

Module 1

Chapter 1: Introduction to Databases

1.1 Introduction

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

Database

A **database** is a collection of related data. By **data**, we mean known facts that can be recorded 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**. 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.

Size/Complexity: A database can be of any size and complexity. For example, the list of names and addresses referred to earlier may consist of only a few hundred records, each with a simple structure. An example of a large commercial database is Amazon.com. It contains data for over 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 system that 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**.

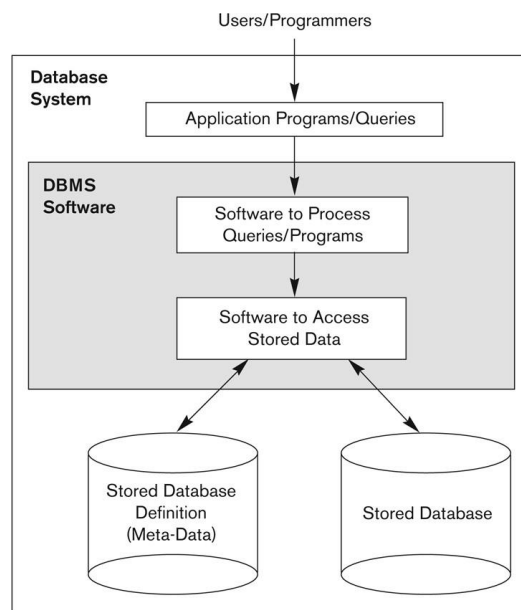


Fig 1.1(a): A simplified database system environment

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 is a string of alphabetic characters, Student_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 for Smith in the STUDENT file is related to two records in the GRADE_REPORT file that specify Smith's grades in two sections. Similarly, each record in the PREREQUISITE file 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 as follows:

- 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

As with software in general, design of a new application for an existing database or design of a brand 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 computerized tools so that it can be easily maintained, modified, and transformed into a database implementation.

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

1.2 Characteristics of the Database Approach

- The main characteristics of the database approach versus the file-processing approach are the following:
 1. Self-describing nature of a database system
 2. Insulation between programs and data, and data abstraction
 3. Support of multiple views of the data
 4. 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 ,every application 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-relational systems , users can define operations on data as part of the database definitions.
- 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 the data types of its arguments (or parameters). The *implementation* (or *method*) of the operation is

specified separately and can be changed without affecting the interface.

- User application programs can operate on the data by invoking these operations through 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-operation independence is called **data abstraction**.
- A DBMS provides users with a conceptual representation of data that does not include many of the details of how the data is stored or how the operations are implemented.

3. Support of Multiple Views of the Data

- A database typically has many users, each of whom may require a different perspective or **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.
- For example, one user of the database of Figure 1.2 may be interested only in accessing and printing the transcript of each student; 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.
- The DBMS includes **concurrency control** software to ensure that several users trying to update the same data do so in a controlled manner so that the result of the updates is correct.
- For example, when several reservation agents try to assign a seat on an airline flight, the DBMS should ensure that each seat can be accessed by only one agent at a time for assignment to a passenger. These types of applications are generally called **online**

transaction processing (OLTP) applications.

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.
- 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.
- By placing all the data together, we do not have to search multiple files to collect this data. This is known as denormalization.

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 provides a **security and authorization subsystem**, which the DBA use 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**.

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.

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.

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.

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.

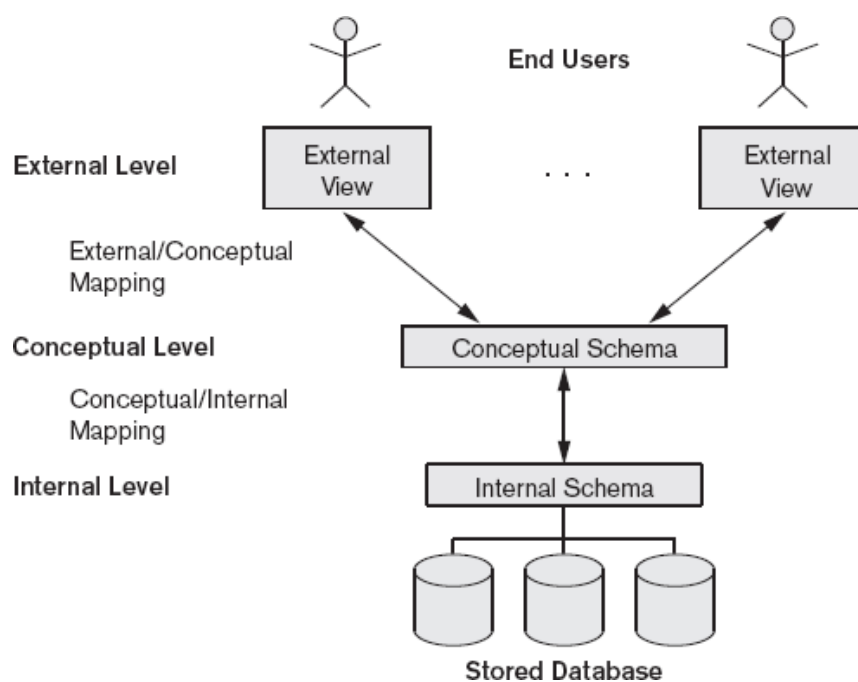


Figure 2.2: The three-schema architecture.

- 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 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.

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.
- 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.
- 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 rather the query is composed step by step picking options from a menu that is displayed by the system
- Pull-down menus are a very popular technique in Web-based user interfaces.

2. Apps for mobile devices

- Present mobile user with apps to access their data
- For example, banking, reservations and insurance companies provide apps that allow users to access their data through a mobile phone
- The apps have built-in programmed interfaces that allow users to login using their account name and password
- Provide a limited menu of options for mobile access to user data and options for paying bills, making reservations

3. 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.

4. 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.

5. 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.

6. Keyword-based database search

- Similar to web search engines which accepts strings of natural language words and matches them with documents at specific sites
- Use predefined indexes on words and use ranking functions to retrieve and present resulting documents

7. 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.

8. Interfaces for Parametric Users

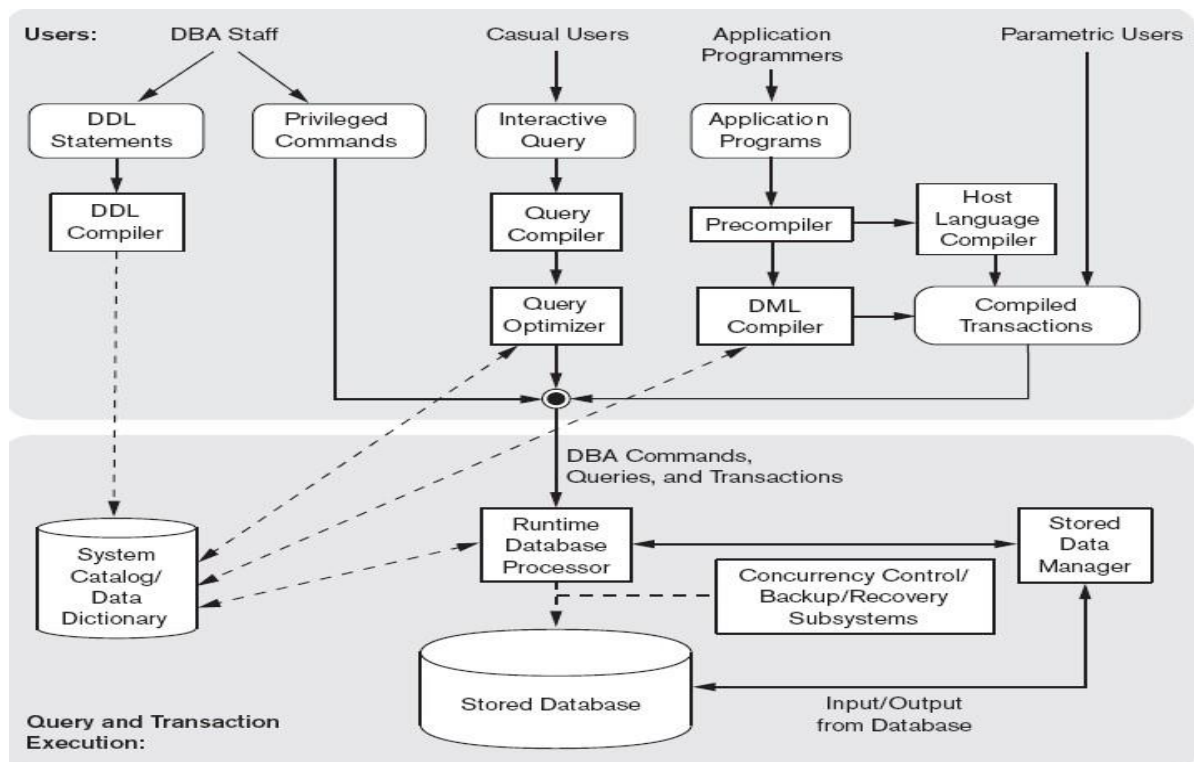
- Parametric users, such as bank tellers, often have a small set of operations that they must perform repeatedly.
- For example, a teller is able to use single function keys to invoke routine and repetitive transactions such as account deposits or withdrawals, or balance inquiries
- Usually a small set of abbreviated commands is included, with the goal of minimizing the number of keystrokes required for each request

9. 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:
 - the privileged commands
 - the executable query plans, and
 - the canned transactions with runtime parameters.
- **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 development, 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

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

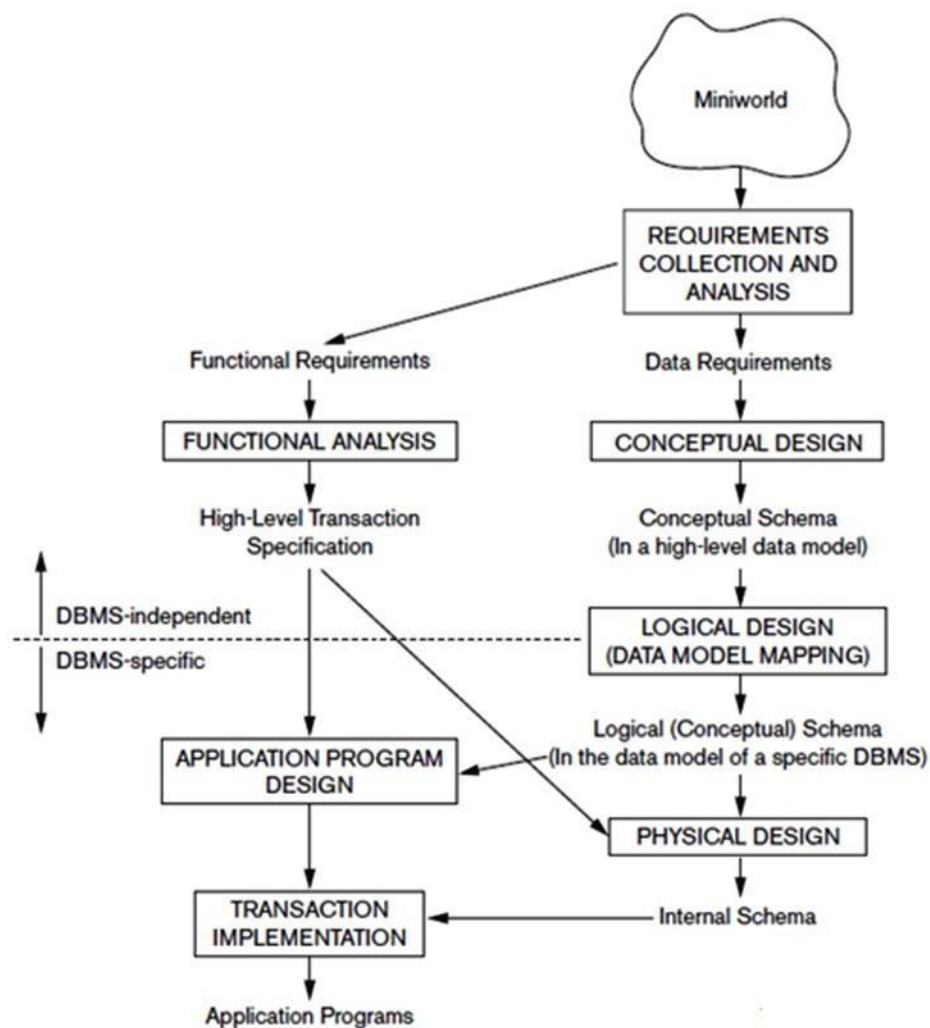


Figure 3.1: A simplified diagram to illustrate the main phases of database design.

- 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 user defined **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.

1.2 Entity Types, Entity Sets, Attributes, and Keys

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

1.2.1 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.

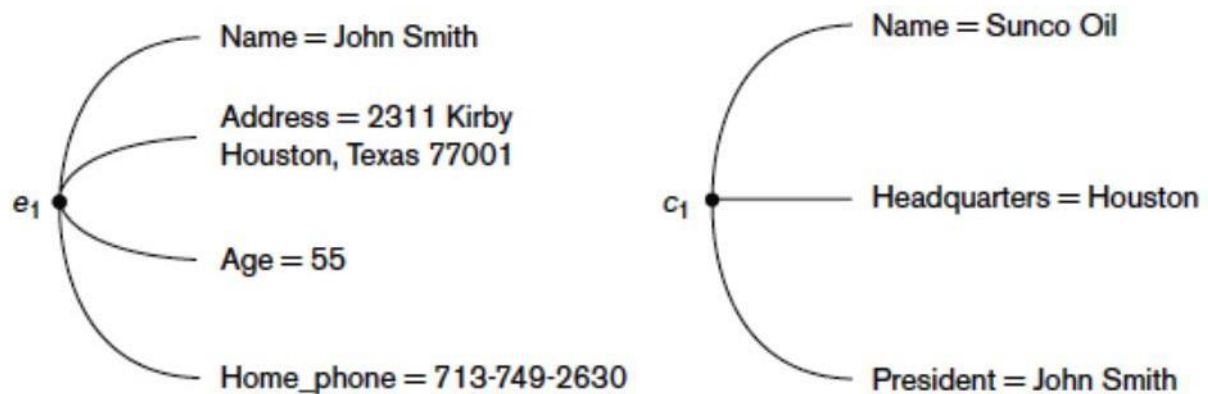


Figure 3.2.1(a): Two entities, EMPLOYEE e_1 , and COMPANY c_1 , and their attributes

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, AGE of Person.

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.

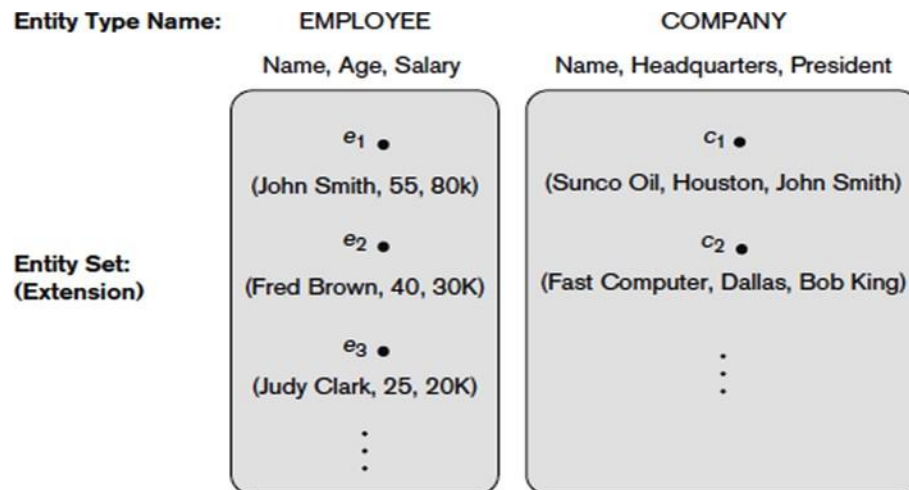


Figure 3.2.2(a): Two entity types, EMPLOYEE and COMPANY, and some member entities of each.

- 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.

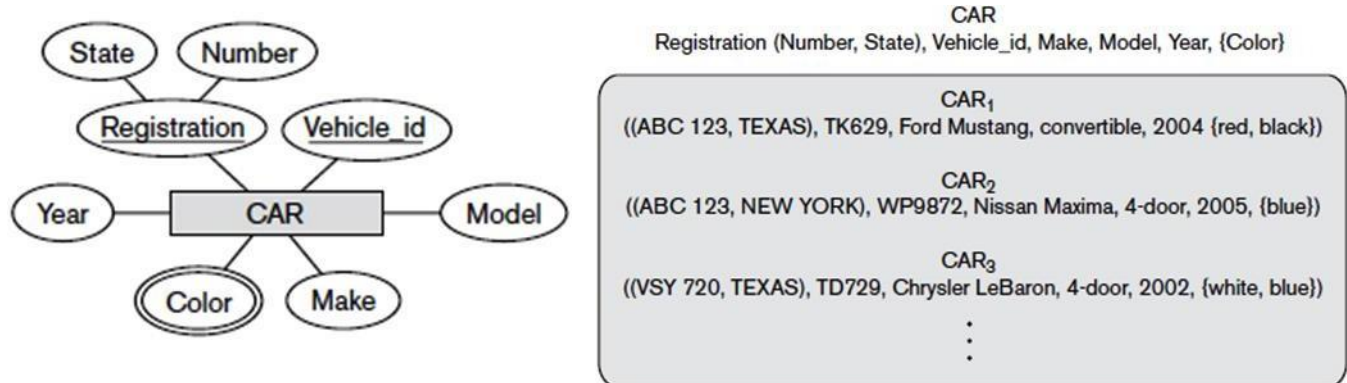


Figure 3.2.2(b) : ER diagram notation

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**:

$$A : E \rightarrow P(V).$$

- 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.

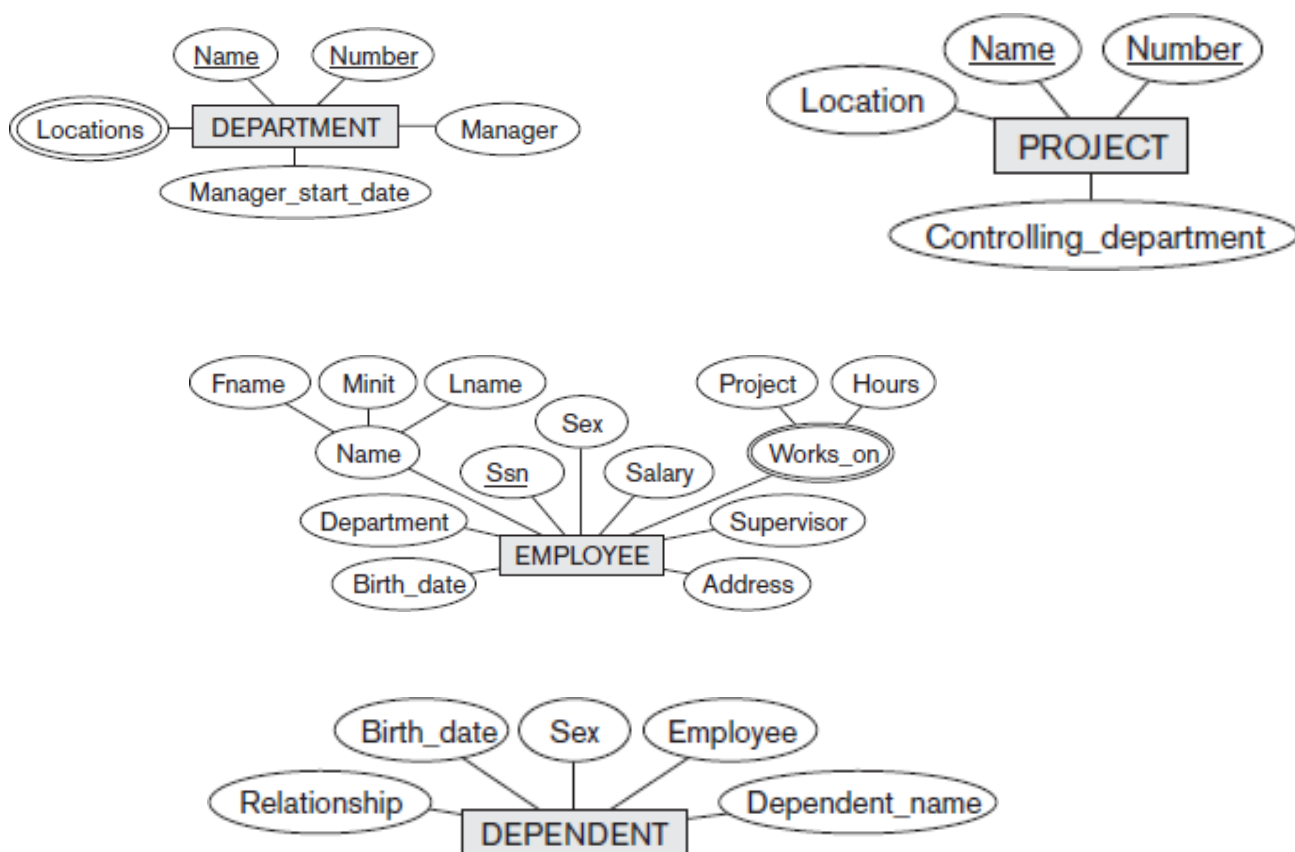


Figure 3.3(a): Preliminary design of entity types for the COMPANY database. Some of the shown attributes will be refined into relationships.

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

1.4.1 Relationship Types, Sets, and Instances

A **relationship type R** among n entity types $E_1, E_2, \dots, E_n$ 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 .

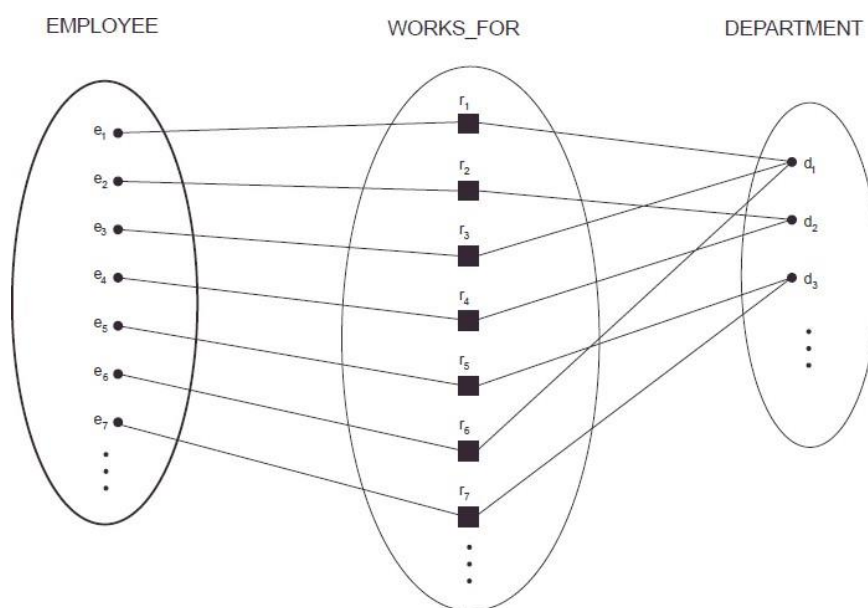


Figure 3.4.1: Some instances in the WORKS_FOR relationship set, which represents a relationship type WORKS_FOR between EMPLOYEE and DEPARTMENT.

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(refer ppt)

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

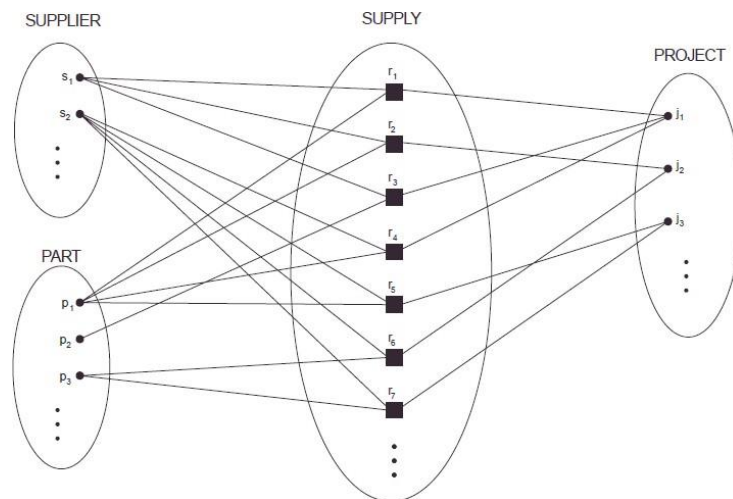


Figure 3.4.2(a): Some relationship instances in the SUPPLY ternary relationship set.

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.
- 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.
- 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)..

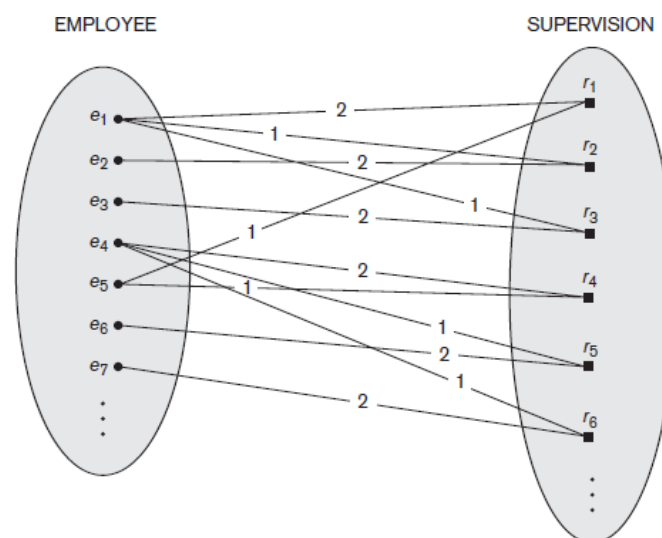


Figure 3.4.2(b): A recursive relationship SUPERVISION between EMPLOYEE in the supervisor role (1) and EMPLOYEE in the subordinate role (2).

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. T
- hese 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 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:
 1. Total
 2. 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

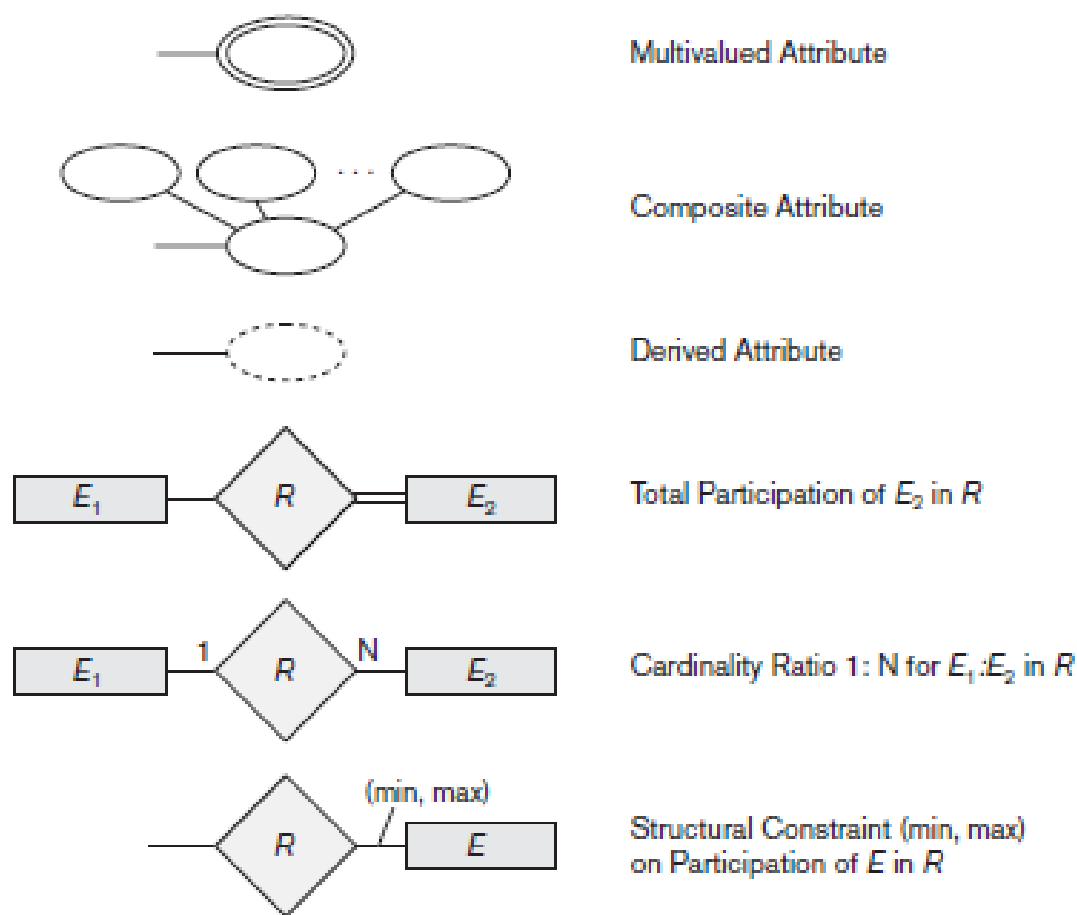
- Entity types that do not have key attributes of their own are called **weak entity** types.
- Entities belonging to a weak entity type are identified by being related to specific entities from another entity type in combination with one of their attribute values.
- We call this other entity type the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

ER Diagrams, Naming Conventions, and Design Issues

3.6.1 Summary of Notation for ER Diagrams

Symbol	Meaning
	Entity
	Weak Entity
	Relationship
	Identifying Relationship
	Attribute
	Key Attribute



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.

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:

[Type here]

- 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.

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 $0 \leq \min \leq \max$ and $\max \geq 1$.

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.

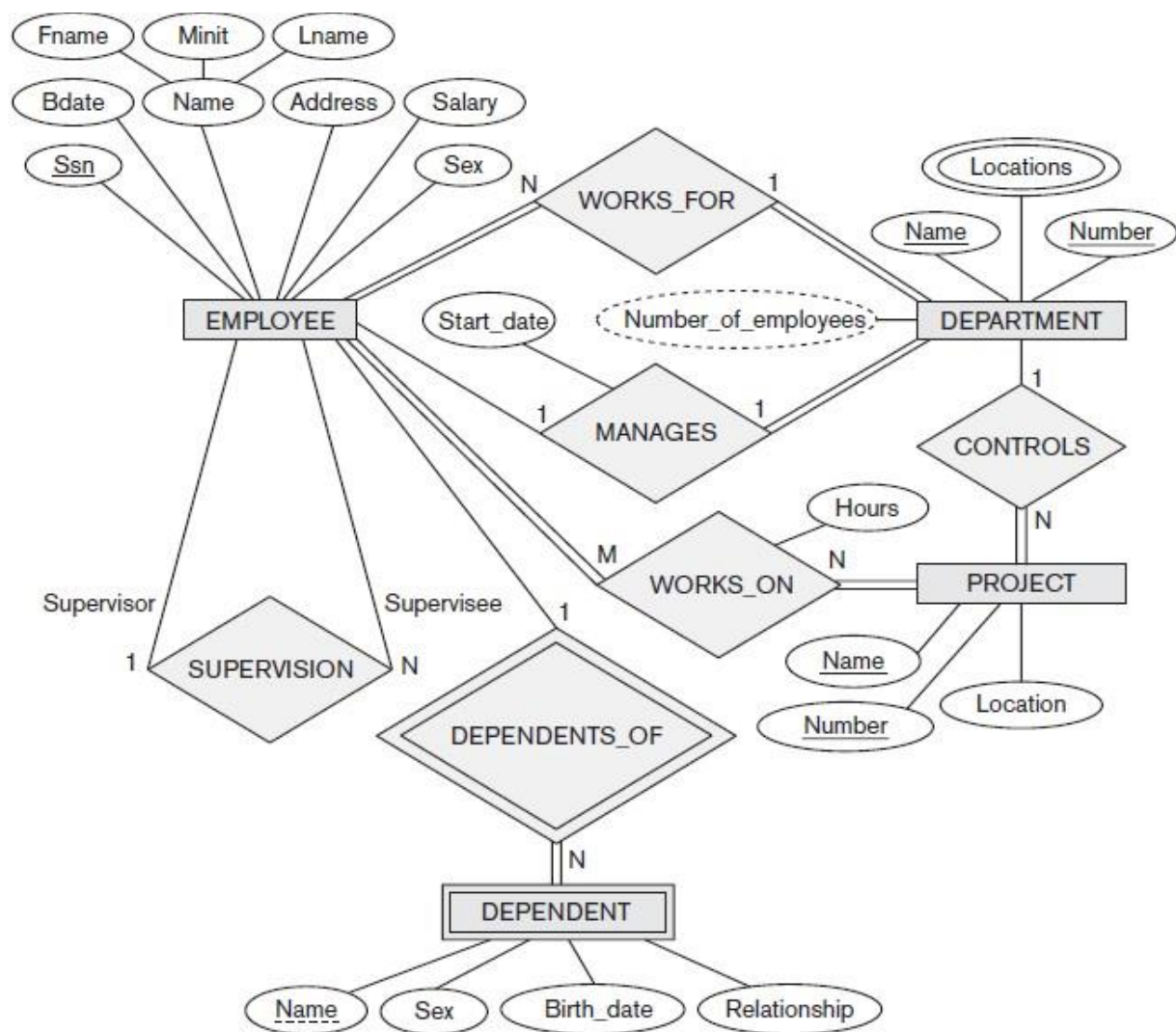


Figure 3.6.4 (a)

Figure 3.6.4 (a) : ER diagram for Company Database

[Type here]

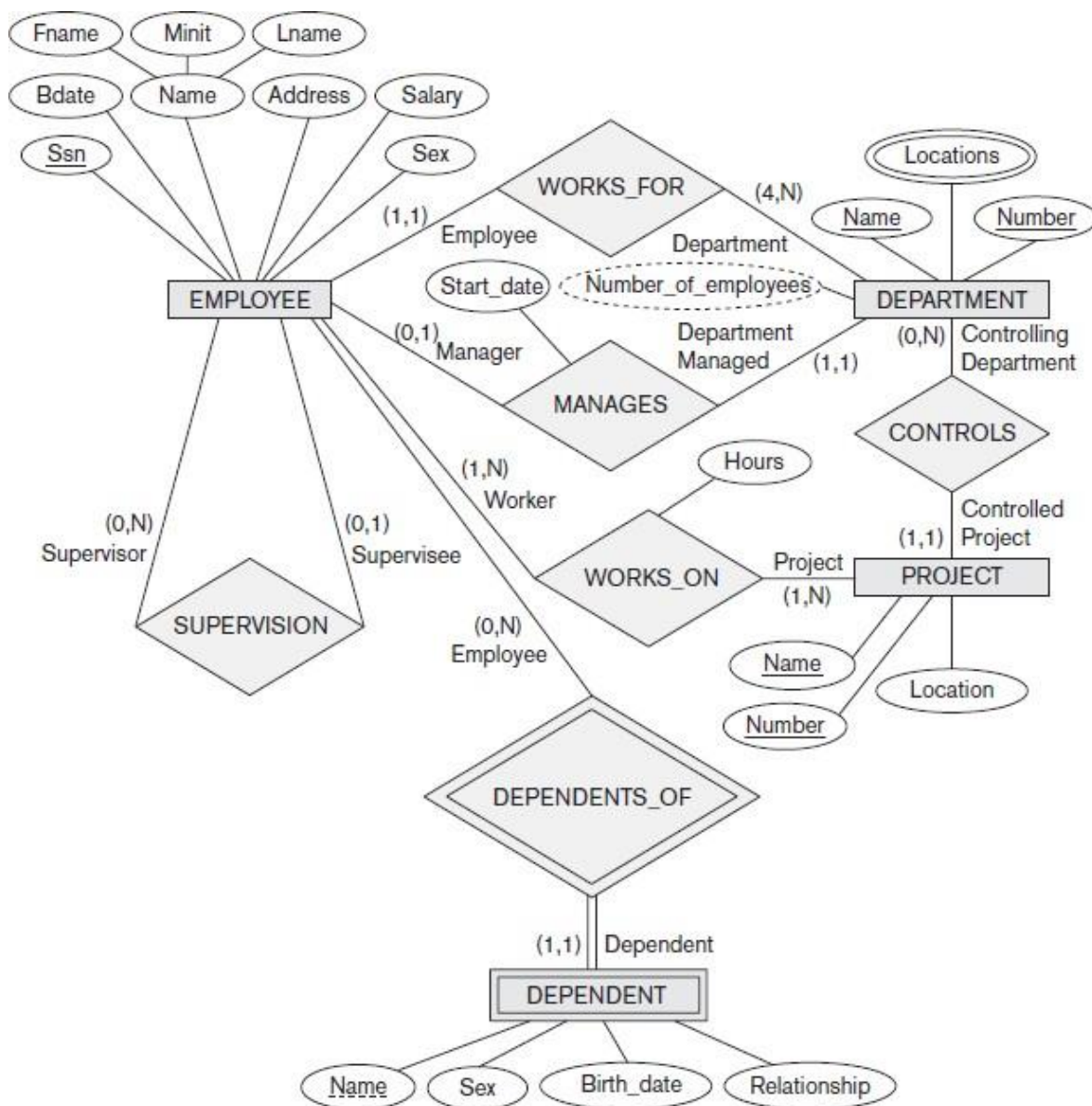


Figure 3.6.4(b): ER diagram for Company Database (using alternative notation)