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 COMPUTING

DEPARTMENT OF COMPUTER SCIENCE

UNIT – I DATABASE MANAGEMENT SYSTEM- SBS1205

SBS1205 DATABASE MANAGEMENT SYSTEM

Unit – I

Basic Concepts: Database and Database users – Database system concepts and architecture – Data modeling using Entity Relationship model – Enhanced entity relationship and object modeling – Record storage and file organizations – Index Structures for files.

Basic Concepts

DATA

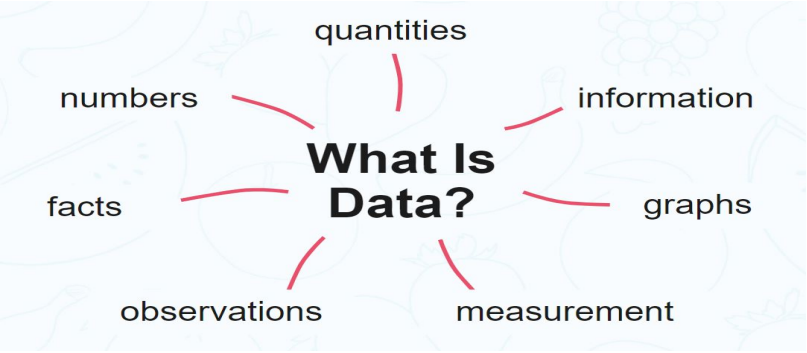


Figure 1.1 about Data

- Data are characteristics or information

  Data is a collection of facts, such as numbers, words, measurements, observations or just descriptions of things.Data can be qualitative or quantitative.Qualitative data is descriptive information(it describes something).Quantitative data is numerical information (numbers)

  QUANTITATIVE DATA

- Quantitative data can be Discrete or Continuous.Discrete data can only take certain values (like whole numbers)

  EX He has 4 legs, He has 2 brothers

- Continuous data can take any value (within a range)

  EX He weighs 60.5 kg, He is 545 mm tall

Discrete data is counted, Continuous data is measured.

QUALITATIVE DATA

- is descriptive information (it *describes* something)
- EX
- He is brown and black

- He has long hair
- He has lots of energy
- names in your city

DATABASE

A database is an organized collection of structured information. a structured set of data. Is a collection of information that is organized so that it can be easily accessed, managed and updated. Example Telephone Book

- A Database is a well organized collection of data. A database is integrated, structured, shared

Data means known facts or raw facts. 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.

Database Management System

- A Database Management System (DBMS) is software designed to store, retrieve, define, and manage data in a database.
- is system software for creating and managing databases.
- A set of programs to access the interrelated data.
- DBMS contains information about a particular enterprise.
- Computerized record keeping system
- Creation, insertion, Deletion, Updating & Retrieval of information.

Figure 1.2 DBMS

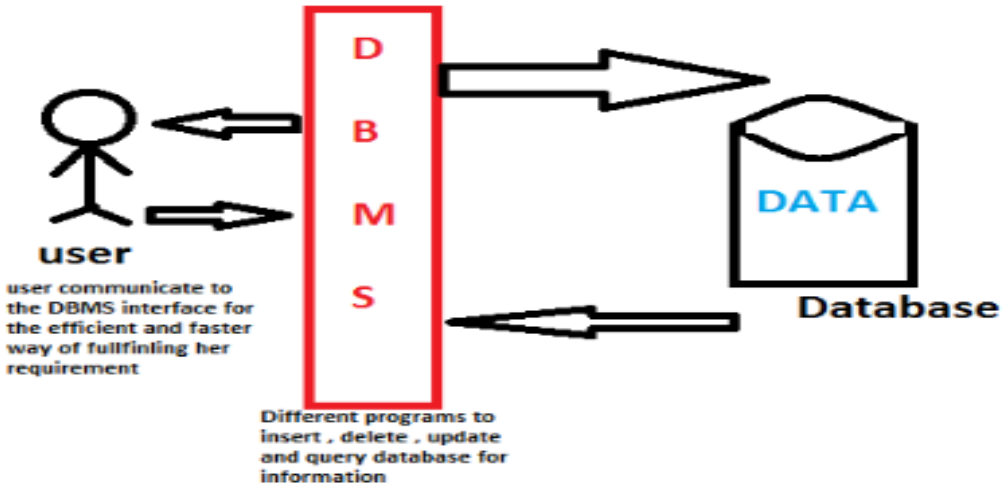


Figure 1.3 DBMS

DATABASE USERS

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

These are seven types of data base users in DBMS.

1. **Database Administrator (DBA) :**

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

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

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

DBA also monitors the recovery and back up and provide technical support.
The DBA has a DBA account in the DBMS which called a system or super user account.
DBA repairs damage caused due to hardware and/or software failures.

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

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

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

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

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

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

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

Types of DBA

- **Administrative DBA –** This DBA is mainly concerned with installing, and maintaining DBMS servers. His prime tasks are installing, backups, recovery, security, replications, memory management, configurations and tuning. He is mainly responsible for all administrative tasks of a database.
- **Development DBA –** He is responsible for creating queries and procedure for the requirement. Basically his task is similar to any database developer.
- **Database Architect –** Database architect is responsible for creating and maintaining the users, roles, access rights, tables, views, constraints and indexes. He is mainly responsible for designing the structure of the database depending on the requirement. These structures will be used by developers and development DBA to code.
- **Data Warehouse DBA –**DBA should be able to maintain the data and procedures from various sources in the datawarehouse. These sources can be files, COBOL, or any other programs. Here data and programs will be from different sources. A good DBA should be able to keep the performance and function levels from these sources at same pace to make the datawarehouse to work.
- **Application DBA –**He acts like a bridge between the application program and the database. He makes sure all the application program is optimized to interact with the database. He ensures all the activities from installing, upgrading, and patching, maintaining, backup, recovery to executing the records works without any issues.
- **OLAP DBA –** He is responsible for installing and maintaining the database in OLAP systems. He maintains only OLAP databases.

Advantage of Database Management System (DBMS):

Some of them are given as following below.

1. **Better Data Transferring:** Database management creates a place where users have an advantage of more and better managed data. Thus making it possible for end-users to have a quick look and to respond fast to any changes made in their environment.

2. **Better Data Security:**

   As number of users increases data transferring or data sharing rate also increases thus increasing the risk of data security. It is widely used in corporation world where companies invest money, time and effort in large amount to ensure data is secure and is used properly. A Database Management System (DBMS) provide a better platform for data privacy and security policies thus, helping companies to improve Data Security.

3. **Better data integration:**
   Due to Database Management System we have an access to well managed and synchronized form of data thus it makes data handling very easy and gives integrated view of how a particular organization is working and also helps to keep a track on how one segment of the company affects other segment.

4. **Minimized Data Inconsistency:**

   Data inconsistency occurs between files when different versions of the same data appear in different places.

   For Example, data inconsistency occurs when a student name is saved as “John Wayne” on a main computer of school but on teacher registered system same student name is “William J. Wayne”, or when the price of a product is $86.95 in local system of company and its National sales office system shows the same product price as $84.95.

   So if a database is properly designed then Data inconsistency can be greatly reduced hence minimizing data inconsistency.

5. **Faster data Access:**
   The Data base management system (DBMS) helps to produce quick answers to database queries thus making data accessing faster and more accurate. For example, to read or update the data. For example, end users, when dealing with large amounts of sale data, will have enhanced access to the data, enabling faster sales cycle.Some queries may be like:
   - What is the increase of the sale in last three months?
   - What is the bonus given to each of the salespeople in last five months?
   - How many customers have credit score of 850 or more?

6. **Better decision making:**

   Due to DBMS now we have Better managed data and Improved data accessing because of which we can generate better quality information hence on this basis better decisions can be made. Better Data quality

improves accuracy, validity and time it takes to read data.DBMS does not guarantee data quality, it provides a framework to make it is easy to improve data quality.

7. **Increased end-user productivity:** The data which is available with the help of combination of tools which transform data into useful information, helps end user to make quick, informative and better decisions that can make difference between success and failure in the global economy.

8. **Simple:**
Data base management system (DBMS) gives simple and clear logical view of data. Many operations like insertion, deletion or creation of file or data are easy to implement.

Three schema Architecture

- The three schema architecture is also called ANSI/SPARC architecture or three-level architecture.
- This framework is used to describe the structure of a specific database system.
- The three schema architecture is also used to separate the user applications and physical database.
- The three schema architecture contains three-levels. It breaks the database down into three different categories.

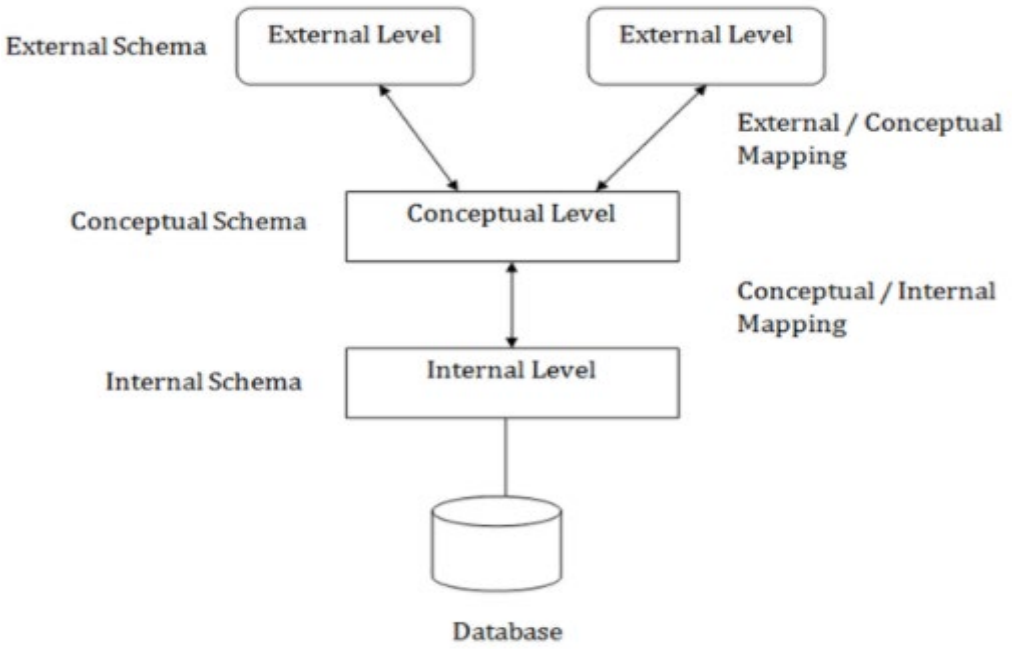


Figure 1.4 DBMS Architecture

In the above diagram:

- It shows the DBMS architecture.
- Mapping is used to transform the request and response between various database levels of architecture.

- Mapping is not good for small DBMS because it takes more time.
- In External / Conceptual mapping, it is necessary to transform the request from external level to conceptual schema.
- In Conceptual / Internal mapping, DBMS transform the request from the conceptual to internal level.

1. Internal Level
   - The internal level has an internal schema which describes the physical storage structure of the database.
   - The internal schema is also known as a physical schema.
   - It uses the physical data model. It is used to define that how the data will be stored in a block.
   - The physical level is used to describe complex low-level data structures in detail.
2. Conceptual Level
   - The conceptual schema describes the design of a database at the conceptual level. Conceptual level is also known as logical level.
   - The conceptual schema describes the structure of the whole database.
   - The conceptual level describes what data are to be stored in the database and also describes what relationship exists among those data.
   - In the conceptual level, internal details such as an implementation of the data structure are hidden.
   - Programmers and database administrators work at this level.
3. External Level
   - At the external level, a database contains several schemas that sometimes called as subschema. The subschema is used to describe the different view of the database.
   - An external schema is also known as view schema.
   - Each view schema describes the database part that a particular user group is interested and hides the remaining database from that user group.
   - The view schema describes the end user interaction with database systems.

Data model

- Data models define how the logical structure of a database is modeled.

- Data models define how data is connected to each other and how they are processed and stored inside the system.

- A Database model defines the logical design and structure of a database and defines how data will be stored, accessed and updated in a database management system
- A Data Model in Database Management System (DBMS), is the concept of tools that are developed to summarize the description of the database.
- A Data Model in Database Management System (DBMS), is the concept of tools that are developed to summarize the description of the database.
- It is classified into 3 types:

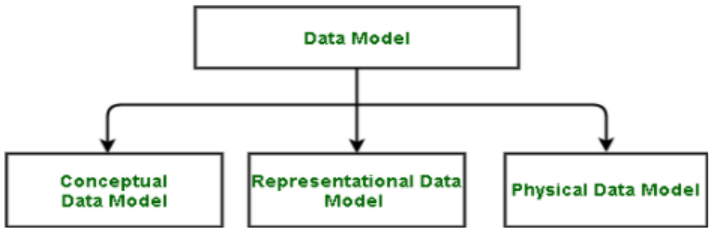


Figure 1.5 Data Model

Data Models in DBMS are:

- Hierarchical Model
- Network Model
- Entity-Relationship Model
- Relational Model
- Object-Oriented Data Model
- Object-Relational Data Model
- Flat Data Model
- Semi-Structured Data Model
- Associative Data Model
- Context Data Model

ENTITY RELATIONSHIP MODEL(ER MODEL)

Figure 1.6 Charles Bachman

- He was introduced “Bachman Diagrams”
- It is described data structures concepts
- Brown published works on real-world systems modeling
- James Martin added ERD refinements.
- The work of Chen, Bachman, Brown, Martin and others also contributed to the development of Unified Modeling Language (UML),
- It is used in software design

Figure 1.7 Peter Chen (a.k.a. Peter Pin-Shan Chen)

- Entity–relationship modeling was developed for database and design by Peter Chen and published in a 1976
- ER model becomes an abstract data model,
- defines a data or information structure which can be implemented in a database, typically a relational database.
- ENTITY-RELATIONSHIP DIAGRAM (ERD) displays the relationships of entity set stored in a database.
- ER diagrams help you to explain the logical structure of databases.
- ER diagram looks very similar to the flowchart
- ER Diagram includes many specialized symbols, and its meanings make this model unique
- The purpose of ER Diagram is to represent the entity framework infrastructure.
- ENTITY-RELATIONSHIP DIAGRAM (ERD) describes interrelated things of interest in a specific domain of knowledge.
- ER diagrams help to how “entities” such as people, objects or concepts relate to each other within a system.
- ER Diagrams are most often used to design or debug relational databases in the fields of software engineering, business information systems, education and research.

- ER Diagram use a defined set of symbols such as rectangles, diamonds, ovals and connecting lines to depict the interconnectedness of entities, relationships and their attributes.
- The purpose of ER Diagram is to represent grammatical structure
- ER model defines the conceptual view of a database
- It works around real-world entities and the associations among them
- ER model is considered a good option for designing databases.
- ER modeling helps you to analyze data requirements systematically to produce a well-designed database
- The three main components of the ER Model are
- entities,
- attributes
- and relationships.

Facts about ER Diagram Model

- ER model allows you to draw Database Design
- It is an easy to use graphical tool for modeling data Widely used in Database Design
- It is a GUI representation of the logical structure of a Database
- It helps you to identifies the entities which exist in a system and the relationships between those entities

Why use ER Diagrams?

- Helps you to define terms related to entity relationship modeling
- Provide a preview of how all your tables should connect, what fields are going to be on each table
- Helps to describe entities, attributes, relationships
- ER diagrams are translatable into relational tables which allows you to build databases quickly
- ER diagrams can be used by database designers as a blueprint for implementing data in specific software applications
- The database designer gains a better understanding of the information to be contained in the database with the help of ERP diagram
- Enterprise resource planning
- ERD is allowed you to communicate with the logical structure of the database to users

Advantages and Disadvantages of E-R Data Model

- **Advantages of an E-R Model:**
- Straightforward relation representation: Having designed an E-R diagram for a database application, the relational representation of the database model becomes relatively straightforward.
- Easy conversion for E-R to other data model: Conversion from E-R diagram to a network or hierarchical data model can· easily be accomplished.
- **Disadvantages of an E-R Model:**
- No industry standard for notation: There is no industry standard notation for developing an E-R diagram
- Popular for high-level design: The E-R data model is especially popular for high level.

ER MODEL- SYMBOLS

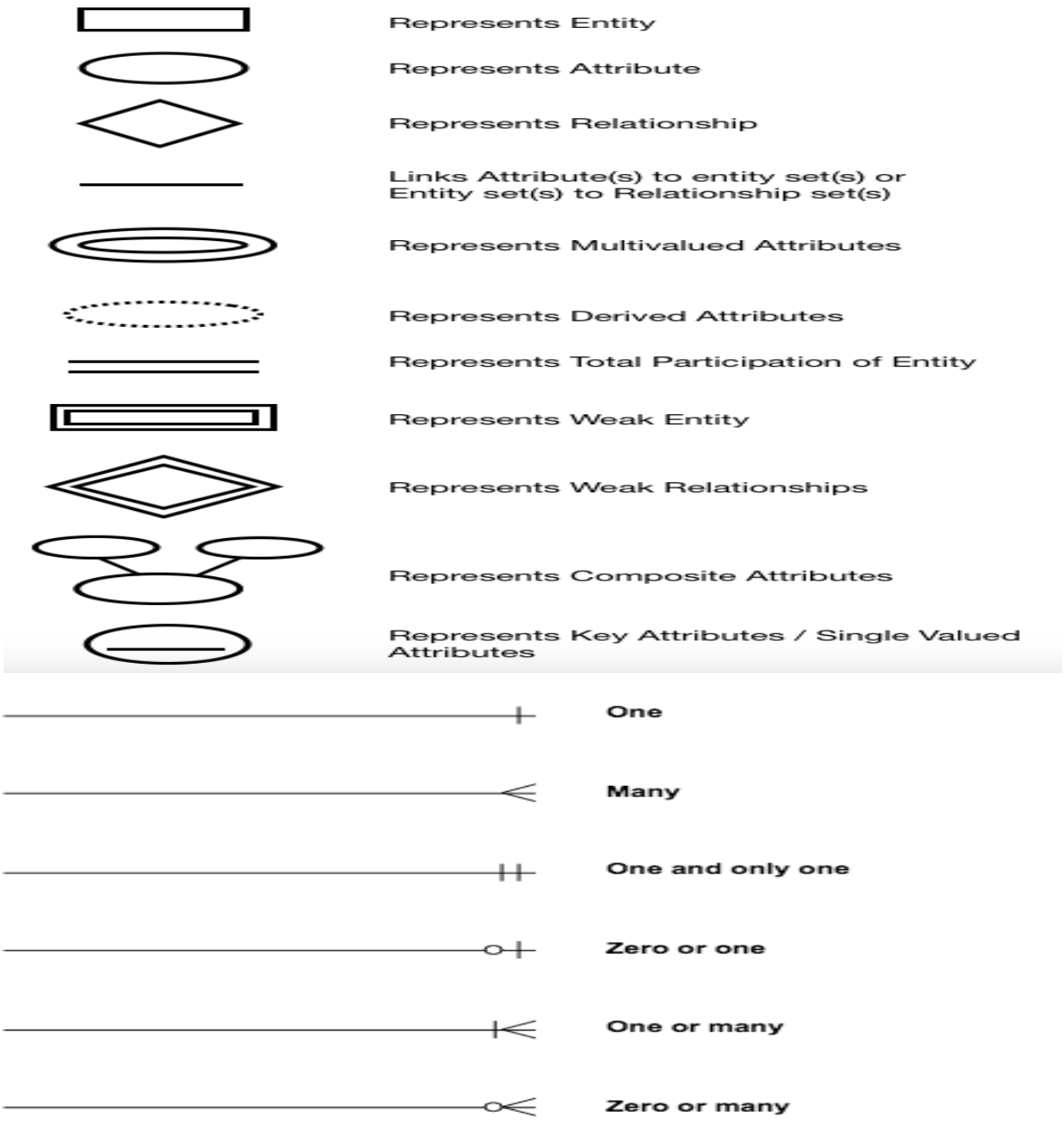


Figure 1.7 ER MODEL- SYMBOLS

Entity, Entity Type, Entity Set

- An Entity may be an object with a physical existence – a particular person, car, house, or employee – or it may be an object with a conceptual existence – a company, a job, or a university course.
- Entity is an object of Entity Type and set of all entities is called as entity set. e.g.; E1 is an entity having Entity Type Student and set of all students is called Entity Set. In ER diagram, Entity Type is represented as:

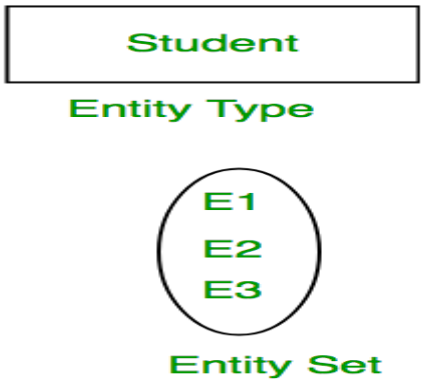


Figure 1.8 Entity Type, Entity Set

Attribute(s)

Attributes are the properties which define the entity type. For example, Roll_No, Name, DOB, Age, Address, Mobile_No are the attributes which defines entity type Student. In ER diagram, attribute is represented by an oval.

Figure 1.9 Attribute

For example, a car entity would be described by attributes such as price, registration number, model number, color etc. Attributes are indicated by ovals in an E-R diagram

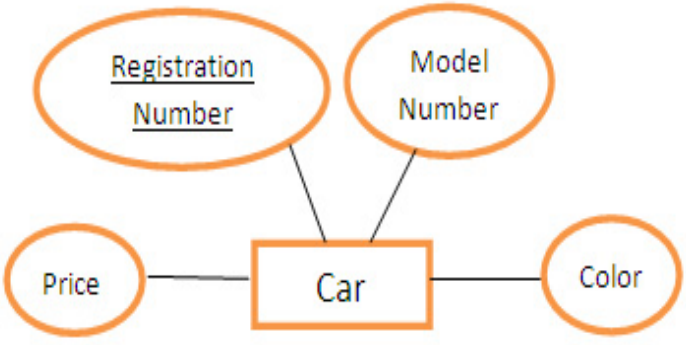


Figure 1.9 Example of Attribute

1. **Key Attribute –**

The attribute which **uniquely identifies each entity** in the entity set is called key attribute.For example, Roll_No will be unique for each student. In ER diagram, key attribute is represented by an oval with underlying lines.

Figure 1.10 Key Attribute

- If an attribute of a particular entity represents single value for each instance, then it is called a single-valued attribute. For example, Ramesh, Kamal and Suraj are the instances of entity 'student' and each of them is issued a separate roll number. A single oval is used to represent this attribute.

2. **Composite Attribute –**

An attribute **composed of many other attribute** is called as composite attribute. For example, Address attribute of student Entity type consists of Street, City, State, and Country. In ER diagram, composite attribute is represented by an oval comprising of ovals.

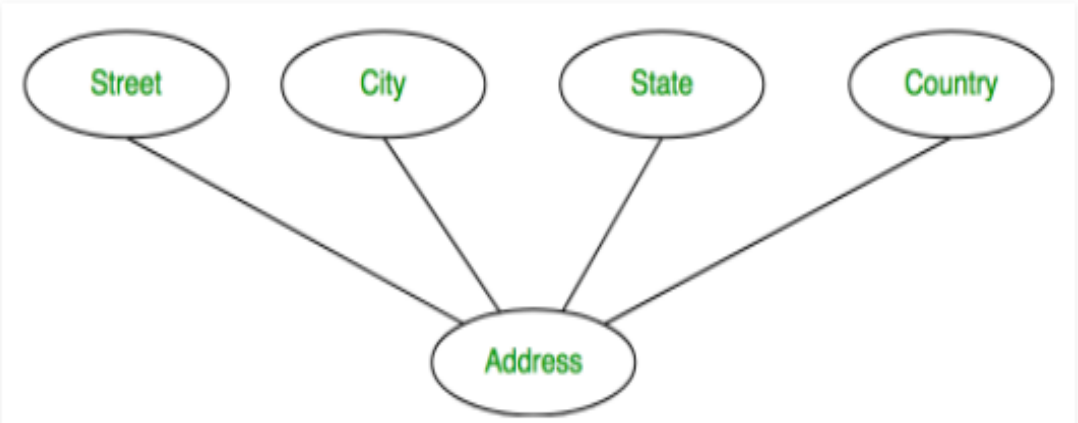


Figure 1.11 Composed Attribute

3. **Multivalued Attribute –**

An attribute consisting **more than one value** for a given entity. For example, Phone_No (can be more than one for a given student). In ER diagram, multivalued attribute is represented by double oval.

Figure 1.12 Multivalued Attribute

4. **Derived Attribute –**

An attribute which can be **derived from other attributes** of the entity type is known as derived attribute. e.g.; Age (can be derived from DOB). In ER diagram, derived attribute is represented by dashed oval.

Figure 1.13 Derived Attribute

Relationships

Relationships:- A relationship is defined as bond or attachment between 2 or more entities. For example,

- An employee assigned a project.
- Teacher teaches a student.
  Author writes a book

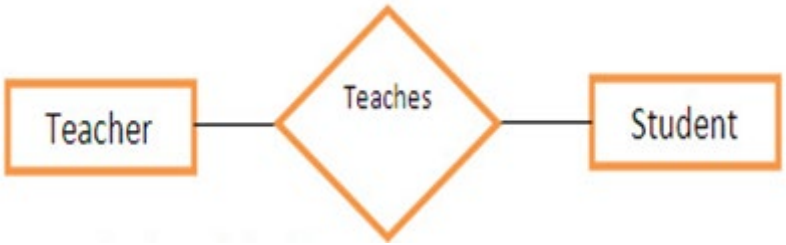


Figure 1.14 Examples of Relationships

Degree of a relationship set:

The number of different entity sets **participating in a relationship** set is called as degree of a relationship set.

1. **Unary Relationship –**

   When there is **only ONE entity set participating in a relation**, the relationship is called as unary relationship. For example, one person is married to only one person.

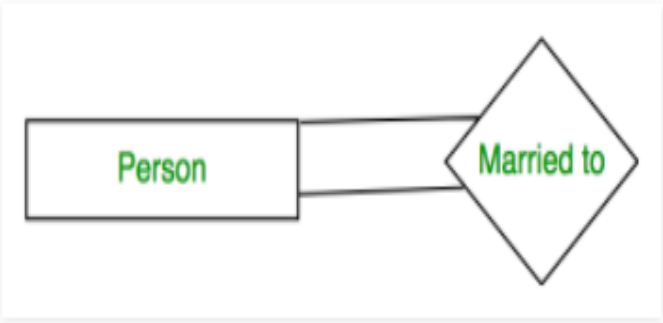


Figure 1.15 Unary Relationships

2. **Binary Relationship –**

   When there are **TWO entities set participating in a relation**, the relationship is called as binary relationship.For example, Student is enrolled in Course.

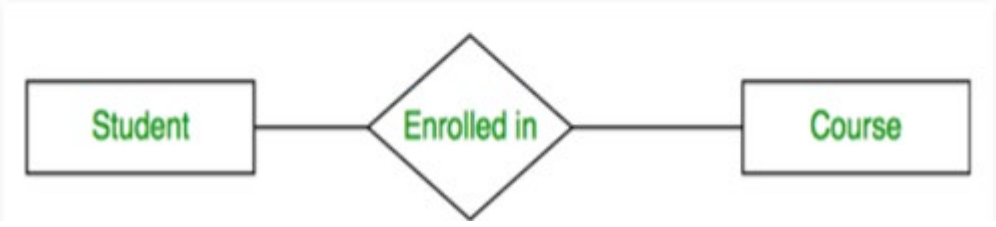


Figure 1.15 Binary Relationships

Cardinality:

The **number of times an entity of an entity set participates in a relationship** set is known as cardinality. Cardinality can be of different types:

1. **One to one –** When each entity in each entity set can take part **only once in the relationship**, the cardinality is one to one. Let us assume that a male can marry to one female and a female can marry to one male. So the relationship will be one to one.

Figure 1.15 One to One Relationships

2. **Many to one –** When entities in one entity set **can take part only once in the relationship set and entities in other entity set can take part more than once in the relationship set,** cardinality is many to one. Let us assume that a student can take only one course but one course can be taken by many students. So the cardinality will be n to 1. It means that for one course there can be n students but for one student, there will be only one course.

Figure 1.15 Many to One Relationships

3. **Many to many –** When entities in all entity sets can **take part more than once in the relationship** cardinality is many to many. Let us assume that a student can take more than one course and one course can be taken by many students. So the relationship will be many to many.

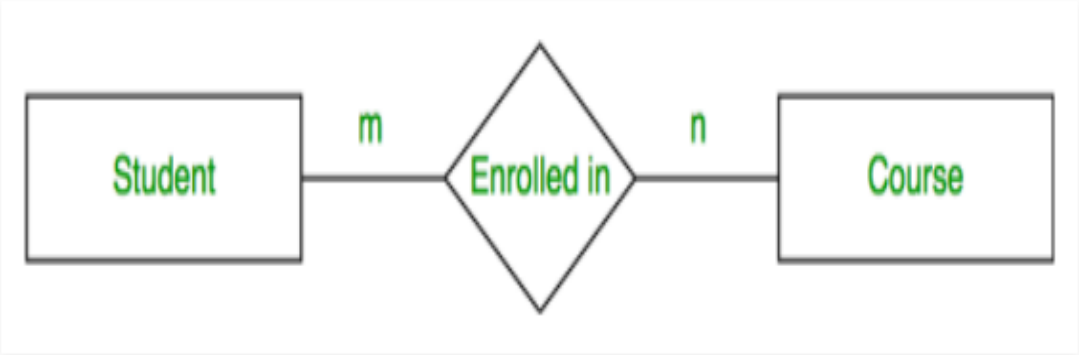


Figure 1.15 Many to Many Relationships

Participation Constraint:

Participation Constraint is applied on the entity participating in the relationship set.

1. **Total Participation –** Each entity in the entity set **must participate** in the relationship. If each student must enroll in a course, the participation of student will be total. Total participation is shown by double line in ER diagram.
2. **Partial Participation –** The entity in the entity set **may or may NOT participat**e in the relationship. If some courses are not enrolled by any of the student, the participation of course will be partial.
   The diagram depicts the 'Enrolled in' relationship set with Student Entity set having total participation and Course Entity set having partial participation.

Figure 1.16 Participation Constraints

Strong Entity

The strong entity has a primary key. Weak entities are dependent on strong entity. Its existence is not dependent on any other entity.

Strong Entity is represented by a single rectangle –

Figure 1.17 Strong Entity

Weak Entity

The weak entity in DBMS do not have a primary key and are dependent on the parent entity. It mainly depends on other entities.

Weak Entity is represented by double rectangle –

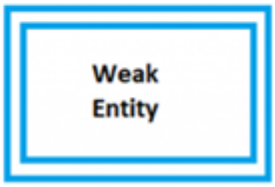


Figure 1.18 Weak Entity

The Enhanced ER Model

- As the complexity of data increased in the late 1980s
- it became more and more difficult to use the traditional ER Model for database modelling.
- Hence some improvements or enhancements were made to the existing ER Model to make it able to handle the complex applications better.
- As the basic concepts of ER modeling are often not enough to represent the requirements of the newer complex applications
- which therefore stimulated the need to develop additional 'semantic' modeling concepts.
- Various semantic data models have been proposed, and some of the most important semantic concepts have been successfully incorporated into the original ER model.
- It is a conceptual data model in which semantic information is included. This means that the model describes the meaning of its instances.(TV-REMOTE)
- The ER model, supported with additional semantic concepts, is called the Enhanced Entity-Relationship (EER) model.
- Enhanced ER Model, along with other improvements, three new concepts were added to the existing ER Model, they were:

1. Generalization
2. Specialization

3. Aggregration

Generalization

- Generalization is a bottom-up approach in which two lower level entities combine to form a higher level entity.
- In generalization, the higher level entity can also combine with other lower level entities to make further higher level entity.
- It's more like Superclass and Subclass system, but the only difference is the approach, which is bottom-up.
- entities are combined to form a more generalised entity, in other words, sub-classes are combined to form a super-class.
- For example, Saving and Current account types entities can be generalised and an entity with name Account can be created, which covers both.

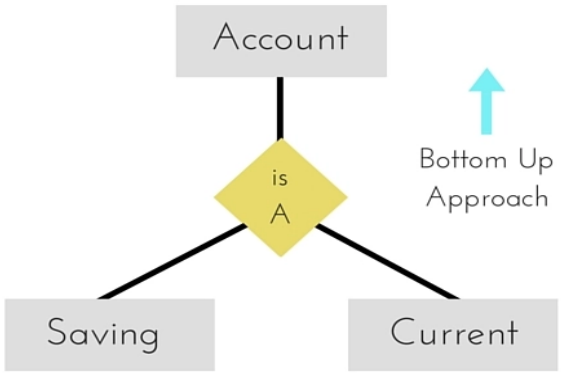


Figure 1.19 Generalizations

Specialization

Specialization is a process that defines a group entities which is divided into sub groups based on their characteristic.

It is a top down approach, in which one higher entity can be broken down into two lower level entity.

It maximizes the difference between the members of an entity by identifying the unique characteristic or attributes of each member.

It defines one or more sub class for the super class and also forms the superclass/subclass relationship.

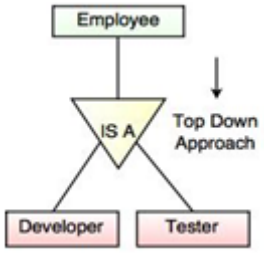


Figure 1.20 Specialization

Aggregation

- In aggregation, the relation between two entities is treated as a single entity. In aggregation, relationship with its corresponding entities is aggregated into a higher level entity.
- Center entity offers the Course entity act as a single entity in the relationship which is in a relationship with another entity visitor. In the real world, if a visitor visits a coaching center then he will never enquiry about the Course only or just about the Center instead he will ask the enquiry about both.

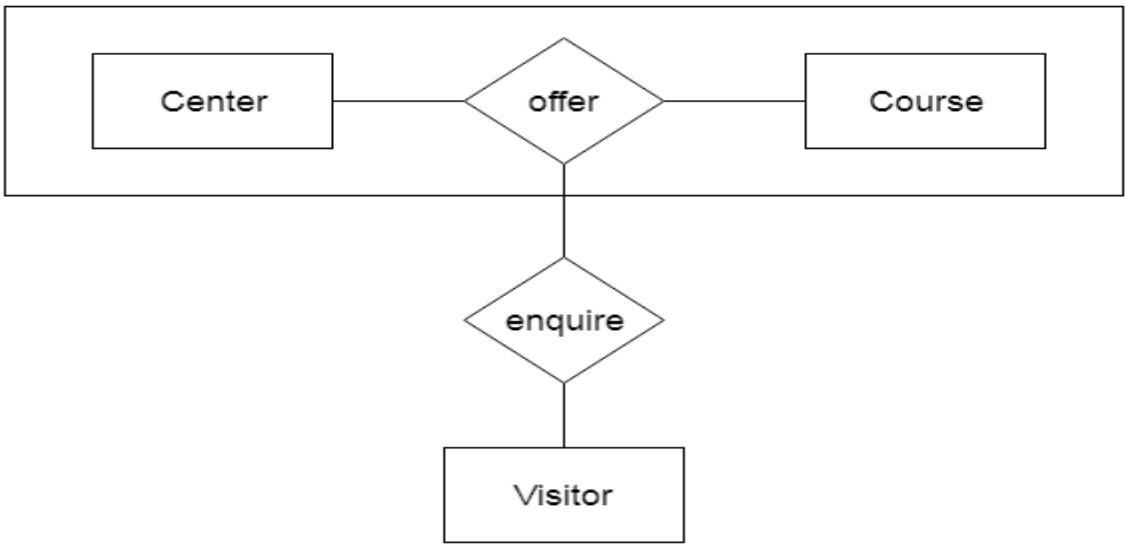


Figure 1.21 Aggregation

Need of Object Oriented Data Model :
To represent the complex real world problems there was a need for a data model that is closely related to real world. Object Oriented Data Model represents the real world problems easily.
Object Oriented Data Model :
In Object Oriented Data Model, data and their relationships are contained in a single structure which is referred as object in this data model. In this, real world problems are represented as objects with different attributes. All objects have multiple relationships between them. Basically, it is combination of Object Oriented programming and Relational Database Model as it is clear from the following figure :
Object Oriented Data Model

= Combination of Object Oriented Programming + Relational database model

Components of Object Oriented Data Model :

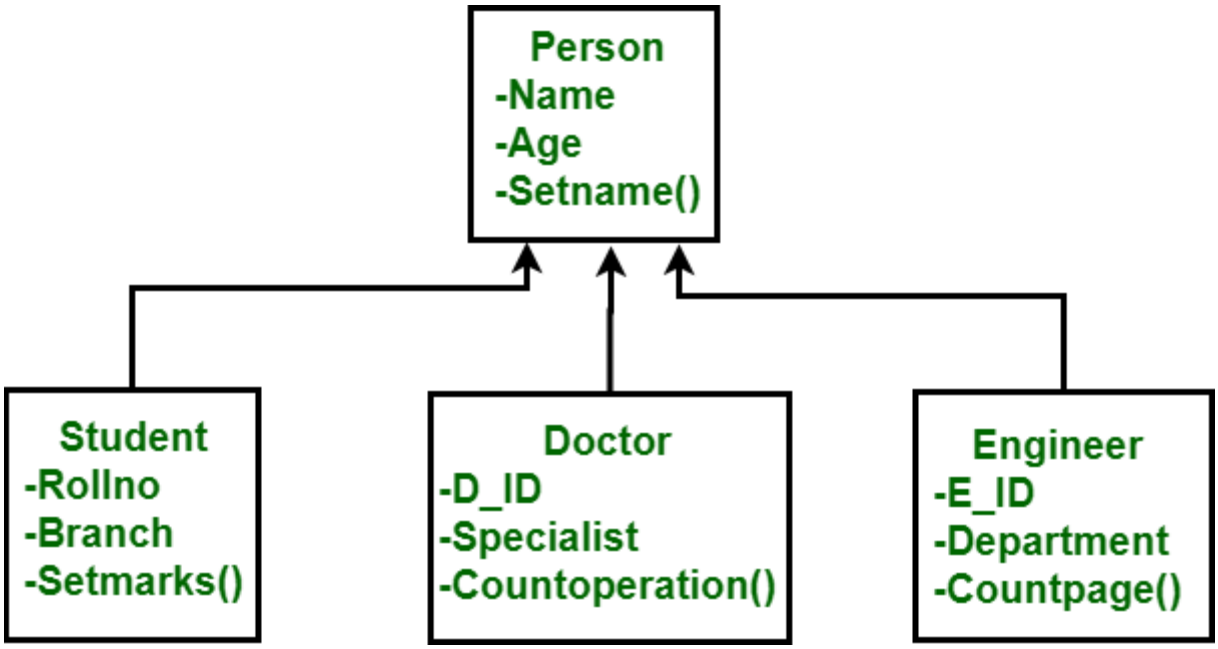


Figure 1.22 Basic Object Oriented Data Model

- **Objects** – An object is an abstraction of a real world entity or we can say it is an instance of class. Objects encapsulates data and code into a single unit which provide data abstraction by hiding the implementation details from the user. For example: Instances of student, doctor, engineer in above figure.
- **Attribute** – An attribute describes the properties of object. For example: Object is STUDENT and its attribute are Roll no, Branch, Semester in the Student class.
- **Methods** – Method represents the behavior of an object. Basically, it represents the real-world action. For example: Finding a STUDENT marks in above figure as Setmarks().
- **Class** – A class is a collection of similar objects with shared structure i.e. attributes and behavior i.e. methods. An object is an instance of class. For example: Person, Student, Doctor, Engineer in above figure.

```
class student
{
   char Name[20];
   int roll_no;
   --
   --
   public:
   void search();
   void update();
}
```

In this example, students refers to class and S1, S2 are the objects of class which can be created in main function.

Inheritance – By using inheritance, new class can inherit the attributes and methods of the old class i.e. base class. For example: as classes Student, Doctor and Engineer are inherited from the base class Person.

Advantages of Object Oriented Data Model :

- Codes can ne reused due to inheritance.
- Easily understandable.
- Cost of maintenance can reduced due to reusability of attributes and functions because of inheritance.

Disadvantages of Object Oriented Data Model :

- It is not properly developed so not accepted by users easily.

Record storage and file organizations

Record Storage

Magnetic disks -

bit - by magnetizing an area on disk in certain ways, we can make it represent a bit value
of either 0 or 1
byte - bits are grouped into either 4 or 8 to make a byte - one character per byte
single-sided - disks that store info on only one of its magnetic surfaces
double-sided - both surfaces are used multiple disks stacked on top of each other form a disk pack
info stored on a disk in concentric circles of small
width - each circle called a track
cylinder - tracks of the same diameter on different disks in a disk pack
sector - section of disk divided by an angle from the center - hard-coded on disk
blocks - division of a track made by operating system during formatting separated by fixed-size interblock gap - include control info identifying next block transfer of info from main mem to disk is done in blocks
hardware address - combination of surface number, track number and block number
buffer - contiguous reserved area in main mem that holds one block
read- block from disk copied to buffer
write - contents of buffer copied to disk block
cluster - group of blocks - sometimes data moved in clusters - buffer size matches cluster size
disk drive - system that includes - read/write head - one for each surface - actuator moves r/w head over a cylinder of tracks - motor to rotate disks
transfer a disk block:
- position read/write head on correct track - seek time
- rotate disk to the desired block - rotational delay
- transfer data to/from main memory - block transfer time
- seek time and rotational delay usually much longer than transfer time
- locating data on disk is a major bottleneck in DB applications
- file structures to minimize number of block transfers needed to locate and transfer required data

Magnetic tape
sequential access
tape drive - reads/writes data from/to tape reel
blocks are larger than on disk - so are interblock gaps used to back up DB
Buffering blocks
can reserve several buffers in main mem to speed up transfer while one buffer is being written, CPU can process data in other buffer
double buffering - can be used to write a continuous stream of blocks from mem to disk permits continuous reading/writing of data on consecutive

disk blocks -eliminating seek time and rotational delay fo all but the first block Placing file records on disk
data stored in *record*s- collection of related data values - each value formed by one or
more bytes - corresponds to *field* of a record
EX: one TOY record consists of the fields TOY#, name, manufacturer, etc...
record type - collection of field names and their data types
data type - (of a field) specifies type of values the field can take number of bytes for each data type is fixed for a given computer system
BLOB - binary large object - large unstructured objects that represent images, digitized video or audio or free text
- usually record includes a pointer to BLOB - somewhere else on disk
file - sequence of records
usually - all records in a file are of the same record type
fixed-length records - all records same size
variable-length records - different sizes
- variable-length fields in record
EX: a text field for customer comments
- repeating field - more than one value for a field in an individual record
EX: ages of children in customer relation
- optional fields
- mixed file - different record types
separator character used to separate variable-length records in a file
records are mapped to disk blocks - the unit of transfer from disk to MM buffers
blocking factor for a file - the number of records per block on disk
let B=size of block (bytes); R=size of record
B >= R ==> bfr = floor(B/R) (rounds down to next integer)
unused space = B - (bfr * R) bytes
unused space can be used by storing part of a record on one block and the rest on another
- a pointer at the end of one block to the other block storing the rest of the record
- called *spanned records*
- must be used if R >= B

bfr for variable-length records represents the average number of records per block.

File organizations

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

What is File Organization?

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. **Pile File Method –** This method is quite simple, in which we store the records in a sequence i.e one after other in the order in which they are inserted into the tables.

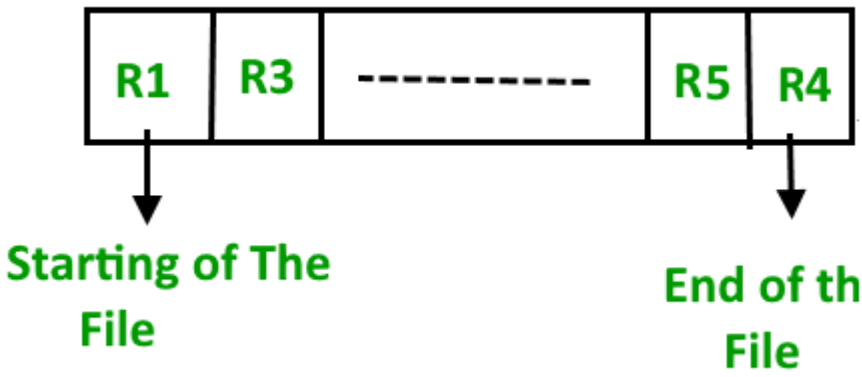


Figure 1.23 Pile File Method

2. **Insertion of new record** –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.

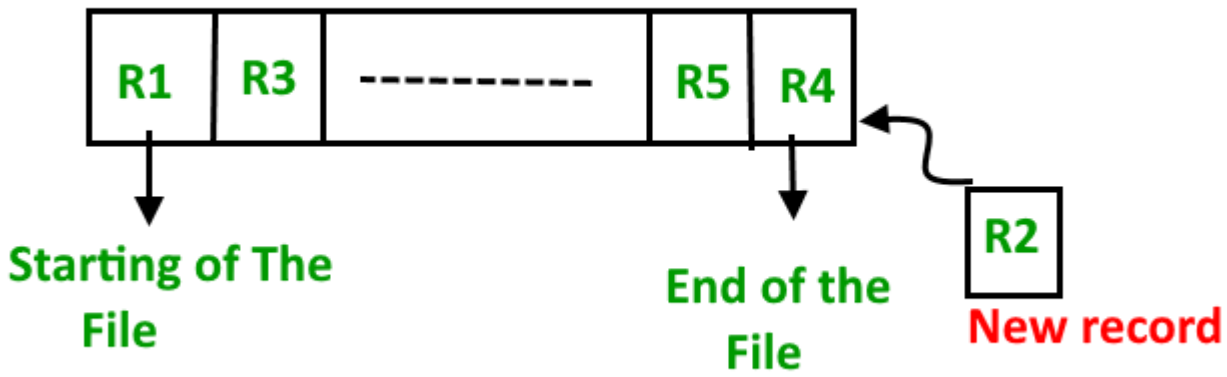


Figure 1.24 Insertion of new record

3. **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.

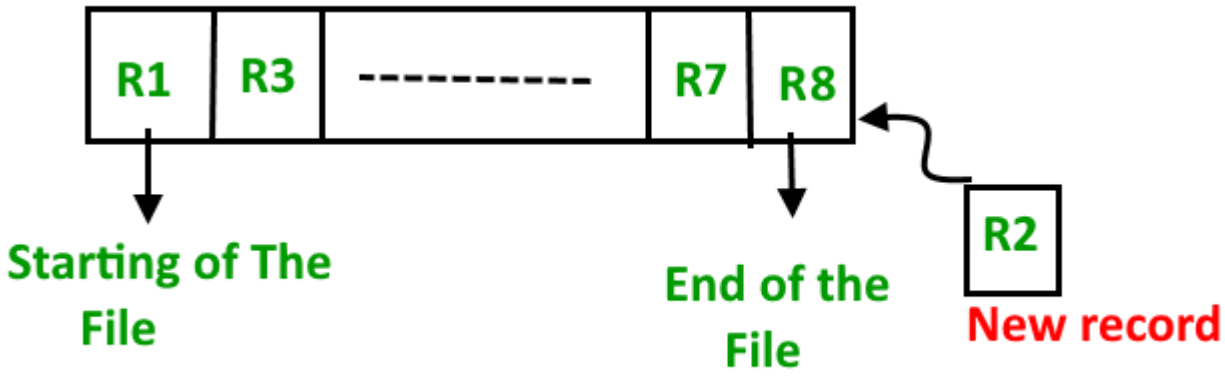


Figure 1.25 Sorted File Method

Insertion of new record –

Let us assume that there is a preexisting sorted sequence of four records R1, R3, and so on upto R7 and R8. Suppose a new record R2 has to be inserted in the sequence, then it will be inserted at the end of the file and then it will sort the sequence .

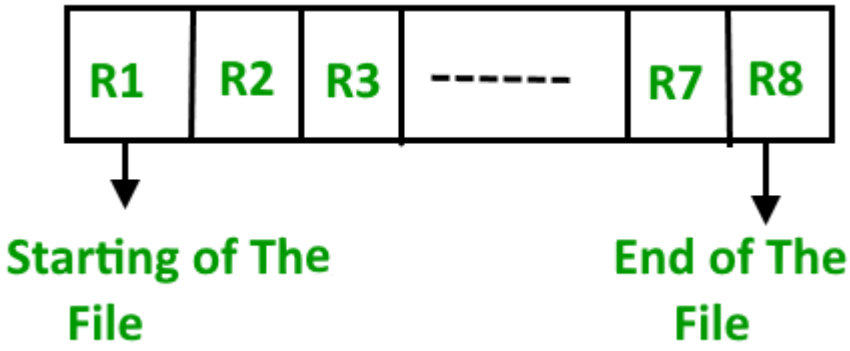


Figure 1.26 Insertion of new record

Pros and Cons of Sequential File Organization –

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.

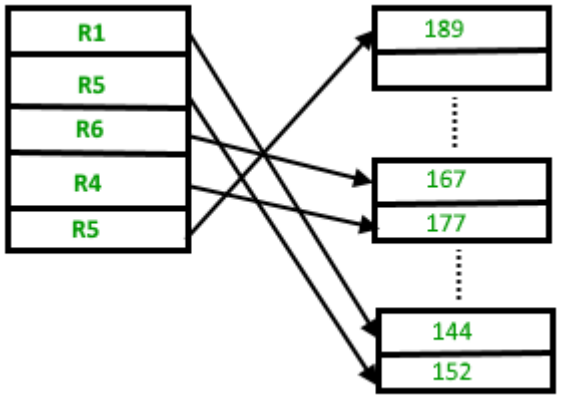


Figure 1.27 Heap File

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.

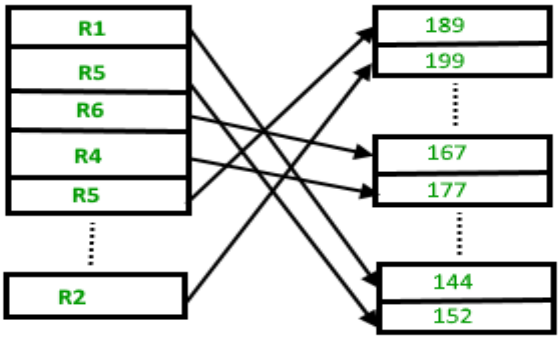


Figure 1.27 Heap File Insertion of new record

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 and Cons of Heap File Organization–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.

Index Structures for files

What is Indexing?

Indexing is a data structure technique which allows you to quickly retrieve records from a database file. An Index is a small table having only two columns. The first column comprises a copy of the primary or candidate key of a table. Its second column contains a set of pointers for holding the address of the disk block where that specific key value stored.

An index -

- Takes a search key as input
- Efficiently returns a collection of matching records.

Types of Indexing

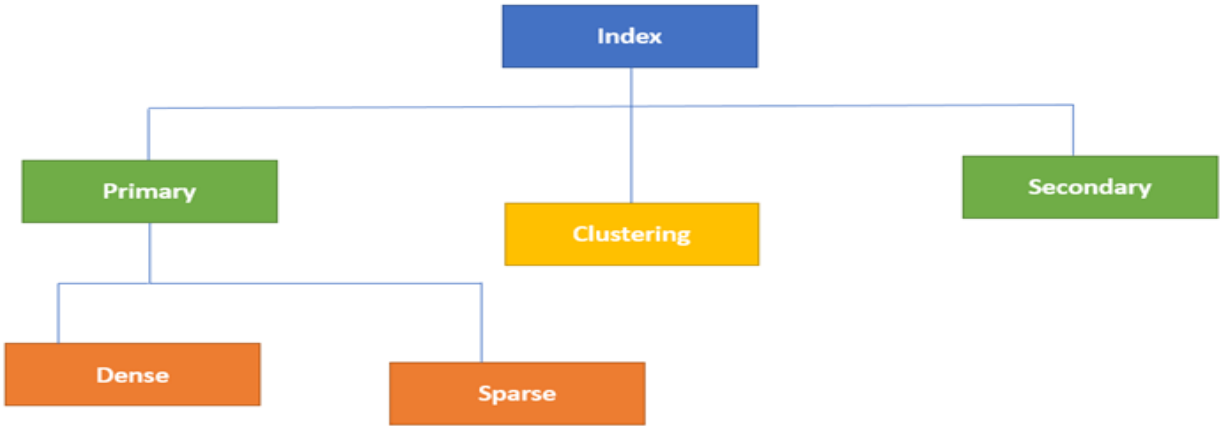


Figure 1.28 Type of Indexes in Database

Indexing in Database is defined based on its indexing attributes. Two main types of indexing methods are:

- Primary Indexing
- Secondary Indexing

Primary Index

Primary Index is an ordered file which is fixed length size with two fields. The first field is the same a primary key and second, filed is pointed to that specific data block. In the primary Index, there is always one to one relationship between the entries in the index table.

The primary Indexing in DBMS is also further divided into two types.

- Dense Index

- Sparse Index

Dense Index

In a dense index, a record is created for every search key valued in the database. This helps you to search faster but needs more space to store index records. In this Indexing, method records contain search key value and points to the real record on the disk.

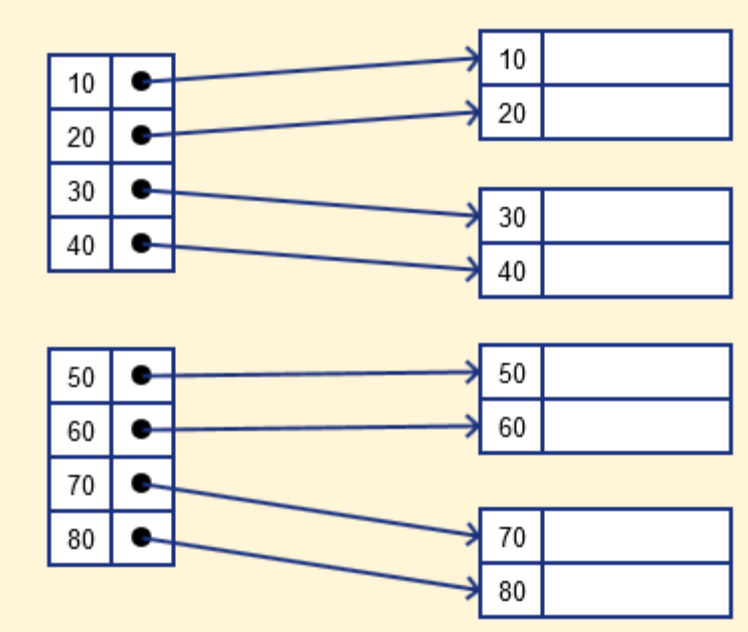


Figure 1.29 Dense index

Sparse Index

It is an index record that appears for only some of the values in the file. Sparse Index helps you to resolve the issues of dense Indexing in DBMS. In this method of indexing technique, a range of index columns stores the same data block address, and when data needs to be retrieved, the block address will be fetched.

However, sparse Index stores index records for only some search-key values. It needs less space, less maintenance overhead for insertion, and deletions but It is slower compared to the dense Index for locating records.

Below is an database index Example of Sparse Index

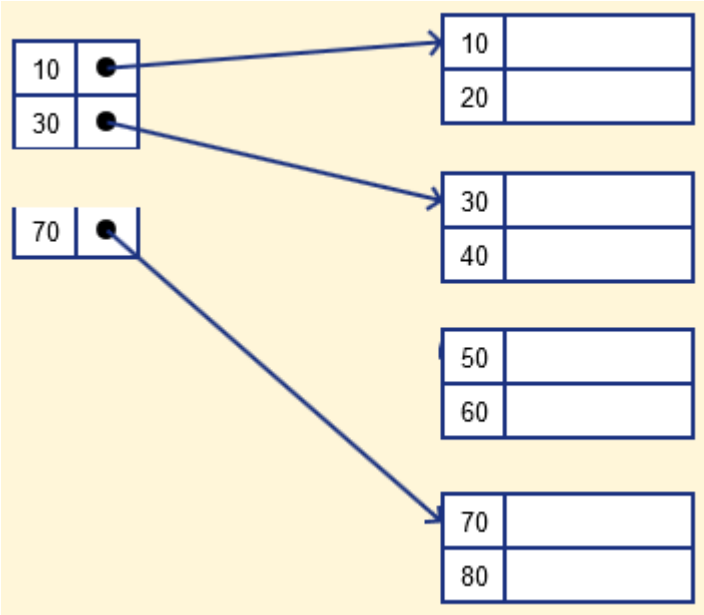


Figure 1.29 Sparse index

Secondary Index

The secondary Index in DBMS can be generated by a field which has a unique value for each record, and it should be a candidate key. It is also known as a non-clustering index.

This two-level database indexing technique is used to reduce the mapping size of the first level. For the first level, a large range of numbers is selected because of this; the mapping size always remains small.

Example of secondary Indexing

Let's understand secondary indexing with a database index example:

In a bank account database, data is stored sequentially by acc_no; you may want to find all accounts in of a specific branch of ABC bank.

Here, you can have a secondary index in DBMS for every search-key. Index record is a record point to a bucket that contains pointers to all the records with their specific search-key value.

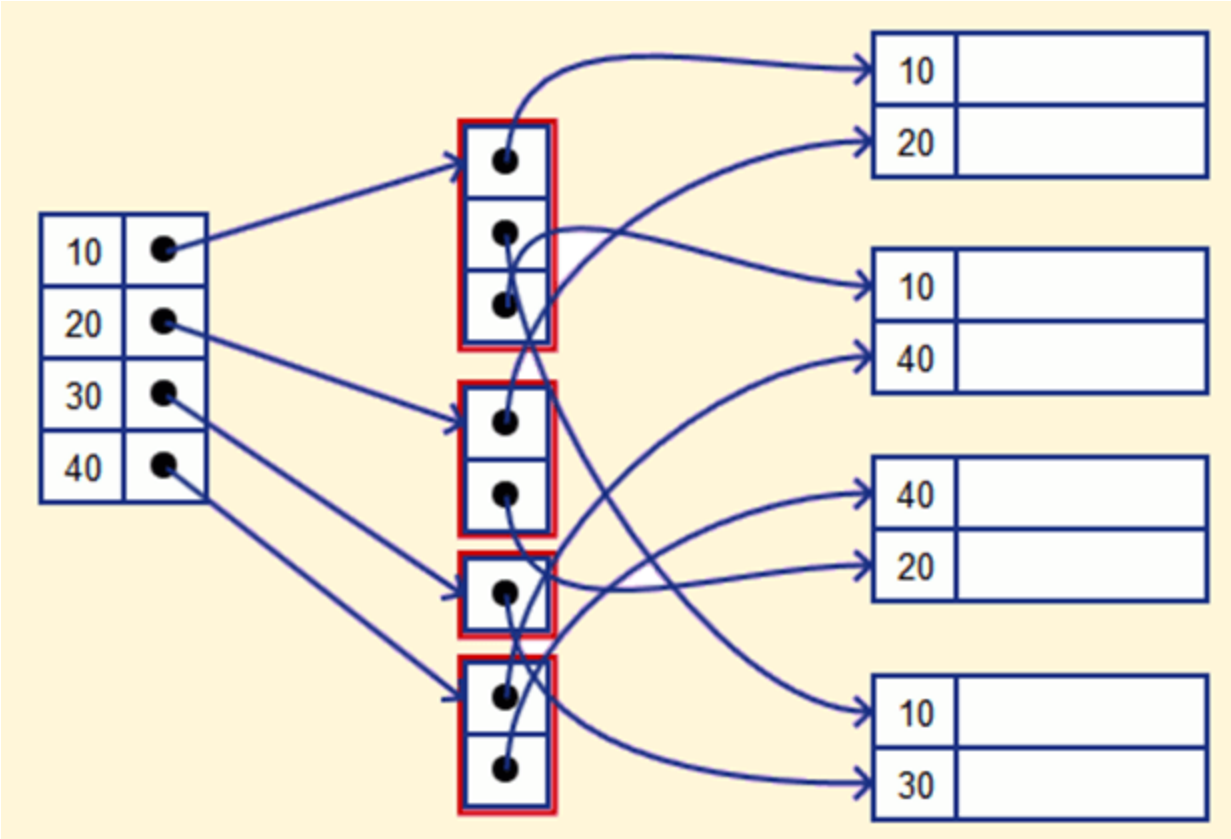


Figure 1.30 Secondary index

Clustering Index

In a clustered index, records themselves are stored in the Index and not pointers. Sometimes the Index is created on non-primary key columns which might not be unique for each record. In such a situation, you can group two or more columns to get the unique values and create an index which is called clustered Index. This also helps you to identify the record faster.

Example:

Let's assume that a company recruited many employees in various departments. In this case, clustering indexing in DBMS should be created for all employees who belong to the same dept.

It is considered in a single cluster, and index points point to the cluster as a whole. Here, Department _no is a non-unique key.

What is Multilevel Index?

Multilevel Indexing in Database is created when a primary index does not fit in memory. In this type of indexing method, you can reduce the number of disk accesses to short any record and kept on a disk as a sequential file and create a sparse base on that file.

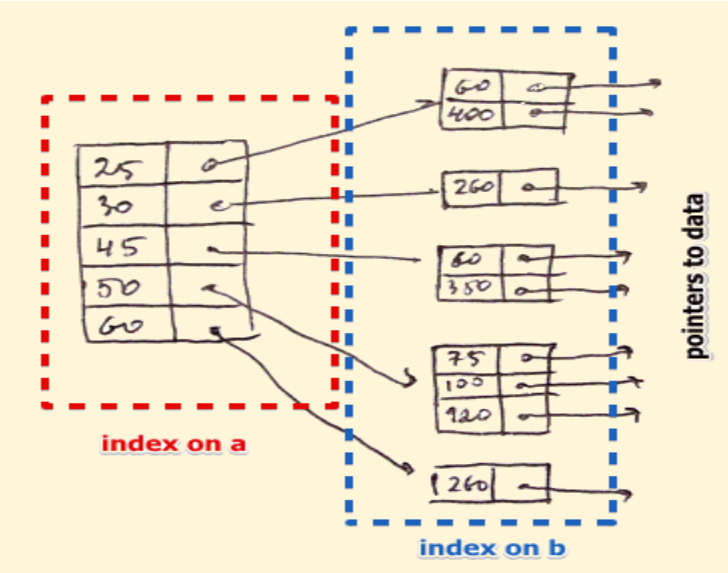


Figure 1.31 Multilevel index

B-Tree Index

B-tree index is the widely used data structures for tree based indexing in DBMS. It is a multilevel format of tree based indexing in DBMS technique which has balanced binary search trees. All leaf nodes of the B tree signify actual data pointers.

Moreover, all leaf nodes are interlinked with a link list, which allows a B tree to support both random and sequential access.

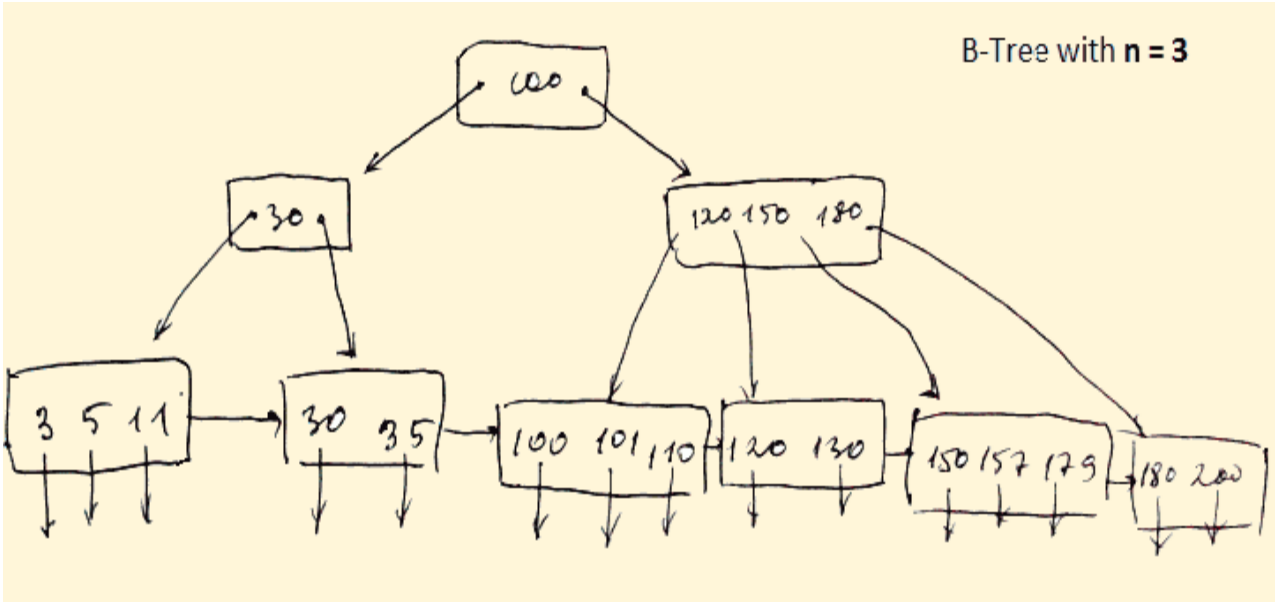


Figure 1.31 B-Tree index

- Lead nodes must have between 2 and 4 values.
- Every path from the root to leaf are mostly on an equal length.

- Non-leaf nodes apart from the root node have between 3 and 5 children nodes.
- Every node which is not a root or a leaf has between n/2] and n children.

Advantages of Indexing

Important pros/ advantage of Indexing are:

- It helps you to reduce the total number of I/O operations needed to retrieve that data, so you don't need to access a row in the database from an index structure.
- Offers Faster search and retrieval of data to users.
- Indexing also helps you to reduce tablespace as you don't need to link to a row in a table, as there is no need to store the ROWID in the Index. Thus you will able to reduce the tablespace.
- You can't sort data in the lead nodes as the value of the primary key classifies it.

Disadvantages of Indexing

Important drawbacks/cons of Indexing are:

- To perform the indexing database management system, you need a primary key on the table with a unique value.
- You can't perform any other indexes in Database on the Indexed data.
- You are not allowed to partition an index-organized table.
- SQL Indexing Decrease performance in INSERT, DELETE, and UPDATE query.

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 COMPUTING

DEPARTMENT OF COMPUTER SCIENCE

UNIT II -DATABASE MANAGEMENT SYSTEM- SBS1205

SBS1205 DATABASE MANAGEMENT SYSTEM

UNIT II

Relational Model, Languages and Systems: The Relational Data Model, Relational Constraints, and the relational Algebra – SQL – The Relational Database Standard – ER and EER to Relational Mapping and Other Relational Languages – Examples of Relational Database Management Systems: Oracle and Microsoft Access.

Data model

- Data models define how the logical structure of a database is modeled.
- Data models define how data is connected to each other and how they are processed and stored inside the system.
- A Database model defines the logical design and structure of a database and defines how data will be stored, accessed and updated in a database management system

A Data Model in Database Management System (DBMS), is the concept of tools that are developed to summarize the description of the database.

Data Models in DBMS are:

- Hierarchical Model
- Network Model
- Entity-Relationship Model
- Relational Model
- Object-Oriented Data Model
- Object-Relational Data Model
- Flat Data Model
- Semi-Structured Data Model
- Associative Data Model
- Context Data Model

RELATIONAL MODEL

Relational model is an example of implementation model or representational model. An implementation model provides concepts that may be understood by end users butthat are not too far removed from the way data is organized within the computer.

CONCEPTS:

The relational model represents database as a collection of relations. A **relation** is atable of values, and each row in the table represents a collection of related data values.In the relational model terminology a row is called a **tuple**, a column header is called an**attribute**. The data type describing the types of values that can appear in each column is called a **domain**. Example
Consider the RELATION: **STUDENT**

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

Relation Schema: A relation schema represents name of the relation with its attributes. e.g.; STUDENT (ROLL_NO, NAME, ADDRESS, PHONE and AGE) is relation schema for STUDENT. If a schema has more than 1 relation, it is called Relational Schema.

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
2	RAMESH	TAMIL NADU	9652431543	18
3	SAM	ANDHRA PRADESH	9156253131	20
4	SURESH	DELHI		18

Tuple: Each row in the relation is known as tuple. The bellow relation contains 3 tuples

1	RAM	DELHI	9455123451	18
2	RAMESH	TAMIL NADU	9652431543	18
3	SAM	ANDHRA PRADESH	9156253131	20

- Attribute: Each column in a Table. Attributes are the properties which define a relation. e.g., Student_Rollno, NAME,etc.
- Tables – In the Relational model the, relations are saved in the table format. It is stored along with its

entities. A table has two properties rows and columns. Rows represent records and columns represent attributes.

- Tuple – It is nothing but a single row of a table, which contains a single record.
- Relation Schema: A relation schema represents the name of the relation with its attributes.
- Degree: The total number of attributes which in the relation is called the degree of the relation.
- Cardinality: Total number of rows present in the Table.
- Column: The column represents the set of values for a specific attribute.
- Relation instance – Relation instance is a finite set of tuples in the RDBMS system. Relation instances never have duplicate tuples.
- Relation key - Every row has one, two or multiple attributes, which is called relation key.
- Attribute domain – Every attribute has some pre-defined value and scope which is known as attribute domain

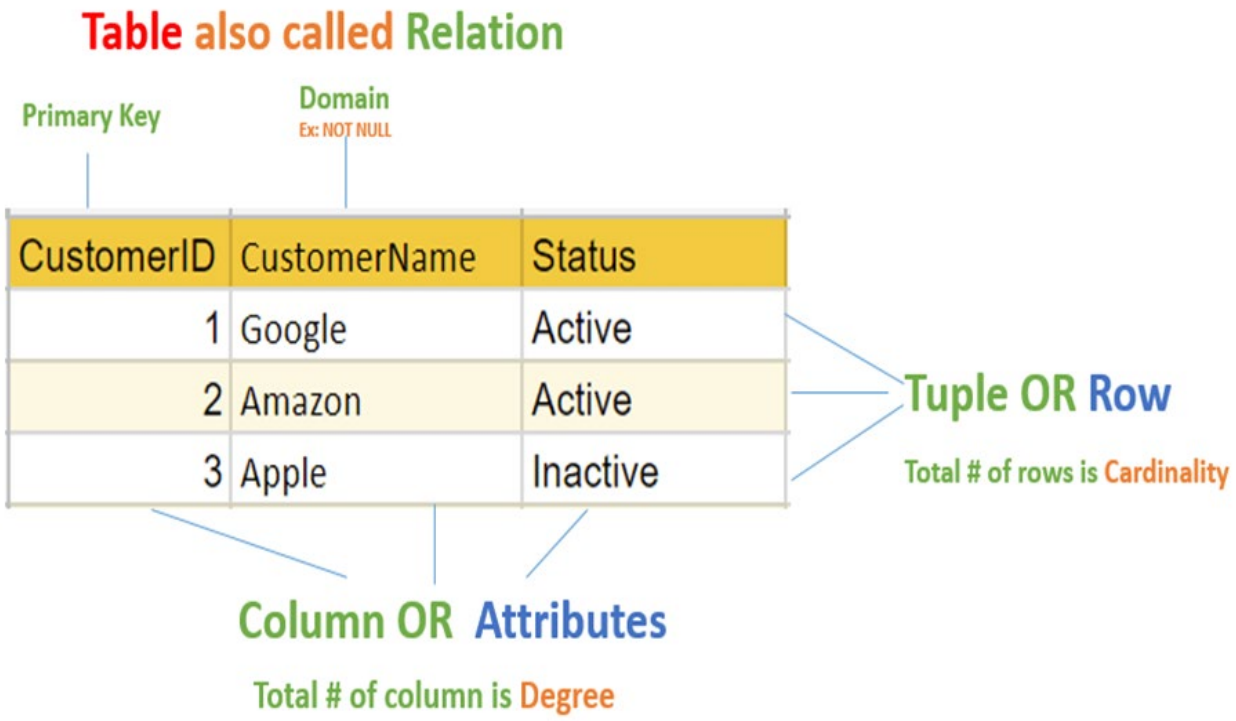


Figure 2.1 Relational concepts

Relational Database Constraints

- Constraints are the rules enforce limits to the data or type of data that can be inserted/updated/deleted from a table.

1Domain constraints

2. Key constraints

3. Entity Integrity constraints

4.Referential integrity constraints

Domain Constraint-

- Domain constraint defines the domain or set of values for an attribute.
- It specifies that the value taken by the attribute must be the atomic value from its domain.

Example- Consider the following Student table-

- Here, value ‘A’ is not allowed since only integer values can be taken by the age attribute**.**

STU_ID	Name	Age
S001	Akshay	20
S002	Abhishek	21
S003	Shashank	20
S004	Rahul	A

Key constraints

An attribute that can uniquely identify a tuple in a relation is called the key of the table. The value of the attribute for different tuples in the relation has to be unique.

Example:

In the given table, CustomerID is a key attribute of Customer Table. It is most likely to have a single key for one customer, CustomerID =1 is only for the CustomerName =" Google".

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive

Entity Integrity Constraints :

Entity Integrity constraints says that no primary key can take NULL value, since using primary key we identify each tuple uniquely in a relation.

- Explanation:
In the above relation, EID is made primary key, and the primary key cant take NULL values but in the third tuple, the primary key is null, so it is a violating Entity Integrity constraints.

EID	Name	Phone
01	Bikash	7002494274
02	Paul	6026526747
NULL	Sony	9234567892

Referential integrity constraints

- Referential integrity constraints is base on the concept of Foreign Keys.
- A foreign key is an important attribute of a relation which should be referred to in other

relationships.

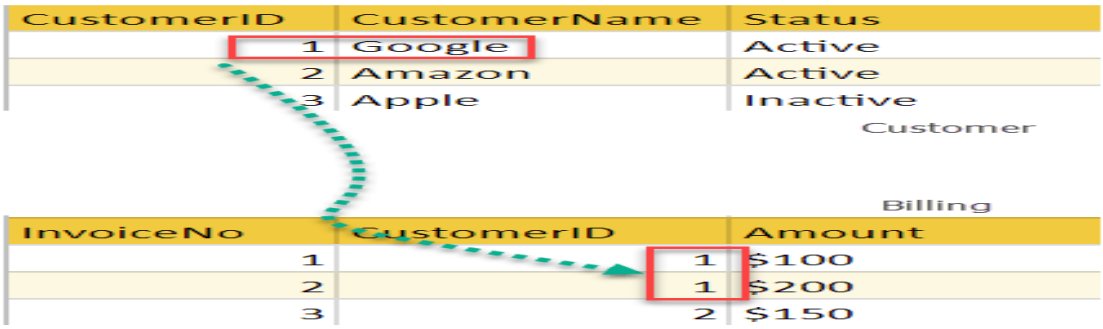
- Referential integrity constraint state happens where relation refers to a key attribute of a different or same relation.

  However, that key element must exist in the table.

- In the example, we have 2 relations, Customer and Billing.

- Tuple for CustomerID =1 is referenced twice in the relation Billing. So we know

  CustomerName=Google has billing amount $300



Object-Oriented Data Model

A data model is a logical organization of the real-world objects(entities), constraints on them and relationships among objects. A data model that captures object- oriented concepts is an object-oriented data model. An objectoriented database is a collection of objects whose behavior and state, and the relationships are definedin accordance with an object-oriented data model.

Object-An object is a software bundle of variables and related methods. Softwareobjects are often used to model real-world objects you find in everyday life.

Messages-Software objects interact and communicate with each other using messages.

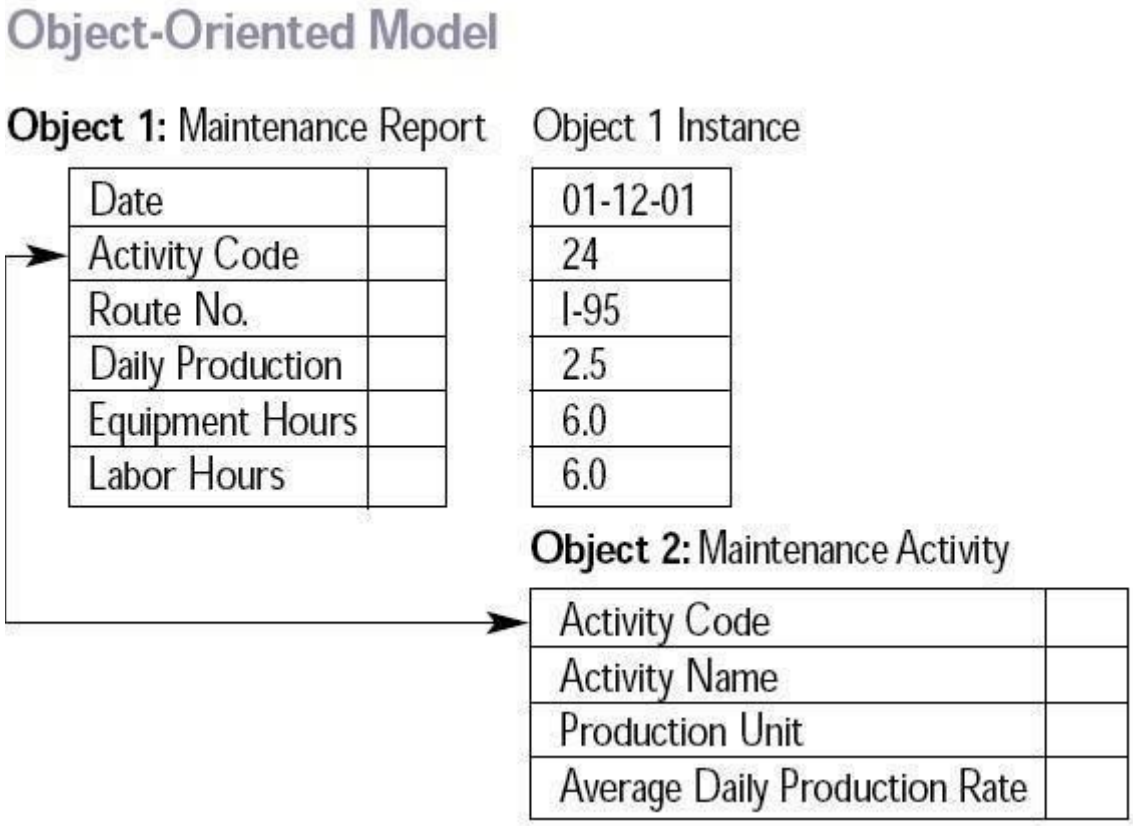
Classes-**A class is a blueprint or prototype that defines the variables and themethods common to all objects of a certain kind.**

Inheritance-A class inherits state and behavior from its *superclass*. Inheritance provides a powerful and natural mechanism for organizing and structuring softwareprograms.

Information Hiding-The ability to protect some components of the object from external

entities. This is realized by language keywords to enable a variable to be declared as *private* or *protected* to the owning *class*.

Polymorphism-The ability to replace an *object* with its *subobjects*. The ability of an*object-variable* to contain, not only that *object*, but also all of its *subobjects*. In contrast to a relational DBMS where a complex data structure must be flattened out to fit into tables or joined together from those tables to form the inmemory structure, object DBMSs have no performance overhead to store or retrieve a webor hierarchy of interrelated objects. This one-to-one mapping of objectprogramming language objects to database objects has two benefits over other storage approaches: it provides higher performance management of objects, and it enables better management of the complex interrelationships between objects. This makes object DBMSs better suited to support applications such as financial portfolio risk analysis systems, telecommunications service applications, world wide web document structures, design and manufacturing systems, and hospital patient record systems, which have complex relationships between data.



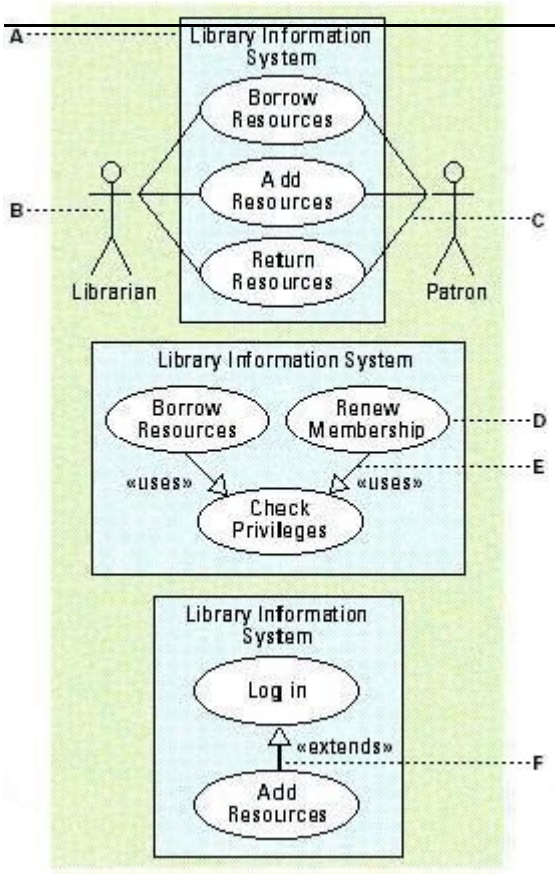


Figure 2.2. Sample user case diagram for object oriented data model.

RELATIONAL MODEL FOR COMPANY DATABASE

EMPLOYEE

FNAME	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO

DEPARTMENT

DNAME	DNUMBER	DMGRSSN	MGRSTARTDATE

DEPT_LOCATIONS

DNUMBER	DLOCATION

PROJECT

PNAME	PNUMBER	PLOCATION	DNUM

WORKS_ON

ESSN	PNO	HOURS

DEPENDENT

ESSN	DEPE_NAME	SEX	BDATE	RELATIONSHIP

---

1. INSERT OPERATION:

The insert operation provides a list of attribute values for a new tuple t that is to be inserted into a relation R. insert can violate any of the four types of constraints.
Example: Insert < *'ProductX', 501, 'Stafford', 4* > Into *Project.*

- If there is a null in the primary key field then that insertion violates entity integrity constraints.
- This insertion may violate the key constraint if there is another tuple with asame project no.
- This insertion may violate the referential integrity constraint if there is no information regarding DNO = 4, which is the foreign key.

2. DELETE OPERATION:

The delete operation is used to delete existing tuples, which satisfy the specified condition. The deletion operation violates referential integrity consraints. Example: Delete The *Employee* Tuple With *Name='John'*

3. UPDATE OPERATION:

The update operation is used to change the values of one or more attributes in a tuple of some relation R. it is necessary to specify a condition on the attributes of the relation to select the tuple to be modified.

Example: Update The *Salary* Of The *Employee* Tuple With *SSN*= '*99988777'* To *28000.*

BASIC RELATIONAL ALGEBRA OPERATIONS:

A basic set of relational model operations constitutes the relational algebra. These operations enable the user to specify basic retrieval requests. A sequence of relational algebra operations forms a relational algebra expression.

Relational algebra includes operations like

1. SELECT
2. PROJECT
3. RENAME
4. SET OPERATIONS LIKE
   a) UNION
   b) NTERSECTION
   c) SET DIFFERENCE
5. JOIN
6. DIVISION

1.SELECT OPERATION:

The SELECT operation is used to select a subset of the tuples from a relation thatsatisfy a selection condition. I.e. it selects some of the rows from the table.

Syntax: □<selection condition> (R)

Where □ (sigma) symbol is used to specify the SELECT operator and the selection condition is a Boolean expression specified on the attributes of relation R. the result is also a relation with the same attributes of R. the selection condition is of the form

<Attribute name> <comparison op> <attribute name/constant>

Comparison operator includes {=, <, >, , □ □, □}.

Example 1: Select the employees with dno 4

Ans.: □dno=4 (EMPOYEE) **Result:**

NAME	SSN	BDATE	ADDRESS	SEX	SALARY	DNO
Mani	384	29-12-78	Chennai	M	17000	4

Example 2: List the employees whose salary is greater than 18000

Ans.: □SALARY>18000 (EMPOYEE) **Result:**

NAME	SSN	BDATE	ADDRESS	SEX	SALARY	DNO
Alex	123	12-07-76	Chennai	M	20000	1

More than one conditions can be connected by Boolean operators AND, OR, NOT to form a general selection condition. Example: consider the following query.

Example 3: Select the employees who either work in dno 3 and receives more than30000 or work in dno 4 and receives more than 15000.

Ans.: □(dno=3 AND salary>30000) OR (dno=4 AND salary >15000) (EMPOYEE) Result:

NAME	SSN	BDATE	ADDRESS	SEX	SALARY	DNO
Mani	384	29-12-78	Chennai	M	17000	4

2. PROJECT OPERATION:

The PROJECT operation is used to select certain set of attributes from a relation. I.e.it selects some of the columns from the table.

Syntax: □ <attribute list> (R)

Where □ (pi) symbol is used to specify the PROJECT operator and the attribute list isthe list of attributes of relation R. the result is also a relation with the selected set of attributes of R.

Example 4: List each employee's name and salary.

Ans.: □ name, salary (EMPOYEE)

Result:

NAME	SALARY
Alex	20000
Siva	15000
Sruthi	17000
Gayathri	14000
Mani	17000

Nesting of SELECT, PROJECT operators are allowed in a relational algebra expression.**Example 5:** retrieve the name and salary of all employees who works in department number 1.

Ans.: □ name, salary□(dno=1 (EMPOYEE))

NAME	SALARY
Alex	20000
Gayathri	14000

It is often simpler to break down complex sequence of operations by specifying intermediate result relations than to write a single relational algebra expression.
RENAME operator allows storing the intermediate results in a new relation.Example 5 can be expressed as following

Dep1_emp□ □dno=1 (EMPOYEE) **Result:** relation: dep1_emp

NAME	SSN	BDATE	ADDRESS	SEX	SALARY	DNO
Alex	123	12-07-76	Chennai	M	20000	1
Gayathri	406	27-01-79	Chennai	F	14000	1

Result □□ name, salary (dep1_emp)

NAME	SALARY
Alex	20000
Gayathri	14000

Dep1_emp, Result are temporary relations created automatically.

3. RENAME OPERATION:

The RENAME operation can rename either the relation name or the attribute namesor both.

Syntax: □S (B1, B2…. B n) (R)

Where the symbol □ denote the RENAME operator, S denote the new relation, B1, B2 are new attribute names.

Example 6: change the name of the EMPLOYEE relation as STAFF LIST

Ans.: □STAFFLIST (EMPLOYEE)

Result: RELATION: STAFFLIST

NAME	SSN	BDATE	ADDRESS	SEX	SALARY	DNO
Alex	123	12-07-76	Chennai	M	20000	1
Siva	407	03-06-80	Bangalore	M	15000	2
Sruthi	905	01-12-78	Hyderabad	F	17000	3
Gayathri	406	27-01-79	Chennai	F	14000	1
Mani	384	29-12-78	Chennai	M	17000	4

Example 7: change the name of the column header SSN as EMPCODE

Ans.: □(NAME, EMPCODE, BDATE, ADDRESS, SEX, SALARY, DNO) (EMPLOYEE)

Result: RELATION: EMPLOYEE

NAME	EMPCODE	BDATE	ADDRESS	SEX	SALARY	DNO
Alex	123	12-07-76	Chennai	M	20000	1
Siva	407	03-06-80	Bangalore	M	15000	2
Sruthi	905	01-12-78	Hyderabad	F	17000	3
Gayathri	406	27-01-79	Chennai	F	14000	1
Mani	384	29-12-78	Chennai	M	17000	4

4. SET OPERATIONS:

Set operations are used to merge the elements of two sets in various ways including UNION, INTERSECTION, and SET DIFFERENCE. The three operations UNION, INTERSECTION, and SET DIFFERENCE can be applied on two union compatible relations. What is a **union compatible relation**? Two relations are said to beunion compatible relation if they have same degree n (same no of attributes) and that each pair of corresponding attributes have the same domain. Consider two union compatible relations.

STUDENT	NAM
	SUS
	RAM
	JOH
	JIMM
	BAR
	FRA
	ERN

TEACHER	NAME
	SUSAN
	JIMMY
	MAYA
	PRIYA
	PATRICK
	RAMESH

UNION:

R ∪ S where R, S are two relations. The result is also a relation that includes all tuplesthat are either in R or in S or in both R and S.

INTERSECTION:

R ∩ S where R, S are two relations. The result is also a relation that includes all tuplesthat are in both R and S.

SET DIFFERENCE:

R – S The result is also a relation that includes all tuples that are in R but not in S.

Results:

R S	Name
	SUSA
	RAM
	JOHN
	JIMM
	BARE
	FRAN
	ERNE
	MAY
	PRIY
	PATR

R S	Nam
	SUS
	RAM
	JIM

R - S	Name
	ERNEST
	FRANCIS
	JOHNNY
	BARBARA

4. CARTISIAN PRODUCT:

It is also known as CROSS JOIN or CROSS PRODUCT and denoted by X. like set operations cross product is also used merge two relations but the relations on which it isapplied need not be union compatible.

Let us consider the relational algebra expression. **Q R X S.**

Where R is a relation with n attributes (A1, A2.An) and S is a relation with m attributes (B1, B2.Bn) and the resulting relation Q with n+m attributes (A1, A2, An, B1, B2, Bn) in that order. The resulting relation Q has one tuple for each combination of tuples one from R and one from S. hence if R has t1 tuples and S has t2 tuples then R X S t1* t2 tuples.

Consider the RELATION: EMPLOYEE

NAME	SSN	BDATE	ADDRESS	SEX	SALARY	DNO
Alex	123	12-07-76	Chennai	M	20000	1
Siva	407	03-06-80	Bangalore	M	15000	3
Sruthi	905	01-12-78	Hyderabad	F	17000	3
Gayathri	406	27-01-79	Chennai	F	14000	1
Mani	384	29-12-78	Chennai	M	17000	4

And the RELATION: DEPARTMENT

DNAME	DNO	MGRSSN
Research	1	123
Accounts	3	905
administration	4	384

Example 8: Retrieve the manager name for each department

The MGRSSN (manager's social security number) is present in the department relation and name of the manager is present in the employee relation. MGRSSN is the foreign key of department relation and SSN is the primary key of employee relation.

Ans.:

Temp1 employee X department

RELATION: TEMP 1

NAME	SSN	…	…	…	…	…	DNAME	DNO	MGRSSN
Alex	123						Research	1	123
Alex	123						Accounts	3	905
Alex	123						administration	4	384
Siva	407						Research	1	123
Siva	407						Accounts	3	905
Siva	407						administration	4	384
Sruthi	905						Research	1	123
Sruthi	905						Accounts	3	905

Sruthi	905						administration	4	384
Gayathri	406						Research	1	123
Gayathri	406						Accounts	3	905
Gayathri	406						administration	4	384
Mani	384						Research	1	123
Mani	384						Accounts	3	905
Mani	384						administration	4	384

The CARTESIAN product creates tuples with the combined attributes of two relations the operation applied by itself is generally meaning less. It is useful when it is followed by a SELECT operation that selects only related tuples from two relations according to the selection condition. Temp2 ssn = mgrssn (temp1)

RELATION: TEMP 2

NAME	**SSN**	**…**	**…**	**…**	**…**	**…**	**DNAME**	**DNO**	**MGRSSN**
Alex	123						Research	1	123
Sruthi	905						Accounts	3	905
Mani	384						administration	4	384

Managerlist dname, name

(temp2)RELATION: MANAGERLIST

DNAME	**NAME**
Research	Alex
Accounts	Sruthi
administration	Mani

6. JOIN OPERATION:

The CARTESIAN product followed by SELECT is commonly used to select related tuples from two relations. Hence a special operation called JOIN was created to specify this sequence as a single operation. Join operation is denotedby **Q R <join condition> S**

Where R (A1, A2, .An), S (B1, B2, Bn) are two relations with degree m, n respectively and Q is the resulting relation with m + n attributes, has one tuple for each combination of tuples one from R and one from S –whenever the combination satisfies the join condition. This is the main difference between CARTESIAN PRODUCT and JOIN: in JOIN, only combinations of tuples satisfying the join condition appear in the result, whereas in the CARTESIAN PRODUCT all combinations of tuples are included in the result.

The JOIN operation with such a general join condition is called **THETA JOIN**. If the comparison operator used is = then that join is called **EQUI JOIN.**

Example 8 can be performed using join operation as following

Temp 1 ← EMPLOYEE SSN=DMGRSSN DEPARTMENT

Managerlist ← dname, name (temp1)

NATURAL JOIN:

Natural join requires that the two join attributes have the same name in both therelations. If this is not the case, a renaming operation is applied first

Q ← R * S

Where * denotes the natural join operation, the relations R and S must have an attribute with same name. If there are more than one attribute pair with same name the NATURAL JOIN operation is performed by equating all attribute pairs that have the same name in both relations.

Example 9: Retrieve the name of the department that GAYATHRI works: GAYATHRI belongs to DNO 1 and the DNAME information is present in DEPARTMENT relation. The foreign key DNO is used to find out these related details. Here both primary key and foreign key are having same name so you can apply naturaljoin

Temp1 ← DEPARTMENT * EMPLOYEE

This is equivalent of applying

Temp 1 ← EMPLOYEE employee.dno=department.dno DEPARTMENT Gayathri_dept

← dname (NAME = ”GAYATHRI” (Temp 1))

Left Outer Join

Use this when you only want to return rows that have matching data in the left table, even ifthere's no matching rows in the right table.

```
SELECT * FROM Individual AS Ind,Publisher AS Pub
WHERE Ind.IndividualId(+)= Pub.IndividualId
```

Example SQL statement

Source Tables

Left Table-Individual

Id	FirstName	LastName	UserName
1	**Fred**	**Flinstone**	**freddo**

2	**Homer**	**Simpson**	**homey**
3	**Homer**	**Brown**	**notsofamous**
4	**Ozzy**	**Ozzbourne**	**sabbath**
5	**Homer**	**Gain**	**noplacelike**

Right Table-Publisher

IndividualId	AccessLevel
1	**Administrator**
2	**Contributor**
3	**Contributor**
4	**Contributor**
10	**Administrator**

Result

IndividualId	FirstName	LastName	UserName	IndividualId	AccessLevel
1	**Fred**	**Flinstone**	**freddo**	**1**	**Administrator**
2	**Homer**	**Simpson**	**homey**	**2**	**Contributor**
3	**Homer**	**Brown**	**notsofamous**	**3**	**Contributor**

4	**Ozzy**	**Osbourne sabbath**	**4**	**Contributor**
5	**Homer**	**Gain noplacelike NULL**	**NULL**	

Right Outer Join

Use this when you only want to return rows that have matching data in the righttable, even if there's no matching rows in the left table.

```
SELECT * FROM Individual AS Ind, Publisher AS Pub
WHERE Ind.IndividualId = Pub.IndividualId(+)
```

Example SQL statement

Source Tables

Left Table- Individual

Id	FirstName	LastName	UserName
1	**Fred**	**Flinstone**	**freddo**
2	**Homer**	**Simpson**	**homey**
3	**Homer**	**Brown**	**notsofamous**
4	**Ozzy**	**Ozzbourne**	**sabbath**
5	**Homer**	**Gain**	**noplacelike**

Right Table - Publisher

IndividualId	AccessLevel
1	**Administrator**
2	**Contributor**

3	**Contributor**
4	**Contributor**
10	**Administrator**

Result

IndividualId	FirstName	LastName	UserName	IndividualId	AccessLevel
1	**Fred**	**Flinstone**	**freddo**	**1**	**Administrator**
2	**Homer**	**Simpson**	**homey**	**2**	**Contributor**
3	**Homer**	**Brown**	**notsofamous**	**3**	**Contributor**
4	**Ozzy**	**Osbourne**	**sabbath**	**4**	**Contributor**
NULL	**NULL**	**NULL**	**NULL**	**10**	**Administrator**

Full Outer Join

Use this when you want to all rows, even if there's no matching rows in the righttable.

```
SELECT * FROM Individual AS Ind,Publisher AS Pub WHERE
Ind.IndividualId(+) = Pub.IndividualId(+)
```

Example SQL statement

Source Tables

Left Table- Individual

Id	FirstName	LastName	UserName
1	**Fred**	**Flinstone**	**freddo**
2	**Homer**	**Simpson**	**homey**
3	**Homer**	**Brown**	**notsofamous**

4	**Ozzy**	**Ozzbourne**	**sabbath**
5	**Homer**	**Gain**	**noplacelike**

Right Table- Publisher

IndividualId	AccessLevel
1	**Administrator**
2	**Contributor**
3	**Contributor**
4	**Contributor**
10	**Administrator**

Result

IndividualId FirstName LastName UserName IndividualId AccessLevel

1 Fred Flinstone freddo 1 Administrator

2 Homer Simpson homey 2 Contributor

3 Homer Brown notsofamous 3 Contributor

4 Ozzy Osbourne sabbath 4 Contributor

5 Homer Gain noplacelike NULL NULL

NULL NULL NULL NULL 10 Administrator

7. AGGREGATE FUNCTIONS AND GROUPING:

Aggregate functions are applied to collection of numeric values include SUM, AVERAGE, MINIMUM, and MAXIMUM. The COUNT function is used for counting tuples or values. Grouping is used to group the tuples in a relation by the value ofsome of their attributes.

VIEWS IN SQL

SQL is a standard language for storing, manipulating and retrieving data in databases.SQL stands for Structured Query Language.It is designed for managing data in a relational database management system (RDBMS).

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 ofthese joins. We then issue queries on the view.

Specification of Views in SQL:

The command 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, anda 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-ON

EMP_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 onwhich 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.

For choosing the most likely update:

- A view with a single defining table is updatable if the view attributes contain the primary key of the base relation, because this maps each view tuple to a single base relation.
- Views defined on multiple tables using joins are generally not updateable.
- Views defined using grouping and aggregate functions are not updatable.

THE TUPLE RELATIONAL CALCULUS

- Relational calculus is a formal query language, where to specify a retrieval request.
- A calculus expression specifies what is to be retrieved rather than hoe to retrieved id.
- There fore, the relational calculus is considered to be a non procedural language.
- This differs from relational algebra, where we must write a sequence of operations to specify a retrieval request.
- Relational calculus is considered as a procedural way of stating a query.
- Expressive power of the 2 languages is identical.

Tuple variable & Range Relation:

- Tuple Relational calculus is based on specifying a number of tuple variables.
- Each tuple variable usually ranges over a particular database relation.
- That is the variable may take as its value any individual tuple from that

relation.

A simple tuple relational calculus query is of the form:

{ t | COND (t) }

where, t – is a tuple variable & COND(t)- includes Range relation of tuple and selective condition.

The result of such a query is the set of all tuples t that satisfy COND(t) Example:Q1. To find all employees whose salary is above 50,000.

Tuple relational Expression:

{ t | Employee(t) and t. salary > 50000}

Q2. To retrieve only some of the attributes:

{ t. fname, t. address | Employee(t) and t. salary > 50000} Information

specified in a tuple calculus expression:

- For each tuple variable t, the range relation R of t. this value is specified by a condition of the form R(t).
- A COND to select particular combinations of tuples. A set of attributes to be retrieved, the requested attributes.

Expressions & Formulas in Tuple Relational Calculus:

A general Expression of the Tuple relational calculus

{ t1.A1, t2. A2, ….., tn. An | COND(t1,t2,…..,tn, tn+1,…..,tn+m) }

t1.A1, t2. A2, ….., tn. An are tuple variables, each Ai is an attribute of the relationon which ti ranges.

- COND is made up of atoms is of the form ti . Ai OP tj . Bj or ti . Ai OP Cor C OP tj . Bj

- Where OP is one of the comparison operators is the set of { =,<,<=,>,>=,!=}and C is constant.

A Formula is made up of one or more atoms connected via the logical operatorsand, or ,not.

Example formulas: F1AND F2, F1 OR F2 , NOT F1

The Existential & Universal Quantifiers:

- In a formula two additional symbols called Quantifiers: Existential( ), universal ( )
- A tuple variable t is BOUND if it is quantified or it is FREE. Example:

F1: d. dname =’Research’

F2: (d)(d.MGESSN = 987654321)

F3: (t) (t.dname = t.dno)

The tuple variable d is FREE in F1 & F3, d is BOUND in F2. The variable t isBOUND in F3.

- If F is a formula, then so is (t) (F), where t is a tuple variable. The formula(t) (F) is true if the formula F evaluates to TRUE for some (at least one) tuple assigned of t in F, otherwise (t) (F) is false.

- If F is a formula, then so is (t) (F), where t is a tuple variable. The formula(t) (F) is true if the formula F evaluates to TRUE for ever tuple assigned of t in F, otherwise (t) (F) is false.

Q3: Retrieve the name , address of all employees who work for the research department.

{ t. fname, t. address | Employee(t) and (d)(Department (d) and d.dname ='Research' and d.Dnumber = t. DNO) }

Safe Expression:

- The expression in relational calculus is the one which produce finite tuplesas its result, then that expression is known as Safe Expression or it is known as Unsafe Expression.
- An expression is safe if all values in its result are from the domain of the expression.
- Example: { t | not (Employee(t) }

It is unsafe, it produce all tuples in the universe that are not employee tuples,which are infinite.

THE DOMAIN RELATIONAL CALCULUS

- The domain relational calculus differs from the tuple relational calculus inthe type of variables used in formula.
- Rather than having domain variables range over tuples, the domain variables ranges over the domain of attributes.

An expression of the Domain calculus is of the form: { x1,x2,…..xn | COND (x1, x2, ….xn, xn+1, …..xn+m)} x1,x2,…..xn are domain variables that range over domain(attributes)

Q1: Retrieve the birthdate address of all employees.

{ tu | employee(pqrstuvwxy)}

Ten variables for the employee relation, one to range over the domain ofeach attribute in order.

Q2: Retrieve the name of all employees whose salary is >35 000

{ p | employee(pqrstuvwxy) and w>35000 }

Q3: Retrieve the all employees ssn who are all working in project number . {a | project (abc) and b=1}

QBE LANGUAGE

- Query By Example (QBE) language, it is one of the first graphical language with minimum syntax developed for database systems.
- It was Developed by IBM
- It uses Point and Click type Interface.
- It differs from SQL, in that the does not have to specify a structured query explicitly; rather, the query is formulated by filling in templates ofrelations that are displayed on monitor screen.
- The user does not have to remember the names of attributes or relation, because they are displayed as part of these templates.
- The user does not have to follow the syntax rules.

Basic Retrieval Queries in QBE:

- The retrieval query is formulated by filling in one or more rows in the templates of the tables.
- Column value which we want to print can be specified with P. operator.

Q1: Retrieve the birthdate, address of all employees.

EMPLOYEE	FNAME	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY
				P.	P.		

- In QBE, the user interface first allow the user to choose the tables needed to formulate a query by displaying a list of all relation namesThe templates for the chosen relations are then displayed.
- The user moves to the appropriate column in the templates and specifies the query.
- Special function keys are provided to move among templates & perform certain functions
- To specify the condition, the user can use CONDITION BOX, which is created by pressing a particular function key.

Q2: List the SSN of employees who work on project no 1

WORKS-ON	ESSN	PNO	HOURS
	p.	1	

Q3: List the SSN of employees who work more than 20 hrs per week on projectno 1

WORKS-ON	ESSN	PNO	HOURS
	p.	__X	__HY

CONDITION BOX

_X = 1 AND_HY >20

Q4: List the name and address of all employees who work for the research department

EMPLOYEE	FNAME	LNAME	SSN	BDATE	ADDRESS	SALARY	DNO
	p.				p.		_DX

DEPARTMENT	DNAME	DNUMBER	MGRSSN	MGRSARTDATE
	_DN	_DY		

CONDITION BOX

_DN='Research' AND _DY=_DX

Grouping, Aggregation, & Database Modification in QBE:

Aggregate functions:
AVG. , SUM. , CNT. , MAX. , MIN. ,

- In QBE the functions AVG. , SUM. , CNT. , MAX. , MIN. are applied to distinct values within a group in the default case.
- If we want these functions to apply to all values, we must use the prefixALL
- Grouping is indicated by G. function

Q5: count the no. of distinct salary values in the employee relation.

EMPLOYEE	FNAME	LNAME	SSN	BDATE	ADDRESS	SALARY	DNO
						P. CNT.	

Q6: count the all salary values in the employee relation.

EMPLOYEE	FNAME	LNAME	SSN	BDATE	ADDRESS	SALARY	DNO
						P. CNT. ALL	

Q7: Retrieve each department no, the no of employees within each department.

EMPLOYEE	FNAME	LNAME	SSN	BDATE	ADDRESS	SALARY	DNO
			P.CNT.ALL				P.G.

Dno column is used for grouping as indicated by G. function.

Q8: Display each project name & the no. of employees working on it for project onwhich more than two employee work.

WORKS-ON	ESSN	PNO	**HOURS**
	P.CNT._EX	G._PX	

PROJECT	PNAME	PNO	**PLOCATION**	**DNUM**
	P.	_PX		

CONDITION BOX

CNT._EX>2

Modifying the Database:

There are three QBE operators for modifying the database:

I. for Insert, D. for Delete, U. for Update

- The insert and delete operators are specified in the template column under the relation name.
- Whereas the update operator is specified under the column to beupdated.

To insert new tuple:

PROJECT	PNAME	PNO	**PLOCATION**	**DNUM**

I.	ProductA	45	Katy, tx	**4**

To Delete , specify D. operator & then condition for the tuple being deleted.

Q9. Delete employee details whose ssn number 987654321

EMPLOYEE	FNAME	LNAME	**SSN**	BDATE	ADDRESS	SALARY	DNO
D.			**987654321**				

To update a tuple, we specify U. operator under the attribute name, followed by anew value of the attribute.

Q10: Increase the salary of John B smith by 500 and also reassign him todepartment 4

EMPLOYEE	FNAME	LNAME	**SSN**	BDATE	ADDRESS	SALARY	DNO
	John	B	**Smith**			U. _sal=_sal+500	U.4

QBE was the first user friendly “visual” relational database language

SCHOOL OF COMPUTING

DEPARTMENT OF COMPUTER SCIENCE

UNIT III -DATABASE MANAGEMENT SYSTEM- SBS1205

UNIT III

Database Design, Theory and Methodology: Functional dependencies and normalization for relational database – Relational database design algorithms and further dependencies – Practical database design and tuning.

Database Design

- what does one entity do with the other, how do they relate to each other?
- For example, customers buy products, products are sold to customers, a sale comprises products, a sale happens in a shop.
- Identifying Entities
- Identifying Relationships

Figure 3.1 Entities type of information

- Database Design is a collection of processes that facilitate the designing, development, implementation and maintenance of enterprise data management systems.
- database are easy to maintain, improves data consistency and are cost effective in terms of disk storage space.
- is the organization of data according to a database model. The designer determines what data must be stored and how the data elements interrelate. With this information, they can begin to fit the data to the database model.

Functional Dependency

- Functional Dependency (FD**)** determines the relation of one attribute to another attribute in a database management system (DBMS) system.
- Functional dependency helps you to maintain the quality of data in the database.
- a functional dependency is a constraint between two sets of attributes in a relation from a database.

- The determination of functional dependencies is an important part of designing databases in the relational model,
- Functional dependency is a relationship that exists when one attribute uniquely determines another attribute.
- If R is a relation with attributes X and Y, a functional dependency between the attributes is represented as X->Y, which specifies Y is functionally dependent on X.
- Here X is a determinant set and Y is a dependent attribute.
- Each value of X is associated with precisely one Y value.
- A functional dependency is denoted by an arrow →.
- The functional dependency of X on Y is represented by X → Y.
- The functional dependency is a relationship that exists between two attributes. It typically exists between the primary key and non-key attribute within a table.

  X → Y
- The left side of FD is known as a determinant, the right side of the production is known as a dependent.
- A functional dependency A->B in a relation holds if two tuples having same value of attribute A also have same value for attribute B. For Example, in relation STUDENT shown in table 1, Functional Dependencies
- STUD_NO->STUD_NAME, STUD_NO->STUD_PHONE **hold**
- but
- STUD_NAME->STUD_ADDR **do not hold**

STUDENT

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

Figure 3.2 Student Database

- **How to find functional dependencies for a relation?**
- Functional Dependencies in a relation are dependent on the domain of the relation. Consider the STUDENT relation given in Table 1.
- We know that STUD_NO is unique for each student. So STUD_NO->STUD_NAME, STUD_NO->STUD_PHONE, STUD_NO->STUD_STATE, STUD_NO->STUD_COUNTRY and STUD_NO -> STUD_AGE all will be true.

- Similarly, STUD_STATE->STUD_COUNTRY will be true as if two records have same STUD_STATE, they will have same STUD_COUNTRY as well.

key terms for functional dependency

Key Terms	Description
Axiom	Axioms is a set of inference rules used to infer all the functional dependencies on a relational database.
Decomposition	It is a rule that suggests if you have a table that appears to contain two entities which are determined by the same primary key then you should consider breaking them up into two different tables.
Dependent	It is displayed on the right side of the functional dependency diagram.
Determinant	It is displayed on the left side of the functional dependency Diagram.
Union	It suggests that if two tables are separate, and the PK is the same, you should consider putting them. together

Rules of Functional Dependencies

- Reflexive rule –. If X is a set of attributes and Y is_subset_of X, then X holds a value of Y.
- Augmentation rule: When x -> y holds, and c is attribute set, then ac -> bc also holds. That is adding attributes which do not change the basic dependencies.
- Transitivity rule: This rule is very much similar to the transitive rule in algebra if x -> y holds and y -> z holds, then x -> z also holds. X -> y is called as functionally that determines y.

Functional Dependencies –Example

Roll No	Name	Marks	Dept	course
100	ram	78	cse	c1
101	ravi	60	ece	c1
102	ram	78	cse	c2
103	ravi	60	eee	c3
104	sam	80	IT	c3
105	john	80	ece	c2

FD: X→Y IF T1.X= T2.X THEN T1.Y = T2.Y

- ROLL NO→ NAME
- YES
- NAME → ROLL NO
- NO
- ROLL NO → MARKS
- YES
- DEPT → COURSE
- NO
- MARKS → DEPT
- NO
- (ROLL NO, NAME) → MARKS
- YES

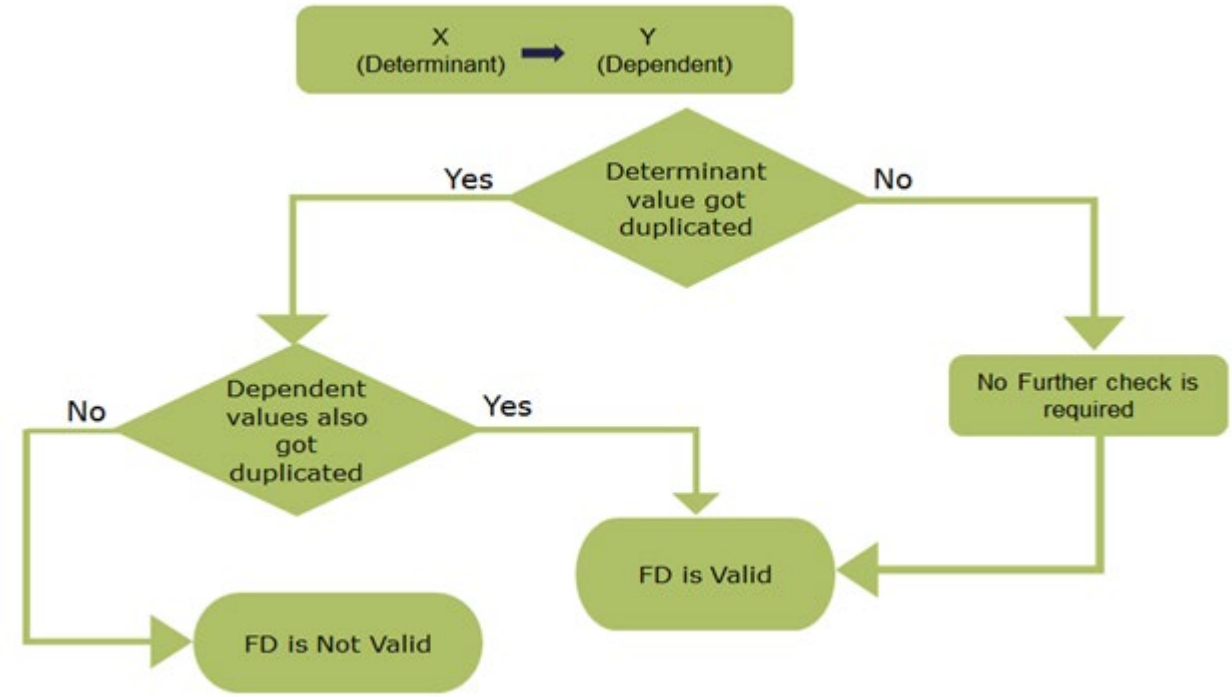


Figure 3.3 Functional Dependencies

Functional Dependencies –Example

A	B	C
1	2	3
2	2	3
3	2	3
4	3	2
5	2	3
6	3	2

- All possible FD's of the above relation:
- **A->A** : **Valid** (Determinant A's values are unique)
- **A->B** : **Valid** (Determinant A's values are unique)
- **A->C** : **Valid** (Determinant A's values are unique)
- **B->A** : **Not Valid** (for same determinant B's value '2', dependent A's value '1','2′ is different)
- **B->B** : **Valid** (Trivial Dependency always valid according to Reflexive Axiom)
- **B->C** : **Valid** (for same determinant B value, same dependent C value, for different B value C value could be anything hence it is valid).

- **C->A** : **Not Valid** (for same determinant C value 3 there are 5 different dependent A values hence not valid)
- **C->B** : **Valid** (for same determinant C value that is 3, same dependent B value that is 2, for different C value B value could be anything hence it is valid)
- **C->C** : **Valid** (Trivial Dependency always valid according to Reflexive Axiom)
- **AB->A** : **Valid** (A's values are unique hence determinant AB's values will also be unique, hence dependent A's value could be anything, AB->any attribute of above relation will be valid)
- **AB->B** : **Valid** (A's values are unique hence determinant AB's values will also be unique, hence dependent B's value could be anything, AB->any attribute of above relation will be valid)
- **AB->C** : **Valid** (A's values are unique hence determinant AB's values will also be unique, hence dependent C's value could be anything, AB->any attribute or attributes combination of above relation will be valid)
- **BC->A** : **Not Valid** (for same value {2,3} of determinant BC there are two values {1},{2}of dependent A, hence not valid)
- **BC->B** : **Valid** (for same value {2,3} o determinant BC, same value {2} of dependent B exist. this holds for any combination of determinant to dependent. hence it is valid)
- **BC->C** : **Valid** (for same value {2,3} o determinant BC, same value {3} of dependent C exist. this holds for any combination of determinant to dependent. hence it is valid)
- **CA->A** : **Valid** (A's values are unique hence determinant AC's or CA's values will also be unique, hence dependent A's value could be anything, CA->any attribute or attributes combination of above relation will be valid)
- **CA->B** : **Valid** (A's values are unique hence determinant AC's or CA's values will also be unique, hence dependent B's value could be anything, CA->any attribute or attributes combination of above relation will be valid)
- **CA->C** : **Valid** (A's values are unique hence determinant AC's or CA's values will also be unique, hence dependent C's value could be anything, CA->any attribute or attributes combination of above relation will be valid)
- **ABC->A** : **Valid** (A's values are unique hence determinant ABC's values will also be unique, hence dependent C's value could be anything, ABC->any attribute or attributes combination of above relation will be valid)
- **ABC->B** : **Valid** (A's values are unique hence determinant ABC's values will also be unique, hence dependent B's value could be anything, ABC->any attribute or attributes combination of above relation will be valid)

- **ABC->C** : **Valid** (A's values are unique hence determinant ABC's values will also be unique, hence dependent C's value could be anything, ABC->any attribute or attributes combination of above relation will be valid)

Types of Functional Dependencies

- Multivalued dependency:
- Trivial functional dependency:
- Non-trivial functional dependency:
- Transitive dependency:
- **Trivial Functional Dependencies-**
- A functional dependency X → Y is said to be trivial if and only if Y ⊆ X.
- Thus, if RHS of a functional dependency is a subset of LHS, then it is called as a trivial functional dependency.
- Examples-
- The examples of trivial functional dependencies are-
- AB → A
- AB → B
- AB → AB
- The Trivial dependency is a set of attributes which are called a trivial if the set of attributes are included in that attribute.
- So, X -> Y is a trivial functional dependency if Y is a subset of X.
- Consider this table with two columns Emp_id and Emp_name.
- {Emp_id, Emp_name} -> Emp_id is a trivial functional dependency as Emp_id is a subset
- of {Emp_id,Emp_name}.

Emp_id	Emp_name
AS555	Harry
AS811	George
AS999	Kevin

Non trivial functional dependency

- A → B has a non-trivial functional dependency if B is not a subset of A.

- When A intersection B is NULL, then A → B is called as complete non-trivial.
- A functional dependency X → Y is said to be non-trivial if and only if Y ⊄ X.
- Thus, if there exists at least one attribute in the RHS of a functional dependency that is not a part of LHS, then it is called as a non-trivial functional dependency.
- AB → BC
- AB → CD
- AB→ C , AB→AC,AC→B

Multivalued dependency

- Multivalued dependency occurs in the situation where there are multiple independent multivalued attributes in a single table

Car_model	M_year	Color
H0012017MetallicH	2017	Green
H0052018Metallic	2018	Blue

- In this example, maf_year and color are independent of each other but dependent on car_model. In this example, these two columns are said to be multivalue dependent on car_model.
- This dependence can be represented like this:
- car_model -> maf_year
- car_model-> colour

Transitive functional dependency

A transitive is a type of functional dependency which happens when t is indirectly formed by two functional dependencies.

Normalization

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

- Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate the undesirable characteristics like Insertion, Update and Deletion Anomalies.
- Normalization divides the larger table into the smaller table and links them using relationship.
- The normal form is used to reduce redundancy from the database table.
- The inventor of the relational model Edgar Codd proposed the theory of normalization with the introduction of the First Normal Form, and he continued to extend theory with Second and Third Normal Form.
- Later he joined Raymond F. Boyce to develop the theory of Boyce-Codd Normal Form.
- When developing the schema of a relational database, one of the most important aspects to be taken into account is to ensure that the duplication is minimized. This is done for 2 purposes:
- Reducing the amount of storage needed to store the data.
- Avoiding unnecessary data conflicts that may creep in because of multiple copies of the same data getting stored.
- Database Normalization is a technique that helps in designing the schema of the database in an optimal manner so as to ensure the above points.
- The core idea of database normalization is to divide the tables into smaller subtables and store pointers to data rather than replicating it.
- Database Normalization is a technique of organizing the data in the database. Normalization is a systematic approach of decomposing tables to eliminate data redundancy(repetition) and undesirable characteristics like Insertion, Update and Deletion Anomalies. It is a multi-step process that puts data into tabular form, removing duplicated data from the relation tables.
- Normalization is used for mainly two purposes,
- Eliminating redundant(useless) data.
- Ensuring data dependencies make sense i.e data is logically stored.

Anomalies

- "What does anomaly mean?". The dictionary defines anomaly as “an abnormality.”
- Ideally, a field value change should be made in only a single place.
- Data redundancy, however, fosters an abnormal condition by forcing field value changes in many different locations.

- Anomalies in DBMS develops when not all of the required changes in the redundant data are made successfully.
- Anomalies are problems that can occur in poorly planned, un-normalised databases where all the data is stored in one table (a flat-file database).
- Normalization is a technique for producing relational schema with the following properties.
- No anomalies
- No information redundancy
- So we are clear when the process of reducing the anomaly comes into action.
- There are three types of anomalies: update, deletion and insertion anomalies.
- **Update Anomaly**: Let say we have 6 columns in a table out of which 2 are called student name and student contact number. Now if one student changes it's contact number then we would have to update the table. But the problem is, if the table is not normalized one student can have multiple entries and while updating all of those entries one of them might get missed.

Dept

Roll No	Name	contact Nmuber	Dept	course
100	ram	9080090800	cse	c1
101	ravi	9080090801	ece	c1
102	ram	9080090802	cse	c2
103	ravi	9080090803	eee	c3
104	sam	9080090804	IT	c3
105	john	9080090805	ece	c2
102	ram	9080090802	cse	c3

COE

Roll No	Name	contact Nmuber	Dept	course
100	ram	9080090800	cse	c1
101	ravi	9080090801	ece	c1
102	ram	9080090802	cse	c2
103	ravi	9080090803	eee	c3
104	sam	9080090804	IT	c3
105	john	9080090805	ece	c2
102	ram	9080090802	cse	c3

- **Insert Anomaly**: Let say we have 6 columns in a table out of which 2 are called student name and student contact number. Now if one student details we would have to insert in the table. But the problem is, if the table is not normalized one student can have multiple entries and while inserting all of those entries one of them might get missed.

Dept

Roll No	Name	contact Nmuber	Dept	course
100	ram	9080090800	cse	c1
101	ravi	9080090801	ece	c1
102	ram	9080090802	cse	c2
103	ravi	9080090803	eee	c3
104	sam	9080090804	IT	c3
105	john	9080090805	ece	c2
102	ram	9080090802	cse	c3

COE

Roll No	Name	contact Nmuber	Dept	course
100	ram	9080090800	cse	c1
101	ravi	9080090801	ece	c1
102	ram	9080090802	cse	c2
103	ravi	9080090803	eee	c3
104	sam	9080090804	IT	c3
105	john	9080090805	ece	c2
102	ram	9080090802	cse	c3

- **Delete Anomaly**: Let say we have 6 columns in a table out of which 2 are called student name and student contact number. Now if one student details we would have to delete in the table. But the problem is, if the table is not normalized one student can have multiple entries and while deleting all of those entries one of them might get missed.

Dept

Roll No	Name	contact Nmuber	Dept	course
100	ram	9080090800	cse	c1
101	ravi	9080090801	ece	c1
102	ram	9080090802	cse	c2
103	ravi	9080090803	eee	c3
104	sam	9080090804	IT	c3
105	john	9080090805	ece	c2
102	ram	9080090802	cse	c3

COE

Roll No	Name	contact Nmuber	Dept	course
100	ram	9080090800	cse	c1
101	ravi	9080090801	ece	c1
102	ram	9080090802	cse	c2
103	ravi	9080090803	eee	c3
104	sam	9080090804	IT	c3
105	john	9080090805	ece	c2
102	ram	9080090802	cse	c3

Types of normalization

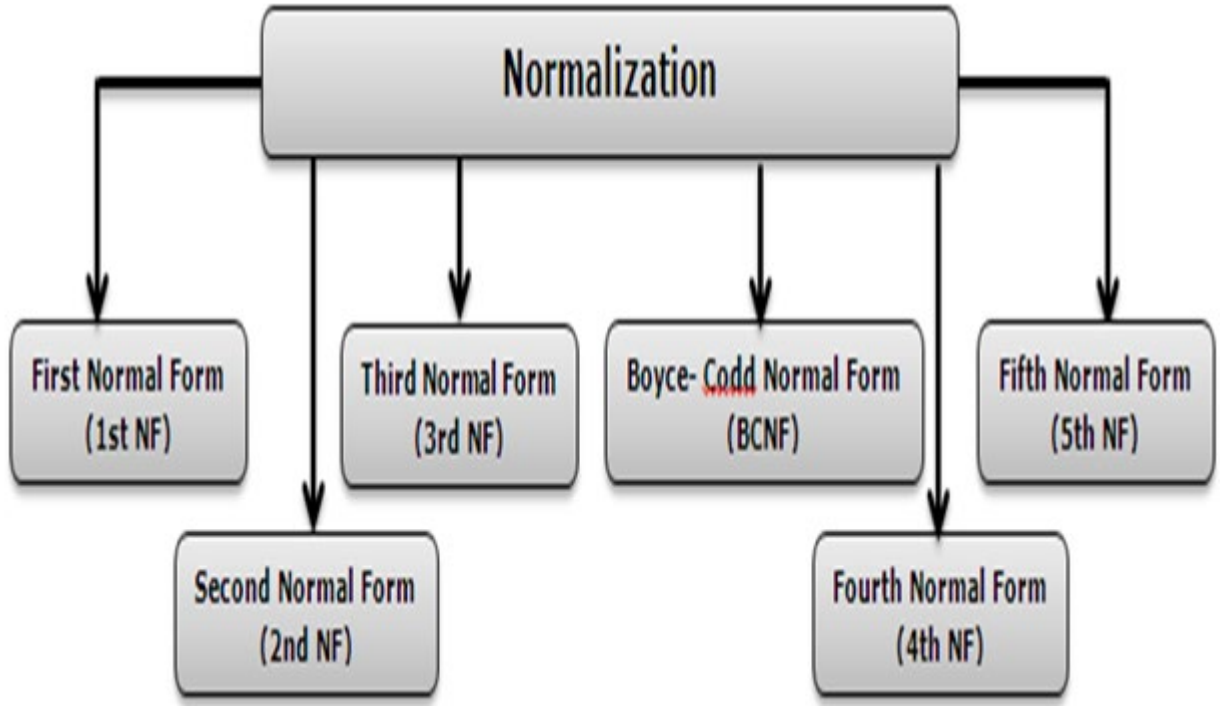


Figure 3.4 Types of normalization

First Normal Form (1NF)

- If a relation contain composite or multi-valued attribute, it violates first normal form, or a relation is in first normal form if it does not contain any composite or multi-valued attribute. A relation is in first normal form if every attribute in that relation is singled valued attribute.
- A relation will be 1NF if it contains an atomic value.
- It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
- First normal form disallows the multi-valued attribute, composite attribute, and their combinations.
- **First normal form** (**1NF**) is a property of a relation in a relational database. A relation is in first normal form if and only if the domain of each attribute contains only atomic (indivisible) values, and the value of each attribute contains only a single value from that domain. The first definition of the term, in a 1971 conference paper by Edgar Codd, defined a relation to be in first normal form when none of its domains have any sets as elements
- A table is in 1 NF iff(Rule)
- There are only Single Valued Attributes.
- Attribute Domain does not change.
- There is a Unique name for every Attribute/Column.
- The order in which data is stored, does not matter.
- **Example:** Relation EMPLOYEE is not in 1NF because of multi-valued attribute EMP_PHONE.

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385, 9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389, 8589830302	Punjab

The decomposition of the EMPLOYEE table into 1NF

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385	UP
14	John	9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389	Punjab
12	Sam	8589830302	Punjab

Example-2:

```
ID    Name    Courses
------------------
1     A       c1, c2
2     E       c3
3     M       C2, c3
```

In the above table Course is a multi valued attribute so it is not in 1NF.

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

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

Second Normal Form (2NF)

- A relation is in second normal form if it is in 1NF and every non key attribute is fully functionally dependent on the primary key.
- For a table to be in the Second Normal Form, it must satisfy two conditions:
- The table should be in the First Normal Form.
- There should be no Partial Dependency.(All non-key (Non prime)attributes are fully functional dependent on the primary key)
- Dependency = functional dependency(All the columns are depends on one primary key

Student

student iD	name	Address

Subject

subject code	Subject name

- Candidate Key

  - When there are 2 or more primary keys in one table is called candidate key.

Student

Reg no	Roll no	Name	age	Gender	email

- Example

Student

student iD	name	Address

Subject

subject code	Subject name

Score

Reg no	Roll no	Subject code	Marks	Professor

- So no partial dependency

Student

student iD	name	Address

Subject

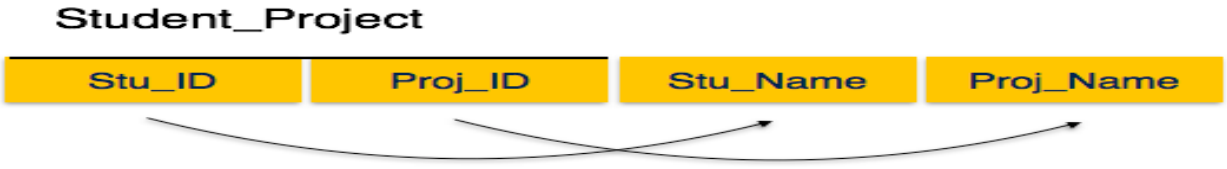
subject code	Subject name	Professor

Score

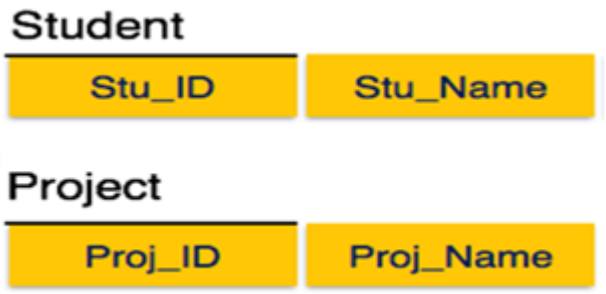
Reg no	Roll no	Subject code	Marks

Second Normal Form (2NF)

- Table is in 1NF (First normal form)
- No non-prime attribute is dependent on the proper subset of any candidate key of table.



- PARTIAL DEPENDENCY IS NOT ALLOWED. MUST BE FULLY FUNCTIONALLY DEPENDENT



Third Normal Form (3NF)

- Second Normal Form (2NF) relations have less redundancy than those in 1NF, they may still suffer from update anomalies.
- If we update only one tuple and not the other, the database would be in an inconsistent state.
- Once a table is in second normal form, we are guaranteed that every column is dependent on the primary key,
- This update anomaly is caused by a transitive dependency.
- We need to remove such dependencies by progressing to Third Normal Form (3NF).
- A relation will be in 3NF if it is in 2NF and not contain any transitive partial dependency.
- 3NF is used to reduce the data duplication. It is also used to achieve the data integrity.
- If there is no transitive dependency for non-prime attributes, then the relation must be in third normal form.
- In this case, the greater than comparison is transitive. In general, if **A** is greater than **B**, and **B** is greater than **C**, then it follows that **A** is greater than **C**.
- the same example, where we have 3 tables, **Student**, **Subject** and **Score**.

Student Table

student_id	name	reg_no	branch	address
10	RAM	39110001	CSE	Kerala
11	RAVI	39110002	IT	Gujarat
12	RAJA	39110003	IT	Rajasthan

- Subject Table

subject_id	subject_name	teacher
1	Java	Java Teacher
2	C++	C++ Teacher
3	Php	Php Teacher

- Score Table

score_id	student_id	subject_id	marks
1	10	1	70
2	10	2	75
3	11	1	80

- In the Score table, we need to store some more information, which is the exam name and total marks, so let's add 2 more columns to the Score table.

score_id	student_id	subject_id	marks	exam_name	total_marks

Requirements for Third Normal Form-For a table to be in the third normal form,

- It should be in the Second Normal form.
- And it should not have Transitive Dependency.
- With exam_name and total_marks added to our Score table,
- Primary key for our Score table is a composite key, which means it's made up of two attributes or columns → student_id + subject_id.
- Our new column exam_name depends on both student and subject. So we can say that exam_name is dependent on both student_id and subject_id.
- Well, the column total_marks depends on exam_name as with exam type the total score changes. For example, practicals are of less marks while theory exams are of more marks.
- But, exam_name is just another column in the score table. It is not a primary key or even a part of the primary key, and total_marks depends on it.
- This is Transitive Dependency. When a non-prime attribute depends on other non-prime attributes rather than depending upon the prime attributes or primary key.
- How to remove Transitive Dependency?
- Again the solution is very simple. Take out the columns exam_name and total_marks from Score table and put them in an **Exam** table and use the exam_id wherever required.
- Score Table: In 3rd Normal Form
- Advantage of removing Transitive Dependency
- The advantage of removing transitive dependency is,
- Amount of data duplication is reduced.

- Data integrity achieved

Boyce–Codd Normal Form (BCNF)

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

EMPLOYEE table:

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

```
EMP_ID → EMP_COUNTRY
EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}
```

Candidate key: {EMP-ID, EMP-DEPT}

The table is not in BCNF because neither EMP_DEPT nor EMP_ID alone are keys.

To convert the given table into BCNF, we decompose it into three tables:

EMP_COUNTRY table:

EMP_ID	EMP_COUNTRY
264	India
264	India

EMP_DEPT table:

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

EMP_DEPT_MAPPING table:

EMP_ID	EMP_DEPT
D394	283
D394	300
D283	232
D283	549

Functional dependencies:

```
EMP_ID  →  EMP_COUNTRY
EMP_DEPT  →  {DEPT_TYPE, EMP_DEPT_NO}
```

Candidate keys:

For the first table: EMP_ID
For the second table: EMP_DEPT
For the third table: {EMP_ID, EMP_DEPT}

Now, this is in BCNF because left side part of both the functional dependencies is a key.

Relational database design algorithms and further dependencies

Normal forms are insufficient on their own as a criteria for a good relational database

schema design.

• The relations in a database must collectively satisfy two other properties - dependency
preservation property and lossless (or nonadditive) join property - to qualify
as a good design.

Properties of Relational Decompositions

The relational database design algorithms discussed in this chapter start from a single universal relation schema R = {A1, A2, . . . , An} that includes all the attributes of the database. Using the functional dependencies F, the algorithms decompose R into a set of relation schemas DECOMP = {R1, R2, . . . , Rm} that will become the relational database schema.

• Examine an individual relation Ri to test whether it is in a higher normal form does not guarantee a good design (decomposition); rather, a set of relations that together form the relation database schema must possess certain additional properties to ensure a good design.

Decomposition and Dependency Preservation

It is not necessary that the exact dependencies specified in F on R appear themselves in individual relations of the composition DECOMP. It is sufficient that the union of the dependencies that hold on individual relations in DECOMP be equivalent to F.

The projection of F on Ri , denoted by πRi (F), is the set of dependencies X → Y in F + such that the attributes in X ∪ Y are contained in Ri .

A decomposition DECOMP = {R1, R2, . . . , Rm} of R is dependency-preserving with respect to F if

$$((\pi_{R_1}(F)) \cup \ldots \cup (\pi_{R_m}(F)))^+ = F^+$$

claim 1: It is always possible to find a dependency-preserving decomposition DECOMP with respect to F such that each relation Ri in DECOMP is in 3NF.

Decomposition and Lossless (Nonadditive) Joins

A decomposition DECOMP = {R1, R2, . . . , Rm} of R has the lossless join property with respect to the set of dependencies F on R if, for every relation state r of R that satisfies F, the following holds, where ∗ is the NATUAL JOIN of all the relations in DECOMP:

$$*(\pi_{R_1}(r), \ldots, \pi_{R_m}(r)) = r$$

The decomposition of EMP PROJ(SSN, PNUMBER, HOURS, ENAME, PNAME, PLOCATION) from Figure 10.3 (Fig 14.3 on e3) into EMP LOCS(ENAME, PLOCATION) and EMP PROJ1(SSN, PNUMBER, HOURS, PNAME, PLOCATION) in does not have the lossless join property. •

Another example: decompose R to R1 and R2 as follows.

R

A	B	C	D
1	2	3	4
1	2	3	5
4	2	3	4
3	1	4	2

R_1

A	B	C
1	2	3
4	2	3
3	1	4

R_2

B	C	D
2	3	4
2	3	5
1	4	2

Is R equal to $R_1 * R_2$?

$R_1 * R_2$

A	B	C	D
1	2	3	4
1	2	3	5
4	2	3	4
4	2	3	5
3	1	4	2

Algorithm : Testing for the lossless (nonadditive) join property

Input: A universal relation R, a decomposition DECOMP = {R1, R2, . . . , Rm} of R, and a set F of functional dependencies.

– 1. Create an initial matrix S with one row i for each relation Ri in DECOMP, and one column j for each attribute Aj in R.

– 2. Set S(i, j) := bij for all matrix entries.

– 3. For each row i For each column j If Ri includes attribute Aj Then set S(i, j) := aj

– 4. Repeat the following loop until a complete loop execution results in no changes to S For each X → Y in F For all rows in S which has the same symbols in the columns corresponding to attributes in X make the symbols in each column that correspond to an attribute in Y be the same in all these rows as follows: if any of the rows has an “a” symbol for the column, set the other rows to the same “a” symbol in

the column. If no “a” symbol exists for the attribute in any of the rows, choose one of the “b” symbols that appear in one of the rows for the attribute and set the other rows to that same “b” symbol in the column 3

– 5. If a row is made up entirely of “a” symbols, then the decomposition has the lossless join property; otherwise it does not.

Example:

$R = \{SSN,\ ENAME,\ PNUMBER,\ PNAME,\ PLOCATION,\ HOURS\}$

$F = \{SSN \rightarrow ENAME,\ PNUMBER \rightarrow \{PNAME,\ PLOCATION\},\ \{SSN,\ PNUMBER\} \rightarrow HOURS\}$

$DECOMP = \{R_1, R_2, R_3\}$

$R_1 = \{SSN,\ ENAME\}$

$R_2 = \{PNUMBER,\ PNAME,\ PLOCATION\}$

$R_3 = \{SSN,\ PNUMBER,\ HOURS\}$

	SSN	ENAME	PNUMBER	PNAME	PLOCATION	HOURS
R_1	a_1	a_2	b_{13}	b_{14}	b_{15}	b_{16}
R_2	b_{21}	b_{22}	a_3	a_4	a_5	b_{26}
R_3	a_1	b_{32}	a_3	b_{34}	b_{35}	a_6

	SSN	ENAME	PNUMBER	PNAME	PLOCATION	HOURS
R_1	a_1	a_2	b_{13}	b_{14}	b_{15}	b_{16}
R_2	b_{21}	b_{22}	a_3	a_4	a_5	b_{26}
R_3	a_1	$b_{32}\ a_2$	a_3	$b_{34}\ a_4$	$b_{35}\ a_5$	a_6

– Try $SSN \rightarrow ENAME$:

$a_1 \rightarrow b_{32}\ a_2$

– Try $PNUMBER \rightarrow \{PNAME,\ PLOCATION\}$:

$a_3 \rightarrow b_{34}\ a_4$

$a_3 \rightarrow b_{35}\ a_5$

In row 3, all symbols are a_i. The decomposition $DECOMP$ has the lossless join property.

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 COMPUTING

DEPARTMENT OF COMPUTER SCIENCE

UNIT IV -DATABASE MANAGEMENT SYSTEM- SBS1205

UNIT- IV

UNIT IV System Implementation Techniques: Database system architectures and the system catalog – Query processing and optimization

System Implementation Techniques

1. System Implementation Techniques
2. System Architectures for DBMS
3. Catalogs for Relational DBMS
4. System Catalog Information in ORACLE
5. Data Dictionary and Data Repository Systems

System Architectures for DBMS

Centralized DBMS Architecture

Architectures for DBMSs followed trends similar to those of general computer systems architectures. Earlier architectures used mainframe computers to provide the main processing for all functions of the system, including user application programs, user interface programs, as well as all the DBMS functionality. As the prices of hardware declined, most users replaced their terminals with personal computers (PCs) and workstations. At first, database systems used these computers in the same way as they had used display terminals, so that the DBMS itself was still a centralized DBMS where all the DBMS functionality, application program execution, and user interface processing were carried out in one machine.

Client-Server Architecture

The client server architecture was developed to deal with computing environments where a large number of PCs, workstations, file servers, printers, database servers, Web servers, and other equipment are connected together via a network. The idea is to define specialized servers with specific functionalities. For example, it is possible to connect a number of PCs or small workstations as clients to a file server that maintains the files of the client machines. Another machine could be designated as a printer server by being connected to various printers; thereafter, all print requests by the clients are forwarded to this machine. Web servers or E-mail

servers also fall into the specialized server category. In this way, the resources provided by specialized servers can be accessed by many client machines.

The concept of client-server architecture assumes an underlying framework that consists of many PCs and workstations as well as a smaller number of mainframe machines, connected via local area networks and other types of computer networks.

A client in this framework is typically a user machine that provides user interface capabilities and local processing. When a client requires access to additional functionality—such as database access—that does not exist at that machine, it connects to a server that provides the needed functionality. A server is a machine that can provide services to the client machines, such as printing, archiving, or database access.

Client-Server Architectures for DBMS

The client-server architecture is increasingly being incorporated into commercial DBMS packages. In relational DBMSs, many of which started as centralized systems, the system components that were first moved to the client side were the user interface and application programs. Because SQL provided a standard language for RDBMSs, it created a logical dividing point between client and server. Hence, the query and transaction functionality remained at the server side.

In such a client-server architecture, the user interface programs and application programs can run at the client side. When DBMS access is required, the program establishes a connection to the DBMS— which is on the server side—and once the connection is created, the client program can communicate with the DBMS. A standard called Open Database Connectivity (ODBC) provides an Application Programming Interface (API), which allows client-side programs to call the DBMS, as long as both client and server machines have the necessary software installed. Most DBMS vendors provide ODBC drivers for their systems.

The second approach to client-server was taken by some object-oriented DBMSs. Because many of those systems were developed in the era of client-server architecture, the approach taken was to divide the software modules of the DBMS between client and server in a more integrated way. For example, the server level may include the part of the DBMS software responsible for handling data storage on disk pages, local concurrency control and recovery, buffering and caching of disk pages, and other such functions. Meanwhile, the client level may handle the user interface, data dictionary functions, DBMS interaction with programming

language compilers, global query optimization/concurrency control/recovery, structuring of complex objects from the data in the buffers, and other such functions. In this approach, the client-server interaction is more tightly coupled and is done internally by the DBMS modules—some of which reside in the client—rather than by the users. The exact division of functionality varies from system to system. In such a client-server architecture, the server has been called a data server, because it provides data in disk pages to the client, which can then be structured into objects for the client programs by the client-side DBMS software itself.

Catalogs for Relational DBMS

In relational DBMSs it is common practice to store the catalog itself as relations and to use the DBMS software for querying, updating, and maintaining the catalog. This allows DBMS routines (as well as users) to access the information stored in the catalog—whenever they are authorized to do so—using the query language of the DBMS, such as SQL.

To include information on secondary key attributes of a relation, we can simply extend the preceding catalog if we assume that an attribute can be a member of one key only. In this case we can replace the MEMBER_OF_PK attribute of REL_AND_ATTR_CATALOG with an attribute KEY_NUMBER; the value of KEY_NUMBER is 0 if the attribute is not a member of any key, 1 if it is a member of the primary key, and $i > 1$ for the secondary key, where the secondary keys of a relation are numbered 2, 3, ..., n.

The definitions of views must also be stored in the catalog. A view is specified by a query, with a possible renaming of the values appearing in the query result.

System Catalog Information in ORACLE

The various commercial database products adopt different conventions and terminology with regard to their system catalog.

In ORACLE, the collection of metadata is called the data dictionary. The metadata is information about schema objects, such as tables, indexes, views, triggers, and more. Access to the data dictionary is allowed through numerous views, which are divided into three categories: USER, ALL, and DBA. These terms are used as prefixes for the various views. The views that have a prefix of USER contain schema information for objects owned by a given user. Those with a prefix of ALL contain schema information for objects owned by a user as well as objects that the user has been granted access to, and those with a prefix of DBA are for the database administrator and contain information about all database objects.

We start with the conceptual schema information. To find the objects owned by a particular user, ‘SMITH’, we can write the following query:

```
SELECT *

FROM ALL_CATALOG

WHERE OWNER  = ‘SMITH’
```

The result of this query,which indicates that three base tables are owned by SMITH: ACCOUNT, CUSTOMERS, and ORDERS, plus a view CUSTORDER. The meaning of each column in the result should be clear from its name.

Other Catalog Information Accessed by DBMS Software Modules

1. DDL (and SDL) compilers: These DBMS modules process and check the specification of a database schema in the data definition language (DDL) and store that description in the catalog. Schema constructs and constraints at all levels—conceptual, internal, and external— are extracted from the DDL and SDL (storage definition language) specifications and entered into the catalog, as is any mapping information among levels, if necessary. Hence, these software modules actually populate (load) the catalog’s minidatabase (or metadatabase) with data, the data being the descriptions of database schemas.

2. Query and DML parser and verifier: These modules parse queries, DML retrieval statements, and database update statements; they also check the catalog to verify whether all the schema names referenced in these statements are valid. For example, in a relational system, a query parser would check that all the relation names specified in the query exist in the catalog and that the attributes specified belong to the appropriate relations and have the appropriate type.

3. Query and DML compilers: These compilers convert high-level queries and DML commands into low-level file access commands. The mapping between the conceptual schema and the internal schema file structures is accessed from the catalog during this process. For example, the catalog must include a description of each file and its fields and the correspondences between fields and conceptual-level attributes.

4. Query and DML optimizer (Note 6): The query optimizer accesses the catalog for access path, implementation information, and data statistics to determine the best way to execute a query or DML command (see Chapter 18). For example, the optimizer accesses the catalog to

check which fields of a relation have hash access or indexes, before deciding how to execute a selection or join condition on the relation.

5. Authorization and security checking: The DBA has privileged commands to update the authorization and security portion of the catalog. All access by a user to a relation is checked by the DBMS for proper authorization by accessing the catalog.

6. External-to-conceptual mapping of queries and DML commands: Queries and DML commands specified with reference to an external view or schema must be transformed to refer to the conceptual schema before they can be processed by the DBMS. This is accomplished by accessing the catalog description of the view in order to perform the transformation.

Data Dictionary and Data Repository Systems

A catalog is closely coupled with the DBMS software; it provides the information stored in it to users and the DBA, but it is mainly accessed by the various software modules of the DBMS itself, such as DDL and DML compilers, the query optimizer, the transaction processor, report generators, and the constraint enforcer. On the other hand, the software package for a stand-alone data dictionary or data repository may interact with the software modules of the DBMS, but it is mainly used by the designers, users, and administrators of a computer system for information resource management. These systems are used to maintain information on system hardware and software configurations, documentation, applications, and users, as well as other information relevant to system administration.

If a data dictionary system is used only by designers, users, and administrators, not by the DBMS software, it is called a passive data dictionary; otherwise, it is called an active data dictionary or data directory. Figure 17.11 illustrates the types of active data dictionary interfaces. Data dictionaries are also used to document the database design process itself, by storing documentation on the results of every design phase and the design decisions.

QUERY PROCESSING

Algorithms for Query Processing and Optimization

Query Processing refers to the range of activities involved in extracting data from adatabase. The activities include translation of queries in high-level database languages into expressions that can be used at the physical level of the file system, a variety of query-optimizing transformations, and actual evaluation of queries.

The steps involved in processing a query are:

- Parsing and translation.
- Optimization.
- Evaluation.

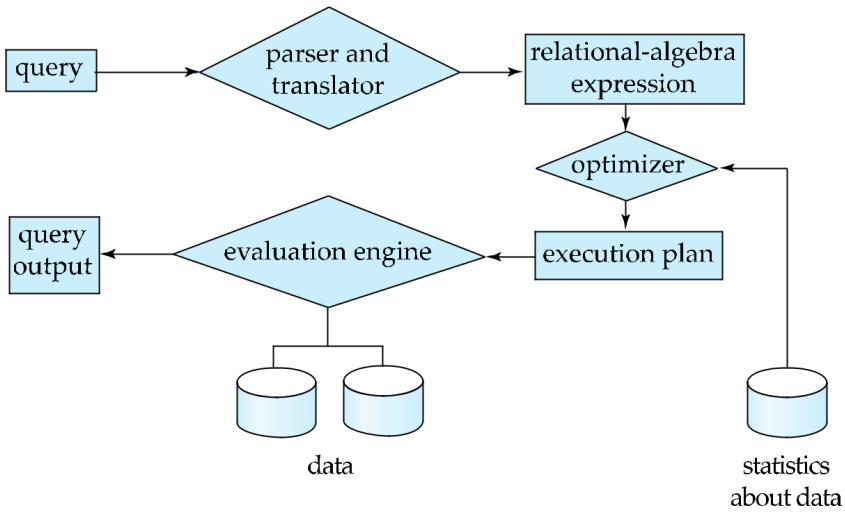


Fig. 4.1 Typical steps when processing a high level query.

- **Parsing and translation**

T

ranslate the query into its internal form. This is then translated into relational algebra.Parser checks syntax, verifies relation.

- **Optimization**

SQL is a very high level language. The users specify what to search for - not how the search is actually done. The algorithms are chosen automatically by the DBMS. For a given SQL query there may be many possible execution plans.

Amongst all equivalent plans choose the one with lowest cost. Cost is estimated using statistical information from the database catalog.

3. **Evaluation**

The query evaluation engine takes a query evaluation plan, executes that plan and returns the answer to that query. As an illustration, consider the query:

select salary from instructor where salary < 75000;

This query can be translated into either of the following relational-algebra expressions:

$$\sigma_{salary\ <75000}\ (\Pi_{salary}\ (instructor))$$

(or)

$$\Pi_{salary}\ (\sigma_{salary<75000}\ (instructor))$$

Further, each relational-algebra operation can be executed by one of several different algorithms. For example, to implement the preceding selection, we can search every tuple in instructor to find tuples with salary less than 75000. If a B^+ tree index is available on the attribute salary, we can use the index instead to locate the tuples. To specify fully how to evaluate a query, providing the relational-algebra expression will not be anought, but also to annotate it with instructions specifying how to evaluate each operation.

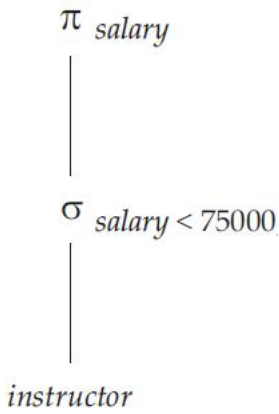


Figure 4.2 A query-evaluation plan

A sequence of primitive operations that can be used to evaluate a query is a query-execution plan or query-evaluation plan. Figure 3.1.2 illustrates an evaluation plan for our example query. The query-execution engine takes a query-evaluation plan, executes that plan, and returns the answers to the query. The different evaluation plans for a given query can have different costs. It is the responsibility of the system to construct a query evaluation plan that minimizes the cost of query evaluation; this task is called query optimization. Once the query plan is chosen, the query is evaluated with that plan, and the result of the query is output. In order to optimize a query, a query optimizer must know the cost of each operation.

Measures of Query Cost

The cost of query evaluation can be measured in terms of a number of different resources, including disk accesses, CPU time to execute a query and in a distributed or parallel database system, the cost of communication.

The response time for a query-evaluation plan could be used as a good measure of the cost of the plan. In large database systems, however, disk accesses are usually the most important cost, since disk accesses are slow compared to in-memory operations. Most people consider the disk accesses cost a reasonable measure of the cost of a query-evaluation plan.

The number of block transfers from disk is also used as a measure of the actual cost. It takes more time to write a block to disk than to read a block from disk. For more accurate measure, find:

- The number of seek operations performed
- The number of blocks read
- The number of blocks written

and then add up these numbers after multiplying them by average seek time, average transfer time for reading a block and average transfer time for writing a block respectively.

Using Heuristics in Query Optimization

Query Optimization - Heuristics

Query Trees and Graphs

selectP.Pnumber, P.Dnum, E.Lname, E.Address, E.Bdate from Project as P, Department as D, Employee as E where P.Dnum=D.Dnumber and .Mgr_ssn=E.Ssn and P.Plocation='Stafford';

πPnumber, Dnum, Lname, Address, Bdate[[(σPlocation='Stafford'(Project)) ⋈Dnum=Dnumber (Department)] ⋈Mgr_ssn=Ssn(Employee)]

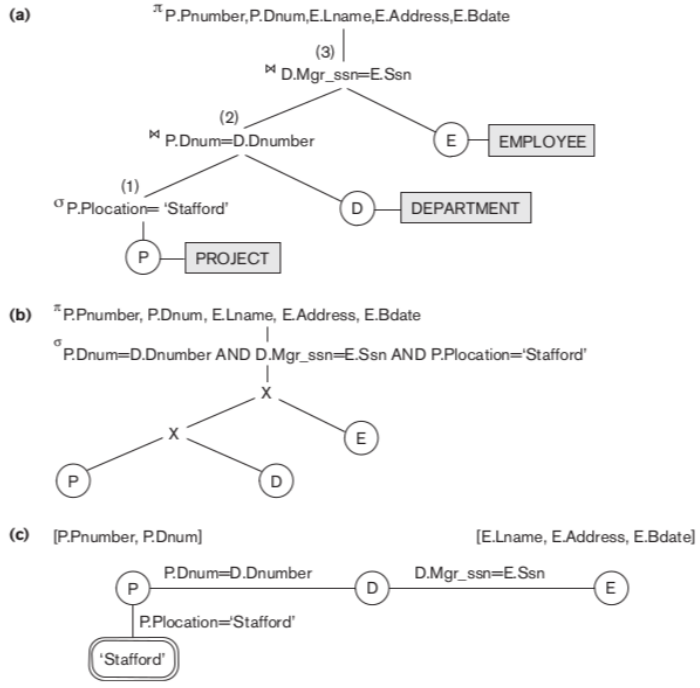


Figure 19.4
Two query trees for the query Q2. (a) Query tree corresponding to the relational algebra expression for Q2. (b) Initial (canonical) query tree for SQL query Q2. (c) Query graph for Q2.

Optimizing Them

In actuality the initial translation is simplistic:

From	×

Where σ

Select π

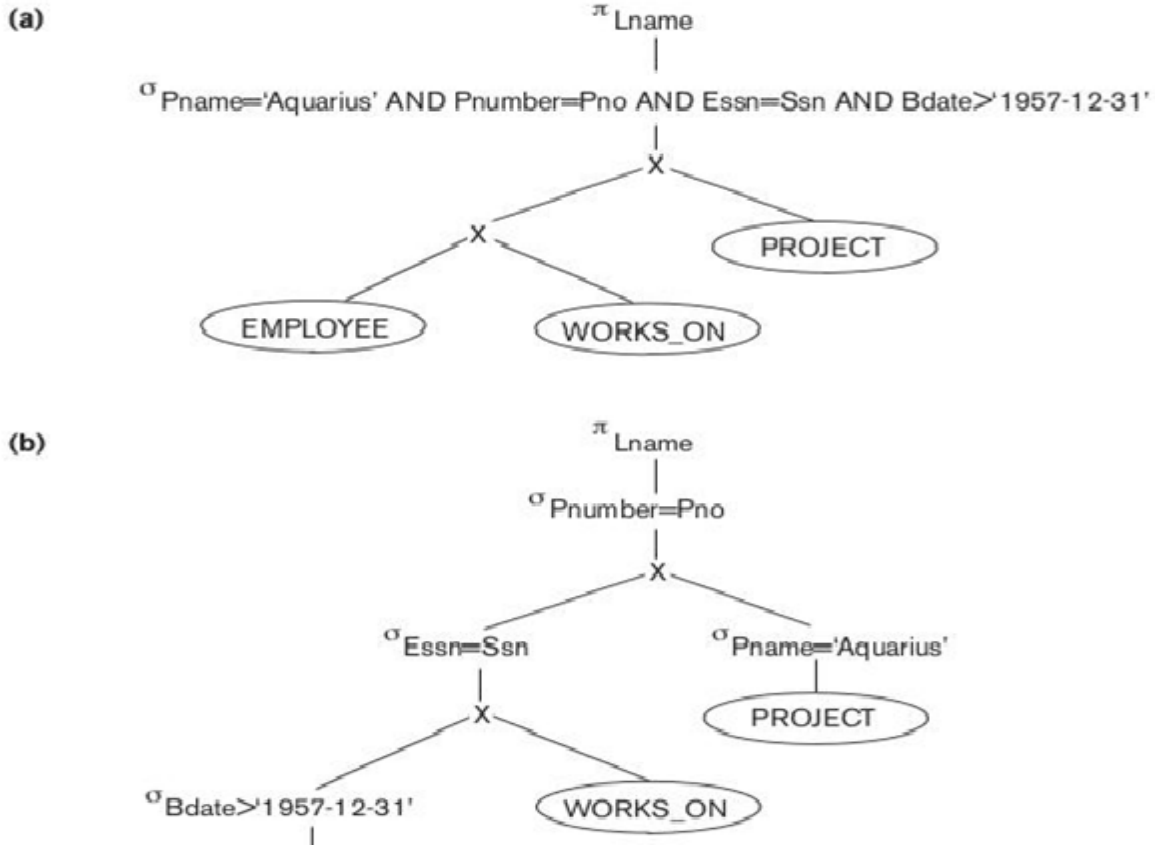
Then optimizations transform the query tree into something more efficient based on reordering and transforming the extended relational operations. After that the tree is converted to a *query execution plan* that chooses the best algorithms to implement the portions of the tree. πPnumber, Dnum, Lname, Address, Bdate(σP.Dnum=D.DnumberΛD.Mgr_ssn=E.SsnΛ

P.Plocation='Stafford'((Project × Department) × Employee))

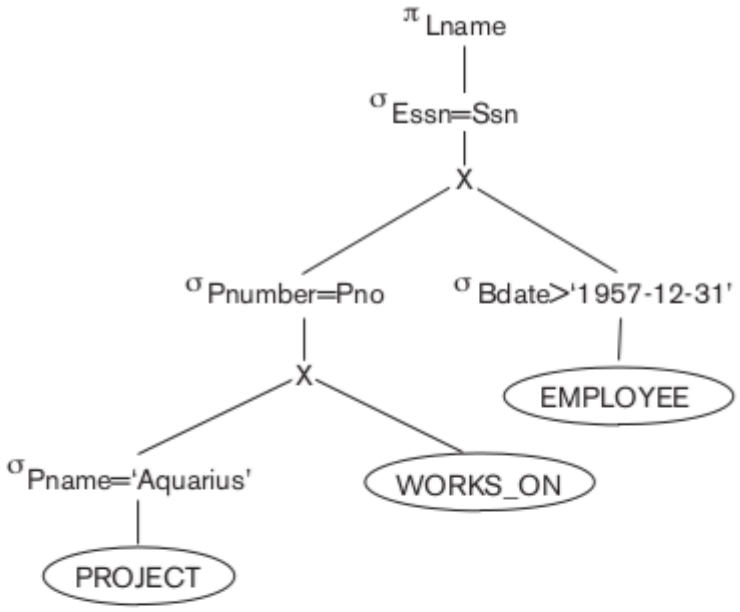
Find the last names of employees born after 1957 who work on a project named ‘Aquarius’.

select Lname fromEmployee, Works_On, Project where Pname='Aquarius' and Pnumber=Pno
and Essn=Ssn and Bdate> '1957-12-31';

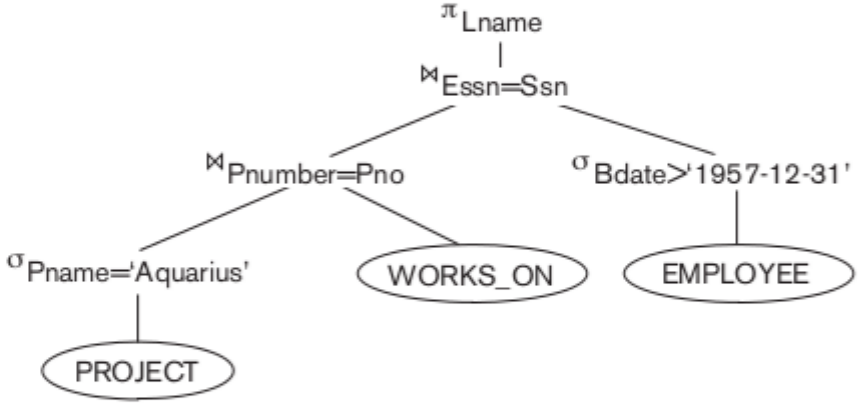
Steps in converting a query tree during heuristic optimization.
(a) Initial (canonical) query tree for SQL query Q.
(b) Moving SELECT operations down the query tree.
(c) Applying the more restrictive SELECT operation first.
(d) Replacing CARTESIAN PRODUCT and SELECT with JOIN operations.
(e) Moving PROJECT operations down the query tree.



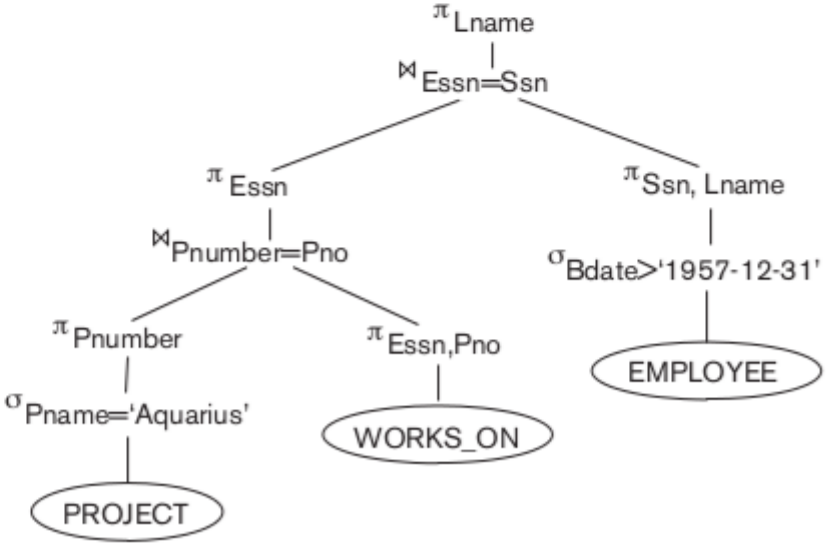
(c)



(d)



(e)



es) *before*

1	Cascade σ	$\sigma c_1 \wedge c_2 \wedge \ldots \wedge cn \equiv \sigma c_1(\sigma c_2(\ldots \sigma cn(R)))$	
2	Commutativity of σ	$\sigma a(\sigma b(R)) \equiv \sigma b(\sigma a(R))$	
3	Cascade ππ	$\pi a(\pi b(\ldots(R))) \equiv \pi a(R)$	
4	Commute σ with π	$\pi a,b,\ldots(\sigma c(R)) \equiv \sigma c(\pi a,b,\ldots(R))$	Only if c only depends on the attributes of π
5	Commutativity of ⋈	$R \bowtie S \equiv S \bowtie R$	Also true of ×
6	Commuting σ with ⋈	$\sigma c(R \bowtie S) \equiv (\sigma c(R)) \bowtie S$	Only if c involves only the attributes of R. Also applies to ×.

6a	Commuting σ with ⋈, conjunction version	$\sigma c_1 \wedge c_2(R \bowtie S) \equiv (\sigma c_1(R$ $\bowtie (\sigma c_2(S))$	

Where c_1 and c_2 involve the attributes of R and S, respectively.

7	Commuting π with ⋈	$\pi L\ (R \bowtie c\ S) \equiv$ $(\pi A_1,\ldots,An(R))$ $\bowtie c(\pi B_1,\ldots,Bm(S))$	So long as the projection attributes L = $\{A_1,\ldots,An,B_1,\ldots,Bm\}$ where A ∈Attr(R), B ∈Attr(S), and the join condition c only dependes on attributes in L.
7a	Commuting π with ⋈, extra join attributes version	$\pi L\ (R \bowtie c\ S) \equiv$ πL $[(\pi A_1,\ldots,An,\ldots,An+k(R))$ $\bowtie c(\pi B_1,\ldots,Bm,\ldots,Bm+p($ $))]$	Here the projection attributes in L are the same, but the join condition c depends on additional attributes from R and S. These attributes are shown as An+1…An+k and Bm+1…Bm+p. The

			original πL must be wrapped around the right-side expression to remove those extra join atttributes one they're unneeded.
8	Commutativity of set operations		∩ and ∪ are commutative but – is not
9	Associativity of ⋈, ×, ∪, and ∩	(R×S)×T≡R×(S×T)	Also true for each of the other three

General approach:

- Start with the canonical tree.
- Move σs as low as possible.
- Merge operations when possible (×,σ → ⋈ is classic).
- Move πs down, but not probably not below σs.

Example query optimization

selectFname, Lname, Address

from Employee as E, Department as D

where D.Super_ssn=E.Ssn and D.Dname like 'Plan%';

Converting Query Trees to Execution Plans

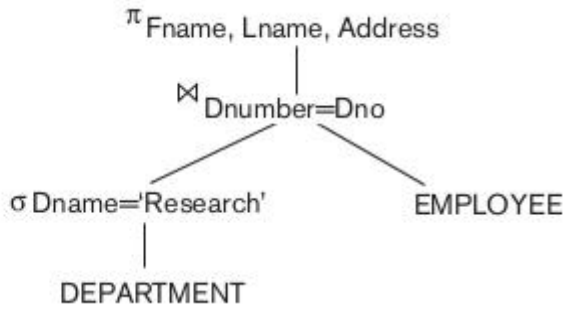


Figure 19.6
A query tree for query Q1.

- If there is an index on Department.Dname, use it for an index search to implement the σDname='Research'.
- If there is an index on Employee.Dno, implement the ⋈Dnumber=Dno by a J2 (Single-loop join).
- Lastly a simple π over those results.
- Ideally all these algorithms would be pipelined as a single step without writing intermediate files.

Using Selectivity and Cost Estimates in Query Optimization

The DBMS attempts to form a good *cost model* of various query operations as applied to the current database state, including the attribute value statistics (histogram), nature of indices,

number of block buffers that can be allocated to various pipelines, selectivity of selection clauses, storage speed, network speed (for distributed databases in particular), and so on.

Access cost to secondary storage

Reading and writing blocks between storage and ram. (time)

Disk storage cost

Cost of temporary intermediate files in storage. (time/space)

Computation cost

Usually slower than storage, but sometimes not, the cpu cost of evaluating the query.

(time)

Memory usage cost

The amount of ram needed by the query. (space)

Communication cost

The cost to transport query data over a network between database nodes. (time) "Typical" databases emphasize access cost, the usual limiting factor. In-memory databases minimize computation cost, while distributed databases put increasing emphasis on communication cost.

Catalog Information Used in Cost Functions

- Basic file data: number of records (r), record size (R), number of blocks (b).
- Primary file organization (heap file, ordered, ordered with index, hashed, overflow?)

- Other indices, if present.
- Number of levels (x) for multilevel indices, top-level block count (bI1).
- Number of distinct values (d) of an attribute. If the values are uniformly distributed, the *selectivity* (sl) is (^{1}/d), and the selection cardinality (s = sl·r). If the attribute values are *skewed* this isn't a good estimate, and a more detailed histogram of sl is appropriate.

Examples of Cost Functions for Select

S1. linear search

On average half the blocks must be accessed for an equality condition on the key, all the blocks otherwise.

S2. binary search

lg b for the search, plus more if it's nonkey.

S3a. primary index for a single record

One more than the number of index levels.

S3b. hash index for a single record

Average of 1 or 2 depending on the type of hash.

S4,.ordered index for multiple records

CS4=x+^{b}/2 as a rough estimate.

S5. clustering index for multiple records

x for the index search to get the cluster block, then ⌈s/bfr⌉ file blocks pointed to by the cluster.

S6. B^+-Tree

Tree height + 1 if its key, that plus the selectivity (s) if not.

S7. conjunctive selection

The sum of the costs of the subconditions, if the set intersection fits in memory.

S8,.conjunctive selection with composite index

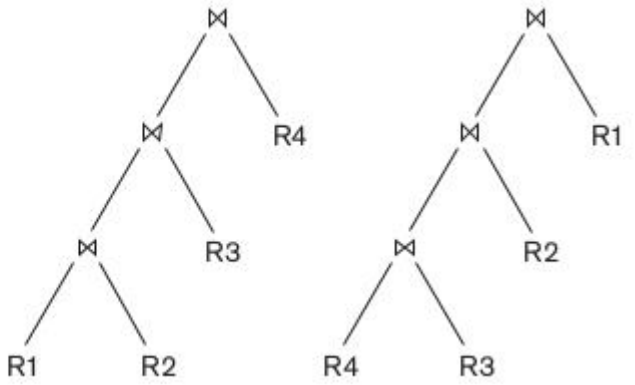
The same as above for whichever type of index.

Examples of Cost Functions for Join

A *join selectivity* (js) is the ratio of the number of tuples produced by a join to the number of tuples produced by a cross product of the underlying relations, or js = |(R ⋈c S)| / |(R×S)| = |(R ⋈c S)| / (|R|·|S|).

Multiple Relation Queries and Join Ordering

Figure 19.7
Two left-deep (JOIN) query trees.



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 COMPUTING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIT – V- Database Management Systems – SBS1205

SBS1205 DATABASE MANAGEMENT SYSTEM

Unit – V

Transaction processing concepts – Concurrency control techniques – Database recovery techniques – Database security and Authorization. Introduction to Emerging Database Technologies and Application

Transactions

A transaction is a program including a collection of database operations, executed as a logical unit of data processing. The operations performed in a transaction include one or more of database operations like insert, delete, update or retrieve data. It is an atomic process that is either performed into completion entirely or is not performed at all. A transaction involving only data retrieval without any data update is called read-only transaction.

Each high level operation can be divided into a number of low level tasks or operations. For example, a data update operation can be divided into three tasks −

- **read_item()** − reads data item from storage to main memory.
- **modify_item()** − change value of item in the main memory.
- **write_item()** − write the modified value from main memory to storage.

Database access is restricted to read_item() and write_item() operations. Likewise, for all transactions, read and write forms the basic database operations.

Transaction Operations

The low level operations performed in a transaction are −

- **begin_transaction** − A marker that specifies start of transaction execution.
- **read_item or write_item** − Database operations that may be interleaved with main memory operations as a part of transaction.
- **end_transaction** − A marker that specifies end of transaction.

- **commit** − A signal to specify that the transaction has been successfully completed in its entirety and will not be undone.
- **rollback** − A signal to specify that the transaction has been unsuccessful and so all temporary changes in the database are undone. A committed transaction cannot be rolled back.

Transaction States

A transaction may go through a subset of five states, active, partially committed, committed, failed and aborted.

- **Active** − The initial state where the transaction enters is the active state. The transaction remains in this state while it is executing read, write or other operations.
- **Partially Committed** − The transaction enters this state after the last statement of the transaction has been executed.
- **Committed** − The transaction enters this state after successful completion of the transaction and system checks have issued commit signal.
- **Failed** − The transaction goes from partially committed state or active state to failed state when it is discovered that normal execution can no longer proceed or system checks fail.
- **Aborted** − This is the state after the transaction has been rolled back after failure and the database has been restored to its state that was before the transaction began.

Desirable Properties of Transactions

Any transaction must maintain the ACID properties, viz. Atomicity, Consistency, Isolation, and Durability.

- **Atomicity** − This property states that a transaction is an atomic unit of processing, that is, either it is performed in its entirety or not performed at all. No partial update should exist.
- **Consistency** − A transaction should take the database from one consistent state to another consistent state. It should not adversely affect any data item in the database.
- **Isolation** − A transaction should be executed as if it is the only one in the system. There should not be any interference from the other concurrent transactions that are simultaneously running.

- **Durability** − If a committed transaction brings about a change, that change should be durable in the database and not lost in case of any failure.

Schedules and Conflicts

In a system with a number of simultaneous transactions, a **schedule** is the total order of execution of operations. Given a schedule S comprising of n transactions, say T1, T2, T3……..Tn; for any transaction Ti, the operations in Ti must execute as laid down in the schedule S.

Types of Schedules

There are two types of schedules −

- **Serial Schedules** − In a serial schedule, at any point of time, only one transaction is active, i.e. there is no overlapping of transactions. This is depicted in the following graph −

- **Parallel Schedules** − In parallel schedules, more than one transactions are active simultaneously, i.e. the transactions contain operations that overlap at time. This is depicted in the following graph −

Conflicts in Schedules

In a schedule comprising of multiple transactions, a **conflict** occurs when two active transactions perform non-compatible operations. Two operations are said to be in conflict, when all of the following three conditions exists simultaneously −

- The two operations are parts of different transactions.
- Both the operations access the same data item.
- At least one of the operations is a write_item() operation, i.e. it tries to modify the data item.

Serializability

A **serializable schedule** of 'n' transactions is a parallel schedule which is equivalent to a serial schedule comprising of the same 'n' transactions. A serializable schedule contains the correctness of serial schedule while ascertaining better CPU utilization of parallel schedule.

Equivalence of Schedules

Equivalence of two schedules can be of the following types −

- **Result equivalence** − Two schedules producing identical results are said to be result equivalent.
- **View equivalence** − Two schedules that perform similar action in a similar manner are said to be view equivalent.
- **Conflict equivalence** − Two schedules are said to be conflict equivalent if both contain the same set of transactions and has the same order of conflicting pairs of operations.

Concurrency controlling techniques

Its ensure that multiple transactions are executed simultaneously while maintaining the ACID properties of the transactions and serializability in the schedules.

In this chapter, we will study the various approaches for concurrency control.

Locking Based Concurrency Control Protocols

Locking-based concurrency control protocols use the concept of locking data items. A **lock** is a variable associated with a data item that determines whether read/write operations can be performed on that data item. Generally, a lock compatibility matrix is used which states whether a data item can be locked by two transactions at the same time.

Locking-based concurrency control systems can use either one-phase or two-phase locking protocols.

One-phase Locking Protocol

In this method, each transaction locks an item before use and releases the lock as soon as it has finished using it. This locking method provides for maximum concurrency but does not always enforce serializability.

Two-phase Locking Protocol

In this method, all locking operations precede the first lock-release or unlock operation. The transaction comprise of two phases. In the first phase, a transaction only acquires all the locks it needs and do not release any lock. This is called the expanding or the **growing phase**. In the second phase, the transaction releases the locks and cannot request any new locks. This is called the **shrinking phase**.

Every transaction that follows two-phase locking protocol is guaranteed to be serializable. However, this approach provides low parallelism between two conflicting transactions.

Timestamp Concurrency Control Algorithms

Timestamp-based concurrency control algorithms use a transaction's timestamp to coordinate concurrent access to a data item to ensure serializability. A timestamp is a unique identifier given by DBMS to a transaction that represents the transaction's start time.

These algorithms ensure that transactions commit in the order dictated by their timestamps. An older transaction should commit before a younger transaction, since the older transaction enters the system before the younger one.

Timestamp-based concurrency control techniques generate serializable schedules such that the equivalent serial schedule is arranged in order of the age of the participating transactions.

Some of timestamp based concurrency control algorithms are −

- Basic timestamp ordering algorithm.
- Conservative timestamp ordering algorithm.
- Multiversion algorithm based upon timestamp ordering.

Timestamp based ordering follow three rules to enforce serializability −

- **Access Rule** − When two transactions try to access the same data item simultaneously, for conflicting operations, priority is given to the older transaction. This causes the younger transaction to wait for the older transaction to commit first.
- **Late Transaction Rule** − If a younger transaction has written a data item, then an older transaction is not allowed to read or write that data item. This rule prevents the older transaction from committing after the younger transaction has already committed.
- **Younger Transaction Rule** − A younger transaction can read or write a data item that has already been written by an older transaction.

Optimistic Concurrency Control Algorithm

In systems with low conflict rates, the task of validating every transaction for serializability may lower performance. In these cases, the test for serializability is postponed to just before commit. Since the conflict rate is low, the probability of aborting transactions which are not serializable is also low. This approach is called optimistic concurrency control technique.

In this approach, a transaction's life cycle is divided into the following three phases −

- **Execution Phase** − A transaction fetches data items to memory and performs operations upon them.

- **Validation Phase** − A transaction performs checks to ensure that committing its changes to the database passes serializability test.
- **Commit Phase** − A transaction writes back modified data item in memory to the disk.

This algorithm uses three rules to enforce serializability in validation phase −

Rule 1 − Given two transactions T_i and T_j, if T_i is reading the data item which T_j is writing, then T_i's execution phase cannot overlap with T_j's commit phase. T_j can commit only after T_i has finished execution.

Rule 2 − Given two transactions T_i and T_j, if T_i is writing the data item that T_j is reading, then T_i's commit phase cannot overlap with T_j's execution phase. T_j can start executing only after T_i has already committed.

Rule 3 − Given two transactions T_i and T_j, if T_i is writing the data item which T_j is also writing, then T_i's commit phase cannot overlap with T_j's commit phase. T_j can start to commit only after T_i has already committed.

Concurrency Control in Distributed Systems

In this section, we will see how the above techniques are implemented in a distributed database system.

Distributed Two-phase Locking Algorithm

The basic principle of distributed two-phase locking is same as the basic two-phase locking protocol. However, in a distributed system there are sites designated as lock managers. A lock manager controls lock acquisition requests from transaction monitors. In order to enforce co-ordination between the lock managers in various sites, at least one site is given the authority to see all transactions and detect lock conflicts.

Depending upon the number of sites who can detect lock conflicts, distributed two-phase locking approaches can be of three types −

- **Centralized two-phase locking** − In this approach, one site is designated as the central lock manager. All the sites in the environment know the location of the central lock manager and obtain lock from it during transactions.
- **Primary copy two-phase locking** − In this approach, a number of sites are designated as lock control centers. Each of these sites has the responsibility of managing a defined set of locks. All the sites know which lock control center is responsible for managing lock of which data table/fragment item.

- **Distributed two-phase locking** − In this approach, there are a number of lock managers, where each lock manager controls locks of data items stored at its local site. The location of the lock manager is based upon data distribution and replication.

Distributed Timestamp Concurrency Control

In a centralized system, timestamp of any transaction is determined by the physical clock reading. But, in a distributed system, any site's local physical/logical clock readings cannot be used as global timestamps, since they are not globally unique. So, a timestamp comprises of a combination of site ID and that site's clock reading.

For implementing timestamp ordering algorithms, each site has a scheduler that maintains a separate queue for each transaction manager. During transaction, a transaction manager sends a lock request to the site's scheduler. The scheduler puts the request to the corresponding queue in increasing timestamp order. Requests are processed from the front of the queues in the order of their timestamps, i.e. the oldest first.

Deadlock is a state of a database system having two or more transactions, when each transaction is waiting for a data item that is being locked by some other transaction. A deadlock can be indicated by a cycle in the wait-for-graph. This is a directed graph in which the vertices denote transactions and the edges denote waits for data items.

For example, in the following wait-for-graph, transaction T1 is waiting for data item X which is locked by T3. T3 is waiting for Y which is locked by T2 and T2 is waiting for Z which is locked by T1. Hence, a waiting cycle is formed, and none of the transactions can proceed executing.

Deadlock Handling in Centralized Systems

There are three classical approaches for deadlock handling, namely −

- Deadlock prevention.
- Deadlock avoidance.
- Deadlock detection and removal.

All of the three approaches can be incorporated in both a centralized and a distributed database system.

Deadlock Prevention

The deadlock prevention approach does not allow any transaction to acquire locks that will lead to deadlocks. The convention is that when more than one transactions request for locking the same data item, only one of them is granted the lock.

One of the most popular deadlock prevention methods is pre-acquisition of all the locks. In this method, a transaction acquires all the locks before starting to execute and retains the locks for the entire duration of transaction. If another transaction needs any of the already acquired locks, it has to wait until all the locks it needs are available. Using this approach, the system is prevented from being deadlocked since none of the waiting transactions are holding any lock.

Deadlock Avoidance

The deadlock avoidance approach handles deadlocks before they occur. It analyzes the transactions and the locks to determine whether or not waiting leads to a deadlock.

The method can be briefly stated as follows. Transactions start executing and request data items that they need to lock. The lock manager checks whether the lock is available. If it is available, the lock manager allocates the data item and the transaction acquires the lock. However, if the item is locked by some other transaction in incompatible mode, the lock manager runs an algorithm to test whether keeping the transaction in waiting state will cause a deadlock or not. Accordingly, the algorithm decides whether the transaction can wait or one of the transactions should be aborted.

There are two algorithms for this purpose, namely **wait-die** and **wound-wait**. Let us assume that there are two transactions, T1 and T2, where T1 tries to lock a data item which is already locked by T2. The algorithms are as follows −

- **Wait-Die** − If T1 is older than T2, T1 is allowed to wait. Otherwise, if T1 is younger than T2, T1 is aborted and later restarted.
- **Wound-Wait** − If T1 is older than T2, T2 is aborted and later restarted. Otherwise, if T1 is younger than T2, T1 is allowed to wait.

Deadlock Detection and Removal

The deadlock detection and removal approach runs a deadlock detection algorithm periodically and removes deadlock in case there is one. It does not check for deadlock when a transaction places a request for a lock. When a transaction requests a lock, the lock manager checks whether it is available. If it is available, the transaction is allowed to lock the data item; otherwise the transaction is allowed to wait.

Since there are no precautions while granting lock requests, some of the transactions may be deadlocked. To detect deadlocks, the lock manager periodically checks if the wait-forgraph has cycles. If the system is deadlocked, the lock manager chooses a victim transaction from each cycle. The victim is aborted and rolled back; and then restarted later. Some of the methods used for victim selection are −

- Choose the youngest transaction.

- Choose the transaction with fewest data items.
- Choose the transaction that has performed least number of updates.
- Choose the transaction having least restart overhead.
- Choose the transaction which is common to two or more cycles.

This approach is primarily suited for systems having transactions low and where fast response to lock requests is needed.

Deadlock Handling in Distributed Systems

Transaction processing in a distributed database system is also distributed, i.e. the same transaction may be processing at more than one site. The two main deadlock handling concerns in a distributed database system that are not present in a centralized system are **transaction location** and **transaction control**. Once these concerns are addressed, deadlocks are handled through any of deadlock prevention, deadlock avoidance or deadlock detection and removal.

Transaction Location

Transactions in a distributed database system are processed in multiple sites and use data items in multiple sites. The amount of data processing is not uniformly distributed among these sites. The time period of processing also varies. Thus the same transaction may be active at some sites and inactive at others. When two conflicting transactions are located in a site, it may happen that one of them is in inactive state. This condition does not arise in a centralized system. This concern is called transaction location issue.

This concern may be addressed by Daisy Chain model. In this model, a transaction carries certain details when it moves from one site to another. Some of the details are the list of tables required, the list of sites required, the list of visited tables and sites, the list of tables and sites that are yet to be visited and the list of acquired locks with types. After a transaction terminates by either commit or abort, the information should be sent to all the concerned sites.

Transaction Control

Transaction control is concerned with designating and controlling the sites required for processing a transaction in a distributed database system. There are many options regarding the choice of where to process the transaction and how to designate the center of control, like −

- One server may be selected as the center of control.
- The center of control may travel from one server to another.
- The responsibility of controlling may be shared by a number of servers.

Distributed Deadlock Prevention

Just like in centralized deadlock prevention, in distributed deadlock prevention approach, a transaction should acquire all the locks before starting to execute. This prevents deadlocks.

The site where the transaction enters is designated as the controlling site. The controlling site sends messages to the sites where the data items are located to lock the items. Then it waits for confirmation. When all the sites have confirmed that they have locked the data items, transaction starts. If any site or communication link fails, the transaction has to wait until they have been repaired.

Though the implementation is simple, this approach has some drawbacks −

- Pre-acquisition of locks requires a long time for communication delays. This increases the time required for transaction.
- In case of site or link failure, a transaction has to wait for a long time so that the sites recover. Meanwhile, in the running sites, the items are locked. This may prevent other transactions from executing.
- If the controlling site fails, it cannot communicate with the other sites. These sites continue to keep the locked data items in their locked state, thus resulting in blocking.

Distributed Deadlock Avoidance

As in centralized system, distributed deadlock avoidance handles deadlock prior to occurrence. Additionally, in distributed systems, transaction location and transaction control issues needs to be addressed. Due to the distributed nature of the transaction, the following conflicts may occur –

- Conflict between two transactions in the same site.
- Conflict between two transactions in different sites.

In case of conflict, one of the transactions may be aborted or allowed to wait as per distributed wait-die or distributed wound-wait algorithms.

Let us assume that there are two transactions, T1 and T2. T1 arrives at Site P and tries to lock a data item which is already locked by T2 at that site. Hence, there is a conflict at Site P. The algorithms are as follows −

- **Distributed Wound-Die**
    - If T1 is older than T2, T1 is allowed to wait. T1 can resume execution after Site P receives a message that T2 has either committed or aborted successfully at all sites.

- If T1 is younger than T2, T1 is aborted. The concurrency control at Site P sends a message to all sites where T1 has visited to abort T1. The controlling site notifies the user when T1 has been successfully aborted in all the sites.

- **Distributed Wait-Wait**
  - If T1 is older than T2, T2 needs to be aborted. If T2 is active at Site P, Site P aborts and rolls back T2 and then broadcasts this message to other relevant sites. If T2 has left Site P but is active at Site Q, Site P broadcasts that T2 has been aborted; Site L then aborts and rolls back T2 and sends this message to all sites.
  - If T1 is younger than T1, T1 is allowed to wait. T1 can resume execution after Site P receives a message that T2 has completed processing.

Distributed Deadlock Detection

Just like centralized deadlock detection approach, deadlocks are allowed to occur and are removed if detected. The system does not perform any checks when a transaction places a lock request. For implementation, global wait-for-graphs are created. Existence of a cycle in the global wait-for-graph indicates deadlocks. However, it is difficult to spot deadlocks since transaction waits for resources across the network.

Alternatively, deadlock detection algorithms can use timers. Each transaction is associated with a timer which is set to a time period in which a transaction is expected to finish. If a transaction does not finish within this time period, the timer goes off, indicating a possible deadlock.

Another tool used for deadlock handling is a deadlock detector. In a centralized system, there is one deadlock detector. In a distributed system, there can be more than one deadlock detectors. A deadlock detector can find deadlocks for the sites under its control. There are three alternatives for deadlock detection in a distributed system, namely.

- **Centralized Deadlock Detector** − One site is designated as the central deadlock detector.
- **Hierarchical Deadlock Detector** − A number of deadlock detectors are arranged in hierarchy.
- **Distributed Deadlock Detector** − All the sites participate in detecting deadlocks and removing them.

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.

Transaction Log

For recovery from any type of failure data values prior to modification (BFIM BeFore Image) and the new value after modification (AFIM – AFter Image) are required.

These values and other information is stored in a sequential file called Transaction log.

Data Update

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

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.

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

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

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.'

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. These operations are recorded in the log as they happen. 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 WriteAhead 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.

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:

Suspend execution of transactions temporarily. Force writes modified buffer data to disk.

Write a [checkpoint] record to the log, save the log to disk. Resume normal transaction execution. During recovery redo or undo is required to transactions appearing after [checkpoint] record.

Steal/No-Steal and Force/No-Force

Possible ways for flushing database cache to database disk:

Steal: Cache can be flushed before transaction commits.

No-Steal: Cache cannot be flushed before transaction commit.

Force: Cache is immediately flushed (forced) to disk.

No-Force: Cache is deferred until transaction commits

These give rise to four different ways for handling recovery:

Steal/No-Force (Undo/Redo)

Steal/Force (Undo/No-redo)

No-Steal/No-Force (Redo/No-undo)

No-Steal/Force (No-undo/No-redo)

Recovery Scheme

a. 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.

b. Deferred Update with concurrent users

This environment requires some concurrency control mechanism to guarantee isolation property of transactions. In a system recovery transactions which were recorded in the log after the last checkpoint were redone. The recovery manager may scan some of the transactions recorded before the checkpoint to get the AFIMs

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.

c. 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 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.

Undo/Redo Algorithm (Concurrent execution)

Recovery schemes of this category applies undo and also redo to recover the database from failure.

In concurrent execution environment a concurrency control is required and log is maintained under WAL.

Commit table records transactions to be committed and active table records active transactions. To minimize the work of the recovery manager check pointing is used.

The recovery performs:

Undo of a transaction if it is in the active table.

Redo of a transaction if it is in the commit table.

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 transactions two directories (current and shadow) are used

The directory arrangement is illustrated below. Here a page is a data item.

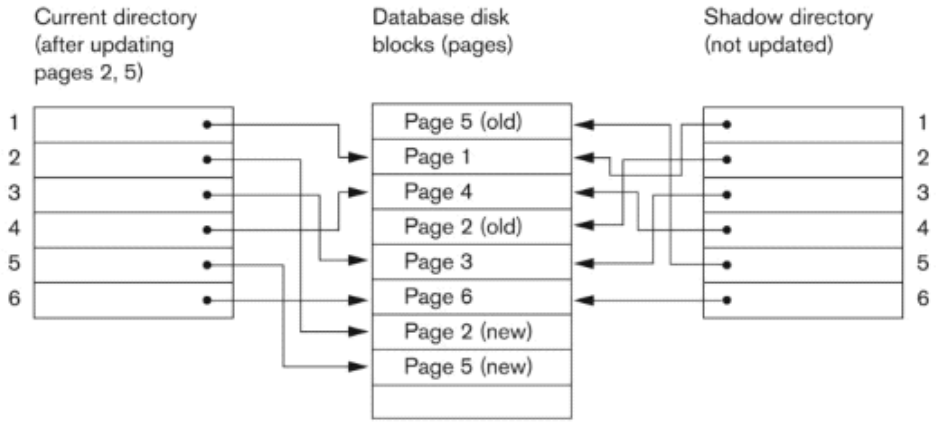


Figure 19.5
An example of shadow paging.

Fig 5.1 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 Log and Log Sequence Number (LSN)

A log record is written for:

- data update
- transaction commit

- transaction abort
- undo
- transaction end

In the case of undo a compensating log record is written. The Log and Log Sequence Number (LSN) (contd.). A unique LSN is associated with every log record. LSN increases monotonically and indicates the disk address of the log record it is associated with.

In addition, each data page stores the LSN of the latest log record corresponding to a change for that page.

A log record stores

(a) the previous LSN of that transaction

(b) the transaction ID

A log record stores:

Previous LSN of that transaction: It links the log record of each transaction. It is like a back pointer points to the previous record of the same transaction , Transaction ID , Type of log record. For a write operation the following additional information is logged:

Page ID for the page that includes the item

Length of the updated item

Its offset from the beginning of the page

BFIM of the item

AFIM of the item

The Transaction table and the Dirty Page table

For efficient recovery following tables are also stored in the log during check pointing:

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.

e. 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

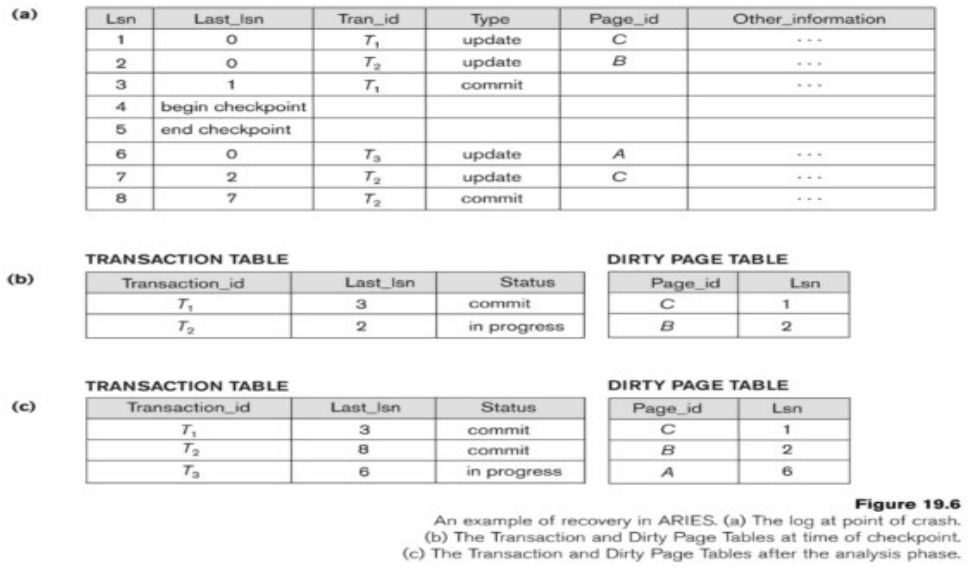
log.Writes the LSN of the begin_checkpoint record to a special file. This special file is accessed during recovery to locate the last checkpoint information. To reduce the cost of checkpointing and allow the system to continue to execute transactions, ARIES uses “fuzzy checkpointing”.

The 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 undoes 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.



(a)

Lsn	Last_lsn	Tran_id	Type	Page_id	Other_information
1	0	T_1	update	C	. . .
2	0	T_2	update	B	. . .
3	1	T_1	commit		. . .
4	begin checkpoint				
5	end checkpoint				
6	0	T_3	update	A	. . .
7	2	T_2	update	C	. . .
8	7	T_2	commit		. . .

(b)

TRANSACTION TABLE

Transaction_id	Last_lsn	Status
T_1	3	commit
T_2	2	in progress

DIRTY PAGE TABLE

Page_id	Lsn
C	1
B	2

(c)

TRANSACTION TABLE

Transaction_id	Last_lsn	Status
T_1	3	commit
T_2	8	commit
T_3	6	in progress

DIRTY PAGE TABLE

Page_id	Lsn
C	1
B	2
A	6

Figure 19.6
An example of recovery in ARIES. (a) The log at point of crash. (b) The Transaction and Dirty Page Tables at time of checkpoint. (c) The Transaction and Dirty Page Tables after the analysis phase.

Fig 5.2 Aries Algorithm

f. 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.

- **Database security and Authorization**

  It can be classified into two categories: system security and data security.

  System security includes the mechanisms that control the access and use of the database at the system level. For example, system security includes:

  •Valid user name/password combinations

  •The amount of disk space available to a user's schema objects

  •The resource limits for a user

  System security mechanisms check whether a user is authorized to connect to the database, whether database auditing is active, and which system operations a user can perform.

  Data security includes the mechanisms that control the access and use of the database at the schema object level. For example, data security includes:

  •Which users have access to a specific schema object and the specific types of actions allowed for each user on the schema object (for example, user SCOTT can issue SELECT and INSERT statements but not DELETE statements using the employees table)

  •The actions, if any, that are audited for each schema object

•Data encryption to prevent unauthorized users from bypassing Oracle and accessing data

Security Mechanisms

The Oracle database provides discretionary access control, which is a means of restricting access to information based on privileges. The appropriate privilege must be assigned to a user in order for that user to access a schema object. Appropriately privileged users can grant other users privileges at their discretion.

Oracle manages database security using several different facilities:

•Authentication to validate the identity of the entities using your networks, databases, and applications

•Authorization processes to limit access and actions, limits that are linked to user's identities and roles.

•Access restrictions on objects, like tables or rows.

•Security policies

•Database auditing

Managing password security and resources

•Manage passwords using profiles.

•Administer profiles.

•Control use of resources using profiles.

•Obtain information about profiles, password management, and resources from the data dictionary.

Profiles are named sets of specific resource limits that can be assigned to any valid username or schema in an Oracle database. They provide a mechanism to more easily manage resource limits and password policies. This chapter provides information on using profiles to make your life as a DBA simpler.

Often it is necessary to carefully manage resource limits and user passwords. When this is the caseand let's face it, you aren't likely to be told that every user on your system should have free reign to use as much CPU and memory as she canyou can use profiles as a means to manage your resources from within the database. To use profiles, you first need to categorize related types of users in a database. Then determine how many

profiles are needed at this point (you can always change it later) to be able to manage and provide equitable resources for all your different types of users. Then you must determine appropriate resource limits for each profile. Now let's get into some more detail.

Manage Passwords Using Profiles

Profiles are named sets of passwords and resource limits. They are assigned to individual users using the CREATE USER or ALTER USER command and can be enabled or disabled. Everydatabase comes with at least one profile, the DEFAULT profile. With a profile, you can do the following:

•Set rules for password aging and expiration.

•Maintain password history so that you can manage reuse.

•Set rules for password complexity and verification.

•Set rules for account locking.

•Set limits on CPU time and I/O operations.

•Set limits on allowable idle time.

•Set limits on allowable connect time.

•Set limits on allowable concurrent sessions.

After a profile is created, the DBA assigns users to that profile. If resource limits are used, Oracle limits the database usage and resources to the defined profile.

DEFAULT Profile

When the database is created, Oracle automatically creates the DEFAULT profile. Users who have not been explicitly assigned to another profile will be assigned to DEFAULT. When DEFAULT is created, it has all its resource limits set to UNLIMITED. You, as the DBA, can change these values so that the limits in your environment reflect the default settings that you want to have applied to your average user.

Profile Usage

Profiles restrict sets of users from performing operations that require heavy use of resources. They ensure either that users log off the database when they are finished or that their session is logged off after it has been idle for a specified amount of time. You can group people and resources together based on similarities in need, a feature that is

particularly useful when managing large complex databases with many different kinds of users.

Profiles have the following characteristics

•Assigning a profile to a user does not affect the currently logged-in session.

•Profiles have to be assigned to users, not to roles or other profiles.

•If you don't assign another profile to a user when that user is created, the DEFAULT profile is automatically assigned.

•Any parameters not set within a profile will take on the values from the DEFAULT profile.

Passwords

Often, it is easier to exercise greater control over database security and password management by using profiles. You can identify similar characteristics that a given set of users has and determine password maintenance characteristics similar to those users.

Account Locking

Account locking enables the automatic locking of an account whenever a user fails to log in to the database after a specific number of attempts. Although it is true that users often forget passwords and need to be able to try a couple of alternatives before they get it right, it is also a fact that people trying to gain illicit access to a database may have a good guess at a userid but not know the password. An unlimited number of attempts may not be a wise security decision. Limiting a user to three or five invalid attempts limits your security exposure and provides users with a little guessing room for passwords they may have forgotten.

Two parameters assist the DBA with controlling account locking. FAILED_LOGIN_ATTEMPTS allows the account to be automatically locked after the value to which this parameter has been set is passed. The account is automatically locked and either can be set to automatically unlock after a specific amount of time has passed (set by the other parameter, PASSWORD_LOCK_TIME) or can be unlocked by the DBA with the ALTER USER command. If an account is locked using the ALTER USER command, it is never automatically unlocked.

Password Aging and Expiration

Password aging and expiration enable you to determine in advance a password's lifetime. After the expiration period, the password must be changed. This limits security exposure because a user who has had the same password for five years may have at

some point given his password to someone else. Forcing the password to be changed limits the length of time that there is a hole in the security.

Two parameters are available to help with maintaining password aging and expiration. PASSWORD_LIFE_TIME sets the maximum lifetime after which the password must bechanged (there is no minimum time that must pass before the user must change the password). You, as the DBA, can specify a grace period (specified by the PASSWORD_GRACE_TIME parameter). As soon as the PASSWORD_LIFE_TIME time has passed, at the next user login, PASSWORD_GRACE_TIME goes into effect. A warning message is generated every time the user tries to log in until the grace period has passed. The user is expected to change the password within the grace period. If the password is not changed within the grace period, the account is locked.

Password History

Password history allows you to preset an amount of time that must pass before a password can be reused. This can be set either as a specific amount of time, for example 90 days, or as a specific number of password changes, three or five or any number that you set.

Oracle provides two parameters that assist with password history maintenance. PASSWORD_REUSE_TIME specifies the given number of days that must pass before a user can reuse a previous password. PASSWORD_REUSE_MAX forces a user to arrive at a password that is not identical to earlier passwords. PASSWORD_REUSE_MAX specifies the number of password changes required before the current password can be reused. If you setPASSWORD_REUSE_MAX to an integer value, you have to set PASSWORD_REUSE_TIME to UNLIMITED.

Password Complexity Verification

Complexity verification ensures that a password fits with rules that you set up ahead of time that deal with how many characters must be in a password and the different configurations that can be used for passwords. (For example, a password must have at least one number and at least one non- number character, must not be the same or similar to your userid, must have at most one set of duplicated letters or numbers within the password, and so forth.)

When users change their passwords, before the password gets assigned, a PL/SQL function can be invoked to verify the validity of the password. Oracle provides a default verification routine, but the DBA can create an alternative PL/SQL function that is a customized way of checking the validity. The Oracle provided parameter, PASSWORD_VERIFY_FUNCTION, can be set to the custom PL/SQL function to enable the use of the custom program unit for password verification.

The Oracle provided password function takes the userid, new password, and old password as parameters and returns a Boolean value to specify its validity. The characteristics that the VERIFY_FUNCTION (the Oracle supplied SYS function for password verification) looks for follow:

•Minimum length of password is four characters.

•Password should not be equal to username.

•Password should have at least one alphabetic, one numeric, and one special character.

Password should differ from immediately previous password by at least three letters.

To use the VERIFY_FUNCTION, you need to have run the utlpwdmg.sql script as the SYS user. This script changes the DEFAULT profile and sets the following defaults:

•PASSWORD_LIFE_TIME = 60

•PASSWORD_GRACE_TIME = 10

•PASSWORD_REUSE_TIME = 1800

•PASSWORD_REUSE_MAX = UNLIMITED

•FAILED_LOGIN_ATTEMPTS = 3

•PASSWORD_LOCK_TIME = 1/1440

•PASSWORD_VERIFY_FUNCTION = verify_function

Whether or not password management is enabled in a profile, the user account can be locked or unlocked with the CREATE USER or ALTER USER command. This is the same method used to assign profiles to users, and password limits are always enforced reliably across users.

Emerging Database Technologies and Application

- Mobile Databases
- Multimedia Databases
- Geographic Information Systems
- GENOME Data Management

1. **Mobile Databases**

- ✓ Recent advances in portable and wireless technology led to mobile computing, a new dimension in data communication and processing.
- ✓ Portable computing devices coupled with wireless communications allow clients to access data from virtually anywhere and at any time.
- ✓ There are a number of hardware and software problems that must be resolved before the capabilities of mobile computing can be fully utilized.
- ✓ Some of the software problems – which may involve data management, transaction management, and database recovery – have their origins in distributed database systems.

In mobile computing, the problems are more difficult, mainly:

- ✓ The limited and intermittent connectivity afforded by wireless communications.
- ✓ The limited life of the power supply (battery).
- ✓ The changing topology of the network.
- ✓ In addition, mobile computing introduces new architectural possibilities and challenges.

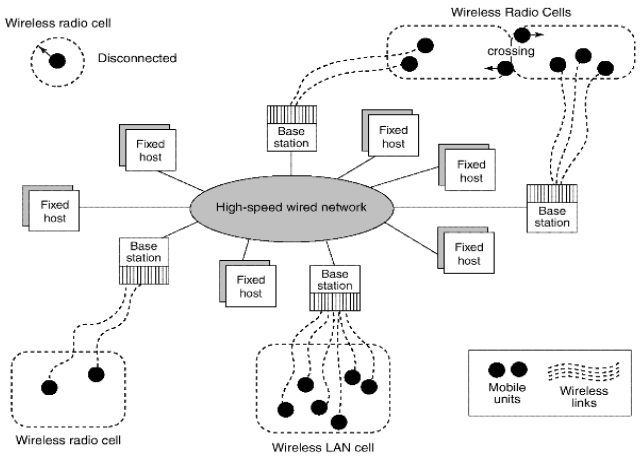


Figure 5.3 Mobile Computing Architecture

2. **Multimedia Databases**

In the years ahead multimedia information systems are expected to dominate our daily lives.

- ✓ Our houses will be wired for bandwidth to handle interactive multimedia applications.
- ✓ Our high-definition TV/computer workstations will have access to a large number of databases, including digital libraries, image and video databases that will distribute vast amounts of multisource multimedia content.

Types of multimedia data are available in current systems

- ✓ **Text:** May be formatted or unformatted. For ease of parsing structured documents, standards like SGML and variations such as HTML are being used.
- ✓ **Graphics:** Examples include drawings and illustrations that are encoded using some descriptive standards (e.g. CGM, PICT, postscript).
- ✓ **Images:** Includes drawings, photographs, and so forth, encoded in standard formats such as bitmap, JPEG, and MPEG. Compression is built into JPEG and MPEG.
    - o These images are not subdivided into components. Hence querying them by content (e.g., find all images containing circles) is nontrivial.
- ✓ **Animations:** Temporal sequences of image or graphic data.
- ✓ **Video:** A set of temporally sequenced photographic data for presentation at specified rates– for example, 30 frames per second.
- ✓ **Structured audio:** A sequence of audio components comprising note, tone, duration, and so forth
- ✓ **Composite or mixed multimedia data:** A combination of multimedia data types such as audio and video which may be physically mixed to yield a new storage format or logically mixed while retaining original types and formats. Composite data also contains additional control information describing how the information should be rendered

Fig 5.4 Multimedia Data

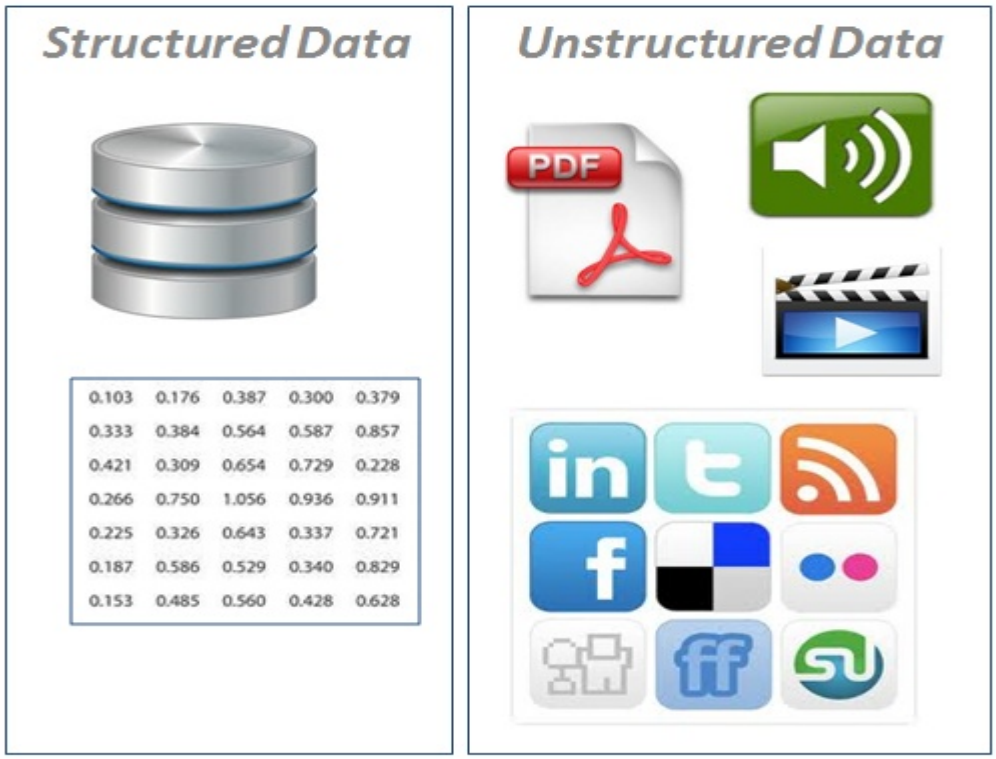


Fig 5.5 Structured and Unstructured Data

Applications based on their data management characteristics:

- ✓ **Repository applications:** A large amount of multimedia data as well as metadata is stored for retrieval purposes. Examples include repositories of satellite images, engineering drawings and designs, space photographs, and radiology scanned pictures.
- ✓ **Presentation applications:** A large amount of applications involve delivery of multimedia data subject to temporal constraints; simple multimedia viewing of video data, for example, requires a system to simulate VCR-like functionality.
- ✓ **Collaborative work using multimedia information:** This is a new category of applications in which engineers may execute a complex design task by merging drawings, fitting subjects to design constraints, and generating new documentation, change notifications, and so forth. Intelligent healthcare networks as well as telemedicine will involve doctors collaborating among themselves, analyzing multimedia patient data and information in real time as it is generated.

Multimedia Database Applications

- ✓ **Large-scale applications of multimedia databases can be expected encompasses a large number of disciplines and enhance existing capabilities.**
- ✓ **Documents and records management**
- ✓ **Knowledge dissemination**
- ✓ **Education and training**
- ✓ **Marketing, advertising, retailing, entertainment, and travel**
- ✓ **Real-time control and monitoring**

3. **Geographic Information System**

The scope of GIS broadly encompasses two types of data:

- ✓ Spatial data, originating from maps, digital images, administrative and political boundaries, roads, transportation networks, physical data, such as

rivers, soil characteristics, climatic regions, land elevations, and

- ✓ Non-spatial data, such as socio-economic data (like census counts), economic data, and sales or marketing information. GIS is a rapidly developing domain that offers highly innovative approaches to meet some challenging technical demands.

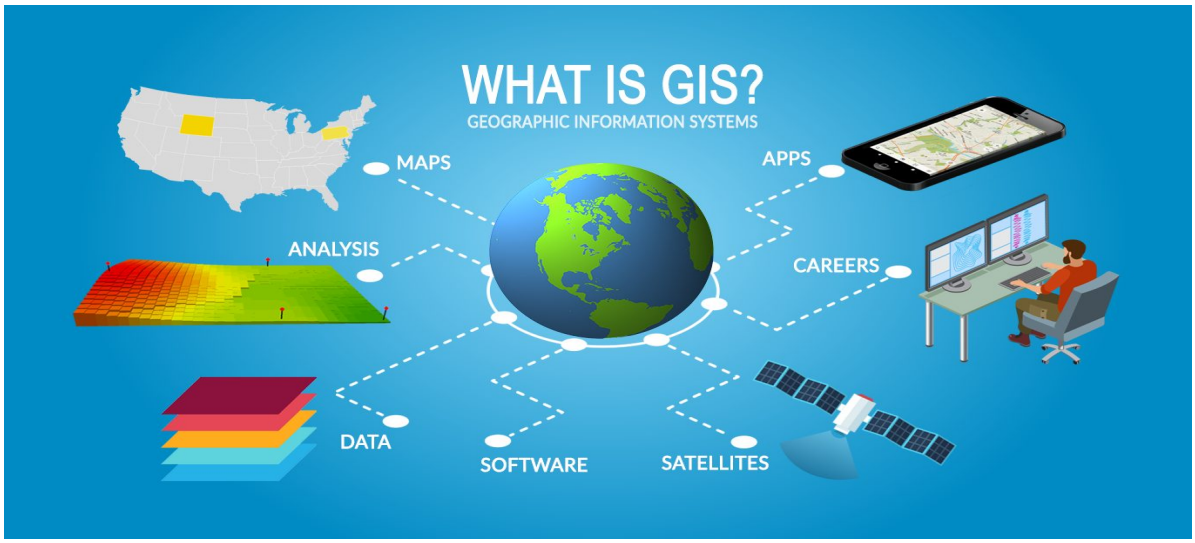


Fig 5.6 GIS

Categorization of GIS:

- Cartographic applications
- Digital terrain modelling applications
- Geographic objects applications

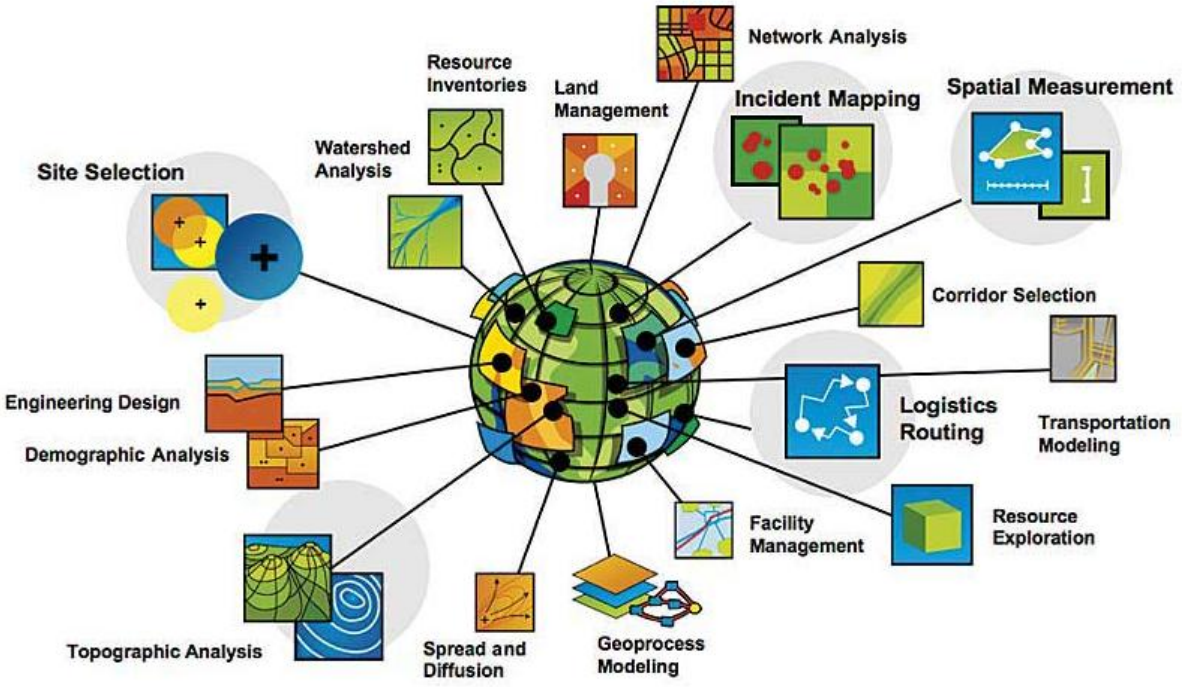


Fig 5.7 Categorization of GIS

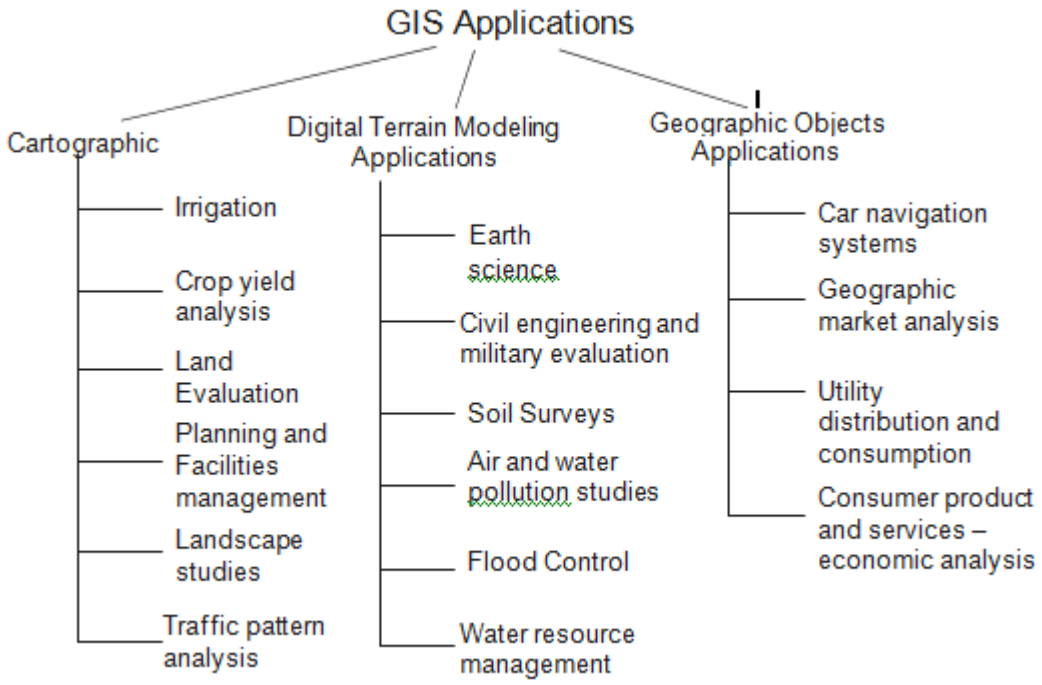


Fig 5.8 GIS Applications

Data Warehousing

Introduction

A data warehouse is constructed by integrating data from multiple heterogeneous sources. It supports analytical reporting, structured and/or ad hoc queries and decision making. This tutorial adopts a step-by-step approach to explain all the necessary concepts of data warehousing. The term "Data Warehouse" was first coined by Bill Inmon in 1990.

According to Inmon, a data warehouse is a subject oriented, integrated, time-variant, and non-volatile collection of data. This data helps analysts to take informed decisions in an organization. An operational database undergoes frequent changes on a daily basis on account of the transactions that take place. Suppose a business executive wants to analyze previous feedback on any data such as a product, a supplier, or any consumer data, then the executive will have no data available to analyze because the previous data has been updated due to transactions. A data warehouses provides us generalized and consolidated data in multidimensional view. Along with generalized and consolidated view of data, a data warehouses also provides us Online Analytical Processing (OLAP) tools. These tools help us in interactive and effective analysis of data in a multidimensional space. This analysis results in data generalization and data mining. Data mining functions such as association, clustering, classification, prediction can be integrated with OLAP operations to enhance the interactive mining of knowledge at multiple level of abstraction. That's why data warehouse has now become an important platform for data analysis and online analytical processing.

Understanding a Data Warehouse

- A data warehouse is a database, which is kept separate from the organization's operational database.
- There is no frequent updating done in a data warehouse.
- It possesses consolidated historical data, which helps the organization to analyze its business.
- A data warehouse helps executives to organize, understand, and use their data to take

strategicdecisions.

- Data warehouse systems help in the integration of diversity of application systems.
- A data warehouse system helps in consolidated historical data analysis.

Why a Data Warehouse is Separated from Operational Databases

A data warehouses is kept separate from operational databases due to the following reasons:

- An operational database is constructed for well-known tasks and workloads such as searching particular records, indexing, etc. In contract, data warehouse queries are often complex and they present a general form of data.
- Operational databases support concurrent processing of multiple transactions. Concurrency control and recovery mechanisms are required for operational databases to ensure robustness and consistency of the database.
- An operational database query allows to read and modify operations, while an OLAP query needs only **read only** access of stored data.
- An operational database maintains current data. On the other hand, a data warehouse maintains historical data.

Data Warehouse Features

The key features of a data warehouse are discussed below:

- **Subject Oriented** - A data warehouse is subject oriented because it provides information around a subject rather than the organization's ongoing operations. These subjects can be product, customers, suppliers, sales, revenue, etc. A data warehouse does not focus on the ongoing operations; rather it focuses on modelling and analysis of data for decision making.
- **Integrated** - A data warehouse is constructed by integrating data from heterogeneous sources such as relational databases, flat files, etc. This integration enhances the effective analysis of data.
- **Time Variant** - The data collected in a data warehouse is identified with a particular time period. The data in a data warehouse provides information from the historical point of view.

- **Non-volatile** - Non-volatile means the previous data is not erased when new data is added to it. A data warehouse is kept separate from the operational database and therefore frequent changes in operational database is not reflected in the data warehouse.

  Note: A data warehouse does not require transaction processing, recovery, and concurrency controls, because it is physically stored and separate from the operational database.

  Data Warehouse Applications - As discussed before, a data warehouse helps business executives to organize, analyze, and use their data for decision making. A data warehouse serves as a sole part of a plan-execute-assess "closed-loop" feedback system for the enterprise management. Data warehouses are widely used in the following fields:

  - Financial services
  - Banking services
  - Consumer goods
  - Retail sectors
  - Controlled manufacturing

Types of Data Warehouse

Information processing, analytical processing, and data mining are the three types of data warehouse applications that are discussed below:

- **Information Processing** - A data warehouse allows to process the data stored in it. The data can be processed by means of querying, basic statistical analysis, reporting using crosstabs, tables, charts, or graphs.
- **Analytical Processing** - A data warehouse supports analytical processing of the information stored in it. The data can be analyzed by means of basic OLAP operations, including slice-and-dice, drill down, drill up, and pivoting.
- **Data Mining** - Data mining supports knowledge discovery by finding hidden patterns and associations, constructing analytical models, performing classification and prediction. These mining results can be presented using the visualization tools.

No.	Data Warehouse (OLAP)	Operational Database(OLTP)
1	It involves historical processing of information.	It involves day-to-day processing.
2	OLAP systems are used by knowledge workers such as executives, managers, and analysts.	OLTP systems are used by clerks, DBAs, or database professionals.
3	It is used to analyze the business.	It is used to run the business.
4	It focuses on Information out.	It focuses on Data in.
5	It is based on Star Schema, Snowflake Schema, and Fact Constellation Schema.	It is based on Entity Relationship Model.
6	It focuses on Information out.	It is application oriented.
7	It contains historical data.	It contains current data.
8	It provides summarized and consolidated data.	It provides primitive and highly detailed data.
9	It provides summarized and multidimensional view of data.	It provides detailed and flat relational view of data.
10	The number of users is in hundreds.	The number of users is in thousands.
11	The number of records accessed is in millions.	The number of records accessed is in tens.
12	The database size is from 100GB to 100 TB.	The database size is from 100 MB to 100 GB.
13	These are highly flexible.	It provides high performance.

Data Mining

Data mining (sometimes called data or knowledge discovery) is the process of analyzing data from different perspectives and summarizing it into useful information - information that can be used to increase revenue, cuts costs, or both. Data mining software is one of a number of analytical tools for analyzing data. It allows users to analyze data from many different dimensions or angles, categorize it, and summarize the relationships identified. Technically, data mining is the process of finding correlations or patterns among dozens of fields in large relational databases.

Data Mining is defined as extracting information from huge sets of data. In other words, we can say that data mining is the procedure of mining knowledge from data. The information or

knowledge extracted so can be used for any of the following applications −

- Market Analysis
- Fraud Detection
- Customer Retention
- Production Control
- Science Exploration

Data Mining Applications

Data mining is highly useful in the following domains −

- Market Analysis and Management
- Corporate Analysis &Risk Management
- FraudDetection

Apart from these, data mining can also be used in the areas of production control, customer retention, science exploration, sports, astrology, and Internet Web Surf-Aid.

Market Analysis and Management

Listed below are the various fields of market where data mining is used −

- **Customer Profiling** − Data mining helps determine what kind of people buy what kind ofproducts.
- **Identifying Customer Requirements** − Data mining helps in identifying the best products for different customers. It uses prediction to find the factors that may attract new customers.
- **Cross Market Analysis** − Data mining performs association/correlations between productsales.
- **Target Marketing** − Data mining helps to find clusters of model customers who share the same characteristics such as interests, spending habits, income,etc.
- **Determining Customer purchasing pattern** − Data mining helps in determining customer purchasingpattern.
- **Providing Summary Information** − Data mining provides us various

multidimensional summaryreports.

Corporate Analysis and RiskManagement

Data mining is used in the following fields of the Corporate Sector−

- **Finance Planning and Asset Evaluation** − It involves cash flow analysis and prediction, contingent claim analysis to evaluate assets.
- **Resource Planning** − It involves summarizing and comparing the resources and spending.
- **Competition** − It involves monitoring competitors and marketdirections.

Fraud Detection

Data mining is also used in the fields of credit card services and telecommunication to detect frauds. In fraud telephone calls, it helps to find the destination of the call, duration of the call, time of the day or week, etc. It also analyzes the patterns that deviate from expected norms.

Knowledge discovery in databases (KDD)

Knowledge discovery in databases (KDD) is the process of discovering useful knowledge from a collection of data. This widely used data mining technique is a process that includes data preparation and selection, data cleansing, incorporating prior knowledge on data sets and interpreting accurate solutions from the observed results.

Here is the list of steps involved in the knowledge discovery process −

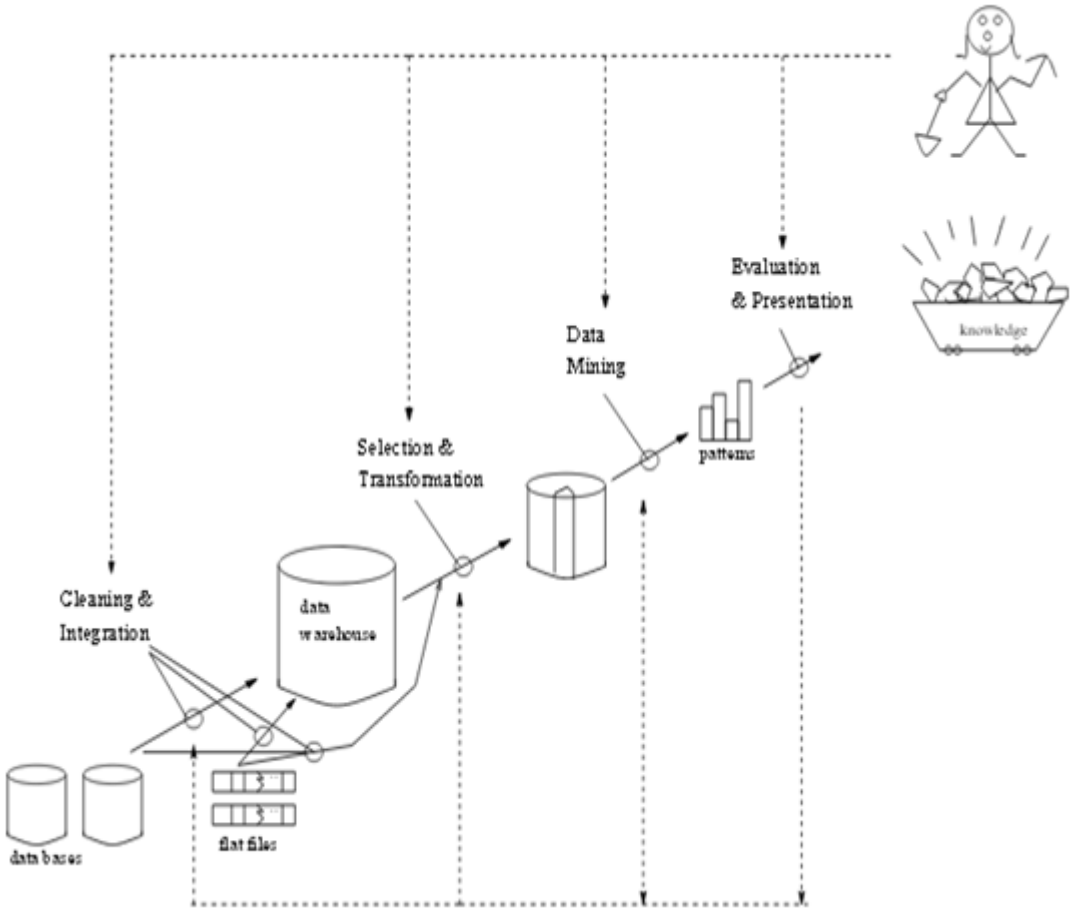


Fig. 5.9 Data Mining as a process of knowledge discovery

- **Data Cleaning** − In this step, the noise and inconsistent data isremoved.
- **Data Integration** − In this step, multiple data sources arecombined.
- **Data Selection** − In this step, data relevant to the analysis task are retrieved from the database.
- **Data Transformation** − In this step, data is transformed or consolidated intoforms appropriate for mining by performing summary or aggregationoperations.
- **Data Mining** − Inthis step, intelligent methods are applied in order to extract data patterns.
- **Pattern Evaluation** − In this step, data patterns areevaluated.

- **Knowledge Presentation** − Inthis step, knowledge isrepresented.

Data Mining Applications

Here is the list of areas where data mining is widely used −

- Financial Data Analysis
- Retail Industry
- Telecommunication Industry
- Biological Data Analysis
- Other Scientific Applications
- Intrusion Detection

Financial Data Analysis

The financial data in banking and financial industry is generally reliable and of high quality which facilitates systematic data analysis and data mining. Some of the typical cases are as follows −

- Design and construction of data warehouses for multidimensional data analysis and data mining.
- Loan payment prediction and customer credit policy analysis.
- Classification and clustering of customers for targeted marketing.
- Detection of money laundering and other financial crimes.

Retail Industry

Data Mining has its great application in Retail Industry because it collects large amount of data from on sales, customer purchasing history, goods transportation, consumption and services. It is natural that the quantity of data collected will continue to expand rapidly because of the increasing ease, availability and popularity of the web.

Data mining in retail industry helps in identifying customer buying patterns and trends that lead to improved quality of customer service and good customer retention and satisfaction. Here is the list of examples of data mining in the retail industry

- Design and Construction of data warehouses based on the benefits of data mining.
- Multidimensional analysis of sales, customers, products, time and region.
- Analysis of effectiveness of sales campaigns.
- Customer Retention.
- Product recommendation and cross-referencing of items.

Telecommunication Industry

Today the telecommunication industry is one of the most emerging industries providing various services such as fax, pager, cellular phone, internet messenger, images, e-mail, web data transmission, etc. Due to the development of new computer and communication technologies, the telecommunication industry is rapidly expanding. This is the reason why data mining is become very important to help and understand the business.

Data mining in telecommunication industry helps in identifying the telecommunication patterns, catch fraudulent activities, make better use of resource, and improve quality of service. Here is the list of examples for which data mining improves telecommunication services −

- Multidimensional Analysis of Telecommunication data.
- Fraudulent pattern analysis.
- Identification of unusual patterns.
- Multidimensional association and sequential patterns analysis.
- Mobile Telecommunication services.
- Use of visualization tools in telecommunication data analysis.

Biological Data Analysis

In recent times, we have seen a tremendous growth in the field of biology such as genomics, proteomics, functional Genomics and biomedical research. Biological data mining is a very important part of Bioinformatics. Following are the aspects in which data mining contributes for biological data analysis −

- Semantic integration of heterogeneous, distributed genomic and proteomic databases.
- Alignment, indexing, similarity search and comparative analysis multiple nucleotide sequences.
- Discovery of structural patterns and analysis of genetic networks and protein pathways.
- Association and path analysis.
- Visualization tools in genetic data analysis.

Other Scientific Applications

The applications discussed above tend to handle relatively small and homogeneous data sets for which the statistical techniques are appropriate. Huge amount of data have been collected from scientific domains such as geosciences, astronomy, etc. A large amount of data sets is being generated because of the fast numerical simulations in various fields such as climate and ecosystem modeling, chemical engineering, fluid dynamics, etc. Following are the applications of data mining in the field of Scientific Applications −

- Data Warehouses and data preprocessing.
- Graph-based mining.
- Visualization and domain specific knowledge.

Intrusion Detection

Intrusion refers to any kind of action that threatens integrity, confidentiality, or the availability of network resources. In this world of connectivity, security has become the major issue. With increased usage of internet and availability of the tools and tricks for intruding and attacking

network prompted intrusion detection to become a critical component of network administration. Here is the list of areas in which data mining technology may be applied for intrusion detection −

- Development of data mining algorithm for intrusion detection.
- Association and correlation analysis, aggregation to help select and build discriminating attributes.
- Analysis of Stream data.
- Distributed data mining.
- Visualization and query tools.

Trends in Data Mining

Data mining concepts are still evolving and here are the latest trends that we get to see in this field −

- Application Exploration.
- Scalable and interactive data mining methods.
- Integration of data mining with database systems, data warehouse systems and web database systems.
- Standardization of data mining query language.
- Visual data mining.
- New methods for mining complex types of data.
- Biological data mining.
- Data mining and software engineering.
- Web mining.
- Distributed data mining.

- Real time data mining.
- Multi database data mining.
- Privacy protection and information security in data mining.