Advanced Database (ADBMS)

By

Bhupendra Singh Saud

For B.Sc. CSIT 7th Semester TU

Course of Contain

Unit 1: The Relational Model of Data and RDBMS Implementation Techniques [5 Hrs.]
Theoretical concepts, Relational model conformity and Integrity, Advanced SQL programming, Query optimization, Concurrency control and Transaction management, Database performance tuning, Distributed relational systems and Data Replication, Security considerations.

Unit 2: The Extended Entity Relationship Model and Object Model [6 Hrs.]

The ER model revisited, Motivation for complex data types, User defined abstract data types and structured types, Subclasses, Super classes, Inheritance, Specialization and Generalization, Constraints and Characteristics of specialization and Generalization, Relationship types of degree higher than two, Relational database design by EER- to relational mapping, basic concepts on UML.

Unit 3: Emerging Database Management System Technologies [18 Hrs.]
Object Oriented Database concepts: object identity, structure, and type constructors; encapsulation of operations, methods, and persistence; type and class hierarchies and inheritance; structures and unstructured complex objects and type extensibility; polymorphism, multiple inheritance and selective inheritance, versions and configurations; Object Relational Database concepts: overview of SQL and its object-relational features (the SQL standard and its components, object-relational support in SQL-99); evolution and current trends of database technology (with respect to the features of the Informix Universal Server and Oracle8); implementation and related issues for extended type systems; the nested relational model; Active database concepts: Generalized model for active databases and oracle Triggers; design and implement issues for active databases potential applications for active databases; Temporal database concepts: Time representation, calendars, and time dimensions; incorporating time in relational databases using tuple versioning, incorporating time in object-oriented databases using attribute versioning, time series data; Multimedia Databases: The nature of multimedia data and applications; spatial database concepts and architecture, introduction to multimedia database concepts; Deductive databases and Query

processing: Prolog/Data log notations, clausal form and horn clauses; interpretations of rules; Mobile Databases: Mobile computing architecture, characteristics of mobile environments, data management issues; Geographic Information Systems: GIS applications, data management requirements of GIS, specific GIS data operations;

Unit 4: New database applications and environments [8 Hrs.]
Data Mining: Overview of data mining technology (associated rules, classification, clustering), applications of data mining; Data Warehousing: Overview of data warehousing, typical functionality of a data warehouse;

Unit 5: Database Related Standards [8 Hrs.]
SQL standards, SQL 1999, SQL 2003, Object Data Management Group (ODMG) version 3.0 standards (ODL, OQL), Standards for interoperability and integration e.g. Web Services, SOAP, XML related specifications, e.g. XML Documents, DTD, XML Schema, X-Query, XPath.

Prerequisite:

- Be familiar with at least one OO Programming language such as .Net or C++ or Java,
- Fundamentals of DBMS, SQL

Reference Books:

1. Elmasri and Navathe, Fundamentals of Database Systems, Pearson Education
2. Raghu Ramakrishnan, Johannes Gehrke, Database Management Systems, McGraw-Hill
3. Korth, Silberchatz, Sudarshan, Database Systems, Design, Implementation and Management, Thomson Learning
4. C.J. Date & Longman, Introduction to Database Systems, Pearson Education

Computer Usage:

Windows or Linux based PC or workstation, Commercial OODBMS software package and MVC software development framework installed at the server.

Unit 1

Theoretical Concept of overall database

Data

The collection of raw facts is called data.

File

A file is the collection of related groups of data for example, payroll file of a company consists of the salary detail and all of these records have the same heads (e.g.; basic pay, HRA, FA etc.).

Records

A file may be further divided into more descriptive subdivisions, called, records. In other words a record is a collection of related data items as a single unit. It is also called column of table.

Tuples

The row of a table is called tuples. It is also called value of a table.

Fields

The column of a table is called fields.

Database

A database is a collection of related data necessary to manage an organization. By data, we mean known facts that can be recorded and have implicit meaning. For example, consider names, telephone numbers and addresses of the people. We may have recorded this data in an indexed address book, or we may have recorded on the hard drive, using a personal computer and software such as MS-Access, or Excel. This is the collection of related data with an implicit meaning and hence is a database. A database is logically coherent collection of data with some inherent meaning. A database is designed, built and populated with data for specific purpose. It excludes transient data such as: input documents, reports and intermediate results obtained during processing.

DBMS

A database management system is a set of procedures that manages the database and provide access to the database in the form required by any application program. It effectively ensures that necessary data in the desired form is available for diverse applications for different departments in an organization. A DBMS is hence a general purpose software system that facilitates the processes of defining, constructing, manipulating, and sharing database among various users and applications. Fig 1 illustrate the difference between database and database management system.

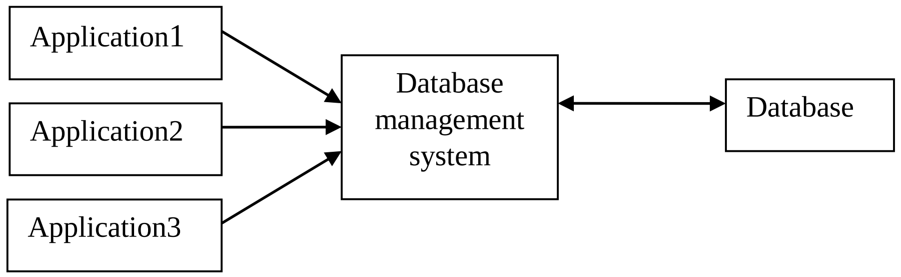


Fig 1 illustrates the distinction between a DBMS and a Database.

A database management system (DBMS) is a collection of programs that control the database. The primary goal of DBMS is store and manages the data conveniently and efficiently.

Why data management is Important?

One of the most valuable resources of any business is the data stored by the business. The efficient management of data is a key business activity. A systematic approach to data management allows data to became information

When data becomes information

When data is organized in a meaningful way, called information that may not have been easily observable becomes apparent for example total sales buying trends for February 2002 etc.

- Effective data management enables raw data to become useful information.
- Data +semantics(Rules) = information
- Any random collection of data is not a database.

Example uses of DBMS

- Airlines – reservations and schedules etc.
- Banking – customer information and accounts etc.
- Universities – student information, grades etc.
- Government – taxes, budgets, census etc.
- Sales – inventory, customer info etc.
- Credit cards – transactions, customer info etc.
- Newspapers – track subscribers, advertising revenue.
- Financial – stock prices, portfolio info.
- Telecommunications – records of calls made.

History

- Integrated data store, first general purpose DBMS, early 1960s, Charles Bachman, general electrics.
- Information management system (IMS), late 1960s, IBM.
- Relational database model, proposed in 1970 Edgar Codd, IBM's san Jose laboratory.
- Structured Query language (SQL) standardized in the late 1980s.
- More powerful query language, complex analysis of data, support for new data types late 1980s to 1990s.
- Packages which came with power customizable application layers.
- Internet.

Objectives of Database Management

1. **Sharing of data**
   The main motivation for designing a system to manage a database in shareability of data. Data is considered as an important resource of an organization to be shared by a number of application .It gives the sharing of data.
2. **Data integrity**
   The term integrity means to the validity of data contained in database. Data integrity can be reduced in many ways, including input typing errors, hardware malfunctions and data transmission errors. To avoid data integrity errors, database program should use validation procedures, which define acceptable input ranges for each field in the record.
3. **Data independence**
   The term data independence refers to the storage of data in such way that it is not locked into use by the particular application. To ensure data independence, it is important to avoid software that uses file storage techniques.
4. **Avoid data redundancy**
   The system should identify existence of common data and avoid duplicate recording. The process of avoiding duplicate data is called data redundancy. Selective redundancy is sometimes allowed to improve performance or for better reliability.
5. **Data Security**
   This is concerned with protecting access to data. Protection is needed at many levels for access, modification, deletion or access.
6. **Data maintenance**
   Good database management involves having system in place for data maintenance. This system includes procedure for adding records, updating records and deleting records.
7. **Management control**

As the dependence of an organization on the database increase, positive management controls should be exercised over addition, deletion, change and disposition of data. Data must be protected to satisfy legal, accounting and auditing requirements.

Purpose of database systems

Using Database system, we can easily insert, delete and update data from the database. We can easily retrieve required result. In a file processing system, permanent records are stored in various files, and different application programs are written to extract records from, and to add records to, the appropriate files Before DBMS system, organization used to store information using file processing system. The main purpose of database is to remove the difficulties of file processing system, which are given below.

Drawbacks of using file systems to store data

1. **Data redundancy and inconsistency**

   The few same data files are often included in many different files, in file processing system. The redundancy in storing the same data multiple time leads to several problems. First, there is the need to perform a single logical update- such as entering data on a new student multiple time: once for each file where student data is recorded. This leads to duplication of effort. Second, storage space is wastage when the same data is stored repeatedly and this problem may be serious for large database. Third files that represent the same data may become inconsistent. This may happen because an update is applied to some of the files but not to others. Even if an update adding a new student is applied to all the appropriate files, the data concerning the student may still be inconsistent because the updates are applied independently by each user group. For example, one user group may enter a student's birth date erroneously as JAN-19- 1984, whereas the other user groups may enter the correct value of JAN – 29 -1984.

2. **Integrity problems**

   Most database applications have certain integrity constraints that must hold for the data. A DBMS should provide capabilities for defining and enforcing these constraints. The data in a database satisfy certain types of the consistency constraints. For example the balances of the bank account never fall below Rs. 500. The file system does not have this facility.

3. **Data isolation**

   In file processing system, data are scattered in various files, and files may be in different formats. It is difficult to write new application programs to retrieve the appropriate data.

4. **Atomicity problem**

In computer, any electrical or mechanical system may fail. When failure has occurred, it is ensured that has been detected and restored to the consistent state that existed prior to the failure. This problem cannot solved by file processing system for example a program to transfer Rs1000 from account A to B. if the system failure occurs during the execution of program, it is possible that Rs1000 was removed from account A but was not credited to account B, resulting in an inconsistent database state. It is essential to database consistency that either both the credit and debit occur or that neither occurs i.e. funds transfer must be atomic. It must happen in its entirety or not at all .it is difficult to ensure this property in a file processing system.

5. **Difficulty in accessing data**

   Database systems must provide capabilities for efficiently executing queries and updates. Because the database is stored on disk the DBMS must provide specialized data structures to speed up disk search for the desired records. Auxiliary files called indexes are used for this purpose. Using DBMS, we can access the required result from the whole database. File processing environment do not allow needed data to be retrieved in a convenient and efficient manner.

6. **Concurrent access anomalies**

   When the number of systems runs concurrently, the result may be inconsistent data. For example, suppose account, A, containing Rs5000, if two customers withdraw Rs400 and Rs500 at about the same time the result of concurrent executions may leave the account in an incorrect state. Both balances may read the amount Rs5000 and Rs4500, respectively these types of problem is solved by database.

7. **Security problem**

   In file processing system there is no security method. But in database system unauthorized person cannot see the data. For example in bank account, there may be number of accounts and only access the information about particular customers.

Advantages of DBMS approach

- Controlled redundancy.
- Restricting unauthorized access.
- Providing persistent storage for program objects.
- Providing storage structures for efficient Query processing.
- Providing backup and recovery.

- Providing multiple user interfaces.
- Representing complex relationships among data.
- Enforcing integrity constraints.

When a DBMS is needed?

- To integrate efficiently and correctly large volumes of related data
- Supports sharing and discovering of information
    - Data mining, deductive databases, active databases
- DBMS ensure true QoS (Quality of Services)

When a DBMS is Inappropriate?

Disadvantages:

- Expensive to buy
- Expensive to maintain (need administrator)
- "Expensive" to run (needs significant resources)

Hence, it is "over-kill" when

- The database is small
- The database has simple structure
- The application is simple and not expected to change
- Can tolerate failures
- Do not require multiple, concurrent users

DBMS Languages

- **Data Definition Language (DDL):** Used by the DBA and database designers to specify the conceptual schema of a database. In many DBMSs, the DDL is also used to define internal and external schemas (views). In some DBMSs, separate storage definition language (SDL) and view definition language (VDL) are used to define internal and external schemas.
- **Data Manipulation Language (DML**): Used to specify database retrievals and updates.
    - DML commands (data sublanguage) can be embedded in a general-purpose programming language (host language), such as COBOL, C or an Assembly Language.
    - Alternatively, stand-alone DML commands can be applied directly (query language).

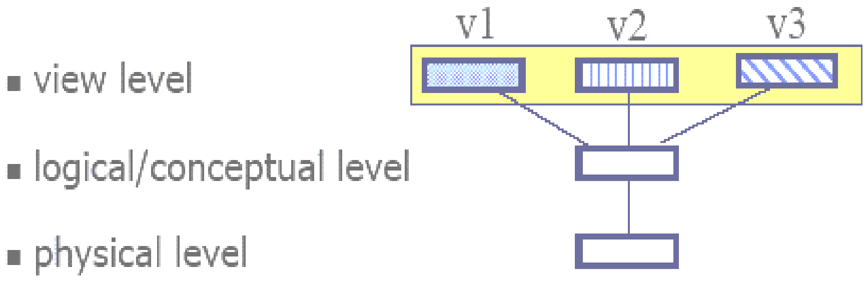
- **High Level or Non-procedural Languages:** e.g., SQL, are set-oriented and specify what data to retrieve than how to retrieve. Also called declarative languages.
- **Low Level or Procedural Languages:** record-at-a-time; they specify how to retrieve data and include constructs such as looping.

View of data

The main purpose of a database is to provide users with an abstract view of the data, i.e. the system hides certain details of how the data are stored and maintained. This is called data abstraction.

Three level architecture (ANSI / SPARC Architecture)

- View level.
- Logical / conceptual level.
- Physical level.



1. **View level (external level)**
   It is the highest level of data abstraction. It describes only part of the entire database. Different users may view the data in different ways. This may involve viewing data in different formats or derived values which are not stored persistently in the database only the data relevant to the particular user need be included in the user's view. eg .CS majors, math major.

2. **Logical level (conceptual level);**
   It describes what data are stored in database and relationship among these data. The entire database schema is described in terms of small number of relatively small structures. The logical structures of the entire database described by the conceptual schema. It includes all data entities attributes and relationship, integrity constrains are defined. All external view are derivable from conceptual schema it is independent of any storage details e.g.
   Course (course no, course name, credits, dept)
   Student (student id Lname, Fname, class, maior)
   Grade-report (students ID, course No, Grade, tem

3. **Physical level (Internal level)**
   It is the lower level of abstraction, describes how data are stored. It describes the physical storage characteristics of the data. Such as storage space allocation, indexes, It is independent of any particular application software. It also describes how the tables are stored, how many bytes / attributes it allocated etc.

Advantages of the 3-schema representation

- Each user/application should have their customized view of the data and should be able to change this view without affecting other users.
- Change to internal level should not affect user views.
- Change to physical storage device should not affect other levels.
- Changes to conceptual level should only affect those users/ applications accessing the changed data.

Database schema

It is the structure of the DB that captures data types, relationships, constraints on the data. It is independent of any application program. It may change frequently. It is similar to types and variables in programming language.

Database instances or state

It is the actual data in the database at a particular moment in time. It is analogous to the value of a variable in programming language.

Mapping

Conceptual / Internal Mapping

It defines the correspondence between the conceptual view and the stored database. It specifies how conceptual records and file are represented at the internal level. If a change is made to the storage structure definition then the conceptual / internal mapping must be changed accordingly so that the conceptual schema can remain invariant. In other words, the effect of such changes must be isolated below the conceptual level, in order to preserve physical data independence.

External / conceptual mapping

It defines the correspondence between a particular external view and the conceptual view. For example, fields can have different data types, fields and records name can be changed several conceptual fields can be combined into a single external fields and so on. Any number of external views can exist at the same time any number of users can share a given external view, different external views can overlap.

Data independence

When a schema at a lower level is changed, only the mapping between this schema and higher level schemas need to be changed in a DBMS that fully supports data independence. The higher level schemas themselves are unchanged. Hence, the application programs need be changed since they refer to the external schemas logical data independence. The three schema architecture can be used to further explain the concept of data independence, which can be defined as the capacity to change the schema at one level of database system without having to change the schema at the next higher level. There are two types of data dependence.

Types of data independence

Logical data independence

The ability to modify the logical schema without changing the view level and application program depends on the logical schema. We may change the conceptual schema to expand the database (by adding record type or data item), to change constraints, or to reduce database (by removing record type or data item).

Physical data independence

The ability to modify the physical schema without changing the logical schema is called physical data independence. Hence the external schema need not be changed as well. Changes to the internal schema may be needed because some physical files had to be recognized. For example, by creating additional access structure – to improve the performance of retrieval or update. If the same data as before remains in the database, we should not have to change the conceptual schema.

Logical data independence is more difficult to achieve than in physical data independence, since the application programs are heavily dependent on the logical structure of data that they access.

The concept of data independence is similar in many respects to the concept of abstract data types in modern programming language. Both hide implementation details from the users to concentrate on the general structure, rather than on low level implementation details.

Data dictionary

The result of compilation of DDL statement is a set of tables that is stored in a special file called data dictionary. A data dictionary is a file that contains metadata that is data about data. This file is consulted before actual data are read or modified in a database system.

Data model

A collection of conceptual tools for describing data, data relationships, data semantics and consistency constraints is called data model. The various data model that have been proposed fall into three different groups.

- Object based logical models.
- Record based logical models.
- Physical model

Object based logical model

Object based logical models are used in describing data at the logical and view levels. They are characterized by the fact that they provide fairly flexible structuring capability and allow data constraints to be specified explicitly. There are many different object based logical models.

- The entity relationship model.
- The object oriented model.
- The semantic data model.
- The functional data model.

The Entity Relationship model

The entity relationship model(ER model) is based on a perception of a real world that consists of a collection of basic objects, called entities, and of relationship among these objects and characteristics of an entity.

Entity-Relationship (E-R) diagram

The diagrammatic representation of entities attributes and their relationship is described by E-R diagram. The E-R diagram is an overall logical structure of a database that can be expressed graphically. It was developed to facilitated database design and the simplicity and pictorial clarity of this diagramming technique have done great help in the designing part of database. The basic components of ER diagram are given below.

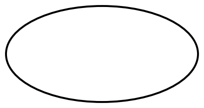
Entity:

An entity is a “thing” or “object” in the real word that is distinguishable from other objects. For example each person is an entity. An entity has a set of properties and the values for some set of properties may uniquely identify an entity. For example, if student is an entity, is identified by registration number. It is represented by rectangle.

Entity

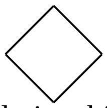
Attribute:

Attributes are properties possessed by on entity or relationship. For example stu_no, stu_namestu_sub are the attributes of the entity student. Attribute is represented by ellipse.


Attribute

Relationship:

A relationship is an association among several entities and represents meaningful dependencies between them. For example the association between teachers and students is teaching. It is represented by diamond.


Relationship

The Object Oriented Model

Like E-R model object oriented model is based on a collection of objects. An object contains values stored in instance variables with in an object. An object also contains bodies of code that operate in on an object. The bodies of code are called methods.

Objects that contain the same types of values and the same methods are grouped together into classes. A class may be viewed as a definition for objects. This combination of data and methods comprising a type definition is similar to programming language abstract data type.

Record Based Logical Model

Record Based Logical Model are used in describing data at logical and view levels. In contrast to object-based data models, they are used both to specify the overall logical structure of the database and to provide a higher level description of the implantation.

Record based models are so named because the database is structured in fixed format records of several types. Each record type defines a fixed number of fields or attributes, and each field is usually of a fixed length. The record based logical models are given below.

1. Relational model

In June 1970, Dr. E.F. codd published a paper entitled. A relational model of data for large shared data banks. This relational model of data for large shared data banks. This relational model, sponsored by IBM, then came to be accepted as the definitive model for relational database management system (RDBMS).

The relational model uses a collection of tables to represent both data and the relationship among those data. Each table has multiple columns, and each column has a unique name .The relational database is relatively new. The first database system was based of either a network model or hierarchical model. This model greatly improves the flexibility of the database management system. It has established itself as a primary data model for commercial data processing application.

Table 1. Teacher

Tea_id	Tea_name	Tea-address
T1	Rajesh	Balaju
T2	Gopal	Naxal
T3	Govinda	Asan
T4	Gopi	Maipi

Table 2: Student

Stu-id	Stu-name	Tea-id
S1	Gita	T1
S2	Radha	T1
S3	Nita	T2
S4	Rajan	T3
S5	Rasendra	T4

A relational database management has following properties:

- Represent data in the form of table.
- Shows the relationship between two or more its physical tables.
- Does not require the user to understand its physical implementation.
- Supports in an iterative query language.
- Provides information about its content and structure in system table.
- Support the concept of null values.

2. Network Model

Data in the network model are represented by collection of records and the relationships among data are represented by links which can be viewed as pointers. The records in the database are organized as collection of arbitrary graphs.

It is an improvement of hierarchical model. Here multiple parent-child relationship is used. The network approach allows us to build up many to many correspondences that mean each child can have more than one parent. This model is more versatile and flexible than the hierarchical model.

A record is in many respects similar to an entity in the entity relationship model. Each record is a collection of fields (attributes). Each of which contains only one data value. A link is an association between precisely two records.
A data structure diagram is a schema representing the design of network database. Such a diagram consists of two basic components, boxes, which correspond to record types, and lines which correspond to links.

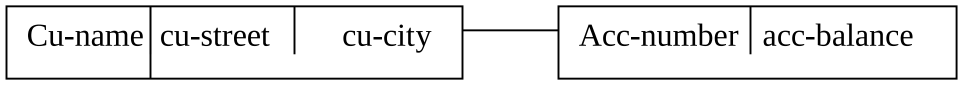
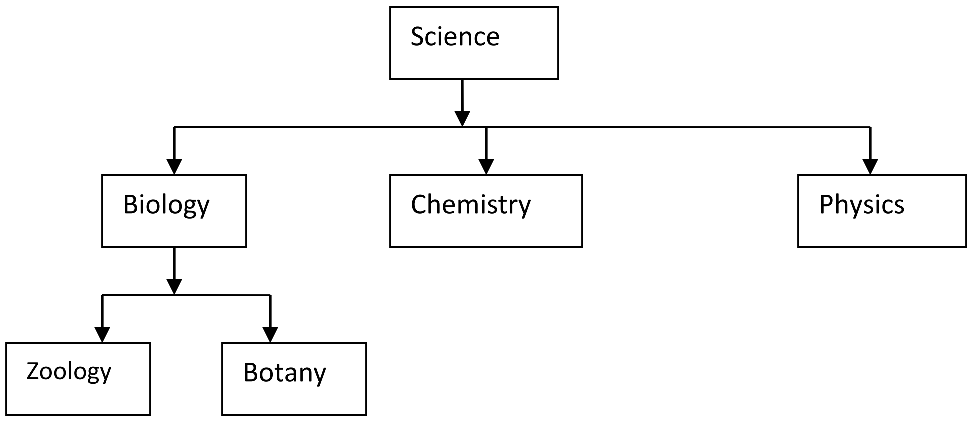


Fig: Data structure diagram

3. Hierarchical Model

The hierarchical model is a similar to the network model in the sense that data and relationship among data are represented by records and links respectively. It differs from the network models are that the records are organized as collection of trees rather than arbitrary graphs.
This model is introduced in the information management system (IMS) depend by IBM in 1968. The top level of data is parent or root and other are sub-root or branches which may have subdivision.



Tree structure diagram is the schema for a hierarchical database such a diagram consists of two basic components, boxes which correspond to record types and lies, which correspond to links.

Disadvantage of Hierarchical Model

- This structure is not flexible enough to represent all the relationships occur in a real world.
- It cannot have many to many relationships.
- The hierarchical model is used only when the concerned data has a clearly hierarchical character with a single route. E.g. DOS directory structures.

4. Physical data model

Physical data models are used to describe data at the lowest level. In contrast to logical data model, there are few physical data models such as unifying model and frame-memory model.

Transaction

A transaction is a collection of operations that performs a single logical function in a database application. Each transaction is a unit of bath atomicity and consistency. Thus, we require that transactions do not violate any data base consistency constraints .i.e. if the database was consistent when the transaction stated, the database must be consistent when the transaction successfully terminated outing the execution of transaction it may be necessary temporarily to allow inconsistency. This temporary inconsistency although necessary

Storage management

- Database requires large amount of storage space. It is measured in gigabytes or terabytes.
- The large amount of data cannot store in main memory so data are moved from main memory to secondary memory as needed and vice versa. It is important that the data base system structure the data so as to minimize the need to move data between disk and main memory.
- The goal of database system is to simplify and facilitate access to data high level views help to active this goal.
- Storage manager is a program module that provides the interface between the low level data stored in the database and the application programs and queries submitted to the system.
- The storage manager is responsible for the interaction with the file manager
- The raw data are stored on the disk using the file system, which is usually provided by a conversion operating system.
- The storage manager translates the various DML statements into low-level file system commands. Thus the storage manager is responsible for storing, retrieving and updating of data in the database.

Database Administrator (DBA)

DBA is a person who understands that data and the needs of the enterprise with respect to that data at a senior management level. Thus it is that a DBA's job to decide what data should be stored in the database in the first place and to establish policies for maintaining and dealing with the data once it has been stored. An example of such a policy might be, one that dictates who can perform what operators in what data in what circumstances, in other words security policy. Data administrator is a manager not a technical.

The DBA will typically have a staff of system programmers and other technical assistants. Thus the DBA is responsible for the overall control of the system at a technical level some tasks of DBA are as follows.

1. **Schema definition**

   DBA creates the original database schema by writing a set of definitions that translated by the DDL compiler to a set of tables that is stored permanently in the data dictionary.

2. **Storage structure and access method definition**

   By writing a set of definitions, DBA creates appropriate storage structure and access methods. Which is translated by the data storage and data definition language compiler.

3. **Schema and physical organization modifications**

DBA has the responsibility to modify schema and physical organization programmers accomplish the relatively modifications either to database schema or the description of the physical storage organization by writing a set of definitions that is used by either the DDL compiler or data storage and data definition language compiler to generate modifications to the appropriate internal system tables (e.g. data dictionary)

4. Granting of authorization for data access

The granting of different types of authorization allows the database administrator to regulate which parts of database various users can access. The authorization information is kept in a special system structure that is consulted by the database system whenever access to the data is attempted in the system.

5. Integrity constraint specification

The data values stored in the database must satisfy certain consistency constraints. For example the no of hours an employ may work in 1 week may not exceed a specified limit. Such a constraint must be specified explicitly by the database administrator. The integrity constraints are kept in a special system structure that is consulted by the database system whenever an update takes place in the system.

6. Defining dumps and reload policies

7. **Monitoring performance and responding to changing requirement using**.

DDL: data definition language.

SDL: storage definition language.

VDL: view definition language.

Database Users

The main goal of database system is to provide an environment for retrieving information from and storing new information into the database. These are four types of database users.

1. **Application Programmers**

   Application programmers are persons who write the program in HLL Such as C, COBOL, PL/1, Pascal etc. They prepare program for bank, hospital, school etc.

2. **Sophisticated users**

   They interact with system without writing programs. They form their requests in the database query language. Each such query is submitted to query processor. Whose function is to break down DML statement into instructions that the storage manager understands. The analyst who submits queries to explore data in a database is called sophisticated users.

3. **Specialized Users**

   They are sophisticated users who write specialized database applications that do not fit into traditional data processing framework. Among these applications are computer aided design system, knowledge base and export system. System that store data with complex data types eg graphic data, audio data etc.

4. **Naïve Users**

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

Overall Database System

The functional components of a database system can be broadly divided into

- Storage Manager
- Query Processor Components

Storage Manager

Storage manager is a program module that provides the interface between the low level data stored in the database and the application programs and queries submitted to the system. The storage manager is responsible for the following tasks.

- Interaction with the file manager.
- Efficient storing, retrieving and updating data.

The storage manager components includes:

1. **Authorization and integrity manager**

Which tests for the satisfaction of integrity constraints and checks the authority of users to access data

2. **Transaction Manager**

   Which ensures that the database remains in a consistent (correct) state despite system failures, and that current transition execution proceed without conflicting

3. **File manager**

   Which manages the allocation of space on disk storage and that concurrent transaction execution proceed without conflicting.

4. **Buffer Manager**

   Which is responsible for fetching data from disk storage into main memory, and deciding what data to cache in main memory. The manager is critical part of the database system, since it enables the database to handle data sizes that are much larger than the size of main memory.

The storage manager implements several data structures as part of the physical system implementation.

- **Data files**

  Which store the database itself.
- **Data Dictionary**

  Which stores metadata about the structure of the database in particular the schema of the database.
- **Indices**

  Which provides fast access to data items that hold particular values.
- **Statistical data**

  Which stores statistical information about data in the database. This information is used by query processor to select efficient ways to execute a query.

The Query processor

The query processor components include.

- **DDL Interpreter**

  Which interprets DDL statements and records the definitions n the data dictionary.
- **DML Compile**

  Which translates DML statements in a query language into an evaluation plan consisting of low level instructions that the query evaluation engine understand.
- **Embedded DML pre-compiler**

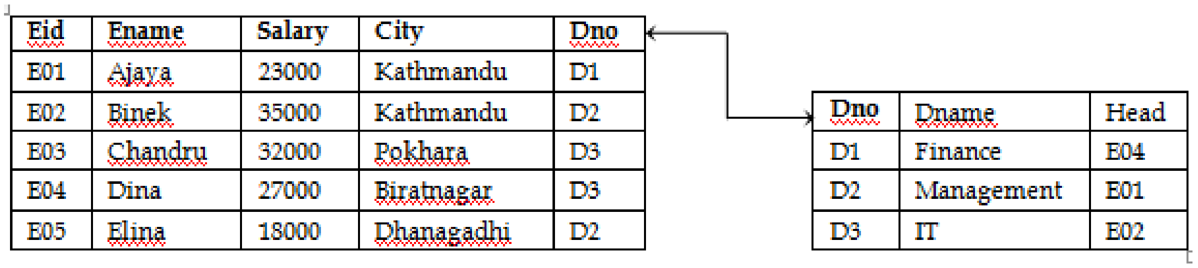
  Which convents DML statements embedded in an application program to normal procedure call in the host language. The pre-compiler must interact with the DML complier to generate the appropriate code.

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

Relational Model

The relational model is today the primary model for commercial data processing applications. It has attained its primary position because of its simplicity as compared to earlier data models such as the network model or the hierarchical model. It is a lower level model that uses a collection of tables (also called relations) to represent both data and the relationship among those data. A table of values is called relation. A relation may be thought of as a **set of rows**. A relation may alternately be thought of as a **set of columns**. Each row represents a fact that corresponds to a real-world **entity** or **relationship**. Each row has a value of an item or set of items that uniquely identifies that row in the table. Sometimes row-ids or sequential numbers are assigned to identify the rows in the table. Each column typically is called by its column name or column header or attribute name. Each table has multiple columns and each column has a unique name.

Example: RDBMS



Eid	Ename	Salary	City	Dno
E01	Ajaya	23000	Kathmandu	D1
E02	Binek	35000	Kathmandu	D2
E03	Chandru	32000	Pokhara	D3
E04	Dina	27000	Biratnagar	D3
E05	Elina	18000	Dhanagadhi	D2

Dno	Dname	Head
D1	Finance	E04
D2	Management	E01
D3	IT	E02

Formal Definitions

A **Relation** may be defined in multiple ways.

The **Schema** of the form: *R* (A1, A2 ...An) is called relation, Relation schema *R* is defined over **attributes** A1, A2 ...An.

For Example:

CUSTOMER (Cust-id, Cust-name, Address, Phone#)

Here, CUSTOMER is a relation defined over the four attributes Cust-id, Cust-name, Address, Phone#, each of which has a **domain** or a set of valid values. For example, the domain of Cust-id is 6 digit numbers.

Tuple

A tuple is an ordered set of values. Each value is derived from an appropriate domain. Each row in the CUSTOMER table may be referred to as a tuple in the table and would consist of four values. <632895, "John Smith", "101 Main St. Atlanta, GA 30332", "(404) 894-2000"> is a tuple belonging to the CUSTOMER relation. A relation may be regarded as a set of tuples (rows or records).

Domain

A domain has a logical definition: e.g., “USA_phone_numbers” are the set of 10 digit phone numbers valid in the U.S. A domain may have a data-type or a format defined for it. The USA_phone_numbers may have a format: (ddd)-ddd-dddd where each d is a decimal digit. E.g., Dates have various formats such as month name, date, year or yyyy-mm-dd, or dd mm, yyyy etc. An attribute designates the role played by the domain. E.g., the domain Date may be used to define attributes “Invoice-date” and “Payment-date”. The relation is formed over the Cartesian product of the sets; each set has values from a domain; that domain is used in a specific role which is conveyed by the attribute name. For example, attribute Cust-name is defined over the domain of strings of 25 characters. The role these strings play in the CUSTOMER relation is that of the name of customers.

Formally,

Given R(A1, A2, ..., An)

$r(R) \subset dom(A1) \times dom(A2) \times \ldots \times dom(An)$

R: schema of the relation

r of R: a specific "value" or population of R.

R is also called the intension of a relation

r is also called the extension of a relation

Example

Let S1 = {0, 1}

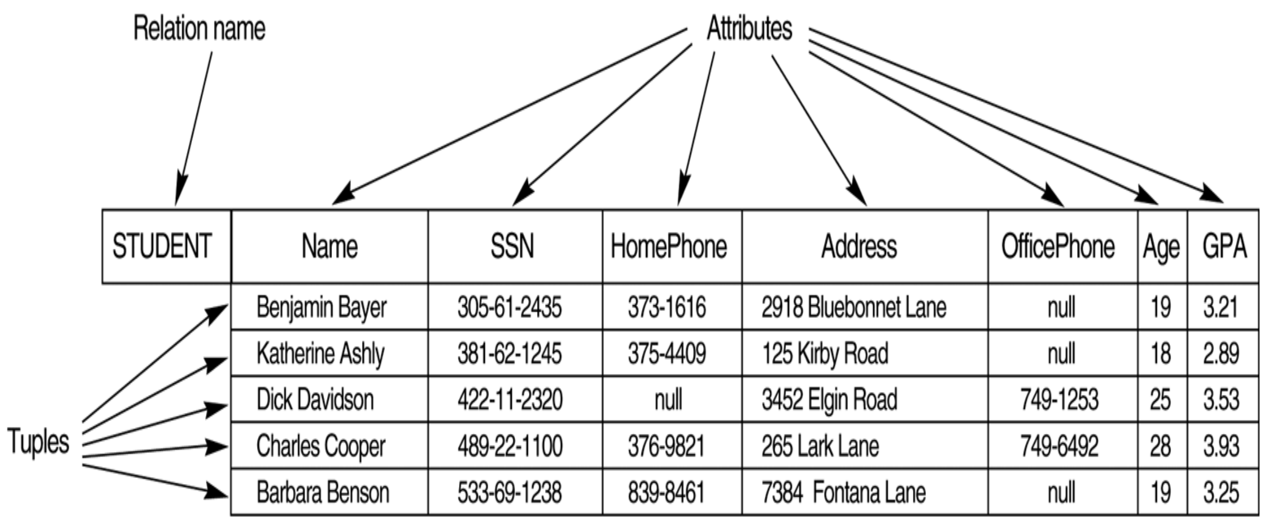
Let S2 = {a, b, c}

Let $R \subset S1 \times S2$

Then for example: r(R) = {<0, a>, <0, b> , <1, c> } is one possible “state” or “population” or “extension” r of the relation R, defined over domains S1 and S2. It has three tuples.

Definition Summary

Informal Terms	Formal Terms
Table	Relation
Column	Attribute/Domain
Row	Tuple
Values in a column	Domain
Table Definition	Schema of a Relation (Intension)
Populated Table	Extension



STUDENT	Name	SSN	HomePhone	Address	OfficePhone	Age	GPA
	Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	null	19	3.21
	Katherine Ashly	381-62-1245	375-4409	125 Kirby Road	null	18	2.89
	Dick Davidson	422-11-2320	null	3452 Elgin Road	749-1253	25	3.53
	Charles Cooper	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
	Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	null	19	3.25

Data Integrity

The fundamental function of the DBMS is to maintain the integrity of the data. Data integrity ensures that the data in the database is consistent, accurate, correct, and valid. It ascertains that the data adhere to the set of rules defined by the database administrator and hence, prevents the entry of the invalid information into database. Data integrity is of four types, namely, *domain integrity*, *entity integrity*, *referential integrity*, and *semantic integrity*. Generally, domain integrity is applied on the attribute; entity integrity is applied on the tuple; referential integrity is applied on the relation; and semantic integrity ensures logical data in the database.

1. **Entity Integrity**

   The entity integrity constraint states that no primary key (nor any part of the primary key) value can be null. This is because the primary key value is used to identify individual tuples in a relation. Having null value for the primary key implies that we cannot identify some tuples. This also specifies that there may not be any duplicate entries in primary key column.

2. **Referential Integrity**

   The referential integrity constraint is specified between two relations and is used to maintain the consistency among tuples in the two relations (Parent and Child). Informally, the referential integrity constraint states that a tuple in one relation that refers to another relation must refer to an existing tuple in that relation. It is a rule that maintains consistency among the rows of the two relations. Examples of Referential integrity constraint in the Customer/Order database:

   Customer (<u>custid</u>, custname)

   Order (<u>orderID</u>, custid, OrderDate)

   To ensure that there are no orphan records, we need to enforce referential integrity.

   An orphan record is one whose foreign key value is not found in the corresponding entity – the entity where the PK is located.

3. **Domain Integrity**

   A **domain** represents a set of values that can be assigned to an attribute. Domain integrity constraint is specified on the column (attribute) of a relation, so that correct values can be entered in the column for each record. The domain integrity states that every element from a relation should respect the type and restrictions of its corresponding attribute. For example, we can specify the value of 'Marks' attribute of "Student" table which must be greater than 0 and less than or equal to 100.

4. **Semantic Integrity**

   To represent real world accurately and consistently, business rules and logical rules must be enforced in database. Such rules are derived from our knowledge of the application semantics and are called semantic integrity constraints. Semantic integrity ensures that data in the database is logically consistent and complete with respect to the real world. This type of integrity cannot be expressed by the model and contains integrity constraints like:

   - Number of pages of a book cannot be zero
   - A book is published by only one publisher
   - An author cannot review his own book etc.

   A constraint specification language may have to be used to express these rules. SQL-99 allows triggers and assertions to allow for some of these.

Advanced SQL

SQL stands for structured query language developed at IBM research for system R. It includes features of relational algebra and tuple relational calculus. It is standard for relational data access. It is DBMS independent. It is one commercially available query language. SQL can define the structure of data create table, index, view alter table etc. modify data in the data base such as select, update, delete, insert etc.

SQL Data type

- Numeric
- Character strings
- Bit strings
- Temporal data
- Null value

Numeric data

- **Exact number**

  Two integer types with different ranges.

INTEGER (or INT) and SMALLINT

- **Approximation numbers**
  Three floating point type:
  FLOAT, REAL and DOUBLE PRECISION.
- User can define the precision for FLOAT
- The precision of REAL and DOUBLE PRECISION is fixed.
- Floating point numbers can be in decimal or scientific notation.

SQL Character strings

A Character string is a sequence of printable chars. In SQL, character string is denoted by enclosing it in single quote; e.g. 'Hello SQL'

Two types of character strings :

Fixed length n: CHAR (n) or CHARACTER (n)

Varying length of maximum n: VARCHAR (n) or CHARACTER (n).

SQL Bit Strings

Bit strings are sequences of binary digits or bits. Bit string types are:

Fixed length n: BIT (n)

Varying length of maximum n: VARBIT (n) or BITVARYING (n). The default value for n is 1.

SQL temporal data

Date data type

Date format is: YYYY-MM-DD

Time data type

Time format is HH: MM: SS

Interval data type

E.g. Admit date – discharge date.

Format is INTERVAL start field (p) [to end – field (fs)]

SQL components

1. **Data definition language (DDL):**
   The SQL DDL provides commands for defining relation schema, deletion relations, creating indices, and modifying relation schemas.
2. **Interactive data definition language (DML):**

The SQL DML includes languages a query language based on both the relational algebra and the tuple relational calculus. It includes also commands to insert tuples into delete tuples from and modify tuples in the database.

3. **Embedded DML:**
   The embedded form of SQL is designed for use with in general purpose programming languages, such as PL/1, COBOL, PASCOL, FORTRAN and C.
4. **View definition:**
   The SQL DDL includes commands for defining views.
5. **Authorization:**
   The SQL DDL includes commands for specifying access rights to relations and views.
6. **Integrity:**
   The SQL DDL includes commands for specifying integrity constraints that the data stored in the database must satisfy. Updates that violate integrity constraints are disallowed.
7. **Transaction control:**
   SQL includes commands for specifying the beginning and ending of transactions. Several implementations also allow explicit locking of data for concurrency control.

Table creation

A new relation can be created using the CREATE TABLE command. The general syntax is as follows.

```
        CREATE TABLE table_name [{column descriptors}];
e.g.    CREATE TABLE DEPARTMENT;
```

Table Creation Example

```
CREATE TABLE employee
(
        Emp-no          INT(6) NOT NULL,
        Name            VARCHAR(20),
        Gender          CHAR,
        DOB             DATE,
        Salary          DECIMAL (8, 2),
        PRIMARY KEY     (EMP-no)
);
```

The relations for banking enterprise are given below:

Branch = (branch_name, branch_city, asserts)

Customer = (customer_name, customer-street, customer- city)
Loan = (branch- name, loan number amount)
Borrower = (customer- name, lone- number)
Account = (branch_name, account_number, balance)
Depositor = (customer- name, account_number)

Select Overall Form

```
SELECT      <attributes and function list >
FROM        < table list >
[WHERE      < condition >]
[GROUP BY   <grouping attributes >]
[HAVING     <group condition >]
[ORDER BY   < {attribute ASC/DESC} list >];
```

Basic Structure

The basic structure of an SQL expression consists of three clauses. SELECT, FROM and WHERE.

The SELECT clause corresponds to the project operation of the relational algebra. It is used to list the attributes desired in the result of a query. The FROM clause corresponds to the Cartesian product operation of relational algebra. It lists the relations to be scanned in the evaluation of the expression. The WHERE clause corresponds to the selection pedicure of the relational algebra. It consists of a predicate involving attributes of the relations that appear in the form clause.

The SQL query is the form

```
SELECT A1   A2,……………,An
 FROM   R1,  R2, ……………Rn
 WHERE P;
```

Where A1 A2,……………,An are attributes, R1, R2, ……………Rn are relations and p is predicate i.e. condition.

The Rename operation

SQL allows renaming attributes.

Syntax

Old_name AS new_name

We can use as clauses in both select and from clause.

Example: loan_no is replaced by loan_id

```
SELECT  customer_name, borrower.loan_number AS loan_id
FROM borrower, loan
WHERE  borrower.loan_number = loan.loan_number AND branch_name = ‘bagbazar’;
```

Tuple Variables

The as clause is used in defining the notion of tuple variable, as is done in the tuple relational calculus. The tuple variable in SQL must be associated with a particular relation. Tuple variables are defined in the FROM clause.

#For all customers who have a loan from the bank, find their names and loan numbers.

```
SELECT DISTINCT customer_name, T.loan_number
FROM borrower AS T, loan AS S
WHERE T.loan_number=S.loan_number;
```

String operations

The commonly used operation on string is pattern matching using the operator “like”. We use two special characters:

Percent (%): it matches any substring.

Underscore (-): it matches any character.

Example: Find the names of all customers whose street address includes the substring "main" is:

```
SELECT customer_name
 FROM Customer
WHERE street like "%main%"
```

Some examples are:

- “Perry%” matches any string beginning with “Perry”.
- “%edge%” matches any string containing “edge” as a sub string.
- “- - -“matches any string of exactly three characters.
- “- - -%” matches any string of the least three characters.

Ordering the display of tuples

The ORDER BY clause is used to list the tuples in sorted order.

Example: To list in alphabetical order all customers who have a loan at Bagbazar branch is:

```
SELECT DISTINCT customer_name
FROM borrower, loan
WHERE Borrower.loan_number = loan.loan_number AND Branch_name = ‘bagbazar’
 ORDER BY customer_name ;
```

We can specify DESC for descending order or ASC for ascending order.

E.g. List all attributes from loan relation amount by descending order and loan_number by ascending order.

```
SELECT *
FROM loan
ORDER BY amount DESC, loan_number ASC.
```

Set operations

Set operations are union, intersect and except.

The union operation

Find all customers having a loan, an account or both is:

```
(SELECT customer_name
FROM depositor)
UNION
(SELECT customer_name
FROM borrower)
```

The union operation automatically eliminates duplicates. For duplicate value we use union all in place of union.

```
(SELECT customer_name
FROM depositer)
UNION ALL
(SELECT customer_name
FROM borrower)
```

The intersection operation

To find all customers who have a both loan and an account at the bank is:

```
(SELECT distinct customer_name
FROM depositor)
INTERSECT
(SELECT distinct customer_name
FROM borrower)
```

The select operation automatically removes duplicates. If we want all duplicates, than intersect all in place of intersect.

E.g.

```
(SELECT customer_name
FROM Depositor)
```

INTERSECT ALL
(SELECT customer_name
FROM borrower)

The except operation

To find all customers who have an account but no loan at the bank is:

(SELECT DISTINCT customer_name FROM depositor)
EXCEPT
(SELECT customer = name FROM borrower)

Except operation automatically removes duplicated. If we want to reliant all duplicates then except is replaced by EXCEPT ALL.

(SELECT eustomer-name FROM depositor)
EXCEPT ALL
(SELECT customer_name FROM borrower)

Aggregate functions

Aggregate functions that take a collection of value as input and returns a single value. Aggregate functions are given below:

- Average : AVG
- Minimum : MIN
- Maximum : MAX
- Total : SUM
- Count : COUNT

To find the average account balance of Bagbazar branch.

SELECT AVG (balance)
FROM account
WHERE branch- name = 'Bagbazar'

For the group of tuples, we can use group by to find the average account balance at each branch is:

SELECT branch_name, AVG (balance)
FROM account
GROUP BY branch- name

If we want duplication is removed then **distinct** is used. To find the number of depositors for each branch is:

Select branch_name, **count** (**distinct** customer- name)

From depositor, account

Where depositor account- number = account_account_number

Group by branch_name.

For the condition in **group by** clause we use having clause

e.g. To list branch_name and average balance group by branch_name having average balance greater than 1200 is:

Select branch_name, **avg** (balance)

From account

Group by branch_name

Having avg (balance) > 1200

To find the average balance for all customers:

Select avg (balance)

From account

To find number of tuples in the customer relation:

Select count (*)

From customer

Find the averages balance for each customer who lives in lalitpur and has at least three accounts.

Select depositor.customer- name, **avg** (balance)
From depositor, account, customer
Where depositor.account_number = account.account_number **and**
depositor.customer_name = customer.customer_name **and** customer_city = "lalitpur"
Group by depositor.customer_name
Having count (**distinct** depositor.Account_number) > = 3

Null values

Null values are values that indicate absence of information about the value of an attribute.
To find the loan number with null values.

Select loan-number
From loan
Where amount **is null;**

Nested Sub queries

A sub query is nested inside another query called nested sub queries. A common use of sub queries is to perform tests for set membership, set companions and set cardinality.

Set membership

The in connective tests for set membership, where the set is a collection of values produced by a **SELECT** clause.

The NOT IN connective tests for the absence of set membership consider the query:

Find all customers who have both a loan and account at the bank. This is solved by union operation and another approach is finding all account holders at the bank who are members of the set of borrowers from the bank. This formation generates the same results as did the previous one but it leads us to write our query using in connective of SQL. We first by finding all account holders.

(**Select** customer_name
  From depositor)

We then need to find those customers who are borrower from banks and who appear in the list of account holders obtained on the sub query. The result is

select distinct customer_name
form borrower.
Where customer_name **in** (**select** customer_name **from** depositor);

Find all customers who have both on account and loan at the Bagbazar branch.

(**Select distinct** customer_name
From borrower, loan
Where borrower . loan number = loan . loan-number.
and branch_name = "bagbazzar" **and** (branch_name, customer_name) **in** (**select** branch_name, customer_name **from** depositor, account **Where** depositor.account_number = account.account_number)

We also use **not in** clause

To find all customer who have a loan at a bank but don't have an account at the bank.

Select distinct customer_name
From borrower
Where customer_name **not in** (select customer_name
From depositor);

Set Comparison

We use comparison operators for set comparison operations. "Greater than at least one" is represented by > **some**.

Find the names of all branches that have assets greater than those of at least one branch located in Bagbazar branch.

```
Select branch_name
From branch
Where assents > some (select asserts From branch Where branch_city = 'Bagbazar');
```

SQL allows < some, < = some, =some, and < > some is not the same as not in.
Find the names of all branches that have asserts greater than that of each branch in Bagbazar.

```
SELECT branch_name
FROM branch
WHERE assets > ALL (SELECT assets FROM branch WHERE branch_city = 'Bagbazar')
```

SQL also allows < all, <= all, > = all, and < > all comparisons
Find the branch that has the highest average balance

```
SELECT branch_name
FROM account
GROUP BY branch_name
HAVING AVG (balance) > = ALL (SELECT AVG (balance)
                              FROM account
                              GROUP BY branch_name);
```

Test for empty relations

SQL allows for testing whether a sub query has any tuples in its result. The EXISTS construct returns the value true if the argument sub query is non empty.
Find a customers who have both an account and a loan at the bank is:

```
SELECT customer_name
FROM borrower
WHERE EXISTS (SELECT * FROM depositor
          WHERE depositer.Customer_name = borrower.Customer_name);
```

There is nonexistence tuples to test in a sub query by using the non-exists construct. We can use the not exists construct to simulate the set containment (i.e. subset) operation. We can write "relation A contains relation B" as "not exists (B except A)"

Find all customers who have an account at all the branches located at Bagbazar.

```
SELECT DISTINCT s.customer_name
FROM depositor AS S
```

```
WHERE NOT EXISTS (SELECT branch_name FROM branch
WHERE branch_city = 'Bagbazar')
EXCEPT
(SELECT R.branch_name FROM depositor AS T, account AS R
WHERE T. account_number = R. account_number AND S. customer name = T. customer
name)
```

Derived Relations

The result of the relation can be renamed and the attributes also renamed by AS clause.

```
(SELECT branch_name, AVG (balance) FROM depositer
 GROUP BY branch_name AS result (branch_name, avg – balance)
```

View

We can create view in SQL by CREATE VIEW command. The syntax is given below:

```
CREATE VIEW V AS < query expression >
```

We can define view for the names of customers who have either an account or a loan is:

```
CREATE VIEW all_customer AS
(SELECT branch_name, customer_name FROM depositor, account
  WHERE depositor.account_number = account.Account_number )
 UNION
  (SELECT branch_name, customer_name
  FROM borrower, loan
  WHERE borrower.loan_number = loan.loan_number);
```

Modification of the Database

SQL allows deleting, inserting and modifying data in a database.

Deletion

SQL allows to delete only whole tuples. We cannot delete values on only particular attributes.

Syntax

```
DELETE FROM r
WHERE p;
```

Where r is the relation in which we have to delete values and p is the predicate condition

E.g. To delete Puspa's account records

```
DELETE FROM depositor
WHERE customer_name = 'puspa';
```

To delete all loan amounts between 4000 and 5000

```
DELETE FROM loan
WHERE amount BETWEEN 4000 AND 5000;
```

To delete all account at every branch located in Bagbazar

```
Delete from account
Where branch_name in (select branch_name from branch
Where branch_city = ‘bagbazar’);
```

To delete the records of all accounts with balance below the average at the bank.

```
Delete from account
Where balance < (select avg (balance) FROM account)
```

Insertion

We can insert tuples in the relation. The attributes values for inserted tuples must be members of the attributes domain. Tuples inserted must be of the correct aritry.

To insert account_no 501 at the Bagbazar branch and the balance is 5000

```
INSERT INTO account VALUES (‘Bagbazar’, 501, 5000)
```

It is equivalent to:

```
INSERT INTO account (branch_name, account_number , balance)
VALUES (‘Bagbazar’, 501, 5000);
```

It is also equivalent to:

```
Insert into account (account_number, branch_name, balance)
Values (501, ‘Bagbazar’, 5000)
```

The insert statement considered only examples in which a value is given for every attribute in inserted tuples. It is possible for inserted tuples to the given values on only some attributes of the schema. The remaining attributes are assigned a null value denoted by NULL e.g.

```
Insert into account
Values (NULL, ‘B-101’, 1500)
```

We know that account B-101 has Rs 1500 but branch name is not known.

Updates

We can change a value in a tuple without changing all values in the tuple. For this, update statement can be used.

To increase the balance by 5 percent

```
UPDATE account
SET balance = balance * 1.05;
```

Account with balance over 10,000 receives 6 percent interest and other receives 5 percent.

```
UPDATE account
SET balance = balance * 1.06
WHERE balance > 10000
UPDATE account
SET balance = balance * 1.05
WHERE balance < = 10000;
```

Update of a view

We can create view by:

```
Create view branch_loan as
Select branch_name, loan_number
From loan;
```

We can update by:

```
Insert into branch_loan
Values ('pokhara', 305)
```

Data Definition Language

The SQL DDL allows the specification of set of relations and information about each relation. The information includes:

- The schema for each relation.
- The domain of values associated with each attribute.
- The integrity constraints.
- The set of indices to be maintained for each relation.
- The security and authorization information for each relation.
- The physical storage structure of each relation on disk.

Schema Definition in SQL

We can define SQL relation using the create table command.

Create table r (A_1 D_1, A_2 D_2,……..,A_n D_n, < integrity_constraint 1>

 …………………….

< integrity_constraint k >);

Where r is the name of the relation each A is the name of an attribute type in the values in the domain of attribute A_i.

Examples:

```
CREATE TABLE customer
(customer_name CHAR (20) NOT NULL,
customer_strees CHAR (30),
customer_city CHAR (30),
PRIMARY KEY (customer_name));

CREATE TABLE Branch
(branch_name CHAR(15) NOT NULL,
Branch- city CHAR (30),
Assets    INTEGER,
PRIMARY KEY (branch_name),
CHECK (assets > = 0);
```

```
CREATE TABLE account
(account_number CHAR(10) NOT NULL,
branch_name CHAR(15),
Balance INTEGER,
PRIMARY KEY (account_number),
CHECK (balance > = 0));

CREATE TABLE depositor
(customer_name CHAR(20) NOT NULL,
account_number CHAR(10) NOT NULL,
PRIMARY KEY (customer_name,
account_number)),
```

Delete Table

To remove a relation from SQL database. We use DROP TABLE command. It removes all information and table also. The command is

```
DROP TABLE r
```

Where r is the relation

Alter Table

We use alter table command in SQL to add attributes to the existing relation. All tuples in the relation are assigned **null** as the value for the new attribute. The form of alter table command is:

```
ALTER TABLE r ADD A D
```

Where r is the name of existing relation, A is the name of attribute to be added and D is the domain of the added attribute. We can drop attributes from a relation using a command:

```
ALTER TABLE r DROP A
```

Where r is the name of an existing relation and A is the name of attribute in a relation.

Integrity constraints

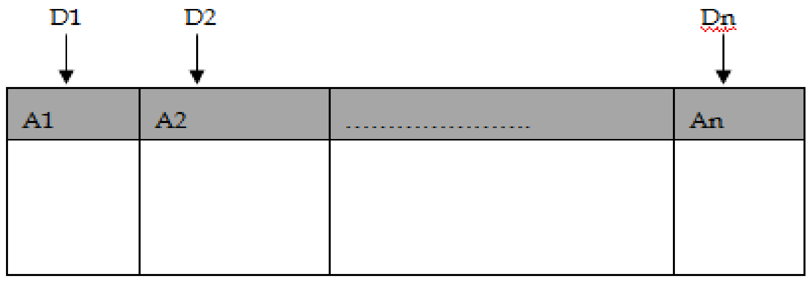
The term integrity refers to the accuracy or correctness of data in the database. Integrity constraint is a condition specified on a database schema which must hold on all of valid relation instances.
Integrity constraints ensure that changes made to the database by authorized users do not result in a loss of data consistency. Thus, integrity constraints guard against accidental damage to the database. Constraints are basically used to impose rules on the table, whenever a row is inserted, updated, or deleted from the table. Constraints prevent the deletion of a table if there are

dependencies. The different types of constraints that can be imposed on the table are domain constraints, referential constraints, trigger, assertions etc. The constraints related to domain constraints are NOT NULL, UNIQUE, PRIMARY KEY and CHECK constraints.

1. Domain constraints
   - NOT NULL constraints
   - UNIQUE constraints
   - PRIMARY KEY constraints
   - CHECK constraints etc.
2. Referential constraints
3. Triggers
4. Assertion

Domain constraint

Domains are used in the relational model to define the characteristics of the columns of a table. Domain refers to the set of all possible values that attribute can take. The domain specifies its own name, data type, and logical size. The logical size represents the size as perceived by the user, not how it is implemented internally. For example, for an integer, the logical size represents the number of digits used to display the integer, not the number of bytes used to store it. The domain integrity constraints are used to specify the valid values that a column defined over the domain can take. We can define the valid values by listing them as a set of values (such as an enumerated data type in a strongly typed programming language), a range of values, or an expression that accepts the valid values. Strictly speaking, only values from the same domain should ever be compared or be integrated through a union operator. The domain integrity constraint specifies that each attribute must have values derived from a valid range



The **create domain** clause can be used to define new domains. For example, to ensure that age must be an integer in the range 1 to 100, we could use:

```
CREATE DOMAIN Ageval INTEGER
CHECK (VALUE >= 1 AND VALUE <= 100)
```

The domain can be restricted to contain only a specified set of values by using IN clause:

```
CREATE DOMAIN AccountType CHAR (10)
```

```
                    CONSTRAINT account-type-test
                     CHECK (VALUE IN (‘Checking’, ‘Saving’)
```

SQL also provides drop domain and alter domain clauses to drop or modify domains that have been created earlier.

CHECK Constraints

CHECK constraint is added to the declaration of the attribute. The CHECK constraint may use the name of the attribute or any other relation or attribute name may in a sub-query. Attribute value check is checked only when the value of the attribute is inserted or updated. CHECK constraints allow users to prohibit an operation on a table that would violate the constraint. It is a local constraint.

Example: let’s create a student table with attributes student id, student name, age and address. If we need to allow only those students in the table whose age must be an integer range 20 to 45, we could use the CHECK constraint during the creation of table as below:

```
CREATE TABLE Student
(
     sid      INTEGER,
     sname VARCHAR(20),
     age      INTEGER,
     PRIMARY KEY (sid),
     CHECK (age>=20 AND age<=45)
)
```

In the above student table if we are trying to insert a new record as

```
       INSERT INTO Student
       VALUES (5, “Rajesh”, 15);
```

We get insertion is rejected message since value of age attribute violated the check condition.

Referential Integrity

In the relational data model, associations between tables are defined through the use of foreign keys. The referential integrity rule states that a database must not contain any unmatched foreign key values. It is to be noted that referential integrity rule does not imply a foreign key cannot be null. There can be situations where a relationship does not exist for a particular instance, in which case the foreign key is null. A referential integrity is a rule that states that either each foreign key value must match a primary key value in another relation or the foreign key value must be null. Referential integrity ensures that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation to establish the relationship between tables.

In relational model we use primary key and foreign key to establish relationships between two relations. Table containing primary key attribute is called master table and the table containing foreign key attribute is called child table. Referential integrity ensures that value appeared in foreign key attribute of child table must also appear in primary key attribute of corresponding master table. Defining referential integrity restricts database modification operations such as insert, delete, and update. To illustrate this consider following two relations:

Employee

Eid	Ename	Salary	Dno
E01	Ramesh	26000	D1
E02	Sohan	19000	D1
E03	Renuka	25000	D2
E04	Dina	22000	D3

Department

Dno	**Dname**	**Location**
D1	IT	Gwarko
D2	Finance	Satdobato
D3	HR	Balkumari

- **Insert:** We cannot insert new tuples containing value of foreign key attribute that do not appear in primary key attribute of master table. For example, we cannot insert new employee that works in D4 department because the department D4 does not exists in department table.

- **Delete:** We cannot delete tuples containing values of primary key attribute that also appear foreign key attribute of related table. For example, we cannot delete tuple containing value D2 of DNO from department table because there are employees working in department D2.

- **Update:** There are two cases of update operation
    - **Case1:** We cannot change the value of primary key attribute if the child table contains related values. For example, we cannot change value of DNO from D1 to D5 in department table because employee table contains employees working in department D1.
    - **Case 2:** We cannot change value of foreign key attribute if the primary key attribute does not contain modified value of foreign key attribute. For example, we cannot change value of DNO from D3 to D5 in employee table because department table does not contain department D5.

Referential Integrity in SQL

Primary keys and foreign keys can be specified as parts of the SQL create table statement as below:

```
CREATE TABLE Books
 (
        ISBN VARCHAR (20),
        Title    VARCHAR (20),
        Category VARCHAR (15),
        Price    INTEGER,
        Pages    INTEGER,
        Year     DATE (10),
        Pid      VARCHAR (5),
        PRIMARY KEY (ISBN),
        FOREIGN KEY (Pid) REFERENCES Publisher (Pid) ON DELETE CASCADE ON
        UPDATE CASCADE
)
```

```
CREATE TABLE Publisher
(
        Pid VARCHAR (10),
        PnameVARCHAR (15),
        City    VARCHAR (20),
        Email  VARCHAR (20),
        PRIMARY KEY (Pid)
)
```

- Alternatively, we can use on delete set null and on update set null.
- Also, we can use on delete set default and on update set default.

Assertions

Assertions are general purpose checks that allow the enforcement of any condition over the entire database. Similar to CHECK but they are global Constraints. When an assertion is made, the system tests it for validity, and tests it again on every update that may violate the assertion. This testing may introduce a significant amount of overhead; hence assertions should be used with great care. An assertion in SQL takes the form:

```
CREATE ASSERTION <assertion-name> CHECK <predicate>
```

Example: The department id of manager relation is always not null since each manager works at least one department.

```
CREATE ASSERTION Noallow CHECK
        (NOT EXISTS (SELECT * FROM MANAGER WHERE DeptId IS NULL));
```

Above assertion ensures that there is no manager who is not assigned any department at any time. Let's take a manager relation in which some records are inserted as

Manager

Mid	Mname	Address	DeptId
M01	Aayan	Pokhara	D11
M02	Bhupi	Lalitpur	D22
M03	Arjun	Kathmandu	D11
M05	Ramesh	Palpa	Null

In the above table the department id of manager 'Ramesh' is NULL due to which assertion is violated and we cannot further modify the database.
Assertions can be dropped using the DROP ASSERTION command.

```
        DROP ASSERTION Noallow;
```

Example 2: The sum of all loan amounts for each branch must be less than the sum of all account balances at the branch.

```
   CREATE ASSERTION sum-constraint
   CHECK (NOT EXISTS (SELECT * FROM branch
                     WHERE (SELECT SUM (amount) FROM loan
                       WHERE loan.branch-name =branch.branch-name) >=   (SELECT
                               SUM (amount) FROM account
                               WHERE loan.branch-name = branch.branch-name)));
```

Example 3: To specify the constraint that the salary of an employee must not be greater than the salary of the manager of the department that the employee works for in SQL, we can write the following assertion:

```
  CREATE ASSERTION Salary_Constraint
  CHECK (NOT EXISTS (SELECT * FROM Employee E, Employee M, DEPARTMENT D
        WHERE E.Salary >M.Salary AND E.Dno=D.Dnumber AND D.Mgr_ssn =M.Ssn));
```

Difference between CHECK constraint and Assertion

A major difference between CREATE ASSERTION and the individual domain constraints and tuple constraints is that the CHECK clauses on individual attributes, domains, and tuples are checked in SQL only when tuples are inserted or updated. Hence, constraint checking can be implemented more efficiently by the DBMS in these cases. The schema designer should use CHECK on attributes, domains, and tuples only when he or she is sure that the constraint can only be violated by insertion or updating of tuples. On the other hand, the schema designer should use CREATE ASSERTION only in cases where it is not possible to use CHECK on attributes, domains, or tuples, so that simple checks are implemented more efficiently by the DBMS.

Triggers

A trigger is a procedure (statement) that is automatically invoked by the DBMS in response to specified changes to the database. A database that has a set of associated triggers is called an active database. Triggers are useful mechanisms for alerting humans or for starting certain tasks automatically when certain conditions are met. It is the most practical way to implement routines and granting integrity of data. Unlike the stored procedures or functions, which have to be explicitly invoked, these triggers implicitly get fired whenever the table is affected by the SQL operation. For any event that causes a change in the contents of a table, a user can specify an associated action that the DBMS should carry out. Trigger follows the Event-Condition-Action scheme (ECA scheme). To design a trigger mechanism, we must meet following three requirements:

- Event: A change to the database that activates the trigger.
- Condition: Trigger performs some action only if a specified condition matches at the occurrence of the event
- Action: A procedure that is executed when the trigger is activated and its condition is true.

General form of trigger:

```
CREATE TRIGGER <trigger-name>
        <Time events>
             ON <list-of-tables>
                  WHEN <Predicate>
                       <Action-name>
```

Need of Triggers

Triggers are useful mechanisms for alerting humans or for starting certain tasks automatically when certain conditions are met.

For example, for every pre-paid account whose balance is less than or equal to 0, the account is automatically marked as "blocked".

```
CREATE TRIGGER overdraft AFTER UPDATE ON pre-paid
       REFERENCING NEW ROW AS nrow
           FOR EACH ROW
                WHEN nrow.balance <= 0
                     UPDATE pre-paid
                          SET blocked = 'T";
```

Let's take a pre-paid relation as

Pre-paid

PNO	Balance	Blocked
1	4000	False
2	5000	False
3	7000	False
4	2000	False

```
UPDATE TABLE pre-paid
SET Balance = Balance-4000;
```

If we execute this query then we get following modified Pre-paid table

PNO	Balance	Blocked
1	4000	T
2	5000	False
3	7000	False
4	2000	T

Query Processing and Optimization

Query Processing is a procedure of transforming a high-level query (such as SQL) into a correct and efficient execution plan expressed in low-level language. A query processing select a most appropriate plan that is used in responding to a database request. When a database system receives a query for update or retrieval of information, it goes through a series of compilation steps, called **execution plan.** In the first phase called **syntax checking** phase, the system parses the query and checks that it follows the syntax rules or not. It then matches the objects in the query syntax with the view tables and columns listed in the system table. Finally it performs the appropriate query modification. During this phase the system validates the user privileges and that the query does not disobey any integrity rules. The execution plan is finally execute to generate a response. So query processing is a stepwise process.

Q. What do you mean by query processing? What are the various steps involved in query processing? Explain with the help of a block diagram.

Ans: Query processing includes translation of high-level queries into low-level expressions that can be used at the physical level of the file system, query optimization and actual execution of the query to get the result. It is a three-step process that consists of parsing and translation, optimization and execution of the query submitted by the user .These steps are discussed below:

- Parsing and translation
- Optimization
- Evaluation

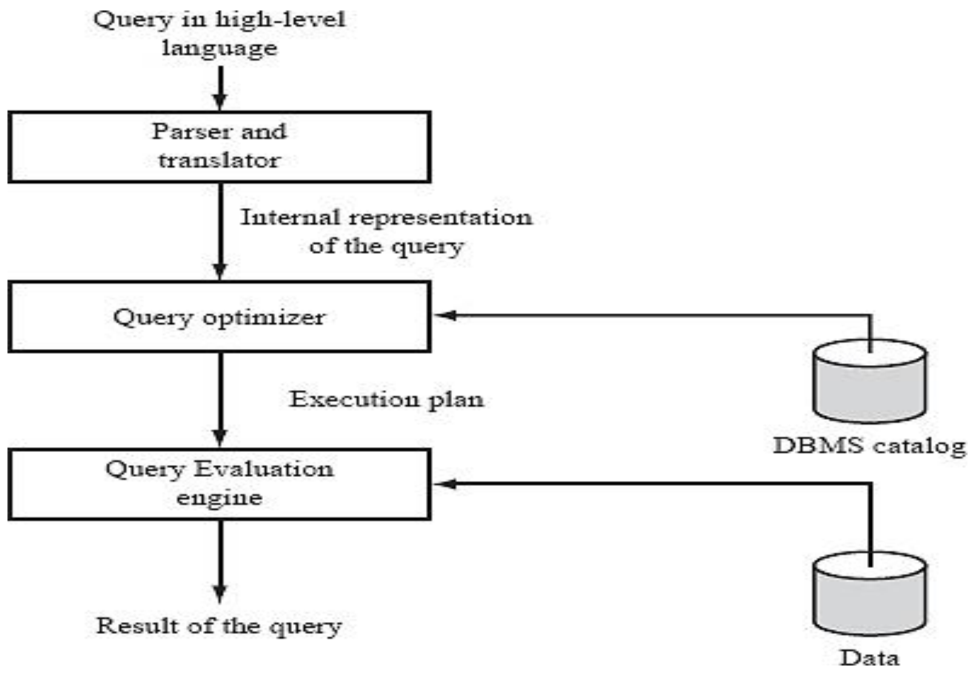


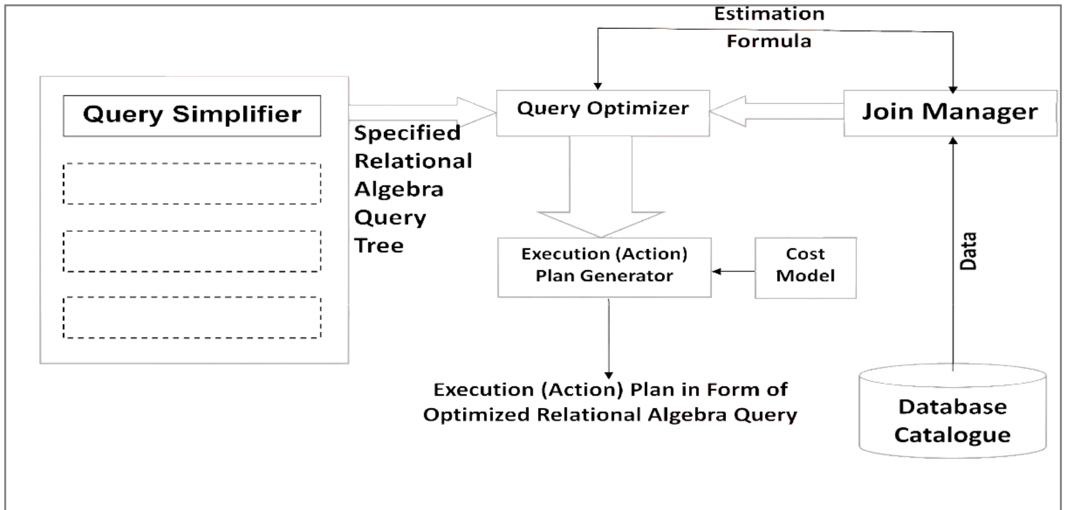
Fig: Query Processing steps

1. Parsing and translation

Check syntax and verify relations. Translate the query into its internal form. This is then translated into relational algebra.

2. Optimization

The primary goal of query optimization is of choosing an efficient execution strategy for processing a query. The query optimizer attempts to minimize the use of certain resources (mainly the number of I/O and CPU time) by selecting a best execution plan (access plan). A query optimization start during the validation phase by the system to validate the user has appropriate privileges. Simply, generate an optimal evaluation plan (with lowest cost) for the query plan is called optimization.


Estimation Formula
Query Simplifier
Specified Relational Algebra Query Tree
Query Optimizer
Join Manager
Execution (Action) Plan Generator
Cost Model
Data
Execution (Action) Plan in Form of Optimized Relational Algebra Query
Database Catalogue

Detail Block Diagram of Query Optimization

3. Evaluation

The query-execution engine takes an (optimal) evaluation plan, executes that plan, and returns the answers to the query.

Query Optimization

The primary goal of query optimization is of choosing an efficient execution strategy for processing a query. DBMS provides two different approaches to query optimization: rule based and cost-based. With the rule-based approach, the optimizer chooses execution plans based on heuristically ranked operations. However, the rule-based approach is being phased out in favor of the cost-based approach, where the optimizer examines alternative access paths and operator algorithms and chooses the execution plan with the lowest estimated cost. The estimated query cost is proportional to the expected elapsed time needed to execute the query with the given execution plan. The optimizer calculates this cost based on the estimated usage of resources, such as I/O, CPU time, and memory needed. The goal of cost-based optimization is to minimize the elapsed time to process the entire query.

Example: Transformation of an SQL-query into an RA-query and find optimized query

Employee (Eno, Ename, Title)

Project (Eno, Pno, Salary, Duration)

Query: Find the names of employees who are managing a project of duration greater than 5 years?

High level query:

```
SELECT Ename
FROM Employee E, Project P
WHERE E.Eno=P.Eno AND Duration>5;
```

Two possible transformations of the query are:

Expression 1: Π_{Ename} ($\sigma_{E.Eno=P.Eno \wedge Duration>5}$ (Employee X Project))

Expression 2: Π_{Ename} (Employee $\bowtie$ ($\sigma_{Duration>5}$ (Project))

Expression 2 avoids the expensive and large intermediate Cartesian product, and therefore typically is better.

Transaction Management

A transaction is a collection of several operations on the database appears to be a single unit from the point of view of the database user. For example, a transfer of funds from a checking account to a savings account is a single operation from the customer's standpoint; within the database system, however, it consists of several operations.

Database transaction is collection of SQL queries which forms a logical one task. For transaction to be completed successfully all SQL queries has to run successfully. Database transaction executes either all or none. For example, if your database transaction contains 4 SQL queries and one of them fails then change made by other 3 queries will be rolled back. This way your database always remain consistent whether transaction succeeded or failed.

Transaction is implemented in database using SQL keyword TRANSACTION, COMMIT and ROLLBACK.

- COMMIT writes the changes made by transaction into database
- ROLLBACK removes temporary changes logged in transaction log by database transaction.

Example:

Operation (Money Transfer)	**T1**
Read balance of account A1	Read (A1)
Subtract 20,000 from A1	A1=A1-20000
Update balance of A1	Write (A1)
Read balance of account A2	Read (A2)
Add Rs 20,000 to A2	A2=A2+20000

Table: Money Transfer Transaction

Why transaction is required in database?

Database is used to store data required by real life application e.g. Banking, Healthcare, Finance etc. All your money stored in banks is stored in database. In order to protect data and keep it consistent any changes in this data needs to be done in transaction so that even in case of failure data remain in previous state before start of transaction. Consider a Classical example of ATM (Automated Tailor Machine); we all use to withdraw and transfer money by using ATM. If you break withdrawal operation into individual steps you will find:

- Verify account details.
- Accept withdrawal request
- Check balance
- Update balance
- Dispense money

Suppose your account balance is 1000$ and you make a withdrawal request of 900$. At fourth step your balance is updated to 900$ and ATM machine stops working due to power outage. Once power comes back and you again tried to withdraw money you surprised by seeing your balance just 100$ instead of 1000$. This is not acceptable by any person in the world) so we need transaction to perform such task.

Properties of Transaction

There are four important properties of database transactions these are represented by acronym ACID and also called ACID properties or database transaction where:

- **Atomicity:** Atom is considered to be smallest particle which cannot be broken into further pieces. Database transaction has to be atomic means either all steps of transaction completes or none of them.
- **Consistency:** Transaction must leave database in consistent state even if it succeed or rollback.

- **Isolation:** Two database transactions happening at same time should not affect each other and has consistent view of database. This is achieved by using isolation levels in database.
- **Durability:** Data has to be persisted successfully in database once transaction completed successfully and it has to be saved from power outage or other threats. This is achieved by saving data related to transaction in more than one places along with database.

Transaction States

Whenever a transaction is submitted to a DBMS for execution, either it executes successfully or fails due to some reasons. During its execution, a transaction passes through various states that are active, partially committed, committed, failed, and aborted.

- **Active state -** It is initial state. Transaction stays in this state while it is executing.
- **Partially committed state -** After the final statement has been executed, a transaction is in partially committed state.
- **Committed state -** After successful completion, a transaction is in committed state.
- **Failed state -** After the discovery that normal execution can no longer proceed, a transaction is in failed state.
- **Terminated State** – This state corresponds to the transaction leaving the system. The transaction information that is maintained in system tables while the transaction has been running is removed when the transaction terminates. Failed or aborted transactions may be restarted later – either automatically or after being resubmitted by the user – as brand new transactions.

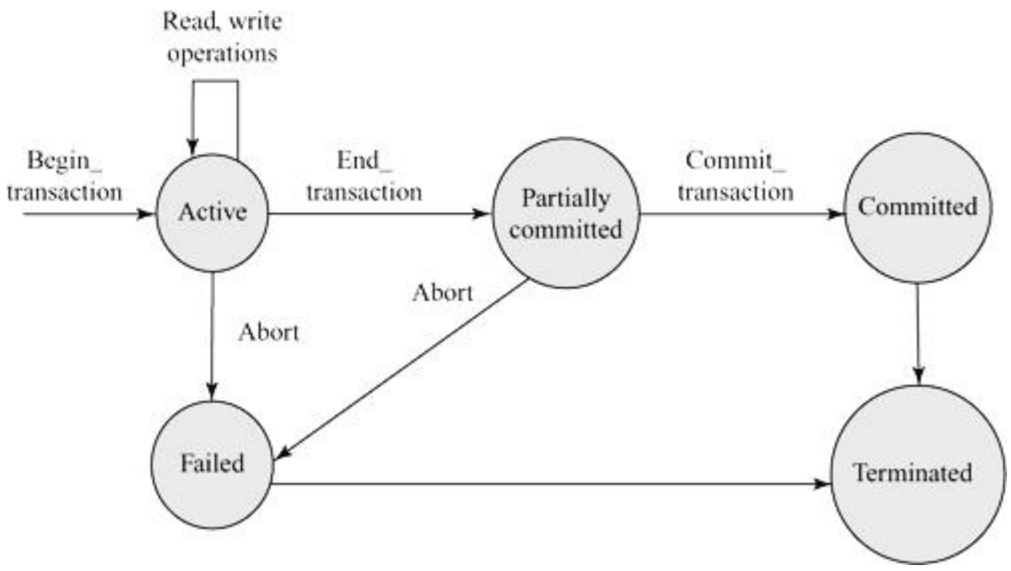


Fig: State diagram of Transaction

Concurrency Control

When multiple transactions are trying to access the same sharable resource, there could arise many problems if the access control is not done properly. There are some important mechanisms to which access control can be maintained. Earlier we talked about theoretical concepts like serializability, but the practical concept of this can be implemented by using Locks and Timestamps. Here we shall discuss some protocols where Locks and Timestamps can be used to provide an environment in which concurrent transactions can preserve their Consistency and Isolation properties.

Objectives of concurrency control mechanism can be list as below:

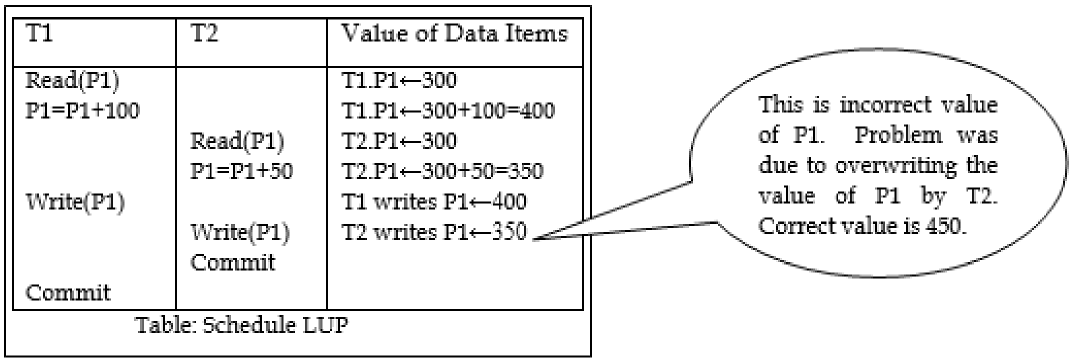
- To enforce Isolation (through mutual exclusion) among conflicting transactions.
- To preserve database consistency through consistency preserving execution of transactions.
- To resolve read-write and write-write conflicts.

Why concurrency Control needed?

If transactions are executed serially, i.e., sequentially with no overlap in time, no transaction concurrency exists. However, if concurrent transactions with interleaving operations are allowed in an uncontrolled manner, some unexpected, undesirable result may occur. Here are some typical examples:

- **The Lost Update Problem**: This problem occurs when two transactions that access the same database items have their operations interleaved in such a way that value of the data item written by one transaction is overlapped by another transaction and hence results to the incorrect value of the database item. Consider the example given below where two concurrent transactions try to update inventory of product P1 simultaneously.

Example: Assume initial value of data item P1 is 300



T1	T2	Value of Data Items
Read(P1)		T1.P1←300
P1=P1+100		T1.P1←300+100=400
	Read(P1)	T2.P1←300
	P1=P1+50	T2.P1←300+50=350
Write(P1)		T1 writes P1←400
	Write(P1)	T2 writes P1←350
	Commit	
Commit		

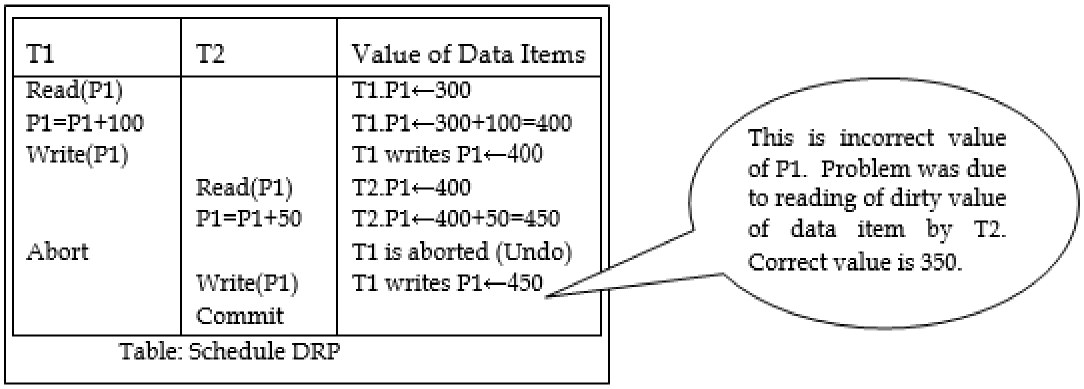
This is incorrect value of P1. Problem was due to overwriting the value of P1 by T2. Correct value is 450.

Table: Schedule LUP

- **The Dirty Read (Temporary Update) Problem:**

  This problem occurs when one transaction updates a database item and then the transaction fails for some reason. The updated item is accessed by another transaction before it is changed back to its original value which causes transaction T2 to have dirty value of the data item. Again consider the example where two concurrent transactions try to update inventory of product P1 simultaneously. This example is same as above example except that its operations are interleaved in slightly different way and first transaction is failed.

Example: Assume initial value of data item P1 is 300



T1	T2	Value of Data Items
Read(P1)		T1.P1←300
P1=P1+100		T1.P1←300+100=400
Write(P1)		T1 writes P1←400
	Read(P1)	T2.P1←400
	P1=P1+50	T2.P1←400+50=450
Abort		T1 is aborted (Undo)
	Write(P1)	T1 writes P1←450
	Commit	

This is incorrect value of P1. Problem was due to reading of dirty value of data item by T2. Correct value is 350.

Table: Schedule DRP

Log based protocol for concurrency control

One way to ensure serializability is to access the data items in mutually exclusive manner. This means while one transaction is accessing data item, no other transaction should be allowed to access the data item. Lock variables are most commonly used approach for achieving mutual exclusion. Locks are of two kinds: Binary Locks and Shared/Exclusive Locks.

Binary lock

Binary lock is a variable that can be only in two states. It is either locked or unlocked. Normally, locked state is represented by value 1 and unlocked state is represented by value 0. A distinct lock is associated with each database item x. If the value of the lock on data item x is 1, item x cannot be accessed by a database operation that requests the item. If the value of the lock on x is 0, the item can be accessed when requested. Two operations, lock and unlock, are used with binary locking and these two operations must be implemented atomically.

If the simple binary locking scheme described above is used, every transaction must obey the following rules:

1. A transaction T must issue the operation lock(x) before performing any read(x) or write(x) operations.
2. A transaction T must issue the operation unlock(x) after finishing all read(x) and write(x) operations.
3. A transaction T will not issue a lock(x) operation if the data item x is already locked by it.
4. A transaction T will not issue an unlock(x) operation if the data item x is not locked by it.

Shared/Exclusive Lock

This type of lock is also called multiple mode lock. It is a variable that can be in any of three states: unlocked, read-locked (shared lock) and write-locked (exclusive lock). If a transaction acquires shared lock (read-lock) on data item x then other transactions can also acquire shared lock on data item x. But no transaction can acquire exclusive lock (write-lock) on data item x. On the other hand, if a transaction acquires exclusive lock (write-lock) on data item x then no other transactions can acquire shared/exclusive lock (read/write lock) on the data item.

When shared/exclusive locking scheme discussed above is used, the system must enforce the following rules:

1. A transaction T must issue the operation read_lock(x) or write_lock(x) before performing read(x) operation.
2. A transaction T must issue the operation write_lock(x) before performing write(x) operation.
3. A transaction T must issue the operation unlock(x) after finishing all its read(x) and write(x) operations.
4. A transaction T will not issue a read_lock(x) operation if it already holds a shared lock (read-lock) or exclusive lock (write-lock) on item x.
5. A transaction T will not issue a write_lock(x) operation if it already holds a shared lock (read-lock) or exclusive lock (write-lock) on item x.
6. A transaction T will not issue an unlock(x) operation if it does not hold a shared lock (read-lock) or exclusive lock (write-lock) on item x.

Two-Phase Locking Protocol

The two-phase locking protocol is used to ensure the serializability in Database. This protocol is implemented in two phase. *Growing Phase* and *Shrinking Phase***.**

- **Growing Phase**: In this phase we put read or write lock based on need on the data. In this phase we does not release any lock. Remember that all lock operation must precede first unlock operation appeared in a transaction.
- **Shrinking Phase:** This phase is just reverse of growing phase. In this phase we release read and write lock but doesn't put any lock on data. Unlock operations can only appear after last lock operation.

For a transaction these two phases must be mutually exclusive. This means, during locking phase unlocking phase must not start and during unlocking phase locking phase must not begin.

T1	T2	Data Items Values
Lock(x)		
Read(x)		x←50
x=x+100		x←150
Write(x)		T1 writes x←150
Lock(y)		
Unlock(x)		
	Lock(x)	
	Read(x)	x←150
	x=x*2	x←300
	Wtite(x)	T2 writes x←300
Read(y)		y=50
y=y+100		y=150
Write(y)		T1 writes y←150
Unlock(y)		
	Lock(y)	
	Unlock(x)	
	Read(y)	y←150
	y=y*2	y←300
	Write(y)	T2 writes y←300
	Unlock(y)	

Time-Stamp Based Protocol

The most commonly used concurrency protocol is time-stamp based protocol. This protocol uses either system time or logical counter to be used as a time-stamp. Lock based protocols manage the order between conflicting pairs among transaction at the time of execution whereas time-stamp based protocols start working as soon as transaction is created. Every transaction has a time-stamp associated with it and the ordering is determined by the age of the transaction. A transaction created at 0002 clock time would be older than all other transaction, which come after it. For example, any transaction 'y' entering the system at 0004 is two seconds younger and priority may be given to the older one.

A timestamp can be implemented in two ways. The simplest one is to directly assign the current value of the clock to the transaction or the data item. The other policy is to attach the value of a logical counter that keeps incrementing as new timestamps are required.

The concurrency control algorithm must check whether convicting operations violate the time stamp ordering in the following two cases.

Case I: Transaction T Issues Write(x) Operation

- If ReadTS(x) > TS(T) or if WriteTS (x) > TS(T), then abort and roll back T and reject the Operation. This should be done because some younger transaction with a time stamp greater than TS(T) – and hence after T in the timestamp ordering –has already read or written the value of item x before T had a chance to write x, thus violating the timestamp ordering.

- If the above condition is not satisfied then execute the Write(x) operation and set WriteTS(x) to TS(T).

Case II: Transaction T Issues Read(x) Operation

- If WriteTS (x) >TS(T), then abort and rollback and reject the operation. This should be done because some younger transaction have timestamp greater than TS (T) – and hence after T in the Timestamp ordering-has already written the value of item x before T had a chance to read x.
- If write TS(x) < = TS (T), then execute the Read(x) operation of T and set ReadTS(x) to the larger to TS(T) and the current ReadTS(x).

Example: Assume timestamps of T1 and T2 is 100 and 110 respectively and initial value of x is 500

T1	T2	Timestamps
Read(x)		ReadTS(x)←100
	Read(x)	ReadTS(x)←110
x=x+200		
	x=x+200	
	Write(x)	WriteTS(x)←110
Write(x)		*WriteTS(x)>TS(T1) and hence*
Abort T1		*Timestamp order violated*

Database performance tuning

Although newer relational databases and faster hardware run most SQL queries with a significantly small response time, there is always room for improvement. After a database is deployed and is in operation, actual use of the applications, transactions, queries, and views reveals factors and problem areas that may not have been accounted for during the initial physical design. Thus database tuning is a process of minimizing response time or improving performance by using various optimization techniques during designing, querying, transactions etc.

The goals of tuning are as follows:

- To make applications run faster.
- To improve (lower) the response time of queries and transactions.
- To improve the overall throughput of transactions.

Tuning a database involves dealing with the following types of problems:

- How to avoid excessive lock contention, thereby increasing concurrency among transactions.
- How to minimize the overhead of logging and unnecessary dumping of data.
- How to optimize the buffer size and scheduling of processes.

- How to allocate resources such as disks, RAM, and processes for most efficient utilization.

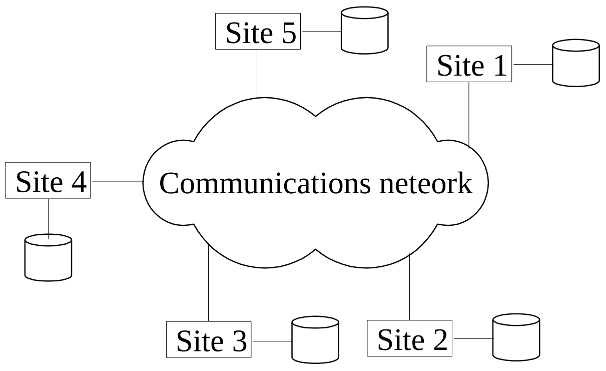
Tuning of database may occur in following sections:

- Tuning of Index
- Tuning of database design
- Tuning of query

Index tuning means reconstructing of index if overflow of data may occur in the table. Tuning of database design means reconstructing overall structure of database and keep in normal form if we need to insert additional requirements to the existing database system. And tuning of query means use less coasty clauses during writing SQL of given statement.

Distributed Database

A distributed database (DDB) is a collection of multiple, logically interrelated databases distributed over a computer network. Distributed databases bring the advantages of distributed computing to the database management domain. It consists of a number of processing elements, not necessarily homogenous, that are interconnected by a computer network, and that cooperate in performing certain assigned tasks. As a general goal, distributed computing systems partition a big, unmanageable problem into smaller pieces and solve it effectively in a coordinated manner.
It provides two major benefits: more computer power can be used to solve a complex task and each autonomous processing element can be managed independently and develop its own applications.



Distributed Database Management System

A distributed database management system (D–DBMS) is the software that manages the DDB and provides an access mechanism that makes this distribution transparent to the users. It consists of a single logical database that is split into a number of fragments. Each fragment is stored on one or more computers under the control of a separate DBMS, with the computers connected by a communications network. Each site is capable of independently processing user requests that require access to local data and is also capable of processing data stored on other computers in the network.

Functions of Distributed database system

- **Distribution and Network transparency**
  Users do not have to worry about operational details of the network. There is Location transparency, which refers to freedom of issuing command from any location without affecting its working. Then there is naming transparency, which allows access to any names object (files, relations, etc.) from any location.
- **Replication transparency:** It allows to store copies of a data at multiple sites for better availability, performance, and reliability making the user unaware of the existence of copies.
- **Fragmentation transparency:** Allows to fragment a relation horizontally (create a subset of tuples of a relation) or vertically (create a subset of columns of a relation) making the user unaware of the existence of fragments.
- **Increased reliability and availability:** Reliability refers to system live time, that is, system is running efficiently most of the time. Availability is the probability that the system is continuously available (usable or accessible) during a time interval. A distributed database system has multiple nodes (computers) and if one fails then others are available to do the job.
- **Improved performance:** A distributed DBMS fragments the database to keep data closer to where it is needed most. This reduces data management (access and modification) time significantly.
- **Easier expansion (scalability):** Allows new nodes (computers) to be added anytime without chaining the entire configuration.
- **Autonomy:** Gives the users tighter control over their own local database.

Additional Functions

- **Keeping track of data:** The ability to keep track of the data distribution, fragmentation, and replication by expanding the DDMS catalog.
- **Distributed query processing:** The ability to access remote sites and transmit queries and data among the various sites via a communication network.
- **Distributed transaction management:** The ability to device execution strategies for queries and transactions that access data from more than one site and to synchronize the access to distributed data and maintain integrity of the overall database.

- **Replicated data management:** The ability to decide which copy of a replicated data item to access and to maintain the consistency of copies of a replicated data item.
- **Distributed database recovery:** The ability to recover from individual site crashes and from new type failures such as the failure of communication link.
- **Security:** Distributed transactions must be executed with the proper management of the security of the data and the authorization/access privileges of users.
- **Distributed directory (catalog) management:** A directory contains information (metadata) about data in the database. The directory may be global for the entire DDB, or local for each site. The placement and distribution of the directory are design and policy issues.

These above functions (in addition to those of a centralized DBMS) themselves increase the complexity of DDBMS over a centralized DBMS.

Data Fragmentation, Replication, and Allocation Techniques

Data Fragmentation: A process of splitting a relation into logically related and correct parts. Fragmentation consists of breaking a relation into smaller relations or fragments, and storing the fragments (instead of the relation itself), possibly at different sites. In horizontal fragmentation, each fragment consists of a subset of rows of the original relation. In vertical fragmentation, each fragment consists of a subset of columns of the original relation. A relation can be fragmented in three ways:

- Horizontal Fragmentation
- Vertical Fragmentation
- Mixed Fragmentation.

Horizontal Fragmentation:

A horizontal fragment of a relation is a subset of the tuples in that relation. The tuples that belong to the horizontal fragment are specified by a condition on one or more attributes of the relation. Horizontal fragmentation divides a relation "horizontally" by grouping rows to create subsets of tuples, where each subset has a certain logical meaning. These fragments can then be assigned to different sites in the distributed system. Derived horizontal fragmentation applies the partitioning of a primary relation to other secondary relations which are related to the primary via a foreign key.

It is a horizontal subset of a relation which contains those of tuples which satisfy selection conditions specified in the SELECT operation of the relational algebra on single or multiple attributes. Consider the Customer relation with selection condition (sex= male). All tuples satisfy this condition will create a subset which will be a horizontal fragment of Customer relation.

Customer

Customer_id	Name	Area	Payment_Type	Sex
1	Nicky	Ktm	Cash	Female
2	Geeta	Lalitpur	Credit card	Female
3	Ruby	Ktm	Cash	Female
4	Aayan	Pokhara	Cash	Male
5	Aarav	Palpa	Credit card	Male
6	Mina	Pokhara	Cash	Female
7	Umesh	Chandani	Credit card	Male

Horizontal Fragmentation are subsets of tuples (rows)

σsex= male(customer)

Fragment 1

Customer_id	Name	Area	Payment_Type	Sex
4	Aayan	Pokhara	Cash	Male
5	Aarav	Palpa	Credit card	Male
7	Umesh	Chandani	Credit card	Male

Fragment 2

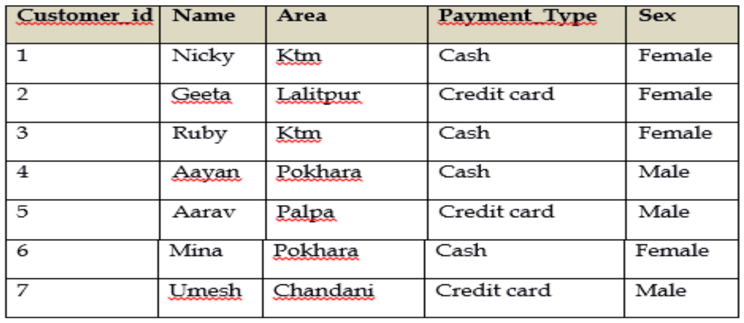
σsex=female(customer)

Customer_id	Name	Area	Payment_Type	Sex
1	Nicky	Ktm	Cash	Female
2	Geeta	Lalitpur	Credit card	female
3	Ruby	Ktm	Cash	female
6	Mina	Pokhara	Cash	female

Vertical Fragmentation:

Vertical fragmentation divides a relation "vertically" by columns. A vertical fragment of a relation keeps only certain attributes of the relation. It is a subset of a relation which is created by a subset of columns. Thus a vertical fragment of a relation will contain values of selected columns. There is no selection condition used in vertical fragmentation. All vertical fragments of a relation are connected by using PROJECT operation of the relational algebra.

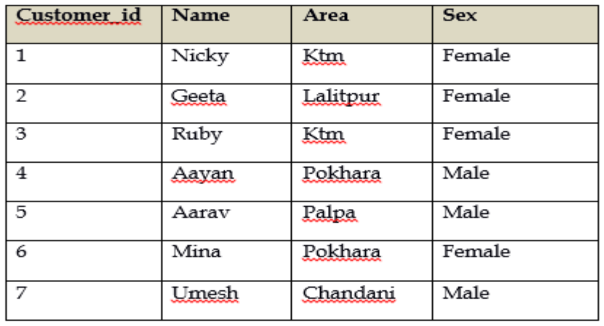
Example:



Customer_id	Name	Area	Payment_Type	Sex
1	Nicky	Ktm	Cash	Female
2	Geeta	Lalitpur	Credit card	Female
3	Ruby	Ktm	Cash	Female
4	Aayan	Pokhara	Cash	Male
5	Aarav	Palpa	Credit card	Male
6	Mina	Pokhara	Cash	Female
7	Umesh	Chandani	Credit card	Male

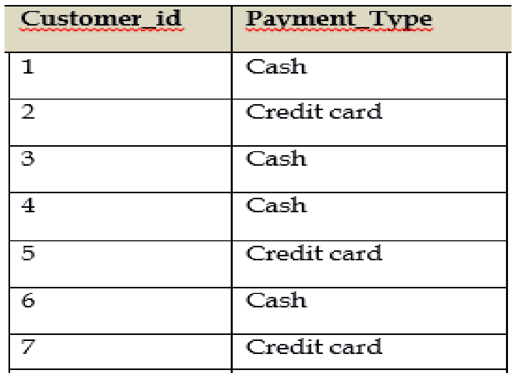
Vertical fragmentation is subset of attributes

Fragment 1



Customer_id	Name	Area	Sex
1	Nicky	Ktm	Female
2	Geeta	Lalitpur	Female
3	Ruby	Ktm	Female
4	Aayan	Pokhara	Male
5	Aarav	Palpa	Male
6	Mina	Pokhara	Female
7	Umesh	Chandani	Male

Fragment 2



Customer_id	Payment_Type
1	Cash
2	Credit card
3	Cash
4	Cash
5	Credit card
6	Cash
7	Credit card

To combine all the vertically fragmented tables we need to perform join operation on the fragments.

```
SELECT customer_id, Name, Area, Sex, Payment_type
FROM Fragment 1 NATURAL JOIN Fragment 2;
```

Mixed (Hybrid) Fragmentation

We can intermix the two types of fragmentation, yielding a mixed fragmentation. The original relation can be reconstructed by applying UNION and OUTER UNION (or OUTER JOIN) operations in the appropriate order. In general a fragment of a relation can be specified by SELECT-PROJECT combination of operations which is represented by $\Pi_L (\sigma_C (R))$.

Data Replication and Allocation

Replication is useful in improving the availability of data. The most extreme case is replication of the whole database at every site in the distributed system, thus creating a fully replicated distributed database. This can improve availability remarkably because the system can continue to operate as long as at least one site is up. It also improves performance of retrieval for global queries because the results of such queries can be obtained locally from any one site; hence, a retrieval query can be processed at the local site where it is submitted, if that site includes a server module. The disadvantage of full replication is that it can slow down update operations drastically, since a single logical update must be performed on every copy of the database to keep the copies consistent. This is especially true if many copies of the database exist. Full replication makes the concurrency control and recovery techniques more expensive than they would be if there was no replication.

In partial replication some selected part is replicated to some of the sites. Data replication is achieved through a replication schema.

The process of assigning each fragment to a particular site in a distributed system is called data distribution (or **data allocation**). This is relevant only in the case of partial replication or partition. The selected portion of the database is distributed to the database sites. The choice of sites and the degree of replication depend on the performance and availability goals of the system and on the types and frequencies of transactions submitted at each site. For example, if many updates are performed, it may be useful to limit replication.

Types of Distributed Database System

- Homogeneous distributed database system
- Heterogeneous distributed database system

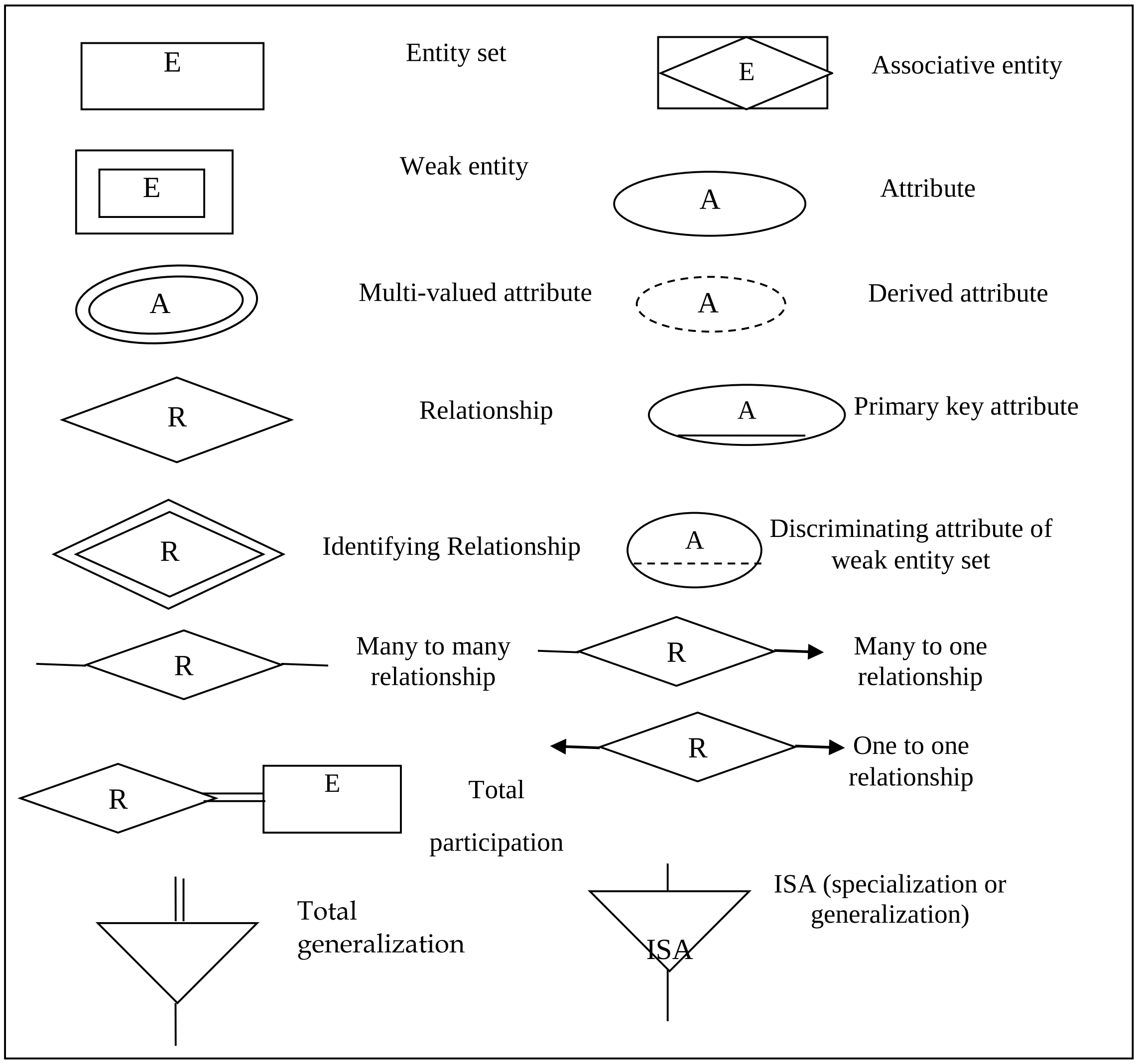
If all servers (or individual local DBMSs) use identical software and all users (clients) use identical software, the DDBMS is homogenous; otherwise, it is heterogeneous.

Unit 2
The Entity Relationship, Extended Entity Relationship Model And Object Model

E-R Model

It is developed to facilitate database design by allowing the specification of an enterprise schema, which represents overall logical structure of database. E-R model is useful in mapping the meanings and interactions of real word objects. Basic objects called entities. Relationship among objects called relationship. E-R model keep the record of entities, their attributes and relationship among those entities.

Symbols Used in ER Diagram



E-R data model consists of three basic notions.

a. Entity sets
b. Relationship sets
c. Attributes

Entity Sets

An entry set is a set of entities of the same type that share the same properties, or attributes for example the set of all persons who are customer at a given bank can be defined as the entity set customer, loan etc.

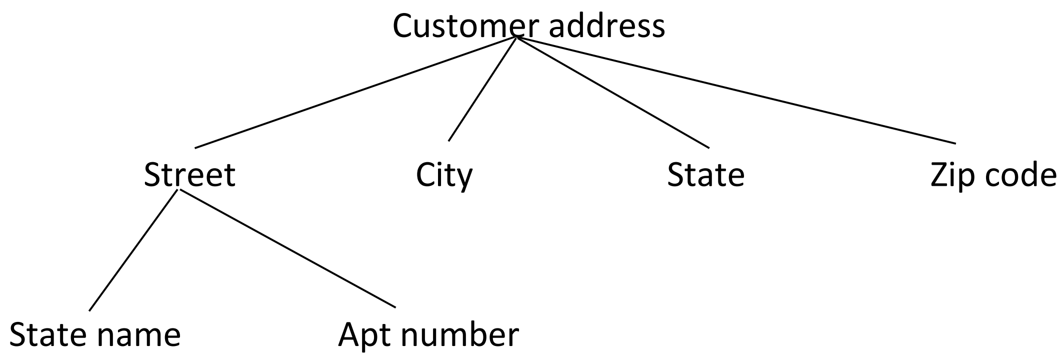
Attributes

An entity is represented by set of characteristics called attributes. Each attributes has set of values called domain. E.g. possible attributes of loan entity set has loan number and loan amount similarly the domain of attribute might be a set of a positive integers.

Types of attributes

1.Simple Vs. Composite:

Simple attributes are these they are not divided into sub parts. Composite attributes can be divided into subpart. A composite attribute is made of one or more simple or composite attributes. E.g. name is made of first name, middle name and last name and where name is composite attribute and first name, middle name and last name are simple attribute. It may come in hierarchy.



2. Single valued Vs. multivalued

A single valued attribute is described by one value e.g. age of a person. A multivalued attribute is described by many values e.g. color attribute of a multicolored bird.

3. Stored Vs. Derived

An attribute which value can be derived from the value of other attribute is called a derived attribute e.g. age can be derived from date of birth and current date. If this is not a case then called stored values e.g. roll no.

4. Null attributes

Attributes that do not have any applicable values are said to have null values. They are also called unknown and missing values. For example, if a social security value of a particular customer is null, we assume, the value is missing.

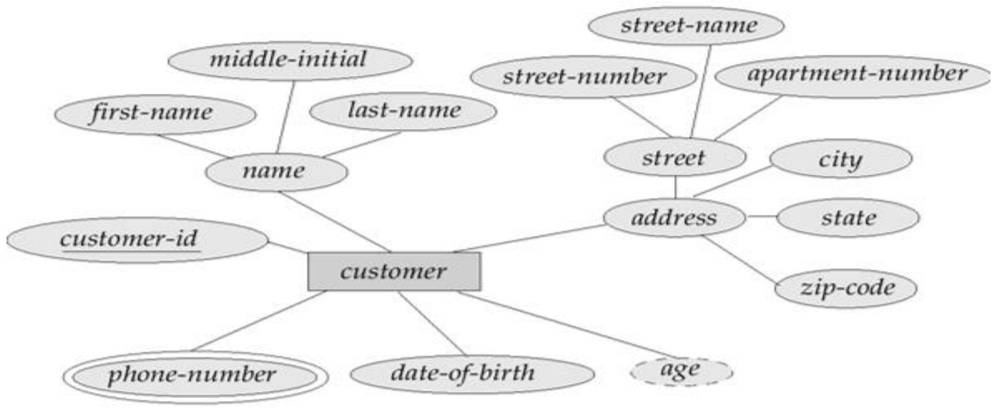
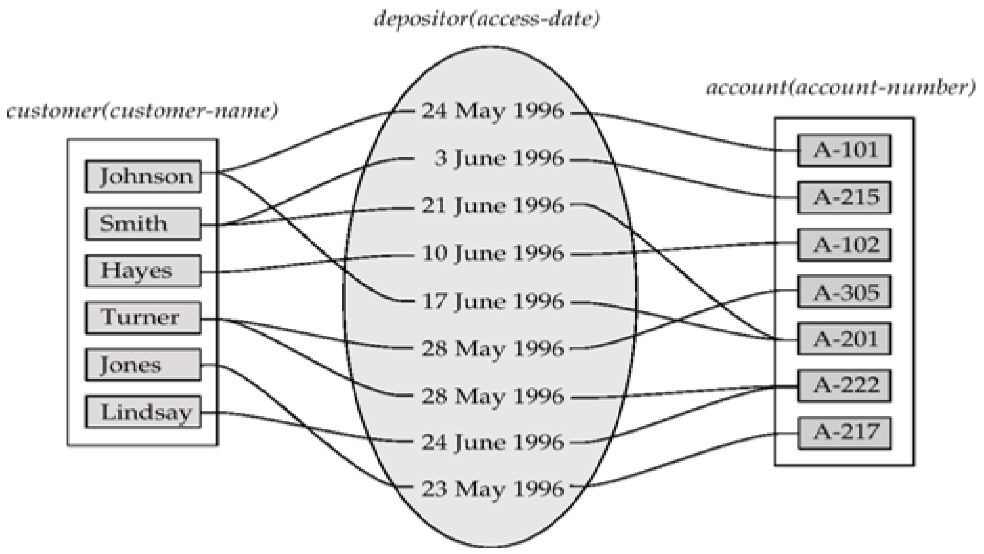


Fig: E-R diagram with *composite*, *multivalued*, and *derived* attributes

Descriptive Attributes

A relationship set may also have attributes called descriptive attributes. For example, the depositor relationship set between entity sets customer and account may have the attribute access-date. See in fig below. A relationship instance in a given relationship set must be uniquely identifiable from other relationship instances, without using descriptive attributes.



Relationship sets

A relationship is an association between several entities. A relationship set is a set of relationships of the same type. Mathematically For non-distinct entity set $n \geq 2$. If E1, E2… En are entity sets then relationship set R is the subset of $\{(e1, e2, e3… en) / e1 \in E1, e2 \in E2……..en \in En\}$.

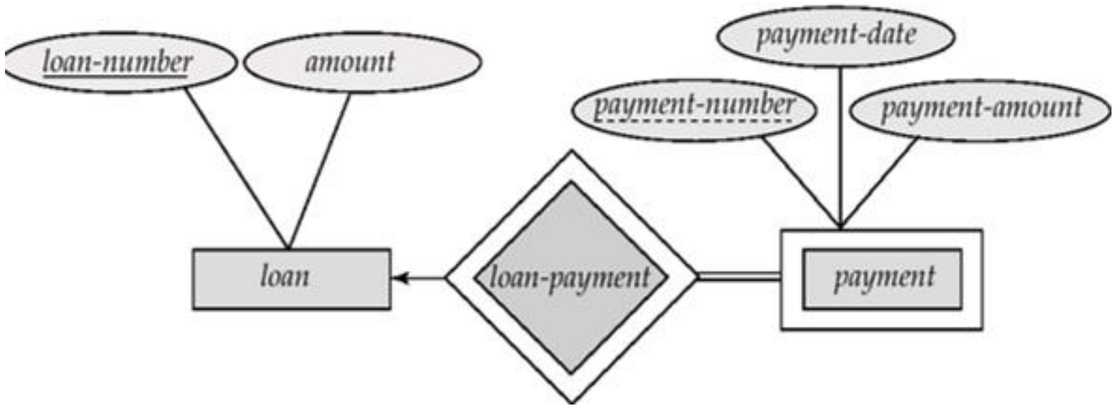
Strong and Weak Entity Set

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

For a weak entity set to be meaningful, it must be associated with another entity set, called the identifying or owner entity set, using one of the key attribute of owner entity set. The weak entity set is said to be existence dependent on the identifying entity set. The relationship associating the weak entity set with the identifying entity set is called the identifying relationship. The identifying relationship is many-to-one form the weak entity set to the identifying entity set, and the participation of the weak entity set in the relationship set is total.

Although a weak entity set does not have a primary key, we use discriminator (or partial key) as a set of attributes that allows the distinction to be made among all the entities in the weak entity set.

In the figure below, payment-number is partial key and (loan-number, payment-number) is primary key for payment entity set.



Constraints on ER Model

Relationship sets in ER model usually have certain constraints that limit the possible combinations of entities that may involve in the corresponding relationship set. Database content must confirm these constraints. The most important constraints are: mapping cardinalities and participation constraints.

Mapping Cardinality Constraints

ER model constraint that describes maximum number of possible relationship occurrences for an entity set participating in a given relationship type is called mapping cardinality. It is also termed as cardinality ratio. On the basis of cardinality ratio, relationships can be categorized into: One-to-One, One-to- Many, Many-to-One, and Many-to-Many. We express cardinality constraints by drawing either a directed line (→), signifying "one," or an undirected line (—), signifying "many," between the relationship set and the entity set.

1. One-to-One Relationship

If every entity in A is associated with at most one entity in B and vice-versa then the relationship is called one-to-one relationship. The following figure shows one to one mapping cardinality between entity sets A and B. For example every bank has only one CEO and a person can be CEO of only one bank therefore it shows one-to-one relationship between Bank and CEO.

2. One-to-Many Relationship

If an entity in A can be associated with any number (zero or more) of entities in B but every entity in B can be associated with at most one entity in A, and then it is called one-to-many relationship. For example, a mother can have any number of children but children can have only one mother therefore it shows one-to-many relationship between mother and child.

3. Many-to-One Relationship

If very entity in A can be associated only one of entities in B but an entity in B can be associated with any number of entities in A, then it is called many-to-one relationship. For example, a Book is always published by only one publisher but a publisher can publish any number of books therefore it shows many-to-one relationship between books and publication.

4. Many-to-Many Relationship

If an entity in A can be associated with any number of entities in B and vice versa then it is called many-to-many relationship. For example, a student can enroll into more than one subject and a subject can be enrolled by many students therefore it shows many-to-many relationship between students and courses.

Participation Constraints

Constraint on ER model that determines whether all or only some entity occurrences participate in a relationship is called participation constraint. It specifies whether the existence of an entity depends on its being related to another entity via the relationship type. There are two types of participation constraints:

- Total Participation Constraints and
- Partial Participation Constraints.

The participation of an entity set A in a relationship set R is said to be **total** if every entity in A participates in relationship at least once.

On the other hand, the participation of an entity set A in a relationship set R is said to be **partial** if only some of the members of an entity set A participate in relationship.

Total participation and partial participation is denoted by single line and double line in ER diagrams respectively.

For example, consider Customer and Loan entity sets in a banking system, and a relationship set borrower between them indicates that only some of the customers have Loan but every Loan should be associated with some customer. Therefore, there is total participation of entity set Loan in the relationship set borrower but participation of entity set customer is partial in relationship set borrower. Here, Loan entity set cannot exist without Customer entity set but existence of Customer entity set is independent of Loan entity set.

Keys

Set of one or more attributes whose values are distinct for each individual entity in the entity set is called key, and its values can be used to identify each entity uniquely. There are different types of keys which are:

- Super key
- Candidate key
- Primary key
- Composite key
- Foreign key

Super Key

A supper key is a set of one or more attributes allow us to identify uniquely in entity set. E.g. social security number attribute of a entity set customer is distinguish from one customer entity to another. Similarly, customer name and social-security is a supper key for an entity set customer. The customer name of entity customer is not super key because several people might have the same name.

Candidate key

A candidate key of an entity set is a minimal super key. That is a super key which does not have any proper subset is called candidate key. For example, student-id is candidate key of the entity set student but set of attributes {roll-number, name, program, semester, section} is not candidate key of the entity set student because it has proper subset {roll-number, program, semester section} which is also key. All candidate keys are super keys but vice versa is not true. Any candidate key other than the one chosen as a primary key is known as alternate key.

Primary key

A primary key is a candidate key that is chosen by the database designer as the principle means of uniquely identifying entities within an entity set. There may exist several candidate keys, one of the candidate keys is selected to be the primary key. For example, entity set student have two candidate keys: student-id, and {roll-number, program, semester section}, if database designer chooses student-id for the purpose of uniquely identifying entities within entity set then it becomes primary key. Primary key must satisfy following two characteristics:

- It cannot be null
- It cannot be duplicate

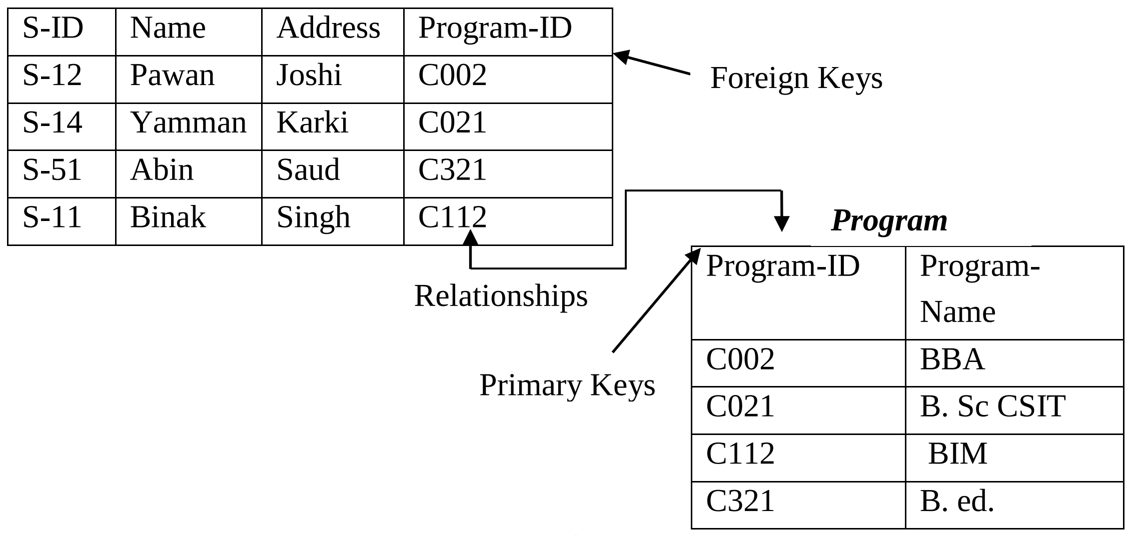
Composite Key

If a primary key contains more than one attribute, then it is called composite key. For example, if database designer chooses student-id as primary key then it not composite key but if database designer chooses {roll-number, program, semester section} as primary key then it is also called composite key.

Foreign key

A foreign key (FK) is an attribute or combination of attributes that is used to establish and enforce relationship between two relations (table). A set of attributes that references primary key of another table is called foreign key. For example, if a student enrolls in program then program-id (primary key of relation program) can be used as foreign key in student relation,

Student



S-ID	Name	Address	Program-ID
S-12	Pawan	Joshi	C002
S-14	Yamman	Karki	C021
S-51	Abin	Saud	C321
S-11	Binak	Singh	C112

Program

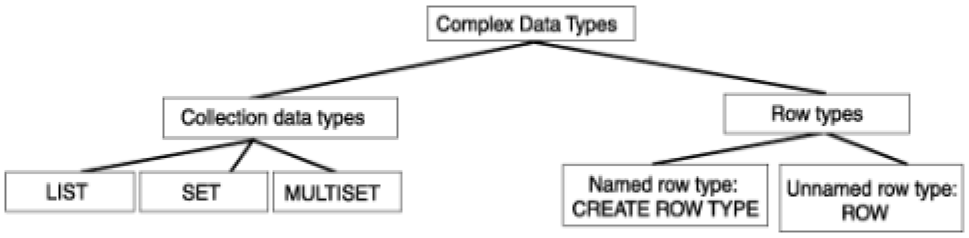
Program-ID	Program-Name
C002	BBA
C021	B. Sc CSIT
C112	BIM
C321	B. ed.

Fig: Primary key and foreign key

Complex Data Types

Any data that does not fall into the traditional field structure (alpha, numeric, dates) of a relational DBMS are called complex data. Examples of complex data types are bills of materials, word processing documents, maps, time-series, images and video. In a relational DBMS, complex data types are stored in a large object (LOB), but either the client application or some middleware is

required to process the data. In an object DBMS or an object-relational DBMS, complex data types are stored as objects that are integrated into and activated by the DBMS.
Complex data types are those that include record type and collections which are used to handle data in record format or in an array format. A complex data type is usually a composite of other existing data types. For example, you might create a complex data type whose components include built-in types, opaque types, distinct types, or other complex types. An important advantage that complex data types have over user-defined types is that users can access and manipulate the individual components of a complex data type.



Extended E-R model (EER model)

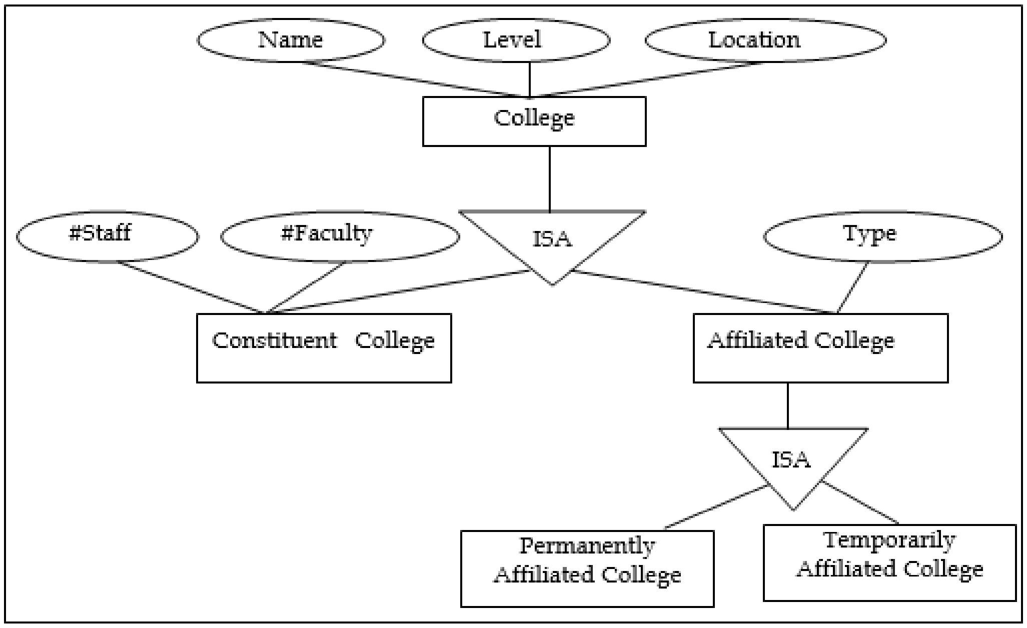
The ER modeling concepts are not sufficient for representing new database applications, which have more complex requirements than do the more traditional applications. To model modern complex systems such as, such as databases for engineering design and manufacturing (CAD/CAM), telecommunications, complex software systems, and Geographic Information Systems (GIS), among many other applications, it is not enough. Thus, modeling of such systems can be made easy with the help of extended ER model. The EER model includes all of the concepts introduced by the ER model. Additionally it includes the concepts of a subclass and super class, along with the concepts of specialization and generalization. There are basically four concepts of EER-Model:

- Subclass/super class relationship
- Specialization and Generalization
- categories (UNION types)
- Aggregation

Subclass/super class relationship

An entity set may have a number of sub-groupings of its entities that are meaningful. Here, the entity set is called superclass while the subgroupings are known as subclasses of the superclass. We call the relationship between a superclass and any one of its subclass a **superclass/subclass** or simply **class/subclass** relationship. An entity that is a member of subclass inherits all attributes from its superclass. That is, there is type inheritance in subclass. The entity also inherits all the relationships in which the superclass participates.

Consider the example of entity set college. We can divide the entity set college into two subgroups: constituent colleges and affiliated colleges. Further affiliated colleges can be divided into two subgroups: permanently affiliated colleges and temporarily affiliated colleges. Here the entity set college is superclass of subgroups entity sets constituent colleges and affiliated colleges and the subgroups are called subclasses of the superclass entity set college. In ER diagram we can represent superclass/subclass relationship by using ISA triangle as below:

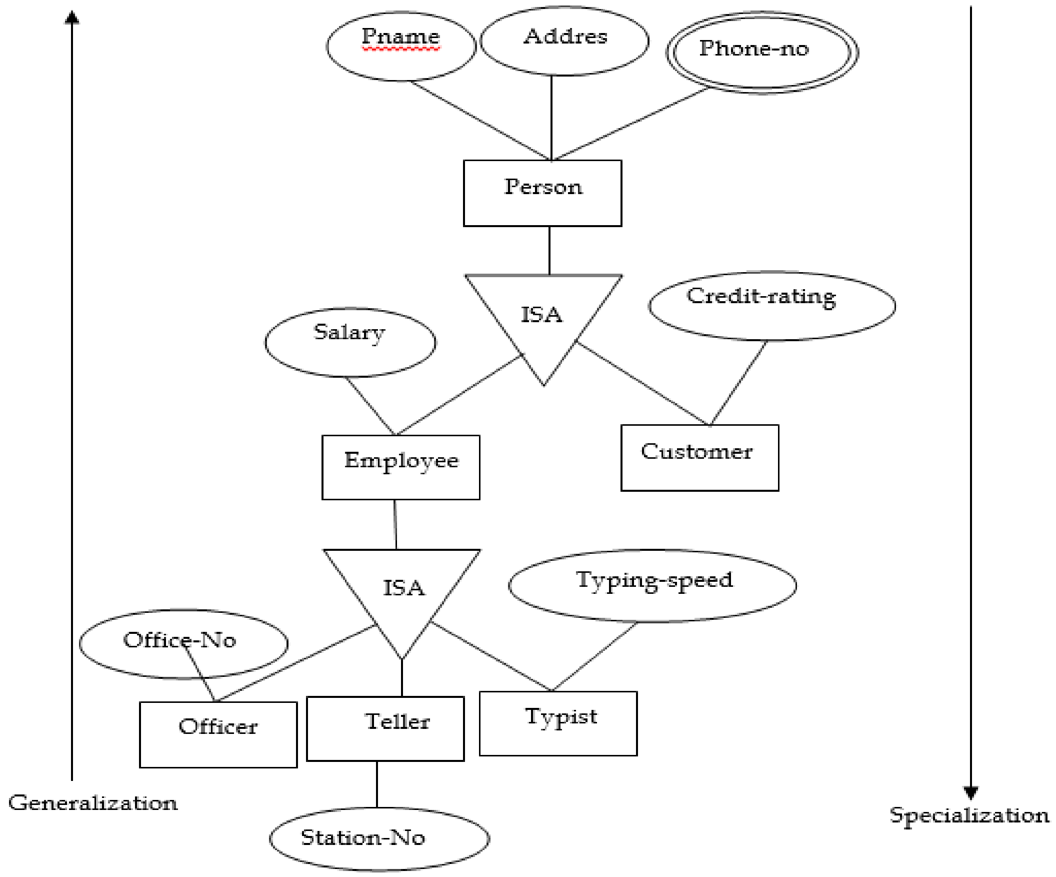


Specialization and Generalization

The process of defining a set of subclasses from a superclass is known as specialization. The set of subclasses is based upon some distinguishing characteristics of the entities in the superclass. It is a top-down design process. For Example, the entity set person can be specialized into entity sets employee, and customer. Further, on the basis of job type the entity set employee can be divided into entity sets manager, teller, typist etc. Thus, specialization may create hierarchy. Attributes of a subclass are called specific attributes. For example, the subclass entity set may have attribute 'typing-speed'.

There may have several specializations of the same superclass. For example, on the basis of pay type the entity set employee can be specialized entity sets "Full-time employees" and "Part-time" employees.

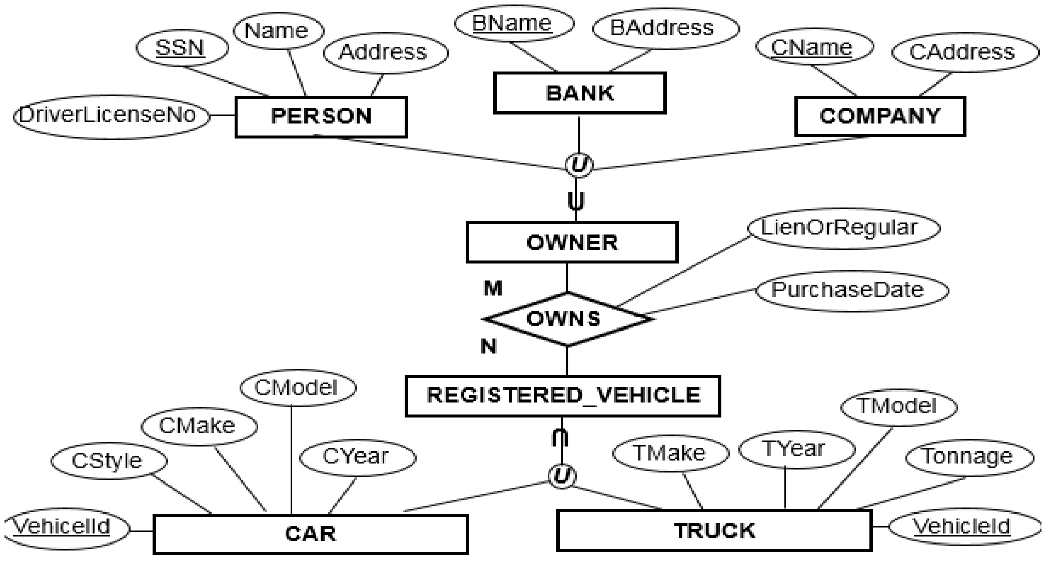
Generalization is the reverse of the specialization. Several classes with common features are generalized into a super class; original classes become its subclasses. It is a bottom-up design process. Here, we combine a number of entity sets that share the same features into a higher-level entity set. Designer applies generalization is to emphasize the similarities among the entity sets and hide their differences. Specialization and generalization are simple inversions of each other; they are represented in an E-R diagram in the same way. The terms specialization and generalization are used interchangeably.



Categories (Union Types)

It is possible that single superclass/subclass relationship has more than one super classes representing different entity types. In this case, the subclass will represent a collection of objects that is the union of distinct entity types; we call such a subclass a union type or a **category**.

A category has two or more super classes that may represent distinct entity types, whereas non-category superclass/subclass relationships always have a single superclass.



Aggregation

Sometimes we have to model a relationship between a collection of entities and relationships. Basic ER model cannot express relationships among relationship sets and relationships between relationship sets and entity sets. Aggregation is an abstraction through which relationship sets are treated as high-level entity sets and can participate in relationship sets. It allows relationships between relationships.

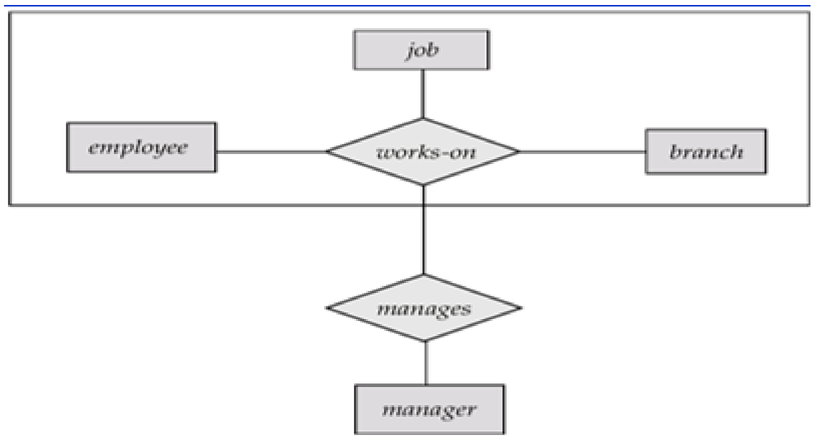


Fig: Aggregation

Constraints on Generalization/Specialization

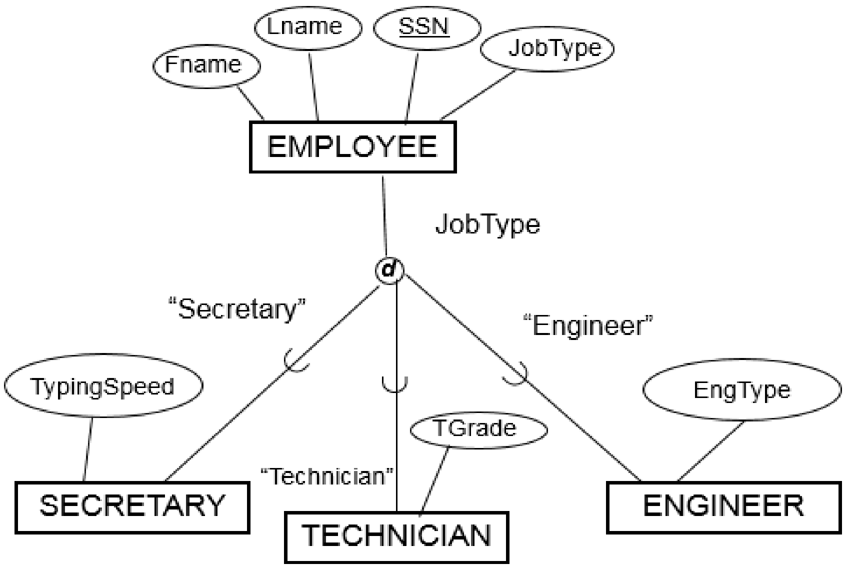
To model real world more accurately by using ER diagram we need to keep certain constraints on it. Constraints on which entities can be members of a given lower-level entity set are discussed below.

- Condition defined constraint
- Disjoint vs. Overlap Constraints

- A total specialization vs. a partial specialization

Condition defined constraint

If we can determine exactly those entities that will become members of each subclass by a condition, the subclasses are called **predicate-defined (or condition-defined**) subclasses. Here, condition is a constraint that determines subclass members. For example, if the EMPLOYEE entity type has an attribute **JobType**, we can specify the condition of membership in the SECRETRY subclass by a condition (JobType="secretary"), which we call defining predicate of the subclass.
We display predicate-defined subclass by writing the predicate condition next to the line that connects the subclass to the specialization circle.
If all subclasses have membership condition on the same attribute of the superclass, then it is called an **attribute defined-subclass**. And, the attribute is called the **defining attribute.**



Disjoint vs. Overlap Constraints

If an entity can be a member of at most one of the subclasses of the specialization, then the subclasses are called disjoint. In EER diagram, d in the circle stands for disjoint. For example, specialized subclasses 'Electronic Book' and 'Paperback Book' are disjoint subclasses of the super class entity set Book.
If the same entity may be a member of more than one subclass of the specialization, then the subclasses are said to be **overlapped**. For example, specialized subclasses 'Action Movie' and 'Comedy Movie' are overlapped subclasses of the super class entity set Movie. This is because there also movies called "Action Comedy" that belong both of the above subclasses.

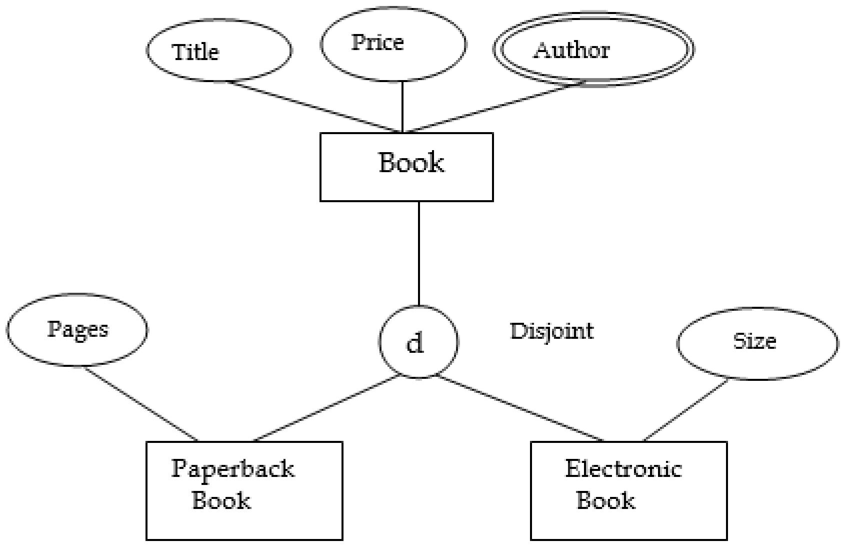


Fig: Disjoint constraint

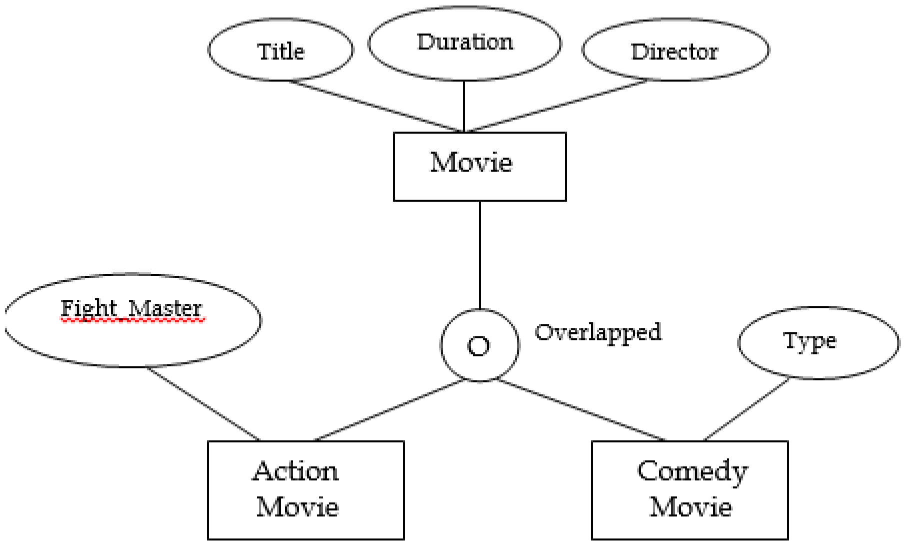
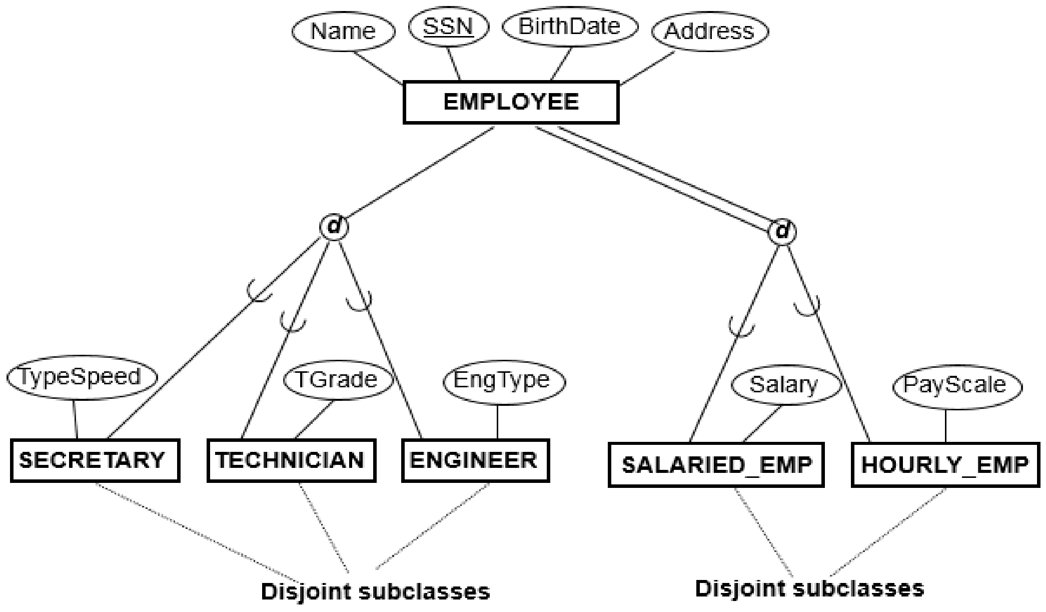


Fig: Overlapped constraint

A total specialization vs. a partial specialization

A total specialization constraint specifies that every entity in the superclass must be a member of some subclass in the specialization. It is represented by a double line connecting the superclass to the circle.

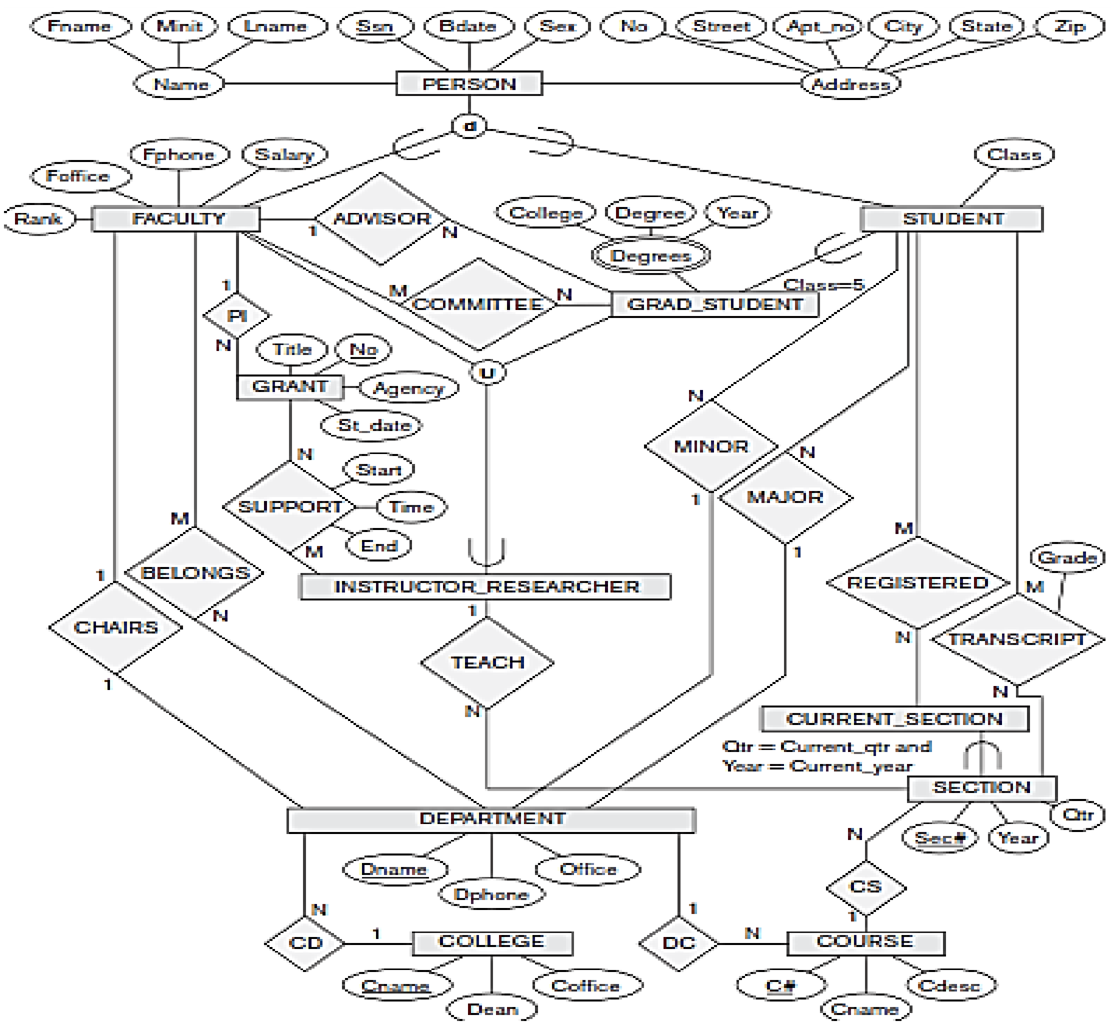
A partial specialization allows an entity not to belong to any of the subclasses, using a single line in EER. e.g., if some EMPLOYEE entities, for example, **sales representatives**, do not belong to any of the subclasses {SECRETARY, ENGINEER, TECHNICIAN}, then the specialization is partial. It is represented by a single line connecting the superclass to the circle. Partial generalization is the default.



Some insertion/deletion rules for specialization/generalization

- Deleting an entity from a superclass
    - it is automatically deleted from all the subclasses to which it belongs
- Inserting an entity in a superclass
    - it is mandatorily inserted in all predicate-defined (or attribute-defined) subclasses for which it satisfies the defining predicate
- Inserting an entity in a superclass of a total specialization
    - it is mandatorily inserted in at least one of subclasses

Example: The UNIVERSITY Database Example



ER-to-Relational Mapping Algorithm

- Step 1: Mapping of Regular Entity Types
- Step 2: Mapping of Weak Entity Types
- Step 3: Mapping of Multivalued attributes.
- Step 4: Mapping of Composite attributes.
- Step 5: Mapping of Binary 1:1 Relation Types
- Step 6: Mapping of Binary 1: N Relationship Types.
- Step 7: Mapping of Binary M: N Relationship Types.
- Step 8: Mapping of N-ary Relationship Types.

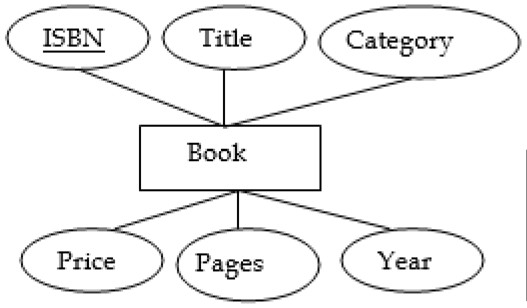
Mapping EER Model constructs to-Relational Algorithm

Step 1: Mapping Specialization or Generalization to relational model
Step 2: Mapping of Union Types (Categories) to relational model

Step 3: Mapping aggregation to relational model

Mapping regular entity set to relational model

An entity set is mapped to a relation in a straightforward way: Each simple attribute of the entity set becomes an attribute (column) of the table and primary key of the entity set becomes primary key of the relation. Domain of each attribute of the relation must be known.

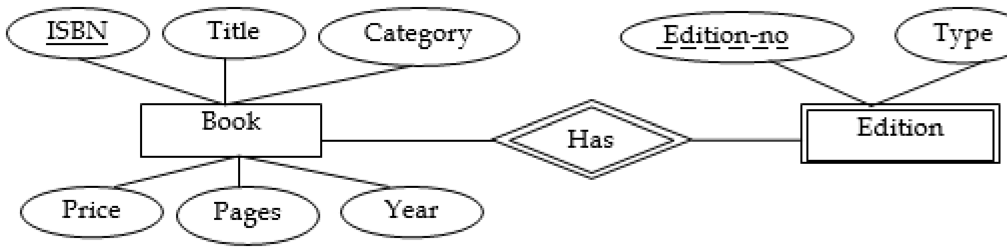


ISBN	Title	Category	Price	Pages	Year

Mapping of Weak Entity Types

A weak entity set does not have its own primary key and always participates in one-to-many relationship with owner entity set and has total participation. For a weak entity set create a relation that contains all simple attributes (or simple components of composite attributes). In addition, relation for weak entity set contains primary key of the owner entity set as foreign key and its primary key is formed by combining partial key (discriminator) and primary key of the owner entity set.

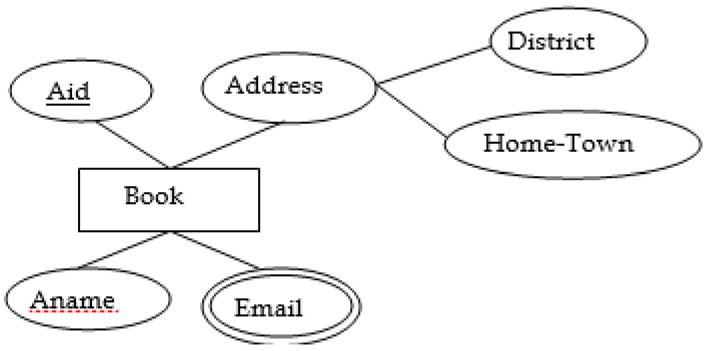
Example: Consider the weak entity set 'Edition' with two attributes edition_no and type, where edition_no is partial key.



ISBN	Edition-no	Type

Mapping of Multivalued and composite attributes

If an entity has composite attributes, no separate attribute (column) is created for composite attribute itself rather attributes (columns) are created for component attributes of the composite attribute. For a multivalued attribute, separate relation is created with primary key of the entity set and multivalued attribute itself.

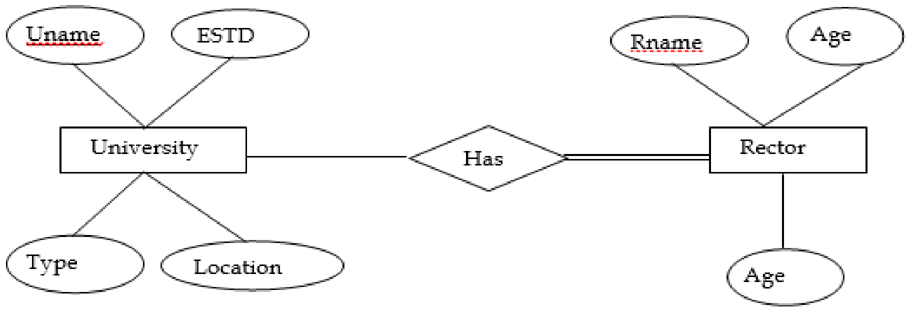


Aid	Aname	District	Home_Town

Aid	Email

Mapping of Binary 1:1 Relation Types

For a binary one to one relationship set separate relation is not created rather primary key of an entity set is included in relation for another entity set as a foreign key. It is better to include foreign key into the entity set with total participation in the relationship.

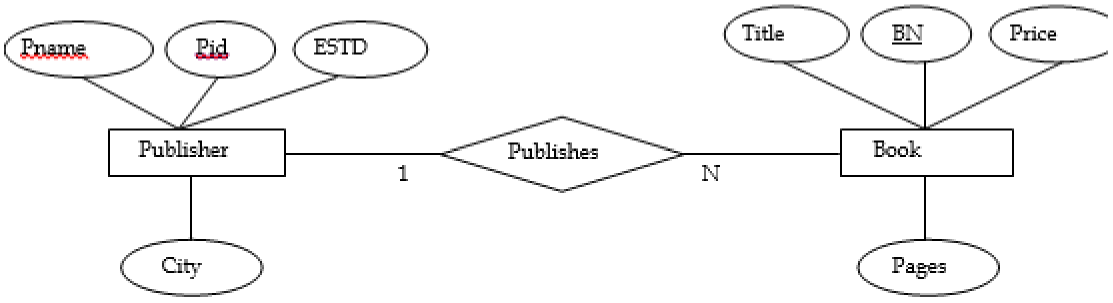


Uname	ESTD	Type	Location

Rname	Uname	Age	Sex

Mapping of Binary 1: N Relation Types

For binary one-to-many relationship identify the relation that represent the participating entity type at the N-side of the relationship type and then include primary key of one side entity set into many side entity set as foreign key. Separate relation is created for the relationship set only when the relationship set has its own attributes

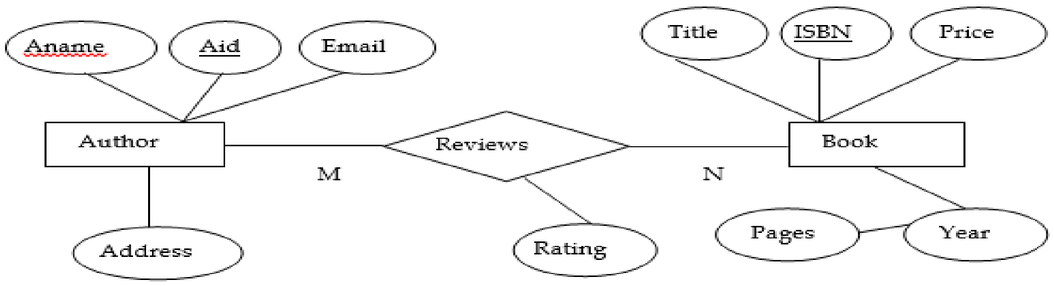


Pid	Pname	ESTD	City

ISBN	Title	Price	Pages	Pid

Mapping of Binary M: N Relation Types

For a binary Many-to-Many relationship type, separate relation is created for the relationship type. Primary key for each participating entity set is included as foreign key in the relation and their combination will form the primary key of the relation. Besides this, simple attributes of the many-to-many relationship type (or simple components of composite attributes) is included as attributes of the relation.



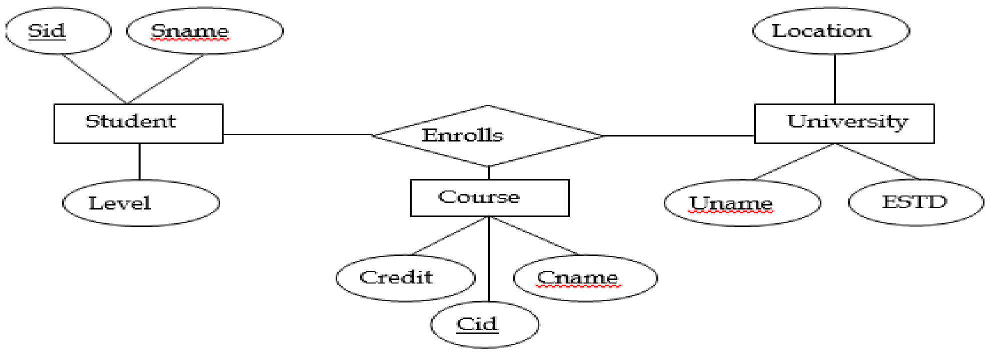
ISBN	Title	Pages	Price	Year

Aid	Aname	Email	Address

ISBN	Aid	Rating

Mapping of N-ary Relationship Types

For each n-ary relationship set for n>2, a new relation is created. Primary keys of all participating entity sets are included in the relation as foreign key attributes. Besides this all simple attributes of the n-ary relationship set (or simple components of composite attributes) are included as attributes of the relation.



Sid	Sname	Level

Uname	ESTD	Location

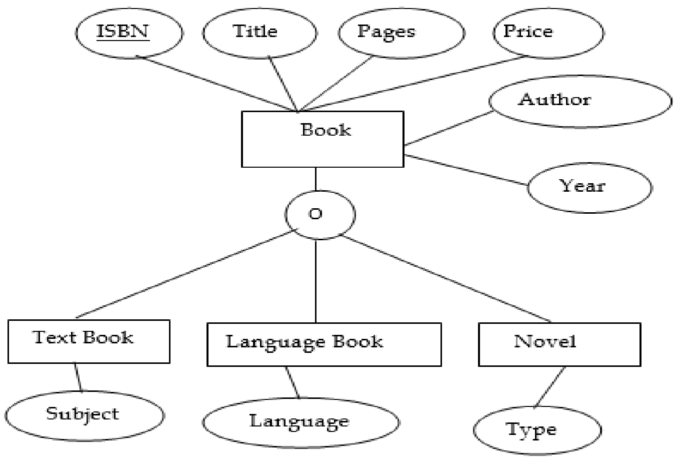
Cid	Cname	Credit

Sid	Uname	Cid

Representing Specialization/Generalization

There are two approaches for translating ER diagrams with specialization or generalization into relations.

- A relation is created for higher level entity set using above discussed methods and then separate relation is created for each of the lower level entity sets. Relation for a subclass entity set includes all of its attributes and key attributes of superclass entity set.
- Another approach is to create relations only for lower level entity sets. Here, relation for a subclass entity set includes all attributes of superclass entity set and all of its own attributes. This approach is possible only when subclasses are disjoint and complete



ISBN	Title	Author	Pages	Price	Year

ISBN	Subject

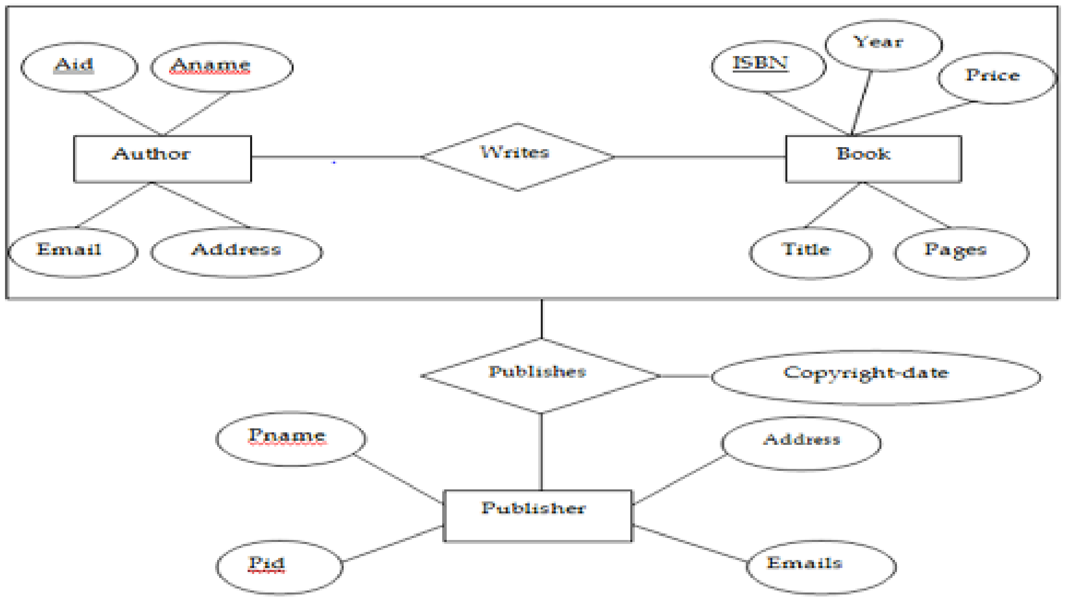
ISBN	Language

ISBN	Type

Representing Aggregation

To represent aggregation, create a table containing

- Primary key of aggregated relationship
- Primary key of the associated entity set
- Any descriptive attributes of the relationship set

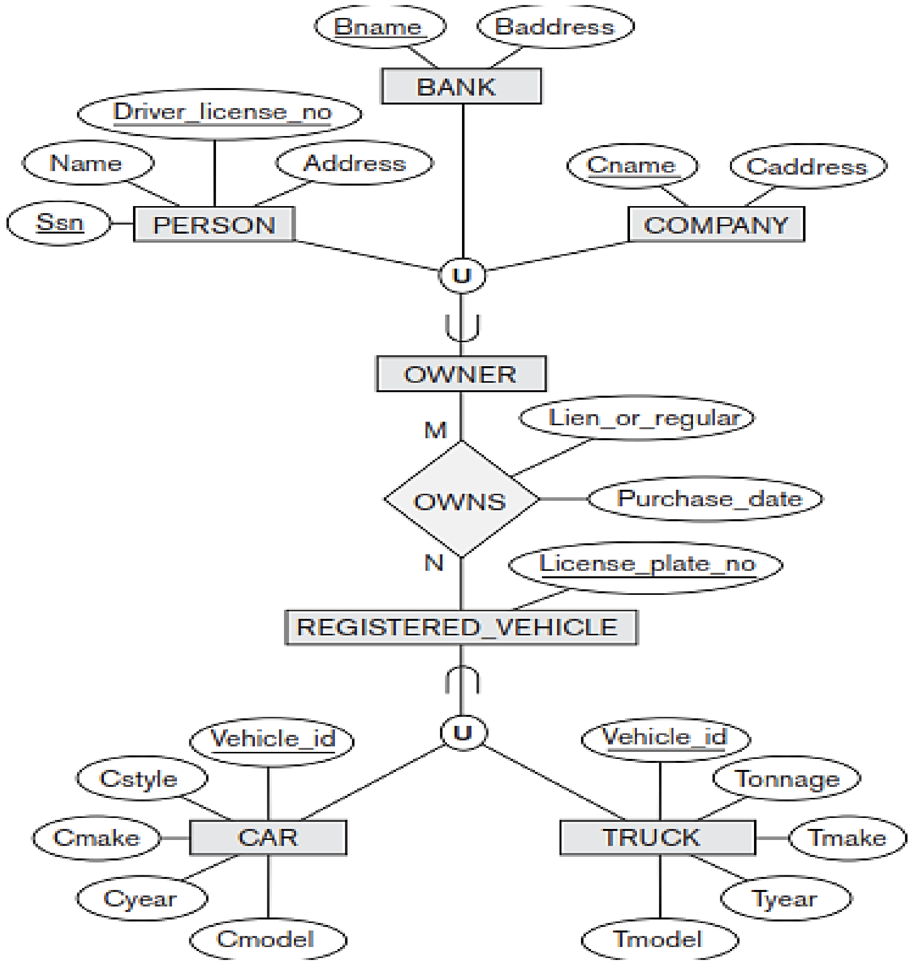


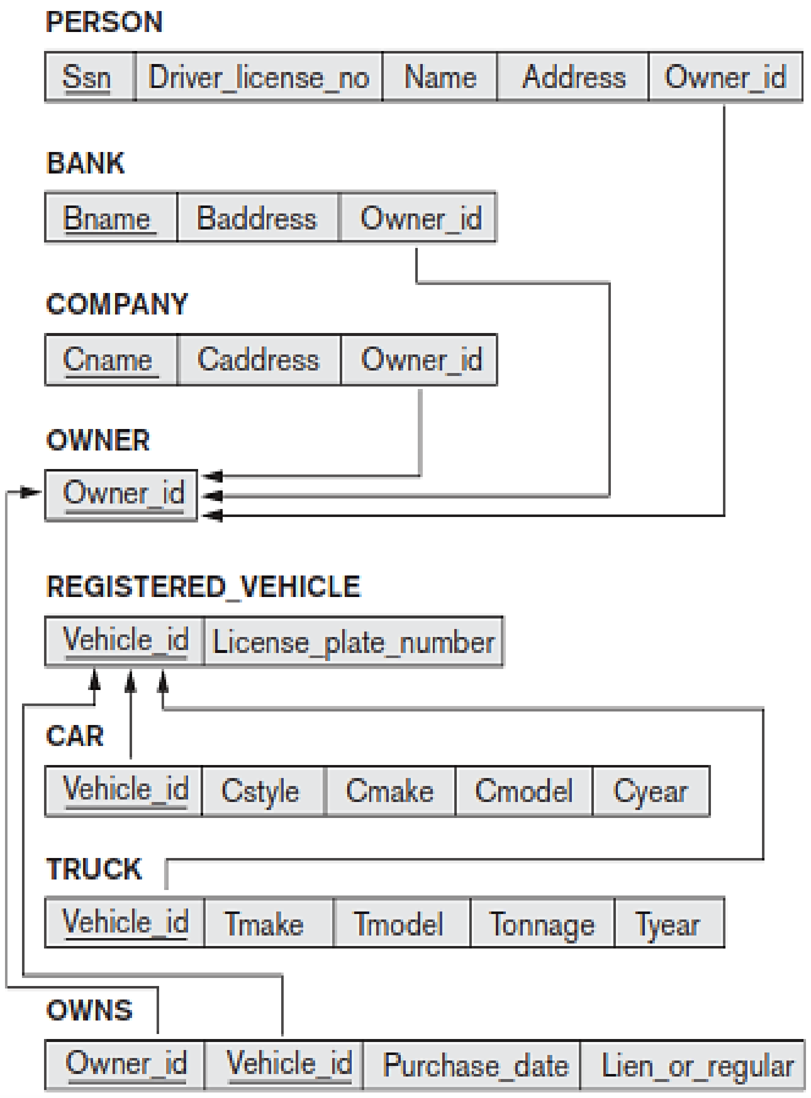
Aid	ISBN	Pid	Copyright_date

Mapping of Union Types (Categories) to relational model

For mapping a category whose defining super-classes have different keys, it is customary to specify a new key attribute, called a **surrogate key**, when creating a relation to correspond to the category. The keys of the defining classes are different, so we cannot use any one of them exclusively to identify all entities in the category. In our example in Figure below, we create a relation OWNER to correspond to the OWNER category, and include any attributes of the category in this relation. The primary key of the OWNER relation is the **surrogate key**, which we called Owner_id. We also include the **surrogate key** attribute Owner_id as foreign key in each relation corresponding to a superclass of the category, to specify the correspondence in values between the **surrogate key** and the key of each superclass. Notice that if a particular PERSON (or BANK or COMPANY) entity is not a member of OWNER, it would have a NULL value for its Owner_id attribute in its corresponding tuple.

Two categories (union types): OWNER and REGISTERED_VEHICLE are shown in fig. below.





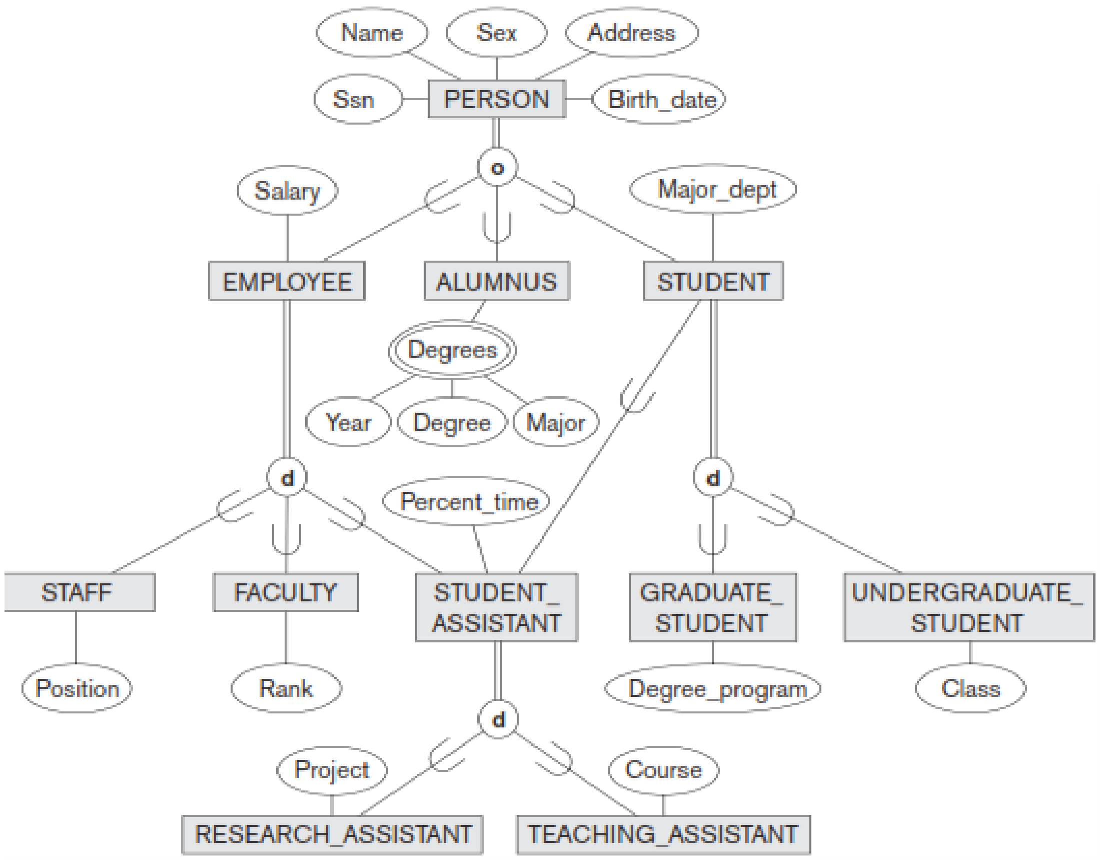
Object Modeling Using UML Class Diagrams

Object modeling methodologies, such as UML (Universal Modeling Language) and OMT (Object Modeling Technique) are becoming increasingly popular. Although these methodologies were developed mainly for software design, a major part of software design involves designing the databases that will be accessed by the software modules. Hence, an important part of these methodologies—namely, the class diagrams are similar to EER diagrams in many ways. Unfortunately, the terminology often differs. We briefly review some of the notation, terminology, and concepts used in UML class diagrams, and compare them with EER terminology and notation. Figure below shows how the COMPANY ER database schema can be displayed using UML notation.

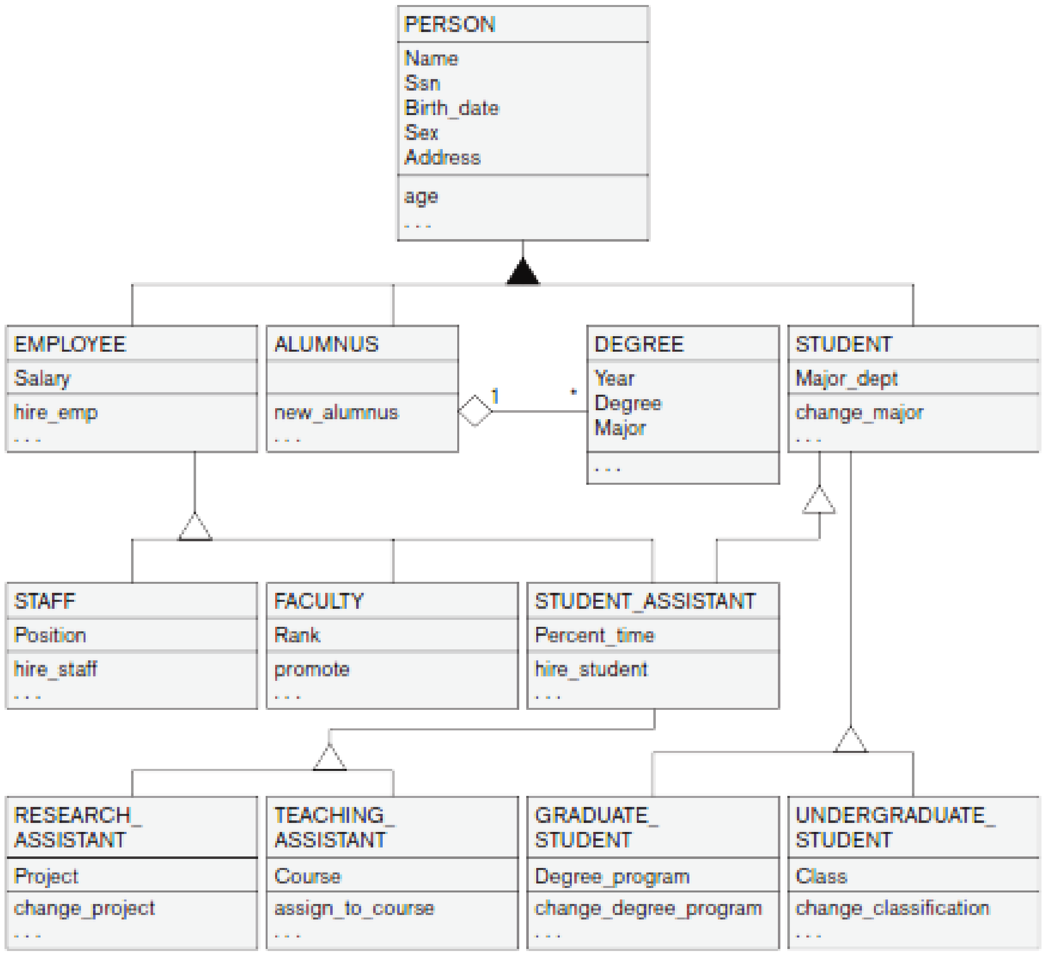
In UML class diagrams, a class is displayed as a box that includes three sections: the top section gives the class name; the middle section includes the attributes for individual objects of the class; and the last section includes operations that can be applied to these objects. Operations are not specified in EER diagrams. Consider the EMPLOYEE class. Its attributes are Name, Ssn, Bdate, Sex, Address, and Salary. The designer can optionally specify the domain of an attribute if desired.

A multivalued attribute will generally be modeled as a separate class, as illustrated by the LOCATION class Relationship types are called associations in UML terminology, and relationship instances are called links. A binary association (binary relationship type) is represented as a line connecting the participating classes (entity types), and may (optional) have a name. A relationship attribute, called a link attribute, is placed in a box that is connected to the association's line by a dashed line.

In UML, there are two types of relationships: **association** and **aggregation**. Figure below illustrates the UML notation for generalization/specialization by giving a possible UML class diagram corresponding to the EER diagram



A UML class diagram corresponding to the EER diagram shown in Figure above is represented by following diagram.



Assignment

1. Construct ER diagram and then translate into a set of **Nepal premier database (NPL)**

Requirements:

- The NPL has many teams,
- Each team has a name, a city, a coach, a captain, and a set of players,
- Each player belongs to only one team,
- Each player has a name, role, a skill level, and a set of injury records
- A team captain is also a player
- A game is played between two teams (referred to as host_team and guest_team) and has a date (such as May 11th, 1999).

2. Construct EER diagram and then map into a set of **University Database**.

Requirements:

- Professors have an Citizenship number, a name, an age, a rank, and a research specialty.

- Projects have a project number, a sponsor name, a starting date, an ending date, and a budget.
- Graduate students have Citizenship number, a name, an age, and a degree program (e.g., M.S. Or Ph.D.).
- Each project is managed by one professor (known as the project's principal investigator).
- Each project is worked on by one or more professors (known as the project's co investigators).
- Professors can manage and/or work on multiple projects.
- Each project is worked on by one or more graduate students (known as the project's research assistants).
- When graduate students work on a project, a professor must supervise their work on the project. Graduate students can work on multiple projects, in which case they will have a (potentially different) supervisor for each one.
- Departments have a department number, a department name, and a main office.
- Departments have a professor (known as the chairman) who runs the department.
- Professors' work in one or more departments, and for each department that they work in, a time percentage is associated with their job.
- Graduate students have one major department in which they are working on their degree.

1. Construct an ER diagram for a car-insurance company whose customers own one or more cars each. Each car has associated with it zero to any number of recorded accidents.
2. Construct an ER diagram for a hospital with a set of patients and a set of doctors. Associate with each patient a log of the various tests and examinations conducted.
3. Construct an ER diagram of the library system in your college.
4. Construct an ER diagram to maintain data about students, instructors, semester, and courses in a college.
5. Construct an ERD to record the marks that students get in different exams of different course offerings.

Unit 3
Emerging Database Management System Technologies
By
Bhupendra Singh Saud

Overview of Object Oriented Concepts

Object oriented databases try to maintain a direct correspondence between real-world and database objects so that the objects do not lose their integrity and identity and can easily be identified and operated upon. Object has two compounds and they are:

- State (value) and
- Behavior (operations)

In an object-oriented database, objects may have an object structure of arbitrary complexity in order to contain all of the necessary information that is required describes the object. On the contrary in traditional database systems, information about a complex object is often scattered over many relations or records leading to loss of direct correspondence between a real-world object and its database representation. The internal structure of an object in OOPLs includes the specification of instance variables which holds the value that defines the internal state of the object. An instance variable is similar to the concept of an attribute except for the instance variables may be encapsulated within the object and thus are not necessarily visible to external users. Some OO models insist that all operations a user can apply to an object must be predefined. This forces to a complete encapsulation of objects.

Why we need object oriented database?

We need object oriented database due to mainly following two reasons:

- Application which required modeling of complex objects:
  Traditional data models and systems have been quite successful in developing the database technologies required for many traditional business database applications. However, they have certain shortcomings when more complex database applications must be designed and implemented—for example, databases for engineering design and manufacturing (CAD/CAM and CIM1), scientific experiments, telecommunications, geographic information systems, and multimedia. These newer applications have requirements and characteristics that differ from

those of traditional business applications, such as more complex structures for stored objects; the need for new data types for storing images, videos, or large textual items; longer-duration transactions; and the need to define nonstandard application-specific operations.

- Vast increase in the use of object-oriented programming languages for developing software applications.
- To maintain a direct correspondence between real-world and database objects

Classes and objects

A class is understood as an entity that has a well-defined role in the application domain. An object is referred to as a particular instance of a class. An object has structure or state (variables) and methods (behavior/operations). An object is described by four characters and they are:

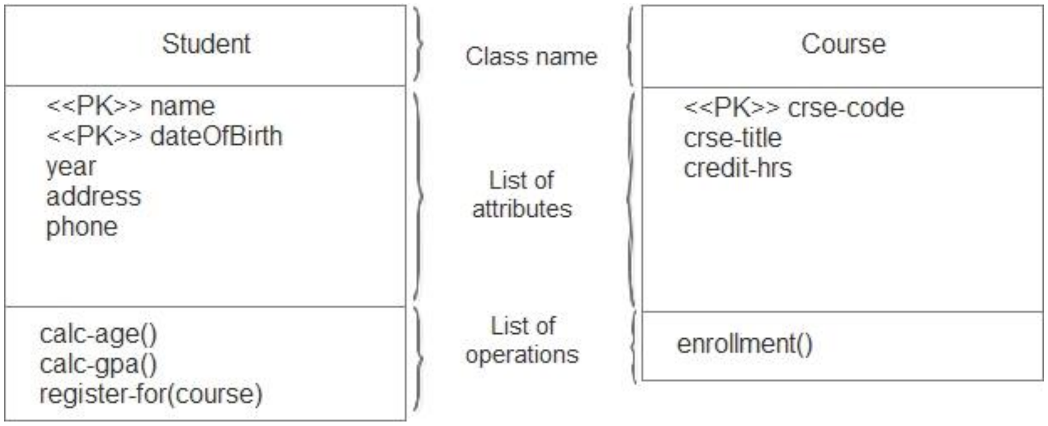
- **Identifier:** A system-wide unique id for an object.
- **Name**: An object may also have a unique name in database which is optional.
- **Lifetime:** It determines if the object is persistent or transient.
- **Structure:** The construction of objects using type constructors.

Object Oriented Database Model

This model represents an entity as a class. A class captures the attributes as well as the behavior. Instances of the class are called objects. Within an object, the class attributes take on specific values which distinguish one object from another. Object Oriented Examples:

- **Class:** Cat.
- **Attributes**: Color, weight, breed.
- **Behavior:** Scratches, sleeps, purrs.

An instance of the cat class is an object with specific attributes.



[TU Question]:- What are the advantages and disadvantages of OODBMS?

Advantages of OODBMS

- Easier Design-Reflect Applications
- Modularity and Reusability
- Incremental refinement and abstraction
- Multiple inheritance
- Support for multiple version and Alternatives
- Designer can specify the structure of objects and their behavior (methods).
- Better interaction with object-oriented languages such as Java and C++
- Definition of complex and user-defined types.
- Encapsulation of operations and user-defined methods.

Disadvantages of OODBMSs

There are following disadvantages of OODBMSs:

- **Lack of universal data model:** There is no universally agreed data model for an OODBMS, and most models lack a theoretical foundation. This .disadvantage is seen as a significant drawback, and is comparable to pre-relational systems.
- **Lack of experience:** In comparison to RDBMSs the use of OODBMS is still relatively limited. This means that we do not yet have the level of experience that we have with traditional systems. OODBMSs are still very much geared towards the programmer, rather than the naïve end-user. Also there is a resistance to the acceptance of the technology. While the OODBMS is limited to a small niche market, this problem will continue to exist
- **Lack of standards:** There is a general lack of standards of OODBMSs. We have already mentioned that there is not universally agreed data model. Similarly, there is no standard object-oriented query language.
- **Competition:** Perhaps one of the most significant issues that face OODBMS vendors is the competition posed by the RDBMS and the emerging ORDBMS products. These products have an established user base with significant experience available. SQL is an approved standard and the relational data model has a solid theoretical formation and relational products have many supporting tools to help .both end-users and developers.
- **Query optimization compromises encapsulations:** Query optimization requires. An understanding of the underlying implementation to access the database efficiently. However, this compromises the concept of incrassation.

- **Locking at object level may impact performance** Many OODBMSs use locking as the basis for concurrency control protocol. However, if locking is applied at the object level, locking of an inheritance hierarchy may be problematic, as well as impacting performance.
- **Complexity:** The increased functionality provided by the OODBMS (such as the illusion of a single-level storage model, pointer sizzling, version management, and schema evolution--makes the system more complex than that of traditional DBMSs. In complexity leads to products that are more expensive and more difficult to use.
- **Lack of support for views:** Currently, most OODBMSs do not provide a view mechanism, which, as we have seen previously, provides many advantages such as data independence, security, reduced complexity, and customization.
- **Lack of support for security:** Currently, OODBMSs do not provide adequate security mechanisms. The user cannot grant access rights on individual objects or classes.

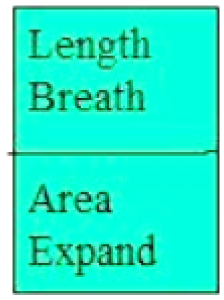
Features of object oriented (O-O) Databases

- Object oriented databases store persistent objects permanently on secondary storage, and allow the sharing of these objects among multiple programs and applications.
- Object oriented databases provide a unique system-generated object identifier for each object. Object oriented databases maintain a direct correspondence between real-world and database objects so that objects don't lose their integrity and identify and can be easily be identified and operated upon.
- In Object Oriented databases, objects can be very complex in order to contain all significant information that may be required to describe the object completely.
- Object oriented databases allow us to store both the class and state of an object between programs. They take the responsibility for maintaining the links between stored object behavior and state away from the programmer, and manage objects outside of programs with their public and private elements intact. They also simplify the whole process of rendering objects persistent by performing such tasks invisibly.

State and Behavior

Object normally have property and some behavior. State of an object is the values of attributes and behavior of object means operations that operates on attributes of given object.

Example: for the rectangle object,



Rectangle

Length, breadth acts as states of rectangle and calculating area of rectangle act as behavior of rectangle object.

Type Constructors

Type constructor is the collection of multiple similar basic type under a common name. It determines how the object is constructed. The type constructors can be used to define the data structures for an object oriented database schema. The three most basic constructors are **atom, tuple,** and **set**. Other commonly used constructors include **list, bag,** and **array.**
The type constructors set, list, array, and bag are called collection types (or bulk types), and to distinguish them from basic types and tuple types. Here, the state of the object will be a collection of objects that may be unordered or ordered.

Kinds of Type constructor:

- **Atom:** says that an object is storing atomic values.
  e.g.: "Aarav".
- **Set:** set of values of same type with duplication allowed.
  e.g.: {123,456,123}.
- **Bag:** set with no duplicate items.
  e.g.: {123,456,678}
- **List:** ordered collection of items of the same type with infinite size.
  e.g.: [123,456,678]
- **Array:** similar to list but fixed size.

[TU Question]:- What is OID? How persistent objects are maintained in OO Database?
[TU Question]:- What is the difference between persistent and transient objects? How persistence is handled in typical OO database systems?

OIDs (Object Identifiers)

When a program terminates then all the objects that is in the primary memory are lost. These kind of objects called **transient object**. They do not live forever.

However, OO allows us to store objects permanently in the database, these objects are called **persistent objects**. Because they persist the beyond the life of program.

OIDs is the mechanism to refer to persistent objects. An ODMS provides a unique identity to each independent object stored in the database. This unique identity is typically implemented via a unique, system-generated object identifier (OID). The value of an OID is not visible to the external user, but is used internally by the system to identify each object uniquely and to create and manage inter-object references.

The main property required of an OID is that it be immutable; that is, the OID value of a particular object should not change. This preserves the identity of the real-world object being 4rdrepresented.

Example: In Figure below, the attributes that refer to other objects—such as **Dept** of EMPLOYEE or **Projects** of DEPARTMENT are basically OIDs that serve as references to other objects to represent relationships among the objects. For example, the attribute **Dept** of EMPLOYEE is of type DEPARTMENT, and hence is used to refer to a specific DEPARTMENT object (the DEPARTMENT object where the employee works). The value of such an attribute would be an OID for a specific DEPARTMENT object.

```
Define type EMPLOYEE
    Tuple (Fname: string;
    Minit: char;
    Lname: string;
    Ssn: string;
    Birth_date: DATE;
    Address: string;
    Sex: char;
    Salary: float;
    Supervisor: EMPLOYEE;
    Dept: DEPARTMENT ;);

Define type DATE
    Tuple (Year: integer;
    Month: integer;
    Day: integer ;);

Define type DEPARTMENT
    Tuple (Dname: string;
    Dnumber: integer;
```

```
                    Mgr: tuple (Manager: EMPLOYEE;
                    Locations: set (string);
                    Start_date: DATE ;);
                    Employees: set (EMPLOYEE);
                    Projects: set (PROJECT) ;);
```

Fig: Specifying the object types EMPLOYEE, DATE, and DEPARTMENT using type constructors.

Similarities and dissimilarities of Objects and Literals

Objects and literals are the basic building blocks of the object model. The main difference between the two is that an object has both an object identifier and a state (or current value), whereas a literal has a value (state) but no object identifier. In either case, the value can have a complex structure. The object state can change over time by modifying the object value. A literal is basically a constant value, possibly having a complex structure, but it does not change.

Every object must have an immutable OID, whereas a literal value has no OID and its value just stands for itself. Thus, a literal value is typically stored within an object and cannot be referenced from other objects. An object has five aspects: **identifier, name, lifetime, structure,** and **creation**.

- The object identifier is a unique system-wide identifier (or Object_id). Every object must have an object identifier.
- Some objects may optionally be given a unique name within a particular ODMS—this name can be used to locate the object, and the system should return the object given that name.
- The lifetime of an object specifies whether it is a persistent object (that is, a database object) or transient object (that is, an object in an executing program that disappears after the program terminates). Lifetimes are independent of types—that is, some objects of a particular type may be transient whereas others may be persistent.
- The structure of an object specifies how the object is constructed by using the type constructors. The structure specifies whether an object is atomic or not. An atomic object refers to a single object that follows a user-defined type, such as Employee or Department. If an object is not atomic, then it will be composed of other objects.
- Object creation refers to the manner in which an object can be created. This is typically accomplished via an operation new for a special Object_Factory interface.

In the object model, a literal is a value that does not have an object identifier. However, the value may have a simple or complex structure. There are three types of literals: **atomic, structured,** and **collection.**

- **Atomic literals** correspond to the values of basic data types and are predefined. The basic data types of the object model include long, short, and unsigned integer numbers, regular and double precision floating point numbers, Boolean values, single characters, character strings, and enumeration types.

- **Structured literals** correspond roughly to values that are constructed using the tuple constructor. The built-in structured literals include Date, Interval, Time, and Timestamp. User-defined structures are created using the STRUCT keyword in ODL, as in the C and C++ programming languages.
- **Collection literals** specify a literal value that is a collection of objects or values but the collection itself does not have an Object_id. The collections in the object model can be defined by the type generators set<T>, bag<T>, list<T>, and array<T>, where T is the type of objects or values in the collection.

[TU Question]:- What is the difference between structured and unstructured complex object? Differentiate identical versus equal objects with examples.

Complex objects (Object structure)

It means there is no restriction in the structure in the object. In ODBs, the value of a complex object can be constructed from other objects. Each object is represented by triple. Complex objects are built from simpler ones by applying constructors to them. The simplest objects are objects such as integers, characters, byte strings of any length, Booleans and floats (one might add other atomic types). There are various complex object constructors: tuples, sets, bags, lists, and arrays are examples.

An object is defined by a triple (OID, type constructor, state) or (i, c, v) where OID is the unique object identifier, type constructor is its type (such as atom, tuple, set, list, array, bag, etc.) and state is its actual value.

Example:

- (i1, atom, 'John')
- (i2, atom, 30)
- (i3, atom, 'Mary')
- (i4, atom, 'Mark')
- (i5, atom 'Vicki')
- (i6, tuple, [Name: i1, Age: i3])
- (i7, set, {i4, i5})
- (i8, tuple, [Name: i3, Friends: i7])
- (i9, set, {i6, i8})

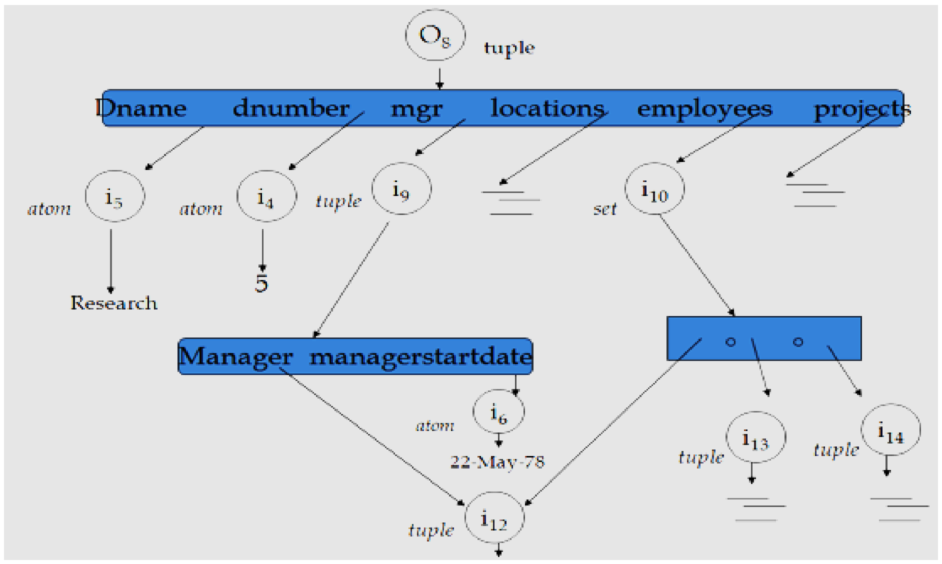


Fig: Structure of Complex object

There are two types of complex objects in object oriented database system which are:

1. Structured complex object and
2. Unstructured complex object

1. Structured Complex object

Structured complex object is defined by repeated application of the type constructors provided by the OODBMS. Simply structured complex objects are constructed by using type constructors (set, atom, tuple etc.). Hence, the object structure is defined and known to the OODBMS. The OODBMS also defines methods or operations on it.

Two types of reference semantics (**ownership semantics** and **reference semantics**) exist between a complex object and its components at each level.

- **Ownership semantics** applies when the sub-objects of a complex object are encapsulated within the complex object and are hence considered part of the complex object. It is also called is-part-of or is-component-of relationship. e.g., "Arjun" atomic value owned by employee. Means that 'Arjun' is dependent on owner.
- **Reference semantics** applies when the components of the complex object are themselves independent objects but may be referenced from the complex object. It is also called is-associated-with relationship. e.g., Department referenced by the employee object.

2. Unstructured complex objects

It is a data type provided by a DBMS and permits the storage and retrieval of large objects that are needed by the database application. These objects are unstructured in the sense that the DBMS does not know what their structure is, only the application programs that uses them can interpret

their meaning. These objects are considered complex because they require large area of storage and are not part of the standard data types provided by traditional DBMSs.
Typical examples of such objects are **bitmap images** and **long text strings** (such as documents); they are also known as binary large objects, or BLOBs for short. Character strings are also known as character large objects, or CLOBs for short.

Identical vs. Equal objects

Two objects are said to have **identical states** (deep equality) if the graphs representing their states are identical in every respect, including the OIDs at every level.

Two objects are said to have **equal states** (shallow equality) if the graphs representing their states are same, including all the corresponding atomic values. However, some corresponding internal nodes in the two graphs may have objects with different OIDs.

Example: This example illustrates the difference between the two definitions for comparing object states for equality.

$o_1 = (i_1, \text{tuple}, <a_1:i_4, a_2:i_6>)$

$o_2 = (i_2, \text{tuple}, <a_1:i_5, a_2:i_6>)$

$o_3 = (i_3, \text{tuple}, <a_1:i_4, a_2:i_6>)$

$o_4 = (i_4, \text{atom}, 10)$

$o_5 = (i_5, \text{atom}, 10)$

$o_6 = (i_6, \text{atom}, 20)$

In this example, the objects o_1 and o_2 have **equal states** (shallow equality), since their states at the atomic level are the same but the values are reached through distinct objects o_4 and o_5.

However, the objects o_1 and o_3 have **identical states** (deep equality), even though the objects themselves are not because they have distinct OIDs. Similarly, although the states of o_4 and o_5 are identical, the actual objects o_4 and o_5 are equal but not identical, because they have distinct OIDs.

Encapsulation of Operations, Methods, and Persistence

Encapsulation means that an object contains both the data structure and the set of operations that can be used to manipulate it. Often cases, adopting encapsulation hides the implementation from the users do not necessarily have to know the detail of it.

Encapsulation is related to the concepts of abstract data types and information hiding in programming languages. Here the main idea is to define the behavior of a type of object based on the operations that can be externally applied to objects of that type. The internal structure of the object is hidden, and the object is only accessible through a number of predefined operations. Some operations may be used to create or destroy objects; other operations may update the object value and other may be used to retrieve parts of the object value or to apply some calculations to the object value.

The external users of the object are only made aware of the interface of the object, which defines the names and arguments of each operation. The implementation of the object is hidden from the external users; it includes the definition of the internal data structure of the object and the implementation of the operations that access these structures.

In object oriented - OO terminology, the interface part of each operation is called the signature, and the operation implementation is called a method. A method is invoked by sending a message to the object to execute the corresponding method.

Not all objects are meant to be stored permanently in the database. Transient objects exist in the executing program and disappear once the program terminates.

Persistent objects are stored in the database and persist after program terminates. The typical mechanism for persistence involves giving an object a unique persistent name through which it can be retrieved.

In traditional database models and systems, this concept was not applied, since it is usual to make the structure of database objects visible to users and external programs.

Inheritance

Inheritance is deriving objects from existing objects. The derived objects inherit properties from their parent object. Parent objects are those objects from which other objects are derived. Inheritance is a way of reusing the existing code.

Polymorphism

Polymorphism concept allows the same operator name or symbol to be bound to two or more different implementation of the operator, depending on the type of objects to which the operator is applied.

Multiple Inheritance and Selective Inheritance

Multiple inheritance in a type hierarchy occurs when a certain subtype T is a subtype of two (or more) types and hence inherits the functions (attributes and methods) of both super types.
For example, we may create a subtype ENGINEERING_MANAGER that is a subtype of both MANAGER and ENGINEER. This leads to the creation of a type lattice rather than a type hierarchy.

One problem that can occur with multiple inheritance is that the super types from which the subtype inherits may have different functions of the same name, creating an ambiguity.

If a function is inherited from some common super type, then it is inherited only once. In such a case, there is no ambiguity; the problem only arises if the functions are distinct in the two super types. Some languages do not allow multiple inheritance.

Selective inheritance occurs when a subtype inherits only some of the functions of a super type. Other functions are not inherited. This mechanism is not typically provide in OO database system.

Versions and configurations

Versions:

- Ability to maintain several versions of an object
- Commonly found in many software engineering and concurrent engineering environments
- Merging and reconciliation of various versions is left to the application program
- Some systems maintain a version graph

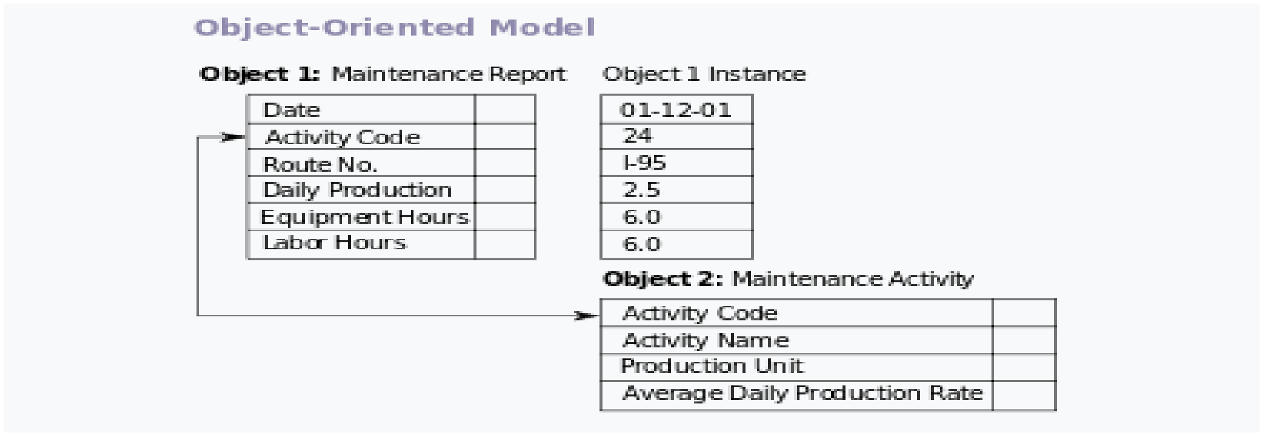
Configuration:

- A configuration is a collection compatible versions of modules of a software system (a version per module)

Object Relational Database concepts

Object relational DBMSs (ORDBMSs) are also called object-relational or enhanced systems. These systems emerged as a way of enhancing the capabilities of relational DBMSs (RDBMSs) with some of the features that appeared in object oriented DBMSs (ODBMSs). Object-relational database systems provide a convenient migration path for users of relational database who wish to use object-oriented features

An object-relational database (ORD), or object-relational database management system (ORDBMS), is a database management system (DBMS) similar to a relational database, but with an object-oriented database model: objects, classes and inheritance are directly supported in database schemas and in the query language. In addition, just as with pure relational systems, it supports extension of the data model with custom data-types and methods.



Example of an object-oriented database model

An object-relational database can be said to provide a middle ground between relational databases and object-oriented databases (object database).

Comparison of RDBs vs. ORDBs

- Very easy to compare because both are based on relational model.
- An RDB does not support abstract data types (ADT), all attribute values must be atomic and relations must be in first normal form (flat relation).
- An ORDB supports ADTs, attributes can be multivalued, and does not require first normal form.
- The underlying basic data structures of RDBs are much simpler but less versatile than ORDBs.
- ORDBs support complex data whereas RDBs don't.
- ORDBs support wide range of applications.
- RDBs have only one construct i.e. Relation, whereas the type system of ODBs is much richer and complex.
- RDBs require primary keys and foreign keys for implementing relationships, ODBs simply don't.
- ODBs support complex data whereas RDBs don't.
- ODBs support wide range of applications.
- ODBs are much faster than RDBs but are less mature to handle large volumes of data.
- There is more acceptance and domination of RDBs in the market than that for ODBs.
- Both support ADTs, collections, OIDs, and inheritance, though philosophically quite different.
- ORDBs extended RDBs whereas ODBs add persistence and database capabilities to OO languages.
- Both support query languages for manipulating collections and nested and complex data.
- SQL3 is inspired from OO concepts and is converging towards OQL (object query language).
- ORDBs carries all the benefits of RDBs, whereas ODBs are less benefited from the technology of RDBs.

Overview of SQL and its object-relational features

SQL stands for structured query language. It is a database query language used to communicate and manage data stored on the database. SQL was specified in 1970s. It was enhanced substantially

in 1989 and 1992. A new standard called SQL3 added object-oriented features. A subset of SQL3 standard, now known as SQL-99 has been approved.
The following are some of the object database features that have been included in SQL:

- Some type constructors have been added to specify complex objects. These include the row type, which corresponds to the tuple (or struct) constructor. An array type for specifying collections is also provided. Other collection type constructors, such as set, list, and bag constructors, were not part of the original SQL/Object specifications but were later included in the standard.
- A mechanism for specifying object identity through the use of reference type is included.
- Encapsulation of operations is provided through the mechanism of user defined types (UDTs) that may include operations as part of their declaration. These are somewhat similar to the concept of abstract data types that were developed in programming languages. In addition, the concept of user defined routines (UDRs) allows the definition of general methods (operations).
- Inheritance mechanisms are provided using the keyword UNDER.

[TU Model Question):- Describe the steps of the algorithm for object database design by EER-to-OO mapping.

Mapping an EER Schema to an ODB Schema

It is relatively straightforward to design the type declarations of object classes for an ODBMS from an EER schema that contains neither categories nor n-ary relationships with $n > 2$. However, the operations of classes are not specified in the EER diagram and must be added to the class declarations after the structural mapping is completed.
The outline of the mapping from EER to ODL is as follows:
Step 1. Create an ODL class for each EER entity type or subclass. The type of the ODL class should include all the attributes of the EER class. Multivalued attributes are typically declared by using the **set, bag, or list** constructors. If the values of the multivalued attribute for an object should be ordered, the list constructor is chosen; if duplicates are allowed, the bag constructor should be chosen; otherwise, the set constructor is chosen. Composite attributes are mapped into a tuple constructor (by using a struct declaration in ODL). Declare an extent for each class, and specify any key attributes as keys of the extent.

Step 2. Add relationship properties or reference attributes for each binary relationship into the ODL classes that participate in the relationship. These may be created in one or both directions. If a binary relationship is represented by references in both directions, declare the references to be relationship properties that are inverses of one another, if such a facility exists. If a binary relationship is represented by a reference in only one direction, declare the reference to be an attribute in the referencing class whose type is the referenced class name. Depending on the cardinality ratio of the binary relationship, the relationship properties or reference attributes may be single-valued or collection types. They will be single-valued for binary relationships in the 1:1 or N:1 directions; they are collection types (set-valued or list-valued) for relationships in the 1:N or M:N direction. If relationship attributes exist, a tuple constructor can be used to create a structure of the form < reference, relationship attributes>, which may be included instead of the reference attribute.

Step 3. A constructor method should include program code that checks any constraints that must hold when a new object is created. A destructor method should check any constraints that may be violated when an object is deleted. Other methods should include any further constraint checks that are relevant.

Step 4. An ODL class that corresponds to a subclass in the EER schema inherits (via extends) the type and methods of its superclass in the ODL schema.

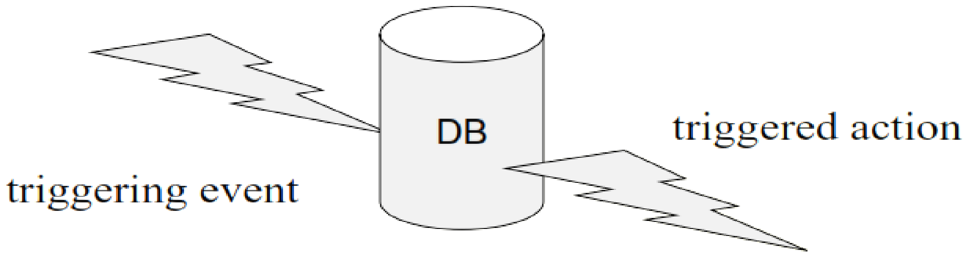
Step 5. Weak entity types can be mapped in the same way as regular entity types. An alternative mapping is possible for weak entity types that do not participate in any relationships except their identifying relationship; these can be mapped as though they were composite multivalued attributes of the owner entity type, by using the set < struct <... >> or list <struct <... >> constructors. The attributes of the weak entity are included in the struct <... > construct, which corresponds to a tuple constructor.

Step 6. Categories (union types) in an EER schema are difficult to map to ODL. It is possible to create a mapping similar to the EER-to-relational mapping by declaring a class to represent the category and defining 1:1 relationships between the category and each of its super classes. Another option is to use a union type, if it is available

Step 7. An n-ary relationship with degree n > 2 can be mapped into a separate class, with appropriate references to each participating class. These references are based on mapping a 1:N relationship from each class that represents a participating entity type to the class that represents the n-ary relationship. An M:N binary relationship especially if it contains relationship attributes, may also use this mapping option, if desired.

Active database concepts

A **trigger** is a procedure that is automatically invoked by the DBMS in response to specified changes to the database, and is typically specified by the DBA. A database that has a set of associated triggers is called an **active database**.



Generalized Model for Active Databases and Oracle Triggers

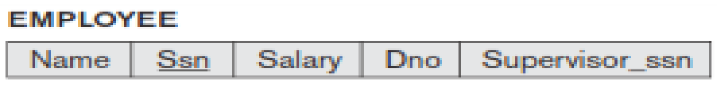
The model that has been used to specify active database rules is referred to as the Event-Condition-Action (ECA) model. A rule in the ECA model has three components:

1. **The event(s) that triggers the rule:**
   These events are usually database update operations that are explicitly applied to the database. However, in the general model, they could also be temporal events or other kinds of external events.
2. **The condition that determines whether the rule action should be executed**:
   Once the triggering event has occurred, an optional condition may be evaluated. If no condition is specified, the action will be executed once the event occurs. If a condition is specified, it is first evaluated, and only if it evaluates to true will the rule action be executed.
3. **The action to be taken:**
   The action is usually a sequence of SQL statements, but it could also be a database transaction or an external program that will be automatically executed.

A syntax summary for specifying triggers in the Oracle system

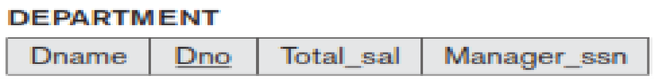
```
CREATE TRIGGER <trigger name>
    AFTER / BEFORE <triggering events>
        ON <table name>
            [FOR EACH ROW]
                [WHEN <condition>]
                    <Trigger actions>;
```

Example: For the following schema diagram;



EMPLOYEE

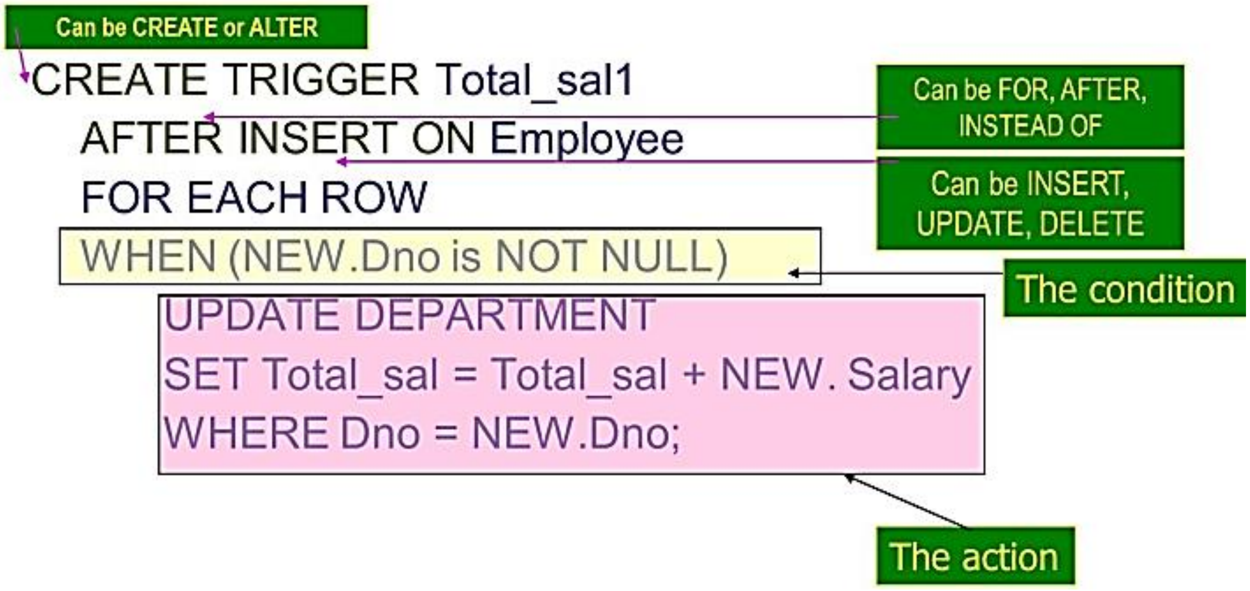
Name	Ssn	Salary	Dno	Supervisor_ssn



DEPARTMENT

Dname	Dno	Total_sal	Manager_ssn

Suppose, we need to compute Total_sal if the new employee is immediately assigned to a department—that is, if the value of the Dno attribute for the new employee tuple is not NULL



```
CREATE TRIGGER Total_sal1
   AFTER INSERT ON Employee
   FOR EACH ROW
   WHEN (NEW.Dno is NOT NULL)
      UPDATE DEPARTMENT
      SET Total_sal = Total_sal + NEW. Salary
      WHERE Dno = NEW.Dno;
```

Design and Implement Issues for Active Databases

In addition to creating rules, an active database allows users to **activate**, **deactivate**, **drop,** and **group** rules by refereeing their rules

- ❖ A **deactivated rule** will not be triggered by the triggering event. This feature allows users to selectively deactivate rules for certain periods of time when they are not needed
- ❖ The **active command** will make the rule active again
- ❖ The drop command deletes the rule from the system
- ❖ Another option is to group rules into named **rule sets**, so the whole set of rules could be activated, deactivated, or dropped. It is also useful to have a command that can trigger a rule or rule set via an explicit **PROCESS RULE** command issued by the user.
- ❖ Whether the triggered action should be executed **before**, **after**, or **concurrently** with the triggering event. Whether the action being executed should be considered as a **separate transaction** or whether it should be part of the same transaction that triggered the rule

There are three main possibilities for *condition evaluation* (also known as *rule consideration*)

- ❖ **Immediate consideration:** The condition is evaluated as part of the same transaction as the triggering event, and is evaluated immediately. This case can be further categorized into three options:
  - ■ Evaluate the condition before executing the triggering event
  - ■ Evaluate the condition after executing the triggering event
  - ■ Evaluate the condition instead of executing the triggering event
- ❖ **Deferred consideration**. The condition is evaluated at the end of the transaction that included the triggering event. In this case, there could be many triggered rules waiting to have their conditions evaluated.
- ❖ **Detached consideration.** The condition is evaluated as a separate transaction, spawned from the triggering transaction.

Potential Applications for Active Databases

- ❖ **Notification** – allow notification of certain conditions that occur
- ❖ **Enforce integrity constraint** – specify the types of events that may cause the constraints to be violated and then evaluating appropriate condition that check whether the constraints are actually violated by the event or not
- ❖ Automatic **maintenance of derived data**
- ❖ Maintain the consistency of **materialized views** whenever the base relations are modified
- ❖ Maintain **replicated tables** whenever the master table is modified

Temporal database concepts

Temporal databases, in the broadest sense, encompass all database applications that require some aspect of time when organizing their information. It encompass all DB applications that require some aspect of time when organizing their information. A **temporal** database is a database that has certain features that support time-sensitive status for entries. Where some databases are considered current databases and only support factual data considered valid at the time of use, a temporal database can establish at what times certain entries are accurate.

Temporal DB applications have been developed since the early days of database usage. However, in creating these applications, it was mainly left to the application developers to discover, design, program, and implement the temporal concepts.

There are many examples of applications where some aspect of time is needed to maintain the information in a DB.

- ***Health care*:** patient histories need to be maintained
- ***Insurance*:** claims and accident histories are required

- ***Finance*****:** stock price histories need to be maintained.
- ***Personnel management*****:** salary and position history need to be maintained
- ***Banking*****:** credit histories

Valid Time and Transaction Time Dimensions

Valid time

Given a particular event or fact that is associated with a particular time point or time period in the database, the association may be interpreted to mean different things. The most natural interpretation is that the associated time is the time that the event occurred, or the period during which the fact was considered to be true in the real world. If this interpretation is used, the associated time is often referred to as the **valid time**.

Simply, the **valid time** denotes the time period during which an event occur or a fact is true with respect to the real world. A temporal database using this interpretation is called a **valid time** database.

Example:

Imagine that we come up with a temporal database storing data about the 18th century. The valid time of these facts is somewhere between 1700 and 1799, whereas the transaction time starts when we insert the facts into the database, for example, January 21, 1998. Assume we would like to store data about our employees with respect to the real world. Then, the following table could result:

EmpID	Name	Department	Salary	ValidTimeStart	ValidTimeEnd
10	John	Research	11000	1985	1990
10	John	Sales	11000	1990	1993
10	John	Sales	12000	1993	Now
11	Paul	Research	10000	1988	1995
12	George	Research	10500	1991	Now
13	Ringo	Sales	15500	1988	Now

The above valid-time table stores the history of the employees with respect to the real world. The attributes **ValidTimeStart** and **ValidTimeEnd** actually represent a time interval which is closed at its lower and open at its upper bound. Thus, we see that during the time period [1985 – 1990], employee John was working in the research department, having a salary of 11000. Then he changed to the sales department, still earning 11000. In 1993, he got a salary raise to 12000. Note that it is now possible to store information about past states. We see that Paul was employed from 1988 until 1995. In the corresponding non-temporal table, this information was (physically) deleted when Paul left the company.

Transaction Time

However, a different interpretation can be used, where the associated time refers to the time when the information was actually stored in the database; that is, it is the value of the system time clock when the information is valid in the system. In this case, the associated time is called the **transaction time**. A temporal database using this interpretation is called a **transaction time database.**

Simply, the **Transaction time** is the time period during which a fact is stored in the database.

Example:

EmpID	Name	Department	Salary	TransactionTimeStart	TransactionTimeEnd
10	John	Research	11000	1985-10-2, 10:02:33	1990-10-2, 11:33:04
10	John	Sales	11000	1990-04-11, 05:04:33	1993-05-11, 06:22:55
10	John	Sales	12000	1993-7-30, 5:33:05	Now
11	Paul	Research	10000	1988-5-23, 7:23:34	1995-8-30, 5:33:55
12	George	Research	10500	1991-8-23, 12:44:34	Now
13	Ringo	Sales	15500	1988-7-24, 11:23:55	Now

The above valid-time table stores the history of the employees with respect to the real world. The attributes **TransactionTimeStart** and **TransactionTimeEnd** actually represent a transaction time interval which is closed at its lower and open at its upper bound.

Bitemporal Database

In some applications, only one of the dimensions is needed and in other cases both time dimensions are required, in which case the temporal database is called a **bitemporal database.** It uses both valid time and transaction time in a single database.

Deductive databases

A deductive database system typically specify rules through a **declarative language** – a language in which we specify what to achieve rather than how to achieve it. It is a database system that can make deductions (i.e., conclude additional facts) based on rules and facts stored in the (deductive) database. It is also related to the field of logic programming and the **Prolog** language.

A variation of Prolog called **Datalog** can also be used to define rules declaratively in conjunction with an existing set of relations. A deductive database used two main types of specifications: **facts** and **rules.**

Facts are specified in a manner similar to the way relations are specified, except that it is not necessary to include the attribute names.

Rules are somewhat similar to relational views. They specify virtual relations that are not actually stored but that can be formed from the facts by applying inference mechanisms based on the rule specifications.

The deductive database work based on logic has used Prolog as a starting point. A variation of Prolog called Datalog is used to define rules declaratively in conjunction with an existing set of relations, which are themselves treated as literals in the language.

The notation used in Prolog/Datalog is based on providing predicates with unique names.

A predicate has an implicit meaning, which is suggested by the predicate name, and a fixed number of arguments. The predicate‘s type is determined by its arguments:

- If the arguments are all constant values, the predicate simply states that a certain fact is true
- If the predicate has variables as arguments, it is either considered as a query or as part of a rule or constraint.

[TU Question]:- Describe multimedia database and what are the different types of multimedia data that are available in current systems?

[TU Question]:- Enumerate the limitations of conventional database compared to multimedia database.

Multimedia Databases

Multimedia databases provide features that allow users to store and query different types of multimedia information, which includes **images**, **video clips**, **audio clips**, and **documents.**

Simply, a Multimedia database (MMDB) is a collection of related multimedia data. The multimedia data include one or more primary media data types such as text, images, graphic objects (including drawings, sketches and illustrations) animation sequences, audio and video.

Multimedia databases provide features that allow users to store and query different types of multimedia information, which includes images (such as photos or drawings), video clips (such as movies, newsreels, or home videos), audio clips (such as songs, phone messages, or speeches), and documents (such as books or articles). The main types of database queries that are needed involve locating multimedia sources that contain certain objects of interest. For example, one may want to locate all video clips in a video database that include a certain person, say Michael Jackson. One may also want to retrieve video clips based on certain activities included in them, such as video clips where a soccer goal is scored by a certain player or team.

Advantages of Multimedia database

- It is very user-friendly. It doesn’t take much energy out of the user, in the sense that you can sit and watch the presentation, you can read the text and hear the audio.

- It is multi sensorial. It uses a lot of the user's senses while making use of multimedia, for example hearing, seeing and talking.
- It is integrated and interactive. All the different mediums are integrated through the digitization process. Interactivity is heightened by the possibility of easy feedback.
- It is flexible. Being digital, this media can easily be changed to fit different situations and audiences.
- It can be used for a wide variety of audiences, ranging from one person to a whole group.

Disadvantages

- Information overload. Because it is so easy to use, it can contain too much information at once.
- It takes time to compile. Even though it is flexible, it takes time to put the original draft together.
- It can be expensive. Multimedia makes use of a wide range of resources, which can cost you a large amount of money.
- Too much makes it unpractical. Large files like video and audio has an effect of the time it takes for your presentation to load. Adding too much can mean that you have to use alarger computer to store the files.

Characteristics of different multimedia sources

Image

- Typically stored either in raw form as a set of pixel or cell values, or in compressed form to save space
- Each image can be represented by an m by n grid of cells and each cell contains a pixel value that describe the cell content
- Compression standards, such as GIF or JPEG, use various mathematical transformations to reduce the number of cells stored.

Video

- Typically represented as a sequence of frames, where each frame is a still image
- The video is divided into video segments, where each segment is made up of sequence of contiguous frames that includes the same objects/activities. Each segment is identified by its starting and ending frames
- An indexing technique called frame segment trees has been proposed for video indexing. The index includes both objects and activities
- Videos are also often compressed using standards such as MPEG

Text/Document

- The full text of some article, book, or magazine
- Typically indexed by identifying the keywords that appear in the text and their relative frequencies. Because there are too many keywords when attempting to index, a technique called singular value decomposition (SVD) can be used to reduce the number of keywords
- An indexing technique called telescoping vector trees, or TV-trees can be used to group similar documents together

Audio

- Include stored recorded messages
- Discrete transforms can be used to identify the main characteristics of a certain person's voice in order to have similarity based indexing and retrieval. Futures include loudness, intensity, pitch, and clarity

Types of Multimedia are available in current systems:

DBMSs have been constantly adding to the types of data they support. Today many 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. 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.
- **Audio**: Sample data generated from aural recordings in a string of bits in digitized form. Analog recordings are typically converted into digital form before storage.
- **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.

[TU Question]:- Discuss some applications of active database. How do spatial databases differ from regular database?

<u>Spatial Database</u>

A spatial database is a database that is enhanced to store and access spatial data or data that defines a geometric space. The special data stored in the form of co-ordinate form. These data are often associated with geographic locations and features, or constructed features like cities. Data on spatial databases are stored as coordinates, points, lines, polygons and topology. Some spatial databases handle more complex data like three-dimensional objects, topological coverage and linear networks.

The main goal of a spatial database system is the effective and efficient handling of spatial data types in two, three or higher dimensional spaces, and the ability to answer queries taking into consideration the spatial data properties.

A common example of spatial data can be seen in a road map. A road map is a 2-dimensional object that contains points, lines, and polygons that can represent cities, roads, and political boundaries such as states or provinces. A road map is a visualization of geographic information.

Examples of spatial data types are:

- **Point:** characterized by a pair of (x, y) values,
- **Line segment:** characterized by a pair of points,
- **Rectangle:** characterized by its lower-left and upper-right corners,
- **Polygon:** comprised by a set of points, defining its corners.

Examples of spatial datasets

Three types of spatial queries

- **Range query** – Finds the objects of a particular type that are within a particular distance from a given location. For example, finds all hospitals within the Kathmandu city area, or finds all ambulances within five miles of an accident location

- **Nearest neighbor query –** Finds an object of a particular type that is closest to a given location. For example, finds the police car that is closest to a particular location
- **Spatial joins or overlays –** Typically joins the objects of two types based on some spatial condition, such as the objects intersecting or overlapping spatially or being within a certain distance of one another. For example, find all cities that fall on a major highway or finds all homes that are within two miles of a lake

Geographic Information Systems (GIS)

Geographic information systems (GIS) are used to collect, model, store, and analyze information describing physical properties of the geographical world. GIS technology integrates common database operations such as query and statistical analysis with the unique visualization and geographic analysis benefits offered by maps. Map making and geographic analysis are not new, but a GIS performs these tasks better and faster than do the old manual methods. And, before GIS technology, only a few people had the skills necessary to use geographic information to help with decision making and problem solving. Today, Professionals in every field are increasingly aware of the advantages of thinking and working geographically.

The scope of GIS broadly encompasses two types of data:

1. **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
2. **Non-spatial data**, such as socio-economic data (like census data), economic data, or sales or marketing information.

Components of GIS

A working Geographic Information System seamlessly integrates five key components: hardware, software, data, people, and methods.

Hardware: Hardware includes the computer on which a GIS operates, the monitor on which results are displayed, and a printer for making hard copies of the results.

GIS software: It provides the functions and tools needed to store, analyze, and display geographic information. Key software components include tools for the input and manipulation of geographic information, a database management system (DBMS), tools that support geographic query, analysis, and visualization, and a graphical user interface (GUI) for easy access to tools.

Data: Possibly the most important component of a GIS is the data. A GIS will integrate spatial data with other data resources and can even use a database management system, used by most organizations to organize and maintain their data, to manage spatial data. There are three ways to obtain the data to be used in a GIS. Geographic data and related tabular data can be collected in-house or produced by digitizing images from aerial photographs or published maps.

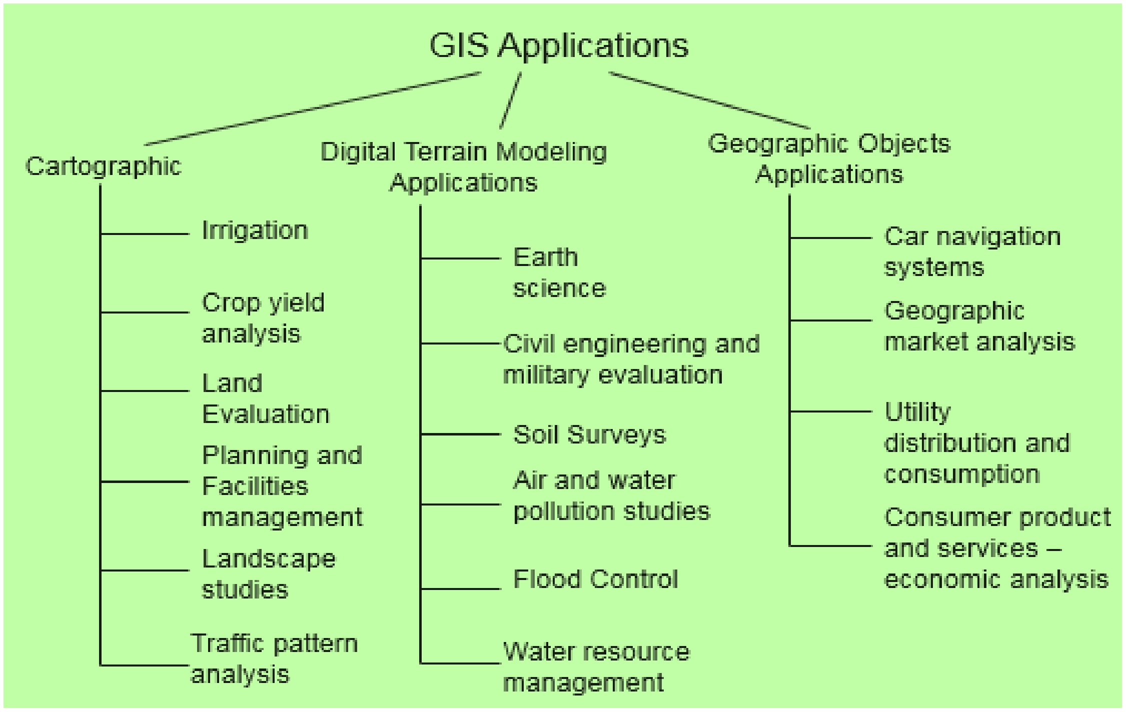
People: GIS users range from technical specialists who design and maintain the system to those who use it to help them perform their everyday work.

Methods: A successful GIS operates according to a well-designed plan and business rules, which are the models and operating practices unique to each organization.

GIS applications

It is possible to divide GISs into three categories:

1. Cartographic applications
2. Digital terrain modeling applications, and
3. Geographic objects applications


GIS Applications
Cartographic
Irrigation
Crop yield analysis
Land Evaluation
Planning and Facilities management
Landscape studies
Traffic pattern analysis
Digital Terrain Modeling Applications
Earth science
Civil engineering and military evaluation
Soil Surveys
Air and water pollution studies
Flood Control
Water resource management
Geographic Objects Applications
Car navigation systems
Geographic market analysis
Utility distribution and consumption
Consumer product and services – economic analysis

In cartographic and terrain modeling applications, variations in spatial attributes are captured – for example, soil characteristics, crop density, and air quality.
In geographic object applications, objects of interest are identified from a physical domain – for example, power plants, electoral districts, property parcels, product distribution districts, and city

landmarks; These objects are related with pertinent application data – for example, power consumption, voting patterns, property sales volumes, product sales volume, and traffic density. The first two categories of GIS applications require a field-based representation, whereas the third category requires an object-based one. The cartographic approach involves special functions that can include the overlapping of layers of maps to combine attribute data. Digital terrain modeling requires a digital representation of parts of earth's surface using land elevations at sample points that are connected to yield a surface model showing the surface terrain. In object-based geographic applications, additional spatial functions are needed to deal with data.

Data management requirements of GIS

The functional requirements of the GIS applications translate into the following database requirements.

- Data Modeling and Representation.
- Data Analysis
- Data Integration
- Data Capture

1. Data Modeling and Representation

GIS data can be broadly represented in tow formats:

- vector and
- raster

Vector data represents geometric objects such as points, lines, and polygons

Raster data is characterized as an array of points, where each point represents the value of an attribute for a real-world location. Informally, raster images are n-dimensional array where each entry is a unit of the image and represents an attribute. Two-dimensional units are called pixels, while three-dimensional units are called voxels. Three-dimensional elevation data is stored in a raster-based digital elevation model (DEM) format. This model is commonly used for analytical applications such as modeling, map algebra, and more advanced techniques of feature extraction and clustering.

2. Data Analysis

GIS data undergoes various types of analysis for example, in applications such as soil erosion studies, environmental impact studies, or hydrological runoff simulations, data may undergo various types of **geomorphometric analysis** – measurements such as slope values, gradients (the rate of change in altitude), aspect (the compass direction of the gradient), profile convexity (the rate of change of gradient), plan convexity (the convexity of contours and other parameters).

3. Data Integration

GISs must integrate both vector and raster data from a variety of sources. Sometimes edges and regions are inferred from a raster image to form a vector model, or conversely, raster images such as aerial photographs are used to update vector models

4. Data Capture

The first step in developing a spatial database for cartographic modeling is to capture the two-dimensional or three-dimensional geographical information in digital form – a process that is sometimes impeded by source map characteristics such as resolution, type of projection, map scales, cartographic licensing, diversity of measurement techniques, and coordinate system differences. Spatial data can also be captured from remote sensors in satellites such as Landsat, NORA, and Advanced Very High Resolution Radiometer as well as SPOT HRV (High Resolution Visible Range Instrument.

Specific GIS data operations

GIS applications are conducted through the use of special operators such as the following:

- **Interpolation** – derives elevation data
- **Interpretation** – involves the interpretation of operations on terrain data such as editing, smoothing, reducing details, and enhancing.
- **Proximity analysis** – computation of "zones of interest" around objects
- **Raster image processing** – can be divided into (1) map algebra and (2) digital image analysis
- **Analysis of networks** – analysis of networks for segmentation, overlays, and so on

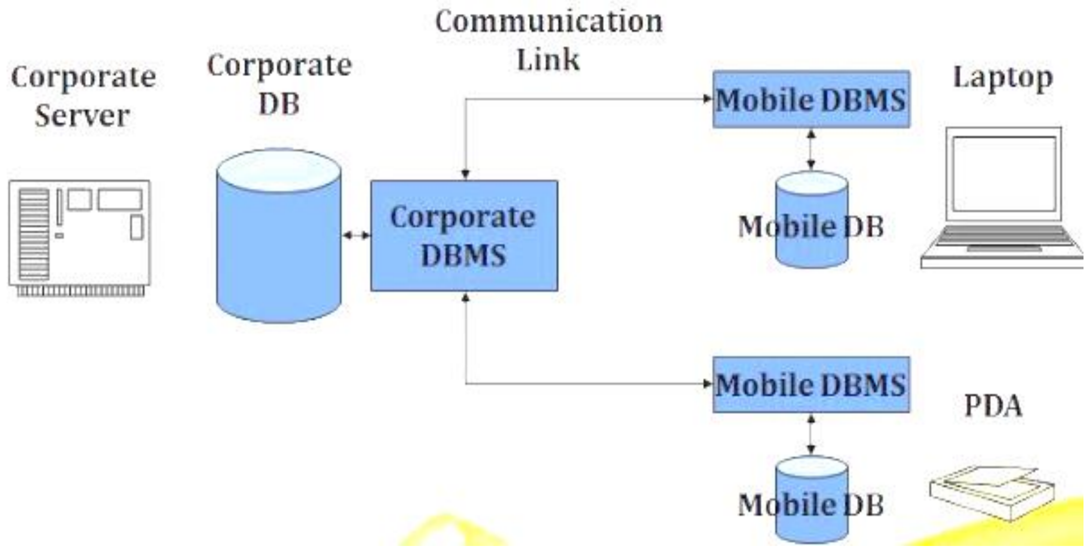
Mobile Databases

Recent advances in portable and wireless technology have led to **mobile computing**, a new dimension in data communication and processing. A mobile database is a database that can be connected to by a mobile computing device over a mobile network. It is portable and physically separate from the corporate database server. But Mobile Database is capable of communicating with that corporate database server from remote sites allowing the sharing of corporate database.

Mobile computing devices (e.g., smartphones and PDAs) store and share data over a mobile network, or a database which is actually stored by the mobile device. A mobile database is a database that resides on a mobile device such as a PDA, a smart phone, or a laptop. Such devices are often limited in resources such as memory, computing power, and battery power.

Mobile computing architecture, characteristics of mobile environments, data management issues. It can also be defined as a system with the following structural and functional properties.

- Distributed system with mobile connectivity (A mode in which a client or a server can establish communication with each other whenever needed)
- Full database system capability
- Complete spatial mobility
- Wireless and wired communication capability



Ability of mobile DBMS

- Communicate with centralized database server through modes such as wireless or Internet access
- Replicate data on centralized database server and mobile device
- Synchronize data on centralized database server and mobile device
- Capture data from various sources such as Internet
- Manage/analyze data on the mobile device
- Create customized mobile applications

[TU Question: - Describe the characteristics of mobile computing environment in detail]

[TU Question: - Explain mobile computing architecture with suitable diagram]

Mobile Computing Architecture

The general architecture of a mobile platform is a distributed architecture where a number of computers, generally referred to as **Fixed Hosts** and **Base Stations**, are interconnected through a high speed wired network. Fixed hosts are general purpose computers configured to manage mobile units. Base stations function as a gateways to the fixed network for the **Mobile Units**; they are equipped with wireless interfaces and offer network access services of which mobile units are clients.

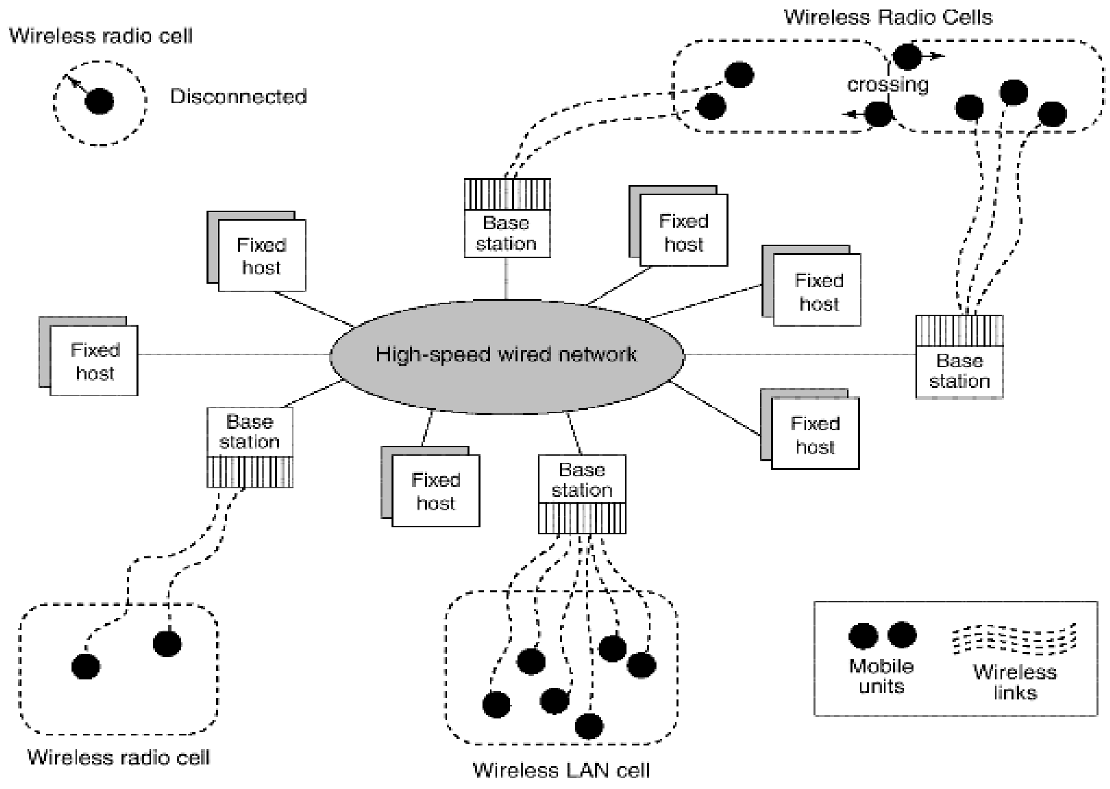


Fig: A general architecture of a mobile platform

Wireless Communications

The wireless medium on which mobile units and base stations communicate have bandwidths significantly lower than those of a wired network. Some wireless access options allow seamless roaming throughout a geographical region (e.g., cellular networks), whereas Wi-Fi networks are localized around a base station; Some wireless networks, such as Wi-Fi and Bluetooth, use unlicensed areas of frequency spectrum, which may cause interference with other appliances, such as cordless telephones; Modern wireless networks can transfer data in units called packets, that are commonly used in wired networks in order to conserve bandwidth; Wireless applications must consider these characteristics when choosing a communication option

Client/Network Relationships

Mobile units can move freely in a **geographic mobility domain**, an area that is circumscribed by wireless network coverage; To manage the mobility of units, the entire geographic mobility domain is divided into one or more smaller domains, called **cells**, each of which is supported by at least one base station; The mobile discipline requires that the movement of mobile units be unrestricted throughout the cells of a geographic mobility domain, while maintaining information **access contiguity** – i.e., movement, especially intercell movement, does not negatively affect the data retrieval process; This architecture is designed for a fixed network, emulating a traditional client-server architecture

Characteristics of Mobile Computing Environments

The characteristics of mobile computing include **high communication latency**, **intermittent wireless connectivity**, **limited battery life**, and **changing client location**

- **Latency** is caused by the processes unique to the wireless medium, such as coding data for wireless transfer, and tracking and filtering wireless signals at the receiver.
- **Intermittent connectivity** can be intentional or unintentional; unintentional disconnections happen in areas where wireless signals cannot reach, e.g., elevator shafts or subway tunnels; Intentional disconnections occur by user intent, e.g., during an airplane takeoff, or when the mobile device is powered down.
- **Battery life** is directly related to battery size, and indirectly related to the mobile device's capabilities and overall efficiency.
- **Client locations** are expected to change, which alters the network topology and may cause their data requirements to change.

  First, servers must keep track of client locations in order to route messages to them efficiently. Second, client data should be stored in network location that minimize the traffic necessary to access it. Keeping data in a fixed location increases access latency if the client moves far away from it.

All these characteristics impact data management, and robust mobile applications must consider them. To compensate for high latencies and unreliable connectivity, clients' cache replicas of important, frequently accessed data, and work offline, if necessary; besides increasing data availability and response time, caching can also reduce client power consumption by eliminating the need to make energy-consuming wireless data transmission for each data access

Data Management Issues

From a data management standpoint, mobile computing may be considered a variation of distributed computing. Mobile databases can be distributed under two possible scenarios:

1. The entire database is distributed mainly among the wired components, possibly with full or partial replication; a base station or fixed host manages its own database with a DBMS-like functionality, with additional functionality for locating mobile units and additional query and transaction management features to meet the requirements of mobile environments.
2. The database is distributed among wired and wireless components; Data management responsibility is shared among base stations or fixed hosts and mobile units.

The distributed data management issues can also be applied to mobile databases with the following additional considerations and variations:

- **Data distribution and replication** – Data is unevenly distributed among the base stations and mobile units.
- **Transactions models** – Issues of fault tolerance and correctness of transactions are aggravated.
- **Query processing** – Awareness of where data is located is important and affects the cost/benefit analysis of query processing; Query optimization is more complicated because of mobility and rapid resource changes of mobile units.
- **Recovery and fault tolerance** – The mobile database environment must deal with site, media, transaction, and communication failure.
- **Mobile database design** – The global name resolution problem for handling queries is compounded because of mobility and frequent shutdown.
- **Location-based service** – As clients move, location-dependent cache information may become stale.
- **Division of labor** – Certain characteristics of the mobile environment force a change in the division of labor in query processing.
- **Security** – Mobile data is less secure than that which is left at the fixed location.

Unit 4
New database applications and environments

By

Bhupendra Singh Saud

Data Mining

Data mining refers to the mining or discovery of new information in terms of patterns or rules from vast amounts of data. It is also defined as the process of finding interesting structure in data. Data mining employs one or more computer learning techniques such as machine learning, statistics, neural networks, and genetic algorithms to automatically analyze and extract knowledge from data. To be practically useful, data mining must be carried out efficiently on large files and databases

The process of Discovering meaningful patterns & trends often previously unknown, by shifting large amount of data, using pattern recognition, statistical and Mathematical techniques is called **data mining**. It is also defined as a group of techniques that find relationship that have not previously been discovered.

Data mining is a logical process that is used to search through large amount of data in order to find useful data. The goal of this technique is to find patterns that were previously unknown. Once these patterns are found they can further be used to make certain decisions for development of their business.

Three steps involved are:

1. Exploration
2. Pattern identification
3. Deployment

Exploration: In the first step of data exploration data is cleaned and transformed into another form, and important variables and then nature of the data based on the problem are determined.

Pattern identification: Once data is explored, refined and defined for the specific variables the second step is to form pattern identification. Identify and choose the patterns which make the best prediction.

Deployment: Patterns are deployed for desired outcome.

Functions of data mining

The types of information obtainable from data mining include associations, sequences, classifications, clusters, and forecasts.

- **Association:** Association is one of the best known data mining technique. In association, a pattern is discovered based on a relationship between items in the same transaction. That is the reason why association technique is also known as relation technique. The association technique is used in market basket analysis to identify a set of products that customers frequently purchase together. For instance, books that tends to be bought together. If a customer buys a book, an online bookstore may suggest other associated books. If a person buys a camera, the system may suggest accessories that tend to be bought along with cameras.
- **Prediction:** The prediction, as it name implied, is one of a data mining techniques that discovers relationship between independent variables and relationship between dependent and independent variables. For instance, the prediction analysis technique can be used in sale to predict profit for the future if we consider sale is an independent variable, profit could be a dependent variable. For instance, when a person applies for a credit card, the credit-card company wants to predict if the person is a good credit risk. The prediction is to be based on known attributes of the person, such as age, income, debts, and past debt repayment history.
- **Classification:** Classification is a classic data mining technique based on machine learning. Basically classification is used to classify each item in a set of data into one of predefined set of classes or groups. Classification method makes use of mathematical techniques such as decision trees, linear programming, neural network and statistics. For example, we can apply classification in application that "given all records of employees who left the company; predict who will probably leave the company in a future period." In this case, we divide the records of employees into two groups that named "leave" and "stay". And then we can ask our data mining software to classify the employees into separate groups.
- **Clustering:** Clustering is a data mining technique that makes meaningful or useful cluster of objects which have similar characteristics using automatic technique. The clustering technique defines the classes and puts objects in each class, while in the classification techniques, objects are assigned into predefined classes. For

example in a library, there is a wide range of books in various topics available. The challenge is how to keep those books in a way that readers can take several books in a particular topic without hassle. By using clustering technique, we can keep books that have some kinds of similarities in one cluster or one shelf and label it with a meaningful name.

Applications of Data Mining

Data mining is widely used in diverse areas. There are a number of commercial data mining system available today and yet there are many challenges in this field. In this tutorial, we will discuss the applications and the trend of data mining.

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.

Web mining

The discovery and analysis of useful patterns and information from the World Wide Web or simply web is called web mining. Web mining is the application of data mining technique to find interesting and potentially useful knowledge from web data. So web mining is the application of data mining technique to extract knowledge from web data, including web documents, hyperlinks between documents, usage logs of web sites etc.
Businesses might turn to Web mining to help them understand customer behavior, evaluate the effectiveness of a particular Web site, or quantify the success of a marketing campaign. For instance, marketers use Google Trends and Google Insights for Search services, which track the popularity of various words and phrases used in Google search queries, to learn what people are interested in and what they are interested in buying.

Data Warehouse

A data warehouse is a repository of multiple heterogeneous data sources organized under a unified schema at a single site to facilitate management decision making. A data warehouse is a subject-oriented, integrated, time-variant and nonvolatile collection of data in support of management's decision-making process.

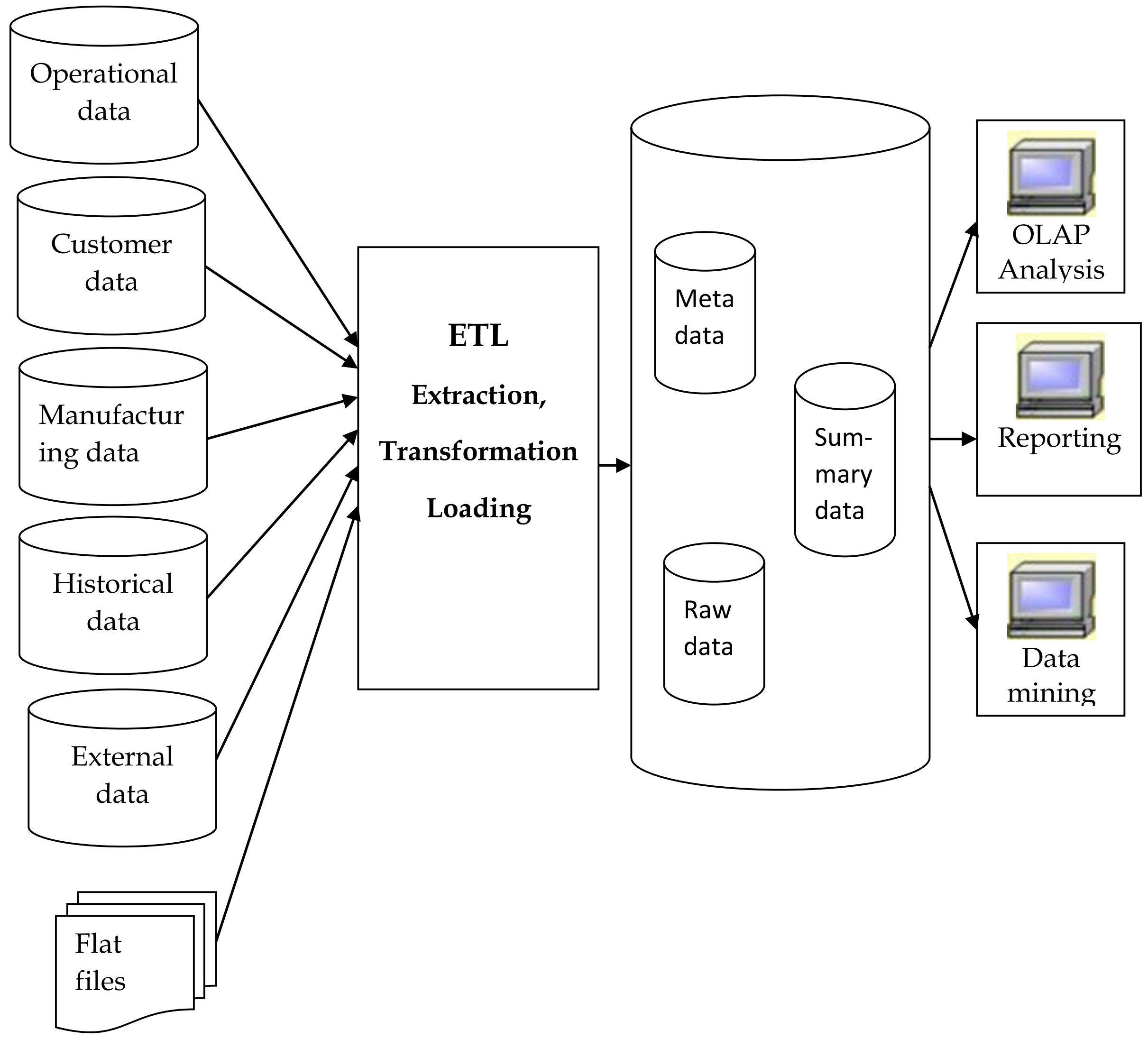
a. **Subject-Oriented**: A data warehouse can be used to analyze a particular subject area. For example, "sales" can be a particular subject.

b. **Integrated**: A data warehouse integrates data from multiple data sources. For example, source A and source B may have different ways of identifying a product, but in a data warehouse, there will be only a single way of identifying a product.

c. **Time-Variant**: Historical data is kept in a data warehouse. For example, one can retrieve data from 3 months, 6 months, 12 months, or even older data from a data warehouse. This contrasts with a transactions system, where often only the most recent data is kept. For example, a transaction system may hold the most recent address of a customer, where a data warehouse can hold all addresses associated with a customer.

d. **Non-volatile**: Once data is in the data warehouse, it will not change. So, historical data in a data warehouse should never be altered.

A **data warehouse** is a repository of current and historical data of an organization that are organized to facilitate reporting and analysis. The data originate in many core operational transaction systems, such as systems for sales, customer accounts, and manufacturing, and may include data from Web site transactions. The data warehouse consolidates and standardizes information from different operational databases so that the information can be used across the enterprise for management analysis and decision making. Figure below illustrates how a data warehouse works. The data warehouse makes the data available for anyone to access as needed, but it cannot be altered. A data warehouse system also provides a range of ad hoc and standardized query tools, analytical tools, and graphical reporting facilities. Many firms use intranet portals to make the data warehouse information widely available throughout the firm.



Features of Data warehouse

- It is separate from operational database.
- Integrates data from heterogeneous systems
- Store huge amount of data, more historical data than current data
- Does not require data to be highly accurate
- Queries are generally complex
- Provides an integrated and total view of the enterprise
- Makes the enterprise's current and historical information easily available for decision making
- Makes decision support transaction possible without hindering operational systems
- Renders the organization's information consistent
- Presents a flexible and interactive source of strategic information

Need of data warehouse

Data warehouse is needed for the following reasons:

1. **Business users:** Business users require data warehouse to view summarized data from past. Since these people are non-technical, the data may be presented to them in a very simple form.
2. **Make strategic decision:** Some strategies may be depending upon the data in data warehouse. So data warehouse contribute in making strategic decisions.
3. **Store historical data:** Data warehouse is required to store the time variable data from past. This data is made to be used for various purposes.
4. **For data quality and consistency:** Bringing the data from different sources at a common place, user can efficiently undertake to bring the uniformity and consistency in data.
5. **High response time**: Data warehouse has to be ready for fairly unexpected loads and types of queries, which demands a high degree of flexibility and quick response time.

How does a data warehouse differ from a database?

There are a number of fundamental differences which separate a data warehouse from a database. The biggest difference between them is that most database place an emphasis on a single application, and this application will generally be one that is based on transaction. If the data is analyzed, it will be done within a single domain. In contrast, data warehouses deal with multiple domains simultaneously.

Because data warehouse deals with multiple subject areas, the data warehouse finds connections between them. This allows the data warehouse to show how the company is performing as a whole, rather than in individual areas.

Another powerful aspect of data warehouse is their ability to support the analysis of trends. They are not volatile, and the information stored in them doesn't change as much as it would in a common database. Some of the major differences between them are listed below:

Database	Data Warehouse
1. In database tables and joins of different tables are complex since they are normalized for RDBMS. This is done to reduce redundant data and to save storage space. 2. Entity Relational modeling techniques are used for RDBMS database design. 3. Performance is low for analysis queries. 4. Database is the place where the data is taken as a base and managed to get available fast and efficient access. 5. Optimized for write operation. 6. Used for Online Transaction Processing (OLTP) but can be used for other purpose such as data warehousing. This records the data from the user for history.	1. In data warehouse tables and joins are simple since they are de-normalized. This is done to reduce the response time for analytical queries. 2. Data modeling techniques are used for Data Warehouse design. 3. High performance for analytical queries 4. Data warehouse is the place where the application data is managed for analysis and reporting purpose. 5. Optimized for read operations. 6. Used for Online Analytical Processing (OLAP). This reads the historical data for the users for business decision.

Meta Data

Meta data is the data about data or documentation about the data that is needed by the users. Another description of metadata is that it is structured data which describes the characteristics of a resource. Several examples of metadata are:

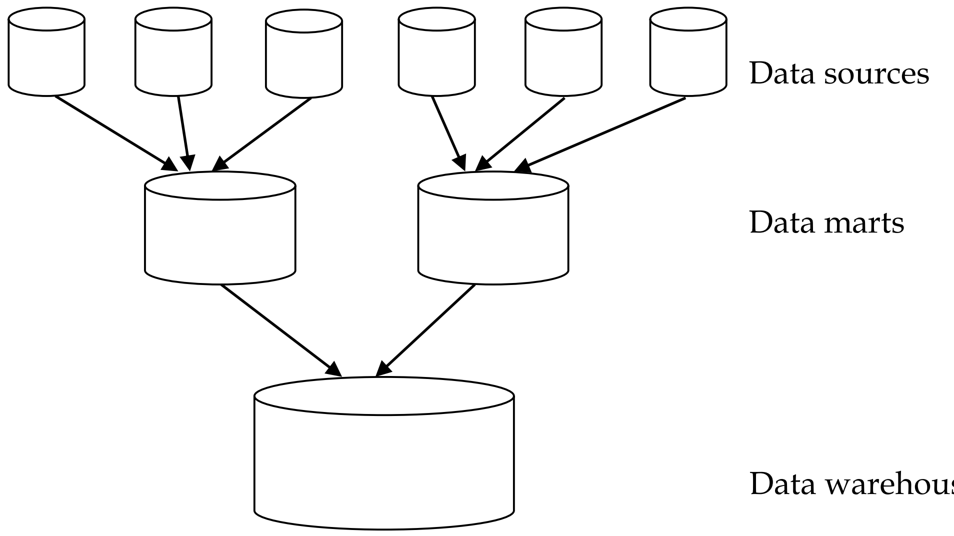
1. The table of contents and the index in a book may be considered metadata for the book.

2. A library catalogue may be considered metadata. The catalogue metadata consists of a number of predefined elements representing specific attributes of a resource, and each element can have one or more values.
3. Another example of metadata is data about the tables and figures in a document. A table has a name and there are column names of the table that may be considered metadata. The figures also have titles or names.

Data Marts

Data mart is a database that contains a subset of data present in a data warehouse. Data marts are created to structure the data in a data warehouse according to issues such as hardware platforms and access control strategies. We can divide a data warehouse into data marts after the data warehouse has been created. The implementation cycle of the data mart is likely to be measured in weeks rather than months or years.

Companies often build enterprise-wide data warehouses, where a central data warehouse serves the entire organization, or they create smaller, decentralized warehouses called data marts. A **data mart** is a subset of a data warehouse in which a summarized or highly focused portion of the organization's data is placed in a separate database for a specific population of users. For example, a company might develop marketing and sales data marts to deal with customer information. A data mart typically focuses on a single subject area or line of business, so it usually can be constructed more rapidly and at lower cost than an enterprise-wide data warehouse.



Reasons for creating a data mart

- Creates collective view by a group of users
- Easy access to frequently needed data
- Ease of creation
- Improves end-user response time
- Lower cost than implementing a full data warehouse
- Potential users are more clearly defined than in a full data warehouse
- Contains only business essential data and is less cluttered

Tools for business Intelligence

Once data have been captured and organized in data warehouses and data marts, they are available for further analysis using tools for business intelligence. Business intelligence tools enable users to analyze data to see new patterns, relationships, and insights that are useful for guiding decision making. Principal tools for business intelligence include software for database querying and reporting, tools for multidimensional data analysis (online analytical processing), and tools for data mining.

The key general categories of business intelligence tools are:

- Spreadsheets
- Reporting and querying software: tools that extract, sort, summarize, and present selected data
- OLAP: Online analytical processing
- Digital dashboards
- Data mining
- Data warehousing
- Local information systems

Online analytical processing (OLAP): Multidimensional data analysis

OLAP supports multidimensional data analysis, enabling users to view the same data in different ways using multiple dimensions (data cube). Multidimensional data models are designed expressly to support data analyses. The goal of multidimensional data models is to support analysis in a simple and faster way by executives, managers and business professionals. These people are not interested in the overall architecture.

Suppose your company sells five different products—Laptops, Computers, TVs, Camera and Mobiles—in the East, West, North and Central regions. If you wanted to ask a fairly straightforward question, such as how many Computers were sold in the last week, you could easily find the answer by using sales database. But what if you wanted to know how many Computers sold in each of your sales regions and compare actual results with projected sales, then the querying becomes complicated. In such a case OLAP is used.

Each aspect of information—product, pricing, cost, region, or time period—represents a different dimension. So, a product manager could use a multidimensional data analysis tool to learn how many Computers were sold in the East reason in this week, how that compares with the previous week, and how it compares with the sales forecast. OLAP enables users to obtain online answers to ad hoc questions such as these in a fairly rapid amount of time, even when the data are stored in very large databases, such as sales figures for multiple years.

Time	Product	Location	Sales
2072-01-01	Computer	East region	30
2072-01-01	Laptop	West region	20
2072-01-01	Camera	Central region	50
2072-01-07	Mobile	East region	11
2072-01-07	TV	North region	23
2072-01-15	Computer	West region	54
2072-01-15	Laptop	Central region	09
2072-01-25	Laptop	East region	32
2072-01-25	TV	West region	19

Fig: Tabular representation

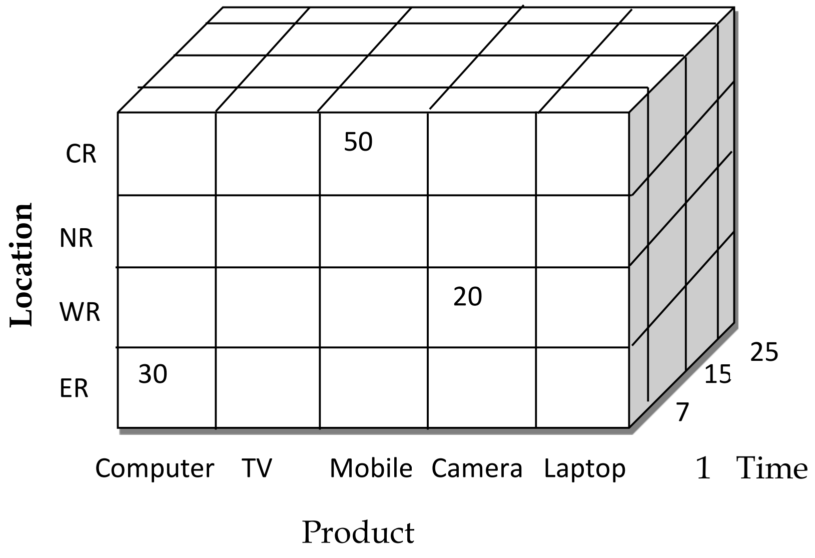


Fig: multidimensional representation

Figure above shows a multidimensional model that could be created to represent products, regions, time, and sales.

OLAP consists of four basic analytical operations: consolidation (roll-up), drill-down, slicing and dicing, and pivoting.

- **Slicing**: The slice operation is the act of picking a rectangular subset of a cube by choosing a single value for one of its dimensions, creating a new cube with one fewer dimension. For instance, the analyst may wish to see a cross-tab on *item-name* and *color* for a fixed value of *size*, for example, large, instead of the sum across all sizes.
- **Dicing:** The dice operation produces a sub-cube by allowing the analyst to pick specific values of multiple dimensions. For example, the analyst may wish to see information about medium and large sized shorts and pants in dark and pastel color.
- **Drill Down**: Drill-down is the reverse of roll-up operation. It navigates from less detailed data to more detailed data. It allows the user to navigate among levels of data ranging from the most summarized (up) to the most detailed (down). It means it is the operation of moving from finer-granularity data to a coarser granularity. For example, analyst may move from summary of pant sales to color wise sales of pant.
- **Roll-up:** A roll-up involves summarizing the data along a dimension. The summarization rule might be computing totals along a hierarchy or applying a set of formulas such as "profit = sales - expenses". . For example, analyst may move from color wise sales of pant sales to total sales of pant.
- **Pivoting or rotating:** ***Pivoting is t***he process of selecting or changing the dimensions used in a cross-tab**.** Each cross-tab is a two-dimensional view on a multidimensional data cube. For instance the analyst may select a cross-tab on *item-name* and *size*, or a cross-tab on *color* and *size*.

OLAP **vs OLTP (online transaction processing)**

OLAP is an acronym for Online Analytical Processing. **OLAP** performs multidimensional analysis of business data and provides the capability for complex calculations, trend analysis, and sophisticated data modeling.

OLTP (online transaction processing) is a class of software programs capable of supporting transaction-oriented applications on the Internet. Typically, **OLTP** systems are used for order entry, financial transactions, customer relationship management (CRM) and retail sales.

OLAP	**OLTP**
Involves historical processing of information	Involves day-to-day processing
OLAP systems are used by knowledge workers such as executives, managers and analysts.	OLTP systems are used by clerks, DBAs, or database professionals.
Useful in analyzing the business.	Useful in running the business.
Based on Star Schema, Snowflake, Schema and Fact Constellation Schema.	Based on Entity Relationship Model.
Provides summarized and consolidated data.	Provides primitive and highly detailed data.
Highly flexible.	Provides high performance.

Unit 5
Database Related Standards
By
Bhupendra Singh Saud

Database related standards

SQL standards

The language SQL is standardized by international standard bodies such as ISO and ANSI. By using standard SQL it should be easier to move applications between different database systems without the need to rewrite a substantial amount of code. Using standard SQL does not give any warranty though as all vendors does not implement all features in the standard.

SQL provides statements for a variety of tasks, including:

- Querying data
- Inserting, updating, and deleting rows in a table
- Creating, replacing, altering, and dropping objects
- Controlling access to the database and its objects
- Guaranteeing database consistency and integrity

Mimer Information Technology's policy is to develop Mimer SQL as far as possible in accordance with the established standard. This enables users to switch to and from Mimer SQL easily. The standard for SQL is ISO/IEC 9075:1999, referred to here as SQL-99.

The SQL standard has gone through a number of revisions

Year	**Name**	**Alias**	Comments
1986	SQL-86	SQL-87	First published by ANSI. Ratified by ISO in 1987.
1989	SQL-89		Minor revision.
1992	SQL-92	SQL2	Major revision.
1999	SQL:1999	SQL3	Added object-oriented features, introduced OLAP,…
2003	SQL:2003		Introduced XML-related features,…

SQL 1999

- Intended as a major enhancement.
- Characterized as "object-oriented SQL“.
- In addition to the object oriented extensions, there are some other new features like; Triggers, Stored procedures and user-defined functions, Recursive queries, OLAP, SQL procedural constructs, Expressions in ORDER BY Savepoints, Update through unions and joins.
- The new features are divided into five category: new data types, new predicates, enhanced semantics, additional security, and active database.
- SQL:1999 has four new data types:
    a. Large Object (LOB) type
        i. CHARACTER LARGE OBJECT (CLOB)
        ii. BINARY LARGE OBJECT (BLOB)
    b. Boolean type
    c. Two new *composite* types: ARRAY (storing collections of values in a column) and ROW (storing structured values in single columns of the database)
    d. Distinct types
-

SQL: 1999 is much more than merely SQL-92 plus object technology. It involves additional features that we consider to fall into SQL’s relational heritage, as well as a total restructuring of the standards documents themselves with an eye towards more effective standards progression in the future. The features of SQL: 1999 can be partitioned into its “relational features” and its “Object-oriented features”. Although we call this category of features “relational”, we’ll quickly recognize that it’s more appropriately categorized as “features that relate to SQL’s traditional role and data model” somewhat less pithy phrase. The features here are not strictly limited to the relational model, but are also unrelated to object orientation. These features are often divided into about five groups: new data types, new predicates, enhanced semantics, additional security, and active database.

New Data Types SQL: 1999 has four new data types (although one of them has some identifiable variants). The first of these types is the LARGE OBJECT, or LOB, type. This is the type with variants: CHARACTER LARGE OBJECT (CLOB) and BINARY LARGE OBJECT (BLOB). CLOBs behave a lot like character strings, but have restrictions that disallow their use in PRIMARY KEYs or UNIQUE predicates, in FOREIGN KEYs, and in comparisons other than purely equality or inequality tests.

BLOBs have similar restrictions. (By implication, LOBs cannot be used in GROUP BY or ORDER BY clauses, either.) Applications would typically not transfer entire LOB values to and from the database (after initial storage, that is), but would manipulate LOB values through a special client-side type called a LOB locator. In SQL:1999, a locator is a unique binary value that acts as a sort of surrogate for a value held within the database; locators can be used in operations (such as SUBSTRING) without the overhead of transferring an entire LOB value across the client-server interface.

Another new data type is the BOOLEAN type, which allows SQL to directly record truth values true, false, and unknown. Complex combinations of predicates can also be expressed in ways that are somewhat more user-friendly than the usual sort of restructuring might make them

```
WHERE COL1 > COL2 AND
   COL3 = COL4 OR
UNIQUE (COL6) IS NOT FALSE
```

SQL: 1999 also has two new composite types: ARRAY and ROW. The ARRAY type allows one to store collections of values directly in a column of a database table. For examples WEEKDAYS VARCHAR (10) ARRAY[7]

New Predicates

SQL: 1999 has three new predicates, one of which we'll consider along with the object-oriented features. The other two are the SIMILAR predicate and the DISTINCT predicate. Since the first version of the SQL standard, character string searching has been limited to very simple comparisons (like =, >, or <>) and the rather rudimentary pattern matching capabilities of the LIKE predicate:

WHERE NAME LIKE '%SMIT_'

SQL 2003

- Makes revisions to all parts of SQL: 1999.
- Adds a brand new part: SQL/XML (XML-Related Specifications).
- New features are categorized as:

    - New data types,
    - Enhancements to SQL-invoked routines,
    - Extensions to CREATE TABLE statement,
    - A new MERGE statement,
    - A new schema object - the sequence generator,
    - Two new sorts of columns – identity columns and generated columns.
- Retains all data types that existed in SQL: 1999 with the exception of the BIT and BIT VARYING data types.
- Introduces three new data types:
    - BIGINT
    - MULTISET
    - XML
- SQL-invoked function that returns a “table”.
    - Table functions give increased functionality by allowing sets of tuples from any external data sources to be invoked (as if they were a table).
    - Table function execution can be parallelized giving improved speed and scalability.
- In addition to the three statements for updating the database, (INSERT, UPDATE, and DELETE) SQL: 2003 adds MERGE.
- Combining INSERT and UPDATE into MERGE.
- Transferring a set of rows from a “transaction table” to a “master table” maintained by the database.
- Mechanism for generating unique values automatically.
- User can define minimum value, a maximum value, a start value, an increment, and a cycle option for the sequence generator they are creating.

```
CREATE SEQUENCE PARTSEQ AS INTEGER
START WITH 1
INCREMENT BY 1
MINVALUE 1
MAXVALUE 10000
NO CYCLE
```

ODMG 3.0 (Object data management group)

ODMG 3.0 was developed by the Object Data Management Group (ODMG). The ODMG is a consortium of vendors and interested parties that work on specifications for object database and object-relational mapping products.

ODMG 3.0 is a portability specification. It is designed to allow for portable applications that could run on more than one product. ODMG 3.0 uses the Java, C++, and Smalltalk languages as much as possible, to allow for the transparent integration of object programming languages.
The ODMG object model is the data model upon which the object definition language (ODL) and object query language (OQL) are based. In fact, this object model provides the data types, type constructors, and other concepts that can be utilized in the ODL to specify object database schemas. Hence, it is meant to provide a standard data model for object-oriented databases, just as the SQL report describes a standard data model for relational databases. It also provides a standard terminology in a field where the same terms were sometimes used to describe different concepts.
Major components of ODMG 3.0 are described below;

Objects and Literals

Objects and literals are the basic building blocks of the object model. The main difference between the two is that an object has both an object identifier and a state (or current value), whereas a literal has only a value but no object identifier. In either case, the value can have a complex structure. The object state can change over time by modifying the object value. A literal is basically a constant value, possibly having a complex structure that does not change.
An object is described by four characteristics: (1) identifier, (2) name, (3) lifetime, and (4) structure. The object identifier is a unique system-wide identifier (or Object_Id . Every object must have an object identifier. In addition to the Object_Id, some objects may optionally be given a unique name within a particular database—this name can be used to refer to the object in a program, and the system should be able to locate the object given that name. Obviously, not all individual objects will have unique names. Typically, a few objects, mainly those that hold collections of objects of a particular object type—such as extents—will have a name. These names are used as entry points to the database; that is, by locating these objects by their unique name, the user can then locate other objects that are referenced from these objects. Other important objects in the application may also have unique names. All such names within a particular database must be unique.

The lifetime of an object specifies whether it is a persistent object (that is, a database object) or transient object (that is, an object in an executing program that disappears after the program terminates). Finally, the structure of an object specifies how the object is constructed by using the type constructors. The structure specifies whether an object is atomic or a collection object.. The term atomic object is different than the way we defined the atom constructor. It is quite different from an atomic literal.

In the object model, a literal is a value that does not have an object identifier. However, the value may have a simple or complex structure. There are three types of literals: (1) atomic, (2) collection, and (3) structured. Atomic literals correspond to the values of basic data types and are predefined.

The basic data types of the object model include long, short, and unsigned integer numbers (these are specified by the keywords Long, Short, Unsigned Long, Unsigned Short in ODL), regular and double precision floating point numbers (Float, Double), boolean values (Boolean), single characters (Char), character strings (String), and enumeration types (Enum), among others. Structured literals correspond roughly to values that are constructed using the tuple constructor. They include Date, Interval, Time, and Timestamp as built-in structures, as well as any additional user-defined type structures as needed by each application.

User-defined structures are created using the Struct keyword in ODL, as in the C and C++ programming languages. Collection literals specify a value that is a collection of objects or values but the collection itself does not have an Object_Id. The collections in the object model are Set<t>, Bag<t>, List<t>, and Array<t>, where t is the type of objects or values in the collection. Another collection type is Dictionary <k,v>, which is a collection of associations <k,v> where each k is a key (a unique search value) associated with a value v; this can be used to create an index on a collection of values.

Object Definition Language

The concepts of ODMG 2.0 object model can be utilized to create an object database schema using the object definition language ODL. The ODL is designed to support the semantic constructs of the ODMG 2.0 object model and is independent of any particular programming language. Its main use is to create object specifications—that is, classes and interfaces. Hence, ODL is not a full programming language. A user can specify a database schema in ODL independently of any programming language, then use the specific language bindings to specify how ODL constructs can be mapped to constructs in specific programming languages, such as C++, SMALLTALK, and JAVA.

Figure of EER shows one possible set of ODL class definitions for the UNIVERSITY database. In general, there may be several possible mappings from an object schema diagram (or EER schema diagram) into ODL classes.

Figure EER shows the straightforward way of mapping part of the UNIVERSITY database. Entity types are mapped into ODL classes, and inheritance is done using EXTENDS. However, there is no direct way to map categories (union types) or to do multiple inheritance, the classes Person, Faculty, Student, and GradStudent have the extents persons, faculty, students, and grad_students, respectively. Both Faculty and Student EXTENDS Person, and GradStudent EXTENDS Student. Hence, the collection of students (and the collection of faculty) will be constrained to be a subset of the collection of persons at any point in time. Similarly, the collection of grad_students will be a subset of students. At the same time, individual Student and Faculty objects will inherit the properties (attributes and relationships) and operations of Person, and individual GradStudent objects will inherit those of Student.

The classes Department, Course, Section, and CurrSection are straightforward mappings of the corresponding entity types. However, the class Grade requires some explanation. The Grade class corresponds to the M:N relationship between Student and Section. The reason it was made into a separate class (rather than as a pair of inverse relationships) is because it includes the relationship attribute grade. Hence, the M:N relationship is mapped to the class Grade, and a pair of 1:N relationships, one between Student and Grade and the other between Section and Grade. These two relationships are represented by the following relationship properties: completed_sections of Student; section and student of Grade; and students of Section Finally, the class Degree is used to represent the composite, multivalued attribute degrees of GradStudent.

Object Query Language

The object query language (OQL) is the query language proposed for the ODMG object model. It is designed to work closely with the programming languages for which an ODMG binding is defined, such as C++, SMALLTALK, and JAVA. Hence, an OQL query embedded into one of these programming languages can return objects that match the type system of that language. In addition, the implementations of class operations in an ODMG schema can have their code written in these programming languages. The OQL syntax for queries is similar to the syntax of the relational standard query language SQL, with additional features for ODMG concepts, such as object identity, complex objects, operations, inheritance, polymorphism, and relationships.

Simple OQL Queries, Database Entry Points, and Iterator Variables

The basic OQL syntax is a select . . . from . . . where. . Structure, as for SQL. For example, the query to retrieve the names of all departments in the college of ‘Engineering’ can be written as follows:

```
Q1: SELECT d.dname
        FROM d IN Departmant
        WHERE d.college = ”Engineering”;
```

In general, an **entry point** to the database is needed for each query, which can be any *named persistent object.* For many queries, the entry point is the name of the **extent** of a class. Recall that the extent name is considered to be the name of a persistent object whose type is a collection (in most cases, a set) of objects from the class. the named object departments is of type set<Department>; persons is of type set<Person>; faculty is of type set<Faculty>; and so on.

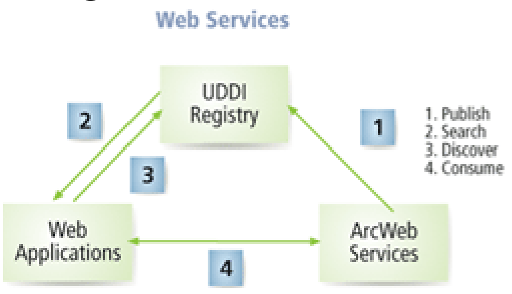
The use of an extent name—departments in Q0—as an entry point refers to a persistent collection of objects. Whenever a collection is referenced in an OQL query, we should define an **iterator variable.** In Q0—that ranges over each object in the collection. In many cases, as in Q0, the query will select certain objects from the collection, based on the conditions specified in the where-clause. In Q0, only persistent objects d in the collection of departments that satisfy the condition d.college = 'Engineering' are selected for the query result. For each selected object d, the value of d.dname is retrieved in the query result. Hence, the *type of the result* for Q0 is bag<string>, because the type of each dname value is string (even though the actual result is a set because dname is a key attribute). In general, the result of a query would be of type *bag* for **select** . . . **from** . . . and of type *set* for **select distinct** . . . **from** . . ., as in SQL (adding the keyword distinct eliminates duplicates).

Query Results and Path Expressions

The result of a query can in general be of any type that can be expressed in the ODMG object model. A query does not have to follow the **select . . . from . . . where** . . . structure; in the simplest case, any persistent name on its own is a query, whose result is a reference to that persistent object.

Web Services—A Standards-Based Framework for Integration

Web services are software components that can be accessed over the Web through standards-based protocols such as HTTP or SMTP for use in other applications. They provide a fundamentally new framework and set of standards for a computing environment that can include servers, workstations, desktop clients, and lightweight "pervasive" clients such as phones and PDAs. Web services are not limited to the Internet; they supply a powerful architecture for all types of distributed computing.



Web services standards are the glue that allows computers and devices to interact. UDDI allows clients to discover Web services.

Web services standards are the glue that allows computers and devices to interact, forming a greater computing whole that can be accessed from any device on the network.

In Web services, computing nodes have three roles—client, service, and broker.

- A client is any computer that accesses functions from one or more other computing nodes on the network. Typical clients include desktop computers, Web browsers, Java applets, and mobile devices. A client process makes a request for a computing service and receives results for that request.

- A service is a computing process that receives and responds to requests and returns a set of results.
- A broker is essentially a service metadata portal for registering and discovering services. Any network client can search the portal for an appropriate service.

Because Web services can support the integration of information and services that are maintained on a distributed network, they are appealing to local governments and other organizations that have departments that independently collect and manage spatial data but must integrate these datasets.

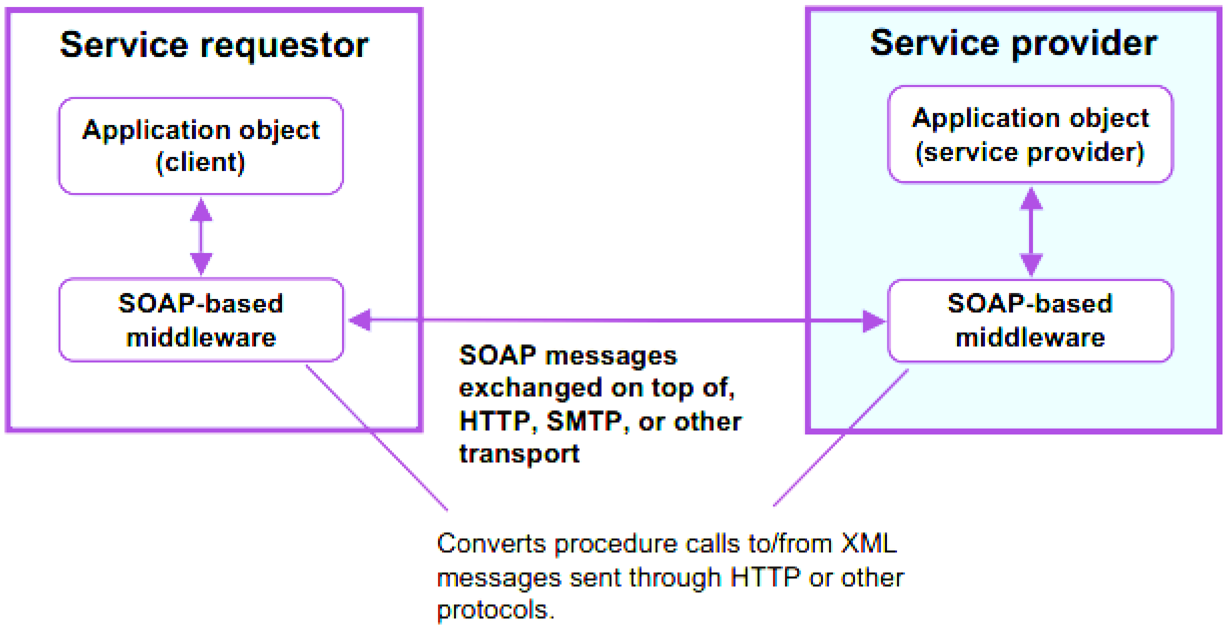
The use of a connecting technology (Web services) coupled with an integrating technology (GIS) can efficiently support this requirement. Various layers of information can be dynamically queried and integrated but will still be maintained independently in a distributed computing environment. Esri's Web services technology, ArcWeb Services, is built on top of ArcIMS. ArcWeb Services leverage core business logic in ArcGIS and support Internet-based distributed computing.

A series of protocols—eXtensible Markup Language (XML); Simple Object Access Protocol (SOAP); Web Service Description Language (WSDL); and Universal Description, Discovery, and Integration (UDDI)—provides the key standards for Web services and supports sophisticated communications between various nodes on a network. These protocols enable smarter communication and collaborative processing among nodes built within any Web services-compliant architecture.

Simple Object Access Protocol (SOAP)

SOAP (originally **Simple Object Access Protocol**) is a protocol specification for exchanging structured information in the implementation of web services in computer networks. Its purpose is to induce extensibility, neutrality and independence. It uses XML Information Set for its message format, and relies on application layer protocols, most often Hypertext Transfer Protocol (HTTP) or Simple Mail Transfer Protocol (SMTP), for message negotiation and transmission.

SOAP allows processes running on disparate operating systems (such as Windows and Linux) to communicate using Extensible Markup Language (XML). Since Web protocols like HTTP are installed and running on all operating systems, SOAP allows clients to invoke web services and receive responses independent of language and platforms.


Service requestor
Application object (client)
SOAP-based middleware
Service provider
Application object (service provider)
SOAP-based middleware
SOAP messages exchanged on top of, HTTP, SMTP, or other transport
Converts procedure calls to/from XML messages sent through HTTP or other protocols.

SOAP covers the following four main areas:

- **A message format** for one-way communication describing how a message can be packed into an XML document.
- **A description** of how a SOAP message should be transported using HTTP (for Web-based interaction) or SMTP (for e-mail-based interaction).
- **A set of rules** that must be followed when processing a SOAP message and a simple classification of the entities involved in processing a SOAP message.
- **A set of conventions** on how to turn an RPC call into a SOAP message and back.

What is SOAP?

- SOAP stands for Simple Object Access Protocol
- SOAP is a communication protocol
- SOAP is for communication between applications
- SOAP is a format for sending messages
- SOAP communicates via Internet
- SOAP is platform independent
- SOAP is language independent
- SOAP is based on XML
- SOAP is simple and extensible
- SOAP allows you to get around firewalls
- SOAP is a W3C recommendation

Why SOAP?

- It is important for application development to allow Internet communication between programs. Today's applications communicate using Remote Procedure Calls (RPC) between objects like DCOM and CORBA, but HTTP was not designed for this. RPC represents a compatibility and security problem; firewalls and proxy servers will normally block this kind of traffic.
- A better way to communicate between applications is over HTTP, because HTTP is supported by all Internet browsers and servers. SOAP was created to accomplish this.
- SOAP provides a way to communicate between applications running on different operating systems, with different technologies and programming languages.

SOAP Building Blocks

A SOAP message is an ordinary XML document containing the following elements:

- An Envelope element that identifies the XML document as a SOAP message
- A Header element that contains header information
- A Body element that contains call and response information
- A Fault element containing errors and status information

Syntax Rules

Here are some important syntax rules:

- A SOAP message MUST be encoded using XML
- A SOAP message MUST use the SOAP Envelope namespace
- A SOAP message MUST use the SOAP Encoding namespace
- A SOAP message must NOT contain a DTD reference
- A SOAP message must NOT contain XML Processing Instructions

The SOAP Envelope Element

The required SOAP Envelope element is the root element of a SOAP message. This element defines the XML document as a SOAP message.

Example

```
<? xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  ...
Message information goes here
```

```
  ...
</soap: Envelope>
```

XML (Extensible Markup Language)

What is XML?

XML stands for Extensible Markup Language. A markup language specifies the structure and content of a document because it is extensible, XML can be used to create a wide variety of document types. XML is a subset of a Standard Generalized Markup Language (SGML) which was introduced in the 1980s. SGML is very complex and can be costly. Hypertext Markup Language (HTML) is a more easily used markup language. XML can be seen as sitting between SGML and HTML – easier to learn than SGML, but more robust than HTML.

Main features of XML:

- XML is a markup language much like HTML
- XML was designed to carry data, not to display data
- XML tags are not predefined. You must define your own tags
- XML is designed to be self-descriptive
- XML files are text files, which can be managed by any text editor.
- XML is very simple, because it has less than 10 syntax rules.
- XML is extensible, because it only specifies the structural rules of tags. No specification on tags them self.

Example of document structure

```
      <bank>
       <account>
      <account-number> A-101     </account-number>
      <branch-name>     Downtown </branch-name>
      <balance>           500       </balance>
       </account>
       <Depositor>
      <account-number> A-101    </account-number>
      <customer-name> Johnson </customer-name>
       </depositor>
    </bank>
```

Comparison of XML with Relational Data

- **Inefficient:** tags, which in effect represent schema information, are repeated
- **Better than relational tuples as a data-exchange format**
  - ✓ Unlike relational tuples, XML data is self-documenting due to presence of tags

    - ✓ Non-rigid format: tags can be added
    - ✓ Allows nested structures
    - ✓ Wide acceptance, not only in database systems, but also in browsers, tools, and applications
- **The data is *self-describing:*** e.g. the meaning of the data is included: identifiers surround every bit of data, indicating what it means
- **Far more *flexible* method of representing transmitted information**
- **Open, standard technologies for moving, processing and validating the data:** e.g. the XML parser can automatically parse, validate, and feed the information to an application, instead of every application having to include this functionality

XML Declaration

The XML document can optionally have an XML declaration. It is written as follows:

<? xml version="1.0" encoding="UTF-8"?>

Where version is the XML version and encoding specifies the character encoding used in the document.

Syntax Rules for XML Declaration

- The XML declaration is case sensitive and must begin with "<? xml>" where "xml" is written in lower-case.
- If the document contains XML declaration, then it strictly needs to be the first statement of the XML document.
- The XML declaration strictly needs be the first statement in the XML document.
- An HTTP protocol can override the value of encoding that you put in the XML declaration.

What is an XML Element?

An XML element is everything from (including) the element's start tag to (including) the element's end tag.

```
<price>29.99</price>
```

An element can contain:

- text
- attributes
- other elements
- or a mix of the above

Example:

```
<bookstore>
  <book category="children">
    <title>Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="web">
    <title>Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

In the example above:

<title>, <author>, <year>, and <price> have text content because they contain text (like 29.99).
<bookstore> and <book> have element contents, because they contain elements.
<book> has an attribute (category="children").

Empty XML Elements

An element with no content is said to be empty.
In XML, you can indicate an empty element like this:

```
<element></element>
```

You can also use a so called self-closing tag:

```
<element />
```

Tags and Elements

An XML file is structured by several XML-elements, also called XML-nodes or XML-tags. The names of XML-elements are enclosed in triangular brackets < > as shown below:

```
        <element>
```

Syntax Rules for Tags and Elements

Element Syntax: Each XML-element needs to be closed either with start or with end elements as shown below:

```
<element>....</element>
```

or in simple-cases, just this way:

```
<element/>
```

XML Attributes

An attribute specifies a single property for the element, using a name/value pair. An XML- element can have one or more attributes. For example:

```
<a href="http://www.tutorialspoint.com/">Tutorialspoint!</a>
```

Here href is the attribute name and http://www.tutorialspoint.com/ is attribute value.

Syntax Rules for XML Attributes

- Attribute names in XML (unlike HTML) are case sensitive. That is, HREF and href are considered two different XML attributes.
- Same attribute cannot have two values in a syntax. The following example shows incorrect syntax because the attribute b is specified twice:

  ```
  <a b="x" c="y" b="z">....</a>
  ```
- Attribute names are defined without quotation marks, whereas attribute values must always appear in quotation marks. Following example demonstrates incorrect xml syntax:

  ```
  <a b=x>....</a>
  ```

In the above syntax, the attribute value is not defined in quotation marks.

CDATA

The term CDATA means, Character Data. CDATA is defined as blocks of text that are not parsed by the parser, but are otherwise recognized as markup.

The predefined entities such as &lt;, &gt;, and &amp; require typing and are generally difficult to read in the markup. In such cases, CDATA section can be used. By using CDATA section, you are commanding the parser that the particular section of the document contains no markup and should be treated as regular text.

Syntax

Following is the syntax for CDATA section:

```
<![CDATA[
        characters with markup
]]>
```

The above syntax is composed of three sections:

- **CDATA Start section:** CDATA begins with the nine-character delimiter <![CDATA[
- **CDATA End section:** CDATA section ends with]]> delimiter.
- **CData section:** Characters between these two enclosures are interpreted as characters, and not as markup. This section may contain markup characters (<, >, and &), but they are ignored by the XML processor.

Example

The following markup code shows an example of CDATA. Here, each character written inside the CDATA section is ignored by the parser.

```
<script>
<![CDATA[
        <message> Welcome to NSC </message>
]] >
</script >
```

In the above syntax, everything between <message> and </message> is treated as character data and not as markup

CDATA Rules

The given rules are required to be followed for XML CDATA:

- CDATA cannot contain the string "]]>" anywhere in the XML document.
- Nesting is not allowed in CDATA section

XML Encoding

Encoding is the process of converting Unicode characters into their equivalent binary representation. When the XML processor reads an XML document, it encodes the document depending on the type of encoding. Hence, we need to specify the type of encoding in the XML declaration.

Encoding Types

There are mainly two types of encoding:

- UTF-8
- UTF-16

UTF stands for UCS Transformation Format, and UCS itself means Universal Character Set. The number 8 or 16 refers to the number of bits used to represent a character. They are either 8 (one byte) or 16 (two bytes). For the documents without encoding information, UTF-8 is set by default.

Syntax

Encoding type is included in the prolog section of the XML document. The syntax for UTF-8 encoding is as follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
```

The syntax for UTF-16 encoding is as follows:

```
<?xml version="1.0" encoding="UTF-16" standalone="no" ?>
```

Example

Following example shows the declaration of encoding:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<contact-info>
        <name>College</name>
        <company>NSC</company>
        <phone>01-456543</phone>
</contact-info>
```

In the above example encoding="UTF-8", specifies that 8-bits are used to represent the characters. To represent 16-bit characters, UTF-16 encoding can be used. The XML files encoded with UTF-8 tend to be smaller in size than those encoded with UTF-
16 format

Nesting of element

Nesting of data is useful in data transfer

Example: elements representing customer-id, customer name, and address nested within an order element

Nesting is not supported, or discouraged, in relational databases

With multiple orders, customer name and address are stored redundantly

normalization replaces nested structures in each order by foreign key into table storing customer name and address information

Nesting is supported in object-relational databases

But nesting is appropriate when transferring data

External application does not have direct access to data referenced by a foreign key

Example of Nested Elements

```
<bank-1>
    <customer>
            <customer-name> Hayes </customer-name>
            <customer-street> Main </customer-street>
            <customer-city>    Harrison </customer-city>
            <account>
           <account-number> A-102 </account-number>
           <branch-name>     Perryridge </branch-name>
           <balance>             400 </balance>
            </account>
            ………….
            …………..
               </customer>
      </bank-1>
```

XML Comment

XML comments are similar to HTML comments. The comments are added as notes or lines for understanding the purpose of an XML code.

Comments can be used to include related links, information, and terms. They are visible only in the source code; not in the XML code. Comments may appear anywhere in XML code.

Syntax

XML comment has the following syntax:

```
        <!-------Your comment----->
```

A comment starts with <!-- and ends with -->. You can add textual notes as comments between the characters. You must not nest one comment inside the other.

Example: Following example demonstrates the use of comments in XML document:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!---Students grades are uploaded by months---->
<class_list>
        <student>
                <name>Tanmay</name>
                <grade>A</grade>
        </student>
</class_list>
```

Any text between <!-- and --> characters is considered as a comment.

XML Document

An XML document is a basic unit of XML information composed of elements and other markup in an orderly package. An XML document can contain a wide variety of data. For example, database of numbers, numbers representing molecular structure or a mathematical equation.

XML Document Example

A simple document is shown in the following example:

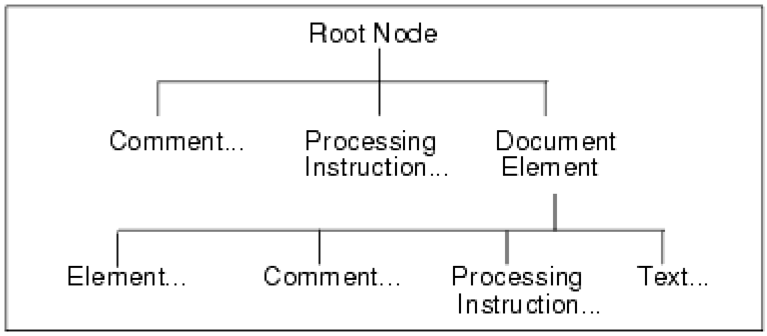
```
<?xml version="1.0"?>
<contact-info>
        <name>NSC</name>
        <company>Education_company</company>
        <phone>01-449839</phone>
```

</contact-info>

The following image depicts the parts of XML document.

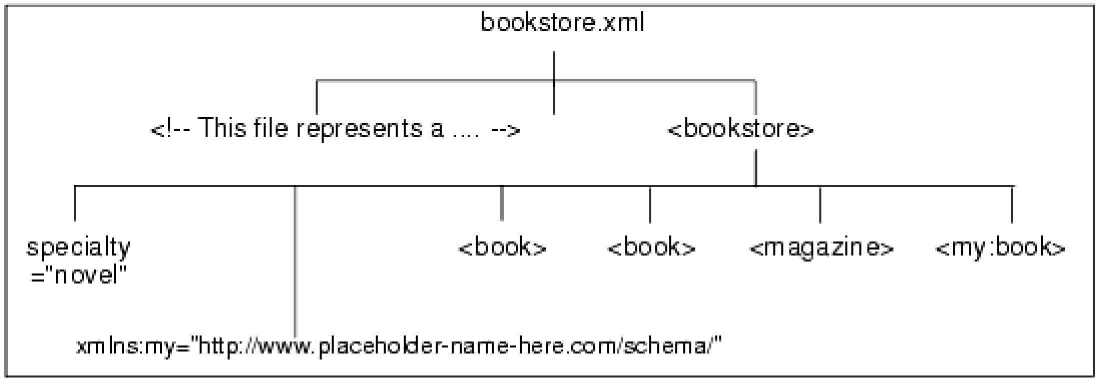
XML Document Structure:

The XPath processor operates on a tree representation of XML data that looks like the following figure:



The root node has no actual text associated with it. You can think of the file name as the root node. A document can include zero or more comments and zero or more processing instructions.

A document element is required, and there can be only one. The document element contains all elements in the document. For example:


bookstore.xml
<!-- This file represents a .... -->
<bookstore>
specialty ="novel"
<book>
<book>
<magazine>
<my:book>
xmlns:my="http://www.placeholder-name-here.com/schema/"

In the preceding figure, bookstore.xml is the name of a file that contains XML data. There is a comment near the beginning of the document that starts with "This file represents a ..." The document element is bookstore. The immediate children of bookstore include an attribute, a namespace declaration (not supported by Stylus Studio), three book elements (one is in the my namespace), and a magazine element. The book and magazine elements contain elements and attributes

An XML Document

A complete well-formed XML document may look like:

<?xml version="1.0"?>

```
 <bank>
              <account>
             <account-number> A-101     </account-number>
             <branch-name>      Downtown </branch-name>
             <balance>             500        </balance>
              </account>
              <Depositor>
             <account-number> A-101    </account-number>
             <customer-name> Johnson </customer-name>
              </depositor>
        </bank>
```
The Bank element, above, is the root element.

XML Document Type Declaration (DTDs)

The XML Document Type Declaration, commonly known as DTD, is a way to describe XML language precisely. DTDs check vocabulary and validity of the structure of XML documents against grammatical rules of appropriate XML language.

An XML DTD can be either specified inside the document, or it can be kept in a separate document and then liked separately.

Syntax

Basic syntax of a DTD is as follows:

```
<! DOCTYPE element DTD identifier
[
   declaration1
   declaration2
   ........
]>
```
In the above syntax,

- The DTD starts with <! DOCTYPE delimiter.
- An element tells the parser to parse the document from the specified root element.
- DTD identifier is an identifier for the document type definition, which may be the path to a file on the system or URL to a file on the internet. If the DTD is pointing to external path, it is called External Subset.
- The square brackets [ ] enclose an optional list of entity declarations called Internal Subset.

Internal DTD

A DTD is referred to as an internal DTD if elements are declared within the XML files. To refer it as internal DTD, standalone attribute in XML declaration must be set to yes. This means, the declaration works independent of external source.

Syntax

The syntax of internal DTD is as shown:

```
        <! DOCTYPE root-element [element-declarations]>
```

Where root-element is the name of root element and element-declarations is where you declare the elements.

Example

Following is a simple example of internal DTD:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
        <!DOCTYPE address [
                <!ELEMENT address (name,company,phone)>
                <!ELEMENT name (#PCDATA)>
                <!ELEMENT company (#PCDATA)>
                <!ELEMENT phone (#PCDATA)>
        ]>
        <address>
                <name>Tanmay Patil</name>
                <company>NDC</company>
                <phone>0146575655</phone>
        </address>
```

Let us go through the above code:

Start Declaration- Begin the XML declaration with following statement

```
        <?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
```

DTD- Immediately after the XML header, the document type declaration follows, commonly referred to as the DOCTYPE:

```
        <!DOCTYPE address [
```

The DOCTYPE declaration has an exclamation mark (!) at the start of the element name. The DOCTYPE informs the parser that a DTD is associated with this XML document.

DTD Body- The DOCTYPE declaration is followed by body of the DTD, where you declare elements, attributes, entities, and notations:

```
<!ELEMENT address (name,company,phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
```

<!ELEMENT phone_no (#PCDATA)>

Several elements are declared here that make up the vocabulary of the <name> document. <!ELEMENT name (#PCDATA)> defines the element *name* to be of type "#PCDATA". Here #PCDATA means parse-able text data.

End Declaration - Finally, the declaration section of the DTD is closed using a closing bracket and a closing angle bracket (]>). This effectively ends the definition, and thereafter, the XML document follows immediately.

Rules

- The document type declaration must appear at the start of the document (preceded only by the XML header) — it is not permitted anywhere else within the document.
- Similar to the DOCTYPE declaration, the element declarations must start with an exclamation mark.
- The Name in the document type declaration must match the element type of the root element.

External DTD

In external DTD elements are declared outside the XML file. They are accessed by specifying the system attributes which may be either the legal .dtd file or a valid URL. To refer it as external DTD, standalone attribute in the XML declaration must be set as no. This means, declaration includes information from the external source.

Syntax

Following is the syntax for external DTD:

```
<!DOCTYPE root-element SYSTEM "file-name">
```

where file-name is the file with .dtd extension.

Example

The following example shows external DTD usage:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE address SYSTEM "address.dtd">
<address>
    <name>Tanmay Patil</name>
    <company>NSC</company>
    <phone>0167655565</phone>
</address>
```

The content of the DTD file address.dtd are as shown:

```
<!ELEMENT address (name,company,phone)>
<!ELEMENT name (#PCDATA)>
```

```
<!ELEMENT company (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
```

XML Schema

XML Schema is commonly known as XML Schema Definition (XSD). It is used to describe and validate the structure and the content of XML data. XML schema defines the elements, attributes and data types. Schema element supports Namespaces. It is similar to a database schema that describes the data in a database.

Syntax

You need to declare a schema in your XML document as follows:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

Example

The following example shows how to use schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="contact">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="name" type="xs:string" />
         <xs:element name="company" type="xs:string" />
         <xs:element name="phone" type="xs:int" />
      </xs:sequence>
   </xs:complexType>
</xs:element>
</xs:schema>
```

The basic idea behind XML Schemas is that they describe the legitimate format that an XML document can take.

Elements

As we saw in the XML - Elements chapter, elements are the building blocks of XML document. An element can be defined within an XSD as follows:

```
         <xs:element name="x" type="y"/>
```

Definition Types

You can define XML schema elements in following ways:

Simple Type - Simple type element is used only in the context of the text. Some of predefined simple types are: xs:integer, xs:boolean, xs:string, xs:date. For example:

```
<xs:element name="phone_number" type="xs:int" />
```

Complex Type - A complex type is a container for other element definitions. This allows you to specify which child elements an element can contain and to provide some structure within your XML documents. For example:

```
<xs:element name="Address">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="name" type="xs:string" />
            <xs:element name="company" type="xs:string" />
         <xs:element name="phone" type="xs:int" />
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

In the above example, *Address* element consists of child elements. This is a container for other <xs:element> definitions, that allows to build a simple hierarchy of elements in the XML document.

Global Types - With global type, you can define a single type in your document, which can be used by all other references. For example, suppose you want to generalize the *person* and *company* for different addresses of the company. In such case, you can define a general type as below:

```
<xs:element name="AddressType">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="name" type="xs:string" />
            <xs:element name="company" type="xs:string" />
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

Now let us use this type in our example as below:

```
<xs:element name="Address1">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="address" type="AddressType" />
            <xs:element name="phone1" type="xs:int" />
```

```
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="Address2">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="address" type="AddressType" />
                <xs:element name="phone2" type="xs:int" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Instead of having to define the name and the company twice (once for *Address1* and once for *Address2*), we now have a single definition. This makes maintenance simpler, i.e., if you decide to add "Postcode" elements to the address, you need to add them at just one place.

X-Query

XQuery is a standardized language for combining documents, databases, Web pages and almost anything else. It is very widely implemented. It is powerful and easy to learn. XQuery is replacing proprietary middleware languages and Web Application development languages. XQuery is replacing complex Java or C++ programs with a few lines of code. XQuery is simpler to work with and easier to maintain than many other alternatives.

Characteristics

- XQuery is the language for querying XML data
- XQuery for XML is like SQL for databases
- XQuery is built on XPath expressions
- XQuery is supported by all major databases

Benefits of XQuery

- Using XQuery, both hierarchical and tabular data can be retrieved.
- XQuery can be used to query tree and graphical structures.
- XQuery can be directly used to query webpages.
- XQuery can be directly used to build webpages.
- XQuery can be used to transform xml documents.
- XQuery is ideal for XML-based databases and object-based databases. Object databases are much more flexible and powerful than purely tabular databases.

Example

Following is a sample XML document containing the records of a bookstore of various books.

books.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<books>
    <book category="JAVA">
    <title lang="en">Learn Java in 24 Hours</title>
    <author>Robert</author>
    <year>2005</year>
    <price>30.00</price>
  </book>

 <book category="DOTNET">
    <title lang="en">Learn .Net in 24 hours</title>
    <author>Peter</author>
    <year>2011</year>
    <price>70.50</price>
  </book>

  <book category="XML">
    <title lang="en">Learn XQuery in 24 hours</title>
    <author>Robert</author>
    <author>Peter</author>
    <year>2013</year>
    <price>50.00</price>
  </book>

  <book category="XML">
    <title lang="en">Learn XPath in 24 hours</title>
    <author>Jay Ban</author>
    <year>2010</year>
    <price>16.50</price>
  </book>
</books>
```

Following is a sample XQuery document containing the query expression to be executed on the above XML document. The purpose is to get the title elements of those XML nodes where the price is greater than 30.

books.xqy

```
for $x in doc("books.xml")/books/book
where $x/price>30
return $x/title
```

Result

```
<title lang="en">Learn .Net in 24 hours</title>
<title lang="en">Learn XQuery in 24 hours</title>
```

XSLT

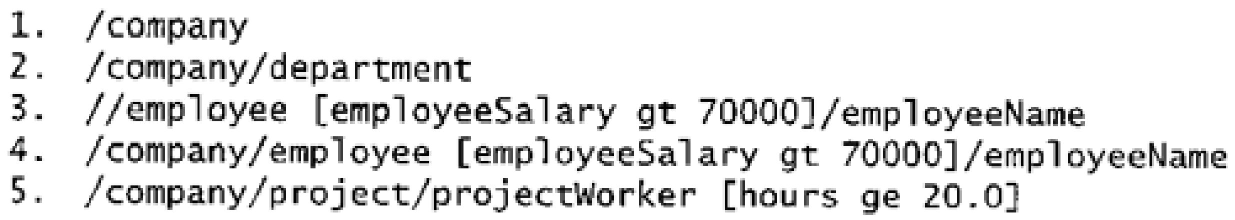
XSLT (Extensible Style sheet Language Transformations) is a language for transforming XML documents into other XML documents, or other formats such as HTML for web pages, plain text or XSL Formatting Objects, which may subsequently be converted to other formats, such as PDF, PostScript and PNG.

X-path

XPath is a major element in the XSLT standard. **XPath** can be used to navigate through elements and attributes in an XML document. **XPath** is a syntax for defining parts of an XML document. **XPath** uses **path** expressions to navigate in XML documents. **XPath** contains a library of standard functions.

- An XPath expression returns a collection of element nodes that satisfy certain patterns specified in the expression.
- The names in the XPath expression are node names in the XML document tree that are either tag (element) names or attribute names, possibly with additional **qualifier conditions** to further restrict the nodes that satisfy the pattern.
- There are two main **separators** when specifying a path: **single slash (/) and double slash (//).**
- A single slash before a tag specifies that the tag must appear as a direct child of the previous (parent) tag, whereas a double slash specifies that the tag can appear as a descendant of the previous tag *at any level.*

Some examples are given below:



```
1.  /company
2.  /company/department
3.  //employee [employeeSalary gt 70000]/employeeName
4.  /company/employee [employeeSalary gt 70000]/employeeName
5.  /company/project/projectWorker [hours ge 20.0]
```

XML Applications

- **Storing and exchanging data with complex structures**
    - E.g. Open Document Format (ODF) format standard for storing Open Office and Office Open XML (OOXML) format standard for storing Microsoft Office documents
    - Numerous other standards for a variety of applications
        - ChemML, MathML
- **Standard for data exchange for Web services**
    - Remote method invocation over HTTP protocol
    - More in next slide
- **Data mediation**
    - Common data representation format to bridge different systems