MODULE - 1

INTRODUCTION TO DATABASE SYSTEMS & E-R MODEL

1.1 Overview

The amount of information available and to use is literally exploding, and in today's world the value of data as an organizational asset is widely recognized. To get most out of the large and complex datasets in any enterprise, users require tools that simplify the task of managing the data and extracting useful information in a timely fashion, otherwise the cost of acquiring and managing the data exceeds the value derived from it.

What is a database (DB)?

- ➔ A collection of data that exists over a long period of time, often many years or is the collection of interrelated data, describing the activities of one or more related organizations.
- ➔ Managed through a database management system

What is a database management system (DBMS)?

- ✓ DBMS or simply known as `database Management system'
- ✓ It refers to a set of programs to access the data
- ✓ A powerful tool for creating and managing [and manipulating] large amounts of data efficiently-and allowing it to-persist-over long periods of time,-safely-focus on secondary, rather than main, memory
- ✓ Powerful, but simple, programming interface, DBMS provides an environment that is both convenient and efficient to use.
- ✓ Database Applications:
    - **Banking:** all transactions
    - **Airlines:** reservations, schedules
    - **Universities:** faculty, students, registration, admission, grades etc.
    - **Sales:** customers, products, purchases
    - **Manufacturing:** production, inventory, orders, supply chain
    - **Human resources:** employee records, salaries, tax deductions

  Databases touch all aspects of our lives

Why do we need database systems?

- ➢ To solve the data management problem
- ➢ To bridge the gap and the analogies in the personal information space
- ➢ DBMS contains information about a particular enterprise.

1.1.1 Managing data with DBMS

The various parameters considered while managing data are:

- ➔ **Database Design and Application Development:** It describe the various issues like-
  - How can a user describe a real-world enterprise (e.g.: university) in terms of the data stored in a DBMS?
  - What the factors to be considered in deciding how to organize the stored data.
  - The methods to develop applications that relay upon a DBMS.
- ➔ **Data Analysis**
  - How can a user answer to the questions about the enterprise by posing queries over the data in the DBMS?
- ➔ **Concurrency and Robustness**
  - How does the DBMS allow many users to access data concurrently?
  - How does it protect the data in the event of system failures?
- ➔ **Efficiency and Scalability**
  - How does a DBMS store large datasets and answer questions against this data efficiently?

1.2 Historical Perspective

- In 1960s the first **general purpose DBMS**, designed by **Charles Bachman** has `information-centric' ideas and with others developed was called IDS (Integrated Data Store), which formed the basis for the network DBMS called **Network data model**, and was standardized by the conference on data system languages (CODASYL). Bachman was the first recipient of ACMs Turning Award in 1973.
- In the late 1960s, IBM developed the **Information management System (IMS) DBMS**, which were used widely to access data over network. This IMS formed the basis of an alternative data representation framework called the Hierarchical data model.
- In 1970s Edgar F. Codd proposed a new data representation frame work, at IBMs San Jose research lab called **Relational model of data**, for which Codd won the Turning Award in 1981. After this the use of DBMS matured as an academic discipline, and rise of relational DBMS emerged popularity for the use of DBMS for managing corporate data as a standard practice.
- **1971:** two camps, with differing viewpoints, formed
  - ❖ Bachman's COBOL/ CODASYL (Committee on Data Systems and Languages) camp championing network model
    - DBTG (Database Task Group)
    - espoused record-oriented queries
    - said relational camp is too theoretical
    - no efficiency in relational approach
  - ❖ relational, theoretical ideas
    - espoused set-oriented queries
    - said COBOL/CODASYL camp is not theoretical
- 1975: ACM SIGMOD panel discussion (`The Great Debate'): Codd vs. Bachman
- **Late 1970s:**
  - ❖ Two camps understand each other better

- More usable relational languages
    - QUEL (1975)
    - **SQL (1976)**
- Relational prototypes, with reasonable efficiency, built
    - System R
    - INGRES
- Went commercial
    - Evolved into ESVAL (formed by Eswaran) and then into
        - HP's ALLBASE
        - Cullinet's IDMS/ SQL; Britton-Lee (which was part of NCR) helped to build the IDM software
- Later independently became IBM's DB2 and SQL/DS
- Eventually become Oracle

- **In the 1980s,** the relational model consolidated its position as a dominant DBMS paradigm and database systems gained to get a widespread use.
    - Development of SQL queries language by IBMs System R project, as standard query language.
    - SQL was standardized in late 1980s and the current standard is SQL: 1999 and adopted by the American National Standards Institute (ANSI) and International Organization for Standardization (ISO).
- Development of concurrent execution of database programs called Transactions; James Gray won the Turning Award for contributions to database transaction management in 1999.
- **In late 1980s and 1990s,** advances were made in many areas of database systems, with research into more powerful query languages and richer data models that emphasized on supporting complex analysis of data from all parts of the enterprise. The various vendors like :
    - IBMs DB2
    - **Oracle 8**
    - Informix UDS etc with their ability to process and store new data types such as images and text and to store more complex queries.
- Specialized systems were developed by numerous vendors for creating data warehouses, consolidating data from several databases, and carrying out specialized analysis.
- Emergence of several enterprise resource planning (ERP) and Management resource planning (MRP) packages, that add substantial layer of application oriented features on the top of DBMS. The widely used packages are Baan, Oracle, PeopleSoft, SAP, and Seibel. These packages identify a set of common tasks such as Inventory management; human resources planning, financial analysis used by large number of organizations and the data stored are customized.

1.3 Files System versus DBMS

In the early days, database applications were built on top of file systems. Considering a large collection of data (for e.g:500 GB3) on employees, departments, products, sales, and so on. This data needs to accessed concurrently by several employees, and the questions about the data need to be answered quickly, changes made to the data by different users must be applies consistently, and access to certain part of the data (such as salaries) needs to be restricted.

Drawbacks of using file systems to store data:

- 500 GB of main memory to hold the data is probably not available, and therefore needs storage device such as disk or tape and needs to bring relevant parts into main memory for processing as needed.
- Even if the computer possess 500GB main memory, computer systems with 32-bit addressing, cannot refer directly to more than about 4 GB of data
- To answer to each question by the user about the data, specials programs needs to be developed which could be complex because large volume needs to be searched resulting in :
  - ✓ **Data redundancy and inconsistency**
    Multiple file formats, duplication of information in different files
  - ✓ **Difficulty in accessing data**
    Need to write a new program to carry out each new task. i.e., application specific program has to be generated.
  - ✓ **Data isolation** multiple files and formats
  - ✓ **Integrity problems**
    - ❖ Integrity constraints (e.g. account balance > 0) become part of program code
    - ❖ It's Hard to add new constraints or change existing
- The problem to protect the data from inconsistent changes made by different users accessing the data concurrently.
  - ❖ **Concurrent access by multiple users**
    - ⅄ Concurrent accessed needed for performance
    - ⅄ Uncontrolled concurrent accesses can lead to inconsistencies
      -E.g. two people reading a balance and updating it at the same time
  - ❖ **Security problems-** The operating systems provide only password mechanism for security, but which is not sufficient to enforce security policies in which different users have permission to access subsets of data.
- Needs to ensure that the data is restored to a consistent state if the file crashes while the changes are made. i.e.
  - ❖ **Atomicity of updates**
    Failures may leave the data in an inconsistent state with partial updates carried out
    E.g. transfer of funds from one account to another should either complete or not happen at all

Database systems offer solutions to all the above problems. DBMS is software designed to make these problems easier way, where the DBMSs features to manage the data in a robust and efficient manner.

1.4 Advantages of DBMS

Using a DBMS to manage data has many advantages:

- **Data Independence:** The application programs should not be exposed to details of data representation and storage. The DBMS should provide an abstract view of the data that hides such details.
- **Efficient data Access:** A DBMS utilizes a variety of sophisticated techniques to store and retrieve data efficiently.
- **Data Integrity and security:** If the data is always accessed through the DBMS, the DBMS can enforce integrity constraints. For example, before inserting salary information for an employee, the DBMS can check the required integrity constraint that the budget is not exceeded. It also can enforce access controls that governs that the data is visible to different classes to users.
- **Data Administration:** When several users use the data, it is required to centralize the administration of data, as it is only the experienced professional who understand the nature of data being managed, and how different users use it, is also responsible to organize the data to avoid redundancy and the fine tuning of the storage and efficient retrieval.
- **Concurrent Access and Crash recovery:** A DBMS schedules concurrent access to the data, where the user considers the data being accessed by only one user at a time. And also the DBMS protects the users from the effects of system failures.
- **Reduced Application Development Time:** Since the DBMS has many functions in common to many applications accessing the data in the database, this facilitates quick application development.

1.5 Describing and storing data in a DBMS

The user of a DBMS is concerned with some real world enterprise, and the data to be stored describes various aspects of this enterprise. E.g.: Students, faculty and courses in a university and the data in a university database describes these entities and their relationships.

1.5.1 Data Models

A Data model is a collection of tools for describing the factors such as,

- Data
- Data relationships
- Data semantics
- Data constraints

The data model is also described as a collection of high level data description constructs that hide low level storage details. i.e., a DBMS allows a user to define the data to be stored in terms of a data model.

The various types of models are:

- Entity-Relationship model
- Relational model (E.g.: IBMs DB2,Informix,Oracle,Sybase,Microsoft Access, Foxbase)
- Other models:
  - Object-oriented model (E.g. :Object Store & Versant)
  - Object relational model (E.g. :Object Store , Versant, Informix, Oracle)
  - Semi-structured data models

- Older models:
  - **Network model** (E.g.: IDS & IDMS)
  - **Hierarchical model** (E.g:IBMs IMS DBMS)

1.5.1.1 Entity-Relationship Model

Entity-Relationship (ER-Model) is a semantic data model, which allows making pictorial representation of entities and relationship among them.

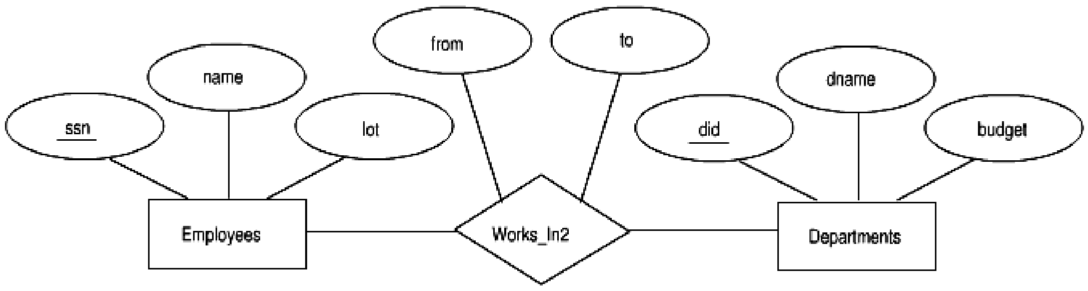


Figure: 1.0 An ER -model

- E-R model of real world
  - **Entities (objects)**

    E.g. Employees, Departments
  - **Relationships between entities**

    E.g. Employee with Emp-id, 'ssn' works_in Department with Did D2.

    Relationship set Works_In2 associates Employees with Departments
- Widely used for database design

  Database design in E-R model is usually converted to design in the relational model which is used for storage and processing

1.5.1.2 The Relational Model

The central data description construct in a relational model is a 'relation', which can be described as a **set of records**. A description of data in terms of a data model is called a **schema**. In the relational model, the scheme for a relation specifies its name, the name of each field) or attribute or column), and the type of each field. i.e., Schema is the **logical structure** of the database.

e.g., the database consists of information about a set of customers and accounts and the relationship between them).

As another example, Consider student information in a university database may be stored in a relation with the following schema:

Students(*sid:* string, *name:* string, *login:* string, *age:* integer, *gpa:* real)

This schema explains that each record in the Students relation has five fields, with field names and types as indicated in the instance of the Students relation given below Figure: 1.1

Analogous to type information of a variable in a program, the schema is classified as,

- ✓ **Physical schema:** database design at the physical level
- ✓ **Logical schema:** database design at the logical level

sid	*name*	*login*	*age*	*gpa*
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2
53650	Smith	smith@math	19	3.8
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	2.0

Figure: 1.1 An instance of the Students Relation

In the above mentioned Instance, each row in the student's relation is a record that describes a student. The description of a collection of student records becomes more precise by specifying the **Integrity constraints**. The **Instance** of a relation is the actual content of the database at a particular point in time, analogous to the value of a variable. Physical Data Independence refers to the ability to modify the physical schema without changing the logical schema.

- Applications depend on the logical schema
- In general, the interfaces between the various levels and components should be well defined so that changes in some parts do not seriously influence others.

1.5.2 Levels of Abstraction in a DBMS

The data in a DBMS is described at three levels of abstraction as in the Figure: 1.2.The database description consists of a schema at each of these three levels of abstraction:

➔ **Conceptual Schema**
➔ **Physical Schema**
➔ **External Schema**

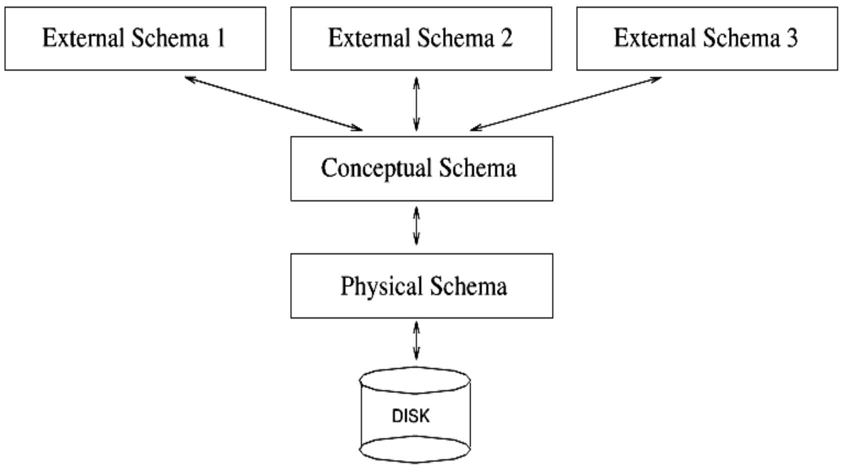


Figure: 1.2 Levels of Abstraction in a DBMS

Data Definition Language (DDL) - The is primarily used to define the external and conceptual schemas. The most widely used DDL database language is SQL. The information about the conceptual, external and physical schemas is stored in system catalogs.

Description:

- DDL is the specification notation for defining the database schema
    - E.g.
        - create table account (
        - account-number   char(10),
        - balance   integer)
- DDL compiler generates a set of tables stored in a data dictionary
- Data dictionary contains metadata (i.e., data about data)
    - ✓ Database schema
    - ✓ Data storage and definition language
        - Language in which the storage structure and access methods used by the database system are specified
        - Usually an extension of the data definition language.

Data Manipulation Language (DML)

- Language for accessing and manipulating the data organized by the appropriate data model, Therefore, DML also known as query language
- The Two classes of DML languages are:
    - **Procedural** user specifies what data is required and how to get those data
    - **Non-Procedural** user specifies what data is required without specifying how to get those data
- **SQL** is the most widely used query language

SQL

- SQL: widely used non-procedural language

  E.g. find the name of the customer with customer-id 192-83-7465

```
select   customer.customer-name
    from     customer
    where  customer.customer-id = 192-83-7465
```

E.g. find the balances of all accounts held by the customer with customer-id 192-83-7465

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

- Application programs generally access databases through one of the,
    - Language extensions to allow embedded SQL
    - Application program interface (e.g. ODBC/JDBC) which allow SQL queries to be sent to a database

1.5.2.1 Conceptual Schema

The Conceptual schema is also called Logical schema that describes the stored data in terms of the data model of the DBMS. In a relational DBMS, the conceptual schema describes all the relations that are stored in the database. In the sample university database, these relations contain information about entities, such as students and faculty, and about relationships, such as students enrollment in courses. All student entities can be described using records in a student relation. The Following given collection of entities and relationships can be used to describe the following Conceptual schema.

> **Students (*sid:* string, *name:* string, *login:* string,**
> ***age:* integer, *gpa:* real)**
> **Faculty (*fid:* string, *fname:* string, *sal:* real)**
> **Courses (*cid:* string, *cname:* string, *credits:* integer)**
> **Rooms (*rno:* integer, *address:* string, *capacity:* integer)**
> **Enrolled (*sid:* string, *cid:* string, *grade:* string)**
> **Teaches (*fid:* string, *cid:* string)**
> **Meets In (*cid:* string, *rno:* integer, *time:* string)**

The process of arriving at a good conceptual schema is called Conceptual database design.

1.5.2.2 Physical Schema

The physical schema specifies additional storage details, the physical schema summarizes how the relations described in the conceptual schema are actually stored on secondary storage devices such as disk and tapes. The decision regarding the file organization to be used to store the relations and the creation of the auxiliary storage data structures, called indexes, so that data retrieval operations can be speeded up. The sample physical schema for the university database is as follows:

- Store all the relations as unsorted file of records.
- Create indexes on the first column of the students, faculty, and courses relations, the sal column of the faculty, and the capacity column of rooms.

The various decisions about the physical scheme are based on the understanding of how the data that is stored is accessed. The process of arriving at a good physical schema is called **physical database design.**

1.5.2.3 External Schema

External schemas are based on the data model of the DBMS, which allows data access to be customized at the level of individual users or groups of users. Any database has exactly one Conceptual scheme, one Physical schema, but has several external schemas, based on the group of users. Each external schema consists of a collection of one or more views and relations from the conceptual schema. A view is conceptually a relation, but the records in the view are not stored in the DBMS, but they are computed using definitions for the view, in terms of relations stored in the DBMS.

The External schema is basically designed by the end user requirements. For example, to compute names of the faculty members teaching courses as well as course enrollments can be done by defining the following view:
Courseinfo(*cid:* string, *fname:* string, *enrollment:* integer)

Views can provide advantages over tables:

- Views can represent a subset of the data contained in a table

- Views can join and simplify multiple tables into a single virtual table
- Views can act as aggregated tables, where the database engine aggregates data (sum, average etc) and presents the calculated results as part of the data
- Views can hide the complexity of data; for example a view could appear as Sales2000 or Sales2001, transparently partitioning the actual underlying table
- Views take very little space to store; the database contains only the definition of a view, not a copy of all the data it presents
- Depending on the SQL engine used, views can provide extra security
- Views can limit the degree of exposure of a table or tables to the outer world

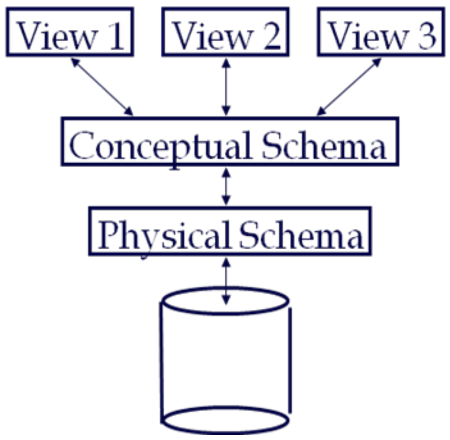


Figure: 1.3 Views of data.

The view also can be considered as a relation and can be queried about the records in the view, even though the record in the view is not explicitly stored, but is computed as when required. The Courseinfo is not included in the conceptual schema because we can compute Courseinfo from the relations in the conceptual schema, and storing it in addition would create redundancy, leading to wastage of space and inconsistencies in the database. For example: a tuple when inserted into the Enrolled relation, indicating the enrolment of a particular student for some course, without incrementing the value in the enrollment field of the corresponding record of Courseinfo.

1.5.3 Data Independence

An important advantage of using DBMS is that it offers data independence; it is achieved through use of three levels of data abstraction, the conceptual schema, Physical schema, and the external schema.

The relations in the external schema are the view relations generated or computed on demand from the relations corresponding to the conceptual schema, and if the this data is reorganized, the conceptual schema changes, the definition of the view relation can be modified so as to compute the same relation as before stored. For example: suppose the Faculty relation in the university database is replaced by the following two relations:

Faculty public(*fid:* string, *fname:* string, *office:* integer)
Faculty private(*fid:* string, *sal:* real)

Here, some confidential information about the faculty is placed in a separate relation and information about offices is also added. The Courseinfo view relation can be redefined as Faculty_public and

Faculty_private, which together contain all the information of faculty, so that the user who queries Courseinfo will get the same answers as before. Here the users are shielded from the changes in the logical structures of the data, or changes in the choice of relations to be stored. This property is called Logical data independence. The conceptual schema insulates users from changes in physical storage details. This property is referred as Physical data independence. i.e., the conceptual schema hides details such as how the data is originally stored on disk, it file structure, the choice of indexes etc

1.6 Transaction management

- A transaction is a collection of operations that performs a single logical function in a database application or A transaction is any one execution of a user program in a DBMS (Executing the same program several times will generate several transactions).
- Transaction-management component ensures that the database remains in a consistent (correct) state despite system failures (e.g., power failures and operating system crashes) and transaction failures.
- Concurrency-control manager controls the interaction among the concurrent transactions, to ensure the consistency of the database.

Examples:

Consider a database that holds information about airline reservations. At any given instant, it is possible (and likely) that several travel agents are looking up information about available seats on various flights and making new seat reservations. When several users access (and possibly modify) a database concurrently, the DBMS must order their requests carefully to avoid conflicts. For example, when one travel agent looks up Flight 100 on some given day and finds an empty seat, another travel agent may simultaneously be making a reservation for that seat, thereby making the information seen by the first agent obsolete.

The two major issues addressed in Transaction Management are:

- Concurrent execution of Transactions
- Incomplete Transactions and System Crashes

1.6.1 Concurrent execution of Transactions

One of the important tasks of a DBMS is to schedule concurrent accesses to data, so that each user is ignoring or unaware that the other users are accessing the data concurrently by submitting their requests to the DBMS independently. A DBMS allows the users to think of their programs as if they are executing in isolation, one after the other in isolation choosen by some order by the DBMS. For example: it the program that deposits cash into an account is submitted to the DBMS at the same time as another program debits money from the same account, either a program is run first by the DBMS, but their steps will not be interleaved such that they interfere with each other.

A **locking protocol** is a set of rules to be followed by each transaction to ensure that, even though the actions of several transactions is interleaved, the net effect is identical to executing all transactions in some serial order. A **Lock** is a mechanism used to control access to the database objects. The two kinds of locks used by the DBMS are: **shared Locks** on an object can be held by two different transactions at the same time, but an **exclusive lock** on an object ensures that no other transactions hold any lock on this object.

Example:

Suppose that the following locking protocol is followed: Every transaction begins by obtaining a shared lock on each data object that it needs to read and an exclusive lock on each data object that it needs to modify, and then releases all its locks after completing all actions. Consider two transactions *T*1 and *T*2 such that *T*1 wants to modify a data object and *T*2 wants to read the same object. Intuitively, if *T*1's requestor an exclusive lock on the object is granted first, *T*2 cannot proceed until *T*1 releases this lock, because *T*2's request for a shared lock will not be granted by the DBMS until then. Thus, all of *T*1's actions will be completed before any of *T*2's actions are initiated.

1.6.2 Incomplete Transactions and System Crashes

Transactions can be interrupted before running to completion due to the reasons such as system crashes etc. A DBMS ensures that the changes made by such incomplete transactions are removed from the database. For example, if the DBMS is in the middle of transferring money from account A to account B and has debited the first account but not yet credited the second account when the crash occurs, the money debited from account A must be restored when the system comes back up after the crash. For this purpose the DBMS maintains a log of all writes to the database. The property of the log is that after each write action, it must be recorded in the log (on disk) before the corresponding change is reflected in the database, otherwise the database will not be able to detect and undo the change that is made. This property is called Write-Ahead Log, or WAL. This is implemented by the DBMS by selectively forcing page in memory to disk. The log also ensures that the changes made by a successfully completed transaction are not lost due to a system crash. The time required to recover from a crash can be reduced by periodically forcing some information to disk, and this periodic operation is called a checkpoint.

1.7 Structure of a DBMS

Figure: 1.5 shows the structure of a typical DBMS based on the relational data model. The DBMS accepts SQL commands generated from a variety of user interfaces, produces query evaluation plans, executes theses plans against the database, and returns the answers.

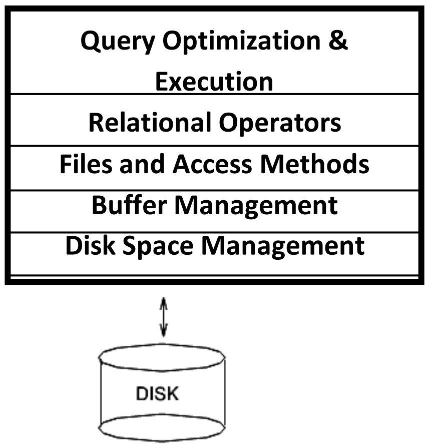


Figure: 1.4 General structure of a DBMS.

When the user issues a query, the parsed query is presented to a query optimizer, which uses information about how the data is stored to produce an efficient execution plan for evaluating the query. An execution plan acts as a blueprint for evaluating a query, represented as a tree of relational operators. Here the relational operators act as a building block for evaluating queries posed against the data.

The code which implements relational operators sits on the top of the **file and access method layer**. This layer supports the concept of a file, which, in a DBMS, is a collection of pages or a collection of records. **Heap files** or files of unordered pages, as well as indexes are supported. In addition to keeping track of the pages in a file, this layer organizes the information within the page.
The files and access methods layer code sits on top of the **buffer manager**, which brings pages in from disk to main memory as needed in response to read requests. The lowest layer of the DBMS software deal with management of space on disk, where the data is stored. The higher layers allocate, de-allocate, read, and write pages through routines provided by this layer, called the **disk space manager.**

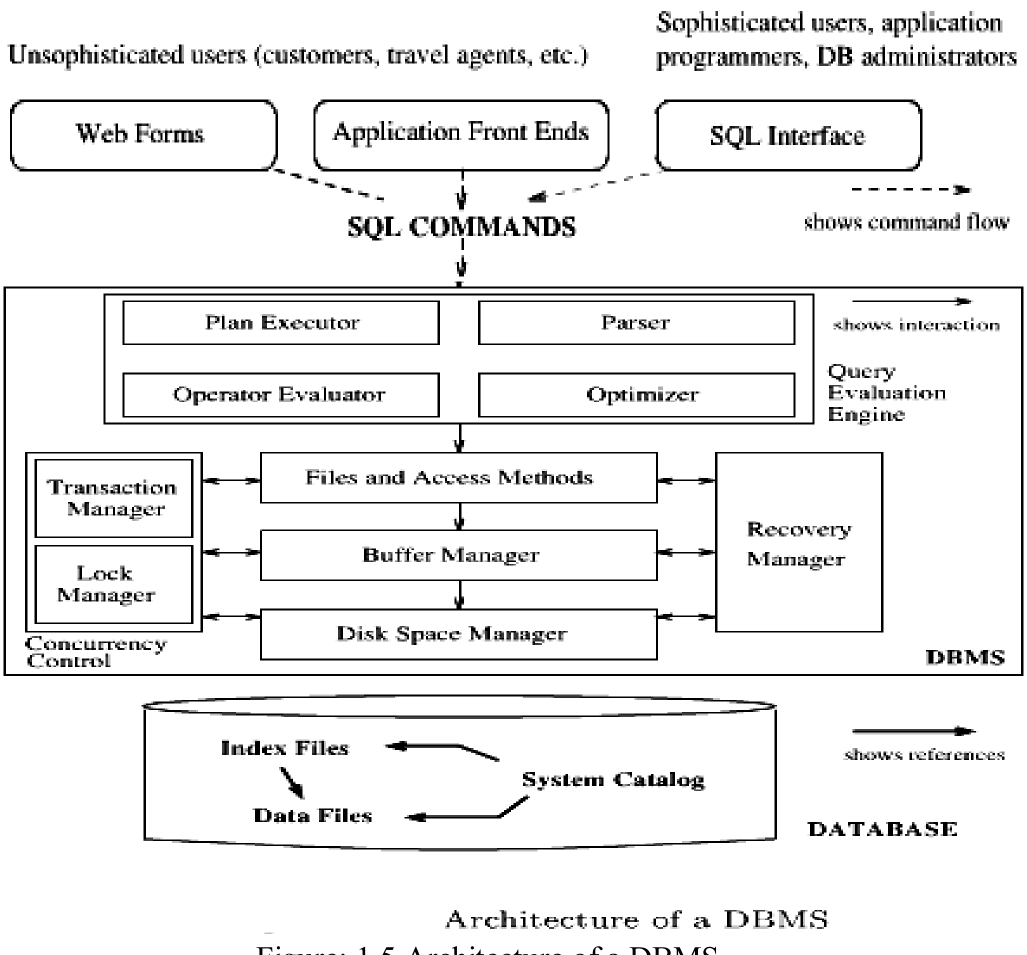


Figure: 1.5 Architecture of a DBMS.

The DBMS supports concurrency and crash recovery by carefully scheduling the various user requests and maintaining a log of all changes to the database. The DBMS components associated with concurrency control and recovery from crashes include the **transaction manager**, which is the component that ensures that the transactions request and release locks according to a suitable locking protocol and schedules the execution of transactions. The **lock manager** keeps track of the requests of the locks and grants locks on database objects when they become available. The **recovery manger** is responsible for, maintaining a log and restoring the system to a consistent state after a crash. The disk space manger, buffer manger, and file and access method layers will interact with these components.

1.7.1 Application Architecture

Two-tier architecture: E.g. client programs using ODBC/JDBC to communicate with a database Three-tier architecture: E.g. web-based applications and applications built using middleware.

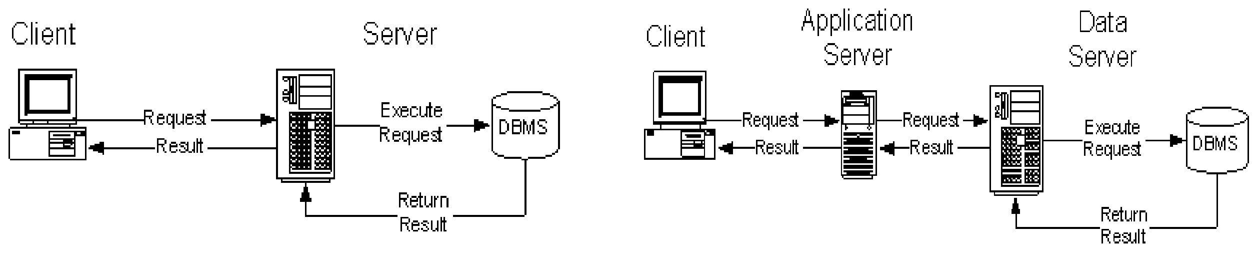


Figure: 1.6 Two tier Application Architecture and Three tier Application Architecture.

1.8 People who work with Databases

A variety of people are associated with the creation and use of databases. They mainly include: **Database Implementers and End users.**

The **Database Implementers** are people who build DBMS software, who work for vendors such as IBM or Oracle. The **End Users** are people who wish to store and use data in a DBMS and come from a diverse and increasing number of fields. The end users known as Naive users mainly simply use the applications written the database application programmers and so require only little technical knowledge about DBMS software. Whereas the **sophisticated users** require more extensive use of a DBMS, such as writing their own queries, which require deeper understanding of its features.

Database Users

Users are differentiated by the way they expect to interact with the system

- **Application programmers** interact with system through DML calls.
- **Sophisticated users** form requests in a database query language.
- **Specialized users** write specialized database applications that do not fit into the traditional data processing framework.
- **Naive users** invoke one of the permanent application programs that have been written previously

  E.g. people accessing database over the web, bank tellers, clerical staff

In addition to the Database Implementers and End users, two other classes of people are associated with a DBMS called, **Application programmers and Database Administrators.**

Database Application programmers develop packages that facilitate data access for the end users, who are not computer professionals, using the host or data languages and software tools that the

DBMS vendors provide, which include report writers, spreadsheets, statistical packages etc. These application programs access data at the lower level through external schema.
The personnel database is maintained by the individual who owns it and uses it. The corporate or enterprise-wide databases are important to be maintained and are complex enough that the task of designing and maintaining the database is entrusted to a professional, called the **database Administrator (DBA).**

1.8.1 Database Administrator

- Coordinates all the activities of the database system; the database administrator has a good understanding of the enterprises information resources and needs.
- Database administrator's duties include:
  - Schema definition
  - Storage structure and access method definition
  - Schema and physical organization modification
  - Granting user authority to access the database
  - Specifying integrity constraints
  - Acting as liaison with users
  - Monitoring performance and responding to changes in requirements

The DBA is responsible for the following tasks mentioned:

- **Design of the conceptual and Physical Scheme**
  The DBA is responsible for interacting with the users of the system to understand what data to be stored in the DBMS and how it is to be used, based on this knowledge, the DBA designs the conceptual scheme(the various relation to be stored), and the physical schema(Decision on how to store the various relations). The DBA also designs the major widely used portion of the external schema.
- **Security and Authorization**
  The DBA responsible for ensuring that the unauthorized access is not permitted, that, not everyone should be able to access all the data. In a relational DBMS, users need to be granted permission to access only certain views and relations. For example: The DBA enforces the permission of access to various classes of users like, the students are allowed to find out the course enrollments, the teachers who teaches the course, but need not permit students to see the faculty salaries or each other's grade information.
- **Data Availability and Recovery from Failures**
  The DBA takes steps to ensure that if the system fails , users can continue to access as much of the uncorrupted data as possible . The DBA also restores the data to a consistent state. The DBMS provides software support for such functions, but the DBA is responsible for implementing procedures to back up the data periodically and maintain logs of system activity to facilitate recovery from crash situations.
- **Database Tuning**

  The user's needs always tends to be evolving with time, the DBA is responsible for modifying the database, particularly the conceptual and physical schema, to ensure adequate performance as the requirements changes.

1.9 Overview of Database Design

The **entity-relationship(ER)** data model allows us to describe the data involved in a real world enterprise in terms of objects and their relationships and is widely used to develop an initial database design. The ER model is used in a phase called conceptual database design.

Database design and ER Diagrams

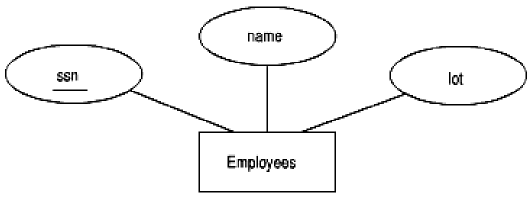
The database design process can be divided into six steps, where the ER model is relevant for the first three steps. They include:

- ☞ **Requirement Analysis:** This is the very first step in the database design, which is to understand what data is to be stored in a database, what applications must be built on top of it, and what operations are most frequent and subject to performance requirements. This analysis finds out what users want from the database, mainly collected from informal discussion with the user groups, studying the current operation environment.
- ☞ **Conceptual database Design:** The information gathered from the requirement analysis phase is used to develop a high-level data description of the data stored in the database, along with the constraints to be on the data, which is mainly done using the ER model. The ER model is the appropriate description of the data, constructed through subjective evaluation of the information collected during requirement analysis.
- ☞ **Logical Database Design:** In the logical database design a DBMS is chosen to implement the database design, and convert the conceptual database design to the database scheme in the data model of the chosen DBMS.
- ☞ **Schema Refinement:** This refers to the identification of potential problems of the collection of relations in the relational database schema, and to refine it, mainly using the normalization theories.
- ☞ **Physical database design:** Considering the amount of data handled by the database , it is further refined in design to ensure that it meets the required performance criteria, by building suitable indexes on the stored tables and clustering them with required redesign of the structure.
- ☞ **Application of the Security Design:** the design methodologies like UML addresses the complete software design and development cycle, identifying the entities and processes involved in the application, which involves describing the role of each entity and process in the application task, as part of the workflow for that task.

1.10 Entities, Attributes and Entity Sets

An **Entity** is an object in the real world that is distinguishable from other objects. An **Entity set** is the collection of similar entities grouped together. Example: An employee in an enterprise is an entity, but the set of all the employees in the enterprise comprise the entity set.

An entity is often described using the set of **Attributes,** which are the properties or features that describe the entity. The attributes for all the entities in an entity set is similar. For example the employee entity set can be described with name, social security number (ssn), and parking lot (lot) as attributes, where to record the details of an employee from the employee entity set, we use name, ssn, lot for each employee as shown in the Figure: 1.7.



The Employees Entity Set

Figure: 1.7

For each attribute associated with an entity set, a domain of possible values are identified, for example the domain associated with the attribute name of employees might be set of 20-character strings, the lot values may be drawn from a range of 1 to 10 of integer values etc. Every entity set is identified using a Key. A **Key** is a minimal set of attributes whose values uniquely identifies an entity in the entity set. A set of key values which is used to identify an entity in the entity in the entity set is called **Candidate key**. In a candidate key, one of the key values is the primary key, i.e., Candidate key is the primary key in conjunction with any other key values, which is used to identify the entity in the entity set.

ER Diagram

Entity relationship diagramming is a technique that is widely used in the world of business and information technology to show how information is, or should be, stored and used within a business system. The success of any organization relies on the efficient flow and processing of information.

An E-R diagram expresses the overall logical structure of a database graphically. A detail for graphic symbols used are shown in the Table: 1.0

Description:

1) Use rectangles representing entity sets.
2) Use ellipses representing attributes
3) Use diamonds representing relationship sets.
4) Use lines linking attributes to entity sets and entity sets to relationship sets.
5) The Bold underline for the attribute represents the Primary Key.
6) The Dotted underline for the attribute represents the Partial Key.
7) Use of Dotted ellipses representing derived attributes.
8) Dual ellipse represents multivalued attribute.
9) Use of Dual diamonds representing Identifying relationship set.
10) Use of Dual rectangles representing weak entity sets.

1.11 Relationships and Relationship sets

A Relationship is the association between two or more entities. A set of similar relationships are collected into a Relationship set. A relationship set can be thought of as a set of n-tuples represented as $\{ (e_1, \ldots, e_n) / e_1 \in E_1, \ldots, e_n \in E_n\}$, where each n-tuple denotes a relationship involving n entities e1 through en, where entity ei is in entity Ei. In the Figure: 1.8, Works_In relationship indicates a department in which an employee works.

Example: 1 Simple Relationship Set

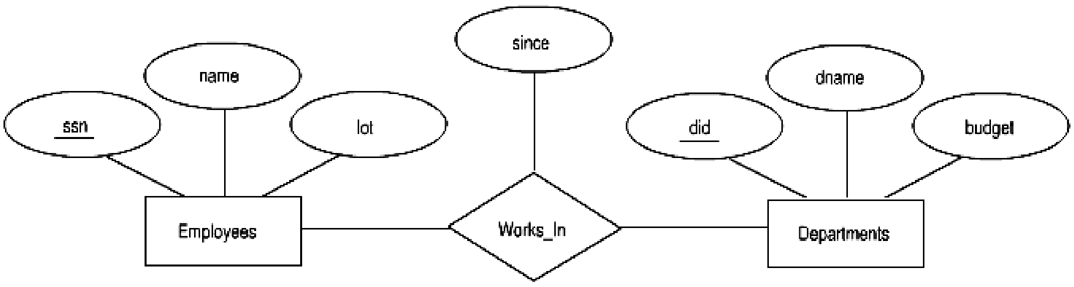


Figure: 1.8

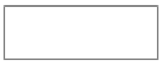
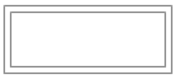
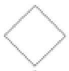
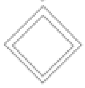
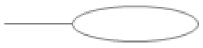
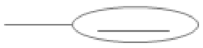
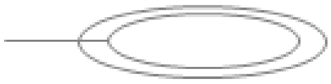
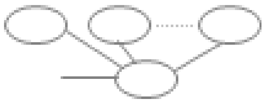
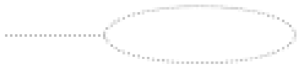
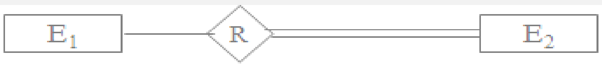
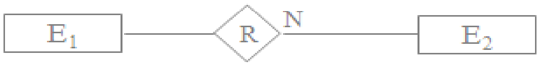
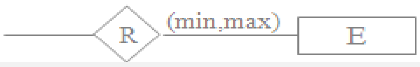
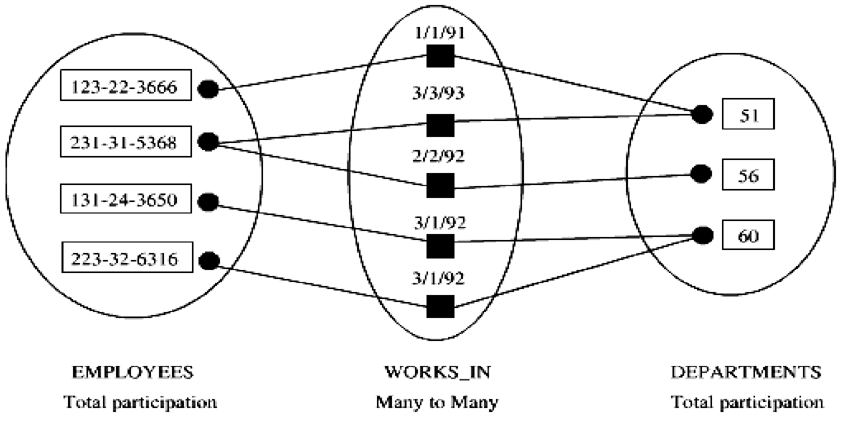
SYMBOL	MEANING
	ENTITY TYPE
	WEAK ENTITY TYPE
	RELATIONSHIP TYPE
	IDENTIFYING RELATIONSHIP TYPE
	ATTRIBUTE
	KEY ATTRIBUTE
	MULTIVALUED ATTRIBUTE
	COMPOSITE ATTRIBUTE
	DERIVED ATTRIBUTE
	TOTAL PARTICIPATION OF E_2 IN R
	CARDINALITY RATIO 1:N FOR E_1:E_2 IN R
	STRUCTURAL CONSTRAINT (min, max) ON PARTICIPATION OF E IN R

Table: 1.0

The attribute that describes a relationship is referred as **Descriptive attribute**. Descriptive attributes are used to record information about the relationships. For example, any employee E1 in the

Employees entity set working in any Department (as pharmacy), the additional information about the relationship between the employee E1 and the Pharmacy Department can be recorded as by adding an, Since, which is descriptive attribute to the relationship Works_In. The employee E1 is working since January 1991 in the Pharmacy department. Here the relationship set is uniquely identified by the participating entity set, i.e., combination of employee ssn and department did is used to uniquely identify the relationship among the entities, where there cannot be more than one value that can identify a relationship among the entities through the since attribute value.



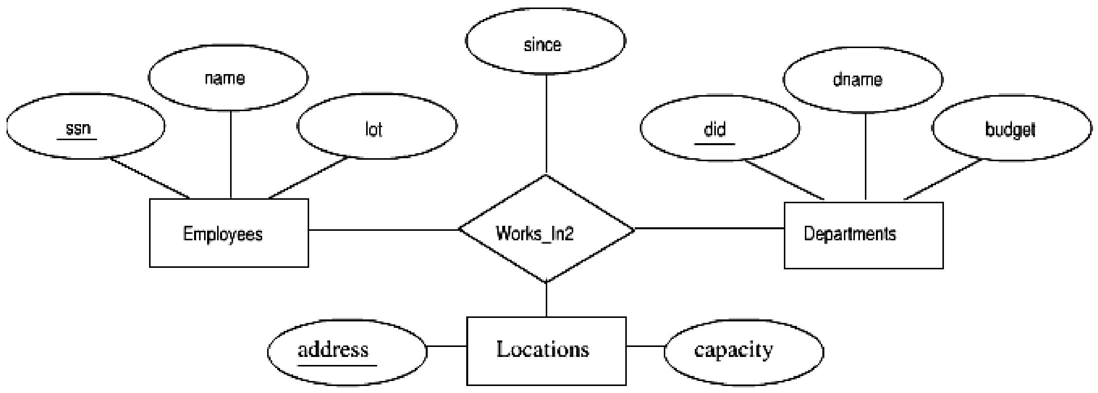
An Instance of the Works_In Relationship Set

Figure: 1.9

An Instance of a relationship set is the set of relationships at any point of time, that is, it is referred as the snapshot of the relationship set at some instant of time. The snapshot of Works_In relationship set is shown in the Figure: 1.9, where the employee entity is identified using its ssn and the departments entity is identified using its 'did' key and the descriptive attribute since value is recorded as a **many-to-may** and **total participation constraints**.

Example: 2: Ternary relationship Set

Consider an ER Diagram that represents the departments which has offices in several locations and records the locations in which an employee from the employee entity set works. This relationship is referred as Ternary relationship as shown in the Figure: 1.10, because there is a need to record an association between an employee, a department, and a location.

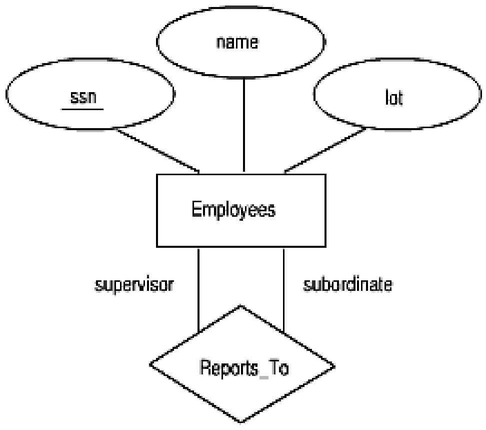


A Ternary Relationship Set

Figure: 1.10

Example: 3: Relationship set with role indicators

The participating entity set in a relationship set need not be distinct, where a relationship might involve two entities in the same entity set, for example, consider the Reports_To relationship set in the Figure:1.11, since the lower level employee that reports to another employee of higher grade is in turn an employee itself, from the same Employee entity set, where every relationship in Reports_To is of the form (emp1,emp2), such that both emp1 and emp2 are employees of the Employees set, but they play different Roles, emp1 reports to the managing employee emp2, which can be reflected by the **Role indicators** : supervisor and subordinate in the Figure:1.11. In such cases where an entity set plays more than one role, the role indicator together with an attribute name from the entity set gives a unique identity to each attribute in the relationship set.



The Reports_To Relationship Set

Figure: 1.11

As shown in the Figure:1.11, the Reports_To relationship set has attributes, which corresponds to ssn of the supervisor and the ssn which corresponds to subordinate, and the names of these attributes are supervisor_ssn and subordinate_ssn.

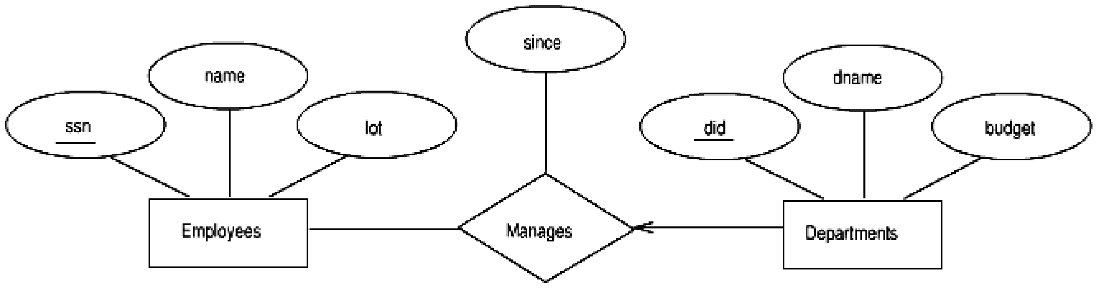
1.12 Additional Features of E-R Model

The additional feature of the ER model illustrates the expressiveness of the ER model, and its widespread use.

1.12.1 Key Constraints

Considering the example as mentioned in diagram in figure: 1.10 and the corresponding instance diagram in Figure: 1.12, an employee can work in several departments, and a department can have several employees, as shown in the Works_In instance diagram. Employee 231-31-5368 has worked in Department 51 since 3/3/93 and in department 56 since 2/2/92, also Department 51 has two employees. Considering another case in the relationship set called manages between the employees and the departments entity sets, such that each department has at the most one manager, that is, only a single employee is allowed to manage more than one department. Here the restriction that each department has at the most one manager is an example of a key constraint. This restriction of a Key constraint is represented using an arrow from Departments to Manages.

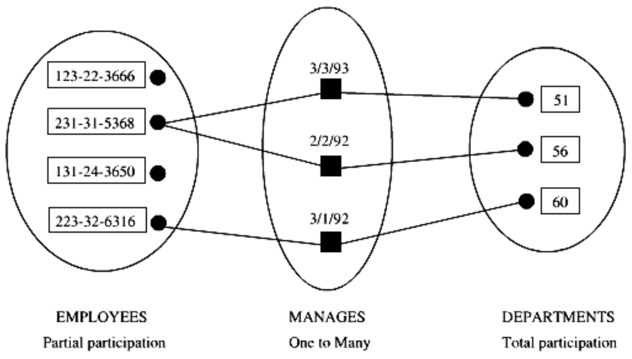
An instance of the manages relationship set is represented in the Figure: 1.13. here the relationship set Manages is a one-to-many, indicating that one employee can be associated with many departments in the post of manger, whereas each department can be associated with at most one employee as its manger.



Key Constraint on Manages

Figure: 1.12

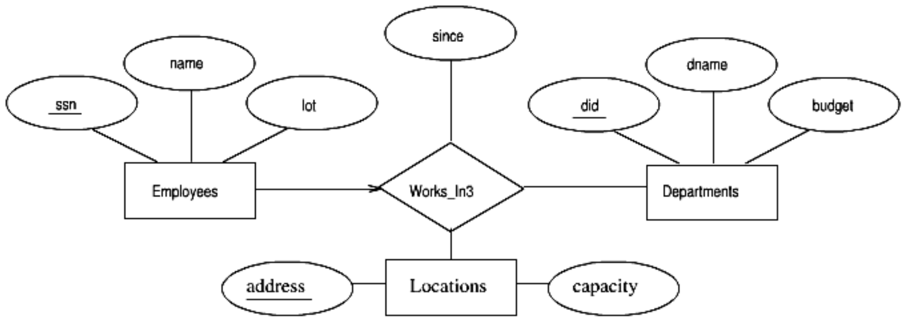
Whereas the Works_In relationship set, where an employee is allowed to work in several departments and a department is allowed to have several employees, is referred as many-to-many relationship. If the condition that an employee can manage only at the most one Department to the Manages relationship set, it is indicated by an arrow from the employee to manages, and is called a one-to-one relationship set.



An Instance of the Manages Relationship Set

Figure: 1.13

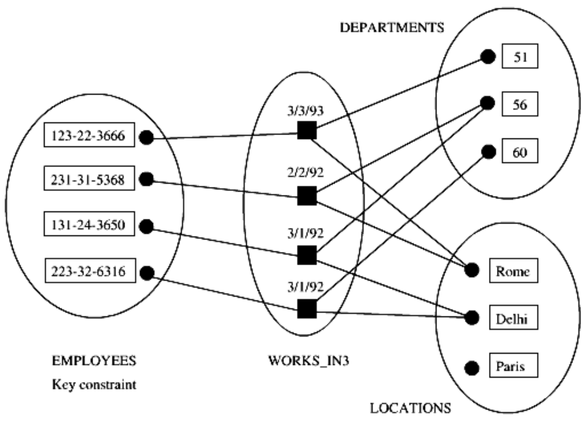
1.12.2 Key Constraints for Ternary Relationships



A Ternary Relationship Set with Key Constraints

Figure: 1.14

The key constraint concept involving relationship set with more entity sets or more than two entity sets is referred as ternary relationship. If entity set E, with a key constraint in the relationship set R, with each entity in an instance of E appearing in at the most on relationship in R. In the Figure: 1.14, a Ternary relationship set Works_In, with more than two entities Employee, Departments, Locations, is represented with a Key constraint is explained as, each employee works in at the most one Department and at a single Location. The key constraint here is explained as each department can be associated with several employees and locations and each location can be associated with several departments and employees, but each employee is associated with a single department and location. The instance diagram of the Works_In relationship set is shown in the Figure:1.15.

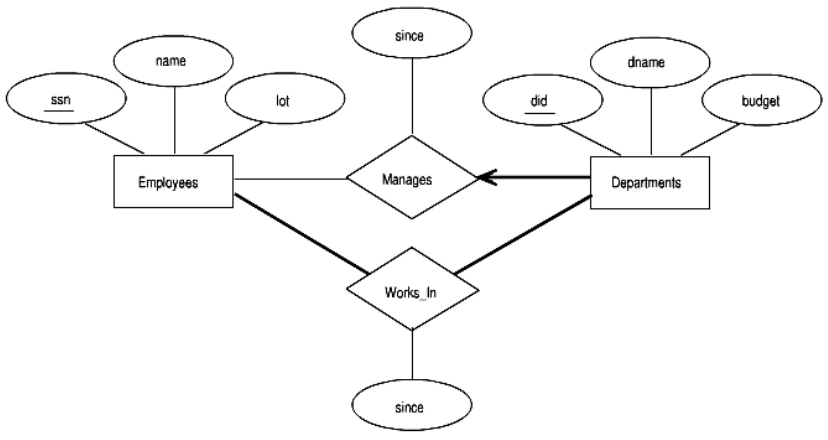


An Instance of Works_In3

Figure: 1.15

1.12.3 Participation Constraint

The participation constraint can be explained, with the question that does every department is required to have a manager? In the ER diagram mentioned in the Figure: 1.16, the participation of the entity set Departments in the relationship set Manages is considered as **Total participation**.



Manages and Works_In

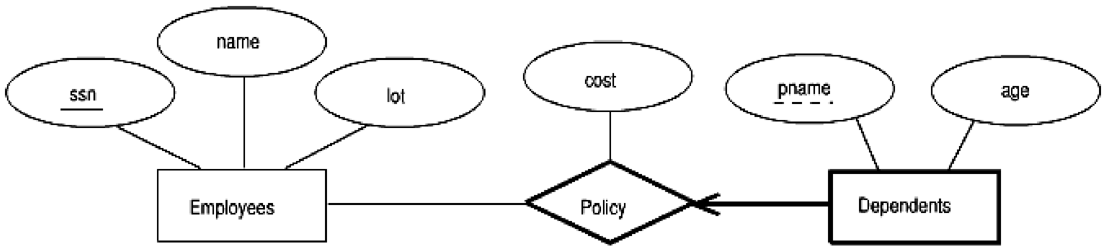
Figure: 1.16

A participation which is not total is referred as Partial participation. As an example, the participation of the entity set manages is partial, as not every employee gets a chance to manage a Department.

In the ER diagram mentioned in the Figure: 1.16, for the participation of entity set with Total participation, is indicated with Thick line, the presence of an arrow represents key constraint. Here the participation of both Employees and Departments in Works_In relationship set is **Total**, whereas the participation constraint of the employee in the Manages relationship set is **partial**.

1.12.4 Weak Entities

Those entity sets which cannot be identified uniquely using its attributes, is referred as Weak entity set. A weak entity is identified uniquely, by considering some of the attributes of the weak entity set in conjunction with the primary key of another entity, referred as identifying owner, such that the owner entity set and the weak entity set must participate in one-to-many relationship set, where the owner entity is associated with one or more weak entities, but each weak entity has a single owner. Along with this, a weak entity set must have total participation in the identifying relationship set. The set of attributes of a weak entity set that uniquely identify a weak entity for any given owner entity is called a partial key of the weak entity set.



A Weak Entity Set

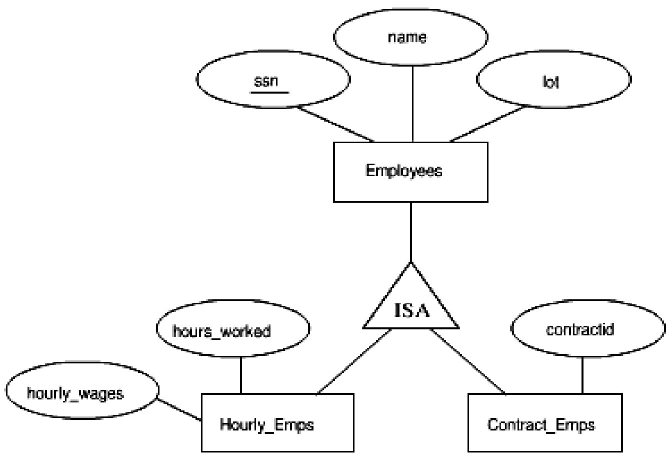
Figure: 1.17

Consider an Employees entity set covering insurance policies to cover their dependents, to record information about the policies, including the persons covered by each policies called as Dependents of an employee. If any employee quits from the organization the corresponding policy and all the dependent information is as well deleted from the database. In the ER diagram as mentioned in the Figure:1.17, the attributes of the dependent is as dependent name(pname) and age, but the attributes pname or age does not identify a dependent uniquely, as there could be different employees with same dependent names. The key for Employees is ssn. Here the Dependents entity can be identified uniquely only if the key of the owning Employees entity (ssn) and the pname of the Dependents entity are used together. Here pname is a partial key for the Dependents.
In the Figure: 1.17, the total participation of the Dependents in Policy is indicated by linking them with dark line. The arrow from dependents to Policy indicates that each Dependents entity appears in at most one policy relationship. To identify Dependents as weak entity set and policy as its identifying relationships, both are represented s using Dark lines, to indicate that pname is a partial key for Dependents, it is underlined using broken line, indicating that two dependents with same pname is possible.

1.12.5 Class Hierarchies

Class hierarchies are used to classify the entities in an entity set into subclasses. For example, as Hourly Emps entity set and a Contract Emps entity set to distinguish the basis on which they are paid. Attributes such as *hours worked* and *hourly wage* can be defined for Hourly Emps and an attribute *contracted* defined for Contract Emps.

Here the semantics that every entity in one of these sets is also an Employees entity, and will have all of the attributes of Employees defined for both the subclasses. Therefore, the attributes defined for an Hourly Emps entity are the attributes for Employees plus Hourly Emps. That is the attributes for the entity set Employees are **inherited** by the entity set Hourly Emps, and that Hourly Emps **ISA** (read *is a*) Employees. A query that asks for all Employees entities must consider all Hourly Emps and Contract Emps entities as well. Figure 1.18 illustrates the class hierarchy.



Class Hierarchy

Figure: 1.18

A different form of classifying the entity set Employees is using a different criterion. For example, we might identify a subset of employees as Senior Emps. We can modify Figure 1.18 to reflect this change by adding a second ISA node as a child of Employees and making Senior Emps a child of this node. Each of these entity sets might be classified further, creating a multilevel ISA hierarchy. A **class hierarchy can be viewed in one of two ways:**

- An employee entity is specialized into subclasses. **Specialization** is the process of **identifying subsets of an entity set (the super class) that share some distinguishing characteristic**. The super class is defined first, and the subclasses are defined next with the Subclass-specific attributes and relationship added next.

- Hourly Emps and Contract Emps are generalized by Employees. As another example, two entity sets Motorboats and Cars may be generalized into an entity set Motor Vehicles. **Generalization** consists of **identifying some common characteristics of a collection of entity sets** and creating a new entity set that contains entities possessing these common characteristics. Here the subclasses are defined first, the super class is defined next, and any relationship sets that involve the super class are then defined.

To specify two kinds of constraints with respect to **ISA hierarchies**,

- ***overlap* and**
- ***Covering* constraints.**

Overlap constraints determine whether two subclasses are allowed to contain the same entity. For example, can Ajay be both an Hourly Emps entity and a Contract Emps entity? The answer is, no. Can Ajay be both a Contract Emps entity and a Senior Emps entity? Intuitively, yes. This could be denoted by writing **'Contract Emps OVERLAPS Senior Emps.'**

Covering constraints determine whether the entities in the subclasses collectively include all entities in the super class. For example, does every Employees entity have to belong to one of its subclasses? The answer is, no, or with the second example, Does every Motor Vehicles entity have to be either a Motorboats entity or a Cars entity? The answer is, yes; a characteristic property of generalization hierarchies is that **every instance of a super class is an instance of a subclass**. This constraint is denoted this by writing **'Motorboats AND Cars COVER Motor_vehicles.**

1.12.6 Aggregation

A relationship set is an association between entity sets. The need to model a relationship between a collection of entities and *relationships*, we consider an entity set called Projects and that each Projects entity is sponsored by one or more departments. Here the Sponsors relationship set captures this information. A department that sponsors a project might assign employees to monitor the sponsorship. Here, Monitors should be a relationship set that associates a Sponsors relationship (rather than a Projects or Departments entity) with an Employees entity. In aggregation relationships to associate two or more *entities* are defined.

To define a relationship set such as Monitors, a new feature of the ER model, called *aggregation*. **Aggregation** allows to indicate that a relationship set (identified through a dashed box) participates in another relationship set. This is illustrated in Figure 1.19, with a dashed box around Sponsors (and its participating entity sets) used to denote aggregation. This effectively allows to treat Sponsors as an entity set for purposes of defining the Monitors relationship set.

Aggregation is used when we need to express a relationship among relationships. Can't we express relationships involving other relationships without using aggregation? In the example, why Sponsors is made a ternary relationship? The answer is that there are really two distinct relationships, Sponsors and Monitors, each possibly with attributes of its own.

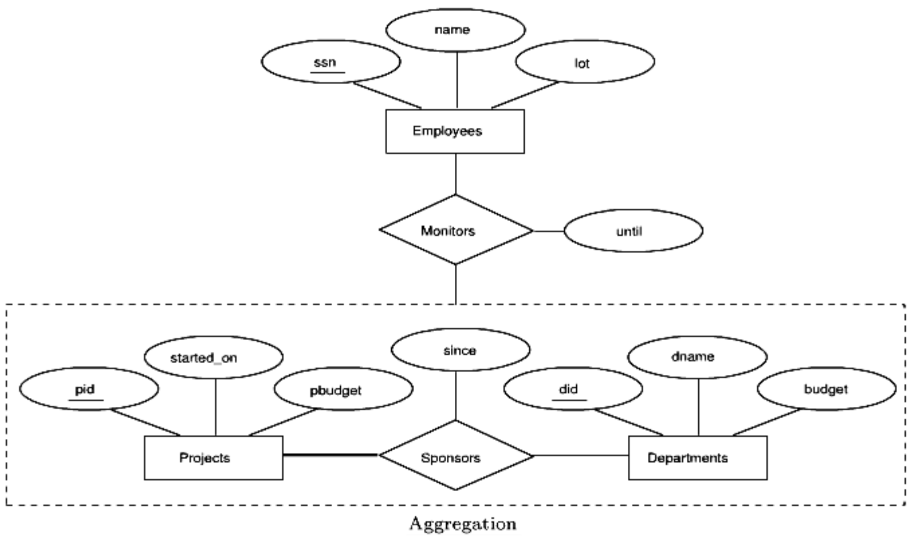


Figure: 1.19

For instance, the Monitors relationship has an attribute *until* that records the date until when the employee is appointed as the sponsorship monitor. Comparing this attribute with the attribute *since* of Sponsors, which is the date when the sponsorship took effect. The use of aggregation versus a ternary relationship is also guided by certain integrity constraints.

1.13 Conceptual Design with the E-R Model

Developing an ER diagram presents several choices, including the following:

- ☞ Should a concept be modeled as an entity or an attribute?
- ☞ Should a concept be modeled as an entity or a relationship?
- ☞ What are the relationship sets and their participating entity sets? Should we use
- ☞ Binary or ternary relationships?
- ☞ Should we use aggregation?

The conceptual data base design with ER model, is guided by the choices made above.

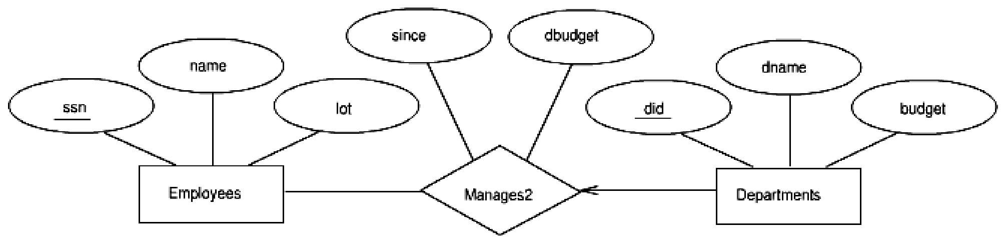
1.13.1 Entity versus Attribute

While identifying the attributes of an entity set, it is sometimes not clear whether a property should be modeled as an attribute or as an entity set. For example, consider adding address information to the Employees entity set.

The various options to model an attribute are:

- ➢ One option is to use an attribute *address*. This option is appropriate if we need to record only one address per employee, and address is a string.
- ➢ An alternative is to create an entity set called Addresses and to record associations between employees and addresses using a relationship (say, Has Address). This second complex alternative is considered in two situations:
  - ✍ If we have to record more than one address for an employee and if we want to capture the structure of an address in our ER diagram. For example, the address can be broken down into city, state, country, and Zip code, in addition to a string for street information. By representing an address as an entity with these attributes, as needed can be used to support queries such as "Find all employees with an address in Madison, WI.", which otherwise is not possible.

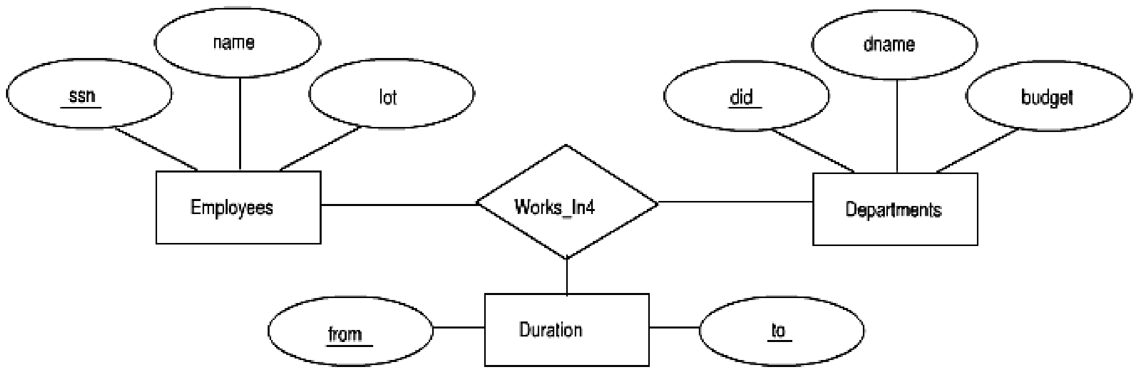
For another example of when to model a concept as an entity set rather than as an attribute, consider the relationship set (called Works In2) shown in Figure 1.20.



Entity versus Relationship

Figure: 1.20

The Works In relationship set of Figure 1.14 has attributes *from* and *to*, instead of *since*. Here, it records the interval during which an employee works for a department. In certain cases it is possible for an employee to work in a given department over more than one period.
The problem is that we want to record several values for the descriptive attributes for each instance of the Works In2 relationship. We can address this problem by introducing an entity set called, say, Duration, with attributes *from* and *to*, as shown in Figure 1.21.

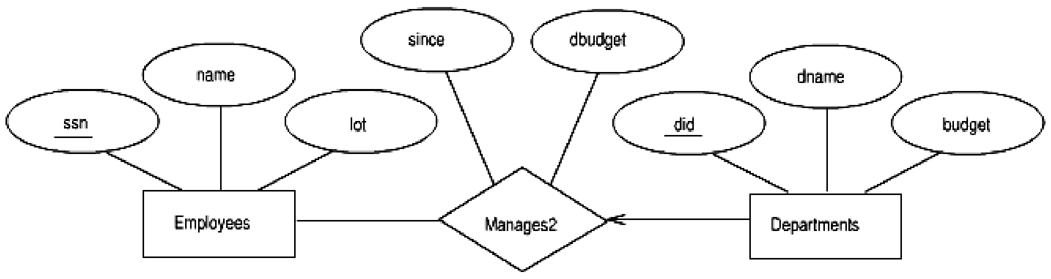


The Works_In4 Relationship Set
Figure: 1.21

In ER models, attributes are allowed to take on sets as values. In this feature, Duration is made an attribute of Works In, rather than an entity set; associated with each Works In relationship, where as set intervals are represented. This approach is more useful than modeling Duration as an entity set. Where, such set-valued attributes are translated into the relational model, as the ER diagram does not support set-valued attributes, the resulting relational schema is very similar to what we get by regarding Duration as an entity set.

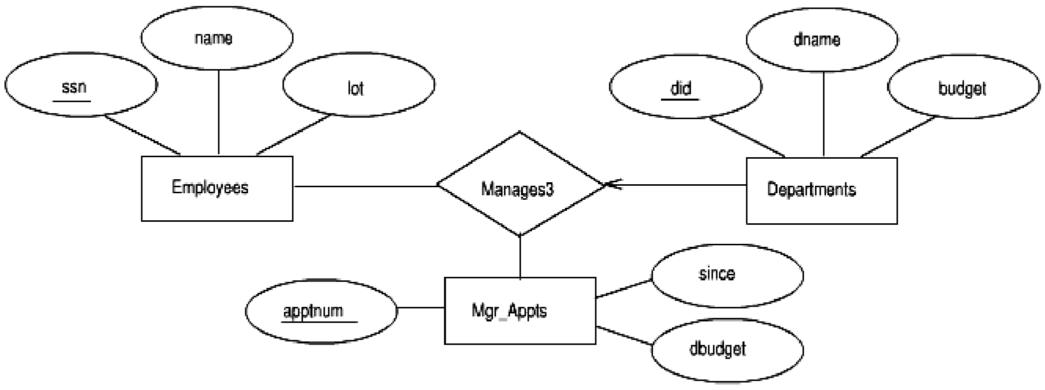
1.13.2 Entity versus Relationship

Consider the relationship set called Manages in Figure 1.12. Suppose that each department manager is given a discretionary budget (*dbudget*), as shown in Figure 1.22, in which we have also renamed the relationship set to Manages2.



Entity versus Relationship
Figure: 1.22

There can be one employee managing a department, but another case is that a given employee could manage several departments; where it is required to store the starting date and discretionary budget for each manager-department pair. This approach is usual, if a manager receives a separate discretionary budget for each department that he or she manages. On the other hand, if the discretionary budget is a sum that covers *all* departments managed by an employee, In this case each Manages2 relationship that involves a given employee will have the same value in the *dbudget* field. In this case redundancy could be significant and could cause a variety of problems.



Entity Set versus Relationship

Figure: 1.23

In this case, this problem is addressed, by associating *dbudget* with the appointment of the employee as manager of a *group* of departments and, we model the appointment as an entity set, as Mgr Appt as shown in the Figure:1.23, and use a ternary relationship, for Manages3, to relate a manager, an appointment, and a department. The details of an appointment (such as the discretionary budget) are not repeated for each department that is included in the appointment now, although there is still one Manages3 relationship instance per such department. Also each department has at most one manager, as before, because it is the key constraint as illustrated in Figure 1.23.

1.13.3 Binary versus Ternary Relationships

Consider the ER diagram shown in Figure 1.23. It models a situation, in which an employee can own several policies, and each policy can be owned by several employees, and each dependent can be covered by several policies.

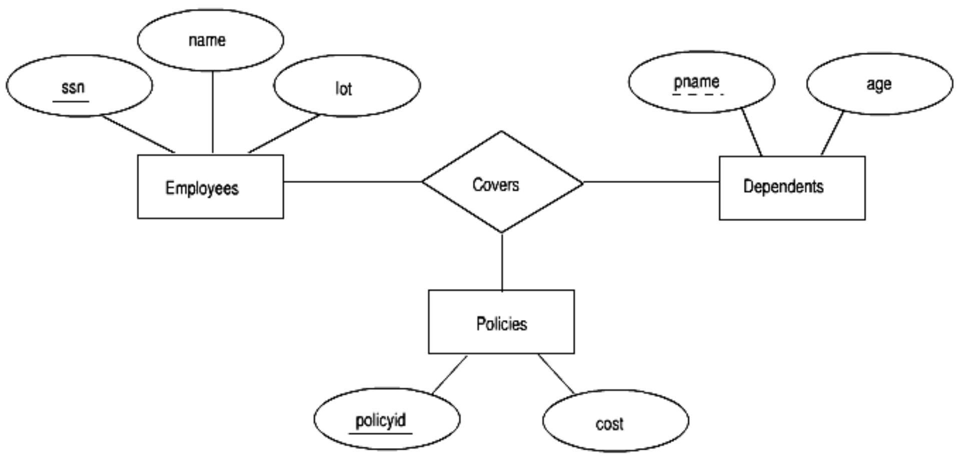
Suppose that we have the following additional requirements:

- A policy cannot be owned jointly by two or more employees.
- Every policy must be owned by some employee.
- A dependent is a weak entity set, and each dependent entity is uniquely identified by taking *pname* in conjunction with the *policyid* of a policy entity (which, intuitively, covers the given dependent).

The first requirement suggests that we impose a key constraint on Policies with respect to Covers, but this constraint has the unintended side effect that a policy can cover only one dependent.

The second requirement suggests that we impose a total participation constraint on Policies. This solution is acceptable if each policy covers at least one dependent.

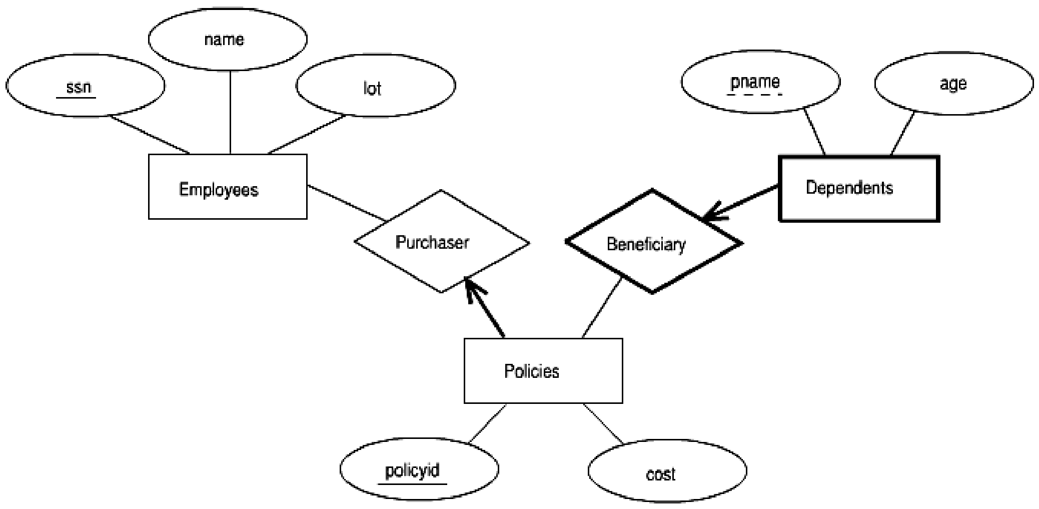
The third requirement forces us to introduce an identifying relationship that is binary.
The solution to model with this situation is to use two binary relationships, as shown in Figure 1.24.



Policies as an Entity Set

Figure: 1.24

This example has two relationships involving Policies, and the attempt to use a single ternary relationship (Figure 1.24) is inappropriate. There are situations, however, where a relationship inherently associates more than two entities, such an example in Figure 1.23 and also Figures 1.24 and 1.25.



Policy Revisited

Figure: 1.25

As a good example of a **ternary relationship**, consider entity sets Parts, Suppliers, and Departments, and a relationship set Contracts (with descriptive attribute *qty*) that involves all of them. A contract specifies that a supplier will supply (some quantity of) a part to a department. This relationship cannot be adequately captured by a collection of binary relationships (without the use of aggregation).

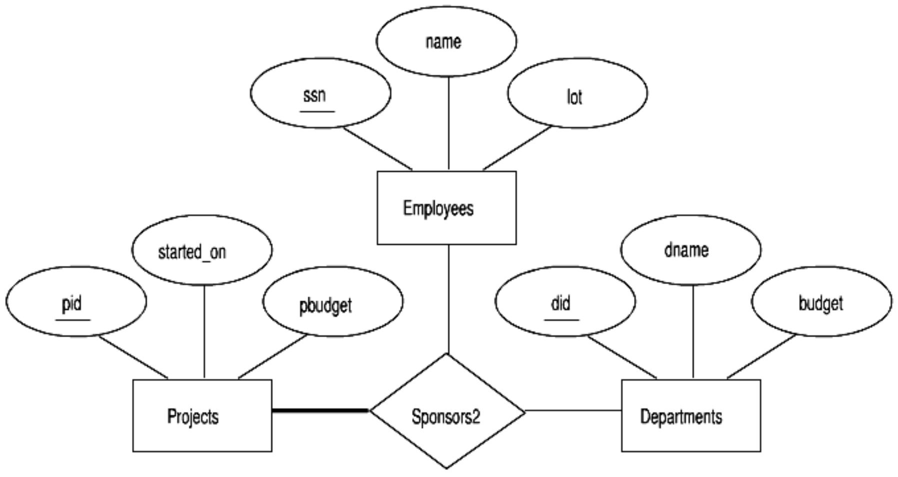
With binary relationships, it denotes that a supplier 'can supply' certain parts, that a department 'needs' some parts, or that a department 'deals with' a certain supplier. No combination of these
a relationship expresses the meaning of a contract adequately, for at least two reasons:

- The facts that supplier S can supply part P, that department D needs part P, and that D will buy from S do not necessarily imply that department D indeed buys part P from supplier S!
- We cannot represent the *qty* attribute of a contract cleanly.

1.13.4 Aggregation versus Ternary Relationships

The choice between using aggregation or a ternary relationship is mainly determined by the existence of a relationship that relates a *relationship set* to an entity set (or second relationship set). The choice may also be guided by certain integrity constraints that we want to express.
For example, consider the ER diagram shown in Figure 1.19. According to this diagram, a project can be sponsored by any number of departments, a department can sponsor one or more projects, and each sponsorship is monitored by one or more employees. If there is no need to record the *until* attribute of Monitors, then we can use a ternary relationship, say, Sponsors2, as shown in Figure1.25.

Consider the constraint that each sponsorship (of a project by a department) be monitored by at most one employee. It is not possible to represent this constraint in terms of the Sponsors2 relationship set. On the other hand, we can easily express the constraint by drawing an arrow from the aggregated relationship Sponsors to the relationship Monitors in Figure 1.19. Thus, the presence of such a constraint is another reason for using aggregation rather than a ternary relationship set.



Using a Ternary Relationship instead of Aggregation

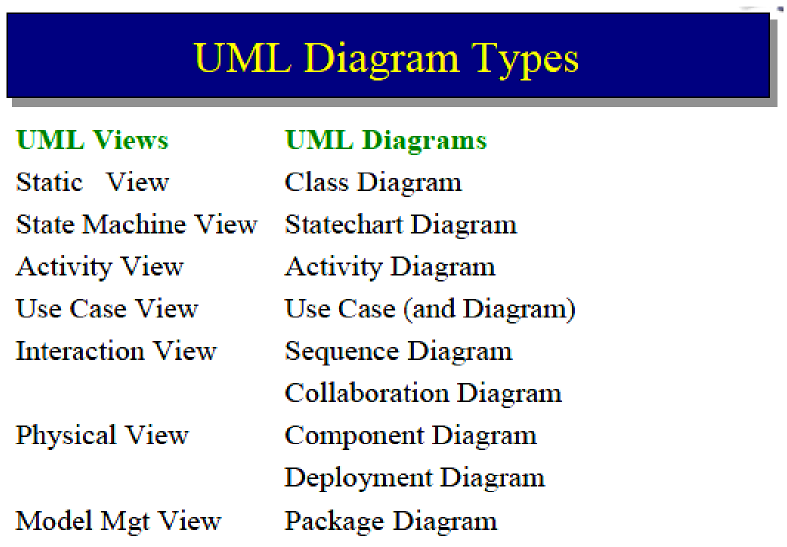
Figure: 1.26

1.14 Overview of Unified Modeling Languages

Unified Modeling Language ("UML") is the industry standard "language" for describing, visualizing, and documenting object-oriented (OO) systems. UML is a collection of a variety of diagrams for differing purposes. Each type of diagram models a particular aspect of OO design in an easy to understand, visual manner. The UML standard specifies exactly how the diagrams are to be drawn and what each component in the diagram means. UML is not dependent on any particular programming language, instead it focuses on the fundamental concepts and ideas that model a system. Using UML enables anyone familiar with its specifications to instantly read and understand diagrams drawn by other people. There are UML diagram for modeling static class relationships, dynamic temporal interactions between objects, the usages of objects, the particulars of an implementation, and the state transitions of systems

In general, a UML diagram consists of the following features:

- ***Entities***: These may be classes, objects, users or systems behaviors.
- ***Relationship Lines*** that model the relationships between entities in the system.
  - ***Generalization*** -- a solid line with an arrow that points to a higher abstraction of the present item.
  - ***Association*** -- a solid line that represents that one entity uses another entity as part of its behavior.
  - ***Dependency*** -- a dotted line with an arrowhead that shows one entity depends on the behavior of another entity.



UML Diagram Types

UML Views	UML Diagrams
Static View	Class Diagram
State Machine View	Statechart Diagram
Activity View	Activity Diagram
Use Case View	Use Case (and Diagram)
Interaction View	Sequence Diagram
	Collaboration Diagram
Physical View	Component Diagram
	Deployment Diagram
Model Mgt View	Package Diagram

There several types of UML diagrams available explain as follows:

- **Use-case Diagram:** Shows actors, use-cases, and the relationships between them.
- **Class Diagram:** Shows relationships between classes and pertinent information about classes themselves.
- **Object Diagram:** Shows a configuration of objects at an instant in time.
- **Interaction Diagrams:** Show an interaction between a group of collaborating objects.

Two types: Collaboration diagram and sequence diagram

- ❒ **Package Diagram:** Shows system structure at the library/package level.
- ❒ **State Diagram:** Describes behavior of instances of a class in terms of states, stimuli, and transitions.

- **Activity Diagram:** Very similar to a flowchart—shows actions and decision points, but with the ability to accommodate concurrency.
- **Deployment Diagram:** Shows configuration of hardware and software in a distributed system.

Class Diagrams

UML class diagrams model static class relationships that represent the fundamental architecture of the system. Note that these diagrams describe the relationships between *classes*, not those between specific *objects* instantiated from those classes. Thus the diagram applies to *all the objects* in the system.

A class diagram consists of the following features:

- ***Classes***: These titled boxes represent the classes in the system and contain information about the name of the class, fields, methods and access specifiers. Abstract roles of the class in the system can also be indicated.
- ***Interfaces***: These titled boxes represent interfaces in the system and contain information about the name of the interface and its methods.
- ***Relationship Lines*** that model the relationships between classes and interfaces in the system.
  - ***Generalization***
    - ***Inheritance***: a solid line with a solid arrowhead that points from a sub-class to a super class or from a sub-interface to its super-interface.
    - ***Implementation***: a dotted line with a solid arrowhead that points from a class to the interface that it implement
  - ***Association*** -- a solid line with an open arrowhead that represents a "has a" relationship. The arrow points from the containing to the contained class. Associations can be one of the following two types or not specified.
    - ***Composition***: Represented by an association line with a solid diamond at the tail end. A composition models the notion of one object "owning" another and thus being responsible for the creation and destruction of another object.
    - ***Aggregation***: Represented by an association line with a hollow diamond at the tail end. An aggregation models the notion that one object uses another object without "owning" it and thus is *not* responsible for its creation or destruction.
  - ***Dependency*** -- a dotted line with an open arrowhead that shows one entity depends on the behavior of another entity. Typical usages are to represent that one class instantiates another or that it uses the other as an input parameter.

Example:

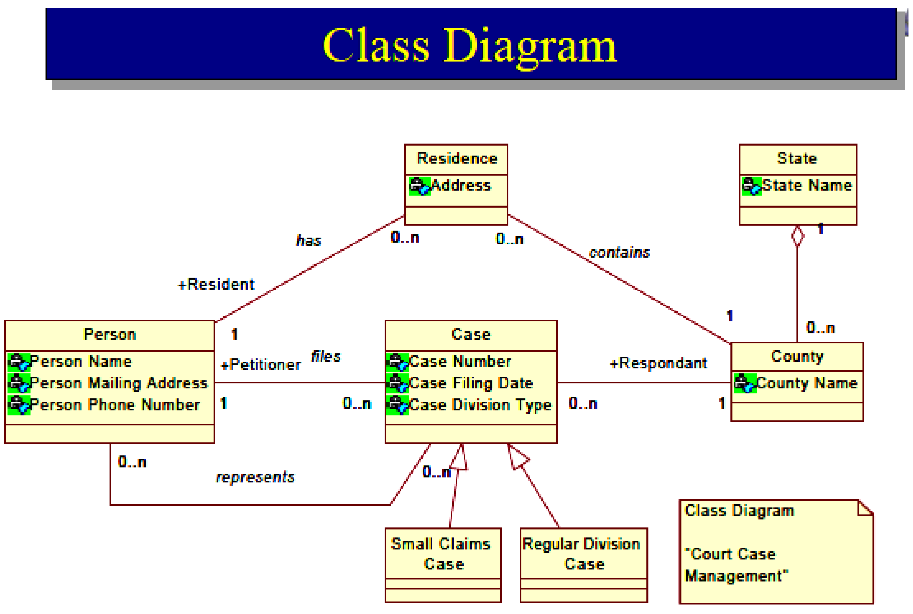


Figure: 1.27

The above diagram contains classes, interfaces, inheritance and implementation lines, aggregation lines, dependency lines and notes.

Consider a simple class to represent a point on a Cartesian plane:

```
class Point {
        private double x, y;
        Point(double x, double y) { this.x = x; this.y = y; }
        public double getX( ) { return x; }
        public double getY( ) { return y; }
        public double getDistFromOrigin( ) { ... }
        }
```

The corresponding UML class diagram:

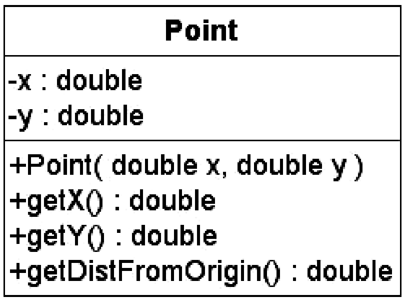


Figure: 1.28

Three *compartments* are shown: class name, attributes, and operations.