the dependencies in the output of the design process. The design is minimally dependency preserving if there are no extraneous dependencies in the output of the design process and the original dependencies cannot be derived from a subset of the dependencies in the output of design process.

5.5, 5.6 Check your progress

A) Fill in the blanks

1) ……………………….. represents good database design which eliminates the anomalies and Inconsistencies.

2) The Normalization was defined by ……………………….. as a mathematical basis for deriving tables

3) The process of taking a normalized schema and making it nonnormaized is called ………………………..

⟶

5.7 SUMMARY

Normalization is the process of reducing or completely eliminating the occurrence of redundant data in the database. Although in most cases redundant data is not completely eliminated, the greater the reduction of redundancies, the easier data integrity is to maintain. Normalization improves on the structure of a database. The drawback of normalizing a database is the inevitable degradation of database performance. When data is normalized, multiple entities are created that are related through unique identifiers (UIDs). In order to retrieve desired data, entities are merged.

Normal forms represent the level to which a database has been normalized. These are 1NF to 5^{Th} NF.

In this chapter we showed pitfalls in database design, and how to systematically design a database schema that avoids the pitfalls. The pitfalls included repeated information and inability to represent some information. Here we also described the assumptions of atomic domains and first normal form.

We also introduced the concept of functional dependencies, and used it to present two normal forms, Boyce-codd normal form (BCNF) and third normal form (3NF)

Denormalization is the process of recombining attributes in a database that has been normalized. Basically redundant data is allowed in the database for the purpose of improving performance. Denormalization has drawbacks as well, such as increased redundancy, increased storage requirements and increased difficulty in the management of data integrity.

5.8 CHECK YOUR PROGRESS-ANSWERS

5.2, 5.3, 5.4
A)
1)   Super key
2)   Relations
B)
1)   True

5.5, 5.6
A)
1)   Normalization
2)   E.F.Codd
3)   denormalization
B)
1)   False
2)   True

5.9 QUESTIONS FOR SELF - STUDY

1)   Discuss the need for Normalization

2)   What is denormalization? When it is useful?

3)   Discuss the undesirable properties of bad database design.

4)   Explain the concept of Functional dependencies.

5)   Explain 1NF, 2NF, 3NF and BCNF with example.

6)   Explain properties of decomposition.

❖ ❖ ❖

NOTES

SQL Relational Database Language

6.0 OBJECTIVES

In this chapter we study SQL, the most influential commercially marketed query language.

After studying this chapter you will be able to –

- Explain SQL.
- Describe Data types in SQL.
- Discuss DDL, DML, DCL.
- Explain Queries.

6.1 INTRODUCTION

SQL – Relational Database Language

Ideally, a database language must enable us to create database and table structures, to perform basic data management chores (add, delete and modify), and to perform complex queries designed to transform the raw data into useful information. Moreover it must perform such basic functions with minimal user effort, and its command structure and syntax must be easy to learn. Finally it must be portable; that is, it must conform to some basic standard so that we do not have to relearn the basics when we move from one RDBMS to another.

Structured query language, more commonly referred to as SQL, is the standard language used to issue commands to and communicate with a relational database. With SQL, you can easily enter data into the database, modify data, delete and retrieve data. SQL is a non-procedural language, which means that you tell the database server what data to access, not necessarily what methods to access the data.
SQL contains the following three sublanguages that allow you to perform nearly any operation desirable within a relational database.

SQL is relatively easy to learn. Its command set has a basic vocabulary of less than 100 words. Better yet SQL is a nonprocedural language: You merely have to command what is to be done; you don't have to worry about how it is to be done.

Thus it uses combination of Relational algebra and Relational calculus.

Data Types In SQL

1)   char
Values of this data type are fixed length characters strings of maximum length 255 characters.

2) **varchar/varchar2**
   Values of this datatype are variable length character strings of maximum length 2000.
3) **number**
   It is used to store numbers (fixed or floating point).Numbers of virtually any magnitude may be stored upto 38 digits of precision.
4) **date**
   The standard format is DD-MM-YY. To change this format we use appropriate functions
5) **Long**
   It can store variable length character strings containing up to 65,535 characters. Long data can be used to store arrays of binary data in ASCII format.

6.2 DDL – DATA DEFINATION LANGUAGE

DBMS provides a facility known as Data Defination Language (DDL), which can be used to define the conceptual scheme and also give some details about how to implement this scheme in the physical devices used to store the data. This definition includes all the entity sets and their associated attributes as well as the relationships among the entity sets. The definition also contains any constraints that have to be maintained, including the constraints on the value that can be assigned to a given attribute and the constraints on the values assigned to different attributes in the same or different records.

These definations ,which can be described as meta-data about the data in the database are expressed in the DDL of the DBMS and maintained in a compiled form (usually as a set of tables). The compiled form of the definations is known as a data dictionary, dictionary or system catalog.
The database management system maintains the information on the file structure, the method used to effectively access the relevant data; it also provides a method whereby the application programs indicate their data requirements. The application program could use a subset of the conceptual data definition language or a separate language.

The internal schema is specified in a somewhat similar data definition language called data storage definition language. The definition of the internal view is compiled and maintained by the DBMS
 We specify a database schema by a set of definations expressed by a special language called a data definition language (DDL). The DDL is also used to specify additional properties of the 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. In general, a constraint can be an arbitrary predicate pertaining to the database. However arbitrary predicates may be costly to test.

The DDL, just like any other programming language, gets as input some instructions (statements) and generates some output. The output of the DDL is placed in the data dictionary, which contains metadata – that is data about data. The data dictionary is considered to be a special type of table, which can only be accessed and updated by the database system itself (not a regular user). A database system consults the data dictionary before reading or modifying actual data.

Data Defination Language is a part of SQL that is responsible for the creation, updation and deletion of tables. It is responsible for creation of views and indexes also. The list of DDL commands is given below.

CREATE TABLE

Syntax:

Create table table name (columnname datatype (size), columnname datatype (size)…);

ALTER TABLE

Syntax: For addition of new column

1)   Alter table tablename add (newcolumnname datatype (size), newcolumnname datatype (size),….);

2)   For modifying existing column information

        Alter table tablename modify (columnname newdatatype (size));

DROP TABLE

Syntax: for deleting table

Drop table tablename;

CREATE VIEW

Syntax:

Create view viewname as <query expression>

CREATE INDEX

Syntax:

Create index indexname on tablename (columnname);

6.3 DML - DATA MANIPULATION LANGUAGE

The language used to manipulate data in the database is called Data Manipulation Language (DML). Data manipulation involves retrieval of data from the database, insertion of new data into the database, and deletion or modification of existing data.

The first of these data manipulation operations is called a <u>Query</u>. A query is a statement in the DML that requests the retrieval of data from the Database.

The subset of the DML used to pose a query is known as Query Language.

The DML provides commands to select and retrieve data from the database. Commands are also provided to insert, update and delete records. They could be used in, an interactive mode or embedded in conventional programming languages such as Assember, COBOL, FORTRAN, PASCAL or PL/I. The data manipulation functions provided by the DBMS can be invoked in application programs directly by procedure calls or by preprocessor statement. E.g.

<u>Procedure call</u>

Call retrieve (EMPLOYEE, Name, EMPLOYEE. Address)

<u>Preprocessor statement:-</u>

% select EMPLOYEE. Name, EMPLOYEE. Address from EMPLOYEE;

These preprocessor statements indicated by the presence of the leading % symbol, would be replaced by data manipulation language statements in the compiled version of application programs.

There are two types of DMLs:-

1)   Procedural DML:-It requires a user to specify what data are needed and how to get these data.

2)   Non Procedural DML:-It requires a user to specify what data are needed without specifying how to get those data.

In an integrated environment data definition and manipulation are achieved using a uniform set of constructs that forms part of the user's programming environment.

The Query language of SQL is nonprocedural. It takes as an input several tables (possibly only one) and always returns a single table.

Data Manipulation commands manipulate (insert, delete, update and retrieve) data. The DML language includes commands that run queries and changes in data. It includes following commands.

1) **SELECT**
   Syntax: for global data extract
   Select * from tablename;

   Syntax: The retrieval of specific columns from a table
   Select columnname, columnname from tablename;

   Syntax: Elimination of duplicates from the select statement
   Select distinct columnname, columname from tablename;

   Syntax: Sorting of data in a table
   Select columnname, columnname from tablename order by columnname, columnname;

   Syntax: Selecting a data set from table data
   select columnname, columnname from tablename where search condition;

2) **UPDATE**
   Syntax: for updating the contents of a table
   update tablenameset columnname = expression, columnname = expression where columnname = expression ;

3) **DELETE**
   Syntax :Deletion of all rows
   delete from tablename ;

   Syntax:Deletion of a specified number of rows
   delete from tablename where search condition;

4) **INSERT**
   Syntax: Inserting data into table from another table
   Insert into tablename select columnname, columnname from tablename;

   Syntax: Insertion of selected data into table from another table
   Insert into tablename select columnname, columnname from tablename where column = expression;

   Here is an example of SQL query that finds the names of all customers who reside in 'Pune' :

   select customer.customer_name from customer
   where customer.customer_city='Pune' ;

   The query specifies that those rows from the table *customer* where the *customer_city* is *pune* must be retrieved, and the *customer_name* attribute of those rows must be displayed.

   More specifically, the result of executing this query is a table with a single column labeled *customer_name*, and a set of rows, each of which contains the name of a customer whose *customer_city* is *Pune.*

   Queries may involve information from more than one table. For instance, the following query finds the account numbers and corresponding balances of all accounts owned by the customer with *customer_id 192-84-7654*

   Select account.account_number, account.balance from depositor, account
   where depositor.customer_id='192-84-7654' and
   depositor.account_number=account.account_number;

6.4 DCL- DATA CONTROL LANGUAGE

The data control language(DCL) commands are related to the security of the database They perform tasks of assiging privileges. So users can access certain objects in the database.

The DCL commands are
GRANT
REVOKE
COMMIT
ROLLBACK

1)   GRANT
Granting permission using Grant statement :-
The objects created by one user are not accessible by another user unless the owner of those objects give such permissions to other users. These permissions can be given by using the Grant statement. The grant statement provides various types of access to database objects such as tables, views and sequence.

Syntax :-   GRANT { object privileges}
                   ON object Name
                   To UserName
                   {with GRANT OPTION}

2)   REVOKE
The revoke statement is used to deny the grant given on an object
Syntax :-   REVOKE {object privilege}
                   ON object Name
                   FROM User Name ;

3)   COMMIT
Commit command is used to permanently record all changes that the user has made to the database since the last commit command was issued or since the beginning of the database session.
Syntax :-   commit ;

4)   ROLLBACK
The Rollback statement does the exact opposite of the commit statement .It ends the transaction but undoes any changes made during the transaction. Rollback is useful for following 2 reasons.

1)   If you have made a mistake such as deleting the wrong row for a table ,you can use rollback to restore the original data.
2)   Rollback is useful if you have started a transaction that you cannot complete.

         Syntax: rollback[work][to[savepoint]savepoint]

6.1, 6.2, 6.3, 6.4 Check your progress

A) Fill in the blanks

1)   DDL stands for ………………………………………

2)   …………………………….. stands for data control language

3)   ………………………………………is a part of SQL that is responsible for the creation, updation and deletion of tables

4)   The ……………………….. statement does the exact opposite of the commit statement

B) State true or false

1)   DML stands for Data control language

2)   SQL stands for structured query language

3)   Rollback is useful if you have started a transaction that you can not complete.

4)   The revoke statement is used to deny the grant given on an object

5)   delete command is used to remove table

6.5 SIMPLE QUERIES

Consider following tables
EMP
Empno,Ename,Job,Deptno

DEPT
Deptno,Dname,Loc

SQL QUERIES

1) List all those employees who are 'clerk'
   SELECT ENAME,JOB FROM EMP WHERE JOB='CLERK' ;

2) List all those employees who are either 'Analyst' or 'Clerk' with their names,department number and their jobs

   SELECT ENAME,DEPTNO,JOB FROM EMP WHERE
   JOB IN('ANALYST','CLERK')

3) List all employees sorted on their names

   SELECT ENAME FROM EMP ORDER BY ENAME;

4) List all available jobs from EMP

   SELECT DISTINCT JOB FROM EMP;

5) List employee names and their corresponding work/Job locations

   SELECT ENAME, LOC FROM EMP,DEPT WHERE
   EMP.DEPTNO=DEPT.DEPTNO ;

6) List employees, their salaries who work at 'New York'

   SELECT ENAME, SAL FROM EMP,DEPT WHERE
   EMP.DEPTNO=DEPT.DEPTNO AND
   LOC='NEW YORK'

7) List all employees from EMP who work in department number 30

   SELECT ENAME FROM EMP WHERE DEPTNO=30 ;

8) List employee names from EMP whose names are only 4 characters long.

   SELECT ENAME FROM EMP WHERE LENGTH(ENAME) = 4 ;

9) List employee names from EMP whose names end with letter 'R'

   SELECT ENAME FROM EMP WHERE ENAME LIKE '%R';

10) List employees, department names from EMP ,DEPT tables who work in either 'SALES' or 'RESEARCH' department

   SELECT ENAME,DNAME FROM EMP,DEPT WHERE
   EMP.DEPTNO=DEPT.DEPTNO AND
   DNAME IN('SALES','RESEARCH');

11) List all departments whose department numbers are less than 20
   SELECT DEPTNO,DNAME FROM DEPT WHERE DEPTNO<20;

6.6 SUMMARY

The SQL language may be considered one of the major reasons for the success of relational databases in the commercial world. The name SQL is derived

from structured Query Language and was designed and implemented at IBM research as the interface for an experimental relational database system called SYSTEM R. SQL is now the standard language for commercial relational DBMSs.

SQL is a comprehensive database language, it has statements for data definition, query and update. Hence it is both DDL and DML. In addition, it has facilities for defining views on the database, for specifying security and authorization, for defining integrity constraints, and for specifying transaction controls

SQL includes a variety of language constructs for queries on the database. All the relational algebra operations, including the extended relational algebra operations can be expressed by SQL.SQL also allows ordering of query results by sorting on specified attributes.

SQL also allows nested sub queries in the where clause.

SQL supports several types of outer join with several types of join conditions.

―――――――――――――――――――――――――

6.7 CHECK YOUR PROGRESS-ANSWERS

6.1, 6.2, 6.3, 6.4
A)
1) Data definition language
2) DCL
3) Data definition language
4) Rollback

B)
1) False
2) True
3) True
4) True
5) False

6.8 QUESTIONS FOR SELF - STUDY

1) Explain the use of SQL.
2) Which data types are available in SQL
3) Write short notes on following
    -DDL   -DML   -DCL
4) Give syntax of following SQL commands
    a) create table   b) alter table   c)drop table
    d) create view   e) create index   f) select
    g) update   h) delete   i) insert
    j) grant   k) revoke   l)commit
    m) rollback
5) Explain difference between procedural and non procedural DML.

6) Explain the use of distinct clause in SQL commands.

7) What is difference between commit and rollback?

8) Explain granting and revoking permissions.

9) When do you use command 'alter table' and 'update'. Explain with example.

10) What data you get if you try to retrieve data from a table after dropping the same?

❖ ❖ ❖

NOTES

CHAPTER - 7

Security

7.0 OBJECTIVES

This chapter covers authorization and security.

After studying this chapter you will be able to –

- Describe granting access to users.

- Discuss extending and restricting privileges.

- Explain views of security.

7.1 INTRODUCTION

This chapter teaches you granting and restricting privileges to users and views of security. The different types of authorizations are reading data, inserting new data, updating data and deleting data.

7.2 GRANTING ACCESS TO USERS

We may assign a user several forms of authorization on parts of the database. For example

- Authorization to read data
- Authorization to insert new data
- Authorization to update data
- Authorization to delete data

Each of these types of authorizations is called a privilege. We may authorize the user all, none, or a combination of these types of privileges on specified parts of a database, such as a relation or a view.

The SQL standard include the privileges **select, insert, update** and **delete**. The **select** privilege authorizes a user to read data. In addition to these forms of privileges for access to data, SQL supports several other privileges, such as privilege to create, delete or modify relations, and the privilege to execute procedures. The privilege **all privileges** can be used as a short form for all the allowable privileges. A user who creates a new relation is given all privileges on that relation automatically.

A user who has some form of authorization may be allowed to pass on(grant) this authorization to other users, or to withdraw (revoke) an authorization that was granted earlier.

The SQL data-defination language includes commands to grant and revoke privileges. The **grant** statement is used to confer authorization. The basic form of this statement is **grant** <privilege list> **on** <relation name or view name> **to** <user /role list>

The privilege list allows granting of several privileges in one command.

The following grant statement grants the database users *Ashwini* and *Mira* selects authorization on the *account* relation.
grant select on account to Ashwini, Mira

The update authorization may be given either on all attributes of the relation or only some. If **update** authorization is included in a **grant** statement, the list of attributes on which update authorization is to be granted optionally appears in parentheses immediately after the **update** keyword. If the list of attributes is omitted, the update privilege will be granted on all attributes of the relation.

This **grant** statement gives users *Ashwini* and *Mira* update authorization on the amount attribute of the loan relation. grant update(amount) on loan to Ashwini, Mira

The **insert** privilege may also specify a list of attributes; any inserts to the relation must specify only these attributes, and the system either gives each of the remaining attributes default values (if a default is defined for the attribute) or sets them to null.

The user name **public** refers to all current and future users of the system. Thus privileges granted to **public** are implicitly granted to all current and future users.

By default, a user/role that is granted a privilege is not authorized to grant that privilege to another user/role. SQL allows a privilege grant to specify that the recipient may further grant the privilege to another user.

7.2 Check your progress

A) Fill in the blanks

1)   Various types of authorizations are called ……………………………………….

B) State true or false

1)   The select privilege authorizes a user to update data

2)   The privilege **all privileges** can be used as a short form for all the allowable privileges

---

▰▰▰▰▰   7.3 EXTENDING AND RESTRICTING PRIVILEGES   ▰▰▰

To revoke an authorization, we use the **revoke** statement. It takes a form almost identical to that of **grant.**

revoke <privilege list> **on <relation name or view name >**
from <user /role list>[restrict/cascade]

Thus, to revoke the privileges that we granted previously, we write
revoke select on branch **from** Ashwini, ira
revoke update (amount) **on** loan **from** Ashwini, Mira

Revocation of privileges is more complex if the user from whom the privilege is revoked has granted the privilege to another user.

The revocation of a privilege from a user/role may cause other users/roles also to lose that privilege. This behavior is called cascading of the revoke. In most database systems, cascading is the default behavior, the keyword cascade can thus be omitted.

7.3 Check your progress

A) Fill in the blanks

1)   To revoke an authorization, we use the ……………………… statement.

B) State true or false

1)   Revoke is used to authorize the privileges

7.4 USING VIEWS OF SECURITY

It is not desirable for all users to see entire logical model. Security considerations may require that certain data to be hidden from users.

Any relation that is not part of the logical model, but is made visible to user as a virtual relation, is called a **view.** It is possible to support a large number of views on top of any given set of actual relations.

View Defination

We define a view in SQL by using the **create view** command. To define a view, we must give the view a name and must state the query that computes the view. The form of the **create view** command is **Create view v as <**query expression>

Where <query expression> is any legal query expression. The view name is represented by v.

As an example

create view branch_total_loan(branch_name, total_loan) as select branch_name, sum(amount) from loan group by branch_name

The preceding view gives for each branch the sum of amounts of all the loans at the branch. Since the expression sum(amount) does not have a name, the attribute name is specified explicitly in the view definition.

Intuitively, at any given time ,the set of tuples in the view relation is the result of evaluation of the query expression that defines the view at that time. Thus, if a view relation is computed and stored, it may become out of date if the relations used to define it are modified. To avoid this, views are usually implemented as follows. When we define a view, the database system stores the definition of the view itself, rather than the result of evaluation of the relational–algebra expressions that defines the view Wherever a view relation appears in a query, it is replaced by the stored query expression. Thus, whenever we evaluate the query, the view relation gets recomputed.

Certain database systems allow view relations to be stored, but they make sure that, if the actual relations used in the view definition change, the view is kept up to date. Such views are called **materialized views**. The process of keeping the view up to date is called **view maintenance**. Applications that use a view frequently benefit from the use of materialized views, as do applications that demand fast response to certain view-based queries. Of course, the benefits to queries from the materialization of a view must be weighed against the storage costs and the added overhead for updates. Views are also defined by using other views.

A view can hide data that a user does not need to see. The ability of views to hide data serves both to simplify usage of the system and to enhance security. Views simplify system usage because they restrict the user's attention to the data of interest. Although a user may be denied direct access to a relation, that user may be allowed to access part of that relation through a view. Thus a combination of relational-level security and view-level security limits a user's access to precisely the data that the user needs.

In our banking example, consider a clerk who needs to know the names of all customers who have a loan at each branch. This clerk is not authorized to see information regarding specific loans that the customers may have. Thus the clerk must be denied direct access to the *loan* relation. But, if she is to have access to the information needed, the clerk must be granted access to the view cust_loan, which consist of only the names of customers and the branches at which they have a loan. This view can be defined in SQL as follows.
create view cust_loan as

(select branch_name,customer_name from borrower, loan where borrower.loan_number =loan.loan_number)

suppose that the clerk issues the following SQL query
select * from cust_loan

Clearly, the clerk is authorized to see the result of this query. However, when the query processor translates it into a query on the actual relations in the database, it produces a query on *borrower* and *loan.* Thus the system must check authorization on the clerk's query before it begins query processing.

7.4 Check your progress

A) Fill in the blanks

1)   A ………………….. can hide data that a user does not need to see

B) State true or false

1)   It is possible to support a large number of views on top of any given set of actual relations.

2)   Views are also defined by using other views.

7.5 SUMMARY

A user who has been granted some form of authority may be allowed to pass on this authority to other users. However, we must be careful about how authorization can be passed among users if we are to ensure that such authorization can be revoked at some future time.

In this chapter we have seen granting and revoking permissions to/from users.

Any relation that is not a part of the physical database, i.e a virtual relation, is made available to the users as a view. A view represents a different perspective of a base relation or relations.

7.6 CHECK YOUR PROGRESS-ANSWERS

7.2
A)
1)   privileges

B)
1)   false
2)   true

7.3
A)
1)   revoke

B)
1)   False

7.4
A)
1)   view

B)
1)   True
2)   True

7.7 QUESTIONS FOR SELF - STUDY

1)   Explain granting and revoking privileges to users with example.

2)   Write a note on views of security

3)   Which are different data authorizations that you can assign to user

❖ ❖ ❖

NOTES

NOTES

Chapter - 8

Transaction Processing

8.0 OBJECTIVES

In this chapter the concept of transaction is introduced along with properties of transaction.

After studying this chapter you will be able to –

- Explain transaction processing.
- Discrible properties of transaction.
- Discrible states of transaction.
- Discuss serializing and its need.

8.1 INTRODUCTION

This chapter makes you familiar to concepts of transaction, properties and states of transaction, concept of schedule and serializability.

8.2 TRANSACTION PROCESSING

The term transaction refers to a collection of operations that form a single logical unit of work. For example, transfer of money from one account to another is a transaction consisting of two updates, one to each account. The transaction management components of a database system allows application developers to focus on the implementation of individual transactions, ignoring the issues of concurrency and fault tolerance.

A database system must ensure proper execution of transactions despite failures – either the entire transaction executes, or none of it does. Usually a transaction is initiated by a user program written in a high level data-manipulation language or programming language (for example SQL, c++ or Java), where it is delimited by statements (or function calls) of the form **begin transaction** or **end transaction.**

Transaction Model

1)   A transaction is a unit of program execution that accesses and possibly updates various data items.
2)   If the database was in consistent state before a transaction, then on execution of transaction, the database will be in consistent state.
3)   Transaction thus basically is reading information from database and writing information to database.
4)   A transaction must see a consistent database.
5)   During transaction execution the database may be inconsistent.
6)   When the transaction is committed, the database must be consistent.
7)   Two main issues to deal with:
      a.   Failures of various kinds, such as hardware failures and system crashes
      b.   Concurrent execution of multiple transactions

8.3 PROPERTIES OF TRANSACTION

To ensure the integrity of data, database system maintains following properties of transaction. These are **called ACID properties**.

1)   Atomicity
Atomicity property ensures that at the end of transaction, either no changes have occurred to the database or the database has been changed in a consistent manner. At the end of transaction, the updates made by the transaction will be accessible to other transactions and processes outside the transaction

2)   Consistency
Consistency property of transaction implies that if the database was in consistent state before the start of transaction, then on termination of transaction the database will also be in a consistent state

3)   Isolation
This property indicates that action performed by a transaction will be hidden from outside the transaction until the transaction terminates. Thus each transaction is unaware of other transaction executing concurrently in the system.

4)   Durability
Durability ensures that once a transaction completes, i.e successfully commits, the changes it has made to the database persist, even if there are system failures

Example of Fund Transfer

- Transaction to transfer $50 from account *A* to account *B*:
  1.   **read**(A)
  2. $A := A - 50$
  3.   **write**(A)
  4.   **read**(B)
  5. $B := B + 50$
  6.   **write**(B)
- Consistency requirement – the sum of *A* and *B* is unchanged by the execution of the transaction.
- Atomicity requirement — if the transaction fails after step 3 and before step 6, the system should ensure that its updates are not reflected in the database, else an inconsistency will result.
- Durability requirement — once the user has been notified that the transaction has completed (i.e., the transfer of the $50 has taken place), the updates to the database by the transaction must persist despite failures.
- Isolation requirement — if between steps 3 and 6, another transaction is allowed to access the partially updated database, it will see an inconsistent database
  (the sum *A* + *B* will be less than it should be). Can be ensured trivially by running transactions *serially,* that is one after the other. However, executing multiple transactions concurrently has significant benefits, as we will see.
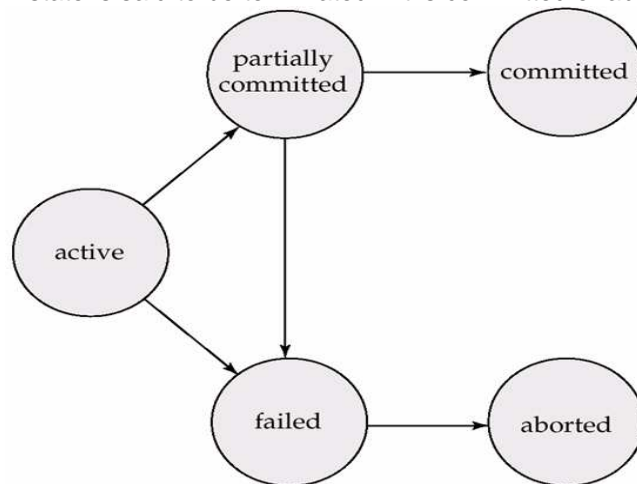
States of transaction

1)   Active
Transaction is active when it is executing. This is the initial state of transaction.

---

2)   Partially committed

When a transaction completes its final statement but it is to be returned to database, then it is in partially committed state.

3)   Failed

If the system decides that the normal execution of the transaction can no longer proceed, the transaction is termed as failed.

4)   Committed

When the transaction completes its execution successfully it enters committed state from partially committed state (i.e transaction is returned to db)

5)   Aborted /Terminated

To ensure the atomicity property, changes made by failed transaction are undone i.e The transaction is rolled back. After roll back, that transaction enters in aborted state.

A state is said to be terminated if it is committed or aborted.

8.3 Check your progress

A) Fill in the blanks

1)   The term ACID stands for Atomicity, consistency, Isolation,

....

2)   Active, ... committed, failed, aborted are the 5 states of transaction.

B) State true or false

1)   Transaction is active when it is executing

2)   Active state of transaction is not an initial state of transaction.

3)   When the transaction completes its execution successfully it enters aborted state

---

8.4 SCHEDULES

Transaction processing systems usually allow multiple transactions to run concurrently. Allowing multiple transactions to update data concurrently causes several complications with consistency of data. Ensuring concurrency in spite of concurrent execution of transactions requires extra work; it is far easier to insist that transactions run serially – that is one at a time, each starting only after the previous one has completed. However there are two good reasons for allowing concurrency :

- Improved throughput and resource utilization

Throughput means the number of transactions executed in a given amount of time Correspondingly, the processor and disk utilization also increase; in other words, the processor and disk spend less time idle, and not performing any useful work.

- Reduced waiting time

If transactions run serially, a short transaction may have to wait for a preceding long transaction to complete, which can lead to unpredictable delays in running a transaction.

If the transactions are operating on different parts of the database, it is better to let them run concurrently, sharing the CPU cycles and disk accesses among them. Concurrent execution reduces the unpredictable delays in running transactions . Moreover, it also reduces the *average response time: the average time for a transaction to be completed after it has been submitted.*

The motivation of using concurrent execution in a database is essentially the same as the motivation for using multiprogramming in an operating system.

Suppose the current values of accounts A and B are $1000 and $2000, respectively.

Suppose also that the two transactions are executed one at a time in the order T1 followed by T2. This execution sequence appears in schedule1.

The final values of accounts A and B, after the execution takes place, are $855 and $2145, respectively. Thus, the total amount of money in accounts A and B -that is, the sum A+B is preserved after the execution of both transactions.

Similarly, if the transactions are executed one at a time in the order T2 followed by T1, then the corresponding execution sequence is that shown in schedule 2. Again as expected, the sum A+B is preserved, and the final values of accounts A and B are $850 and $2150 respectively.

The execution sequences just described are called **schedules**. They represent the chronological order in which instructions are executed in the system. Clearly, a schedule for a set of transactions must consist of all instructions of those transactions, and must preserve the order in which the instructions appear in each individual transaction.

For example, in above example of funds transfer transaction T1, the instruction write(A) must appear before the instruction read(B), in any valid schedule. In the following discussion, we shall refer to the first execution sequence (T1 followed by T2) as schedule 1, and to the second execution sequence (T2 followed by T1) as schedule 2.

These schedules are serial : Each serial schedule consists of a sequence of instructions from various transactions, where the instructions belonging to one single transaction appear together in that schedule. Thus for a set of n transactions, there exists number of different valid serial schedules.

When the database system executes several transactions concurrently, the corresponding schedule no longer needs to be serial. If two transactions are running concurrently, the operating system may execute one transaction for a little while, then perform a context switch, execute the second transaction for some time, and then switch back to the first transaction for some time, and so on. With multiple transactions, the CPU time is shared among all the transactions.

Several execution sequences are possible, since the various instructions from both transactions may now be interleaved. In general, it is not possible to predict exactly how many instructions of a transaction will be executed before the CPU switches to another transaction.

Schedule 1- A serial schedule in which T1 is followed by T2

T1	T2
Read(A)	
A:=A-50	
Write(A)	
Read(B)	
B:=B+50	
Write(B)	read(A)
	temp :=A*0.1
	A := A - temp
	Write(A)
	Read(B)
	B :=B + temp
	Write(B)

Schedule 2- A serial schedule in which T2 followed by T1

T1	T2
	read(A)
	temp :=A*0.1
	A := A - temp
	Write(A)
	Read(B)
	B :=B + temp
	Write(B)
Read(A)	
A:=A-50	
Write(A)	
Read(B)	
B:=B+50	
Write(B)	

Schedule 3- A concurrent schedule equivalent to schedule 1

T1	T2
Read(A)	
A :=A-50	
Write(A)	
	read(A)
	temp :=A*0.1
	A :=A-temp
	Write(A)
Read(B)	
B :=B+50	
Write(B)	
	Read(B)
	B :=B+temp
	Write(B)

Returning to our previous example, suppose that the two transactions are executed concurrently. One possible schedule appears as schedule 3.

After this execution takes place, we arrive at the same state as the one in which the transactions are executed serially in the order of T1 followed by T2. The sum A+B is indeed preserved.

Not all concurrent executions result in a correct state. To illustrate, consider the following schedule. After the execution of this schedule, we arrive at a state where the final values of accounts A and B are $950 and $2100, respectively. This final state is an inconsistent state, since we have gained $50 in the process of the concurrent execution. Indeed, the sum A+B is not preserved by the execution of the two transactions.

If control of concurrent execution is left entirely to the operating system, many possible schedules, including ones that leave the database in an inconsistent state such as the one just described, are possible. It is the job of the database system to ensure that any schedule that gets executed will leave the database in a consistent state. The **concurrency control component** of the database system carries out this task.

Schedule 4 - A concurrent schedule

T1	T2
Read(A)	
A :=A-50	
	Read(A)
	Temp :=A*0.1
	A :=A-temp
	Write(A)
	Read(B)
Write(A)	
Read(B)	
B :=B+50	
Write(B)	
	B :=B + temp
	Write(B)

8.5 SERIALIZING AND ITS NEED

The database system must control concurrent execution of transactions, to ensure that the database state remains consistent .Before we examine how the database system can carry out this task, we must first understand which schedules will ensure consistency, and which schedules will not.

Since transactions are programs, it is computationally difficult to determine exactly what operations a transaction performs and how operations of various transactions interact. For this reason, we shall not interpret the type of operations that a transaction can perform on a data item. Instead we consider only two operations : read and write.

We thus assume that ,between a read(Q) instruction and a write(Q) instruction on a data item Q, a transaction may perform an arbitrary sequence of operations on the copy of Q that is residing in the local buffer of the transaction.

Thus the only significant operations of a transaction, from a scheduling point of view, are its read and write instructions. We shall therefore usually show only read and write instructions in schedules.

A transaction when gets executed, generates a schedule in memory . This schedule is nothing but sequential operations of the transaction.

A serializable schedule is defined as : Given (an interleaved execution)a concurrent schedule for n transactions, the following conditions hold for each transactions in a set.

1)   All transactions are correct. i.e If any of the transactions is executed on a consistent database, the resulting database is also consistent.
2)   Any serial execution of the transactions is also correct and preserves the consistency of the database. There are two forms of serializability
1) conflict serializability   2) view serializability

Precedence graph is used to test the serializability i.e it selects the schedule which will guarantee that the database will be in a consistent state, the most reliable schedule is selected, a cyclic schedule is not desirable as it may lead to deadlock.

Thus serializability is the result of the concurrent execution of transactions is the same as that of the transactions being executed in serial order. This property is important in multiuser and distributed databases, where several transactions are likely to be executed concurrently.

Serializability of schedules generated by concurrently executing transactions can be ensured through one of a variety of mechanisms called concurrency control schemes.

Conflict serializability

Let us consider schedule S in which there are two consecutive instructions Ii and Ij, of transactions Ti and Tj, respectively. If Ii and Ij refers to different data items, then we

can swap Ii and Ii without affecting the results of any instructions in the schedule. However, if Ii and Ij refers to the same data item Q, then the order of the two steps may matter. Since we are dealing with only read and write instructions, there are four cases that we need to consider.

1) Ii = read(Q), Ij = read(Q).The order of Ii and Ij does not matter, since the same value of Q is read by Ti and Tj, regardless of the order.

2) Ii = read(Q), Ij=write(Q). If Ii comes before Ij, then Ti does not read the value of Q that is written by Tj in instruction Ij. If Ij comes before Ii, then Ti reads the value of Q that is written by Tj. Thus the order of Ii and Ij matters.

3) Ii = write(Q), Ij = read(Q). The order of Ii and Ij matters for reasons similar to those of previous case.

4) Ii = wite(Q), Ij =write(Q). Since both instructions are write operations, the order of these instructions does not affect either Ti or Tj. However, the value obtained by the next read(Q) instruction of S is affected, since the result of only the latter of the two write instructions is preserved in the database. If there is no other write(Q) instruction after Ii and Ij in S, then the order of Ii and Ij directly affects the final value of Q in the database state that results from schedule S.

Thus only in the case where both Ii and Ij are read instructions then relative order of their execution does not matter.

We say that Ii and Ij **conflict** if they are operations by different transactions on the **same data item**, and at least one of these instructions is a write operation.

View Serializability

Consider two schedules S1 and S2, where the same set of transactions participates in both schedules. The schedules S1 and S2 are said to be view equivalent if three conditions are met.

T1	T5
Read(A)	
A :=A-50	
Write(A)	
	Read(B)
	B :=B-10
	Write(B)
Read(B)	
B:=B+50	
Write(B)	
	Read(A)
	A :=A+10
	Write(A)

1) For each data item Q, if transaction Ti reads the initial value of Q in schedule S1, then transaction Ti must, in schedule S1, also read the initial value of Q.
2) For each data item Q, if transaction Ti executes read(Q) in schedule S, and if that value was produced by a write(Q) operation executed by transaction Tj, then the read(Q) operation of transaction Ti must ,in schedule S1, also read the value of Q that was produced by the same write(Q) operation of transaction Tj.
3) For each data item Q, the transaction (if any) that performs the final write(Q) operation in schedule S must perform the final write(Q) operation in schedule S1.

Conditions 1 and 2 ensure that each transaction reads the same values in both schedules and, therefore, performs the same computation. Condition 3 coupled with conditions 1 and 2, ensures that both schedules result in the same final system state In our previous examples, schedule 1 is not view equivalent to schedule 2, since in schedule 1, the value of Account A read by transaction T2 was produced by T1, whereas this case does not hold in schedule 2. However schedule 1 is view equivalent to schedule 3, because the values of account A and B read by transaction T2 were produced by T1 in both schedules.

The concept of view equivalence leads to the concept of view serializability. We say that a schedule S is view serializable if it is view equivalent to a serial schedule.

Every conflict-serializable schedule is also view serializable, but there are view serializable schedules that are not conflict serializable.

Observe that in following schedule, transactions T4 and T6 performs write(Q) operations without having performed a read(Q) operation. Writes of this sort are called **blind writes.** Blind writes appear in any view serializable schedule that is not conflict serializable.

8.6 SUMMARY

In this chapter we discussed transaction, properties of transaction etc.A transaction is a program unit whose execution may change the contents of the database. If the database was in a consistent state before a transaction, then on completion of the execution of the program unit corresponding to the transaction the database will be in a consistent state. This requires that the transaction be considered atomic: it is executed successfully or, in case of errors, the user views the transaction as not having been executed at all.

Concurrent execution of transactions implies that the operations from these transactions may be interleaved. This is not the same as serial execution of the transactions, where each transaction is run to completion before the next transaction is started. The serializability criterion is used to test whether or not an interleaved execution of the operations from a number of concurrent transactions is correct. The serializability test consist of generating a precedence graph from a interleaved execution schedule. If the precedence graph is acyclic, the schedule is serializable, which means that the database will have the same state at the end of the schedule as some serial execution of the transaction.

8.7 CHECK YOUR PROGRESS - ANSWERS

8.2
A)
    1)   Inconsistent
    2)   consistent

B)
    1)   True

8.3
A)
    1)   Durability
    2)   Partially committed

B)
    1)   True
    2)   False
    3)   False

8.4
A)
    1)   Throughput

2) average response time

B)

1) True
2) False

8.5

A)

1) Schedule
2) Transaction

B)

1) True

1) Define transaction. Explain transaction model.

2) Describe properties of transaction and their need by giving proper example.

3) Describe states of transaction with diagram.

4) Explain the term schedule and use of it.

5) Write a note on serializability and its need.

6) What do you mean by 'blind writes'.

7) Define the term throughput.

8) Explain conflict and view serializability.

❖ ❖ ❖

NOTES

Chapter - 9

Backup and Recovery

9.0 OBJECTIVES

Friends,

In this chapter we discuss recovery of the data contained in a database system following failures of various types and present the different approaches to database recovery.

After studying this chapter you will be able to –

- Explain types of failure and storage systems.
- Discuss need for backup and recovery.

9.1 INTRODUCTION

This chapter makes you familiar about different types of failures and storage systems. It also describes need for backup and recovery.

9.2 TYPES OF FAILURES AND STORAGE SYSTEMS

A computer system is an electromagnetical device subject to failures of various types. The reliability of the database system is linked to the reliability of the computer system on which it runs. Database recovery techniques are methods of making the database fault tolerant. The aim of the recovery scheme is to allow database operations to be resumed after a failure with minimum loss of information at an economically justifiable cost.

The failure of system occurs when the system does not function according to its specifications and fails to deliver the service for which it was intended

A system is considered reliable if it functions as per its specifications and produces a correct set of output values for a given set of input values. For a computer system, reliable operation is attained when all components of the system work according to the specification. The failure of system occurs when the system does not function according to its specifications and fails to deliver the service for which it was intended

Types of Failures
1)   *Hardware Failures*
       Design errors, poor quality control (poor connections and defective subsystems), over utilization and overloading, wear out (Insulation on wire could undergo chemical changes)
2)   *Software Failures*
       Design errors, poor quality control(undetected errors in entering program code), over utilization and overloading(Buffers and stacks may be overloaded), wear out(usefulness of s/w system may become absolute due to introduction of new versions with additional features).

3) Storage Medium Failure

Storage Media can be classified as volatile, Non volatile, permanent/stable storage

Volatile storage

Eg of this type of storage is semiconductor memory requiring uninterruptible power supply for correct operation. A volatile storage failure can occur due to spontaneous shutdown of the computer system sometimes referred to as a system crash. The cause of the shutdown could be a failure in the power supply unit or a loss of power. A system crash will result in loss of information stored in the volatile storage medium. Another source of data loss from volatile storage can be due to parity errors in more bits than could be corrected by the parity checking unit, such errors will cause partial loss of data..

Non Volatile storage

Examples of this type of storage are magnetic tape and magnetic disk. These types of storage devices do not require power supply for maintaining the stored information. A power failure or system shutdown will not result in loss of information stored on such devices. However nonvolatile storage devices such as magnetic disks may experience a mechanical failure in the form of read/write head crash, which could result in some loss of information. It is vital that failures that can cause the loss of ordinary data should not also cause the loss of redundant data that is to be used for recovery of ordinary data. One method of avoiding this double loss is store the recovery data on separate storage devices. To avoid the loss of recovery data (primary recovery data) one can provide a further set of recovery data (secondary recovery data) and so on.

Permanent or stable storage

It is achieved by redundancy. Thus instead of having a single copy of data on a non volatile storage medium, multiple copies of the data are stored. Each such copy is made on a separate non volatile storage device. Since these independent storage devices have independent failure modes, it is assumed that at least one of these multiple copies will survive any failure and be usable. The amount and type of data stored in stable storage depends on the recovery scheme used in particular DBMS. The status of database at a given point in time is called the archive database and such archive data is usually stored in stable storage. Recovery data that could be used to recover from the loss of volatile as well as nonvolatile storage is also stored on stable storage. Failure of stable storage could be due to natural or manmade disasters. Manually assisted database regeneration is the only possible remedy to permanent stable storage failure.

9.2 Check your progress

A) Fill in the blanks

1) The failure of system occurs when the system does not function according to its ………………………..

2) Examples of this ………………… storage are magnetic tape and magnetic disk.

3) The status of database at a given point in time is called the

……………………. database.

B) State true or false

1) A computer system is an electromagnetical device subject to failures of various types.

2) semiconductor memory is Nonvolatile storage.

3) Failure of stable storage could be due to natural or man made disasters.

9.3 NEED FOR BACKUP AND RECOVERY

The types of failures that the computer system is likely to be subjected to include failures of components or subsystems, s/w failures, power cutages, accidents, unforeseen situations and the natural or manmade disasters. Database recovery techniques are the methods of making the database fault tolerant. The aim of the

---

recovery scheme is to allow database operations to be resumed after a failure with minimum loss of information at an economically justifiable cost.

Log Based Recovery

A log, which is usually written to the stable storage, contains the redundant data required to recover from volatile storage failures and also from errors discovered by the transaction or the database system. For each transaction the following data is recorded on the log.

- A start of transaction marker
- The transaction identifier, which could include the who and where information
- The record identifiers, which include the identifiers for the record occurrences.
- The operation(s) performed on the records (insert, delete, modify)
- The previous values of the modified data. This information is required for undoing the changes made by a partially completed transaction, it is called the undo log .Where the modification made by the transaction is the insertion of a new record, the previous values can be assumed to be null.
- The updated values of the modified record(s). This information is required for making sure that the changes made by a committed transaction are in fact reflected in the database and can be used to redo these modifications. This information is called the redo part of the log. In case the modification made by the transaction is the deletion of a record, the updated values can be assumed to be null.
- A commit transaction marker if the transaction is committed, otherwise an abort or rollback transaction marker.

The Log is written before any updates are made to the database. This is called the write ahead log strategy. In this strategy a transaction is not allowed to modify the physical database until the undo portion of the log (i.e portion of the log that contains the previous values of the modified data) is written to the stable storage

Furthermore the log write-ahead strategy requires that a transaction is allowed to commit only after the redo portion of the log and the commit transaction marker are written to the log. In effect both the undo and redo portions of the log will be written to stable storage before a transaction commit. Using this strategy, the partial updates made by an uncommitted transaction can be undone using the undo portion of the log, and a failure occurring between the writing of the log and the completion of updating the database corresponding to the actions implied by the log can be redone.

Refer to diagram of Write Ahead Log Strategy

Let us see how the log information can be used in the case of a system crash with the loss of volatile information. Consider a number of transactions. The fig shows the system start-up at time t0 and a number of concurrent transactions T0,T1,T2….Ti+6 are made on the database. Suppose a system crash occurs at time tx.

We have stored the log information for transactions T0 through Ti+2 on stable storage, and we assume that this will be available when the system comes up after the crash.

Furthermore, we assume that the database existing on the nonvolatile storage will also be available. It is clear that the transactions that were not committed at the time of system crash will have to be undone. The changes made by these uncommitted transactions will have to be rolled back. The transactions that have not been committed can be found by examining the log, and those transactions that have a start of transaction marker but no commit or abort transaction marker are considered to have been active at the time of the crash. These transactions have to be rolled back to restore the database to a consistent state. In figure, the transactions Ti and Ti+6 started before the crash, nut they had not been committed and hence, are undone.

Checkpoints

In an on-line database system, for example in airline reservation system, there could be hundreds of transactions handled per minute. The log for this type of database contains a very large volume of information. A scheme called checkpoint is used to limit the volume of log information that has to be handled and processed in the event of a system failure involving loss of volatile information. The checkpoint scheme is an additional component of the logging scheme described above.

In the case of a system crash, the log information being collected in buffers will be lost. A checkpoint operation performed periodically, copies log information onto stable storage. The information and operations performed at each checkpoint consist of the following.

- A start of checkpoint record giving the identification that it is a checkpoint along with the time and date of the checkpoint is written to the log on a stable storage device.
- All log information from the buffers in the volatile storage is copied to the log on stable storage.
- All database updates from the buffers in the volatile storage are propagated to the physical database.
- An end of checkpoint record is written and the address of the checkpoint record is saved on a file accessible to the recovery routine on start-up after a system crash.

For all transactions active at checkpoint, their identifiers and their database modification actions, which at that time are reflected only in the database buffers, will be propagated to the appropriate storage.

The frequency of check pointing is a design consideration of the recovery system. A checkpoint can be taken at fixed intervals of time (say, every 15 minutes). If this approach is used, a choice has to be made regarding what to do with the transactions that are active when the checkpoint signal is generated by a system timer. In one alternative, called **transaction consistent checkpoint**, the transactions that are active when the system timer signals a checkpoint are allowed to complete, but no new transactions (requiring modifications to the database) are allowed to be started until the checkpoint is completed. This scheme though attractive makes the database unavailable at regular intervals and may not be acceptable for certain online applications. In addition, this approach is not appropriate for long transactions. In the second validation, called **action consistent checkpoint, active** transactions are allowed to complete the current step before the checkpoint and no new actions can be started on the database until the checkpoint is completed, during the checkpoint no actions are permitted on the database. Another alternative, called **transaction oriented checkpoint,** is to take a checkpoint at the end of each transaction by forcing the log of the transaction onto stable storage . In effect, each commit transaction is a checkpoint.

Suppose, as before, the crash occurs at time tx. Now the fact that a checkpoint was taken at time tc indicates that at that time all log and data buffers were propagated to storage. Transactions T0.....Ti-1 as well as Ti+1 and Ti+3 were committed, and their modifications are reflected in the database. With the checkpoint mechanism these transactions are not required to be redone during recovery operation following system crash occurring after time tc. A transaction such as Ti (which started before checkpoint time tc), as well as Transaction Ti+6(which started after checkpoint time tc), were not committed at the time of the crash and have to be rolled back. Transactions such as Ti+4 and Ti+5 which started after checkpoint time tc and were committed before the system crash, have to be redone. Similarly, transactions such as Ti+2, which started before the checkpoint time and were committed before the system crash, will have to be redone. However if the commit transaction information is missing for any of the transactions Ti+2,Ti+4 or Ti+5 then they have to be undone.

Let us now see how the system can perform a recovery at time tx. Suppose all transactions that started before the checkpoint time but were not committed at that time, as well as the transactions started after the checkpoint time, are placed in an undo list, which is a list of transactions to be undone. The undo list for the transactions is given below

UNDO List (Ti,Ti+2,Ti+4,Ti+5,Ti+6)

Now the recovery system scans the log in a backward direction from the time tx of the system crash If it finds that a transaction in the undo list has committed, that transaction is removed from the undo list and placed in the redo list. The redo list contains all the transactions that have to be redone. The reduced undo list and the redo list for the transactions is

REDO List (Ti+4, Ti+5, Ti+2)
UNDO List (Ti, Ti+6)

Obviously, all transactions that were committed before the checkpoint time need not be considered for the recovery operation. In this way the amount of work required to be done for recovery from a system crash is reduced. Without the checkpoint scheme, the redo list will contain all transactions except Ti and Ti+6. A system crash occurring during the checkpoint operation requires recovery to be done using the most recent previous checkpoint.

Shadow Page Table
- Shadow paging is an alternative to log-based recovery; this scheme is useful if transactions execute serially
- Idea: maintain two page tables during the lifetime of a transaction – the current page table, and the shadow page table
- Store the shadow page table in nonvolatile storage, such that state of the database prior to transaction execution may be recovered.
  - o Shadow page table is never modified during execution
- To start with, both the page tables are identical. Only current page table is used for data item accesses during execution of the transaction.
- Whenever any page is about to be written for the first time
  - o A copy of this page is made onto an unused page.
  - o The current page table is then made to point to the copy
  - o The update is performed on the copy

To commit a transaction :
- *   Flush all modified pages in main memory to disk
- *   Output current page table to disk
- *   Make the current page table the new shadow page table, as follows:
- *   keep a pointer to the shadow page table at a fixed (known) location on disk.
- *   to make the current page table the new shadow page table, simply update the pointer to point to current page table on disk
- *   Once pointer to shadow page table has been written, transaction is committed.
- *   No recovery is needed after a crash — new transactions can start right away, using the shadow page table.
- *   Pages not pointed to from current/shadow page table should be freed (garbage collected).
- ✓ Advantages of shadow-paging over log-based schemes
  - o no overhead of writing log records
  - o recovery is trivial

- ✓ Disadvantages :
  - o Copying the entire page table is very expensive
    - *   Can be reduced by using a page table structured like a B+-tree
    - *   No need to copy entire tree, only need to copy paths in the tree that lead to updated leaf nodes
  - o Commit overhead is high even with above extension
    - *   Need to flush every updated page, and page table
  - o Data gets fragmented (related pages get separated on disk)
  - o After every transaction completion, the database pages containing old versions of modified data need to be garbage collected
  - o Hard to extend algorithm to allow transactions to run concurrently
    - *   Easier to extend log based schemes

The shadow page scheme is one possible form of the indirect page allocation. The memory that is addressed by a process (a program in execution is a process) is called virtual memory. It is divided into pages that are assumed to be of a certain size, let us say 1024 (1 k) bytes or more commonly 4096 (or 4k) bytes. The virtual or logical pages are mapped onto physical memory blocks of the same size as the pages, and the mapping is provide by means of a table known as page table. The page table contains one entry for each logical page of the process's virtual address space. With this scheme, the consecutive logical pages need not be mapped onto consecutive physical blocks.

In the shadow page scheme, the database is considered to be made up of logical units of storage called pages. The pages are mapped into physical blocks of storage (of the same size as the logical pages) by means of a page table of the database. This entry contains the block number of the physical storage where this page is stored.

The shadow page table scheme illustrated in diagram uses two page tables for a transaction that is going to modify the database. The original page table is called the shadow page table and the transaction addresses the database using another page table known as the current page table. Initially both page tables point to the same blocks of physical storage. The current page table entries may change during the life of the transaction. The changes are made whenever the transaction modifies the database by means of a write operation. The pages that are affected by transaction are copied to new blocks of physical storage and these blocks along with the blocks not modified, are accessible to the transaction via the current page table. The old version of the changed pages remains unchanged and these pages continue to be accessible via the shadow page table.

The shadow page table contains the entries that existed in the page table before the start of the transaction and points to the blocks that were never changed by the transaction. The shadow page table remains unaltered by the transaction and is used for undoing the transaction.

Now let us see how the transaction accesses data during the time it is active. The transaction uses the current page table to access the database blocks for retrieval. Any modification made by the transaction involves a write operation to the database. The shadow page table scheme handles the first write operation to a given page as follows.

- A free block of nonvolatile storage is located from the pool of free blocks accessible by the database system.
- The block to be modified is copied onto this block.
- The original entry in the current page table is changed to point to this new block.
- The updates are propagated to the block pointed to by the current page table, which in this case would be the newly created block.

Subsequent write operations to a page already duplicated are handled via the current page table. Any changes made to the database are propagated to the blocks pointed to by the current page table. Once a transaction commits, all modifications made by the transaction and still in buffers are propagated to the physical database (i.e the changes are written to the blocks pointed to by the current page table). The propagation is confirmed by adopting the current page table as the table containing the consistent database. The current page table or the active portion of it could be in volatile storage. In this case a commit transaction causes the current page table to be written to nonvolatile storage.

In the case of a system crash before the transaction commits, the shadow page table and the corresponding blocks containing the old database, which was assumed to be in a consistent state, will continue to be accessible.

To recover from system crashes during the life of transaction, all we have to do is revert to the shadow page table so that database remains accessible after the crash. The only precaution to be taken is to store the shadow page table on stable storage and have a pointer that points to the address where the shadow page table is stored and that is accessible to the database through any system crash.

Committing a transaction in the shadow page table scheme requires that all the modifications made by the transaction be propagated to physical storage and the current page table be copied to stable storage.

9.3 Check your progress

A) Fill in the blanks

1) A checkpoint operation performed periodically, copies log information onto ………………….. storage.

2) The shadow page table scheme uses …………………… page tables for a transaction that is going to modify the database.

B) State true or false

1) A log, is usually written to the stable storage.

2) Shadow page table is modified during execution

9.4 SUMMARY

A computer system, like any other mechanical or electrical device, is subject to failure. There are a variety of causes of such failure, including disk crash, power failure and software errors. In each of these cases, information concerning the database system is lost.

The various types of storage in a computer are volatile storage, nonvolatile storage and stable storage. Data in volatile storage, such as in RAM, are lost when the computer crashes. Data in nonvolatile storage, such as disk, are not lost when the computer crashes,

But many occasionally be lost because of failures such as disk crashes. Data in stable storage are never lost.

In case of failure, the state of the database system may no longer be consistent, that is it may not reflect a state of the world that the database is supposed to capture. To preserve consistency, we require that each transaction be atomic. It is the responsibility of the recovery scheme to ensure the atomicity and durability property.

Efficient implementation of a recovery scheme requires that the number of writes to the database and to stable storage be minimized.

To recover from failures that result in the loss of nonvolatile storage, we must dump the entire contents of the database onto stable storage periodically-say once per day.

9.5 CHECK YOUR PROGRESS - ANSWERS

9.2
A)
    1)   specifications
    2)   nonvolatile
    3)   archive
B)
    1)   True
    2)   False
    3)   True

9.3
A)
    1)   stable
    2)   two
B)
    1)   True
    2)   False

9.6 QUESTIONS FOR SELF - STUDY

1) Explain the difference between the three storage types –volatile, nonvolatile and stable.

2) Explain the purpose of checkpoint mechanism.

3) Explain the difference between a system crash and a "disaster"

4) Explain what is failure. what are different types of failures.

5) Explain the need for backup and recovery .

6) Which are different recovery mechanisms available. Describe each in detail.

❖ ❖ ❖

NOTES

Chapter - 10

Concurrency Control and Recovery Techniques

10.0 OBJECTIVES

In this chapter the concept of concurrency, problems that arises due to it, deadlocks and concurrency control mechanisms are introduced.

After studying this chapter you will be able to –

- Discuss concurrency problems.
- Explain concurrency control mechanism
- Describe dead-lock situation.

10.1 INTRODUCTION

This chapter introduces you with the concept of concurrency. It also describes different concurrency problems, concurrency control mechanisms, deadlocks and deadlock handling detection and prevention.

10.2 CONCURRENCY PROBLEMS

Concurrency occurs in a multi-user system where at a time more than one user try to make transactions and it is called concurrency.

So when several transactions are executed concurrently, the corresponding schedule is called concurrent schedule. But due to concurrency following problems may occur.

1)   **Lost Update Problem**
     Consider a concurrent modification of A's bank account with original balance of 500$.

     Due to concurrent transactions T1 and T2, the changes made by transaction T1 is lost. i.e The original balance is 500$, after A's transaction T1 becomes 450 but due to concurrent transaction T2 made by T2 made by B before updating the changes of transaction T1. Balance after, B's transaction is 400$ instead of 350$.

2)   **Inconsistent Read Problem**
     When only one transaction modifies a given set of data while that set of data is being used by other transaction. In following case we have two transactions
     1)   Transaction T1 -- Transfer Rs.100 from A(Bal 500) to B(Bal 1000)
     2)   Transaction T2--Calculate the sum of current balance of A and B

In this case due to inconsistent read problem 3 different results may possible
1)   500+1000 = 1500
2)   500+1100 = 1600
3)   400+1000= 1400

This happens because Transaction t2 calculates the sum of balance of A and B at the same time when Transaction T1 is happening.

3)   The phantom phenomenon

Inventory	**Receive**	**Usage**
Part_no	Part_no	Part_no
Bal_qty	Rec_qty	Used_qty

In above 3 database files inventory maintains the current balance of every part after new transaction i.e on received and used of that part. If some parts are received it is necessary to lock usage file, so that inventory file updates the information first and update quantity is used for the usage.

But if lock is not available on usage and 2nd transaction takes place simultaneously i.e accepting qty for a part and using the qty for the same part. In this case since usage file is unlock the 2nd transaction may be available in a flash and record may be entered in a usage file without checking the previous update. This is known as phantom phenomenon.

4)   Semantics of concurrent transaction
T1=(salary(500)+1000)*1.1   answer is   1650
T2=(salary(500)*1.1)+1000   answer is   1550

In above two transactions if they work concurrently the data storage problem occurs because the sequence of two transactions is different.

10.3 CONCURRENCY CONTROL MECHANISMS

1)   Time stamp ordering
In the timestamp based method, a serial order is created among the concurrent transaction by assigning to each transaction a unique non decreasing number. The usual value assigned to each transaction is the system clock value at the start of the transaction, hence name Timestamp ordering.

A transaction with a smaller timestamp value is considered to be an 'older' transaction than another transaction with a larger timestamp value.

The serializability that the system enforces in the chronological order of the timestamps of the concurrent transactions. If two transactions Ti and Tj with the timestamp values ti and tj respectively, such that ti<tj are to run concurrently, then the schedule produced by the system is equivalent to running the older transaction Ti first, followed by the younger one Tj.

The contention problem between two transactions in the timestamp ordering system is resolved by rolling back one of the conflicting transactions. A conflict is said to occur when an older transaction tries to read a value that is written by a younger transaction or when an older transaction tries to modify a value already read or written by a younger transaction. Sometimes rollback could be of type cascading rollback where a single transaction failure leads to a series of transaction rollback.

Thus the timestamp-ordering protocol guarantees serializability since all the arcs in the precedence graph are of the form:

- There will be no cycles in the precedence graph
- Timestamp protocol ensures freedom from deadlock as no transaction ever waits.
- But the schedule may not be cascade-free, and may not even be recoverable.
- Problem with timestamp-ordering protocol:
  - ★ Suppose T_i aborts, but T_j has read a data item written by T_i
  - ★ Then T_j must abort; if T_j had been allowed to commit earlier, the schedule is not recoverable.
  - ★ Further, any transaction that has read a data item written by T_j must abort
  - ★ This can lead to cascading rollback --- that is, a chain of rollbacks
  - Solution:
    - ★ A transaction is structured such that its writes are all performed at the end of its processing
    - ★ All writes of a transaction form an atomic action; no transaction may execute while a transaction is being written

A transaction that aborts is restarted with a new timestamp

2) Locking Protocol

From the point of view of locking scheme a database can be considered as being made up of set of data items.

A lock is a mechanism to control concurrent access to a data item

A lock is a variable associated with each data item which gives status about the availability of the data item. The value of lock variable is used in the locking scheme to control the concurrent access and manipulation of data associated data item. The locking is done by a subsystem of the database management system called Lock Manager .

Lock requests are made to concurrency-control manager. Transaction can proceed only after request is granted.

A **locking protocol** is a set of rules followed by all transactions while requesting and releasing locks. Locking protocols restrict the set of possible schedules.

Three types of locks are there.
1) Exclusive Locks :

It is also called update or write lock. The intension of this mode of locking is to provide exclusive use of the data item to one transaction. If a transaction T locks the data item Q in an exclusive mode, no other transaction can access Q, not even to read Q until the lock is released by T.

2) Shared Lock :

It is also called as read lock. Any no of transactions can concurrently lock and access a data item in a shared mode .But none of these transactions can modify the data item .A data item locked in a shared mode can't be locked in exclusive mode, until the shared lock is released.

3) wait lock:

It is used when data item is locked by some other transaction. Normally a wait queue is maintained to keep track of such transactions

To implement the locking concept following protocols are used.
1) Two phase protocol

It uses two phases for locking .
i.e release of the locks is delayed until all the locks on all data items required by the transaction have been acquired. Two phases are
a) Growing phase

In this the number of locks increases from zero to maximum for the transaction. In this phase the transaction may obtain lock but may not release any lock.
b) Shrinking phase

In this the number of locks decreases from maximum to zero. A transaction may release locks, but may not obtain any new lock.

Thus

This is a protocol which ensures conflict-serializable schedules.

Phase 1: Growing Phase
- ★ transaction may obtain locks
- ★ transaction may not release locks

Phase 2: Shrinking Phase
- ★ transaction may release locks
- ★ transaction may not obtain locks

The protocol assures serializability. It can be proved that the transactions can be serialized in the order of their **lock points** (i.e. the point where a transaction acquired its final lock).

2) *Graph based or intension locking protocol.*

It provides explicit locking at the lower level of the tree and intension locks are applied in all ancestors. Advantage of this protocol is that lock manager knows that some where the lower level node is locked without having the actual examination of lower level nodes.

Thus Graph-based protocols are an alternative to two-phase locking

Impose a partial ordering $\rightarrow$ on the set D = {d_1, d_2 ,..., d_h} of all data items.

- ★ If $d_i \rightarrow d_j$ then any transaction accessing both d_i and d_j must access d_i before accessing d_j.
- ★ Implies that the set D may now be viewed as a directed acyclic graph, called a *database graph*.

The *tree-protocol* is a simple kind of graph protocol.

Tree protocol
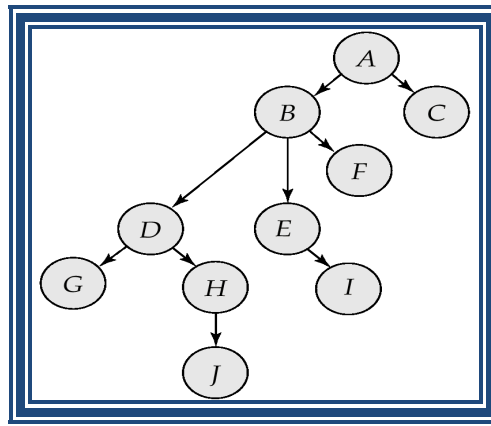
- Only exclusive locks are allowed.
- The first lock by T_i may be on any data item. Subsequently, a data Q can be locked by T_i only if the parent of Q is currently locked by T_i.
- Data items may be unlocked at any time.
- The tree protocol ensures conflict serializability as well as freedom from deadlock.
- Unlocking may occur earlier in the tree-locking protocol than in the two-phase locking protocol.
  - ★ shorter waiting times, and increase in concurrency
  - ★ protocol is deadlock-free, no rollbacks are required
  - ★ the abort of a transaction can still lead to cascading rollbacks.
- However, in the tree-locking protocol, a transaction may have to lock data items that it does not access.
  - ★ increased locking overhead, and additional waiting time
  - ★ potential decrease in concurrency
- Schedules not possible under two-phase locking are possible under tree protocol, and vice versa.

3)   Validation Techniques

In validation techniques or optimistic scheduling scheme it is assumed that all data items can be successfully updated at the end of the transaction and to read in the data values without locking. Reading is done if the data item is found to be inconsistent (with respect to value read in) at the end of the transaction, then the transaction is rolled back. There are 3 phases in optimistic scheduling.

1)   Read phase:
Various data items are read and stored in temporary local variables. All operations are performed on these variables without actually updating the database.

2)   validation phase
All concurrent data items in the database are checked. i.e It checks the serializability. If serializability is violated the transaction is aborted and started again.

3)   Write phase
If the transaction passes validation phase the modification made by the transaction are committed i.e Written to the database. Optimistic scheduling does not use any lock hence it is deadlock free. However it faces the problem of data starvation.

Thus in validation based protocols
Execution of transaction T_i is done in three phases.
1.   Read and execution phase: Transaction T_i writes only to temporary local variables
2.   Validation phase: Transaction T_i performs a "validation test" to determine if local variables can be written without violating serializability.
3.   Write phase: If T_i is validated, the updates are applied to the database; otherwise, T_i is rolled back.

-    The three phases of concurrently executing transactions can be interleaved, but each transaction must go through the three phases in that order.
-    Also called as *optimistic concurrency control* since transaction executes fully in the hope that all will go well during validation

4)   Multiversion Technique (MVT)

In MVT each write and update of the data item is achieved by making a new copy (version) of data item. MVT is called as Time Domain Addressing Scheme. It follows accounting principle of "Never overwrite a transaction". Hence effect of the latest version is taken into consideration.

History of evolution of data items is recorded in the database. In case of read operations database management system selects one of the versions for processing.
-    Multiversion schemes keep old versions of data item to increase concurrency.
    ★   Multiversion Timestamp Ordering
    ★   Multiversion Two-Phase Locking
-    Each successful **write** results in the creation of a new version of the data item written.
-    Use timestamps to label versions.
-    When a **read**(*Q*) operation is issued, select an appropriate version of *Q* based on the timestamp of the transaction, and return the value of the selected version.
-    **read**s never have to wait as an appropriate version is returned immediately.

There are two major problems in case of MVT.
1)   It is little bit slower in operation as every time a new version is generated.
2)   Rollbacks are expensive(cascading)

10.4 DEADLOCK

In the concurrent mode of operation each concurrently running transaction may be allowed to exclusively claim one or more of a set of resources. Some of the problems with this mode of operations are that of deadlock and starvation

A deadlock is said to occur when there is a circular chain of transactions, each waiting for the release of a data item held by the next transaction in the chain .The algorithm to detect a deadlock is based on the detection of such a circular chain in the current system **wait-for-graph.** The wait-for-graph is a directed graph and contains nodes and directed arcs; the nodes of the graph are active transactions. An arc of the graph is inserted between two nodes if there is a data item required by the node at the tail of the arc, which is being held by the node at the head of the arc.

- Consider the following two transactions:
  T_1: write (X) T_2: write(Y)
          write(Y) write(X)

- **Schedule with deadlock**

T_1	T_2
lock-X on X write (X)	
	lock-X on Y write (X) wait for **lock-X** on X
wait for **lock-X** on Y	

- System is deadlocked if there is a set of transactions such that every transaction in the set is waiting for another transaction in the set.

10.5 DEADLOCK HANDLING DETECTION AND PREVENTION

- *Deadlock prevention* protocols ensure that the system will *never* enter into a deadlock state. Some prevention strategies

  ★ Require that each transaction locks all its data items before it begins execution (pre declaration).
  ★ Impose partial ordering of all data items and require that a transaction can lock data items only in the order specified by the partial order (graph-based protocol).

Deadlock can be precisely described in terms of a directed graph called a **wait-for – graph.** To detect deadlock the system needs to maintain the wait for graph and periodically to invoke an algorithm that searches for a cycle in the graph

In above example the wait graph in (a) is acyclic but wait graph in (b) is cyclic which can lead to deadlock. Thus a deadlock can be detected.

In the deadlock detection and recovery approach, the philosophy is to do nothing to avoid a deadlock. However, the system monitors the advance of the concurrent transactions and detects the symptoms of deadlock, namely a chain of transactions all waiting for a resource that the next transaction in the chain has obtained in an exclusive mode.

Wait-die

One solution in a case of contention for a data item is as follows
- If the requesting transaction is older than the transaction that holds the lock on the requested data item, the requesting transaction is allowed to wait.
- If the requesting transaction is younger than the transaction that holds the lock on the requested data item, the requesting transaction is aborted and rolled back.
This is called the wait-die scheme of deadlock prevention.
wait-die scheme — non-preemptive
* older transaction may wait for younger one to release data item. Younger transactions never wait for older ones; they are rolled back instead.
* a transaction may die several times before acquiring needed data item

Wound-Wait

- wound-wait scheme — preemptive
- older transaction *wounds* (forces rollback) of younger transaction instead of waiting for it. Younger transactions may wait for older ones.
- may be fewer rollbacks than *wait-die* scheme

An opposite approach to the wait-die scheme is called the wound-wait scheme. Here the decision whether to wait or abort is as follows.
- If a younger transaction holds a data item requested by an older one, the younger transaction is the one that would be aborted and rolled back(the younger transaction is wounded by the older transaction and dies!)
- If a younger transaction requests a data item held by an older transaction, the younger transaction is allowed to wait.

Both in *wait-die* and in *wound-wait* schemes, a rolled back transactions is restarted with its original timestamp. Older transactions thus have precedence over newer ones, and starvation is hence avoided.

We observe that in neither the wait-die scheme nor the wound-wait scheme is it required to abort and roll back an older transaction. In this way the older transaction would eventually get all the data items it needs and would run to completion. This implies that this scheme minimizes problem of starvation. However, the waiting that may be required by an older transaction could be significantly higher in the wait-die scheme. This is because the older transaction has to wait for younger transaction to finish using popular data items. On the other hand, in the wound-wait scheme, the older a transaction gets, the greater is the probability of acquiring the data item. An older transaction would force the abortion of any younger transaction that holds data items it needs, but would only be aborted by a transaction older than itself. However as a transaction gets older, the number of more senior transactions would decrease.

In the wait-die and the wound-wait schemes the first word of the scheme name indicates what the older transaction does when there is contention for a data item. In the first scheme the older transaction waits for the younger transaction to finish. In the second scheme, the older transaction wounds the younger transaction, which releases the data item for the older transaction. The second component indicates what a younger transaction does when there is a contention with an older transaction. In the
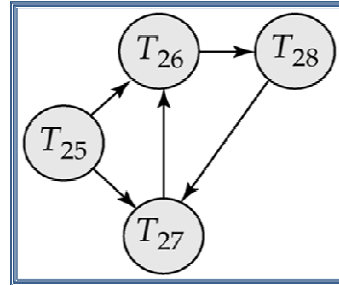
first scheme the younger transaction is aborted and in the second, the younger transaction waits.

The number of aborts and rollbacks tends to be higher in the wait-die scheme than in the wound-wait scheme.

Deadlock Detection (Cont.)
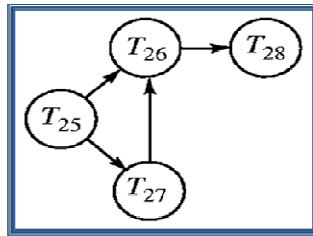
Wait-for graph without a cycle   Wait-for graph with a cycle

- When deadlock is detected :
❋ Some transaction will have to rolled back (made a victim) to break deadlock. Select that transaction as victim that will incur minimum cost.
❋ Rollback -- determine how far to roll back transaction
  ➢ Total rollback: Abort the transaction and then restart it.
  ➢ More effective to roll back transaction only as far as necessary to break deadlock.
❋ Starvation happens if same transaction is always chosen as victim. Include the number of rollbacks in the cost factor to avoid starvation

10.4, 10.5 Check your progress

A) Fill in the blanks

1) A ………………… is said to occur when there is a circular chain of transactions, each waiting for the release of a data item held by the next transaction in the chain

2) Deadlock prevention protocols ensure that the system will ………………… enter into a deadlock state.

3) The ………………… is a directed graph and contains nodes and directed arcs; the nodes of the graph are active transactions.

B) State true or false

1) The number of aborts and rollbacks tends to be higher in the wait-die scheme than in the wound-wait scheme.

2) Both in *wait-die* and in *wound-wait* schemes, a rolled back transactions is restarted with its original timestamp.

3) In wound wait, older transaction may wait for younger one to release data item.

10.6 SUMMARY

Concurrency

When several transactions execute concurrently in the database, the consistency of data may no longer be preserved. It is necessary for the system to control the interaction among the concurrent transactions, and this control is achieved through one of a variety of mechanisms called concurrency control schemes.

To ensure serializability, we can use various concurrency control schemes. All these schemes either delay an operation or abort the transaction that issued the operation. The most common ones are locking protocols, timestamp ordering schemes, validation techniques and multiversion schemes.

Deadlock

A deadlock is said to occur when there is a circular chain of transactions, each waiting for the release of a data item held by the next transaction in the chain

Various locking protocols do not guard against the deadlocks. One way to prevent deadlock is to use an ordering of data items, and to request locks in a sequence consistent with the ordering.

Another way to prevent deadlock is to use preemption and transaction rollbacks. To control the preemption, we assign a unique timestamp to each transaction. The system uses these timestamps to decide whether a transaction should wait or roll back. If a transaction is rolled back, it retains its old timestamp when restarted. The wound-wait scheme is a preemptive scheme.

If deadlocks are not prevented ,the system must deal with them by using a deadlock detection and recovery scheme. To do so, the system constructs a wait-for – graph. A system is in a deadlock state if and only if the wait-for-graph contains a cycle. When the deadlock detection algorithm determines that a deadlock exists, the system must recover from the deadlock. It does so by rolling back one or more transactions to break the deadlock.

Source : *http://ptucse.loremate.com(Link)*

◼ ▬▬▬▬ 10.7 CHECK YOUR PROGRESS - ANSWERS ▬▬▬ ◼

10.2, 10.3
A)
1) MVT
2) Shrinking
3) Exclusive lock

B)
1) True
2) False
3) False
4) False

10.4, 10.5
A)
1) deadlock
2)   never
3)   wait-for-graph

B)
1)   True
2)   True
3)   False

◼ ▬▬▬▬ 10.8 QUESTIONS FOR SELF-STUDY ▬▬▬ ◼

1) Explain the term concurrency.
2) Describe the problems associated with concurrency.
3) Why concurrency control is required. Which are different concurrency control mechanisms. Explain.
4) What do you mean by older transaction in timestamp ordering?
5) Explain what are conflicting transactions.
6) When a deadlock situation occurs ? describe in detail.
7) How deadlock situation is prevented ?
8) What is difference between wait-die and the wound-wait schemes ?
9) What is starvation?
10) What is wait-for-graph ? How deadlock is detected ?

❖ ❖ ❖

NOTES

Chapter - 11

Introduction to Data Warehousing and Data Mining

11.0 OBJECTIVES

After studying this chapter you will be able to –

- Describe data warehouse.

- Describe components of data warehouse.

- Describe data mining and its application.

11.1 INTRODUCTION

This chapter gives you brief idea about data warehousing and data mining. It also makes you familiar with components of data warehouse and applications of data mining.

11.2 DATA WAREHOUSING

Large companies have presences in many places, each of which generate a large volume of data. For instance, large retail chains have hundreds or thousands of stores ,whereas insurance companies may have data from thousands of local branches. Further, large organizations have a complex internal organization structure, and therefore different data may be present in different locations, or on different operational systems, or under different schemas. For instance, manufacturing – problem data and customer – complaint data may be stored on different database systems. Corporate decision makers require access to information from all such sources. Setting up queries on individual sources is both cumbersome and inefficient. Moreover, the sources of data may store only current data, whereas decision makers may need access to past data as well; for instance, information about how purchase patterns have changed in the past year could be of great importance. Data warehouses provide a solution to these problems.

A data warehouse is a repository (or archive) of information gathered from multiple sources, stored under a unified schema, at a single site. Once gathered, the data are stored for a long time ,permitting access to historical data. Thus data warehouses provide the user a single consolidated interface to data, making decision support queries easier to write. Moreover by accessing information for decision support from a data warehouse, the decision maker ensures that online transaction processing systems are not affected by the decision support workload.
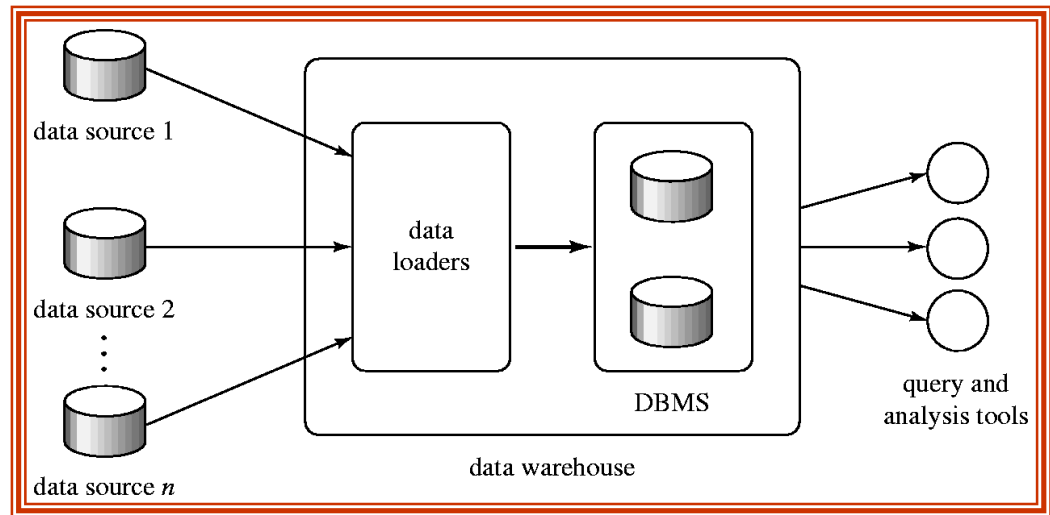
Components of a Data Warehouse

Following figure shows the architecture of a typical data warehouse, and illustrates the gathering of data, the storage of data, and the querying and data analysis support. Among the issues to be addressed in building a warehouse are the following.

1)   **When and how to gather data :**
In a source driven architecture for gathering data, the data sources transmit new information, either continually (as transaction processing takes place), or periodically (nightly, for example). In a destination – driven architecture, the data warehouse periodically sends requests for new data to the sources.

Unless updates at the sources are replicated at the warehouse via two-phase commit, the warehouse will never be quite up-to-date with the sources. Two phase commit is usually far too expensive to be an option, so data warehouses typically have slightly out-of-date data. That however, is usually not a problem for decision support systems.

2)   **What schema to use**
Data sources that have been constructed independently are likely to have different schemas. In fact ,they may even use different data models. Part of the task of a warehouse is to perform schema integration, and to convert data to the integrated schema before they are stored. As a result, the data stored in the warehouse are not just a copy of the data at the sources. Instead, they can be thought of as a materialized view of the data at the sources.

3)   **Data transformation and cleansing**
The task of connecting and preprocessing data is called **data cleansing.** Data sources often deliver data with numerous minor inconsistencies, which can be corrected. For example, names are often misspelled, and addresses may have street/area/city names misspelled, or zip codes entered incorrectly. These can be corrected to a reasonable extent by consulting a database of street names and zip codes in each city. The approximate matching of data required for this task is referred to as **fuzzy lookup.**

4)   **How to propagate updates**
Updates on relations at the data sources must be propagated to the data warehouse. If the relations at the data warehouse are exactly the same as those at the data source, the propagation is straight – forward. If they are not, the problem of propagating updates is basically the view maintenance problem.

5)   **What data to summarize**
The raw data generated by a transaction processing system may be too large to store online. However, we can answer many queries by maintaining just summary data obtained by aggregation on a relation, rather than maintain the entire relation. For example, instead of storing data about every sale of clothing, we can store total sale of clothing by item name and category.

The different steps involved in getting data into a data warehouse are called as **extract, transform, and load** or **ETL** tasks; extraction refers to getting data from the sources, while load refers to loading the data into the data warehouse.

11.2 DATA MINING

The term **data mining** refers loosely to the process of semi automatically analyzing large databases to find useful patterns. Like knowledge discovery in artificial intelligence (also called as machine learning) or statistical analysis, data mining attempts to discover rules and patterns from the data. However data mining differs from machine learning and statistics in that it deals with large volumes of data, stored primarily on disk. That is, data mining deals with "knowledge discovery in databases"

Some types of knowledge discovered from a database can be represented by a set of rules. The following is an example of a rule, stated informally:" young women with annual incomes greater than $50,000 are the most likely people to buy small sports cars". Of course such rules are not universally true, and have degrees of "support" and "confidence", as we shall see. Other types of knowledge are represented by equations relating different variables to each other, or by other mechanisms for predicting outcomes when the values of some variables are known.

There are a variety of possible types of patterns that may be useful, and different techniques are used to find different types of patterns. We shall study a few examples of patterns and see how they may be automatically derived from a database.

Usually there is a manual component to data mining, consisting of preprocessing data to a form acceptable to the algorithms and post processing of discovered patterns to find novel ones that could be useful. There may also be more than one type of pattern that can be discovered from a given database, and manual interaction may be needed to pick useful types of patterns. For this reason, data mining is really a semiautomatic process in real life. However, in our description we concentrate on the automatic aspect of mining.

Applications of Data Mining

The discovered knowledge has numerous applications. The most widely used applications are those that require some sort of prediction. For instance, when a person applies for a credit card, the credit-card company wants to predict if the person is a good credit risk. The prediction is to be based on known attributes of the person such as age, income, debts and past debt repayment history. Rules for making the prediction are derived from the same attributes of past and current credit card holders, along with their observed behavior, such as whether they defaulted on their credit card dues. Other types of prediction include predicting which customers may switch over to a competitor (these customers may be offered special discounts to tempt them not to switch), predicting which people are likely to respond to promotional mail ("junk mail"), or predicting what types of phone calling card usage are likely to be fraudulent.

Another class of applications looks for **associations**, for instance, books that tend to be bought together. If a customer buys a book, an on-line bookstore may suggest other associated books. If a person buys a camera, the system may suggest accessories that tend to be bought along with cameras. A good salesperson is aware of such patterns and exploits them to make additional sales. The challenge is to automate the process. Other types of associations may lead to discovery of causation. For instance, discovery of unexpected associations between a newly introduced machine and cardiac problems led to the finding that the medicine may cause cardiac problems in some people. The medicine was then withdrawn from the market.

Associations are an example of **descriptive patterns. Clusters** are another example of such patterns. For example, over a century ago a cluster of typhoid cases was found around a well, which led to the discovery that the water in the well was contaminated and was spreading typhoid. Detection of clusters of disease remains important even today.

Classification deals with predicting the class of test instances ,by using attributes of the test instances ,based on attributes of training instances, and the actual class of training instances. Classification can be used, for instance to predict the performance of applicants to a university.

Association rules identify items that co-occur frequently, for instance items that tend to be bought by the same customer. Correlations look for deviations from expected levels of associations.

11.3 Check your progress

A) Fill in the blanks

1)   ………………….. deals with "knowledge discovery in databases"

2)   Data mining attempts to discover rules and ………………….. from the data.

B) State true or false

1)   Data mining is same as data warehousing.

2)   Associations are an example of descriptive patterns.

11.4 SUMMARY

Data warehousing

Data warehouses help gather and archive important operational data. Warehouses are used for decision support and analysis on historical data, for instance to predict trends. Data cleansing from input data sources is often a major task in data warehousing. Warehouse schemas tend to be multidimensional involving one or a few very large fact tables and several much smaller dimension tables

Data mining

Data mining is the process of semi automatically analyzing large databases to find useful patterns. There are a number of applications of data mining, such as prediction of values based on past examples, finding of associations between purchases, and automatic clustering of people and movies.

11.5 CHECK YOUR PROGRESS-ANSWERS

11.2
A)
1)   data warehouse
2)   load
3)   data cleansing.
4)   destination –driven

B)
1)   False
2)   False

11.3
A)
    1)   Data mining
    2)   Patterns

B)
    1)   False
    2)   True

11.6 QUESTIONS FOR SELF - STUDY

1)   Explain what do you mean by data warehousing ?

2)   What is data mining ?

3)   Write a note on components of data warehouse.

4)   What do you mean by ETL. Explain.

5)   Write a note on applications of data mining.

❖ ❖ ❖

NOTES

QUESTION BANK

1) Explain what do you mean by data warehousing ?
2) What is data mining ?
3) Write a note on components of data warehouse.
4) What do you mean by ETL. Explain.
5) Write a note on applications of data mining.
6) Explain the term concurrency.
7) Describe the problems associated with concurrency.
8) Why concurrency control is required. Which are different concurrency control mechanisms. Explain.
9) What do you mean by older transaction in timestamp ordering?
10) Explain what are conflicting transactions.
11) When a deadlock situation occurs ? describe in detail.
12) How deadlock situation is prevented ?
13) What is difference between wait-die and the wound-wait schemes ?
14) What is starvation?
15) What is wait-for-graph ? How deadlock is detected ?
16) Explain the difference between the three storage types–volatile, nonvolatile and stable.
17) Explain the purpose of checkpoint mechanism.
18) Explain the difference between a system crash and a "disaster"
19) Explain what is failure. what are different types of failures.
20) Explain the need for backup and recovery.
21) Which are different recovery mechanisms available. Describe each in detail.
22) Define transaction. Explain transaction model.
23) Describe properties of transaction and their need by giving proper example.
24) Describe states of transaction with diagram
25) Explain the term schedule and use of it.
26) Write a note on serializability and its need.
27) What do you mean by 'blind writes'
28) Define the term throughput.
29) Explain conflict and view serializability.
30) Explain granting and revoking privileges to users with example.
31) Write a note on views of security
32) Which are different data authorizations that you can assign to user
33) Explain the use of SQL.
34) Which data types are available in SQL

35) Write short notes on following
   -DDL   -DML   -DCL
36) Give syntax of following SQL commands
   a) create table   b) alter table   c)drop table
   d) create view   e) create index   f) select
   g) update   h) delete   i) insert
   j) grant   k) revoke   l)commit
   m) rollback
37) Explain difference between procedural and non procedural DML
38) Explain the use of distinct clause in SQL commands.
39) What is difference between commit and rollback
40) Explain granting and revoking permissions.
41) When do you use command 'alter table ' and 'update '.Explain with example.
42) What data you get if you try to retrieve data from a table after dropping the same
43) Discuss the need for Normalization
44) What is denormalization? When it is useful?
45) Discuss the undesirable properties of bad database design.
46) Explain the concept of Functional dependencies.
47) Explain 1NF, 2NF, 3NF and BCNF with example.
48) Explain properties of decomposition.
49) Explain the ACID properties
50) What do you mean by concurrency? Which are the problems that arise due to concurrency? Also list the concurrency control mechanisms.
51) Why data recovery is needed? Explain in short t the database recovery mechanisms.
52) Write a short note on views and security.

53) Explain different integrity constraints available
54) Write a note on data security.
55) Explain the distinctions among the terms primary key, candidate key and super key.
56) Explain the difference between a weak and strong entity set with example.
57) Define the concept of aggregation with example.
58) Design a generalization-specialization hierarchy for a motor vehicle sales company. The company sales motorcycles, passenger cars, vans and buses.

   Justify your placement of attributes at each level of the hierarchy. Explain why they should not be placed at a higher or lower level.
59) Explain different conventions that you follow while drawing ERD
60) What do you mean by single valued and multivalued attributes. Explain with example?
61) Explain following terms
   -Entity   -Entity set   -Relationship
   -Relationship set   -Attribute   -Domain
62) Identifying proper entities, d raw entity relationship diagram for hospital management system for OPD (out door patients) only.
63) Discuss the differences between the following file organizations
    - Sequential
    - Index sequential
    - Hash files
64) Write a note on storage devices characteristics.
65) Explain the terms Data, Database ,DBMS
66) What is DBMS, explain its characteristics.
67) Write a note on advantages and Limitations of DBMS
68) State and explain various users of DBMS
69) What do you mean by persistent data? Give any 4 examples.
70) Explain Physical and Logical Data Independence.
71) Explain 3-tier architecture of DBMS
72) Explain software components of DBMS and write their functions
73) What is data model? Describe the type and significance with proper example.
74) State and explain the responsibilities of DBA
75) Describe following terms
   -Attribute   -Domain   -Tuple
   -Relation   -Schema   -Instance
76) Explain codd's rules
77) State difference between single and multi user system
78) State difference between HDM, NDM, and RDM
79) Explain tree structure diagrams and data structure diagrams

❀ ❀ ❀