Module 1

Chapter 1: Introduction to Databases

1.1 Introduction

Databases and database technology have a major impact on the growing use ofcomputers. It is fair to say that databases play a critical role in almost all areas wherecomputers are used, including business, electronic commerce, engineering, medicine, genetics, law, education, and library science.

Database

A **database** is a collection of related data.1 By**data**,we mean known facts that can berecorded and that have implicit meaning. For example, consider the names, telephone numbers, and addresses of the people you know.

A database has the following implicit properties:

- It represents some aspect of the real world, sometimes called the **miniworld**or the **universe of discourse (UoD)**. Changes to the miniworld are reflected in the database.
- It is a logically coherent collection of data, to which some meaning can be attached.
- It is designed, built, and populated with data for a specific purpose.It has an intended group of users and some preconceived applications in which these users are interested.

To summarize: a database has some source (i.e., the miniworld) from which data are derived, some degree of interaction with events in the represented miniworld and an audience that is interested in using it.

Size/Complexity:A database can be of any size and complexity. For example, the list of names andaddresses referred to earlier may consist of only a few hundred records, each with asimple structure. An example of a large commercial database is Amazon.com. It contains data forover 20 million books, CDs, videos, DVDs, games, electronics, apparel, and other items.

Computerized vs. manual: A database may be generated and maintained manually or it may be computerized.For example, simple database like telephone directory may be created and maintained manually. Huge and complex database may be created and maintained either by a group of application programs written specifically for that task or by a database management system.

Database Management System (DBMS)

A **database management system** (DBMS) is a collection of programs enabling users to create and maintain a database. More specifically, The DBMS is a general-purpose software systemthat facilitates the processes of defining, constructing, manipulating, and sharing databases among various users and applications.

- **Defining** a database involves specifying the data types, structures, and constraints of the data to be stored in the database. The database definition or descriptive information is stored by the DBMS in the form of a database catalog or dictionary; it is called **meta-data**.
- **Constructing** the database is the process of storing the data on some storage medium that is controlled by the DBMS.
- **Manipulating** a database includes functions such as querying the database to retrieve specific data, updating the database to reflect changes in the miniworld, and generating reports from the data.
- **Sharing** a database allows multiple users and programs to access the database simultaneously.

Other important functions provided by the DBMS include protecting the database and maintaining it over a long period of time.

- **Protection** includes ***system protection***against hardware or software malfunction (or crashes) and ***security protection***against unauthorized or malicious access.
- A typical large database may have a life cycle of many years, so the DBMS must be able to **maintain** the database system by allowing the system to evolve as requirements change over time.

A database together with the DBMS software is referred to as a **database system.**

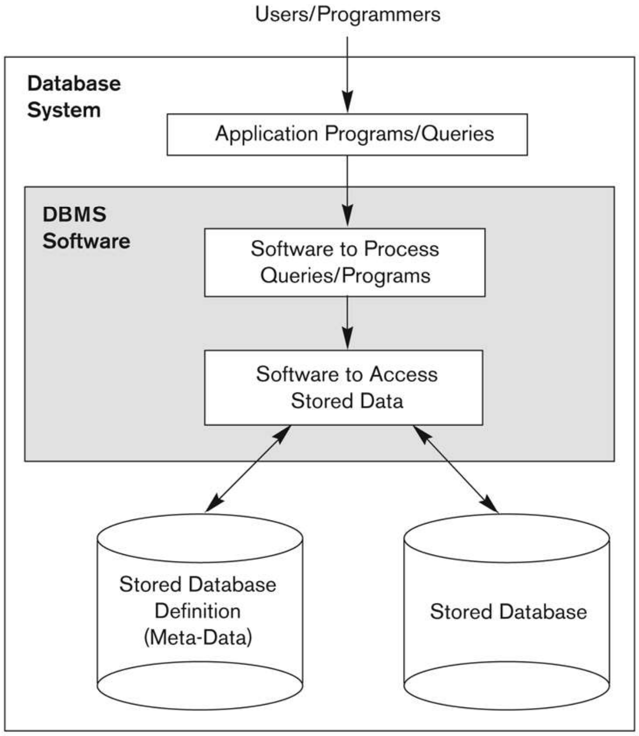
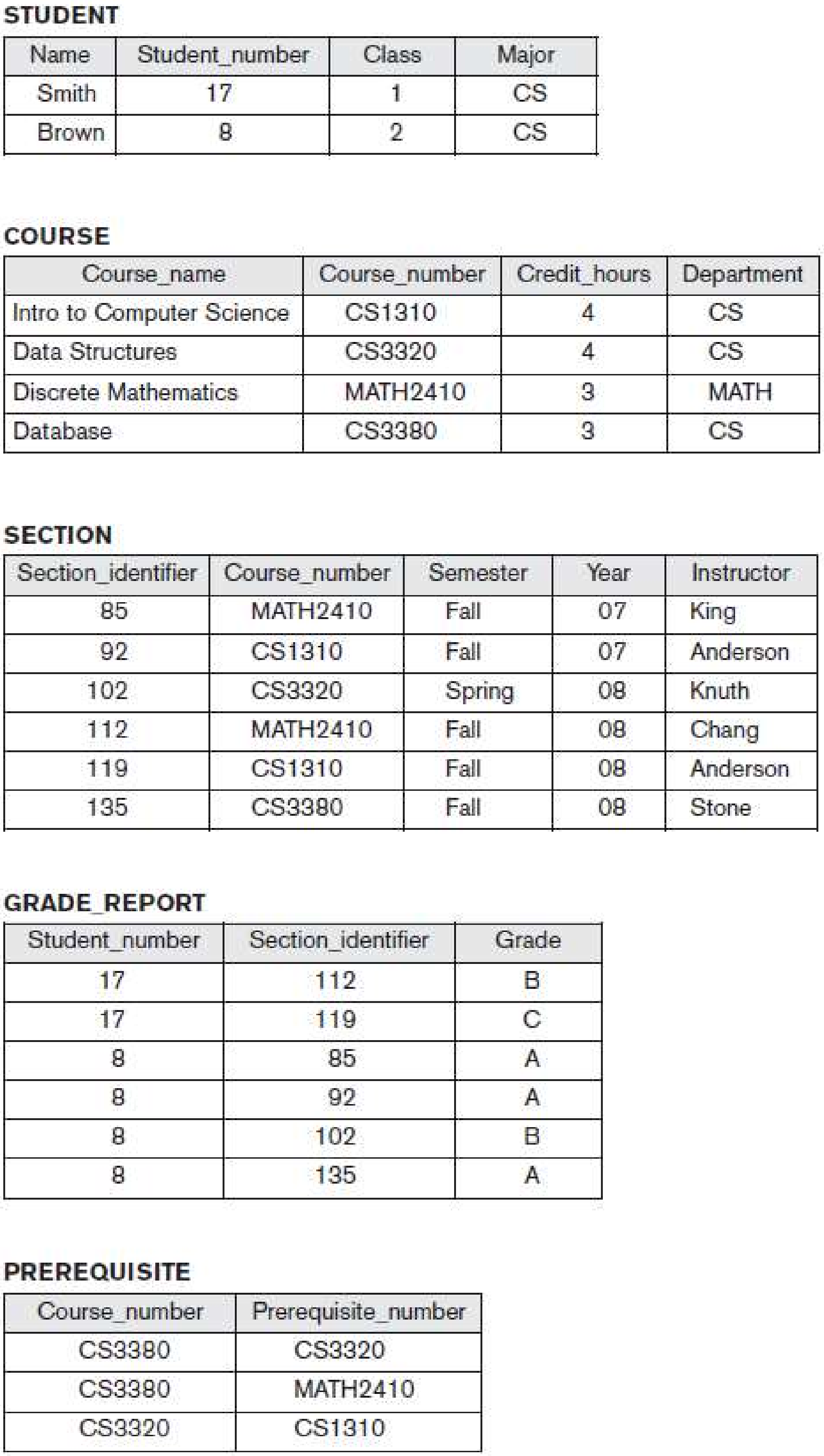

Users/Programmers
Database System
Application Programs/Queries
DBMS Software
Software to Process Queries/Programs
Software to Access Stored Data
Stored Database Definition (Meta-Data)
Stored Database

Fig 1.1(a): A simplified database system environment

An Example

Consider a UNIVERSITY database for maintaining information concerning students, courses, and grades in a university environment. The database is organized as five files, each of which stores **data records** of the same type.

1. STUDENT file: stores data on each student.
2. COURSE file: stores data on each course.
3. SECTION file: stores data on each section of a course.
4. GRADE_REPORT file: stores the grades that students receive in the various sections they have completed.
5. PREREQUISITE file :stores the prerequisites of each course.



STUDENT

Name	Student_number	Class	Major
Smith	17	1	CS
Brown	8	2	CS

COURSE

Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Database	CS3380	3	CS

SECTION

Section_identifier	Course_number	Semester	Year	Instructor
85	MATH2410	Fall	07	King
92	CS1310	Fall	07	Anderson
102	CS3320	Spring	08	Knuth
112	MATH2410	Fall	08	Chang
119	CS1310	Fall	08	Anderson
135	CS3380	Fall	08	Stone

GRADE_REPORT

Student_number	Section_identifier	Grade
17	112	B
17	119	C
8	85	A
8	92	A
8	102	B
8	135	A

PREREQUISITE

Course_number	Prerequisite_number
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310

Fig 1.1(b): A database that stores student and course information

Defining a UNIVERSITY database

- Specify the structure of the records of each file - **data elements** to be stored in each record.For example: each STUDENT record includes data to represent the student's Name, Student_number, Class Major. Similarly each COURSE record includes data to represent the Course_name, Course_number, Credit_hours, and Department.
- Specify a data type for each data element within a record. For example: student's Name isa string of alphabetic charactersStudent_number is an integer.

Constructing the UNIVERSITY database

- To *construct* the UNIVERSITY database, we store data to represent each student,course, section, grade report, and prerequisite as a record in the appropriate file.
- Records in the various files may be related. For example, the record forSmith in the STUDENT file is related to two records in the GRADE_REPORT file thatspecify Smith's grades in two sections. Similarly, each record in the PREREQUISITEfile relates two course records: one representing the course and the other representing the prerequisite.

Manipulating a UNIVERSITY database

Database *manipulation* involves querying and updating.

Examples of queries are asfollows:

- Retrieve the transcript—a list of all courses and grades—of 'Smith'
- List the names of students who took the section of the 'Database' course offered in fall 2008 and their grades in that section
- List the prerequisites of the 'Database' course

Examples of updates include the following:

- Change the class of 'Smith' to sophomore
- Create a new section for the 'Database' course for this semester
- Enter a grade of 'A' for 'Smith' in the 'Database' section of last semester

These informal queries and updates must be specified precisely in the query language of the DBMS before they can be processed.

As with software in general, design of a new application for an existing database or design of abrand new database starts off with a phase called **requirements specification and analysis**. These requirements are documented in detail and transformed into a**conceptual design** that can be represented and manipulated using some computerizedtools so that it can be easily maintained, modified, and transformed into a databaseimplementation.

The design is then translated to a**logical design** that can be expressed in a data model implemented in a commercialDBMS. The final stage is**physical design**, during which further specifications are provided for storing andaccessing the database. The database design is implemented, populated with actualdata, and continuously maintained to reflect the state of the miniworld.

1.2 Characteristics of the Database Approach

Database approach vs. File Processing approach

Consider an organization that is organized as a collection of departments/offices. Each department has certain data processing "needs", many of which are unique to it.

In the file processing approach, each department would control a collection of relevant data files and software applications to manipulate that data. For example, one user, the grade reporting office, maykeep files on students and their grades. Programs to print a student’s transcript andto enter new grades are implemented as part of the application. A second user, theaccounting office, may keep track of students’ fees and their payments. Althoughboth users are interested in data about students, each user maintains separate files—and programs to manipulate these files—because each requires some data not available from the other user’s files.This redundancy in defining and storing data resultsin wasted storage space and in redundant efforts to maintain common up-to-datedata.

In the database approach, a single repository maintains data that is defined onceand then accessed by various users. In file systems, each application is free to namedata elements independently. In contrast, in a database, the names or labels of dataare defined once, and used repeatedly by queries, transactions, and applications.

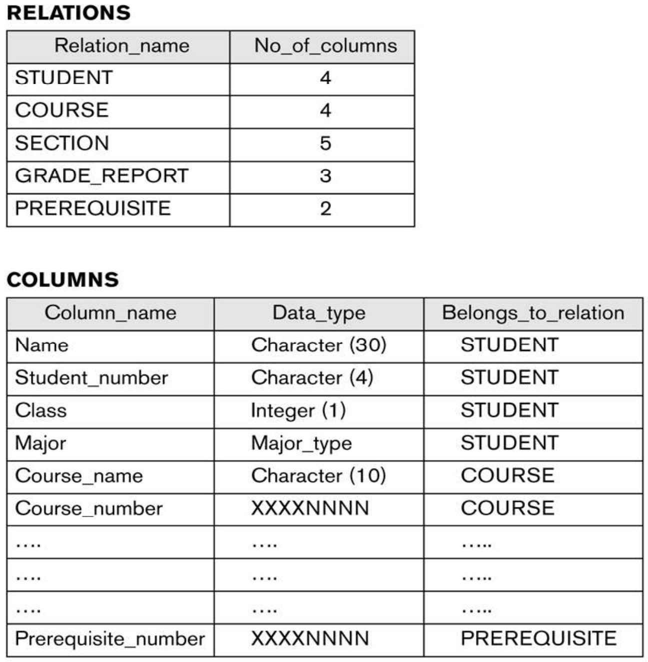
The main characteristics of the database approach versus the file-processingapproach are the following:

- Self-describing nature of a database system
- Insulation between programs and data, and data abstraction
- Support of multiple views of the data
- Sharing of data and multiuser transaction processing

1. Self-Describing Nature of a Database System

A fundamental characteristic of the database approach is that the database system contains not only the database itself but also a complete definition or description of the database structure and constraints.This **meta-data** (i.e., data about data) is stored in the so-called **system catalog**, which contains a description of the structure of each file, the type and storage format of each field, and the various constraints on the data (i.e., conditions that the data must satisfy).

The system catalog is used not only by users but also by the DBMS software, which certainly needs to "know" how the data is structured/organized in order to interpret it in a manner consistent with that structure.



RELATIONS

Relation_name	No_of_columns
STUDENT	4
COURSE	4
SECTION	5
GRADE_REPORT	3
PREREQUISITE	2

COLUMNS

Column_name	Data_type	Belongs_to_relation
Name	Character (30)	STUDENT
Student_number	Character (4)	STUDENT
Class	Integer (1)	STUDENT
Major	Major_type	STUDENT
Course_name	Character (10)	COURSE
Course_number	XXXXNNNN	COURSE
….	….	…..
….	….	…..
….	….	…..
Prerequisite_number	XXXXNNNN	PREREQUISITE

Figure 1.2(a): An example of a database catalog for the database

2. Insulation between Programs and Data, and Data Abstraction

Program-Data Independence: In traditional file processing, the structure of the data files accessed by an application is "hard-coded" in its source code. If, for some reason, we decide to change the structure of the data ,everyapplication in which a description of that file's structure is hard-coded must be changed!

In contrast, DBMS access programs, in most cases, do not require such changes, because the structure of the data is described separately from the programs that access it and those programs consult the catalog in order to ascertain the structure of the data so that they interpret that data properly.

In other words, the DBMS provides a conceptual or logical view of the data to application programs, so that the underlying implementation may be changed without the programs being modified. (This is referred to as *program-data independence*.)

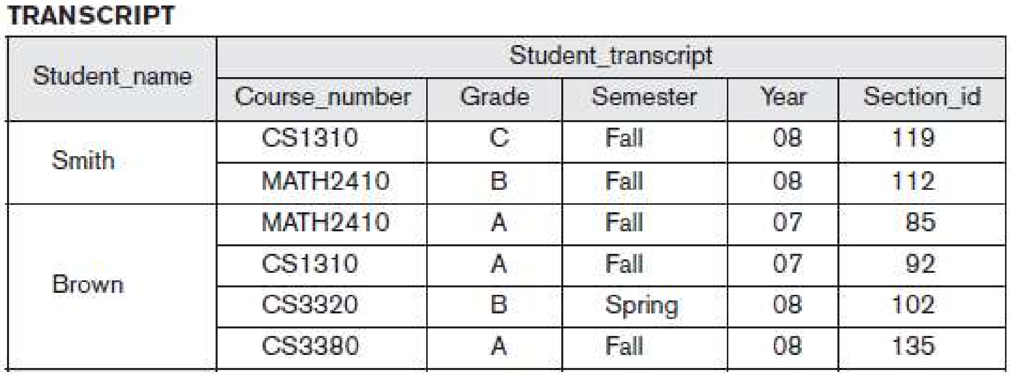
Program-operation independence: In object-oriented and object-relationalsystems , users can define operations on data as part of the databasedefinitions.An **operation** (also called a *function* or *method*) is specified in two parts.The *interface* (or *signature*) of an operation includes the operation name and thedata types of its arguments (or parameters). The *implementation* (or *method*) of theoperation is specified separately and can be changed without affecting the interface.User application programs can operate on the data by invoking these operationsthrough their names and arguments, regardless of how the operations are implemented.This may be termed **program-operation independence**.

Data abstraction

The characteristic that allows program-data independence and program-operationindependence is called **data abstraction**. A DBMS provides users with a conceptual representationof data that does not include many of the details of how the data isstored or how the operations are implemented. Informally, a **data model** is a type ofdata abstraction that is used to provide this conceptual representation. The datamodel uses logical concepts, such as objects, their properties, and their interrelationships,that may be easier for most users to understand than computer storageconcepts. Hence, the data model *hides* storage and implementation details that arenot of interest to most database users.

3. Support of Multiple Views of the Data

A database typically has many users, each of whom may require a different perspectiveor **view**of the database. A view may be a subset of the database or it may contain**virtual data** that is derived from the database files but is not explicitly stored. A multiuser DBMS whose users have a variety of distinct applications mustprovide facilities for defining multiple views. For example, one user of the databaseof Figure 1.2 may be interested only in accessing and printing the transcript of eachstudent; the view for this user is shown in Figure 1.2(b)



TRANSCRIPT

Student_name	Student_transcript				
	Course_number	Grade	Semester	Year	Section_id
Smith	CS1310	C	Fall	08	119
	MATH2410	B	Fall	08	112
Brown	MATH2410	A	Fall	07	85
	CS1310	A	Fall	07	92
	CS3320	B	Spring	08	102
	CS3380	A	Fall	08	135

Fig1.2(b): view derived from the university database

4. **Sharing of Data and Multiuser Transaction Processing**

A multiuser DBMS, as its name implies,must allow multiple users to access the database at the same time. This is essential if data for multiple applications is to be integratedand maintained in a single database. The DBMS must include **concurrencycontrol** software to ensure that several users trying to update the same data do so ina controlled manner so that the result of the updates is correct. For example, whenseveral reservation agents try to assign a seat on an airline flight, the DBMS shouldensure that each seat can be accessed by only one agent at a time for assignment to apassenger. These types of applications are generally called **online transaction processing(OLTP)** applications. A fundamental role of multiuser DBMS software is toensure that concurrent transactions operate correctly and efficiently.

The concept of a **transaction** has become central to many database applications. Atransaction is an *executing program* or *process* that includes one or more databaseaccesses, such as reading or updating of database records. The DBMS must enforce several transaction properties. The **isolation** property ensures that each transaction appears to executein isolation from other transactions, even though hundreds of transactions may beexecuting concurrently. The **atomicity** property ensures that either all the databaseoperations in a transaction are executed or none are.

Database Users

Users may be divided into

- Those who actually use and control the database content, and those who design, develop and maintain database applications called **"Actors on the Scene"**
- Those who design and develop the DBMS software and related tools, and the computer systems operators called **"Workers Behind the Scene"**

Actors on the Scene

1. **Database Administrator** (DBA): chief administrator, who oversees and manages the database system (including the data and software). Duties include authorizing users to access the database, coordinating/monitoring its use, acquiring hardware/software for upgrades, etc.The DBA is accountable for problems such as securitybreaches and poor system response time.In large organizations, the DBA might have a support staff.
2. **Database Designers**: responsible for identifying the data to be stored and for choosing an appropriate way to organize it. Database designers typically interact with each potential groupof users and develop **views** of the database that meet the data and processingrequirements of these groups.The final database design must be capable of supportingthe requirements of all user groups.
3. **End Users**: These are persons who access the database for querying, updating, and report generation. Thereare several categories of end users:
   - **Casual end users:** use database occasionally, needing different information each time; use query language to specify their requests; typically middle- or high-level managers.
   - **Naive/Parametric end users**: biggest group of users; frequently query/update the database using standard **canned transactions** that have been carefully programmed and tested in advance. Examples:
     - bank tellers check account balances, post withdrawals/deposits
     - reservation clerks for airlines, hotels, etc., check availability of seats/rooms and make reservations.
   - **Sophisticated end users**: include engineers, scientists, business analysts, and others who thoroughly familiarize themselves with the facilities of the DBMS in order to implement their own applications to meet their complex requirements.

- **Stand-alone users**: maintain personal databases by using ready-made programpackages that provide easy-to-use menu-based or graphics-based interfaces.

  Ex: user of a tax package that stores a variety of personalfinancial data
  for tax purposes.

4. **System Analysts and Application Programmers (Software Engineers**)
   - **System Analysts**: determine needs of end users, especially naive and parametric users, and develop specifications for canned transactions that meet these needs.
   - **Application Programmers**: Implement, test, document, and maintain programs that satisfy the specifications mentioned above.

Workers behind the Scene

1. **DBMS system designers and implementers:** design and implement the DBMS modules and interfaces as a software package. A DBMS is a very complex software system that consists of many components, or **modules**, including modules for implementing the catalog, query language processing, interface processing, accessing and buffering data, controlling concurrency, and handling data recovery and security.
2. **Tool developers:** design and implement **tools** that facilitate database modeling and design, database system design, and improved performance.
3. **Operators and maintenance personnel** (system administration personnel) : responsible for the actual running and maintenance of the hardware and software environment for the database system.

1.3 Advantages of Using the DBMS Approach

1. Controlling Redundancy

Data redundancy such as tends to occur in the "file processing" approach leads to **wasted storage space**, **duplication of effort** and a higher likelihood of the introduction of **inconsistency**.

In the database approach, the views of different user groups are integrated during database design. This is known as **data normalization**, and it ensures consistency and saves storage

Space. However, it is sometimes necessary to use **controlled redundancy** to improve the performance of queries. For example, we may store Student_name and Course_number redundantly in a GRADE_REPORT file because whenever we retrieve a GRADE_REPORT record, we want to retrieve the student name and course number along with the grade, student number, and section identifier.

A DBMS should provide the capability to automatically enforce the rule that no inconsistencies are introduced when data is updated.

2. **Restricting Unauthorized Access**

   When multiple users share a large database, it is likely that most users will not be authorized to access all information in the database. For example, financial data is often considered confidential and only authorized persons are allowed to access such data. In addition, some users may only be permitted to retrieve data, whereas others are allowed to retrieve and update. Hence, the type of access operation—retrieval or update—must also be controlled. A DBMS should provide a **security and authorization subsystem**, which the DBA uses to create accounts, to specify account restrictions and enforce these restrictions automatically.

3. **Providing Persistent Storage for Program Objects**

   The values of program variables or objects are discarded once a program terminates, unless the programmer explicitly stores them in permanent files, which often involves converting these complex structures into a format suitable for file storage. Object-oriented database systems make it easier for complex runtime objects to be saved in secondary storage so as to survive beyond program termination and to be retrievable at a later time.

   Object-oriented database systems are compatible with programming languages such as C++ and Java, and the DBMS software automatically performs any necessary conversions.

4. **Providing Storage Structures and Search Techniques for Efficient Query Processing**

   DBMS maintains indexes that are utilized to improve the execution time of queries and updates. DBMS has a buffering or caching module that maintains parts of the database in main memory buffers.The query processing and optimization module is responsible for choosing an efficient query execution plan for each query submitted to the system**.**

5. **Providing Backup and Recovery**

   The backup and recovery subsystem of the DBMS is responsible for recovery. For example, if the computer system fails in the middle of a complex update transaction, the recovery subsystem is responsible for making sure that the database is restored to the state it was in before the transaction started executing. Disk backup is also necessary in case of a catastrophic disk failure.

6. **Providing Multiple User Interfaces**

   Because many types of users with varying levels of technical knowledge use a database, a DBMS should provide a variety of user interfaces. These include

   - Query languages for casual users
   - Programming language interfaces for application programmers
   - Forms and command codes for parametric users
   - Menu-driven interfaces and natural language interfaces for standalone users.

7. **Representing Complex Relationships among Data**

   A database may include numerous varieties of data that are interrelated in many ways.

   For example each section record is related to one course record and to a number of GRADE_REPORT records—one for each student who completed that section. A DBMS must have the capability to represent a variety of complex relationships among the data, to define new relationships as they arise, and to retrieve and update related data easily and efficiently.

8. **Enforcing Integrity Constraints**

   Most database applications are such that the semantics of the data require that it satisfy certain restrictions in order to make sense.

The simplest type of integrity constraint involves specifying a data type for each data item.

For example, in student table we specified that the value of Name must be a string of no more than 30 alphabetic characters.

More complex type of constraint is **referential integrity** involves specifying that a record in one file must be related to records in other files. For example, in university database, we can specify that every section record must be related to a course record.

Another type of constraint specifies uniqueness on data item values, such as every course record must have a unique value for Course_number. This is known as a key or **uniqueness constraint**.

It is the responsibility of the database designers to identify integrity constraints during database design.

9. Permitting Inferencing and Actions Using Rules

In a **deductive** database system, one may specify *declarative* rules that allow the database to infer new data. For example, figure out which students are on academic probation. Such capabilities would take the place of application programs that would be used to ascertain such information otherwise.

Active database systems go one step further by allowing "active rules" that can be used to initiate actions automatically. In today's relational database systems, it is possible to associate triggers with tables.

10. Additional Implications of Using the Database Approach

- **Potential for Enforcing Standards :** database approach permits the DBA to define and enforce standards among database users in a large organization which facilitates communication and cooperation among various departments, projects, and users within the organization.Standards can be defined for names and formats of data elements, display formats, report structures and so on.
- **Reduced Application Development Time:** once a database is up and running, substantially less time is generally required to create new applications using DBMS facilities. Development time using a DBMS is estimated to be one-sixth to one-fourth of that for a traditional file system.
- **Flexibility:** It may be necessary to change the structure of a database as requirements change. DBMSs allow changes to the structure of the database without affecting the stored data and the existing application programs.
- **Availability of Up-to-Date Information:** DBMS makes the database available to all users. Availability of up-to-date information is essential for many transaction-processing applications, such as reservation systems or banking databases
- **Economies of Scale:** DBMS approach permits consolidation of data and applications, to overlap between activities of data-processing in different projects or departments. This enables the whole organization to invest in more powerful processors, storage devices, or communication gear, rather than having each department purchase its equipment thus reducing overall costs of operation and management.

1.4 History of Database Applications

- **Early Database Applications Using Hierarchical and Network Systems**

  Early database applications maintained records in large organizations such as corporations, universities, hospitals, and banks. In many of these applications, there were large numbers of records of similar structure. There were also many types of records and many interrelationships among them.

  Problems with the early database systems

  - lack of data abstraction and program-data independence capabilities
  - provided only programming language interfaces. This made it time-consuming and expensive to implement new queries and transactions, since new programs had to be written, tested, and debugged.

- **Providing Data Abstraction and Application Flexibility with Relational Databases**

  Relational databases were originally proposed to separate the physical storage of data from its conceptual representation and to provide a mathematical foundation for data representation and querying. The relational data model also introduced high-level query languages that provided an alternative to programming language interfaces, making it much faster to write new queries. Hence, data abstraction and program-data independence were much improved when compared to earlier systems.

- **Object-Oriented Applications and the Need for More Complex Databases**

  Object-oriented databases (OODBs) mainly used in specialized applications, such as engineering design, multimedia publishing, and manufacturing systems. In addition, many object-oriented concepts were incorporated into the newer versions of relational DBMSs, leading to object-relational database management systems, known as ORDBMSs.

- **Interchanging Data on the Web for E-Commerce Using XML**

  The World Wide Web provides a large network of interconnected computers. Users can create documents using a Web publishing language, such as HyperText Markup Language (HTML), and store these documents on Web servers where other users (clients) can access them. Documents can be linked through **hyperlinks**, which are pointers to other documents.

Currently, eXtended Markup Language (XML) is considered to be the primary standard for interchanging data among various types of databases and Web pages. XML combines concepts from the models used in document systems with database modeling concepts.

- **Extending Database Capabilities for New Applications**

The success of database systems in traditional applications encouraged developers of other types of applications to attempt to use them. The following are some examples of these applications:

- **Scientific** applications that store large amounts of data resulting from Scientific experiments in areas such as high-energy physics, the mapping of the human genome, and the discovery of protein structures.
- Storage and retrieval of **images**, including scanned news or personal photographs, satellite photographic images, and images from medical procedures such as x-rays and MRIs (magnetic resonance imaging).
- Storage and retrieval of **videos,** such as movies, and **video clips** from news or personal digital cameras.
- **Data mining** applications that analyze large amounts of data searching for the occurrences of specific patterns or relationships, and for identifying unusual patterns in areas such as credit card usage.
- **Spatial** applications that store spatial locations of data, such as weather information, maps used in geographical information systems, and in automobile navigational systems.
- **Time series** applications that store information such as economic data at regular points in time, such as daily sales and monthly gross national product figures.

- **Databases versus Information Retrieval**

Database technology is heavily used in manufacturing, retail, banking, insurance, finance, and health care industries, where structured data is collected through forms, such as invoices or patient registration documents. An area related to database technology is **Information Retrieval (IR)**, which deals with books, manuscripts, and various forms of library-based articles. Data is indexed, cataloged, and annotated using keywords. IR is concerned with searching for material based on these keywords, and with the many problems dealing with document processing and free-form text processing.

When Not to Use a DBMS

- DBMS may involve unnecessary overhead costs that would not be incurred in traditional file processing. The overhead costs of using a DBMS are due to the following:
    - High initial investment in hardware, software, and training
    - The generality that a DBMS provides for defining and processing data
    - Overhead for providing security, concurrency control, recovery, and integrity functions
- Therefore, it may be more desirable to use regular files under the following circumstances:
    - Simple, well-defined database applications that are not expected to change at all
    - Stringent, real-time requirements for some application programs that may not be met because of DBMS overhead
    - Embedded systems with limited storage capacity, where a general-purpose DBMS would not fit
    - No multiple-user access to data

Chapter 2: Overview of Database Languages and Architectures

Introduction

The architecture of DBMS packages has evolved from the early monolithic systems, where the whole DBMS software package was one tightly integrated system. Modern DBMS packages are modular in design, with a client/server system architecture. In a basic client/server DBMS architecture, the system functionality is distributed between two types of module. A **client module** is designed to run on a user workstation or personal computer. The

client module handles user interaction and provides the user-friendly interfaces such as forms- or menu-based GUIs. The other kind of module, called a **server module** handles data storage, access, search, and other functions

2.1 Data Models, Schemas, and Instances

Data Model

A data model is a collection of concepts that can be used to describe the structure of a database. By structure of a database we mean the data types, relationships and constraints that apply to the data. Most data models also include a set of basic operations for specifying retrievals and updates on the database. Data model provides the necessary means to achieve abstraction.

Categories of Data Models

Data models can be categorized according to the types of concepts they use to describe the database structure.

1. **High-level** or **conceptual data models:** provide concepts that are close to the way many users perceive data. Conceptual data models use concepts such as entities, attributes, and relationships.
2. **Representational** or **implementation data models**: provide concepts that may be easily understood by end users but that are not too far removed from the way data is organized in computer storage. Representational data models hide many details of data storage on disk but can be implemented on a computer system directly. Representational or implementation data models are the models used most frequently

   in traditional commercial DBMSs. These include the widely used **relational data model**, as well as the so-called legacy data models—the **network** and **hierarchical**

models. Representational data models represent data by using record structures and hence are sometimes called **record-based data models**.

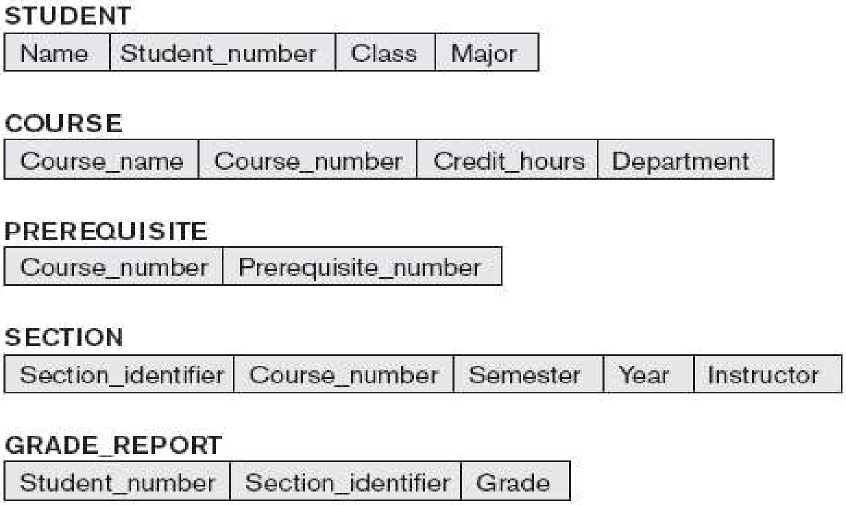
3. **Low-level** or **physical data models:** provide concepts that describe the details of how data is stored on the computer storage media, typically magnetic disks. Physical data models describe how data is stored as files in the computer by representing information such as record formats, record orderings, and access paths.

Database schema

The description of a database is called the **database schema**, which is specified during database design and is not expected to change frequently.

Schema diagram

A displayed schema is called a **schema diagram**. A schema diagram displays only some aspects of a schema, such as the names of record types and data items, and some types of constraints.



STUDENT

Name	Student_number	Class	Major

COURSE

Course_name	Course_number	Credit_hours	Department

PREREQUISITE

Course_number	Prerequisite_number

SECTION

Section_identifier	Course_number	Semester	Year	Instructor

GRADE_REPORT

Student_number	Section_identifier	Grade

Figure 2.1: Schema diagram for the database

Schema construct

Each object in the schema is called schema construct. For example student or course.

Database state or **snapshot**

The data in the database at a particular moment in time is called a **database state** or **snapshot**. It is also called the current set of **occurrences** or **instances** in the database. In a given database state, each schema construct has its own current set of instances; for example, the STUDENT construct will contain the set of individual student entities (records) as its instances.

The distinction between database schema and database state is very important. When we **define** a new database, we specify its database schema only to the DBMS. At this point, the corresponding database state is the *empty state* with no data. We get the *initial state* of the database when the database is first **populated** or **loaded** with the initial data. From then on, every time an update operation is applied to the database, we get another database state. At any point in time, the database has a *current state*.

The DBMS is partly responsible for ensuring that every state of the database is a **valid state**—that is, a state that satisfies the structure and constraints specified in the schema. The

DBMS stores the descriptions of the schema constructs and constraints—also called the **meta-data**—in the DBMS catalog so that DBMS software can refer to the schema whenever it needs to. The schema is sometimes called the **intension**, and a database state is called an **extension** of the schema.

2.2 Three-Schema Architecture and Data Independence

The Three-Schema Architecture

The goal of the three-schema architecture is to separate the user applications from the physical database. In this architecture, schemas can be defined at the following three levels:

1. The **internal level** has an **internal schema**, which describes the physical storage structure of the database. The internal schema uses a physical data model and describes the complete details of data storage and access paths for the database.
2. The **conceptual level** has a **conceptual schema**, which describes the structure of the whole database for a community of users. The conceptual schema hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations, and constraints. Usually, a representational data model is used to describe the conceptual schema when a database system is implemented.
3. The **external** or **view level** includes a number of **external schemas** or **user views**. Each external schema describes the part of the database that a particular user group is interested in and hides the rest of the database from that user group. Each external schema is typically

implemented using a representational data model, possibly based on an external schema design in a high-level data model.

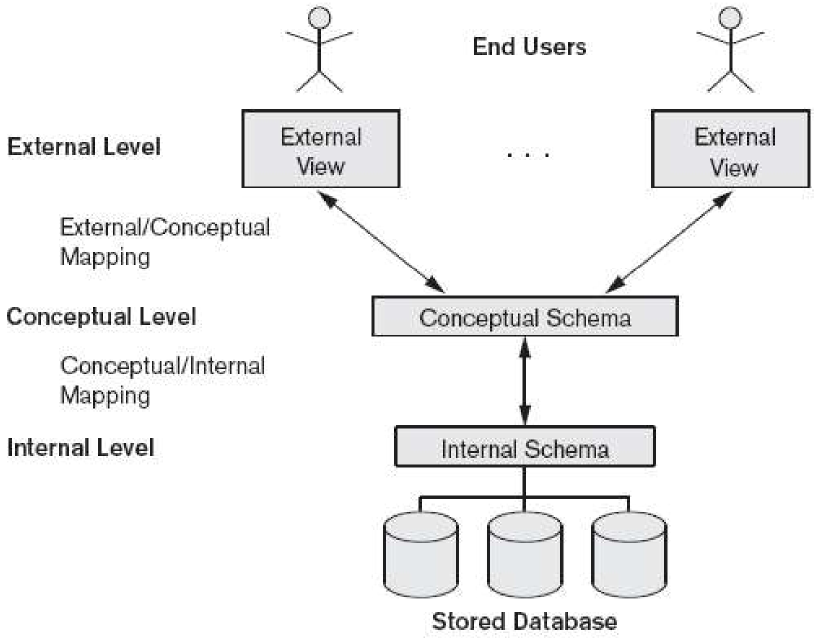


Figure 2.2: The three-schema architecture.

In a DBMS based on the three-schema architecture, each user group refers to its own external schema. Hence, the DBMS must transform a request specified on an external schema into a request against the conceptual schema, and then into a request on the internal schema for processing over the stored database. If the request is a database retrieval, the data extracted
from the stored database must be reformatted to match the user's external view.
The processes of transforming requests and results between levels are called **mappings**.

Data Independence

Data independence can be defined as the capacity to change the schema at one level of a database system without having to change the schema at the next higher level. We can define two types of data independence:

1. **Logical data independence** is the capacity to change the conceptual schema without having to change external schemas or application programs. We may change the conceptual schema to expand the database, to change constraints, or to reduce the database. Only the view definition and the mappings need to be changed in a DBMS that supports logical data independence.

2. **Physical data independence** is the capacity to change the internal schema without having to change the conceptual schema. Hence, the external schemas need not be changed as well. Changes to the internal schema may be needed because some physical files were reorganized—for example, by creating additional access structures—to improve the performance of retrieval or update.

Data independence occurs because when the schema is changed at some level, the schema at the next higher level remains unchanged; only the *mapping* between the two levels is changed.

2.3 Database Languages and Interfaces

The DBMS must provide appropriate languages and interfaces for each category of users.

DBMS Languages

Once the design of a database is completed and a DBMS is chosen to implement the database, the first step is to specify conceptual and internal schemas for the database and any mappings between the two.

Data Definition Language (DDL)

The **data definition language (DDL**) is used by the DBA and by database designers to define both schemas when no strict separation of levels is maintained . The DBMS will have a DDL compiler whose function is to process DDL statements in order to identify descriptions of the schema constructs and to store the schema description in the DBMS catalog.

Storage Definition Language (SDL)

Storage definition language is used when clear separation is maintained between the conceptual and internal levels, the DDL is used to specify the conceptual schema only.
The **storage definition language** (**SDL**), is used to specify the internal schema. The mappings between the two schemas may be specified in either one of these languages.

View Definition Language (VDL),

View definition language is used to specify user views and their mappings to the conceptual schema. In relational DBMSs, SQL is used in the role of VDL to define user or application **views** as results of predefined queries.

Data Manipulation Language (DML)

Data manipulation languages (DML) are used to perform manipulation operation such as retrieval, insertion, deletion, and modification of the data. There are two main types of DMLs :

1. **High-level** or **nonprocedural** DML : can be used on its own to specify complex database operations concisely. Many DBMSs allow high-level DML statements either to be entered interactively from a display monitor or terminal or to be embedded in a general-purpose programming language. In the latter case, DML statements must be identified within the program so that they can be extracted by a precompiler and processed by the DBMS. High-level DMLs, such as SQL, can specify and retrieve many records in a single DML statement; therefore, they are called **set-at-a-time** or **set-oriented** DMLs. A query in a high-level DML often specifies *which* data to retrieve rather than *how* to retrieve it; therefore, such languages are also called **declarative**
2. **Low-level** or **procedural** DML: must be embedded in a general-purpose programming language. This type of DML typically retrieves individual records or objects from the database and processes each separately. language constructs, such as looping, to retrieve and process each record from a set of records. Low-level DMLs are also called **record-at-a-time** DMLs because of this property. DL/1, a DML designed for the hierarchical model, is a low-level DML that uses commands such as GET UNIQUE, GET NEXT, or GET NEXT WITHIN PARENT to navigate from record to record within a hierarchy of records in the database.

Host language

Whenever DML commands, whether high level or low level, are embedded in a general-purpose programming language, that language is called the **host language** and the DML is called the **data sublanguage**.

A high-level DML used in a standalone interactive manner is called a **query language**.

.

DBMS Interfaces

User-friendly interfaces provided by a DBMS may include the following:

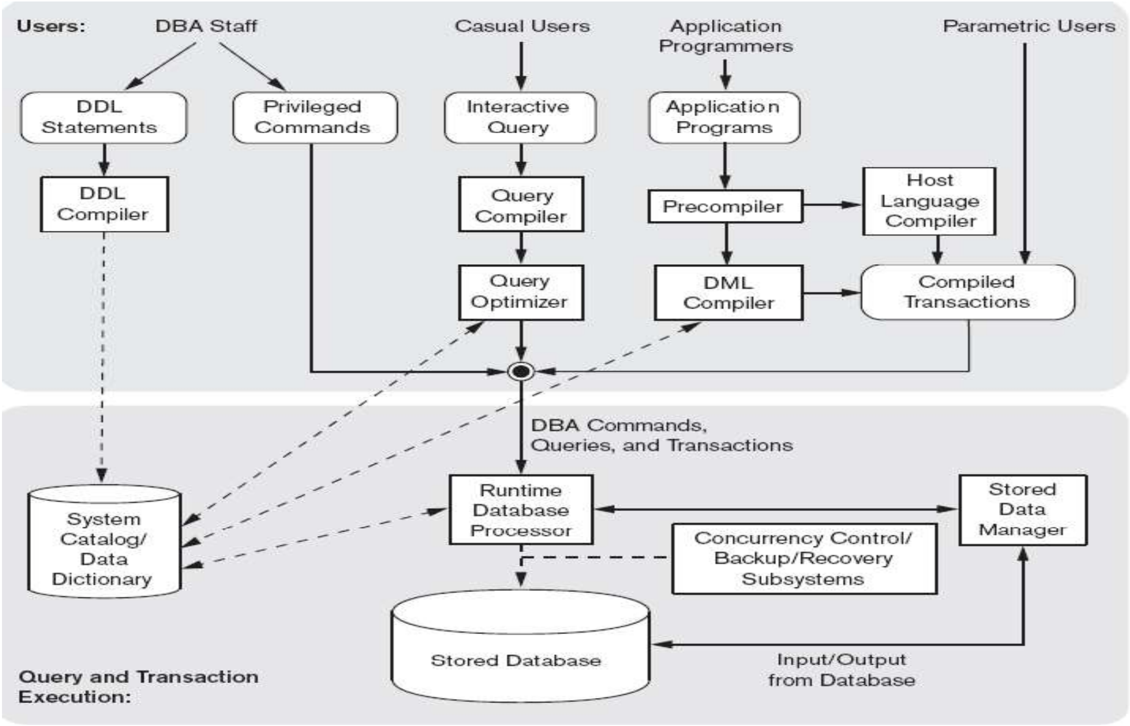
1. **Menu-Based Interfaces for Web Clients or Browsing:** These interfaces present the user with lists of options (called menus) that lead the user through the formulation of a request. There is no need for the user to memorize the specific commands and syntax of a query language. Pull-down menus are a very popular technique in Web-based user interfaces.
2. **Forms-Based Interfaces:** A forms-based interface displays a form to each user. Users can fill out all of the form entries to insert new data, or they can fill out only certain entries, in which case the DBMS will retrieve matching data for the remaining entries. Forms are usually designed and programmed for naive users as interfaces to canned transactions.
3. **Graphical User Interfaces:** A GUI typically displays a schema to the user in diagrammatic form. The user then can specify a query by manipulating the diagram. In many cases, GUIs utilize both menus and forms. Most GUIs use a pointing device, such as a mouse, to select certain parts of the displayed schema diagram.
4. **Natural Language Interfaces:**These interfaces accept requests written in English or some other language and attempt to understand them. A natural language interface usually has its own schema, which is similar to the database conceptual schema, as well as a dictionary of important words. The natural language interface refers to the words in its schema, as well as to the set of standard words in its dictionary, to interpret the request. If the interpretation is successful, the interface generates a high-level query corresponding to the natural language request and submits it to the DBMS for processing; otherwise, a dialogue is started with the user to clarify the request.
5. **Speech Input and Output**: Applications with limited vocabularies such as inquiries for telephone directory, flight arrival/departure, and credit card account information are allowing speech for input and output to enable customers to access this information. The speech input is detected using a library of predefined words and used to set up the parameters that are supplied to the queries. For output, a similar conversion from text or numbers into speech takes place.
6. **Interfaces for Parametric Users:** Parametric users, such as bank tellers, often have a small set of operations that they must perform repeatedly. For example, a teller is able to use single function keys to invoke routine and repetitive transactions such as account deposits or withdrawals, or balance inquiries. Usually a small set of abbreviated commands is included, with the goal of minimizing the number of

keystrokes required for each request.

7. **Interfaces for the DBA:** Most database systems contain privileged commands that can be used only by the DBA staff. These include commands for creating accounts, setting system parameters, granting account authorization, changing a schema, and reorganizing the storage structures of a database.

2.4 The Database System Environment

2.4.1 DBMS Component Modules



The top part of the figure refers to the various users of the database environment and their interfaces. The lower part shows the internals of the DBMS responsible for storage of data and processing of transactions.

DDL compiler-processes schema definitions, specified in the DDL, and stores descriptions of the schemas (meta-data) in the DBMS catalog.

Interactive query interface: interface for Casual users and persons with occasional need for information from the database.

Query compiler- validates for correctness of the query syntax, the names of files and data elements & compiles them into an internal form.

Query optimizer –concerned with the rearrangement and possible reordering of operations, elimination of redundancies, and use of correct algorithms and indexes during execution. It consults the system catalog for statistical and other physical information about the stored data and generates executable code that performs the necessary operations for the query and makes calls on the runtime processor.

Precompiler - extracts DML commands from an application program and sends to the DML compiler for compilation into object code for database access.

Host language compiler - rest of the program is sent to the host language compiler. The object codes for the DML commands and the rest of the program are linked, forming a canned transaction whose executable code includes calls to the runtime database processor.

Runtime database processor executes:

(1) the privileged commands

(2) the executable query plans, and

(3) the canned transactions with runtime parameters.

It works with the system catalog and may update it withstatistics. It also works with the stored data manager, which in turn uses basic operating system services for carrying out low-level input/output (read/write) operations between the disk and main memory. The runtime database processor handles other aspects of data transfer, such as management of buffers in the main memory.

stored data manager uses basic operating system services for carrying out low-level input/output (read/write) operations between the disk and main memory.

concurrency control and **backup and recovery systems** integrated into the working of the runtime database processor for purposes of transaction management.

2.4.2 Database System Utilities

Database utilities help the DBA to manage the database system. Common utilities have the following types of functions:

- **Loading:** used to load existing data files—such as text files or sequential files—into the database.
- **Backup:** creates a backup copy of the database, usually by dumping the entire database onto tape or other mass storage medium. The backup copy can be used to restore the database in case of catastrophic disk failure. Incremental backups are also often used, where only changes since the previous backup are recorded. Incremental backup is more complex, but saves storage space.

- **Database storage reorganization:** used to reorganize a set of database files into different file organizations, and create new access paths to improve performance.
- **Performance monitoring:** monitors database usage and provides statistics to the DBA. The DBA uses the statistics in making decisions such as whether or not to reorganize files or whether to add or drop indexes to improve performance.

Other utilities may be available for sorting files, handling data compression, monitoring access by users, interfacing with the network, and performing other functions.

2.4.3 Tools, Application Environments, and Communications Facilities

- **Tools**
  - **CASE :** used in the design phase of database systems
  - **Data dictionary :** In addition to storing catalog information about schemas and constraints, the data dictionary stores other information, such as design decisions, usage standards, application program descriptions, and user information. Such a system is also called an **information repository**. This information can be accessed directly by users or the DBA when needed.
- **Application development environments**
  - **PowerBuilder (Sybase) or JBuilder (Borland):** provide an environment for developing database applications including database design, GUI evelopment, querying and updating, and application program development.
  - **Communications software:** allow users at locations remote from the database system site to access the database through computer terminals, workstations, or personal computers. Integrated DBMS and data communications system is called a DB/DC system

2.5 Centralized and Client/Server Architectures for DBMSs

2.5.1 Centralized DBMSs Architecture

All DBMS functionality, application program execution, and user interface processing carried out on one machine

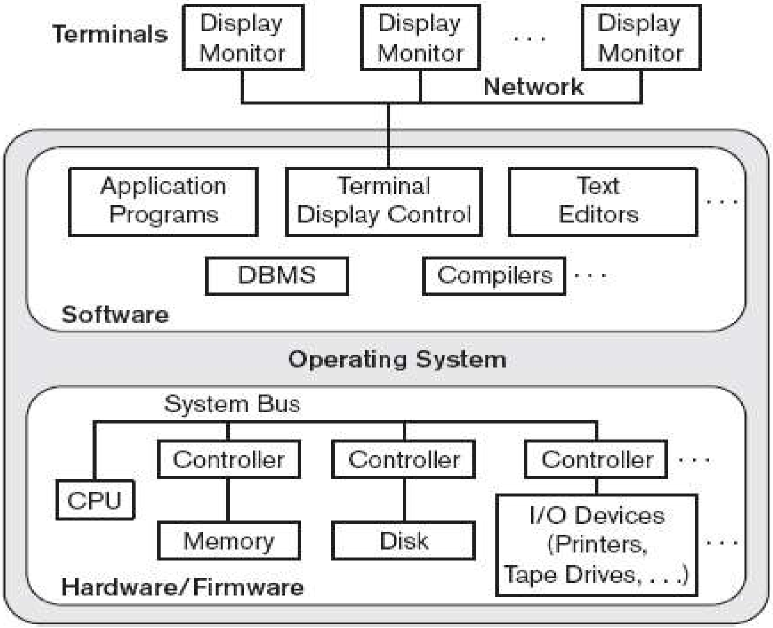


Figure 2.5.1: A physical centralized architecture

- **Disadvantages:**
  - When the central site computer or database system goes down, then everyone is blocked from using the system
  - Communication costs from the terminals to the central site can expensive

2.5.2 Basic Client/Server Architectures

The **client/server architecture** was developed to deal with computing environments in which a large number of PCs, workstations, file servers, printers, database servers, Web servers, e-mail servers, and other software and equipment are connected via a network.

- **idea**
  - define specialized servers with specific functionalities.
  - for example file server that maintains the files of the client machines
  - The resources provided by specialized servers can be accessed by many client machines.

- The client machines provide the user with the appropriate interfaces to utilize these servers and local processing power to run local applications

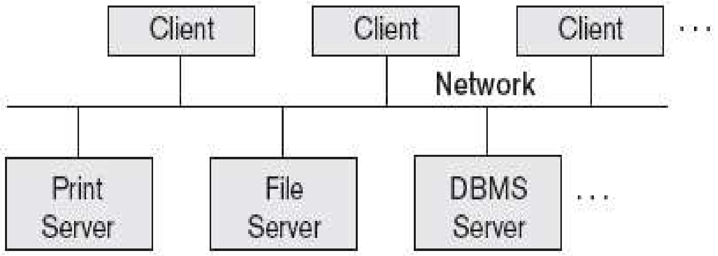


Figure 2.5.2(a) : Logical two-tier client/server architecture

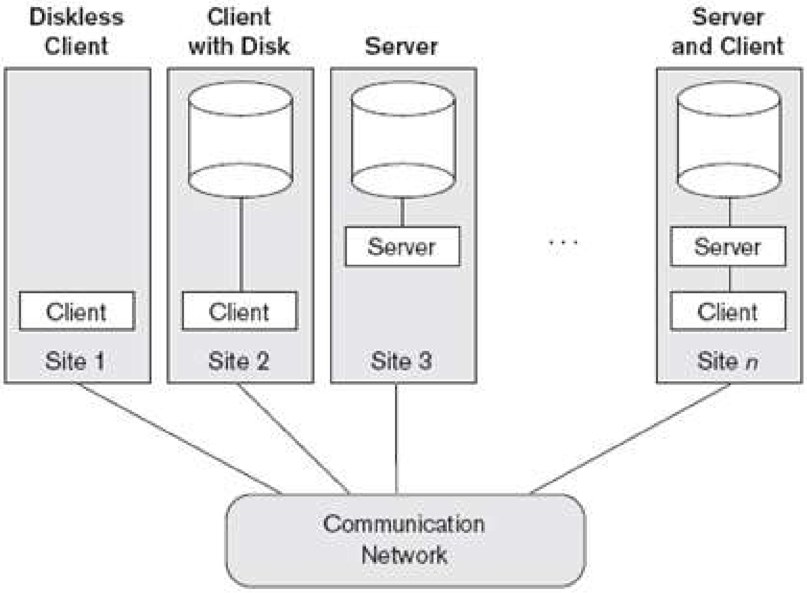


Figure2.5.2(b) : Physical two-tier client/server architecture.

The concept of client/server architecture assumes an underlying framework that consists of many PCs and workstations as well as a smaller number of mainframe machines, connected via LANs and other types of computer networks. A **client** is a user machine that provides user interface capabilities and local processing. When a client requires access to additional functionality such as database access—that does not exist at that machine, it connects to a server that provides the needed functionality. A **server** is a system containing both hardware and software that can provide services to the client machines, such as file access, printing, archiving, or database access.

2.5.3 Two-Tier Client/Server Architectures for DBMSs

The software components are distributed over two systems: client and server

- **Server handles**
  - Query and transaction functionality related to SQL processing
- **Client handles**
  - User interface programs and application programs

The user interface programs and application programs can run on the client side. When DBMS access is required, the program establishes a connection to the DBMS(which is on the server side) once the connection is created, the client program can communicate with the DBMS.

A client program can actually connect to several RDBMSs and send query and transaction requests using the ODBC API, which are then processed at the server sites. Any query results are sent back to the client program, which can process and display the results as needed. A related standard for the Java programming language, called **JDBC**, has also been defined to allow Java client programs to access one or more DBMSs through a standard interface

Object-oriented DBMSs

The different approach to two-tier client/server architecture was taken by some object-oriented DBMSs, where the software modules of the DBMS were divided between client and server in a more integrated way.

- server level

  may include the part of the DBMS software responsible for handling data storage on disk pages, local concurrency control and recovery, buffering and caching of disk pages.
- client level

  may handle the user interface, data dictionary functions, DBMS interactions with programming language compilers, global query optimization, concurrency control, and recovery across multiple servers, structuring of complex objects from the data in the buffers.

In this approach, the client/server interaction is more tightly coupled and is done internally by the DBMS modules—some of which reside on the client and some on the server—rather than by the users/programmers.

2.5.4 Three-Tier and n-Tier Architectures for Web Applications

Many Web applications use an architecture called the three-tier architecture, which adds an intermediate layer between the client and the database server

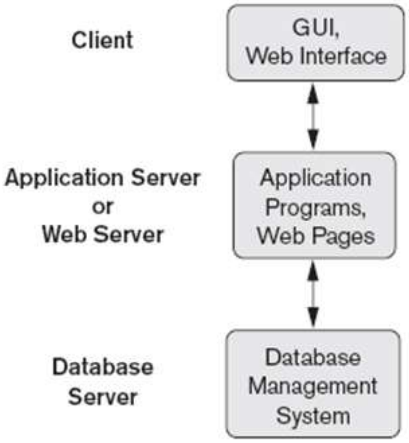


Figure 2.5.4(a): Logical three-tier client/server architecture

- **Client**
  - Contain GUI interfaces and some additional application-specific business rules
- **Application server or the Web server**
  - accepts requests from the client, processes the request and sends database queries and commands to the database server, and then passes processed data from the database server to the clients, where it may be processed further and filtered to be presented to users in GUI format.
  - It can also improve database security by checking a client's credentials before forwarding a request to the database server.

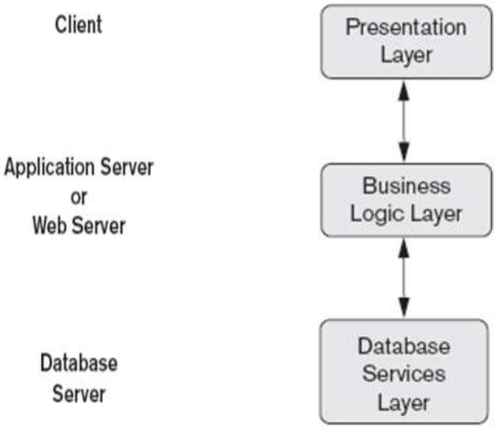


Figure 2.5.4(b): Logical three-tier client/server architecture

Figure 2.5.4(b) shows another architecture used by database and other application package vendors.

- **Presentation layer**

- displays information to the user and allows data entry

- **The business logic layer**
  - handles intermediate rules and constraints before data is passed up to the user or down to the DBMS
  - can also act as a Web server, which retrieves query results from the database server and formats them into dynamic Web pages that are viewed by the Web browser at the client side
- **The bottom layer**
  - includes all data management services

2.5.5 N-tier Architecture

It is possible to divide the layers between the user and the stored data further into finer components, thereby giving rise to n-tier architectures; where n may be four or five tiers. The business logic layer is divided into multiple layers

- **Advantage**
  - any one tier can run on an appropriate processor or operating system platform and can be handled independently.

Vendors of ERP (enterprise resource planning) and CRM (customer relationship management) packages often use a middleware layer*,* which accounts for the front-end modules (clients) communicating with a number of back-end databases (servers).

2.6 Classification of Database Management Systems

Criteria used to classify DBMSs are

1. Data model on which the DBMS is based
   - Relational: represents a database as a collection of tables, where each table can be stored as a separate file.
   - Object: defines a database in terms of objects, their properties, and their operations. Objects with the same structure and behavior belong to a **class**, and classes are organized into **hierarchies** (or **acyclic graphs**). The operations of each class are specified in terms of predefined procedures called **methods**.
   - Hierarchical and network (legacy): The **network model** represents data as record types and also represents a limited type of 1:N relationship, called a **set type**. The

hierarchical model represents data as hierarchical tree structures. Each hierarchy represents a number of related records.

- Native XML DBMS: uses hierarchical tree structures. It combines database concepts with concepts from document representation models. Data is represented as elements; with the use of tags, data can be nested to create complex hierarchical structures.

2. Number of users supported by the system

- Single-user: support only one user at a time and are mostly used with PCs.
- Multiuser: support concurrent multiple users.

3. Number of sites over which the database is distributed

- Centralized: data is stored at a single computer site
- Distributed: can have the actual database and DBMS software distributed over many sites, connected by a computer network
  - **Homogeneous** DDBMSs use the same DBMS software at all the sites
  - **Heterogeneous** DDBMSs can use different DBMS software at each site

4. Cost

- Open source: products like MySQL and PostgreSQL that are supported by third-party vendors with additional services.
- Different types of licensing: Standalone single user versions of some systems like Microsoft Access are sold per copy or included in the overall configuration of a desktop or laptop. In addition, data warehousing and mining features, as well as support for additional data types, are made available at extra cost.

5. On the basis of the **types of access path options for** storing files

- One well-known family of DBMSs is based on inverted file structures.

.6. General purpose or Special purpose

- When performance is a primary consideration, a special-purpose DBMS can be designed and built for a specific application; such a system cannot be used for other applications without major changes. Many airline reservations and telephone directory systems developed in the past are special-purpose DBMSs.

Chapter 3: Conceptual Data Modelling using Entities and Relationships

Introduction

Conceptual modeling is a very important phase in designing a successful database application. Entity-Relationship (ER) model is a popular high-level conceptual data model. This model and its variations are frequently used for the conceptual design of database applications, and many database design tools employ its concepts.

3.1 Using High-Level Conceptual Data Models for Database Design

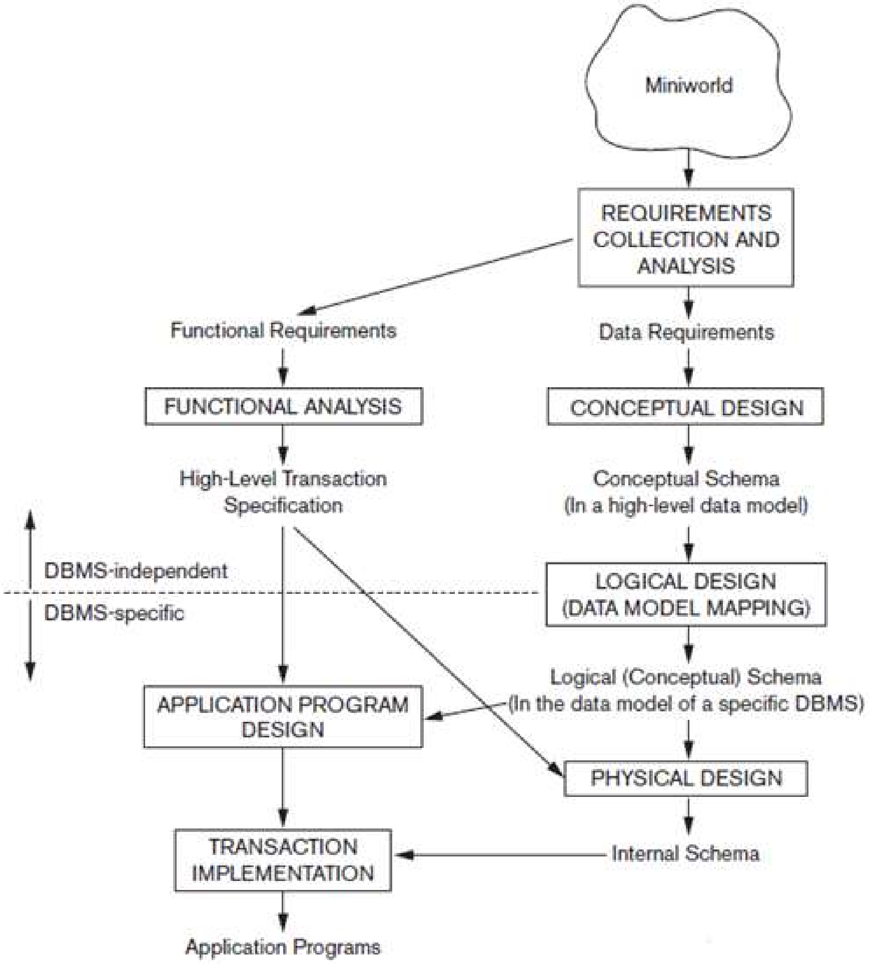

Miniworld
REQUIREMENTS COLLECTION AND ANALYSIS
Functional Requirements
Data Requirements
FUNCTIONAL ANALYSIS
CONCEPTUAL DESIGN
High-Level Transaction Specification
Conceptual Schema (In a high-level data model)
DBMS-independent
DBMS-specific
LOGICAL DESIGN (DATA MODEL MAPPING)
Logical (Conceptual) Schema (In the data model of a specific DBMS)
APPLICATION PROGRAM DESIGN
PHYSICAL DESIGN
TRANSACTION IMPLEMENTATION
Internal Schema
Application Programs

Figure 3.1: A simplified diagram to illustrate the main phases of database design.

The first step shown is **requirements collection and analysis**. During this step, the database designers interview prospective database users to understand and document their **data requirements**. The result of this step is a concisely written set of users' requirements.

These requirements should be specified in as detailed and complete a form as possible. In parallel with specifying the data requirements, it is useful to specify the known **functional requirements** of the application. These consist of the userdefined **operations** (or **transactions**) that will be applied to the database, including both retrievals and updates.

Once the requirements have been collected and analyzed, the next step is to create a **conceptual schema** for the database, using a high-level conceptual data model. This step is called **conceptual design**. The conceptual schema is a concise description of the data requirements of the users and includes detailed descriptions of the entity types, relationships, and constraints; these are expressed using the concepts provided by the high-level data model.

The next step in database design is the actual implementation of the database, using a commercial DBMS. Most current commercial DBMSs use an implementation data model—such as the relational or the object-relational database model—so the conceptual schema is transformed from the high-level data model into the implementation data model. This step is called **logical design** or **data model mapping**; its result is a database schema in the implementation data model of the DBMS.

The last step is the **physical design** phase, during which the internal storage structures, file organizations, indexes, access paths, and physical design parameters for the database files are specified. In parallel with these activities, application programs are designed and implemented as database transactions corresponding to the highlevel transaction specifications.

3.2 Entity Types, Entity Sets, Attributes, and Keys

The ER model describes data as entities, relationships, and attributes.

3.2.1 Entities and Attributes

Entity: a thing in the real world with an independent existence. An entity may be an object with a physical existence (for example, a particular person, car, house, or employee) or it may be an object with a conceptual existence (for instance, a company, a job, or a university course).

Attributes: Particular properties that describe entity. For example, an EMPLOYEE entity may be described by the employee's name, age, address, salary, and job.

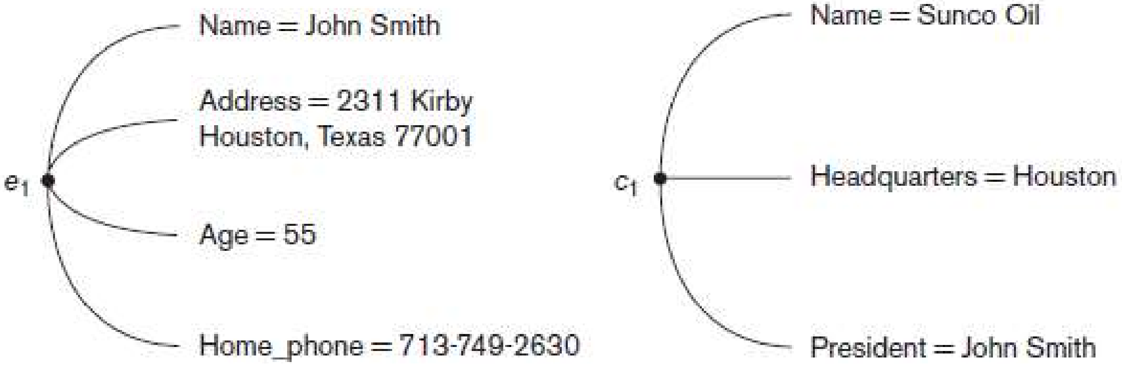


Figure 3.2.1(a): Two entities, EMPLOYEE e1, and COMPANY c1, and their attributes

Types of attributes:

1.Composite versus Simple (Atomic) Attributes

2.Single-valued versus multivalued

3.Stored versus derived

4.NULL values

5.Complex attributes

1. **Composite versus Simple (Atomic) Attributes**

   Composite Attributes can be divided into smaller subparts, which represent more basic attributes with independent meanings.For example, the Address attribute of the EMPLOYEE entity can be subdivided into Street_address, City, State, and Zip.

   Composite attributes can form a hierarchy. For example, Street_address can be further subdivided into three simple component attributes: Number, Street, and Apartment_number. The value of a composite attribute is the concatenation of the values of its component simple attributes.

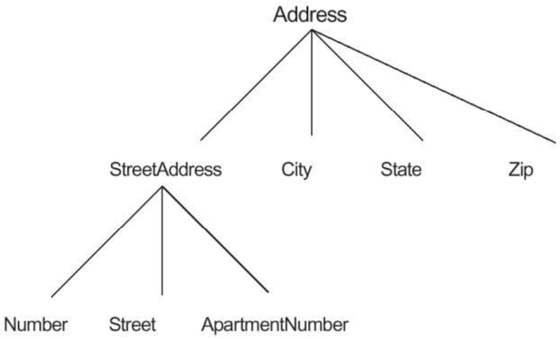


Figure 3.2.1(b): A hierarchy of composite attributes.

Attributes that are not divisible are called **simple** or **atomic attributes**. Example SSN of an employee.

2. Single-Valued versus Multivalued Attributes

Attributes that have a single value for a particular entity are called **single-valued**. For example, Age is a single-valued attribute of a person.

Attributes that can have a set of values for a particular entity are called **Multivalued Attributes.** For example Colors attribute for a car, or a College_degrees attribute for a person. A multivalued attribute may have lower and upper bounds to constrain the number of values allowed for each individual entity. For example, the Colors attribute of a car may be restricted to have between one and three values, if we assume that a car can have three colors at most.

3. Stored versus Derived Attributes

An attribute, which cannot be derived from other attribute are called **stored attribute.** For example, Birth_Date of an employee

Attributes derived from other stored attribute are called **derived attribute.** For example age of an employee can be determined from the current (today's) date and Date of Birth

4. Null Value Attribute(Optional Attribute)

In some cases, a particular entity may not have an applicable value for an attribute. For example, the Apartment_number attribute of an address applies only to addresses that are in apartment buildings and not to other types of residences, such as single-family homes. Similarly, a College_degrees attribute applies only to people with college degrees. For such situations, a special value called **NULL** is created. An address of a single-family home would have NULL for its Apartment_number attribute, and a person with no college degree would have NULL for College_degrees. NULL can also be used if we do not know the value of an attribute for a particular entity

5. Complex Attributes

If an attribute for an entity, is built using composite and multivalued attributes, then these attributes are called complex attributes. For example, a person can have more than one residence and each residence can have multiple phones, an addressphone for a person entity can be specified as :

{ Addressphone (phone {(Area Code, Phone Number)}, Address(Sector Address (Sector Number,House Number),

City, State, Pin))

}

Here {} are used to enclose multivalued attributes and () are used to enclose composite attributes with comma separating individual attributes

3.2.2 Entity Types, Entity Sets, Keys, and Value Sets

Entity Types

An **entity type** defines a collection (or set) of entities that have the same attributes. Each entity type in the database is described by its name and attributes. For example, a company employing hundreds of employees may want to store similar information concerning each of the employees. These employee entities share the same attributes, but each entity has its own value(s) for each attribute.

Entity Sets

The collection of all entities of a particular entity type in the database at any point in time is called an **entity set**; the entity set is usually referred to using the same name as the entity type. For example, EMPLOYEE refers to both a type of entity as well as the current set of all employee entities in the database.

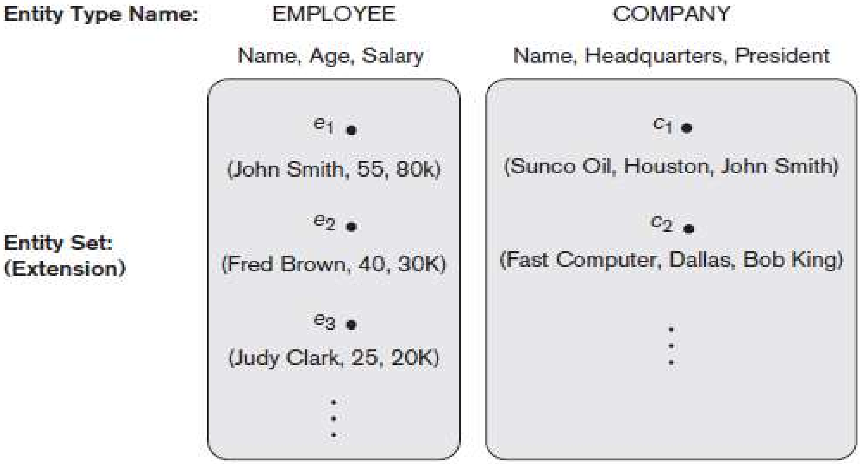


Figure 3.2.2(a): Two entity types, EMPLOYEE and COMPANY, and some member entities of each.

An entity type describes the **schema** or **intension** for a set of entities that share the same structure. The collection of entities of a particular entity type is grouped into an entity set, which is also called the **extension** of the entity type.

An **entity type** is represented in ER diagrams a **rectangular box** enclosing the entity type name. **Attribute names** are enclosed in **ovals** and are attached to their entity type by straight lines. **Composite attributes** are attached to their component attributes by straight lines. **Multivalued attributes** are displayed in **double ovals**

Key Attributes of an Entity Type

An entity type usually has one or more attributes whose values are distinct for each individual entity in the entity set. Such an attribute is called a key attribute, and its values can be used to identify each entity uniquely. For example, the Name attribute is a key of the COMPANY entity because no two companies are allowed to have the same name.

In ER diagrammatic notation, each key attribute has its name underlined inside the oval.Some entity types have more than one key attribute. For example, each of the Vehicle_id and Registration attributes of the entity type CAR is a key in its own right

Example: The CAR entity type with two key attributes, Registration and Vehicle_id.

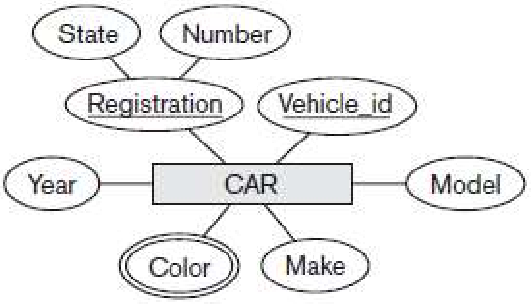
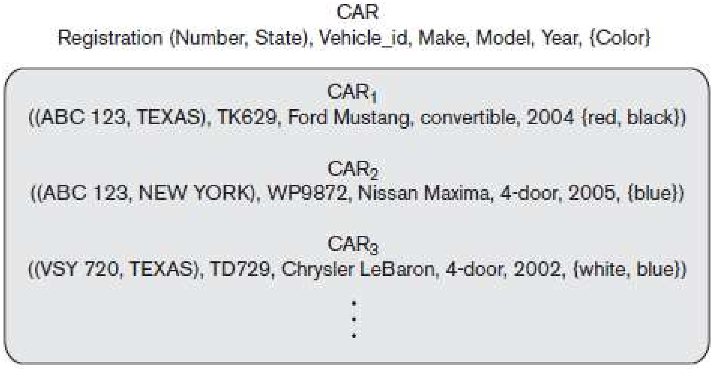


Figure 3.2.2(b) : ER diagram notation



CAR

Registration (Number, State), Vehicle_id, Make, Model, Year, {Color}

CAR_1

((ABC 123, TEXAS), TK629, Ford Mustang, convertible, 2004 {red, black})

CAR_2

((ABC 123, NEW YORK), WP9872, Nissan Maxima, 4-door, 2005, {blue})

CAR_3

((VSY 720, TEXAS), TD729, Chrysler LeBaron, 4-door, 2002, {white, blue})

⋮

Entity set with three entities.

Value Sets (Domains) of Attributes

Each simple attribute of an entity type is associated with a **value set** (or **domain** of values), which specifies the set of values that may be assigned to that attribute for each individual entity. For example, if the range of ages allowed for employees is between 16 and 70, we can specify the value set of the Age attribute of EMPLOYEE to be the set of integer numbers between 16 and 70.

Value sets are not displayed in ER diagrams, and are specified using the basic data types available in most programming languages, such as integer, string, Boolean,float, enumerated type, subrange, and so on.

Mathematically, an attribute A of entity set E whose value set is V can be defined as a **function from E to the power set P(V) of V:** A : E → P(V).We refer to the value of attribute A for entity e as A(e).A NULL value is represented by the empty set.

3.3 A Sample Database Application

Miniworld : COMPANY database keeps track of a company's employees, departments, and projects.

- After the requirements collection and analysis phase, the database designers provide the following description of the *miniworld:*
  - The company is organized into departments.
  - Each department has a unique name, a unique number, and a particular employee who manages the department.We keep track of the start date when that employee began managing the department. A department may have several locations.
  - A department controls a number of projects, each of which has a unique name, a unique number, and a single location.
  - We store each employee's name, Social Security number, address, salary, gender, and birth date.
  - An employee is assigned to one department, but may work on several projects, which are not necessarily controlled by the same department.
  - We keep track of the current number of hours per week that an employee works on each project. We also keep track of the direct supervisor of each employee (who is another employee).
  - We want to keep track of the dependents of each employee for insurance purposes. We keep each dependent's first name, gender, birth date, and relationship to the employee.

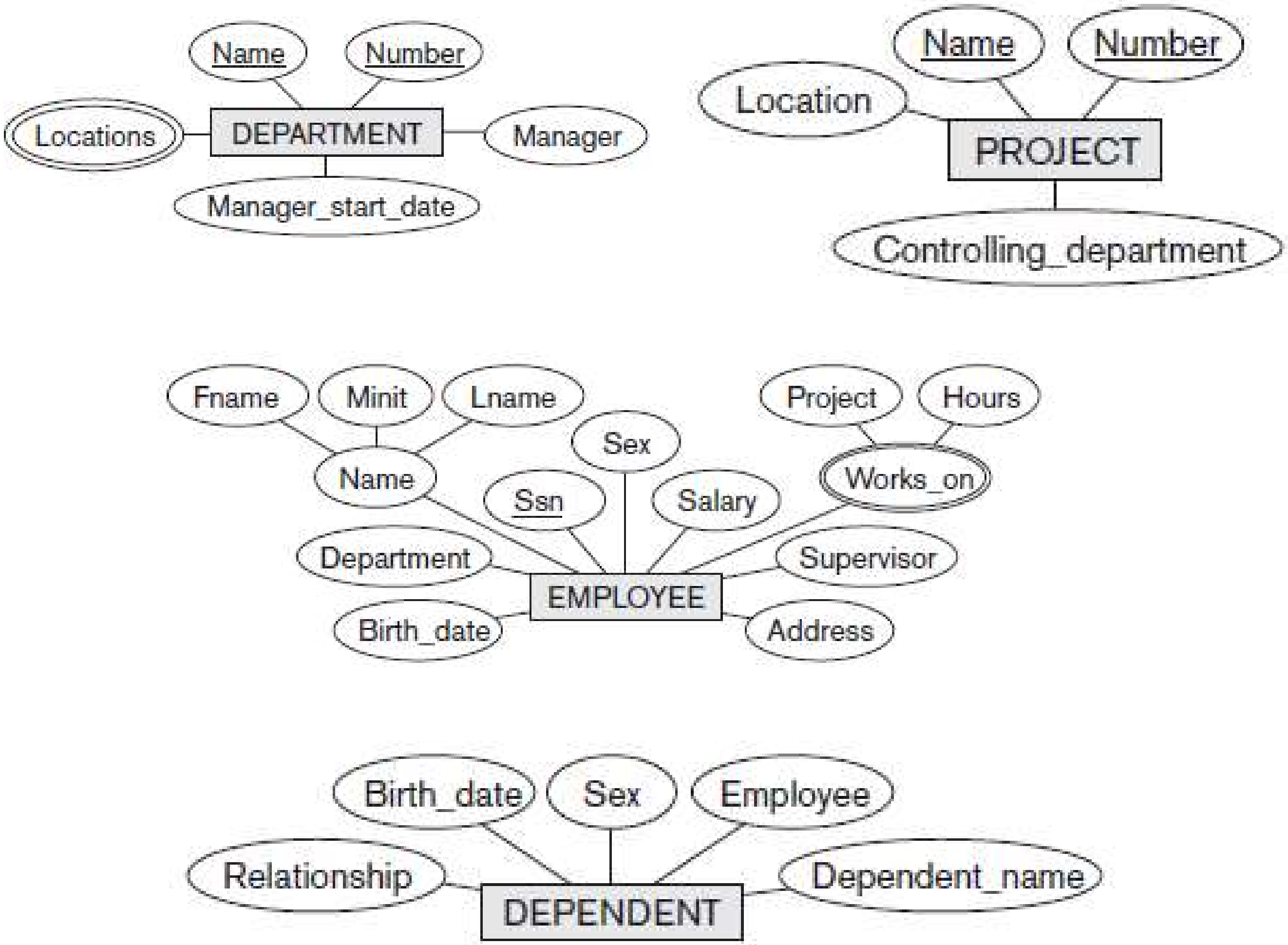


Figure 3.3(a): Preliminary design of entity types for the COMPANY database. Some of the shown attributes will be refined into relationships.

3.4 Relationship Types, Relationship Sets, Roles, and Structural Constraints

There are several implicit relationships among the various entity types. Whenever an attribute of one entity type refers to another entity type, some relationship exists. For example

- The attribute Manager of DEPARTMENT refers to an employee who manages the department
- The attribute Controlling department of PROJECT refers to the department that controls the project
- The attribute Supervisor of EMPLOYEE refers to another employee -the one who supervises this employee
- The attribute Department of EMPLOYEE refers to the department for which the employee works

In the ER model, these references should not be represented as attributes but as relationships

3.4.1 Relationship Types, Sets, and Instances

A **relationship type R** among n entity types E_1, E_2, ..., E_n defines a set of associations—or a **relationship set**—among entities from these entity types. Entity types and Entity sets, a Relationship type and its corresponding Relationship set are usually referred to by the same name, R.

Mathematically, the relationship set R is a set of relationship instances r_i , where each r_i associates n individual entities (e_1, e_2, ..., e_n), and each entity e_i in r_i is a member of entity set E_j , $1\leq j\leq n$. Each of the entity types E_1, E_2, ..., E_n is said to participate in the relationship type R. similarly, each of the individual entities e_1, e_2, ..., e_n is said to participate in the relationship instance r_i = (e_1, e_2, ..., e_n)

Informally, each relationship instance r_i in R is an association of entities, where the association includes exactly one entity from each participating entity type. For example, consider a relationship type WORKS_FOR between the two entity types EMPLOYEE and DEPARTMENT, which associates each employee with the department for which the employee works in the corresponding entity set. Each relationship instance in the relationship set WORKS_FOR associates one EMPLOYEE entity and one DEPARTMENT entity.

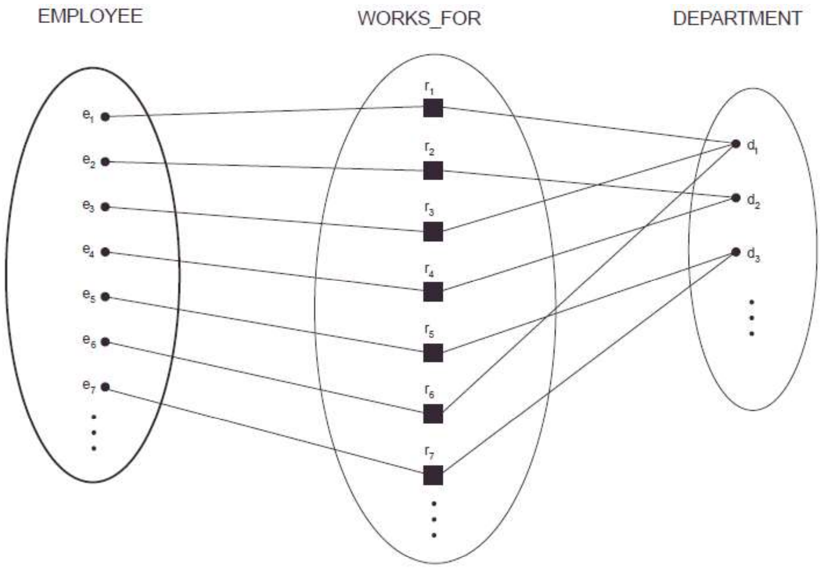


Figure 3.4.1: Some instances in the WORKS_FOR relationship set, which represents a relationship type WORKS_FOR between EMPLOYEE and DEPARTMENT.

employees *e_1* , *e_3*, and *e_6* work for department *d_1*. employees *e_2* and *e_4* work for department *d_2* and employees *e_5* and *e_7* work for department *d_3*.

In ER diagrams, relationship types are displayed as diamond-shaped boxes, which are connected by straight lines to the rectangular boxes representing the participating entity types. The relationship name is displayed in the diamond-shaped box.

3.4.2 Relationship Degree, Role Names, and Recursive Relationships

Degree of a Relationship Type

The degree of a relationship type is the number of participating entity types. A relationship type of degree two is called **binary**, and one of degree three is called **ternary** An example of a binary relationship WORKS_FOR and ternary relationship is SUPPLY

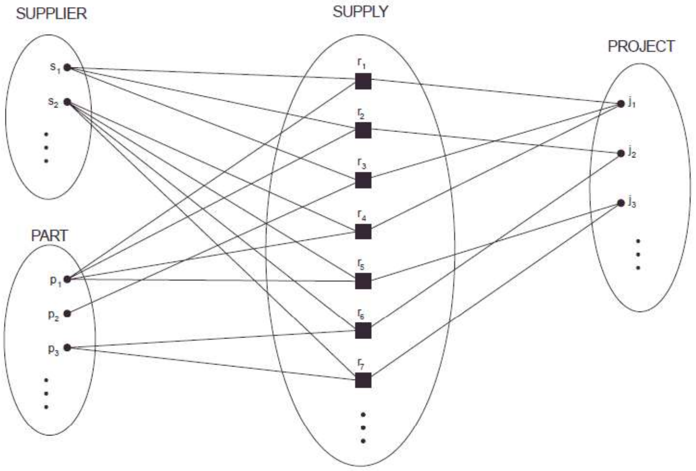


Figure 3.4.2(a): Some relationship instances in the SUPPLY ternary relationship set.

Each relationship instance r_i associates three entities—a supplier s, a part p and a project j—whenever s supplies part p to project j.

Relationships as Attributes

It is sometimes convenient to think of a binary relationship type in terms of attributes. Consider the WORKS_FOR relationship type. One can think of an attribute called Department of the EMPLOYEE entity type, where the value of Department for each EMPLOYEE entity is a reference to the DEPARTMENT entity for which that employee works. This concept of representing relationship types as attributes is used in a class of data models called **functional data models**.

In relational databases, foreign keys are a type of reference attribute used to represent relationships.

Role Names and Recursive Relationships

Each entity type that participates in a relationship type plays a particular role in the relationship.

The **role name** signifies the role that a participating entity from the entity type plays in each relationship instance, and helps to explain what the relationship means.For example, in the WORKS_FOR relationship type, EMPLOYEE plays the role of employee or worker and DEPARTMENT plays the role of department or employer.

Role names are not technically necessary in relationship types where all the participating entity types are distinct, since each participating entity type name can be used as the role name. However, in some cases the same entity type participates more than once in a relationship type in different roles.

In such cases the role name becomes essential for distinguishing the meaning of the role that each participating entity plays. Such relationship types are called **recursive relationships.** Example of recursive relationships **:** SUPERVISION relationship type

The SUPERVISION relationship type relates an employee to a supervisor, where both employee and supervisor entities are members of the same EMPLOYEE entity set. Hence, the EMPLOYEE entity type participates twice in SUPERVISION: once in the role of supervisor (or boss), and once in the role of supervisee (or subordinate). Each relationship instance r_i in SUPERVISION associates two employee entities e_j and e_k , one of which plays the role of supervisor and he other the role of supervisee.

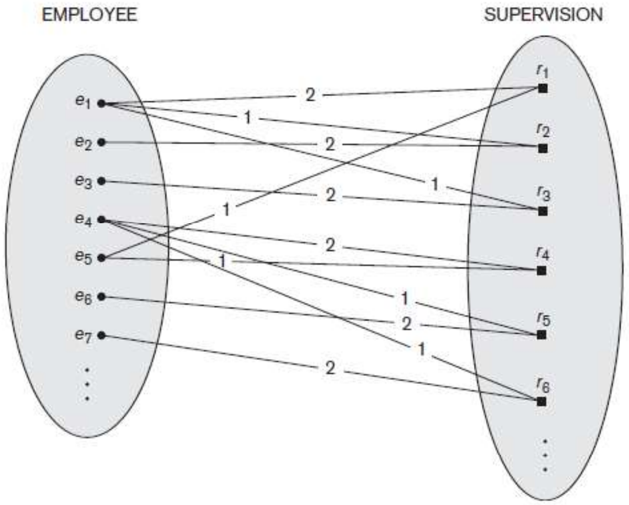


Figure 3.4.2(b): A recursive relationship SUPERVISION between EMPLOYEE in the supervisor role (1) and EMPLOYEE in the subordinate role (2).

3.4.3 Constraints on Binary Relationship Types

Relationship types usually have certain constraints that limit the possible combinations of entities that may participate in the corresponding relationship set. These constraints are determined from the miniworld situation that the relationships represent.For example, if the company has a rule that each employee must work for exactly one department, then we would like to describe this constraint in the schema. Two main types of binary relationship constraints:

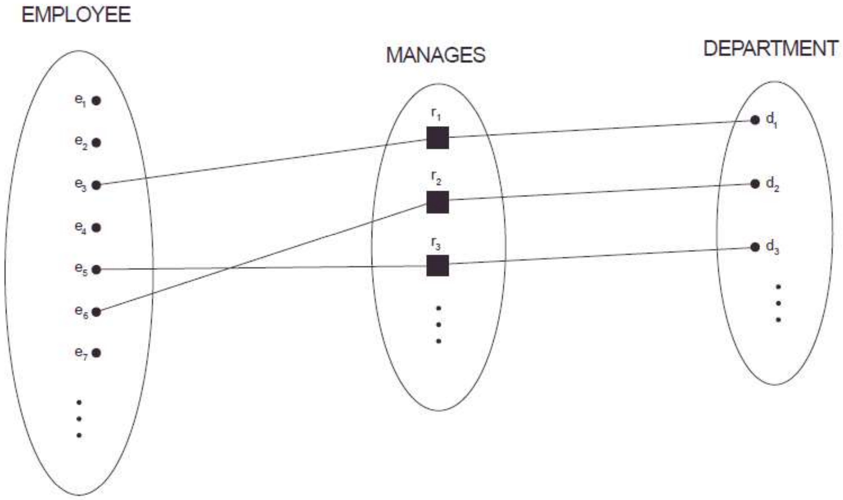
1. cardinality ratio
2. participation.

Cardinality Ratios for Binary Relationships

The cardinality ratio for a binary relationship specifies the maximum number of relationship instances that an entity can participate in. For example, in the WORKS_FOR binary relationship type, DEPARTMENT:EMPLOYEE is of cardinality ratio 1:N, meaning that each department can be related to any number of employees, but an employee can be related to (work for) only one department. The possible cardinality ratios for binary relationship types are 1:1, 1:N, N:1, and M:N.

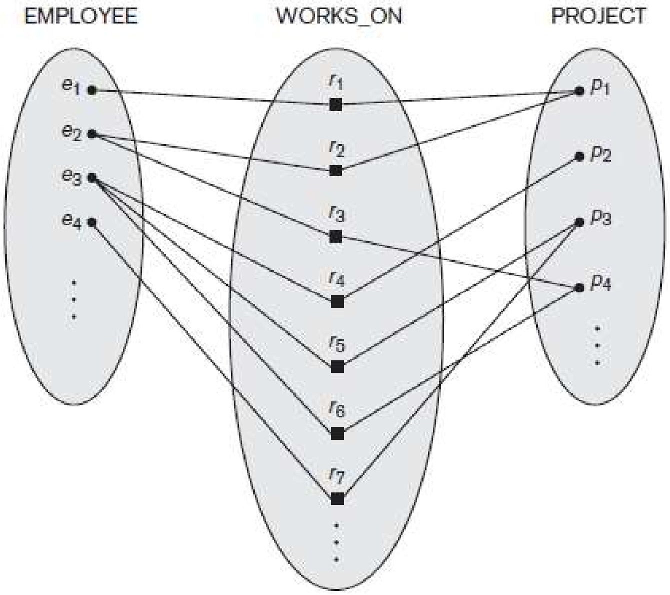
Example of a 1:1 binary relationship

- MANAGES which relates a department entity to the employee who manages that department
- This represents the miniworld constraints that—at any point in time—an employee can manage one department only and a department can have one manager only



Example of a M:N binary relationship

- The relationship type WORKS_ON is of cardinality ratio M:N, because the mini-world rule is that an employee can work on several projects and a project can have several employees.
- Cardinality ratios for binary relationships are represented on ER diagrams by displaying 1, M, and N on the diamonds



Participation Constraints and Existence Dependencies

The participation constraint specifies whether the existence of an entity depends on its being related to another entity via the relationship type. This constraint specifies the minimum number of relationship instances that each entity can participate in, and is sometimes called the minimum cardinality constraint.There are two types of participation constraints:

- Total
- Partial

Total participation

If a company policy states that every employee must work for a department, then an employee entity can exist only if it participates in at least one WORKS_FOR relationship Instance. Thus, the participation of EMPLOYEE in WORKS_FOR is called total participation, meaning that every entity in the total set of employee entities must be related to a department entity via WORKS_FOR. Total participation is also called **existence dependency**

Partial participation

we do not expect every employee to manage a department .So the participation of EMPLOYEE in the MANAGES relationship type is partial, meaning that some or part of the set of employee entities are related to some department entity via MANAGES, but not necessarily all.

In ER diagrams, **total participation** is displayed as a **double line** connecting the participating entity type to the relationship, whereas **partial participation** is represented by a **single line.**

cardinality ratio + participation constraints = structural constraints of a relationship type.

3.4.4 Attributes of Relationship Types

Relationship types can also have attributes, similar to those of entity types. For example, to record the number of hours per week that an employee works on a particular project, we can include an attribute Hours for the WORKS_ON relationship type. Another example is to include the date on which a manager started managing a department via an attribute Start_date for the MANAGES relationship type.
Attributes of 1:1 or 1:N relationship types can be migrated to one of the participating entity types. For a 1:N relationship type, a relationship attribute can be migrated only to the entity type on the N-side of the relationship. For M:N relationship types, some attributes may be determined by the combination of participating entities in a relationship instance, not by any single entity. Such attributes must be specified as relationship attributes.

3.5 Weak Entity Types

Entity types that do not have key attributes of their own are called **weak entity** types. Entities belonging to a weak entity type are identified by being related to specific entities from another entity type in combination with one of their attribute values. We call this other entity type the identifying or **owner entity type.** We call the relationship type that relates a weak entity type to its owner the **identifying relationship** of the weak entity type.

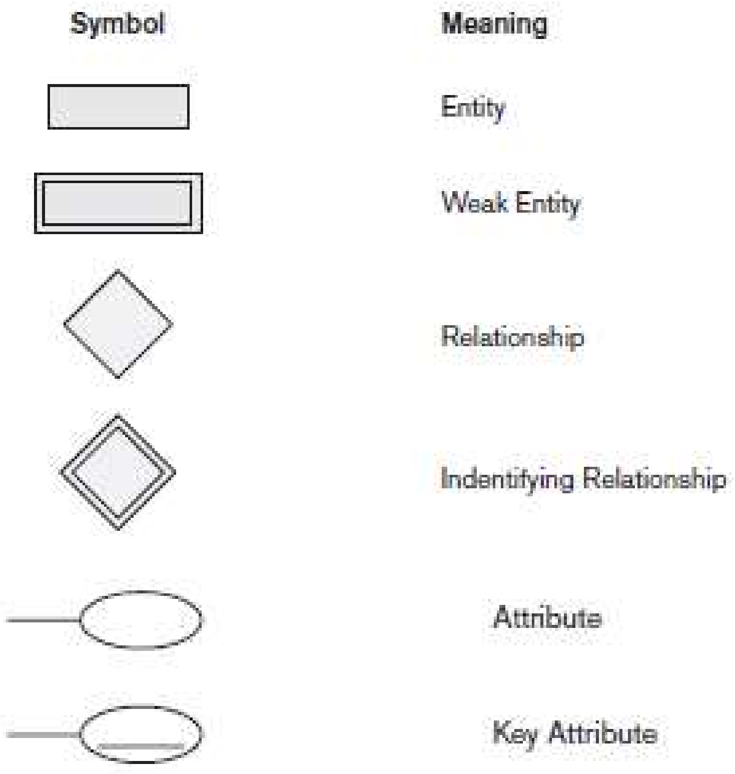
Consider the entity type DEPENDENT, related to EMPLOYEE, which is used to keep track of the dependents of each employee via a 1:N relationship. In our example, the attributes of DEPENDENT are Name,Birth_date, gender, and Relationship (to the employee). Two dependents of two distinct employees may, by chance, have the same values for Name, Birth_date, gender, and Relationship, but they are still distinct entities. They are identified as distinct entities only after determining the particular employee entity to which each dependent is related. Each employee entity is said to own the dependent entities that are related to it.

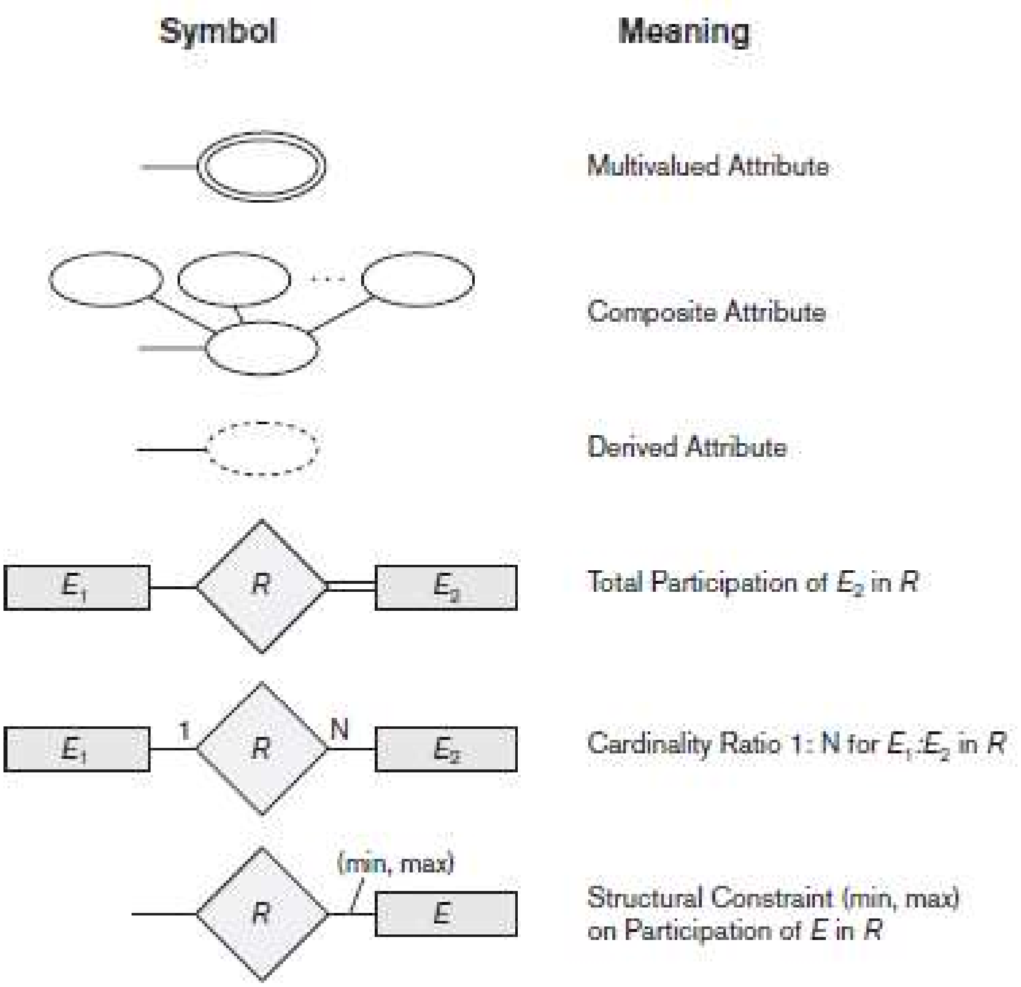
A weak entity type always has a total participation constraint (existence dependency) with respect to its identifying relationship because a weak entity cannot be identified without an owner entity. A weak entity type normally has a **partial key**, which is the attribute that can uniquely identify weak entities that are related to the same owner entity. In our example, if we assume that no two dependents of the same employee ever have the same first name, the attribute Name of DEPENDENT is the partial key.

In ER diagrams, both a weak entity type and its identifying relationship are distinguished by surrounding their boxes and diamonds with double lines. The partial key attribute is underlined with a dashed or dotted line.

3.6 ER Diagrams, Naming Conventions, and Design Issues

3.6.1 Summary of Notation for ER Diagrams




Symbol
Meaning
Multivalued Attribute
Composite Attribute
Derived Attribute
E_1
R
E_2
Total Participation of E_2 in R
1
N
Cardinality Ratio 1: N for E_1:E_2 in R
(min, max)
E
Structural Constraint (min, max) on Participation of E in R

3.6.2 Proper Naming of Schema Constructs

- Choose names that convey, as much as possible, the meanings attached to the different constructs in the schema.
- Use *singular names* for entity types,rather than plural ones, because the entity type name applies to each individual entity belonging to that entity type
- In ER diagrams, entity type and relationship type names are uppercase letters, attribute names have their initial letter capitalized, and role names are lowercase letters.
- As a general practice, given a narrative description of the database requirements, the nouns appearing in the narrative tend to give rise to entity type names, and the verbs tend to indicate names of relationship types. Attribute names generally arise from additional nouns that describe the nouns corresponding to entity types.

- Another naming consideration involves choosing binary relationship names to make the ER diagram of the schema readable from left to right and from top to bottom.

3.6.3 Design Choices for ER Conceptual Design

In general, the schema design process should be considered an iterative refinement process, where an initial design is created and then iteratively refined until the most suitable design is reached. Some of the refinements that are often used include the following:

- A concept may be first modeled as an attribute and then refined into a relationship because it is determined that the attribute is a reference to another entity type. It is often the case that a pair of such attributes that are inverses of one another are refined into a binary relationship.
- Similarly, an attribute that exists in several entity types may be elevated or promoted to an independent entity type. For example, suppose that several entity types in a UNIVERSITY database, such as STUDENT, INSTRUCTOR, and COURSE, each has an attribute Department in the initial design; the designer may then choose to create an entity type DEPARTMENT with a single attribute Dept_name and relate it to the three entity types (STUDENT, INSTRUCTOR, and COURSE) via appropriate relationships.
- An inverse refinement to the previous case may be applied—for example, if an entity type DEPARTMENT exists in the initial design with a single attribute Dept_name and is related to only one other entity type, STUDENT. In this case, DEPARTMENT may be reduced or demoted to an attribute of STUDENT.

3.6.4 Alternative Notations for ER Diagrams

There are many alternative diagrammatic notations for displaying ER diagrams. One alternative ER notation for specifying structural constraints on relationships, which replaces the cardinality ratio (1:1, 1:N, M:N) and single/double line notation for participation constraints. This notation involves associating a pair of integer numbers (min, max) with each *participation* of an entity type *E* in a relationship type *R*, where $0 \leq \text{min} \leq \text{max}$ and $\text{max} \geq 1$.

The numbers mean that for each entity e in E, e must participate in at least min and at most max relationship instances in R at any point in time. In this method, min = 0 implies partial participation, whereas min > 0 implies total participation.

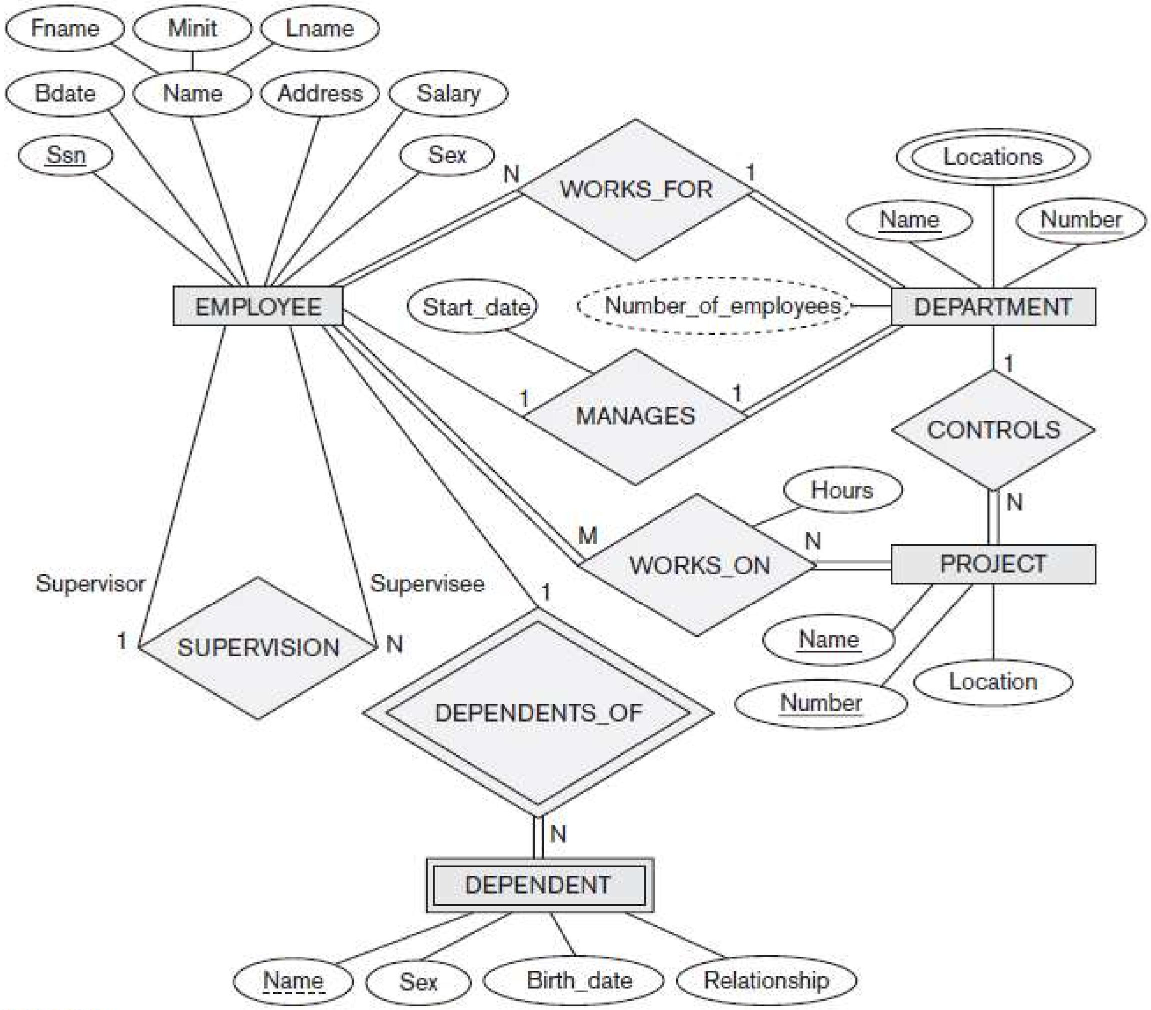


Figure 3.6.4 (a) : ER diagram for Company Database

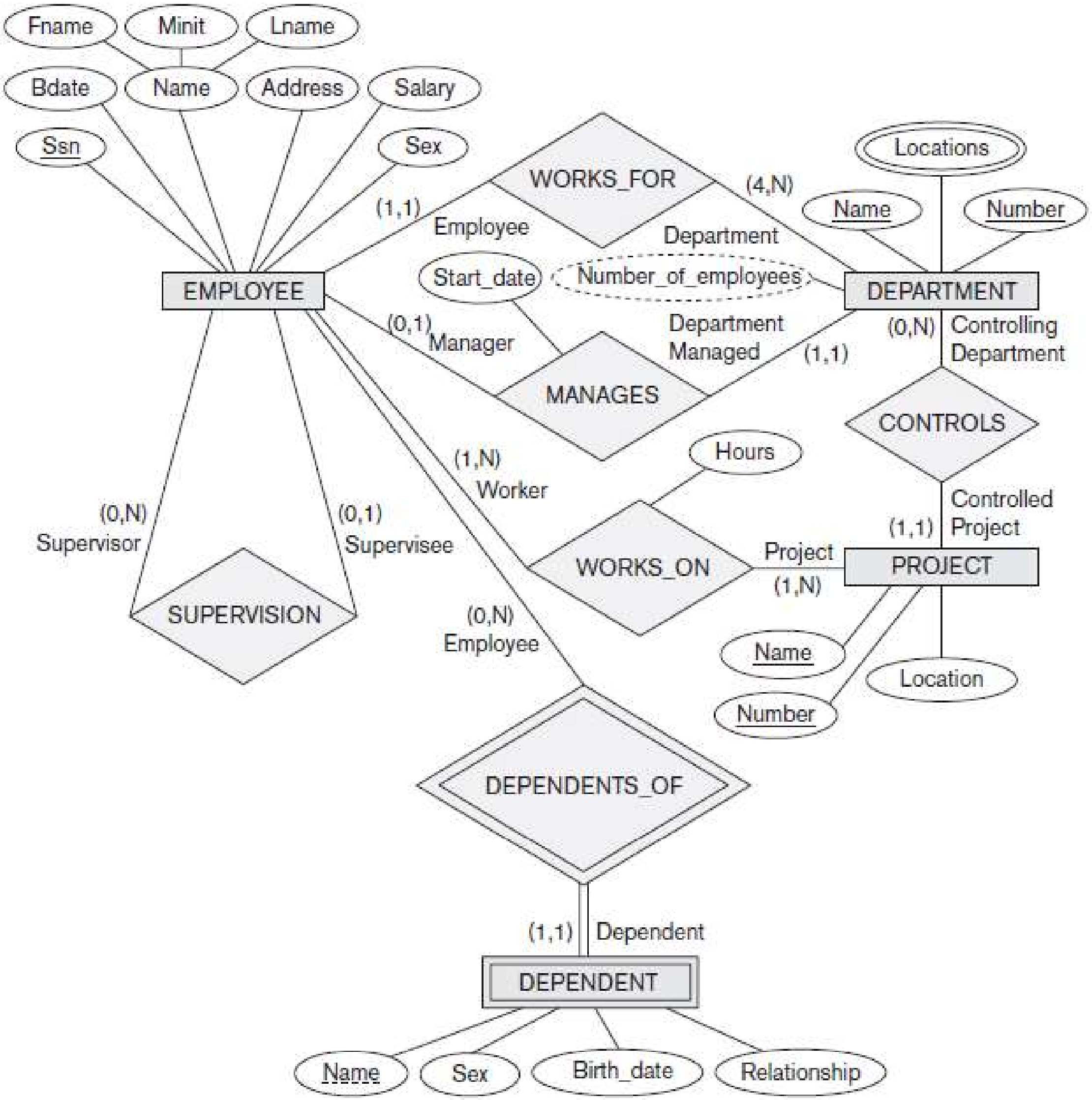


Figure 3.6.4(b): ER diagram for Company Database (using alternative notation)

3.7 Relationship Types of Degree Higher than Two

3.7.1 Choosing between Binary and Ternary (or Higher-Degree) Relationships

A relationship type R of degree n will have n edges in an ER diagram, one connecting R to each participating entity type

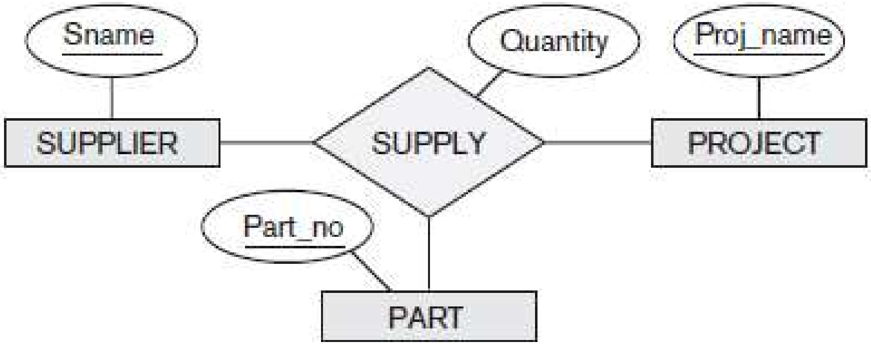


Fig 3.7.1(a): The SUPPLY relationship

Figure 3.7.1(a0 shows the ER diagram notation for a ternary relationship. SUPPLY is a set of relationship instances (*s*, *j*, *p*), where *s* is a SUPPLIER who is currently supplying a PART *p* to a PROJECT *j*.

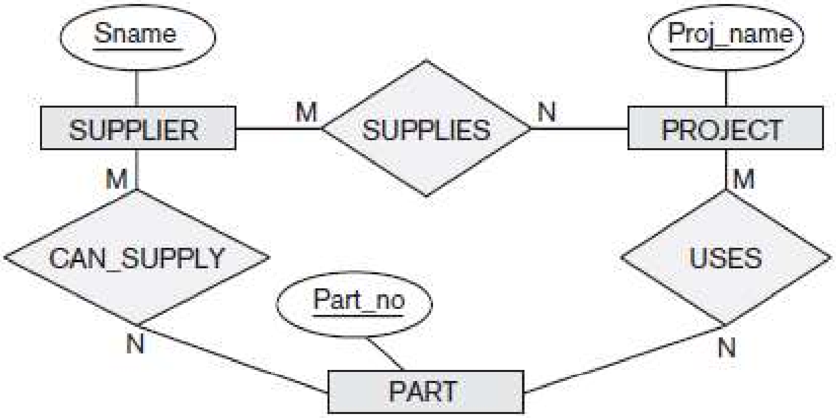


Fig 3.7.1 (b) ER diagram for three binary relationship types CAN_SUPPLY, USES, and SUPPLIES

Figure 3.7.1(b) shows an ER diagram for three binary relationship types CAN_SUPPLY, USES, and SUPPLIES. CAN_SUPPLY between SUPPLIER and PART, includes an instance (s, p) whenever supplier s can supply part p (to any project). USES between PROJECT and PART, includes an instance (j, p) whenever project j uses part p. SUPPLIES between SUPPLIER and PROJECT, includes an instance (s, j) whenever supplier s supplies some part to project j.

Some database design tools are based on variations of the ER model that permit only binary relationships. In this case, a ternary relationship such as SUPPLY must be represented as a weak entity type, with no partial key and with three identifying relationships.

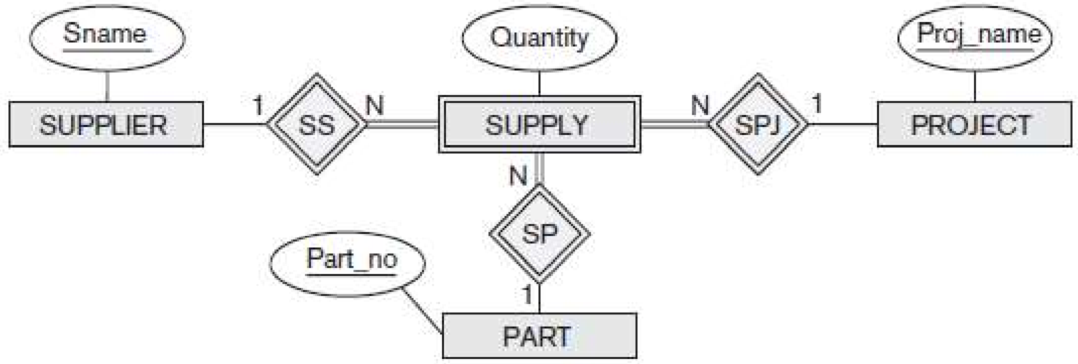


Fig 3.7.1(c): SUPPLY represented as a weak entity type

The three participating entity types SUPPLIER, PART, and PROJECT are together the owner entity types.Hence, an entity in the weak entity type is identified by the combination of its three owner entities from SUPPLIER, PART, and PROJECT.

3.7.2 Constraints on Ternary (or Higher-Degree) Relationships

There are two notations for specifying structural constraints on n-ary relationships

1. based on the cardinality ratio notation of binary relationships displayed
   - 1, M, or N is specified on each participation arc (both M and N symbols stand for many or any number)
2. based on the (min, max) notation
   - specifies that each entity is related to at least min and at most max relationship instances in the relationship set

3.8 Specialization and Generalization

3.8.1 Specialization

Specialization is the process of defining a *set of subclasses* of an entity type; this entity type is called the **superclass** of the specialization. The set of subclasses that forms a specialization is defined on the basis of some distinguishing characteristic of the entities in the superclass.

For example, the set of subclasses {SECRETARY, ENGINEER, TECHNICIAN} is a specialization of the superclass EMPLOYEE that distinguishes among employee entities based on the job type of each employee entity. We may have several specializations of the same entity type based on different distinguishing characteristics. For example, another specialization of the EMPLOYEE entity type may yield the set of subclasses {SALARIED_EMPLOYEE, HOURLY_EMPLOYEE}; this specialization distinguishes among employees based on the method of pay.

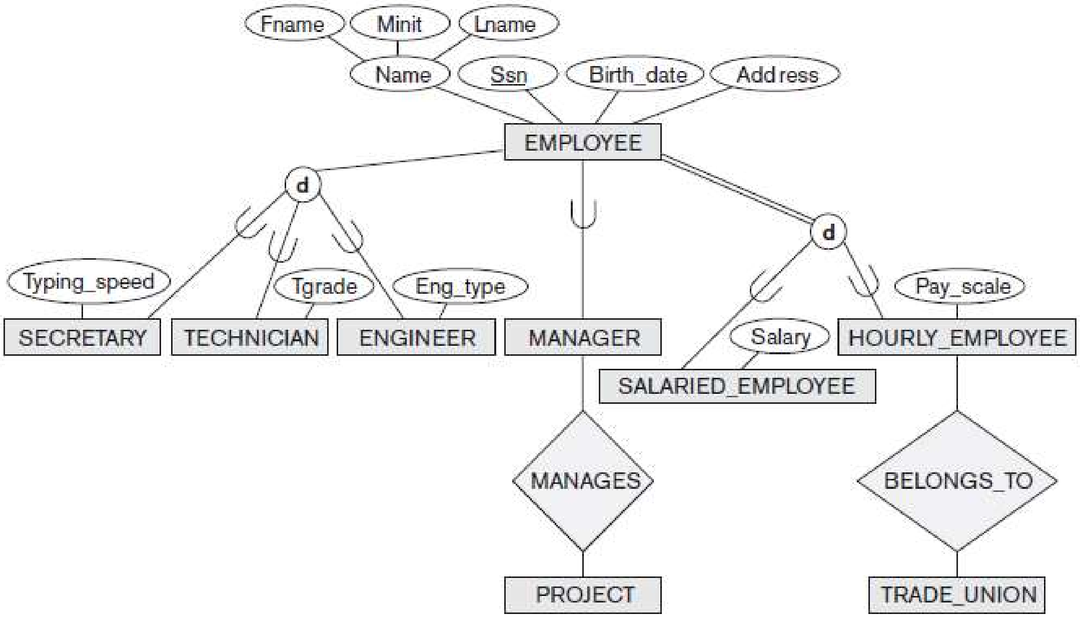


Figure 3.8.1(a): EER diagram notation to represent subclasses and specialization.

Figure 3.8.1(a) shows how we represent a specialization diagrammatically in an EER diagram. The subclasses that define a specialization are attached by lines to a circle that represents the specialization, which is connected in turn to the superclass. The *subset symbol* on each line connecting a subclass to the circle indicates the direction of the superclass/subclass relationship.5 Attributes that apply only to entities of a particular subclass—such as TypingSpeed of

SECRETARY—are attached to the rectangle representing that subclass. These are called **specific attributes** (or **local attributes**) of the subclass.

Similarly, a subclass can participate in **specific relationship types**, such as the HOURLY_EMPLOYEE subclass participating in the BELONGS_TO relationship in Figure Figure 3.8.1(b).

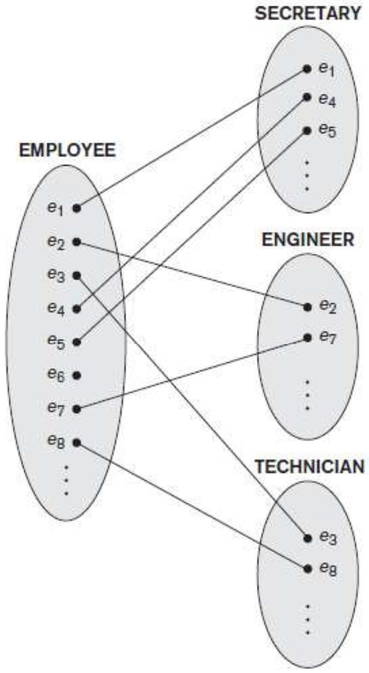


Figure 3.8.1(b): Instances of a specialization

Figure 3.8.1 (b) shows a few entity instances that belong to subclasses of the {SECRETARY, ENGINEER, TECHNICIAN} specialization. An entity that belongs to a subclass represents the same real-world entity as the entity connected to it in the EMPLOYEE superclass, even though the same entity is shown twice; for example, e1 is shown in both EMPLOYEE and SECRETARY.

There are two main reasons for including class/subclass relationships and specializations in a data model.

- The first is that certain attributes may apply to some but not all entities of the superclass. A subclass is defined in order to group the entities to which these attributes apply. The members of the subclass may still share the majority of their attributes with the other members of the superclass.
- The second reason for using subclasses is that some relationship types may be participated in only by entities that are members of the subclass. For example, if only HOURLY_EMPLOYEES can belong to a trade union, we can represent that fact by

creating the subclass HOURLY_EMPLOYEE of EMPLOYEE and relating the subclass to an entity type TRADE_UNION via the BELONGS_TO relationship type

In summary, the specialization process allows us to do the following:

- Define a set of subclasses of an entity type
- Establish additional specific attributes with each subclass
- Establish additional specific relationship types between each subclass and other entity types or other subclasses

3.8.2 Generalization

Generalization process can be viewed as being functionally the inverse of the specialization process. It is a process of defining a generalized entity type from the given entity types. Generalization is the reverse process of abstraction in which we suppress the differences among several entity types, identify their common features, and **generalize** them into a single **superclass**

For example, consider the entity types CAR and TRUCK shown in Figure 3.8.2(a).

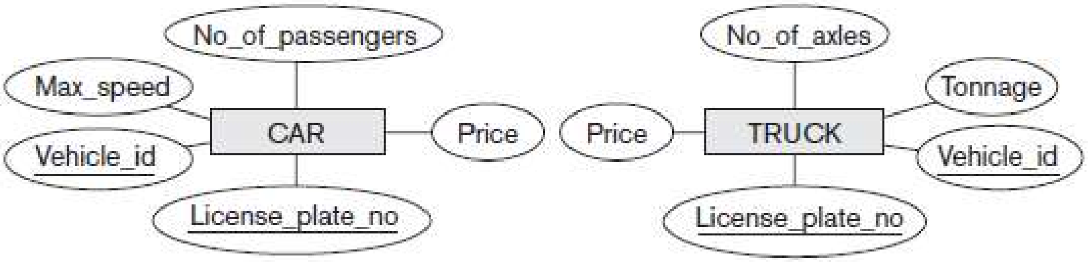


Figure 3.8.2(a): Two entity types, CAR and TRUCK

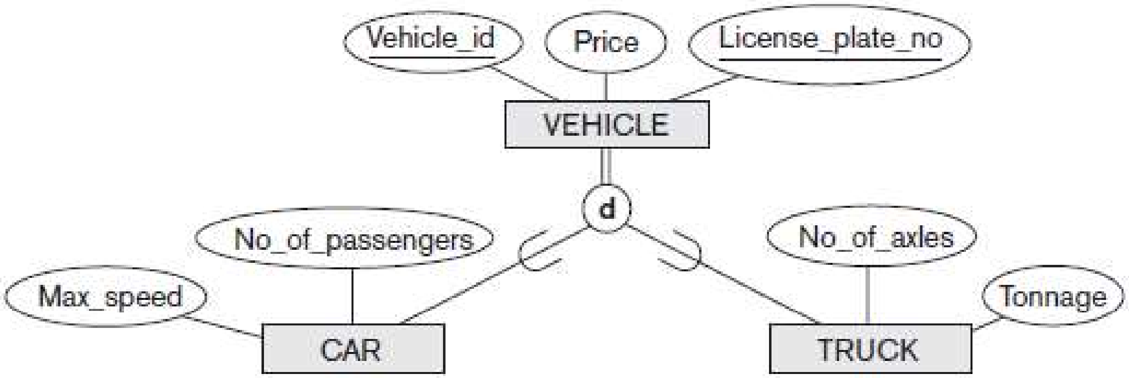


Figure 3.8.2(b): Generalizing CAR and TRUCK into the superclass VEHICLE.

Because they have several common attributes, they can be generalized into the entity type VEHICLE, as shown in Figure 3.8.2(b). Both CAR and TRUCK are now subclasses of the generalized superclass VEHICLE.

A diagrammatic notation to distinguish between generalization and specialization is used in some design methodologies. An arrow pointing to the generalized superclass represents a generalization, whereas arrows pointing to the specialized subclasses represent a specialization.

Question Bank

1. Define the following terms:
   i) data ii) database iii) DBMS iv) program-data independence v) Canned transaction
2. Define the database and briefly explain the implicit properties of the database.
3. Discuss the main Characteristics of the database approach and how does it differ from Traditional file systems?
4. What are the different types of database end users? Discuss the main activities of each.
5. Breifly discuss the advantages of using the DBMS.
6. Define the following terms:
   i) data mode ii) database schema iii) database state iv) schema diagram
7. Describe the three-schema architecture. Why do we need mappings between schema levels?
8. What is the difference between logical data independence and physical data independence?
9. What is the difference between procedural and nonprocedural DMLs?
10. Discuss the various database languages.
11. Discuss the different types of user-friendly interfaces and the types of users who typically use each.
12. Explain the component modules of DBMS and their interaction with the help of a diagram.
13. Discuss some types of database utilities and tools and their functions.
14. Explain two-tier and three-tier architecture.
15. Discuss the classification of database management systems.
16. Explain with a neat diagram, the phases of database design.
17. Define the following terms:
    i) Entity ii) attribute iii) entity type iv)entity set v) key attribute vi) value set v)degree of a relationship type vi) role names vii) cardinality ratio viii) participation constraints
18. Explain the different types of attributes that occur in an ER model with an example.
19. What is meant by a recursive relationship type? Give some examples of recursive relationship types.
20. What is a weak entity type? Explain the role of partial key in the design of weak entity type.
21. List symbols used in ER diagram and their meaning.

22. Discuss the naming conventions used for ER schema diagrams.
23. Explain with an example specialization and generalization.
24. Design an ER diagram for an insurance company. Assume suitable entity types like CUSTOMER, AGENT, BRANCH, POLICY, PAYEMENT and the relationship between them.
25. Design an ER - diagram for the Movie - database considering the following requirements:

    i) Each Movie is identifies by its title and year of release, it has length in minutes and can have zero of more quotes, language.

    ii) Production companies are identified by Name, they have address, and each production company can produce one or more movies.

    iii) Actors are identified by Name and Date of Birth, they can act in one or more movies and
    each actor has a role in a movie.

    iv) Directors are identified by Name and Date of Birth, so each Director can direct one or more movie and each movie can be directed by one or more Directors.

    v) Each movie belongs to anyone category like Horror, action, Drama, etc.
26. Design an Entity Relationship (ER) model for a college database . Say we have the following statements.

    1. A college contains many departments
    2. Each department can offer any number of courses
    3. Many instructors can work in a department
    4. An instructor can work only in one department
    5. For each department there is a Head
    6. An instructor can be head of only one department
    7. Each instructor can take any number of courses
    8. A course can be taken by only one instructor
    9. A student can enroll for any number of courses
    10. Each course can have any number of students
27. Consider the following set of requirements for a UNIVERSITY database that is used to keep track of students' transcripts

    a. The university keeps track of each student's name, student number, Social Security number, current address and phone number, permanent address and phone number, birth date, sex, class (freshman, sophomore, ..., graduate), major department, minor department (if any), and degree program (B.A., B.S., ..., Ph.D.). Some user applications need to refer to the city, state, and ZIP Code of the student's permanent address and to the student's last name. Both Social Security number and student number have unique values for each student.

b. Each department is described by a name, department code, office number, office phone number, and college. Both name and code have unique values for each department.

c. Each course has a course name, description, course number, number of semester hours, level, and offering department. The value of the course number is unique for each course.

d. Each section has an instructor, semester, year, course, and section number. The section number distinguishes sections of the same course that are taught during the same semester/year; its values are 1, 2, 3, ..., up to the number of sections taught during each semester.

e. A grade report has a student, section, letter grade, and numeric grade (0, 1, 2, 3, or 4).

Design an ER schema for this application, and draw an ER diagram for the schema. Specify key attributes of each entity type, and structural constraints on each relationship type. Note any unspecified requirements, and make appropriate assumptions to make the specification complete.

28. Write ER diagram for Airline reservation and Ban database