SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF MANAGEMENT

SBAA7035-Database Management System

UNIT 1 INTRODUCTION **9 Hrs**

Introduction to Database - Data -View of Data, Advantages of DBMS, Disadvantages of DBMS - Database System Structure-Database Language - DDL, DML, DCL, TCL.

UNIT 1

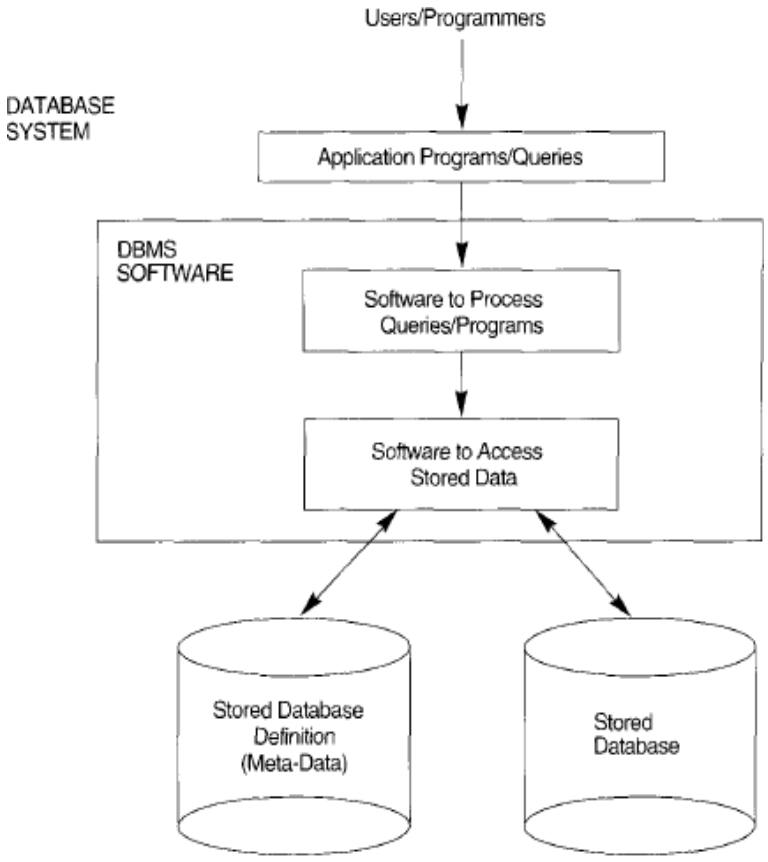
Data means known facts or raw fats. E.g. names, telephone numbers.

Information means processed data.

Database is a collection of related data. E.g. student table consists of name, regno, marks. **Database management system (DBMS)** is collection of programs that enables user to create and maintain a database. A general-purpose software system facilitates process of defining, constructing, and manipulating database for various applications.

Database system includes database and DBMS software.

A simplified database environment.



A simplified database system environment.

Figure 1.1 Database system Environment

CHARACTERISTICS OF DBMS:

- **Self-describing nature of DB:**

  The database system contains data and definition of the database. The data definition is stored on the system **catalog**, which contains the structure of the files, data type for each data item and various constraints on the data. The information stored in the catalog is called **meta data**

- **Insulation between program, data, and data abstraction:**

In the DBMS system, the structure of the file should be stored separately from the access program so, whenever we modify anything in the DB or access program this will not affect the original structure. We call this property as **program-data-independence**.

In object, oriented DB system the operation becomes a part of DB system. This operation consists of two parts called interface and implementation. The interface includes operation name and data type and implementation represents method of the operation. Thus, the method or the implementation should be change without affecting the interface is called as **program-operation-independence**.

- **Support of multiple views of data:**

The multi-user DBMS can provide a facility for defining a multiple views. The view may be a subset of the db or it may contain the virtual data, that it is derived from the original db file

.so, depends upon the user specification the DBMS will display a various types of views.

Example: consider the student table

NAME	REGNO	ADDRESS	PHONENO	PERCENTAGE

VIEW1 VIEW2

- **Sharing of data and multi-user transaction processing:**

The DBMS allow the multi-user to access the db at the same time. It must support the concurrency control software that several users trying to update the same data, the result of the updates should be correct. E.g.: airline ticketing, on line banking, railway reservation.

****

ACTORS ON THE SCENE:

The person those who are involved on the project and those who are using the database are called actors.

1.1.I. Data Base Administrators (DBA):

- The job of DBA is to manage the db resources.

- DBA is responsible for co-ordinating, monitoring and authorizing access to the db, and to provide whatever hardware, software needed . . . Everything should be monitored by DBA.

II DB designer:

- Db designers are responsible for identifying the data to be stored in the db, as well as they choose the appropriate structure to represent this data.
- The db designer should communicate with all the users for understanding their requirements.

III End Users:

- The end users are the people whose job requires accessing the db for querying, updating and generating reports.

Types of End-users:

a) Casual End users:

- They occasionally access the db, but they may need different information each time.
- The user sophisticated db query language to specify their requirements.
- Ex: queries like "list the trains from Chennai to Delhi?"

b) Parametric or Naive End users:

- Their job is constantly querying and updating the db using standard types of queries called **canned transaction**.
- Example: Bank teller, Reservation clerk, etc.

c) Sophisticated End users:

- They have the thorough knowledge about the db.
- Example: Engineers, Scientist, Business analyst etc. They have thorough knowledge about the DBMS.
- They will implement their applications and meet their complex requirements very easily.

d) Stand-alone End-users:

- Maintain the personal db by using the readymade program packages.
- This program packages will provide easy-to-use, menu or graphic based interfaces.

IV System analyst and Application programmers:

- System analyst determines the requirements of end users especiallyparametric end users.
- They develop a specification for canned transactions that meet their requirements.
- Application programmers translate these requirements into programs then they test, debug, document and maintain these canned transaction. Such programmer's are called s/w engineers.

Workers behind the scene:

- DBMS system designer and implementors.
- Tool developers
- Operators and maintain personnel.

****

ADVANTAGES OF DBMS

1. **Controlling Redundancy:-**
   - Redundancy is storing the same data multiple times The storage space is wasted and makes the db file becomes inconsistent.
   - In file processing system, the data files are stored along with the program files. When a user is wants to create an application, he has to create and maintain separate data files along with the program files. Because of this, much of the data is stored more than once. However, in the database system a single database is created and stored once and which can be used by different users.

2) **Restricting unauthorized access:**
   - When multiple users share a database, some users will not be authorized to access all information in the db like some users to read the data only and some users they permitted to modify the data also.Example: Financial data base like banking database, military data etc are accessed only by authorized person.
   - These securities must be provided by **Security and authorization subsystem** in DBMS.

3) **Providing persistent storage for program objects and data structure:**
   Databases can be used to provide permanent storage for program objects and data structures**.**

4) **Providing multiple user interfaces:**
   - Different users have the different knowledge to use a db so, the DBMS should provide a variety of interfaces such as,
     Query language for casual end users
     Programming language for application programmers Forms and commands for parametric end-users Menu-driven interfaces for stand-alone end-users.

   The forms-commands and menu-driven interfaces are called as *Graphical user interfaces (GUI).*

5) **Permitting inferencing and actions using rules:**
   Some data base systems provide capabilities for defining deduction rules for finding new information from stored database. Such systems are called deductive databases.

6) **Representing complex relationship among data:**
   - The database may include varieties of data that are related to each other.
   - The DBMS has the capability to represent the relationships among these different data's.

7) **Enforcing Integrity Constraint:**
   - DBMS should specify a set of rules or restrictions for defining the data in the db.
   - Example:
     Name must be a string of no more than 30 characters. The key field should not be null.

8) **Providing backup and recovery:**
   - DBMS must provide facilities for recovering from h/w or s/w failures.
   - The **backup and recovery subsystem** of the DBMS is responsible for recovery process.

- Example for updating the complex data, at the middle computer system fails then the recovery system is responsible for restoring a state and starts the point at which it was interrupted.

Disadvantages of DBMS

There are many advantages and disadvantages of DBMS (Database Management System). Disadvantages of DBMS are explained as following below.

1. Increased Cost:

These are different types of costs:

1. **Cost of Hardware and Software –**
   This is the first disadvantage of database management system. This is because for DBMS, it is mandatory to have a high speed processor and also a large memory size because now a days there is a large amount of data in every field which needs to be store safely and with a security.
   The requirement of these large amount of space and a high speed processor needs an expensive hardware and an expensive software too. That is, there is a requirement of sophisticated hardware and software which means that we need to upgrade the hardware which is used for file-based system. Hardware and Software, both requires maintenance which costs very high. All the operating, Training (all levels including programming, application development, and database administration), licensing, and regulation compliance costs very high.

2. **Cost of Staff Training –**
   Educated staff (database administrator, application programmers, data entry operations) who maintains the database management system also requires good amount. We need the database system designers to be hired along with application programmers. Alternatively the services of some software house need to be taken. So there is a lot of money which needs to be spent for developing software.

3. **Cost of Data Conversion –**
   We need to convert our data into database management system, there is a requirement of lot of money as it adds on to the cost of the database management system. This is because, for this conversion we need to hire database system designers whom we have to pay a lot of money and also services of some software house will be required. All this shows that a high initial investment for hardware, software and trained staff is required by DBMS. So, altogether Database Management System results in a costlier system.

2. Complexity:

As we all know that now a days all companies are using the database management system as it fulfils lots of requirement and also solves the problem. But a problem arises, that is all these functionality has made database management system an extremely complex software. For the proper requirement of DBMS it is very important to have a good knowledge of it by the developers, DBA, designers and also the end users. This is because if any one of them do not acquire a proper and complete skills than this may lead to data loss or database failure.

These failure may lead to bad design decisions due to which there may be a serious and bad consequences for the organization. So this complex system needs to be understood by everyone using it. As it cannot be managed very easily. All this shows that database management system is not a child's game as it cannot be managed very easily. It requires a lot of management. A good staff is needed to manage this database at times when it becomes very complicated to decide where to pick data from and where to save it.

3. Currency Maintenance:

This is very necessary to keep your system current because efficiency which is one of the biggest factor and need to be overlook must be maximised. That is we need to maximise the efficiency of the database system to keep our system current. For this, frequent updation must be performed on all the components as new threats come daily. DBMS should be updated according to the current scenario. Also, security measures must be implemented. Due to advancement in database technology, training cost tends to be significant.

4. Performance:

Traditional file system is written for small organizations and for some specific applications due to which performance is generally very good. But for the small scale firms, DBMS does not give a good performance as its speed is very slow. As a result some applications will not run as fast as they could. Hence it is not good to use DBMS for the small firms. Because performance is a factor which is overlooked by everyone. If performance is good than everyone (developers, designers, end users) will use it easily and it will be user friendly too. As speed of the system totally depends on the performance so performance needs to be good.

5. Frequency Upgrade/Replacement Cycles:

Now a days in this world we need to stay up-to-date about the latest technologies, developments arriving in the market. Frequent upgrade of the products is done by the DBMS vendors in order to add new functionality to the systems. New upgrade versions of the software often come bundled. Sometimes these updates also need hardware upgrades. Sometimes these changes and updates are so fast that the users find it difficult to work with that system because it is not easy to learn new commands and understand them again when the new upgrades are done. All these upgrades also cost money in order to train users, designers etc. to use the new features.

DATABASE SYSTEM CONCEPTS AND ARCHITECTURE:

Data models:

- ❖ Data model is a collection of concepts that can be used to describe the structure of db.

Categories of Data models:

1) High level or conceptual data model:-

- ➢ Conceptual DM provides concepts that explains the different ways to perceive data and uses the concepts such as entities, attributes and relationships.
- ➢ **Entity** represents the real world object, for example employee or project.

- **Attributes** represents the properties or the further description of entity. For example employee name or salary.
- **Relationship** represents the interaction among the entities. For example works-on relationship between employee and project.
- **Ex:** entity relationship model

2) Low level or Physical DM:-

- This will provide the concept of how the data is stored in the computer
- The storage format is also specified in this Data Model such as, record format, record ordering and access path.

3) Representational or Implementation DM:-

- This is the intermediate DM between high level and low level.
- It provides the concepts that may be understood by end users but that are not too far removed from the way data is stored in the computer.
- Ex: relational model, network model, hierarchical model.

Schema or intension:

The description of a database is called the schema or intension.

Instance or occurrences:

Each row in the database i.e. a set of related data's.

Extension or database state or snapshot:

The data in the database at a particular moment is called database state or extension, which is the current set of instances. At initial state of the database, the database state is said to be empty.

****

1.2 THREE SCHEMA ARCHITECTURE:

The schemas are defined at three levels

1. Internal level or Physical level or Low level
2. Conceptual level or High level
3. External level or View level

Internal level:

- It has an internal scheme, which describes the physical storage structure of the database by means of different data structures link list, queue, stack etc.
- It uses a Physical data model.
- It is useful for computer scientist.

Conceptual Level:

- It has a conceptual schema, which describes the structure of the whole database.
- It describes data as entities, attributes, & relationships.
- It hides the details of physical storage structures.

- It uses high-level data model or implementation data model.
- It can be understood by end users.

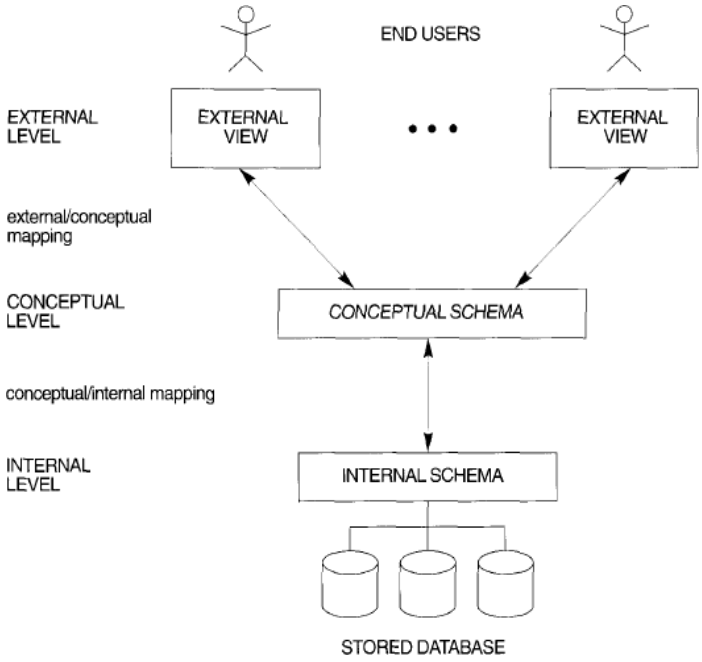


Figure 1.2 Three level Architecture Diagram

External Level:

- It includes a number of external schemas
- It describes the part of the data base that a particular user group is interested in and hides the rest of the data base from that group
- It uses high-level data model or implementation data model.
- It can be understood by end users.

Most DBMS do not separate the 3 levels completely but support three schema architecture to some extent. Some DBMS may include internal schema details in the conceptual schema.

Mappings:

The three schemas are only descriptions of database. The data is actually stored in the database. If a particular user wants to retrieve a data, he has to place a request in the external level. The DBMS must transform this request specified on the external schema into a request against the conceptual schema and then into a request on the internal schema for processing over the stored database. Hence, the retrieved data is reformatted and sent back to the user through the external view. Thus, the processes of transforming requests and results between levels are called Mappings.

DATA INDEPENDENCE:

It is the capacity to change the schema at one level of a database system without having to change at the next higher level. There are two types.

1. Logical data independence
2. Physical data independence

Logical data independence:

It is the capacity to change the conceptual schema without having to change the external schema. Only the mappings between conceptual and external schema need to be changed.

Physical data independence:

It is the capacity to change the internal schema without having to change the conceptual schema. Only the mappings between conceptual and internal schema need to be changed.

STRUCTURED QUERY LANGUAGE

- **Data defintion language(DDL)**
- **Data Manupulation language(DML)**
- Data Query Language
- Transaction Control language(TCL)

- DDL (DATA DEFINITION LANGUAGE) COMMANDS

  1.Create

  2.Alter

  3.Drop

DDL OPERATIONS EXAMPLES:

1.To create an employee table with ename, eid, doj, basic pay, age, dept.

Create table employee (ename varchar2 (15), eid number (5), doj date, basicpay number (8,2),age number (3));

2. To include one more field address (varchar2 (15)).

Alter table employee add (address varchar2 (15));

3.Describe the table.

Desc employee

4. Modify the width of the address field.

Alter table employee modify (address varchar2 (20));

DML (DATA MANIPULATION LANGUAGE) COMMANDS.

DML OPERATIONS EXAMPLES:

1.Insert

2.Update

3.Delete

1. 1.Insert five records to the employee database.

Insert into employee (ename, eid, doj, basicpay, age, address) values („&ename‟, &eid, „&doj‟, &basicpay, &age, „&address‟);

2.Increment the basic pay of all the employees by 5% of their basic pay.

Update employee set basicpay = basicpay + basicpay

* 5 / 100;

3.Delete the third employee record from the table.

Delete from employee where eid=3;

TCL (TRANSACTION CONTROL LANGUAGE) COMMANDS.

Commit

Rollback

Savepoint

TCL OPERATIONS EXAMPLES:

In the student database make the transaction as permanent using Commit command. Select * from student;

Commit;

2 Undo the transaction performed in the database using Rollback. Rollback;

3 Select a point and implement Savepoint command which is later rollbacked to. Select * from student;

Savepoint sp1;

Delete from student where rno =

101; Rollback to spl;

VIEWS IN SQL

- A view in SQL is a single table that is derived from other tables.
- These other tables could be base tables or previously defined views A
- view does not exit in a physical form.
- It is considered as Virtual table.
- View is a way of specifying a table that we need to reference frequently.

For example:

We may frequently issue queries that retrieve the employee name and project names that the employee works on

Rather than having to specify the join of the employee, works on & project tables every time we issue that query, we can define a view that is a result of these joins. We then issue queries on the view.

Specification of Views in SQL:

The comman to specify view is: Syntax:

CREATE VIEW <View name> AS SELECT <Attribute list> FROM <Table list>
WHERE<condition>;

- The view is given a table name (view name), a list of attribute name, and a query to specify the contents of the view.

Example:

CREATE VIEW EMP_PROJ AS SELECT FNAME, LNAME, PNAME, HOURS
FROM EMPLOYEE, PROJECT, WORKS-ON WHERE SSN=ESSN
AND PNO=PNUMBER;

In this case EMP_PROJ inherits the names of the view attributes from the defining tables EMPLOYEE, PROJECT, WORKS-ONEMP_PROJ

FNAME	LNAME	PNAME	HOURS

Retrieve the first name of all employees who work on 'Product-X '
Q: SELECT FNAME, LNAME FROM EMP_PROJ WHERE PNAME='Product-X';

Advantage: It is simplify the specification of certain queries. It is also used as a security and authorization mechanism.
View is always *up to date*; if we modify the tuple in the base tables on which the view is defined, the view must automatically reflect these changes.
If we do not need a view any more we can use the DROP VIEW command to dispose of it.

DROP VIEW EMP_PROJ;

View Implementation and View Update:

- Updating of views is complicated.
- An update on a view defined on a single table without any aggregate functions can be mapped to an update on the underlying base table.
- Update the PNAME attribute of 'John Smith' from 'ProductX' to 'ProductY'.

UPDATE EMP_PROJ SET PNAME='ProductY' WHERE FNAME='John' AND LNAME='Smith' AND PNAME='ProductX';

- A view update is feasible when only one update on the base relations can accomplish the desired update effect on the view.Whenever an update on the view can be mapped to more than one tuple on the underlying base relations, we must have a certain procedure to choose the desired update.

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF MANAGEMENT

SBAA7035-Database Management System

UNIT 2

DATA MODELS : Data Models -Entity-Relationship Model, Network Data Model, Hierarchy Data Model, Relational Data Model, Semantic Data Model- Types of Database Systems - Centralized, Parallel, Client/Server, Distributed Database System.

Data models

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

ENTITY- RELATIONSHIP MODEL (E-R MODEL)

Entity relationship model is a high-level conceptual model, which is useful for end users.

An ER model describes data as

Entities
Attributes
Relationships

Entities:

An entity is defined as the real world object or thing that is described in the database. Examples: employee, student, department, project.

Attributes:

Attributes are the properties that describe an entity .For example an employee entity is described by the employee's name , age, address, salary.
A particular entity will have a value for each of its attributes. Types of attributes:

1. composite versus simple:
2. multi-valued versus single valued
3. Stored versus derived.

Composite attributes can be divided into smaller sub parts which represent more basic attributes with independent meanings.
Examples: address, name of an employee

Attributes that are not divisible are called ***simple or atomic attributes***. Ex: age, Gender

Multi-valued attributes have set of values for the same entity.Example: college degrees attribute for a person, phone numbers.

Single valued attributes have single value for a particular entity. Ex: age.

Derived attributes are derived from related entities (stored attribute). Ex: age attribute is derived from birth date attribute. Age attribute is a derived attribute. And birth date is a ***stored attribute.***

Complex attributes are combination of composite attributes and multi-valued attributes. For representing use ( ) for composite and { } for multi-valued.
Example: address
{Address (street address, city, state, pin code)}. Assume that a person can have more than one residence.

Key attributes:
An entity type usually has an attribute whose values are distinct for each individual entity in the collection. Such an attribute is called a key attribute and its values can be used to identify each entity uniquely.

For example:
Ssn of an employee entity, regno of a student entity, rollno of a student, dno of a department entity.

An entity types can have more than one key attribute . For student entity regno, rollno both are key attributes that uniquely identifies a student.

Weak entity:
Entity types that do not have key attributes of their own is called weak entity. Example:

Consider the entity type **dependent**, which is used to keep track of dependents of each employee. The attributes of dependent are name, birth date, sex and relationship. Two dependents of two distinct employees may by chance have the same values for name, birth date, sex, and relationship. Hence, it is difficult to identify a dependent. so weak entities are always related to specific entities called as **parent entity type** . Dependent entity is always related to employee entity.

Partial key:
A weak entity normally has a partial key, which is the set of attributes that can uniquely identify weak entities. In our example if we assume that no two dependents of the same employee ever have the same name the name attribute is the partial key.

Strong entity:
Entities that do have a key attribute is called strong entity. Example: employee, student, department, project.

Relationships:

Whenever an attribute of one entity type refers to another entity type, some relationship exists between entities.

Degree of relationship:

The degree of a relationship type is the number of participating entity types. In the works_for, relationship that associates the employee and department entity the degree of relationship is two. If the degree is, two it is called as binary relationship and one of degree three is called ternary.

Constraints on relationships:

Relationship types usually have certain constraints that limit the possible combinations of entities that may participate in the corresponding relationship set. There are two main types of relationship constraints:

1. Cardinality ratio:

The cardinality ratio for a binary relationship specifies the number of relationship instances that an entity can participate in.

Example: in the WORKS_FOR binary relationship type, department: employee is of cardinality ratio 1:N. (N stands for any number of related entities) means that each department can be related to numerous employees.

The possible cardinality ratios are 1: N, 1:1, M:N.

2. Participation.

There are two types.

1. **Total participation:**

   The participation of employee in WORKS_FOR is called total participation meaning that every entity in the total set of employees must be related to a department entity via WORKS_FOR relationship.

2. **Partial participation:**

   The participation of employee in manages relationship is called partial participation meaning that the company do not expect each and every employee must be related to department entity. Only some or part of the set of employees are related to department via manages relationship.

Attributes of relationships:

Relationships can have attributes: example: the WORKS_ON relationship, which relates employee and project, can have hours attribute to record the number of hours per week that an employee works on a particular project.

In our company database example, we specify the following relationship types:

1. MANAGES:

   A 1:1 relationship between employee and department. Employee participation is

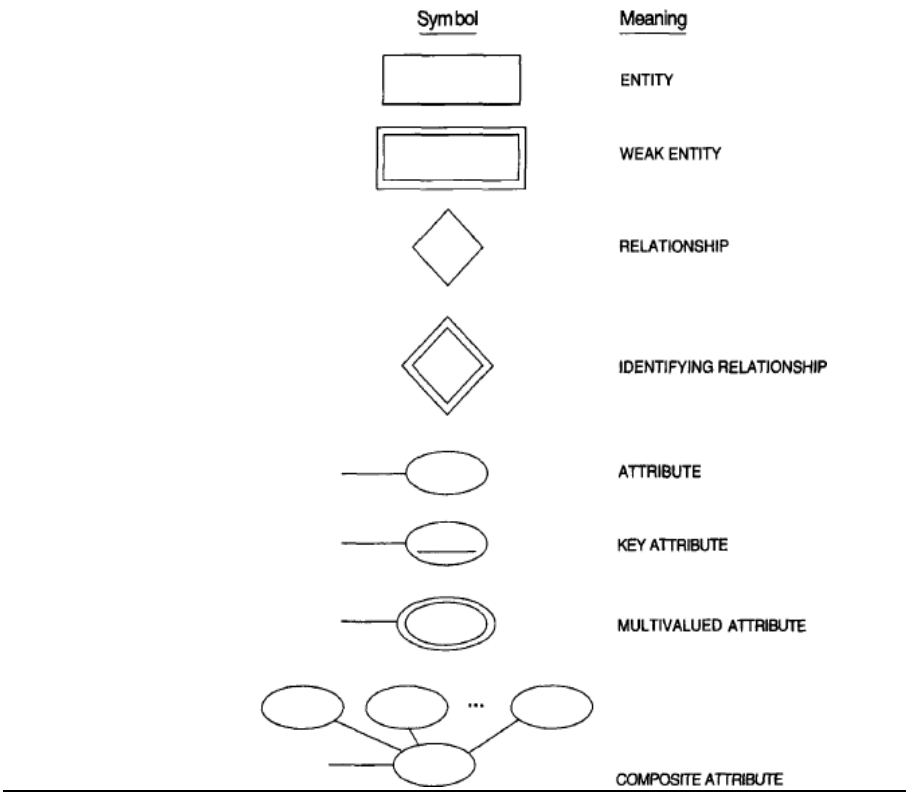
   partial.

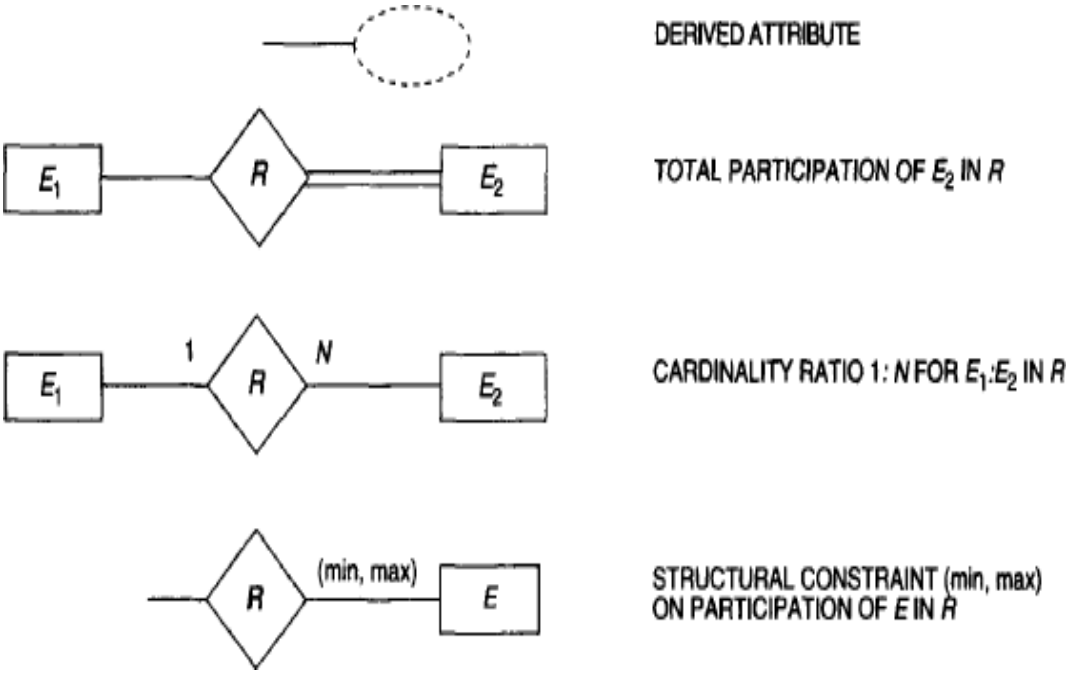
2. WORKS_FOR:
   A 1: N relationship between department and employee. Both participations are total

3. CONTROLS:
   A 1:N relationship between department and project.
4. WORKS_ON:
   A M:N relationship between employee and project. Employee participation is partial.
5. SUPERVISION:
   A 1:1 relationship between employee and employee. it is recursive relationship

6. DEPENDENTS_OF:
   A 1:1 relationship between employee and dependent.

NOTATIONS USED IN E-R DIAGRAM:





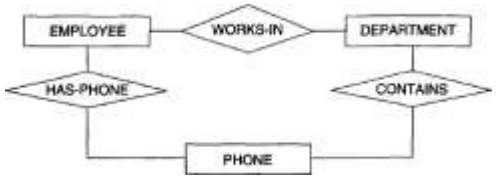


Figure 2.1 Part of a COMPANY database.

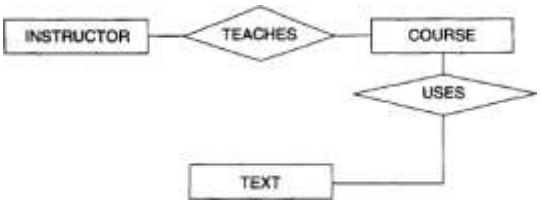


Figure 2.2 ER diagram for a COURSES database.

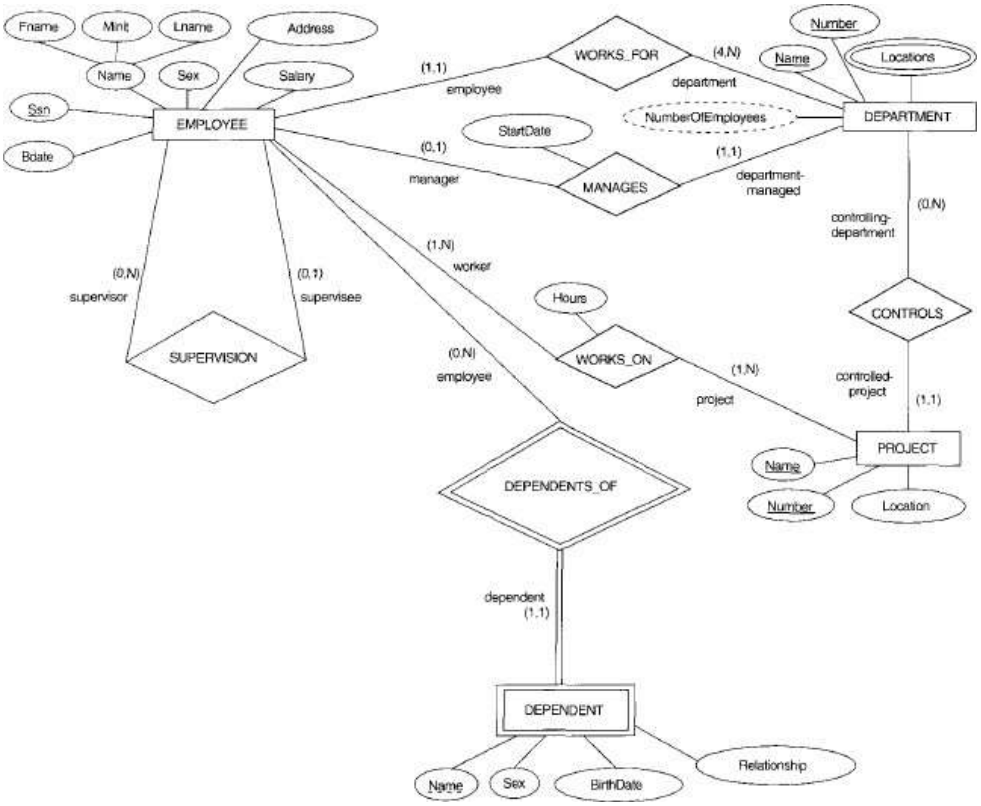


Figure 2.3 ER diagrams for the COMPANY schema

Hierarchical Model

This database model organises data into a tree-like-structure, with a single root, to which all the other data is linked. The heirarchy starts from the Root data, and expands like a tree, adding child nodes to the parent nodes.

In this model, a child node will only have a single parent node.

This model efficiently describes many real-world relationships like index of a book, recipes etc.

- Data model in which Data is represented in the tree-like structure.
- Data is stored in the form of records which are the collection of fields.
- The records are connected through links and the type of record tells which field is contained by the record.
- Each field can contain only one value.
- It must have only one parent for each child node but parent nodes can have more than one child.

- Multiple parents are not allowed.
- This is the major difference between the hierarchical and network Database model.
- The first node of the tree is called the root node.
- When Data needs to be retrieved then the whole tree is traversed starting from the root node.
- This model represents one- to- many relationships.

Advantages

- Data can be retrieved easily due to the explicit links present between the table structures.
- Referential integrity is always maintained i.e. any changes made in the parent table are automatically updated in a child table.
- Promotes data sharing.
- It is conceptually simple due to the parent-child relationship.
- Database security is enforced.
- Efficient with 1: N relationships.
- A clear chain of command or authority.
- Increases specialization.
- High performance.
- Clear results.

Disadvantages

- Data can be retrieved easily due to the explicit links present between the table structures.
- Referential integrity is always maintained i.e. any changes made in the parent table are automatically updated in a child table.
- Promotes data sharing.
- It is conceptually simple due to the parent-child relationship.
- Database security is enforced.
- Efficient with 1: N relationships.
- A clear chain of command or authority.
- Increases specialization.
- High performance.
- Clear results

- M: N relationship is not supported.
- No data manipulation or data definition language.

- Lack of standards.
- Poor flexibility
- Communication barriers
- Organizational Disunity.
- Rigid structure

Features

- Some features are pointed out below:
- **Many to many relationships:** It only supports one – to – many relationships. Many to many relationships are not supported.
- **Problem in Deletion:** If a parent is deleted then the child automatically gets deleted.
- **Hierarchy of data:** Data is represented in a hierarchical tree-like structure.
- **Parent-child relationship:** Each child can have only one parent but a parent can have more than one children.

Hierarchical model Example 1

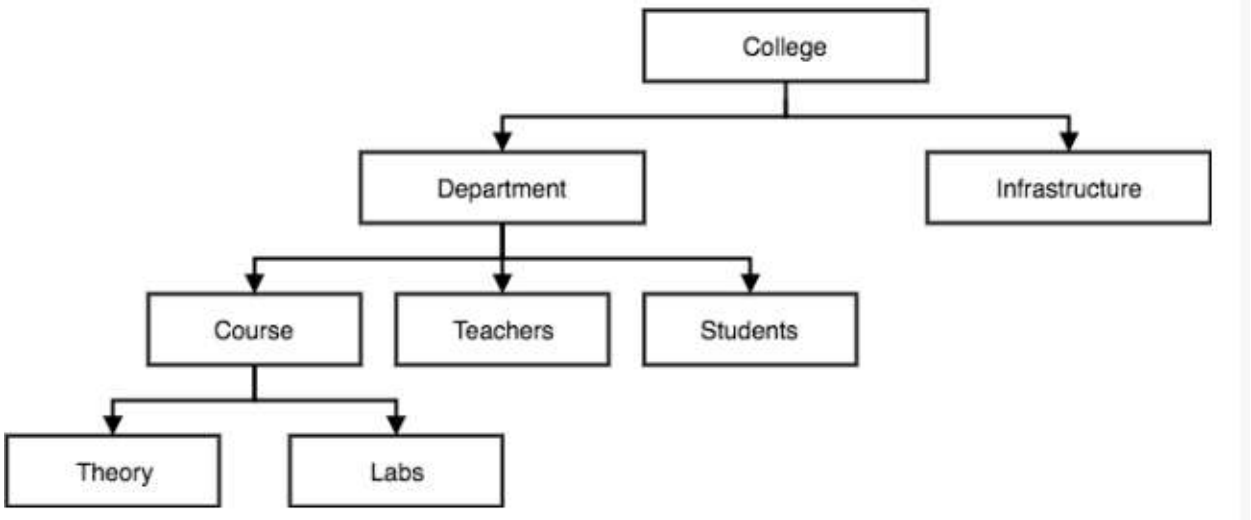


Figure 2.4 Hierarchical model Example 1

Network Model

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

Network Model of database

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

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

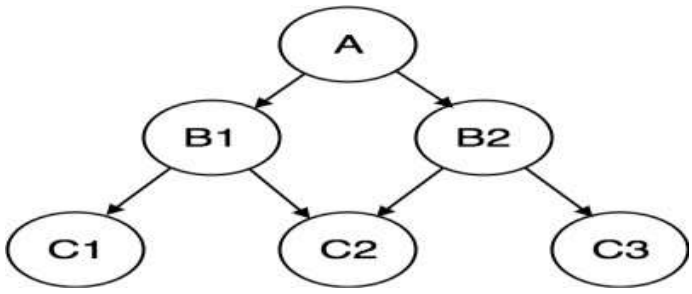


Figure 2.5 Network model Example

Advantages of Network Model

- The network model can support many to many relationships as seen in the diagram.
- D2 and C3 each have multiple masters.
- The masters for D2 are C1 and C2 while for C3 are B1 and B2.
- In this way, the network data model can handle many to many relationships where the hierarchical data model didn't.

Disadvantages of Network Model

- There are some disadvantages in the network model even though it is an improvement over the hierarchical model. These are −
- The network model is much more complicated than the Hierarchical model. As such, it is difficult to to handle and maintain.
- Although the Network model is more flexible than the Hierarchical model, it still has

flexibility problems. Not all relations can handled by assigning them in the form of owners and members.

- The structure of the Network Model is quite complicated and so the programmer has to understand it well in order to implement or modify it.

Semantic data model (SDM) is a high-level semantics-based database description and structuring formalism (database **model**) for databases. This database **model** is designed to capture more of the meaning of an application environment than is possible with contemporary database **models**.

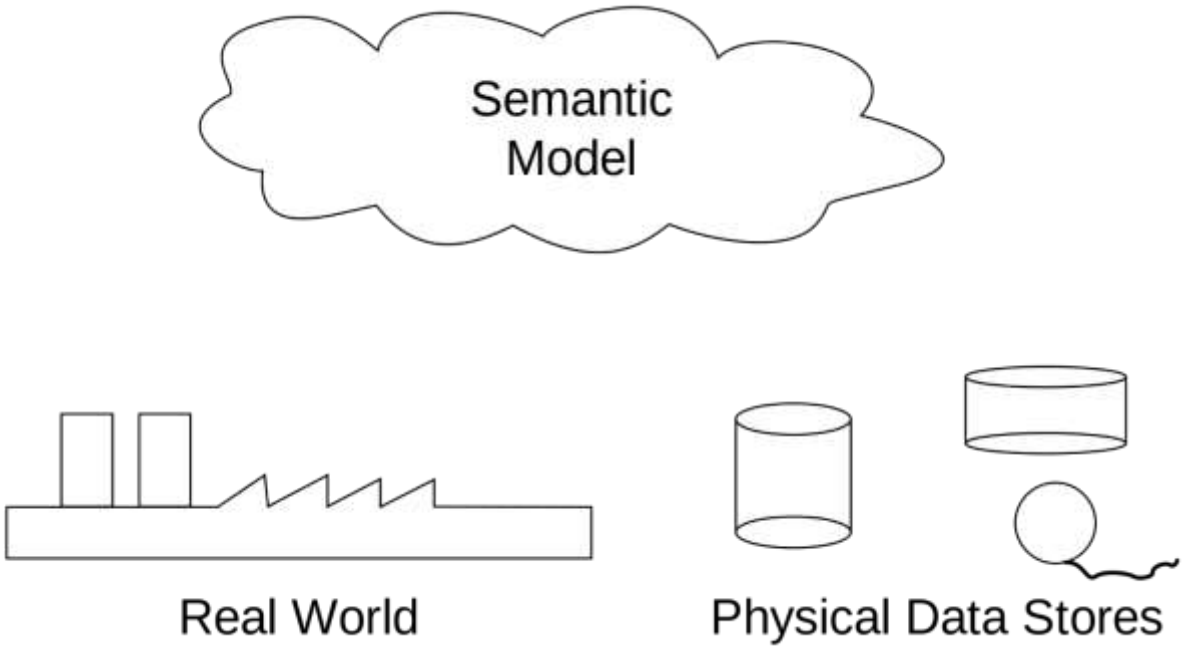


Figure 2.6 Semantic Data Model

Types of Database Systems - Centralized, Parallel, Client/Server, Distributed Database System.

The DBMS can be classified according to the number of users and the database site locations. These are:

On the basis of the number of users:

The database system may be multi-user or single-user. The configuration of the hardware and the size of the organization will determine whether it is a multi-user system or a single user system.

In single user system the database resides on one computer and is only accessed by one user at a time. This one user may design, maintain, and write database programs.

Due to large amount of data management most systems are multi-user. In this situation the data are both integrated and shared. A database is integrated when the same information is not recorded in two places. For example, both the Library department and the Account department of the college database may need student addresses. Even though both departments may access different portions of the database, the students' addresses should only reside in one place. It is the job of the DBA to make sure that the DBMS makes the correct addresses available from one central storage area.

- Centralized Database System
- Parallel Database System
- Distributed Database System
- Client-Server DBMS

Centralized Database System

The centralized database system consists of a single processor together with its associated data storage devices and other peripherals. It is physically confined to a single location. Data can be accessed from the multiple sites with the use of a computer network while the database is maintained at the central site.

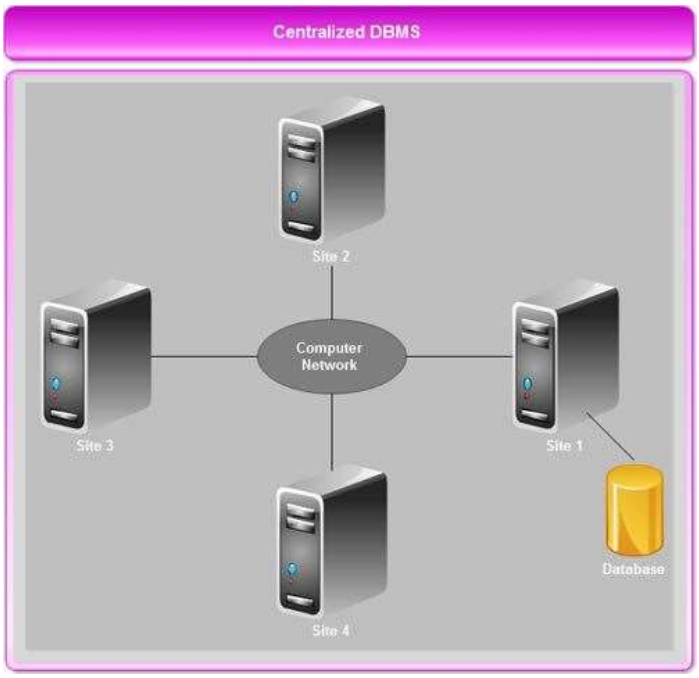


Figure 2.7 Centralized Database System

Disadvantages of Centralized Database System

• When the central site computer or database system goes down, then every one (users) is blocked from using the system until the system comes back.

• Communication costs from the terminals to the central site can be expensive.

Parallel Database System

Parallel database system architecture consists of a multiple Central Processing Units (CPUs) and data storage disk in parallel. Hence, they improve processing and Input/Output (I/O) speeds. Parallel database systems are used in the application that have to query extremely large databases or that have to process an extremely large number of transactions per second.

Advantages of a Parallel Database System

• Parallel database systems are very useful for the applications that have to query extremely large databases (of the order of terabytes, for example, 1012 bytes) or that have to process an extremely large number of transactions per second (of the order of thousands of transactions per second).

• In a parallel database system, the throughput (that is, the number of tasks that can be completed in a given time interval) and the response time (that is, the amount of time it takes to complete a single task from the time it is· submitted) are very high.

Disadvantages of a Parallel Database System

• In a parallel database system, there· is a startup cost associated with initiating a single process and the startup-time may overshadow the actual processing time, affecting speedup adversely.

• Since process executing in a parallel system often access shared resources, a slowdown may result from interference of each new process as it completes with existing processes for commonly held resources, such as shared data storage disks, system bus and so on.

Distributed Database System

A logically interrelated collection of shared data physically distributed over a computer network is called as distributed database and the software system that permits the management of the distributed database and makes the distribution transparent to users is called as Distributed DBMS.

It consists of a single logical database that is split into a number of fragments. Each fragment is stored on one or more computers under the control of a separate DBMS, with the computers connected by a communications network. As shown, in distributed database system, data is spread across a variety of different databases. These are managed by a variety of different DBMS software running on a variety of different operating systems. These machines are spread (or distributed) geographically and connected together by a variety of communication networks.

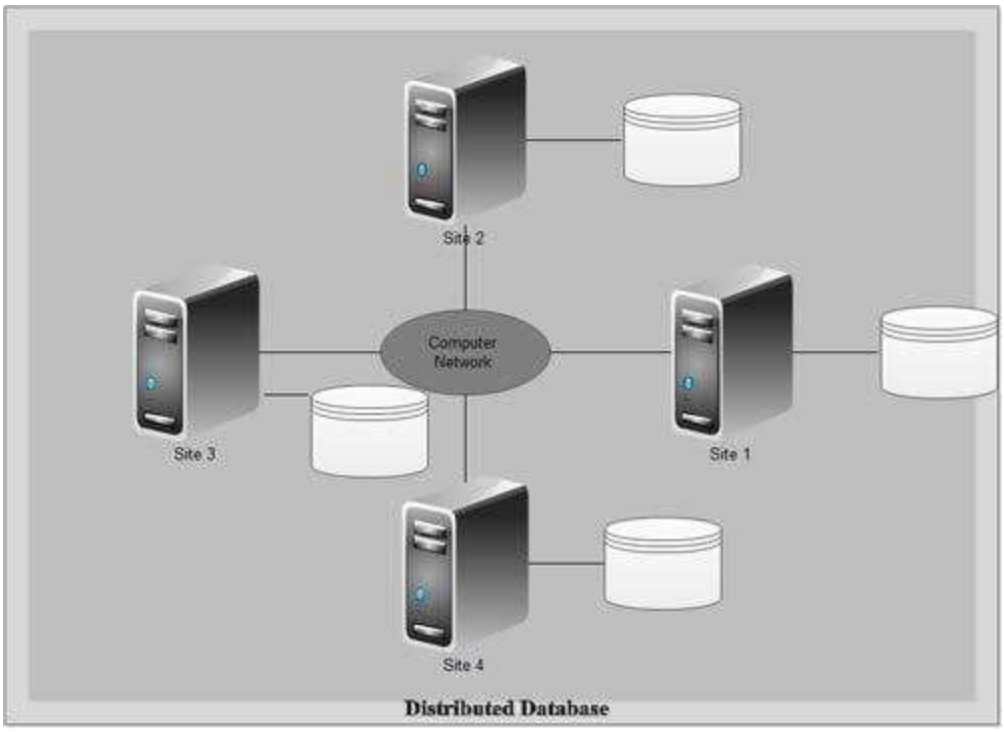


Figure 2.8 Distributed Database systems

Advantages of Distributed Database System

• Distributed database architecture provides greater efficiency and better performance.

• A single database (on server) can be shared across several distinct client (application) systems.

• As data volumes and transaction rates increase, users can grow the system incrementally.

• It causes less impact on ongoing operations when adding new locations.

• Distributed database system provides local autonomy.

Disadvantages of Distributed Database System

• Recovery from failure is more complex in distributed database systems than in centralized systems.

Client-Server DBMS

Client/Server architecture of database system has two logical components namely client, and server. Clients are generally personal computers or workstations whereas server is large workstations, mini range computer system or a mainframe computer system. The applications and tools of DBMS run on one or more client platforms, while the DBMS soft wares reside on the server. The server computer is caned back end and the client's computer is called front end. These server and client computers are connected into a network. The applications and tools act as clients of the DBMS, making requests for its services. The DBMS, in turn, processes these

requests and returns the results to the client(s). Client/Server architecture handles the Graphical User Interface (GUI) and does computations and other programming of interest to the end user. The server handles parts of the job that are common to many clients, for example, database access and updates.

Multi-Tier client server computing models

In a single-tier system the database is centralized, which means the DBMS Software and the data reside in one location and the dumb terminals were used to access the DBMS as shown.

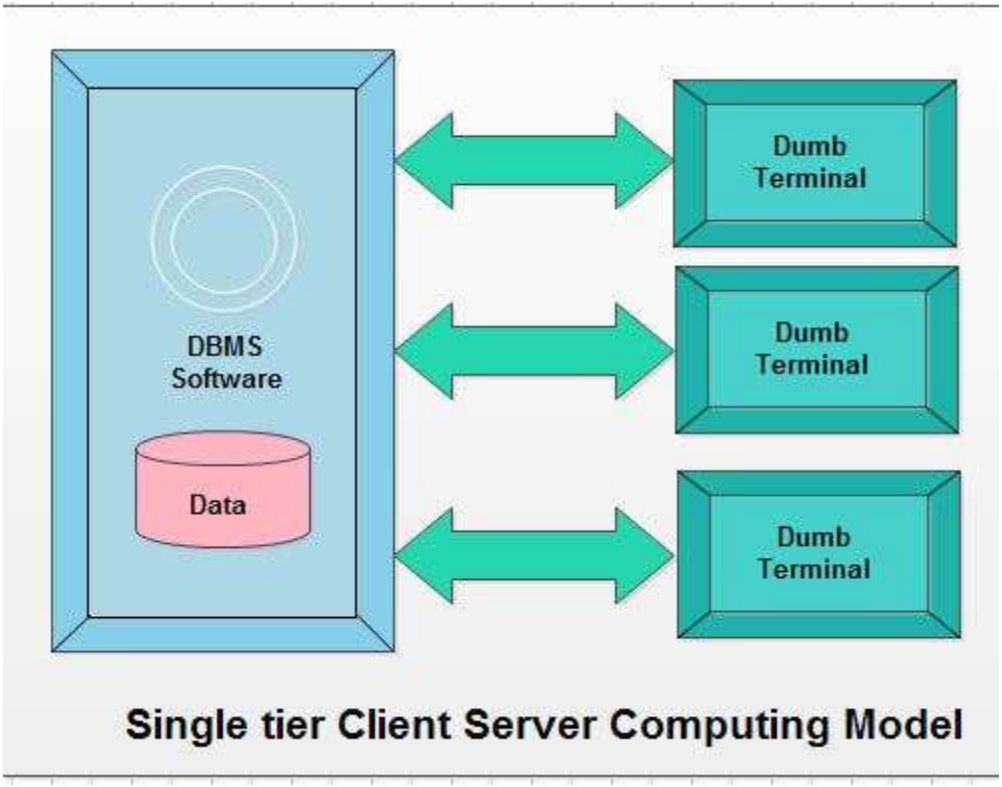


Figure 2.9 Client server computing model

The rise of personal computers in businesses during the 1980s, the increased reliability of networking hardware causes Two-tier and Three-tier systems became common. In a two-tier system, different software is required for the server and for the client. Illustrates the two-tier client server model. At the early stages client server computing model was called two-tier-computing model in which client is considered as data capture and validation tier and Server was considered as data storage tier. This scenario is depicted.

Problems of two-tier architecture

The need of enterprise scalability challenged this traditional two-tier client-server model. In the mid-1990s, as application became more complex and could be deployed to hundreds or thousands of end-users, the client side, now undergoes with following problems:

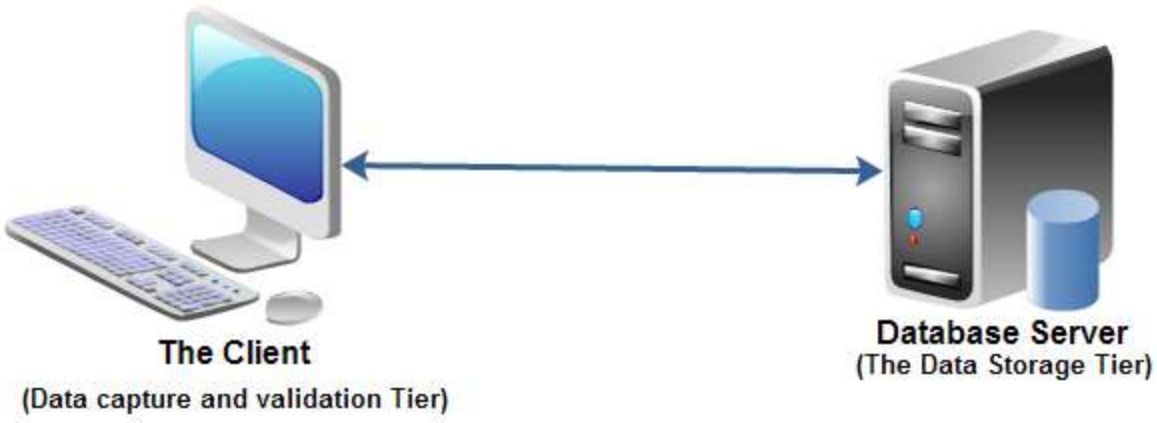


Figure 2.10 Client Server Architecture

A' fat' client requiring considerable resources on client's computer to run effectively. This includes disk space, RAM and CPU.

• Client machines require administration which results overhead.

Three-tier architecture

By 1995, three-tier architecture appears as improvement over two-tier architecture. It has three layers, which are:

• **First Layer:** User Interface which runs on end-user's computer (the client) .

• **Second Layer:** Application Server It is a business logic and data processing layer. This middle tier runs on a server which is called as Application Server.

• **Third Layer**: Database Server It is a DBMS, which stores the data required by the middle tier. This tier may run on a separate server called the database server.

As, described earlier, the client is now responsible for application's user interface, thus it requires less computational resources now clients are called as 'thin client' and it requires less maintenance.

Advantages of Client/Server Database System

• Client/Server system has less expensive platforms to support applications that had previously been running only on large and expensive mini or mainframe computers

• Client offer icon-based menu-driven interface, which is superior to the traditional command-line, dumb terminal interface typical of mini and mainframe computer systems.

• Client/Server environment facilitates in more productive work by the users and making better use of existing data.

Client/Server database system is more flexible as compared to the Centralized system.

• Response time and throughput is high.

• The server (database) machine can be custom-built (tailored) to the DBMS function and thus can provide a better DBMS performance.

• The client (application database) might be a personnel workstation, tailored to the needs of the end users and thus able to provide better interfaces, high availability, faster responses and overall improved ease of use to the user. + A single database (on server) can be shared across several distinct client (application) systems.

Disadvantages of Client/Server Database System

• Programming cost is high in client/server environments, particularly in initial phases.

• There is a lack of management tools for diagnosis, performance monitoring and tuning and security control, for the DBMS, client and operating systems and networking environments.

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF MANAGEMENT

SBAA7035-Database Management System

UNIT 3- NORMALIZATION

Functional Dependencies - Non-loss Decomposition -Normalization -First, Second, Third Normal Forms - Boyce/Codd
Normal Form - Multi-valued Dependencies and Fourth Normal Form - Join Dependencies and Fifth Normal Form.

Unit III

FUNCTIONAL DEPENDENCIES & NORMALIZATION FOR RELATIONAL DATABASES

FUNCTIONAL DEPENDENCIES:

- A functional dependency is a constraint between two sets of attributes from the database.
- A functional dependency denoted by x->y of set R between two sets of attributes x and y that are subsets of R specifies a constraint on the possible tuples that can form a relation state r (R).
- The constraint is: for any two tuples t1 and t2 in r that have,

- This means that the values of the values of the Y component of a tuple in r depend on, or are determined by, the values of the x component.(or)
- The value of the x component of a tuple functionally determine the values of the y component.
- Thus , x functionally determines Y is a relation schema R if and only if,whenever two tuples of r (R) agree on their x-value,they must necessarily agree on their y-value.

Example:

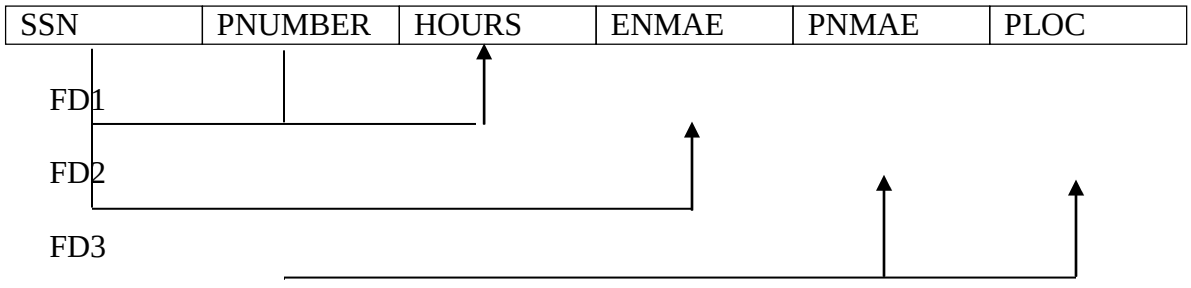


Figure 3.1 Functional dependency example

Functional dependencies in EMP-PROJ relation schema are:
FD2:SSN->ENAME
FD3:PNUMBER->{PNAME,PLOC}

FD1:{SSN,PNUMBER}-> HOURS.

FD1:

Specifies a combination of SSN & PNUMBER values uniquely determines the number of hours the employee works on the project per week.

FD2:

Specifies the value of an employees SSIV uniquely determines the employee name(ENAME).

FD3:

Specifies the value of a project number (PNUMBER)uniquely determines the project name(PNAME)and location.

INFERENCE RULES FOR FUNCTIONAL OF DEPENDENCIES:

- Set of functional dependencies specified on a relation schema is denoted by F.
- The closure F^+ of F is the set of all functional dependencies that can be inferred from F.

Example:

EMP-DEPT

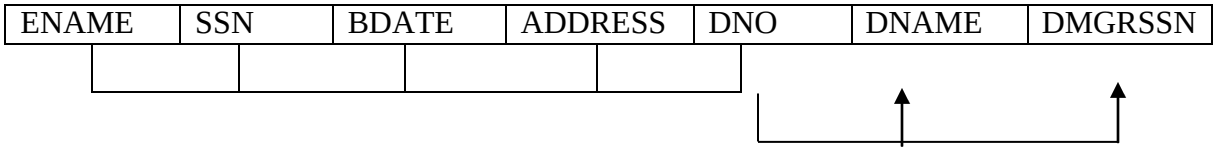


Figure 3.2 Inference rule example

Set of functional dependencies F

F= { SSN->{ENAME,BDATE,ADDRESS,DNO}
DNO->{DNAME,DMGRSSN} }

additional functional dependencies from F:(F^+)

SSN->{DNMAE,DMGRSSN}
SSN->SSN
DNO->DNAME.

INFERENCE RULES FOR FUNCTIONAL DEPENDENCIES:

- An FDX -> Y is inferred from a set of dependencies F specified on R ifX -> Y holds in *every* legal relation state r of R; that is, whenever r satisfies all the dependencies in F,X -> Y also holds in r.
- The closure $F+$ of F is the set of all functional dependencies that can be inferred from F.
- To determine a systematic way to infer dependencies, we must discover a set of inference rules that can be used to infer new dependencies from a given set of dependencies.
- We consider some of these inference rules next. We use the notation F $\models$ X-> Y to denote that the functional dependency X -> Y is inferred from the set of functional dependencies F.

NORMALIZATION:

- Normalization of data is a process of analyzing the given relation schema based on their functional dependencies and primary keys to achieve the desirable properties of

    1. Minimizing redundancy.
    2. Minimizing the insertion , deletion, and update anomalies.

- Normal form tests –are decomposed into smaller relation schemas that meet the tests and hence posses the desirable properties.
- A series of normal form tests that can be carried out on individual relation schemas. So that the relation database can be normalized to any desired degree.

ADDITIONAL PROPERTIES:

- The lossless join or non additive join property, which garantees that the spnrious tuple generation problem. It does not occure with respect to the relation schemas created after decomposition.
- The dependency preservation property, which ensures that each functional dependency is represented in some individual resulting after decomposition.

Denormalization:

It is the process of storing the join of higher normal form relations as a base relation –which in a lower normal form.

- An attribute of relation schema R is called a prime attribute if it is a member of some candidate key of R.

- An attribute is called nonprime if it is not a prime attribute- that is, if it is not a member of any candidate key.

Normal forms based on primary keys:

1.First Normal Form
2.Second Normal Form
3.Third Normal Form

First Normal Form:

- First normal form disallow multivalued attributes composite attributes , and their combinations.
- It states that the domain of an attribute must include only atomic values and that the value of any attribute in a tuple must be a single value from the domain of that attribute.
- First normal form disallow " relations within relations" or " relations as attributes of tuples"
- The only attribute values permitted by 1NF are single atomic values.

Examples:

Consider department table with a field:

DNUMBER	DNAME	LOCATION

Now the department relation schemas is not in 1NF because it contains an attribute locations. Which is multi-valued.

Table 3.1 Department Relation

Dnumber	**Dname**	**Location**
1	Research	Mumbai,banglore,hydera
2.	Operations	calcutta
3.	Marketing	Chennai,delhi

i. The above table can be brought into 1NF by dividing into three component attributes location1,location2 and location3,which makes the relation schema to look like this.

Table 3. 2 Modified Department Relation

Dnumber	**Dname**	**Location 1**	**Location 2**	**Location 3**
1	Research	Mumbai	Banglore	Hyderabad
2	Operations	Calcutta	-	-
3	Marketing	Chennai	Delhi	-

But it introduces null values in the department relation schema if the locations attribute has fewer than three values.

ii. Expand the key so that there is a separate tuple for each location in the department relation schema so that primary key becomes { dynamic locatins).

Table 3. 3 First normal form

Dnumber	**Dname**	**Location**
1	Research	Mumbai
1	Research	Banglore
1	Research	Hyderabad
2	Operations	Calcutta
3	Marketing	Chennai
3	Marketing	Delhi

iii. Another technique to bring the above relation to 1NF is remove the attribute locations that violate 1NF and keep it along with primary key dnumber is a separate key. {Dnumber,Location}

Dept Locations:

Table 3. 4 First normal form Department Relation

Dnumber	**Locations**
1	Mumbai
1	Banglore
1	Hyderabad
2	Calcutta
3	Chennai
3	Delhi

- 1NF also disallow composite attribute. i.e. nested relation i.e., each tuple has a relation within it.

Emp-pro:

Table 3.5 Emp-Proj Relation

E.NO	NAME	PROJECTS	
		Pno	**Hrs**
1	Arjun	1	24
		2	26
2	Basker	**3**	17
3	charles	4	13
		1	20

The schema is represented as EMP-PROJ(eno,name,(projects(pno-hrs}).

- Here Eno is the primary key.
- We can decompose EMP-PRO relation into EMP-PRO1 and EMP-PROJ2.

Table 3.6 Emp-Proj modified Relation

EMP-PROJ1

1	ARJUN
2	BASKER
3	CHARLES

EMP-PROJ2

ENO	PNO	HRS
1	1	24
1	2	26
2	3	12
3	4	13
3	1	20

EMP-PRO1,&EMP-PROJ2 are normalized relations.

SECOND NORMAL FORM:

- SNF is based on the concept of full functional dependency.
- A functional dependency x-> y is a full functional dependency if removal of any attribute a from x means that the dependency does not hold any more.
- ie.,for any attribute a €x,(x-{a}).Does not functionally determine y.
- a functional dependency x-> y is a partial dependency if some attribute a€x can be removed from x and the dependency still holds.

i.e for A € (x {A})→ y

EXAMPLE:
(I) {Ssn, Pnumber) → hours is a full functional dependency .

(II) {Ssn,Pnumber) →Ename is partial because Ssn→Ename holds.

The test for 2NF involves testing for functional dependencies whose left hand side attributes are part of the primary key.

Definition: A relation schema R is in 2NF if every non prime attributes A in R is fully functionally dependent on the primary key of R.

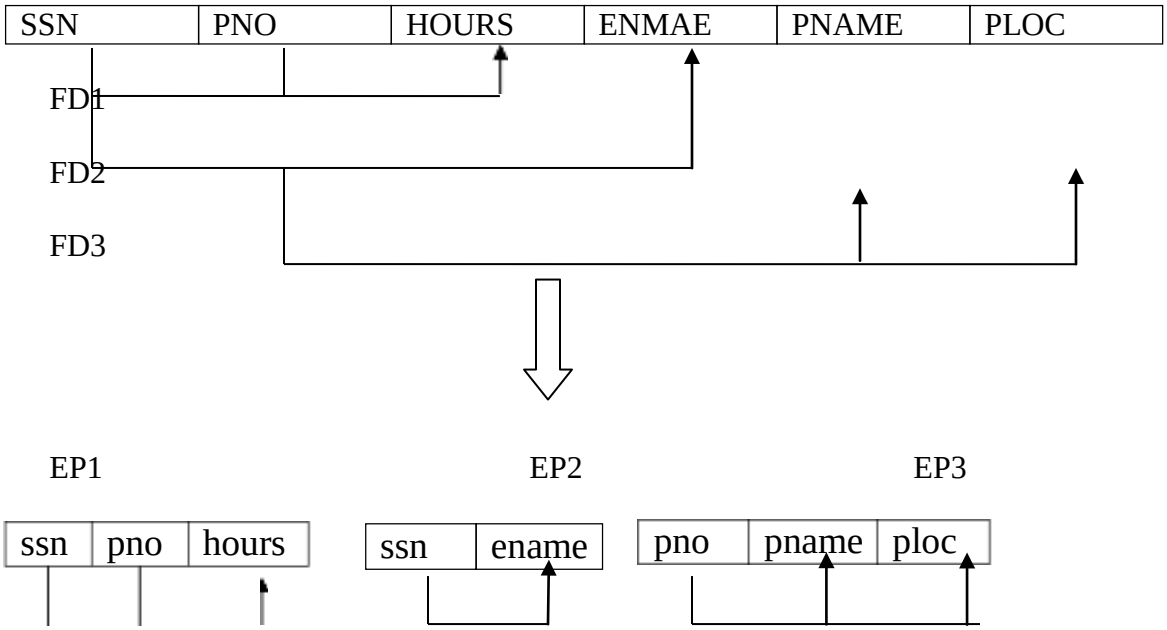


Figure 3.3 Second normal form

(I) {Ssn, Pnumber) → hours is a full functional dependency .

(II) {Ssn,Pnumber) →Ename is partial because Ssn→Ename holds

EP1,EP2, EP3 are full functional dependent.

THIRD NORMAL FORM :

3NF is based on the property on the concept of transitive dependency.

A functional dependency X→ Y in a relational schema R is in transitive dependency if there is a set of attributes Z that is neither a candidate key nor a subset of any key of R and both

X→ Z & Z→ Y hold.

ENMAE	SSN	BDATE	ADD	DNUM	DNAME	DMGRSSN

The dependency SSN→ DMGRSSN is transitive through DNUM is EMP,DEPT relation ,both the dependencies
SSn-→DMGRSSN-→DNUM

SSN→ DNUM is neither a key itself nor a subset of the key of EMP_DEPT.

- A relation schema R is in 3NF if it satisfies 2NF and no non prime attribute of R is transitive dependent on the primary key. It shouldn't transitively dependent on primary key.
- The relational schema EMP-DEPT ,is in 2NF, since no partial dependencies exit.
- However , EMP-dept is not in 3NF because of the transitive dependency of DMGRSSN on SSN via DNUM,
- So EMP-DEPT is normalized in to the two 3NF relation schemas ED1 and ED2.

A natursl join operation on ED1& ED2 will recover the original relation EMP-DEPT.

RELATION1
ENAME
SSN
BDATE
ADD

RELATION 2

DNUM
DNAME
DMGRSSN

Trivial − If a functional **dependency** (FD) X → Y holds, where Y is a subset of X, then it is called a **trivial** FD. **Trivial** FDs always hold. Non-**trivial** − If an FD X → Y holds, where Y is not a subset of X, then it is called a non-**trivial** FD.

BOYCE-CODD NORMAL FORM

- It is a simple form of 3NF,
- Every relation in BCNF is also in 3NF, a relation in 3NF is not necessarily in BCNF.
- The formal definition of BCNF is: A relation schema R is in BCNF if when ever a non trivial functional dependency X → A holds in R then X is a super key of R.
- The difference between the definition if BCNF and 3NF is that condition of 3NF,which allows A to be prime, is absent from BCNF.

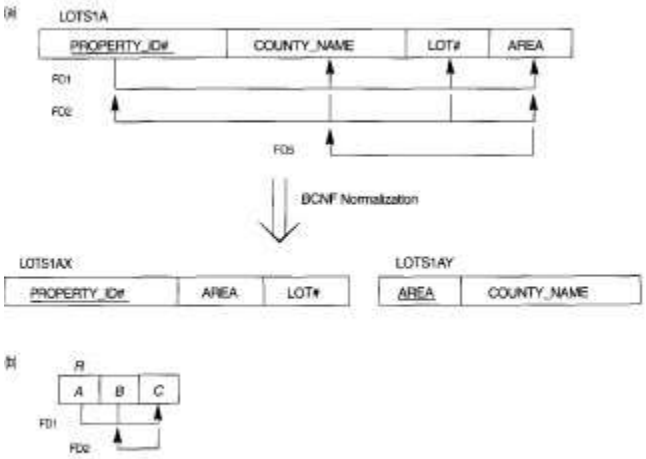


FIGURE 10.12 Boyce-Codd normal form. (a) BCNF normalization of LOTS1A with the functional dependency FD2 being lost in the decomposition. (b) A schematic relation with FDs; it is in 3NF, but not in BCNF.

Figure 3.4 Boyce-Codd Normal Form

- In example ,F.D5 violate BCNF in LOTSIA because AREA is not a super key of LOTSIA.
- FD5 satisfies 3NF in LOTSIA because COUNTRY-NAME is a prime attribute but this condition does not exist in the definition of BCNF
- We can decompose LOTSIA in to 2BCNF relations LOTISIAX & LOTSIAY
- This decomposition loses the functional dependency FD2 because attributes no longer coexists in the same relation.
- Most relational schemas that are in 3NF are also in BCNF.
- Only if X→A holds in a relation schema R with X not being a super key and A being a prime attribute will be in 3nf but not in BCNF.

A	B	C

A relation R is 3NF but not in BCNF.

Example:

Student	Course	instructor

Fd1:{ student,course}->instructor
Fd2:instructor->course

MULTIVALUED DEPENDENCIES:

- Multivalued dependencies are a consequences of 1nf, which disallowed an attribute in a attributes tuple to have a set of values,composite values.

- If we have two or more multivalued independent attributes in the same relation schema, we get into a problem of having to repeate every value of one of the

  attributes with every value of the other attributes to keep the relation state consistent and to maintain the independence among the attribute involved. This constraint is specified by a multivalued dependency.

Example:
EMP

ENAME	PNAME	DNAME
Smith	x	john
Smith	y	anna
Smith	x	john
Smith	y	anna

EMP: relation with two multivalued dependencies.
Ename->pname&ename->dname.

- A tuple in this EMP relation represents the fact that an employee whose name is ename .works on the project whose name is pname.and has a dependent whose name is dname.
- An employee may work on several projects and may have several dependents .
- The employee's projects and dependents are independent of one another.
- To keep the relation state consistent,we must have a separate tuple to representevery combinations of an employee's project.
- The constraint is specified as a multivalued dependency on the EMP relation.

FORMAL DEFINITION OF MULTIVALUED DEPENDENCY:

- A multivalued dependency (MUD)x->y specified on relation schema R, where x and y are both subsets of R,specifies the following constraints on any relation state r of R:
- If two tuples t1 and t2 exist in r such that t1[x]=t2[x] , then two tuples t3 and t4 should also exist in r with the following properties.
- T3[x]=t4[x]=t1[x]=t2[x]
- T3[y]=t1[y]and t4[y]=t2[y]
- T3[z]=t2[z] and t4[z]=t1[z]

Where z denote (R-(xυy))

- Whenever x->y holds we say that
- X multidetermines y whenever x-> y holds in R, so does x-

>z, hence,x->y implies x->z, and therefore it is sometimes written as. x->y/z.

Inference rules for functional and multivalued dependencies

- IR1(reflexive rule for FDs):x≥y then x->y.
- IR2(augmentation rule for FDs):{x->y)≠x2->y2.
- IR3(transitive rule for FDs):{x->y,y->z}≠x->z.
- IR4(complementation rule for MUDs):{x->y}≠{x->(R-(xυy))}.
- IR5(augmentation rule for MUDs): if x->y and w≥z then wx->yz.
- IR6(transitive rule for MUDs):{x->y,y->z}≠x->(z-y).

FOURTH NORMAL FORM:

- A relation schema R is in 4NF with respect to a set of dependencies F(that includes functional dependencies &multivalued dependencies)
- If, for every nontrivial multivalued dependency x-> y in F^+, x is a superkey of R.

Example:

a) **EMP**

Ename	Pname	dname
Smith	x	john
Smith	y	anna
Smith	x	anna
Smith	y	john

b) EMP_PROJECTS:

ENAME	PNAME
Smith	x
Smith	y

EMP_DEPENDENT

ENAME	DNAME
Smith	john
Smith	anna

- The EMP relation is not in 4NF because in the nontrivial MUDs ENAME->Pname and ENAME->DNAME,ENAME is not a superkey of EMP.
- EMP relation is decomposed into EMP_projects& emp_dependents.
- EMP_projects & EMP_dependents are in 4NF.
- Because the MUDs ENAME->PNAME in EMP_projects and ENAME->DNAME in EMP_DEPENTENTS are trivial MUDs.
- No other nontrivial MUDs hold in either EMP_PROJECTS or

EMP_DEPENDENTS.

LOSSLESS JOIN DECOMPOSITION INTO 4NF RELATIONSW:

1. If decomposition does not cause any loss of information it is called a **lossless** decomposition.
2. If a decomposition does not cause any dependencies to be lost it is called a **dependency-preserving** decomposition.
3. Any table scheme can be decomposed in a lossless way into a collection of smaller schemas that are in BCNF form. However the dependency preservation is not guaranteed.
4. Any table can be decomposed in a lossless way into 3^{rd} normal form that also preserves the dependencies.
    - 3NF may be better than BCNF in some cases

Whenever we decomposea relation schema R into R1=(xυy) and R2=(R-y)based on an mutivalued dependencies x->y that holds in R, the decomposition has the lossless join property.

PROPERTY:

The relation schema R1 and R2 form a lossless join decomposition of R if and only if (R1∩R2)->(R1-R2)

(Or)

By symmetry, if and only if (R1∩R2)->(R2-R1)

This property deals with both FDs and MUDs.

JOIN DEPENDENCIES & FIFTH NORMAL FORM:

- Decomposition has the lossless join property.
- In some cases there may be no lossless join decomposition of R into two relations schema.
- But there may be a lossless join decomposition into more than two relation schemas.
- There may be no functional dependency in R that violates any normal form upto BCNF and there may be no nontrivial MUD present in R either that violates 4NF.
- Join dependencies

  A join dependency denoted by JD(R1,R2…Rn) specified on relation schema R, specifies a constraint on the states r of R.
- Join dependency JD(R1,R2….Rn), specified on relation schema R, is a trivial JD if one of the relation schema R is in JD (R1,R2,….Rn) is equal to R.
- Such a dependency is called trivial because it has the lossless join property for any relation state r(R) and hence specify any constraint on R.

FIFTH NORMAL FORM : which is also called project-join normal form.

- **Fifth normal form is satisfied when all tables are broken into as many tables as possible in order to avoid redundancy. Once it is in fifth normal form it cannot be broken into smaller relations without changing the facts or the meaning.**

- A relation schema R is in 5NF with respect to a set F of functional, multivalued and join dependencies if, for every nontrivial join dependency JD(R1,R2,…..Rn) in F^+, every Ri is a superkey of R.

- The relation is in DKNF when there can be no insertion or deletion anomalies in the database.

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF MANAGEMENT

SBAA7035-Database Management System

UNIT 4 TRANSACTION MANAGEMENT AND DATBASE SECURITY

Introduction to Transaction Processing - Concurrency control techniques - Database Recovery Techniques - Database Security - Distributed databases and Client- Server Architecture.

UNIT 4

The concept of transaction provides a mechanism for describing logical units of database processing. **Transaction processing systems** are systems with large databases and hundreds of concurrent users that are executing database transactions. Examples of such systems include systems for reservations, banking, credit card processing, stock markets, supermarket checkout, and other similar systems.

A DBMS is **single-user** if at most one user at a time can use the system, and it is **multiuser** if many users can use the system—and hence access the database—concurrently.

Multiple users can access databases—and use computer systems—simultaneously because of the concept of **multiprogramming,** which allows the computer to execute multiple programs—or **processes**—at the same time.

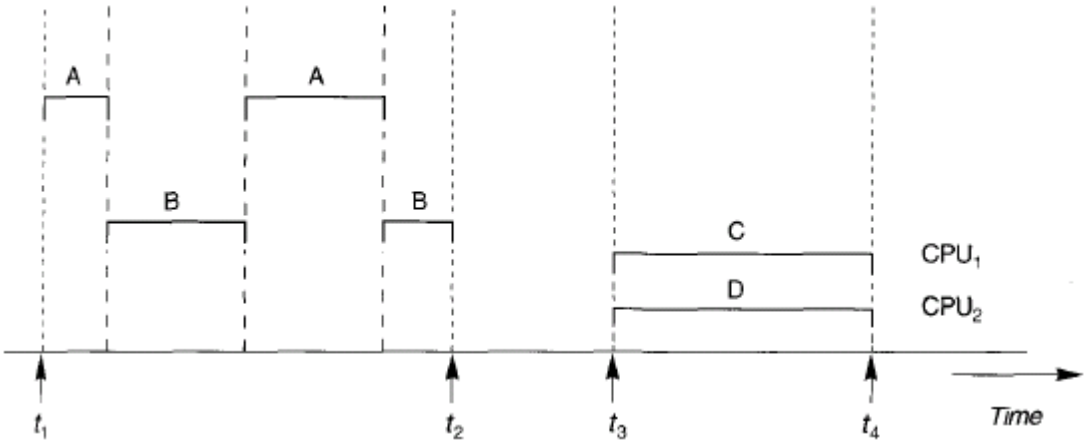


Figure 4.1 Multiprogramming

Parallel versus Interleaved Processing:

Multiprogramming operating systems execute some commands from one process, then suspend that process and execute some commands from the next process, and so on. A process is resumed at the point where it was suspended whenever it gets its turn to use the CPU again. Hence, concurrent execution of processes is actually **interleaved processing, as** illustrated by processes A and B in Figure.

If the computer system has multiple hardware processors (CPUs), **parallel processing** of multiple processes is possible, as illustrated by processes C and D in Figure.

Basic operations in any transaction:

The basic database access operations that a transaction can include are as follows:

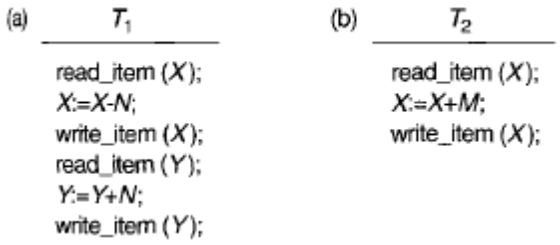
- **read_item(X):** Reads a database item named X into a program variable. To simplify our notation, we assume that the program variable is also named X.
- **write_item(X):** Writes the value of program variable X into the database item named X.

Executing a read_item(X) command includes the following steps:

1. Find the address of the disk block that contains item X.
2. Copy that disk block into a buffer in main memory (if that disk block is not already in some main memory buffer).
3. Copy item X from the buffer to the program variable named X.

Executing a write_item(X) command includes the following steps:

1. Find the address of the disk block that contains item X.
2. Copy that disk block into a buffer in main memory (if that disk block is not already in some main memory buffer).
3. Copy item X from the program variable named X into its correct location in the buffer.
4. Store the updated block from the buffer back to disk (either immediately or at some later point in time).



Two sample transactions T1 & T2

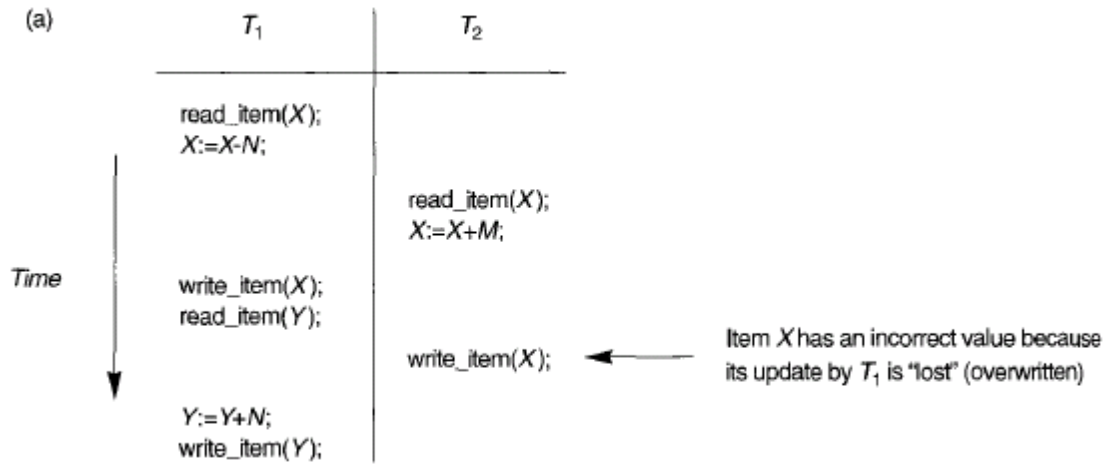
Why Concurrency Control Is Needed:

Several problems can occur when concurrent transactions execute in an uncontrolled manner. In Figure (a) and Figure (b), the transactions T1 and T2 are specific executions of the programs that refer to the specific flights whose numbers of seats are stored in data items X and Y in the database. We discuss the types of problems we may encounter with these two transactions if they run concurrently.

The Lost Update Problem

This problem occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database item incorrect.

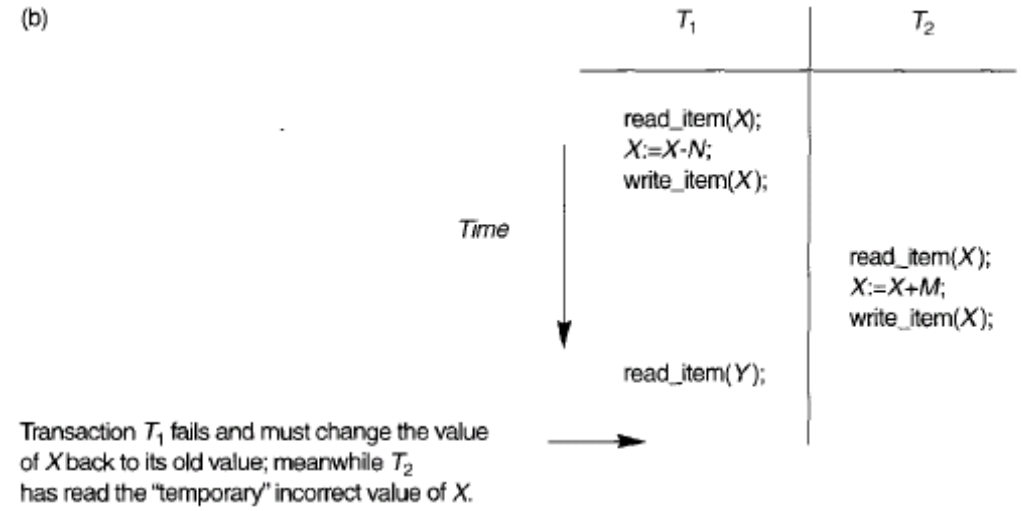
Suppose that transactions T1 and T2 are submitted at approximately the same time, and suppose that their operations are interleaved as shown in Figure (a); then the final value of item X is incorrect, because T2 reads the value of X before T1 changes it in the database, and hence the updated value resulting from T1 is lost.



For example, if X = 80 at the start (originally there were 80 reservations on the flight), N = 5 (transfers 5 seat reservations from the flight corresponding to X to the flight corresponding to Y), and M = 4 (reserves 4 seats on X), the final result should be X = 79; but in the interleaving of operations shown in Figure (a), it is X = 84 because the update in T1 that removed the five seats from X was lost.

The Temporary Update (or Dirty Read) Problem

This problem occurs when one transaction updates a database item and then the transaction fails for some reason.

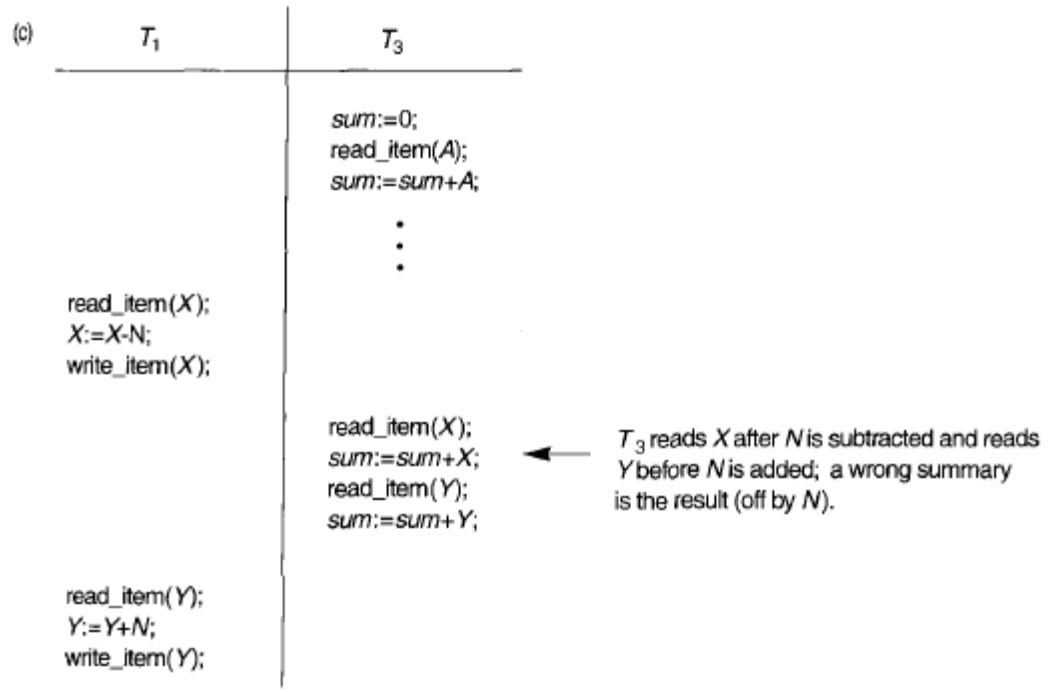


The updated item is accessed by another transaction before it is changed back to its original value. Figure (b) shows an example where T1 updates item X and then fails before

completion, so the system must change X back to its original value. Before it can do so, however, transaction T2 reads the "temporary" value of X, which will not be recorded permanently in the database because of the failure of T1. The value of item X that is read by T2 is called dirty data .Hence this problem is also known as the dirty read problem.

The Incorrect Summary Problem

If one transaction is calculating an aggregate summary function on a number of records while other transactions are updating some of these records, the aggregate function may calculate some values before they are updated and others after they are updated.



For example, suppose that a transaction T3 is calculating the total number of reservations on all the flights; meanwhile, transaction T1 is executing. If the interleaving of operations shown in Figure (c) occurs, the result of T3 will be off by an amount N because T3 reads the value of X after N seats have been subtracted from it but reads the value of Y before those N seats have been added to it.

Transaction States:

A transaction is an atomic unit of work that is either completed in its entirety or not done at all. For recovery purposes, the system needs to keep track of when the transaction starts, terminates, and commits or aborts. Hence, the recovery manager keeps track of the following operations:

• **BEGIN_TRANSACTION:** This marks the beginning of transaction execution.
• **READ or WRITE:** These specify read or write operations on the database items that are

executed as part of a transaction.
• **END_TRANSACTION:** This specifies that READ and WRITE transaction operations have ended and marks the end of transaction execution.
• **COMMIT_TRANSACTION:** This signals a successful end of the transaction so that any changes (updates) executed by the transaction can be safely committed to the database and will not be undone.
• **ROLLBACK (or ABORT):** This signals that the transaction has ended unsuccessfully, so that any changes or effects that the transaction may have applied to the database must be undone.

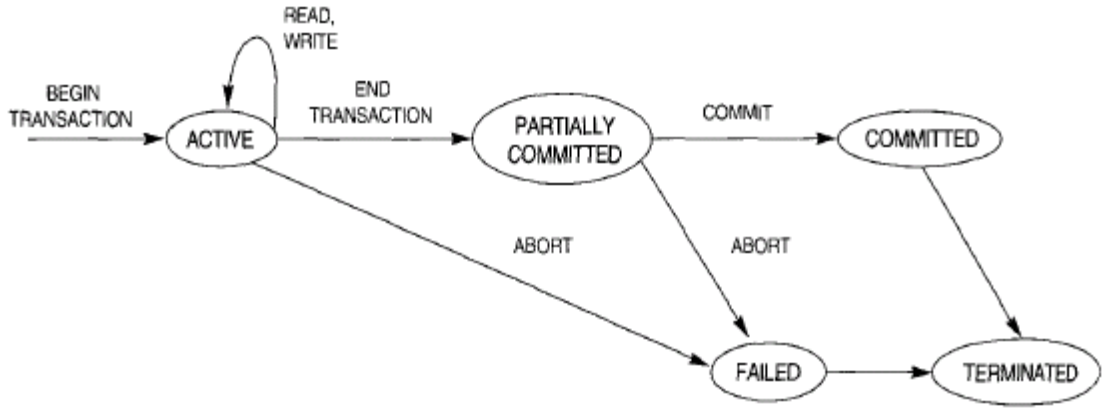


Figure 4.2 state transition

Figure shows a state transition diagram that describes how a transaction moves through its execution states. A transaction goes into an active state immediately after it starts execution, where it can issue READ and WRITE operations. When the transaction ends, it moves to the partially committed state. Once all checks are successful, the transaction is said to have reached its commit point and enters the committed state. Once a transaction is committed, it has concluded its execution successfully and all its changes must be recorded permanently in the database. However, a transaction can go to the **failed state** if one of the checks fails or if the transaction is aborted during its active state. The transaction may then have to be rolled back to undo the effect of its WRITE operations on the database. The **terminated state** corresponds to the transaction leaving the system.

Desirable Properties of Transactions:

Transactions should possess several properties. These are often called the ACID properties, and they should be enforced by the concurrency control and recovery methods of the DBMS. The following are the ACID properties:
1. **Atomicity:** A transaction is an atomic unit of processing; it is either performed in its entirety or not performed at all.
2. **Consistency preservation:** A transaction is consistency preserving if its complete execution take(s) the database from one consistent state to another.
3. **Isolation:** A transaction should appear as though it is being executed in isolation from other transactions. That is, the execution of a transaction should not be interfered with by any other transactions executing concurrently.
4. **Durability or permanency:** The changes applied to the database by a committed transaction

must persist in the database. These changes must not be lost because of any failure.

A **database state** is a collection of all the stored data items (values) in the database at a given point in time. A **consistent state** of the database satisfies the constraints specified in the schema as well as any other constraints that should hold on the database.

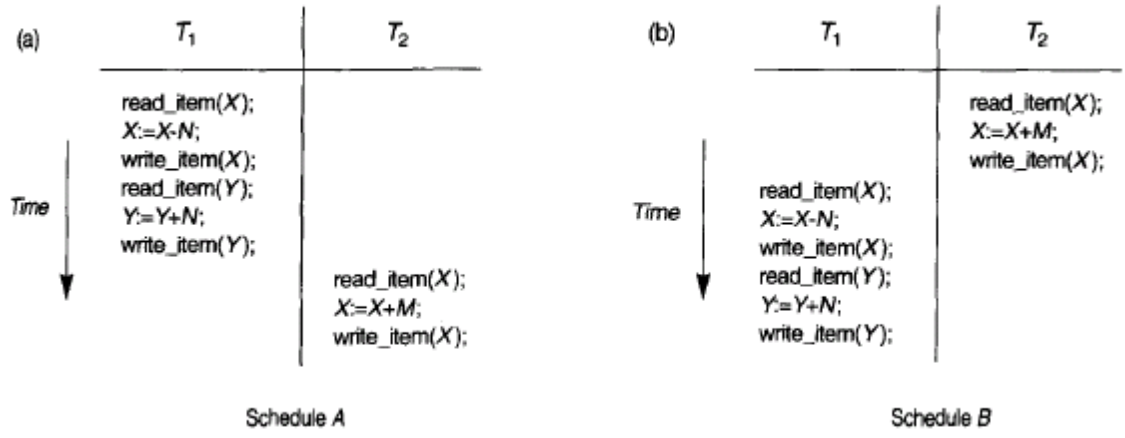
Serializability of Schedule :

We characterize the types of schedules that are considered correct when concurrent transactions are executing. If no interleaving of operations is permitted, there are only two possible outcomes:

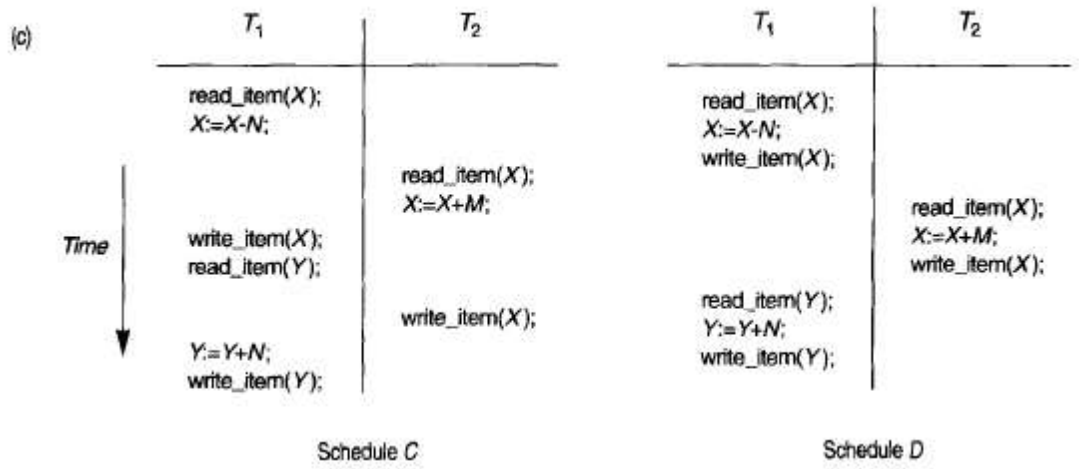
1. Execute all the operations of transaction T1 (in sequence) followed by all the operations of transaction T2 (in sequence).
2. Execute all the operations of transaction T2 (in sequence) followed by all the operations of transaction T1 (in sequence).

If interleaving of operations is allowed, there will be many possible orders in which the system can execute the individual operations of the transactions. The concept of serializability of schedules is used to identify which schedules are correct when transaction executions have interleaving of their operations in the schedules. This section defines serializability and discusses how it may be used in practice.

Serial, Non serial and Conflict-Serializable Schedules



Schedules A and B in Figure (a) and Figure (b) are called serial because the operations of each transaction are executed consecutively, without any interleaved operations from the other transaction. In a serial schedule, entire transactions are performed in serial order.



Schedules C and D in Figure (c) are called nonserial because each sequence interleaves operations from the two transactions.

Formally, a schedule S is **serial** if, for every transaction T participating in the schedule, all the operations of T are executed consecutively in the schedule; otherwise, the schedule is called **nonserial.**

The problem with serial schedules is that they limit concurrency or interleaving of operations. In a serial schedule, if a transaction waits for an I/O operation to complete, we cannot switch the CPU processor to another transaction, thus wasting valuable CPU processing time. In addition, if some transaction T is quite long, the other transactions must wait for T to complete all its operations before commencing. Hence, serial schedules are generally considered unacceptable in practice.

We would like to determine which of the nonserial schedules always give a correct result and which may give erroneous results. The concept used to characterize schedules in this manner is that of serializability of a schedule.A schedule S of n transactions is **serializable** if it is equivalent to some serial schedule of the same n transactions.

Two schedules are called **result equivalent** if they produce the same final state of the database. However, two different schedules may accidentally produce the same final state.

S_1	S_2
read_item(X);	read_item(X);
X:=X+10;	X:=X*1.1;
write_item(X);	write_item(X);

Two schedules that are result equivalent for the initial value of X= 100 but are not result equivalent in general.

For two schedules to be equivalent, the operations applied to each data item affected by the schedules should be applied to that item in both schedules in the same order. Two definitions of equivalence of schedules are generally used: **conflict equivalence and view equivalence.**

Conflict equivalence:

Two schedules are said to be **conflict equivalent** if the order of any two *conflicting operations* is the same in both schedules.

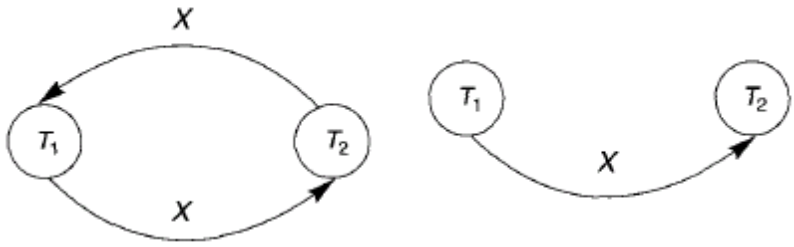
Two operations in a schedule are said to *conflict*

(1) if they belong to different transactions
(2) access the same database item, and
(3) at least one of the two operations is a write_item operation.

Testing for Conflict Serializability of a Schedule:

There is a simple algorithm for determining the conflict serializability of a schedule. Most concurrency control methods *do not* actually test for serializability. Rather protocols, or rules, are developed that guarantee that a schedule will be serializable.

The algorithm looks at only the read_item and write_item operations in a schedule to construct a **precedence graph** (or **serialization graph**), which is a **directed graph** *G* = (*N, E)* that consists of a set of nodes *N* = {T1,T2 , ...,Tn } and a set of directed edges.There is one node in the graph for each transaction Ti in the schedule.



Precedence graph which is not serializable Precedence graph which serializable

Algorithm : Testing conflict serializability of a schedule S.

1. For each transaction Ti participating in schedule S, create a node labeled Ti in the Precedence graph.
2. For each case in S where Tj executes a read_item(X) after *Ti* executes a write_item(X), create an edge (Ti → Tj) in the precedence graph.
3. For each case in S where Tj executes a write_item(X) after *Ti,* executes a read_item(X), create an edge (Ti → Tj) in the precedence graph.
4. For each case in S where Tj executes a write_item(X) after Ti executes a write_item(X), create an edge *(Ti* → Tj) in the precedence graph.
5. The schedule S is serializable if and only if the precedence graph has no cycles.

View Equivalence:

Two schedules Sand S' are said to be view equivalent if the following three conditions hold:

1. The same set of transactions participates in S and S', and S and S' include the same operations of those transactions.
2. For any operation Ri(X) of Ti, in S, if the value of X read by the operation has been

written by an operation Wj(X) of Tj (or if it is the original value of X before the schedule started), the same condition must hold for the value of X read by operation Ri(X) of Ti, in S'.

3. If the operation Wk(Y) of Tk is the last operation to write item Y in S, then Wk(Y) of Tk must also be the last operation to write item Y in S'.

The idea behind view equivalence is that, as long as each read operation of a transaction reads the result of the same write operation in both schedules, the write operations of each transaction must produce the same results.

Database Recovery

Database Recovery Techniques - Database Security – Debate on the distributed databases and Client- Server Architecture with reference to Indian Railway Reservation System.

a. Database Recovery

Purpose of Database Recovery

To bring the database into the last consistent state, which existed prior to the failure.

To preserve transaction properties (Atomicity, Consistency, Isolation and Durability).

Example:

If the system crashes before a fund transfer transaction completes its execution, then either one or both accounts may have incorrect value. Thus, the database must be restored to the state before the transaction modified any of the accounts

Types of Failure

The database may become unavailable for use due to

Transaction failure: Transactions may fail because of incorrect input, deadlock, incorrect

synchronization.

System failure: System may fail because of addressing error, application error, operating system fault, RAM failure, etc.

Media failure: Disk head crash, power disruption, etc.

Data Update

1.Immediate Update: As soon as a data item is modified in cache, the disk copy is updated.

2.Deferred Update: All modified data items in the cache is written either after a transaction ends its execution or after a fixed number of transactions have completed their execution.

3.Shadow update: The modified version of a data item does not overwrite its disk copy but is written at a separate disk location.

4.In-place update: The disk version of the data item is overwritten by the cache version.

5.**Data Caching**

Data items to be modified are first stored into database cache by the Cache Manager (CM) and after modification they are flushed (written) to the disk.

The flushing is controlled by **Modified** and **Pin-Unpin** bits.

Pin-Unpin: Instructs the operating system not to flush the data item.

Modified: Indicates the AFIM of the data item.'

6. **Transaction Roll-back (Undo)** and **Roll-Forward (Redo)**

To maintain atomicity, a transaction's operations are redone or undone.

Undo: Restore all BFIMs on to disk (Remove all AFIMs).

Redo: Restore all AFIMs on to disk.

Database recovery is achieved either by performing only Undos or only

Redos or by a combination of the two.

When **in-place** update (immediate or deferred) is used then log is necessary for recovery and it must be available to recovery manager. This is achieved by **Write-Ahead Logging (WAL)** protocol. WAL states that

- **For Undo**: Before a data item's AFIM is flushed to the database disk (overwriting the BFIM) its BFIM must be written to the log and the log must be saved on a stable store (log disk).

- **For Redo**: Before a transaction executes its commit operation, all its AFIMs must be written to the log and the log must be saved on a stable store.

7.Checkpointing

Time to time (randomly or under some criteria) the database flushes its buffer to database disk to minimize the task of recovery. The following steps defines a checkpoint operation:

8 **. Recovery Scheme**

Deferred Update (No Undo/Redo)

The data update goes as follows:

A set of transactions records their updates in the log.

At commit point under WAL scheme these updates are saved on database disk.

After reboot from a failure the log is used to redo all the transactions affected by this failure. No undo is required because no AFIM is flushed to the disk before a transaction commits.

Deferred Update with concurrent users

- Deferred Update in a single-user system
  There is no concurrent data sharing in a single user system. The data update goes as follows:
  - A set of transactions records their updates in the log.
  - At commit point under WAL scheme these updates are saved on database disk.
- After reboot from a failure the log is used to redo all the transactions affected by this failure. No undo is required because no AFIM is flushed to the disk before a transaction commits.

Recovery Techniques Based on Immediate Update

- **Undo/No-redo Algorithm**
  - In this algorithm AFIMs of a transaction are flushed to the database disk under WAL before it commits.
  - For this reason the recovery manager **undoes** all transactions during recovery.
  - No transaction is **redone**.
  - It is possible that a transaction might have completed execution and ready to commit but this transaction is also **undone**.

Recovery Techniques Based on Immediate Update

- **Undo/Redo Algorithm** (**Single-user** environment)
  - Recovery schemes of this category apply **undo** and also **redo** for recovery.
  - In a single-user environment no concurrency control is required but a log is maintained under WAL.
  - Note that at any time there will be one transaction in the system and it will be either in the commit table or in the active table.
  - The recovery manager performs:
    - **Undo** of a transaction if it is in the **active** table.
    - **Redo** of a transaction if it is in the **commit** table.

Deferred Update with concurrent users

Two tables are required for implementing this protocol:

Active table: All active transactions are entered in this table.

Commit table: Transactions to be committed are entered in this table.

During recovery, all transactions of the **commit** table is redone and all transactions of **active** tables are ignored since none of their AFIMs reached the database. It is possible that a **commit** table transaction may be **redone** twice but this does not create any inconsistency because of a redone is

"**idempotent**", that is, one redone for an AFIM is equivalent to multiple redone for the same AFIM.

a. Recovery Techniques Based on Immediate Update

Undo/No-redo Algorithm

In this algorithm AFIMs of a transaction are flushed to the database disk under WAL before it commits. For this reason the recovery manager **undoes** all transactions during recovery.No transaction is **redone**. It is possible that a transaction might have completed execution and ready to commit but this transaction is also **undone**.

Undo/Redo Algorithm (**Single-user** environment)

Recovery schemes of this category apply **undo** and also **redo** for recovery. In a singluser environment no concurrency control is required but a log is maintained under WAL.

d.Shadow Paging

The AFIM does not overwrite its BFIM but recorded at another place on the disk. Thus, at any time a data item has AFIM and BFIM (Shadow copy of the data item) at two different places on the disk.

X and Y: Shadow copies of data items

X' and Y': Current copies of data items

To manage access of data items by concurrent transactionstwo directories (current and shadow) re used.

The directory arrangement is illustrated below.

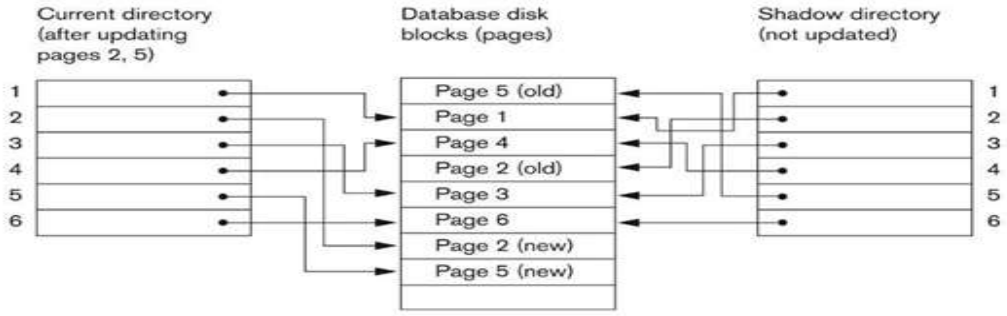


Figure 4.3 Shadow Paging

e. The ARIES Recovery Algorithm

The ARIES Recovery Algorithm is based on:

WAL (Write Ahead Logging)

Repeating history during redo:

ARIES will retrace all actions of the database system prior to the crash to reconstruct the database state when the crash occurred.

Logging changes during undo:

It will prevent ARIES from repeating the completed undo operations if a failure occurs during recovery, which causes a restart of the recovery process.

The ARIES recovery algorithm consists of three steps:

Analysis: step identifies the dirty (updated) pages in the buffer and the set of transactions active at the time of crash. The appropriate point in the log where redo is to start is also determined.

Redo: necessary redo operations are applied.

Undo: log is scanned backwards and the operations of transactions active at the time of crash are undone in reverse order.

The ARIES Recovery Algorithm (contd.)

- The **Transaction table** and the **Dirty Page table**

- For efficient recovery following tables are also stored in the log during checkpointing:
    - **Transaction table**: Contains an entry for each active transaction, with information such as transaction ID, transaction status and the LSN of the most recent log record for the transaction.
    - **Dirty Page table**: Contains an entry for each dirty page in the buffer, which includes the page ID and the LSN corresponding to the earliest update to that page.

- Checkpointing
    - A checkpointing does the following:
        - Writes a begin_checkpoint record in the log
        - Writes an end_checkpoint record in the log. With this record the contents of transaction table and dirty page table are appended to the end of the log.
        - Writes the LSN of the begin_checkpoint record to a special file. This special file is accessed during recovery to locate the last checkpoint information.
    - To reduce the cost of checkpointing and allow the system to continue to execute transactions, ARIES uses "fuzzy checkpointing".

The ARIES Recovery Algorithm (contd.)

- The following steps are performed for recovery
    - **Analysis phase**: Start at the begin_checkpoint record and proceed to the end_checkpoint record. Access transaction table and dirty page table are appended to the end of the log. Note that during this phase some other log records may be written to the log and transaction table may be modified. The analysis phase compiles the set of redo and undo to be performed and ends.
    - **Redo phase**: Starts from the point in the log up to where all dirty pages have been flushed, and move forward to the end of the log. Any change that appears in the dirty page table is redone.
    - **Undo phase**: Starts from the end of the log and proceeds backward while performing appropriate undo. For each undo it writes a compensating record in the log.
- The recovery completes at the end of undo phase.

Recovery in multidatabase system

A multi database system is a special distributed database system where one node may be running relational database system under UNIX, another may be running object-oriented system under Windows and so on.

A transaction may run in a distributed fashion at multiple nodes. In this execution scenario the transaction commits only when all these multiple nodes agree to commit individually the part of the transaction they were executing.

This commit scheme is referred to as "**two-phase commit**" (**2PC**). If any one of these nodes fails or cannot commit the part of the transaction,

then the transaction is aborted.Each node recovers the transaction under its own recovery protocol.

Data base Security

Why Db security?

- **DataBase:**
  - It is a collection of data ,generally stored and accessed
  - electronically from a computer system.
- **Security**
  - It is being free from danger.
- **Database Security**

It is the mechanisms that protect the database against intentional or accidental threats.

Major Security Vulnerabilities

- Bugs in database software components(eg-buffer overflow)
- Improper security configurations
- Use of default user accounts and passwords
- Use of null passwords
- Excessive privileges
- Lack of network isolation(external or internal)

Figure 4.4 Database Security

Types of Security

- Legal and ethical issues
- Policy issues
- System-related issues
- The need to identify multiple security levels

A DBMS typically includes a database security and authorization subsystem that is responsible for ensuring the security portions of a database against unauthorized access.

Two types of database security mechanisms:

- **Discretionary security mechanisms**
- **Mandatory security mechanisms**

DB security issues

- The security mechanism of a DBMS must include provisions for restricting access to the database as a whole; this function is called **access control** and is handled by creating user accounts and passwords to control login process by the DBMS.
- The security problem associated with databases is that of controlling the access to a **statistical database**, which is used to provide statistical information or summaries of values based on various criteria.
- The countermeasures to statistical database security problem is called **inference control** measures.
- Another security is that of **flow control**, which prevents information from flowing in such a way that it reaches unauthorized users.
- Channels that are pathways for information to flow implicitly in ways that violate the security policy of an organization are called **covert channels**.
- A final security issue is **data encryption**, which is used to protect sensitive data (such as credit card numbers) that is being transmitted via some type communication network.
- The data is **encoded** using some coding algorithm. An unauthorized user who access encoded data will have difficulty deciphering it, but authorized users are given decoding or decrypting algorithms (or keys) to decipher data.

Access Protection, User Accounts, and Database Audits

- Whenever a person or group of persons need to access a database system, the individual or group must first apply for a user account.
- The DBA will then create a new **account number** and **password** for the user if there is a legitimate need to access the database.
- The user must **log in** to the DBMS by entering account number and password whenever database access is needed.

Types of Discretionary Privileges

- The *account level*: At this level, the DBA specifies the particular privileges that each account holds independently of the relations in the database.
- The *relation (or table level*): At this level, the DBA can control the privilege to access each individual relation or view in the database.
- The privileges at the **account level**
    - CREATE SCHEMA or CREATE TABLE
    - CREATE VIEW privilege;
    - ALTER privilege,
    - DROP privilege,
    - SELECT privilege

Typical security classes

- Top secret (TS),
- Secret (S),
- Confidential (C),
- unclassified (U),

TS is the highest level and U the lowest

The commonly used model for multilevel security, known as the Bell-LaPadula model.

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF MANAGEMENT

SBAA7035-Database Management System

UNIT 5

DATABASE STORAGE AND IMPLEMENTATION TECHNIQUE 9 Hrs.

Overview of Physical Storage Media - File Organization - Organization of Records in Files - Indexing and Hashing -- Magnetic Disks - RAID Ordered Indices - B+ tree Index Files - B tree Index Files - Static Hashing - Dynamic Hashing.

Unit 5

Overview of Physical Storage Media

Several types of data storage exist in most computer systems. They vary in speed of access, cost per unit of data, and reliability.

- **Cache:** most costly and fastest form of storage. Usually very small, and managed by the operating system.
- **Main Memory (MM):** the storage area for data available to be operated on.
  - General-purpose machine instructions operate on main memory.
  - Contents of main memory are usually lost in a power failure or ``crash".
  - Usually too small (even with megabytes) and too expensive to store the entire database.
- **Flash memory:** EEPROM (*electrically erasable programmable read-only memory*).
  - Data in flash memory survive from power failure.
  - Reading data from flash memory takes about 10 nano-secs (roughly as fast as from main memory), and writing data into flash memory is more complicated: write-once takes about 4-10 microsecs.
  - To overwrite what has been written, one has to first erase the entire bank of the memory. It may support only a limited number of erase cycles (10^4 to 10^6).
  - It has found its popularity as a replacement for disks for storing small volumes of data (5-10 megabytes).
- **Magnetic-disk storage:** primary medium for long-term storage.
  - Typically the entire database is stored on disk.
  - Data must be moved from disk to main memory in order for the data to be operated on.
  - After operations are performed, data must be copied back to disk if any changes were made.
  - Disk storage is called **direct access** storage as it is possible to read data on the disk in any order (unlike sequential access).
  - Disk storage usually survives power failures and system crashes.
- **Optical storage:** CD-ROM (compact-disk read-only memory), WORM (*write-once read-many*) disk (for archival storage of data), and *Juke box* (containing a few drives and numerous disks loaded on demand).
- **Tape Storage:** used primarily for backup and archival data.
  - Cheaper, but much slower access, since tape must be read sequentially from the beginning.
  - Used as protection from disk failures!

- The storage device hierarchy is presented in Figure 5.1 where the higher levels are expensive (cost per bit), fast (access time), but the capacity is smaller.

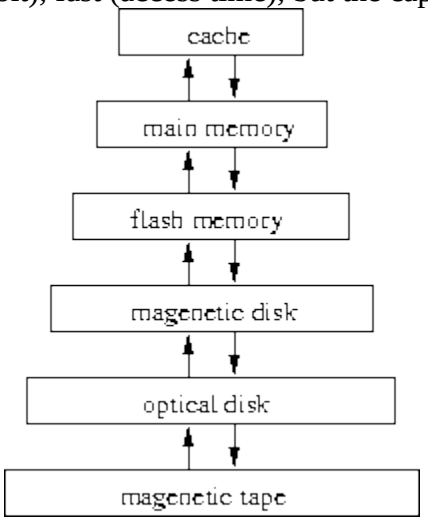


Figure 5.1 Storage-device hierarchy

 Another classification: Primary, secondary, and tertiary storage.

1. Primary storage: the fastest storage media, such as cash and main memory.
2. Secondary (or on-line) storage: the next level of the hierarchy, e.g., magnetic disks.
3. Tertiary (or off-line) storage: magnetic tapes and optical disk juke boxes.

 Volatility of storage. *Volatile storage* loses its contents when the power is removed. Without power backup, data in the volatile storage (the part of the hierarchy from main memory up) must be written to nonvolatile storage for safekeeping.

File organization

A database consist of a huge amount of data. The data is grouped within a table in RDBMS, and each table have related records. A user can see that the data is stored in form of tables, but in acutal this huge amount of data is stored in physical memory in form of files.

File – A file is named collection of related information that is recorded on secondary storage such as magnetic disks, magnetic tables and optical disks.

File Organization refers to the logical relationships among various records that constitute the file, particularly with respect to the means of identification and access to any specific record. In simple terms, Storing the files in certain order is called file Organization. **File Structure** refers to the format of the label and data blocks and of any logical control record.

Types of File Organizations –

Various methods have been introduced to Organize files. These particular methods have advantages and disadvantages on the basis of access or selection . Thus it is all upon the programmer to decide the best suited file Organization method according to his requirements. Some types of File Organizations are :

- Sequential File Organization
- Heap File Organization
- Hash File Organization
- B+ Tree File Organization
- Clustered File Organization

We will be discussing each of the file Organizations in further sets of this article along with differences and advantages/ disadvantages of each file Organization methods.

Sequential File Organization –

The easiest method for file Organization is Sequential method. In this method the file are stored one after another in a sequential manner. There are two ways to implement this method:

1.Insertion
Let the R1, R3 and so on upto R5 and R4 be four records in the sequence. Here, records are nothing but a row in any table. Suppose a new record R2 has to be inserted in the sequence, then it is simply placed at the end of the file.

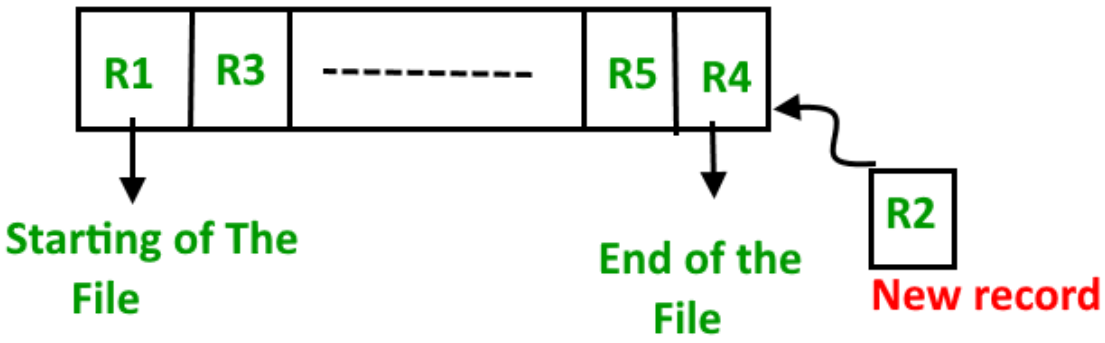


Figure 5.2 Record Insertion

1. **Sorted File Method –**In this method, As the name itself suggest whenever a new record has to be inserted, it is always inserted in a sorted (ascending or descending)

manner. Sorting of records may be based on any primary key or any other key.

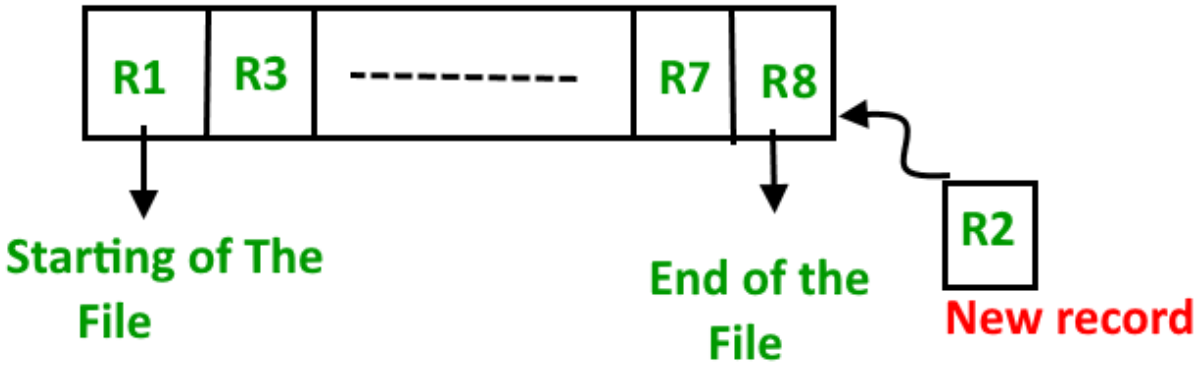


Figure 5.3 Record Insertion

Pros-

- Fast and efficient method for huge amount of data.
- Simple design.
- Files can be easily stored in magnetic tapes i.e cheaper storage mechanism.

Cons –

- Time wastage as we cannot jump on a particular record that is required, but we have to move in a sequential manner which takes our time.
- Sorted file method is inefficient as it takes time and space for sorting records.

Heap File Organization –

Heap File Organization works with data blocks. In this method records are inserted at the end of the file, into the data blocks. No Sorting or Ordering is required in this method. If a data block is full, the new record is stored in some other block, Here the other data block need not be the very next data block, but it can be any block in the memory. It is the responsibility of DBMS to store and manage the new records.

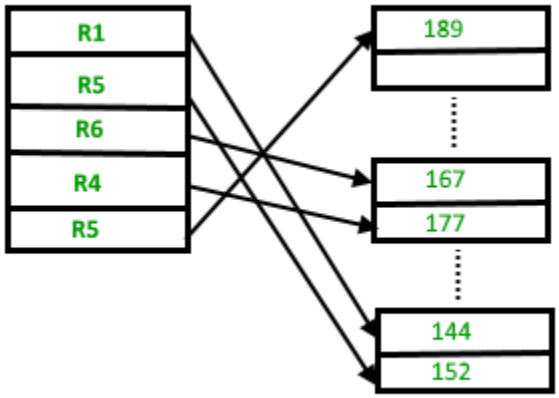


Figure 5.4 Heap file organization

Insertion of new record –
Suppose we have four records in the heap R1, R5, R6, R4 and R3 and suppose a new record R2 has to be inserted in the heap then, since the last data block i.e data block 3 is full it will be inserted in any of the data blocks selected by the DBMS, lets say data block 1.

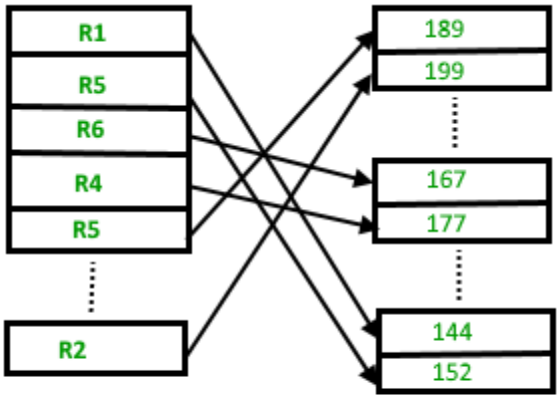


Figure 5.5 Inserting new record in heap file

If we want to search, delete or update data in heap file Organization the we will traverse the data from the beginning of the file till we get the requested record. Thus if the database is very huge, searching, deleting or updating the record will take a lot of time.

Pros –

- Fetching and retrieving records is faster than sequential record but only in case of small databases.
- When there is a huge number of data needs to be loaded into the database at a time, then this method of file Organization is best suited.

Cons –

- Problem of unused memory blocks.
- Inefficient for larger databases.

Indexing and Hashing

Indexing is a way of sorting a number of records on multiple fields. ... **Hashing** is used to **index** and retrieve items in a database because it is faster to find the item using the shorter **hashed** key than to find it using the original value

- Indexing mechanisms used to speed up access to desired data.
  - E.g., author catalog in library
- **Search Key** - attribute to set of attributes used to look up records in a file.

- An **index file** consists of records (called **index entries**) of the form
- Index files are typically much smaller than the original file
- Two basic kinds of indices:
    - **Ordered indices:** search keys are stored in sorted order

    - **What is a hash index? Basically, a hash index is an array of N buckets or slots, each one containing a pointer to a row. Hash indexes use a hash function F(K, N) in which given a key K and the number of buckets N , the function maps the key to the corresponding bucket of the hash index.**
    - **Hash indices:** search keys are distributed uniformly across "buckets" using a "hash function".

Index Evaluation Metrics

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

Secondary Indices

- Frequently, one wants to find all the records whose values in a certain field (which is not the search-key of the primary index) satisfy some condition.
    - Example 1: In the *instructor* relation stored sequentially by ID, we may want to find all instructors in a particular department
    - Example 2: as above, but where we want to find all instructors with a specified salary or with salary in a specified range of values
    - Cost of periodic re-organization
    - Relative frequency of insertions and deletions
    - Is it desirable to optimize average access time at the expense of worst-case access time?
    - Expected type of queries:
    - Hashing is generally better at retrieving records having a specified value of the key.
    - If range queries are common, ordered indices are to be preferred
    - Oracle supports static hash organization, but not hash indices
    - SQLServer supports only B^+-trees

B^+-tree indices are an alternative to indexed-sequential files.

- Disadvantage of indexed-sequential files
- performance degrades as file grows, since many overflow blocks get created.
- Periodic reorganization of entire file is required.
- Advantage of B^+-tree index files:
- automatically reorganizes itself with small, local, changes, in the face of insertions and deletions.
- Reorganization of entire file is not required to maintain performance.
- (Minor) disadvantage of B^+-trees:
- extra insertion and deletion overhead, space overhead.
- Advantages of B^+-trees outweigh disadvantages

Example of B^+-Tree

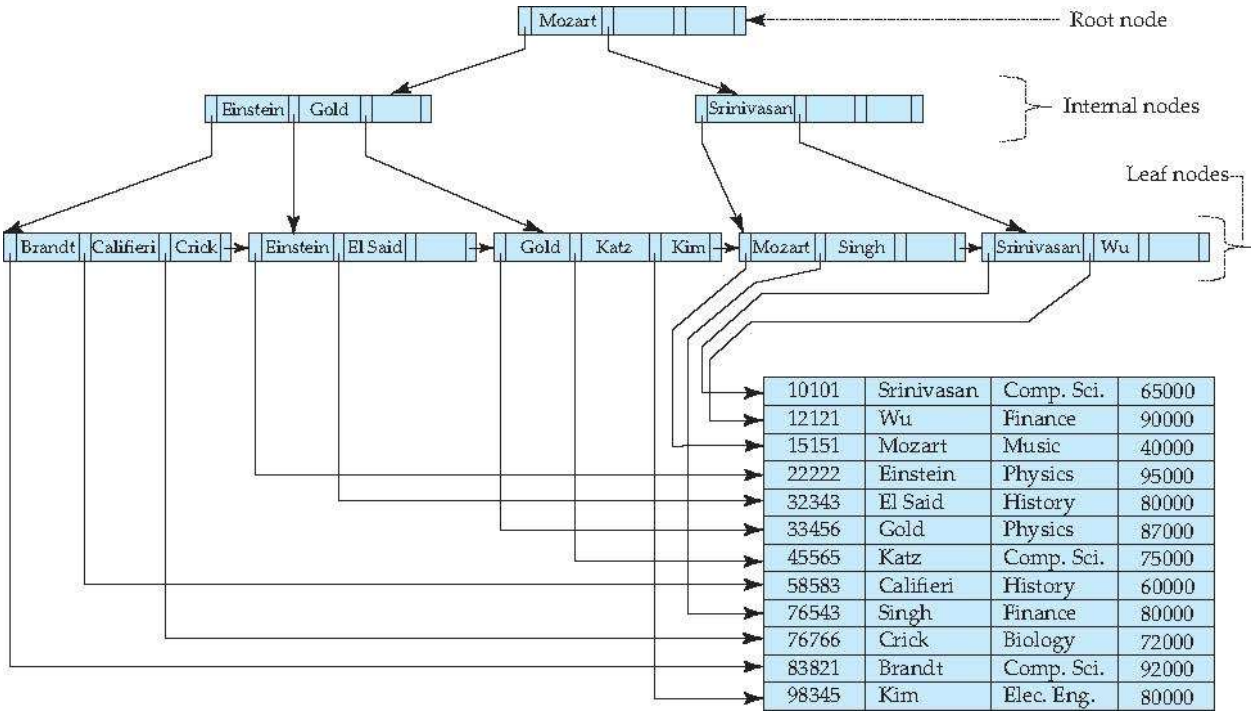


Figure 5.6 **B^+-Tree**

Observations about B^+-trees

- Since the inter-node connections are done by pointers, "logically" close blocks need not be "physically" close.
- The non-leaf levels of the B^+-tree form a hierarchy of sparse indices.
- The B^+-tree contains a relatively small number of levels
    - Level below root has at least $2*\lceil n/2 \rceil$ values
    - Next level has at least $2*\lceil n/2 \rceil * \lceil n/2 \rceil$ values
    - .. etc.
  - If there are K search-key values in the file, the tree height is no more than $\lceil \log_{\lceil n/2 \rceil}(K) \rceil$
  - thus searches can be conducted efficiently.
- Insertions and deletions to the main file can be handled efficiently, as the index can be restructured in logarithmic time (as we shall see).

Static Hashing

- A **bucket** is a unit of storage containing one or more records (a bucket is typically a disk block).
- In a **hash file organization** we obtain the bucket of a record directly from its search-key value using a **hash function.**
- Hash function h is a function from the set of all search-key values K to the set of all bucket addresses B.
- Hash function is used to locate records for access, insertion as well as deletion.
- Records with different search-key values may be mapped to the same bucket; thus entire bucket has to be searched sequentially to locate a record.

Deficiencies of Static Hashing

- In static hashing, function h maps search-key values to a fixed set of B of bucket addresses. Databases grow or shrink with time.
  - If initial number of buckets is too small, and file grows, performance will degrade due to too much overflows.
  - If space is allocated for anticipated growth, a significant amount of space will be wasted initially (and buckets will be underfull).
  - If database shrinks, again space will be wasted.
- One solution: periodic re-organization of the file with a new hash function
  - Expensive, disrupts normal operations
- Better solution: allow the number of buckets to be modified dynamically.

Dynamic Hashing

- Good for database that grows and shrinks in size
- Allows the hash function to be modified dynamically
- **Extendable hashing** – one form of dynamic hashing
    - Hash function generates values over a large range — typically *b*-bit integers, with $b = 32$.
    - At any time use only a prefix of the hash function to index into a table of bucket addresses.
    - Let the length of the prefix be *i* bits, $0 \le i \le 32$.
        - Bucket address table size = $2^{i.}$ Initially $i = 0$
        - Value of *i* grows and shrinks as the size of the database grows and shrinks.
    - Multiple entries in the bucket address table may point to a bucket (why?)
    - Thus, actual number of buckets is $< 2^i$
        - The number of buckets also changes dynamically due to coalescing and splitting of buckets.

RAID (redundant array of independent disks)

RAID (redundant array of independent disks) is a way of storing the same data in different places on multiple hard disks or solid-state drives to protect data in the case of a drive failure. There are different RAID levels, however, and not all have the goal of providing redundancy.

How RAID works

RAID works by placing data on multiple disks and allowing input/output (I/O) operations to overlap in a balanced way, improving performance. Because the use of multiple disks increases the mean time between failures (MTBF), storing data redundantly also increases fault tolerance.

RAID arrays appear to the operating system (OS) as a single logical drive. RAID employs the techniques of disk mirroring or disk striping. Mirroring will copy identical data onto more than one drive. Striping partitions helps spread data over multiple disk drives. Each drive's storage

space is divided into units ranging from a sector (512 bytes) up to several megabytes. The stripes of all the disks are interleaved and addressed in order.

RAID controller

A RAID controller is a device used to manage hard disk drives in a storage array. It can be used as a level of abstraction between the OS and the physical disks, presenting groups of disks as logical units. Using a RAID controller can improve performance and help protect data in case of a crash.

A RAID controller may be hardware- or software-based. In a hardware-based RAID product, a physical controller manages the array. The controller can also be designed to support drive formats such as SATA and SCSI. A physical RAID controller can also be built into a server's motherboard.

With software-based RAID, the controller uses the resources of the hardware system, such as the central processor and memory. While it performs the same functions as a hardware-based RAID controller, software-based RAID controllers may not enable as much of a performance boost and can affect the performance of other applications on the server.

If a software-based RAID implementation isn't compatible with a system's boot-up process, and hardware-based RAID controllers are too costly, firmware or driver-based RAID is another potential option.

Firmware-based RAID controller chips are located on the motherboard, and all operations are performed by the CPU, similar to software-based RAID. However, with firmware, the RAID system is only implemented at the beginning of the boot process. Once the OS has loaded, the controller driver takes over RAID functionality. A firmware RAID controller isn't as pricy as a hardware option, but it puts more strain on the computer's CPU. Firmware-based RAID is also called hardware-assisted software RAID, hybrid model RAID and fake RAID.

RAID levels

Raid devices will make use of different versions, called levels. The original paper that coined the term and developed the RAID setup concept defined six levels of RAID -- 0 through 5. This

numbered system enabled those in IT to differentiate RAID versions. The number of levels has since expanded and has been broken into three categories: standard, nested and nonstandard RAID levels.

Benefits of RAID

Benefits of RAID include the following.

- An improvement in cost-effectiveness because lower-priced disks are used in large numbers.
- The use of multiple hard drives enables RAID to improve on the performance of a single hard drive.
- Increased computer speed and reliability after a crash -- depending on the configuration.
- Reads and writes can be performed faster than with a single drive with RAID 0. This is because a file system is split up and distributed across drives that work together on the same file.
- There is increased availability and resiliency with RAID 5. With mirroring, RAID arrays can have two drives containing the same data, ensuring one will continue to work if the other fails.

Downsides of using RAID

RAID does have it downsides, however. Some of these include:

- Nested RAID levels are more expensive to implement than traditional RAID levels because they require a greater number of disks.
- The cost per gigabyte of storage devices is higher for nested RAID because many of the drives are used for redundancy.
- When a drive fails, the probability that another drive in the array will also soon fail rises, which would likely result in data loss. This is because all the drives in a RAID

array are installed at the same time, so all the drives are subject to the same amount of wear.

- Some RAID levels (such as RAID 1 and 5) can only sustain a single drive failure.
- RAID arrays, and the data in them, are in a vulnerable state until a failed drive is replaced and the new disk is populated with data.
- Because drives have much greater capacity now than when RAID was first implemented, it takes a lot longer to rebuild failed drives.
- If a disk failure occurs, there is a chance the remaining disks may contain bad sectors or unreadable data -- which may make it impossible to fully rebuild the array.

However, nested RAID levels address these problems by providing a greater degree of redundancy, significantly decreasing the chances of an array-level failure due to simultaneous disk failures.