Lecture Notes

DATABASE MANAGEMENT SYSTEM (BCS403)

Module – 1 **08 Hours**

Introduction to Databases: Introduction, Characteristics of database approach, Advantages of using the DBMS approach, History of database applications.

Overview of Database Languages and Architectures: Data Models, Schemas, and Instances. Three schema architecture and data independence, database languages, and interfaces, The Database System environment.

Conceptual Data Modelling using Entities and Relationships: Entity types, Entity sets and structural constraints, Weak entity types, ER diagrams, Specialization and Generalization.

Textbook 1: Ch 1.1 to 1.8, 2.1 to 2.6, 3.1 to 3.10 **RBT: L1, L2, L3**

Teaching-Learning Process Chalk and board, Active Learning, Problem based learning

Chapter 1

Introduction to Databases

1. Introduction to Databases

Databases and database technology have had a major impact on the growing use of computers.

A **database** is a collection of related data.

A data means that can be recorded and that have implicit meaning.

Simple examples: The names, telephone numbers, and addresses of the people you know.

Complex examples: Bank Transactions, Shopping systems etc.

A database has the following implicit properties:

■ A database 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.

■ A database is a logically coherent collection of data with some inherent meaning. A random assortment of data cannot correctly be referred to as a database.

■ A database 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.

A database can be of any size and complexity.

Example, the list of names and addresses referred to earlier may consist of only a few hundred records, each with a simple structure.

Complex example: the computerized catalog of a large library may contain half a million entries organized under different categories—by primary author's last name, by subject, by book title—with each category organized alphabetically.

A database may be generated and maintained manually or it may be computerized.

Amazon(2023): The database occupies over 42 terabytes (a terabyte is 1012 bytes worth of storage) and is stored on hundreds of computers (called servers)

A **database management system (DBMS)** is a computerized system that enables users to create and maintain a database.

The DBMS is a *general-purpose software system* that facilitates the processes of d*efining, 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. This 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.

An **application program** accesses the database by sending queries or requests for data to the DBMS. A **query** typically causes some data to be retrieved.

A **transaction** may cause some data to be read and some data to be written into the database.

Other important functions of DBMS:

Protecting the database -includes *system protection* against hardware or software malfunction (or crashes) and *security protection* against unauthorized or malicious access

***maintaining* it over a long period of time**-typical large database may have a life cycle of many years.

2. An Example

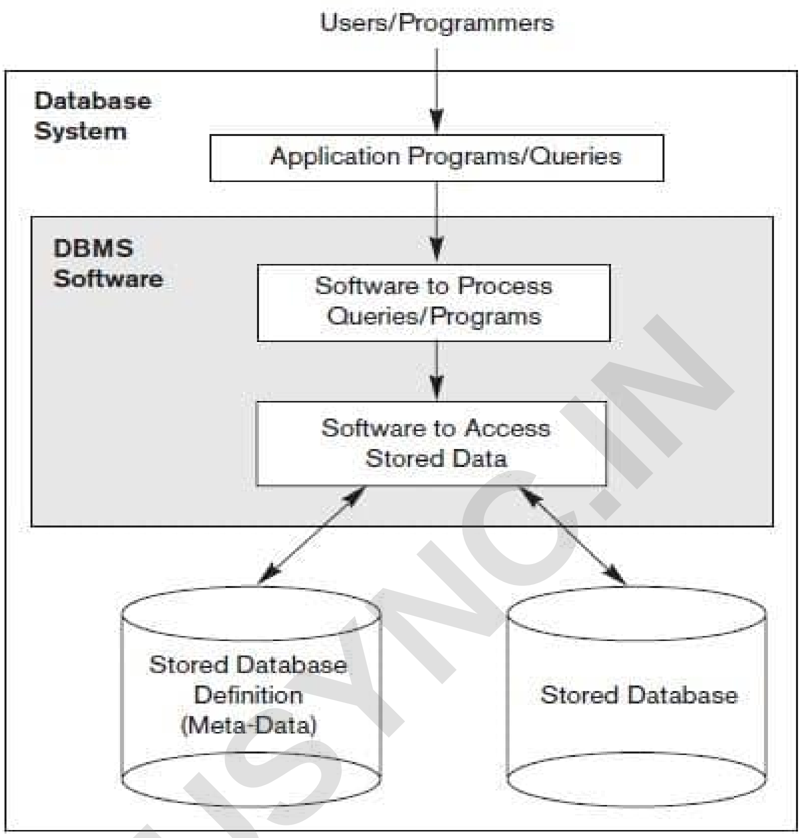
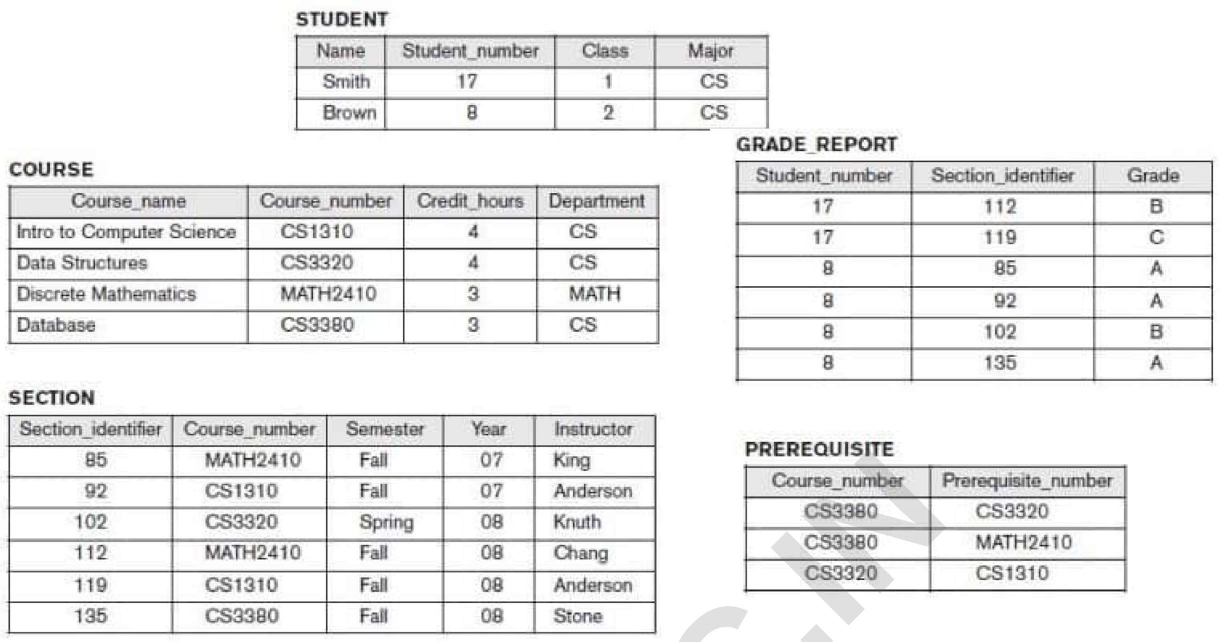

Users/Programmers
Database System
Application Programs/Queries
DBMS Software
Software to Process Queries/Programs
Software to Access Stored Data
Stored Database Definition (Meta-Data)
Stored Database

Figure 1.1: A simplified database system environment.

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



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

Figure1.2: A UNIVERSITY database for maintaining information concerning students, courses, and grades in a university environment

2. Characteristics of the Database Approach

Many characteristics are described that distinguish the database approach from the much older approach of writing customized programs to access data stored in files. In traditional **file processing**, each user defines and implements the files needed for a specific software application as part of programming the application.

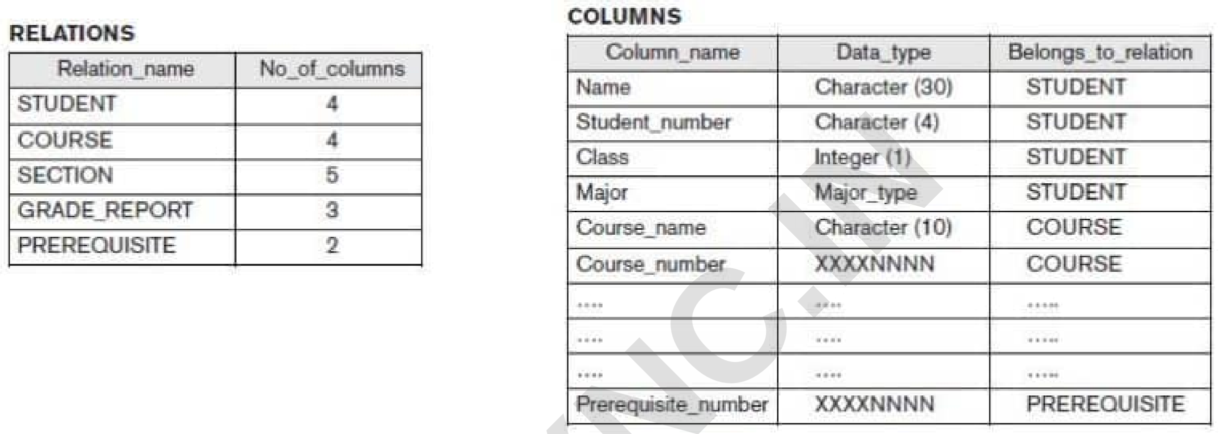
In the database approach, a single repository maintains data that is defined once and then accessed by various users repeatedly through queries, transactions, and application programs.

The main characteristics of the database approach are

- 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

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 definition is stored in the DBMS catalog and are called META-DATA.



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: An example of a database catalog for the database.

Insulation between Programs and Data, and Data Abstraction

The structure of data files is stored in the DBMS catalog separately from the access programs. We call this property **program-data independence**.

In some types of database systems, such as 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.

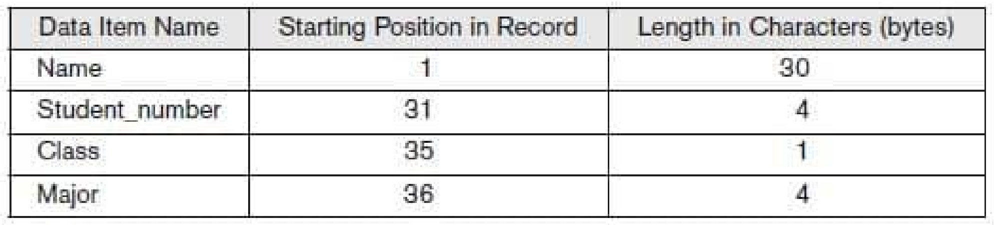
The *interface* (or *signature*) of an operation includes the operation name and the data types of its arguments (or parameters).

The *implementation* (or *method*) of the operation is specified separately and can be changed without affecting the interface.

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.

A data model is a type of data abstraction that is used to provide this conceptual representation



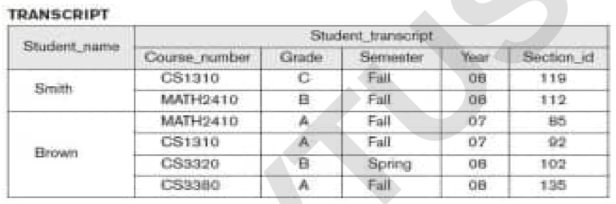
Data Item Name	Starting Position in Record	Length in Characters (bytes)
Name	1	30
Student_number	31	4
Class	35	1
Major	36	4

Figure: Internal storage format for a STUDENT record, based on the database catalog.

Support of Multiple Views of the Data

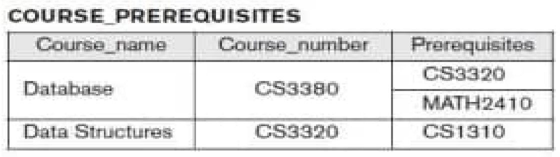
A database typically has many types of users, each of whom may require a different perspective or **view** of the database. A view may be a subset of the database or it may contain **virtual data** that is derived from the database files but is not explicitly stored.

A multiuser DBMS whose users have a variety of distinct applications must provide facilities for defining multiple views.



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



COURSE_PREREQUISITES

Course_name	Course_number	Prerequisites
Database	CS3380	CS3320
		MATH2410
Data Structures	CS3320	CS1310

Figure 1.5: Two views derived from the database (a) The TRANSCRIPT view. (b) The COURSE_PREREQUISITES view.

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 integrated and maintained in a single database. The DBMS must include **concurrency control** software to ensure that several users trying to update the same data

These types of applications are generally called **online transaction processing (OLTP)** applications. A fundamental role of multiuser DBMS software is to ensure that concurrent transactions operate correctly and efficiently.

A **transaction** is an *executing program* or *process* that includes one or more database accesses, such as reading or updating of database records.

The **isolation** property ensures that each transaction appears to execute in isolation from other transactions, even though hundreds of transactions may be executing concurrently.

The **atomicity** property ensures that either all the database operations in a transaction are executed or none are.

3. Actors on the Scene {8 marks question} *

Database Administrators

Database Designers

End Users

System Analysts and Application Programmers (Software Engineers)

1. Database Administrators

- Administering these resources is the responsibility of the **database administrator (DBA)**.
- The DBA is responsible for authorizing access to the database, coordinating and monitoring its use, and acquiring software and hardware resources as needed.
- The DBA is accountable for problems such as security breaches and poor system response time.

2. Database Designers

- **Database designers** are responsible for identifying the data to be stored in the database and for choosing appropriate structures to represent and store this data.
- These tasks are mostly undertaken before the database is actually implemented and populated with data.
- It is the responsibility of database designers to communicate with all prospective database users in order to understand their requirements and to create a design that meets these requirements.

3. **End Users**

End users are the people whose jobs require access to the database for querying, updating, and generating reports; the database primarily exists for their use.

Many categories of end users:

- **Casual end users** occasionally access the database, but they may need different information each time.
- **Naive** or **parametric end users** make up a sizable portion of database end users.
- **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.
- **Standalone users** maintain personal databases by using ready-made program packages that provide easy-to-use menu-based or graphics-based interfaces.

4. **System Analysts and Application Programmers (Software Engineers)**

   System analysts determine the requirements of end users, especially naive and parametric end users, and develop specifications for standard canned transactions that meet these requirements.

   Application programmers implement these specifications as programs; then they test, debug, document, and maintain these canned transactions.

Describe the role of Database administrator { 6 marks}

Explain the role and responsibilities of Database administrator, Database Designer, end users, and others in constructing of database. {8 marks}

4. Workers behind the Scene

- DBMS system designers and implementers design and implement the DBMS modules and interfaces as a software package.
- Tool developers design and implement tools—the software packages that facilitate database modeling and design, database system design, and improved performance.

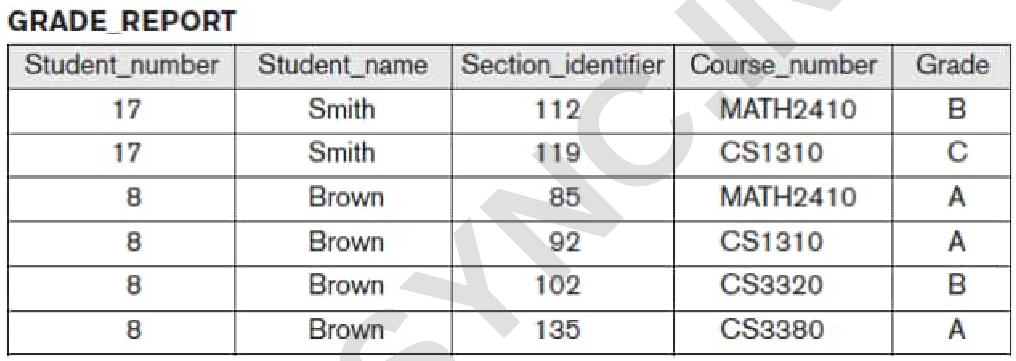
- Operators and maintenance personnel (system administration personnel) are responsible for the actual running and maintenance of the hardware and software environment for the database system.

Explain about the workers behind the database development { 6 marks}

5. Advantages of Using the DBMS Approach { 8 marks question}*

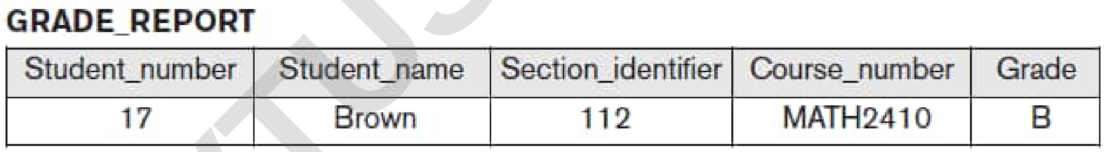
Advantages of using a DBMS and the capabilities that a good DBMS should possess.

Controlling Redundancy: storing the same data multiple times leads to several problems. *duplication of effort, storage space is wasted, inconsistent.*



GRADE_REPORT

Student_number	Student_name	Section_identifier	Course_number	Grade
17	Smith	112	MATH2410	B
17	Smith	119	CS1310	C
8	Brown	85	MATH2410	A
8	Brown	92	CS1310	A
8	Brown	102	CS3320	B
8	Brown	135	CS3380	A



GRADE_REPORT

Student_number	Student_name	Section_identifier	Course_number	Grade
17	Brown	112	MATH2410	B

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.
- A DBMS should provide a **security and authorization subsystem**, which the DBA uses to create accounts and to specify account restrictions. Then, the DBMS should enforce these restrictions automatically.

Providing Persistent Storage for Program Objects

Databases can be used to provide **persistent storage** for program objects and data structures.

Providing Storage Structures and Search Techniques for Efficient Query Processing

Database systems must provide capabilities for *efficiently executing queries and updates.* Because the database is typically stored on disk, the DBMS must provide specialized data structures and search techniques to speed up disk search for the desired records.

Auxiliary files called **indexes** are often used for this purpose.

DBMS often has a **buffering** or **caching** module that maintains parts of the database in main memory buffers.

Providing Backup and Recovery

A DBMS must provide facilities for recovering from hardware or software failures.

The **backup and recovery subsystem** of the DBMS is responsible for recovery.

Providing Multiple User Interfaces

DBMS should provide a variety of user interfaces. These include apps for mobile users, query languages for casual users, programming language interfaces for application programmers, forms and command codes for parametric users, and menu-driven interfaces and natural language interfaces for standalone users.

Representing Complex Relationships among Data

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

Enforcing Integrity Constraints

Most database applications have certain **integrity constraints** that must hold for the data. A DBMS should provide capabilities for defining and enforcing these constraints.

Permitting Inferencing and Actions Using Rules and Triggers

Some database systems provide capabilities for defining *deduction rules* for *inferencing* new information from the stored database facts. Such systems are called **deductive database systems**.

Additional Implications of Using the Database Approach

1. **Potential for Enforcing Standards.** The database approach permits the DBA to define and enforce standards among database users in a large organization.
2. **Reduced Application Development Time.** A prime selling feature of the database approach is that developing a new application.
3. **Flexibility.** It may be necessary to change the structure of a database as requirements change.

4. **Availability of Up-to-Date Information.** A DBMS makes the database available to all users.
5. **Economies of Scale.** The DBMS approach permits consolidation of data and applications.

List and explain the advantages of DBMS approach over File system approach { 8 marks}

6. History of DBMS { 8 marks question}*

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.

Early experimental relational systems developed in the late 1970s and the commercial relational database management systems (RDBMS).

Object-Oriented Applications and the Need for More Complex Databases

The emergence of object-oriented programming languages in the 1980s and the need to store and share complex, structured objects led to the development of object-oriented databases (OODBs).

They also incorporated many of the useful object-oriented paradigms, such as abstract data types, encapsulation of operations, inheritance, and object identity.

They are now mainly used in specialized applications, such as engineering design, multimedia publishing, and manufacturing systems.

Interchanging Data on the Web for E-Commerce Using XML

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

The eXtended Markup Language (XML) is one 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.

- **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 MRI (magnetic resonance imaging) tests
- **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 to search for the occurrences of specific patterns or relationships, and for identifying unusual patterns in areas such as credit card fraud detection
- **Spatial** applications that store and analyze spatial locations of data, such as weather information, maps used in geographical information systems, and 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.

Emergence of Big Data Storage Systems and NOSQL Databases

In twenty-first century, the proliferation of applications and platforms such as social media Web sites, large e-commerce companies, Web search indexes, and cloud storage/backup led to a surge in the amount of data stored on large databases and massive servers.

New types of database systems were necessary to manage these huge databases—systems that would provide fast search and retrieval as well as reliable and safe storage of nontraditional types of data, such as social media posts and tweets.

The term *NOSQL* is generally interpreted as Not Only SQL, meaning that in systems than manage large amounts of data, some of the data is stored using SQL systems**.**

Explain the history and growth of DBMS { 8 marks}

7. When Not to Use a DBMS

The overhead costs of using a DBMS

- 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.

Desirable to develop customized database applications 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

1. Data Models, Schemas, and Instances

Data abstraction generally refers to the suppression of details of data organization and storage, and the highlighting of the essential features for an improved understanding of data.

A data model—a collection of concepts that can be used to describe the structure of a database—provides the necessary means to achieve this abstraction.

Most data models also include a set of basic operations for specifying retrievals and updates on the database.

An example of a user-defined operation could be COMPUTE_GPA, which can be applied to a STUDENT object.

On the other hand, generic operations to insert, delete, modify, or retrieve any kind of object are often included in the *basic data model operations*.

Categories of Data Models

High-level or conceptual data models provide concepts that are close to the way many users perceive data.

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

Between these two extremes is a class of representational (or implementation) data models, which 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.

Conceptual data models use concepts such as entities, attributes, and relationships.

An **entity** represents a real-world object or concept, such as an employee or a project from the miniworld that is described in the database.

An **attribute** represents some property of interest that further describes an entity, such as the employee’s name or salary.

A **relationship** among two or more entities represents an association among the entities, for example, a works-on relationship between an employee and a project.

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—that have been widely used in the past.

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

The object data model as an example of a new family of higher-level implementation data models that are closer to conceptual data models.

A standard for object databases called the ODMG object model has been proposed by the Object Data Management Group (ODMG).

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.

An access path is a search structure that makes the search for particular database records efficient, such as indexing or hashing.

Another class of data models is known as **self-describing data models**.

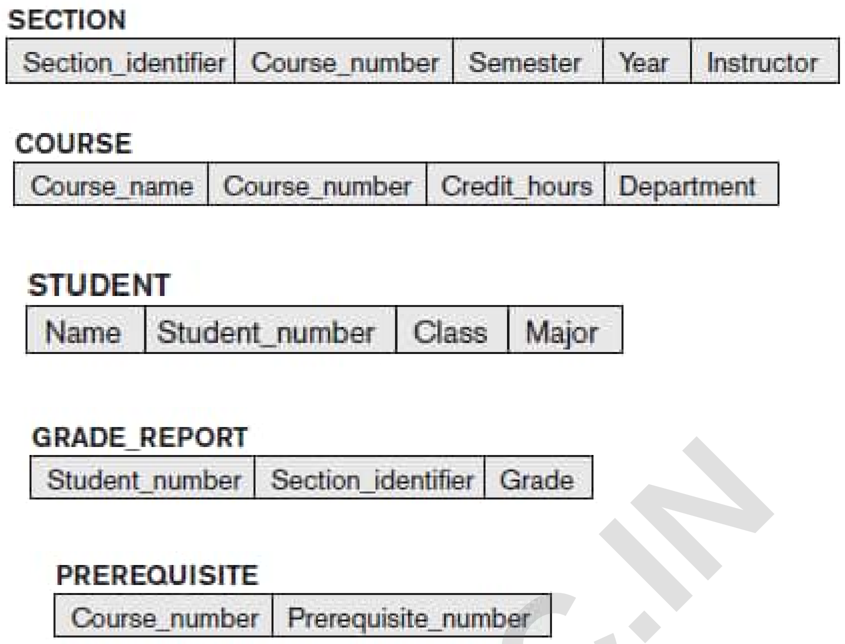
In traditional DBMSs, the description (schema) is separated from the data. These models include **XML** as well as many of the **key-value stores** and **NOSQL systems** that were recently created for managing big data.

Schemas, Instances, and Database State

In a data model, it is important to distinguish between the *description* of the database and the *database itself*.

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

Most data models have certain conventions for displaying schemas as diagrams called **schema diagram.**


SECTION
Section_identifier | Course_number | Semester | Year | Instructor
COURSE
Course_name | Course_number | Credit_hours | Department
STUDENT
Name | Student_number | Class | Major
GRADE_REPORT
Student_number | Section_identifier | Grade
PREREQUISITE
Course_number | Prerequisite_number

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.

The actual data in a database may change quite frequently.

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.

The distinction between database schema and database state is very important.

empty state with no data.

Initial state of the database when the database is first **populated** or **loaded** with the initial data.

Valid state—that is, a state that satisfies the structure and constraints specified in the schema.

At any point in time, the database has a *current state.*

2. The Three-Schema Architecture {8 to 10 marks}*

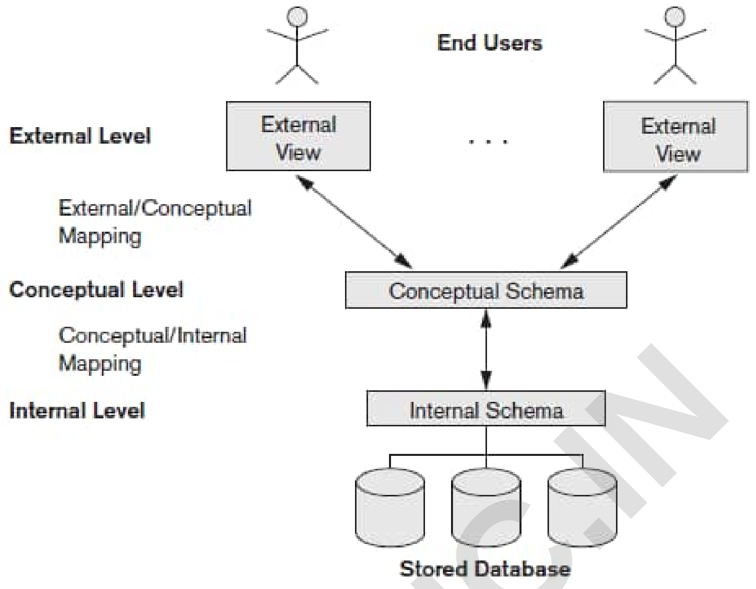


Figure: three schema architecture.

- The **internal level** has an **internal schema**, which describes the physical storage structure of the database.
- The **conceptual level** has a **conceptual schema**, which describes the structure of the whole database for a community of users.
- The **external** or **view level** includes a number of **external schemas** or **user views**.

The three-schema architecture is a convenient tool with which the user can visualize the schema levels in a database system.

The three-level ANSI architecture has an important place in database technology development because it clearly separates the users' external level, the database's conceptual level, and the internal storage level for designing a database.

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.

The processes of transforming requests and results between levels are called **mappings**.

Describe three schema architecture. Why do we need mappings between schema levels? {8 marks}*

2. Database Languages and Interfaces

DBMS Languages

DBMS implementation details is defined by the following languages.

Data Definition Language (**DDL**), is used by the DBA and by database designers to define both schemas.

View Definition Language (**VDL**), to specify user views and their mappings to the conceptual schema, but in most DBMSs *the DDL is used to define both conceptual and external schemas*.

Storage Definition Language (**SDL**), is used to specify the internal schema.

Data Manipulation Language (**DML**) provides a set of instructions for the manipulations include retrieval, insertion, deletion, and modification of the data.

Q. Discuss various forms of languages used in implementation of DBMS {6 marks}*

DBMS Interfaces

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

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.

Pull-down menus are a very popular technique in **Web-based user interfaces**.

Apps for Mobile Devices.

These interfaces present mobile users with access to their data. For example, banking, reservations, and insurance companies, among many others, provide apps that allow users to access their data through a mobile phone or mobile device.

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.

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.

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.

Keyword-based Database Search.

These are somewhat similar to Web search engines, which accept strings of natural language (like English or Spanish) words and match them with documents at specific sites (for local search engines) or Web pages on the Web at large (for engines like Google or Ask).

Speech Input and Output.

Limited use of speech as an input query and speech as an answer to a question or result of a request is becoming commonplace.

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.

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.

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.

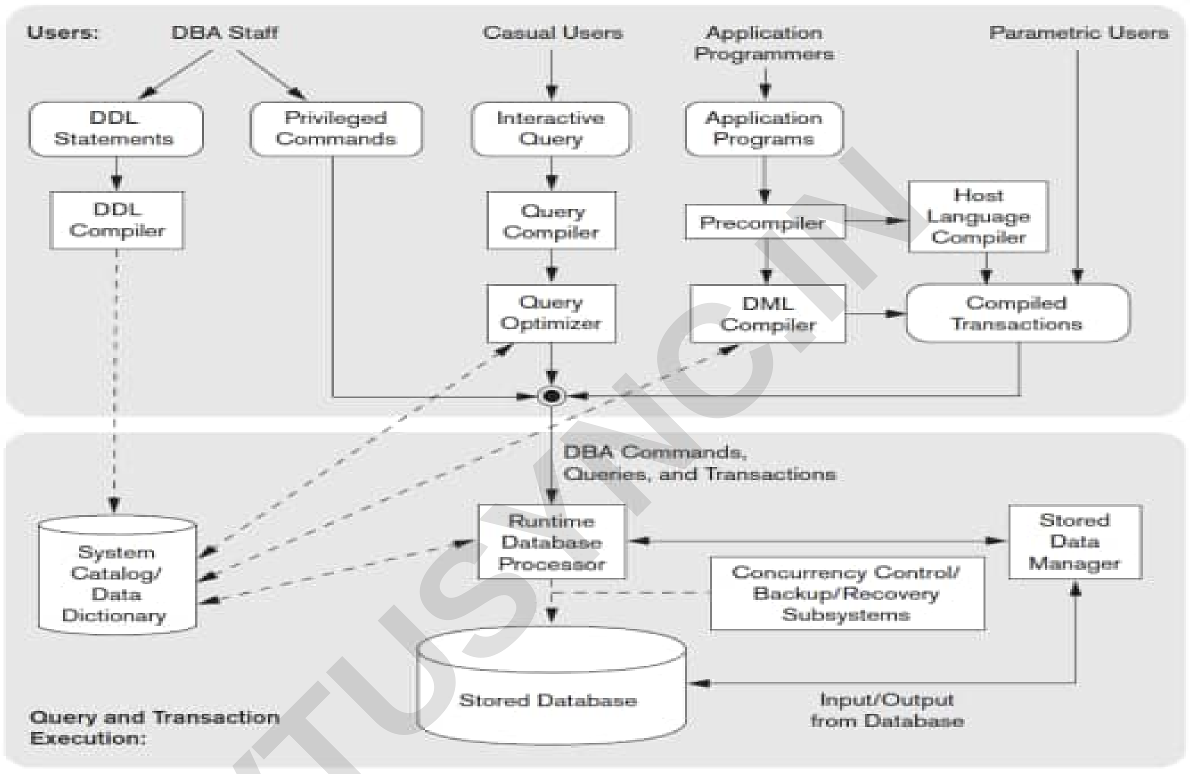
Q. Discuss various ways of languages and interfaces used in DBMS {6 marks}*

Q. Explain the different interfaces utilized in accessing DBMS. {6 marks}

3. The Database System Environment {8 marks} *

DBMS Component Modules

A DBMS is a complex software system.



- The database and the DBMS catalog are usually stored on disk.
- Access to the disk is controlled primarily by the **operating system** (**OS**), which schedules disk read/write.
- Many DBMSs have their own **buffer management** module to schedule disk read/write, because management of buffer storage has a considerable effect on performance.

- A higher-level **stored data manager** module of the DBMS controls access to DBMS information that is stored on disk, whether it is part of the database or the catalog.
- Casual users and persons with occasional need for information from the database interact using the **interactive query** interface.
- A **query compiler** that compiles them into an internal form.
- The **query optimizer** is concerned with the rearrangement and possible reordering of operations, elimination of redundancies, and use of efficient search algorithms during execution.
- The **precompiler** extracts DML commands from an application program written in a host programming language.

The **runtime database processor** executes

(1) The privileged commands,

(2) the executable query plans, and

(3) the canned transactions with runtime parameters.

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.

Concurrency control and **backup and recovery systems** separately as a module in this figure. They are integrated into the working of the runtime database processor for purposes of transaction management.

Q. With a neat diagram, explain the component modules of DBMS and their interactions. {8 marks}*

4. Database System Utilities

DBMSs have **database utilities** that help the DBA manage the database system.

Loading.

A loading utility is used to load existing data files—such as text files or sequential files—into the database.

Backup.

A backup utility creates a backup copy of the database, usually by dumping the entire database onto tape or other mass storage medium.

Database storage reorganization.

This utility can be used to reorganize a set of database files into different file organizations and create new access paths to improve performance.

Performance monitoring.

Such a utility 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.

Tools, Application Environments, and Communications Facilities

Other tools are often available to database designers, users, and the DBMS.

CASE tools are used in the design phase of database systems.

Data dictionary (or **data repository**) **system** -quite useful in large organizations

Application development environments, such as PowerBuilder (Sybase) or JBuilder (Borland), have been quite popular.

The DBMS also needs to interface with **communications software**, whose function is to allow users at locations remote from the database system site to access the database through computer terminals, workstations, or personal computers.

5. Centralized and Client/Server Architectures for DBMSs {8 marks}*

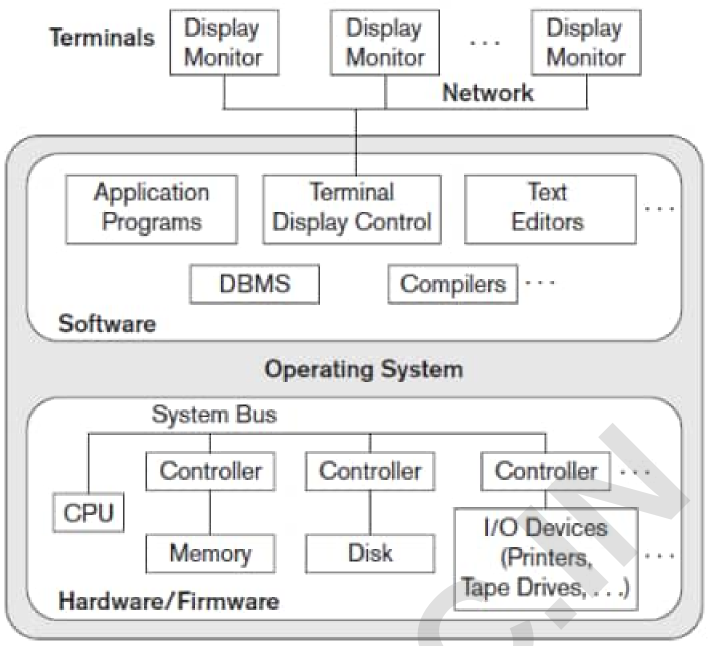
Centralized DBMSs Architecture

Architectures for DBMSs have followed trends similar to those for general computer system architectures.

Older architectures used mainframe computers to provide the main processing for all system functions, including user application programs and user interface programs, as well as all the DBMS functionality. The reason was that in older systems, most users accessed the DBMS via computer terminals that did not have processing power and only provided display capabilities. Therefore, all processing was performed remotely on the computer system housing the DBMS, and only display information and controls were sent from the computer to the display terminals, which were connected to the central computer via various types of communications networks.

At first, database systems used these computers similarly to how they had used display terminals, so that the

DBMS itself was still a **centralized** DBMS in which all the DBMS functionality, application program execution, and user interface processing were carried out on one machine. Figure 2.4 illustrates the physical components in a centralized architecture. Gradually, DBMS systems started to exploit the available processing power at the user side, which led to client/server DBMS architectures.



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. The idea is to define **specialized servers** with specific functionalities. For example, it is possible to connect a number of PCs or small workstations as clients to a **file server** that maintains the files of the client machines. Another machine can be designated as a **printer server** by being connected to various printers; all print requests by the clients are forwarded to this machine. **Web servers** or **e-mail servers** also fall into the specialized server category.
The resources provided by specialized servers can be accessed by many client machines.

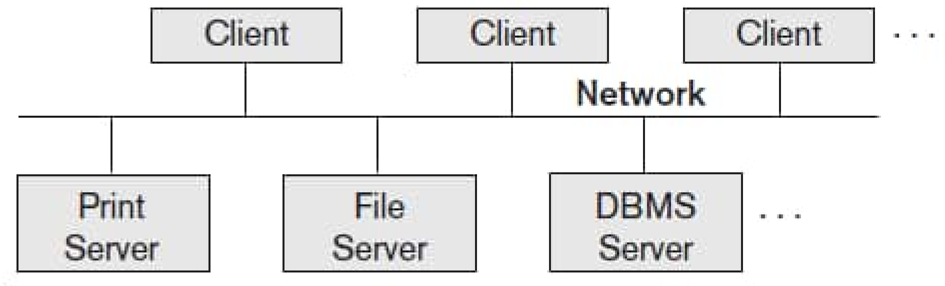


Figure: Logical two-tier client/server architecture.

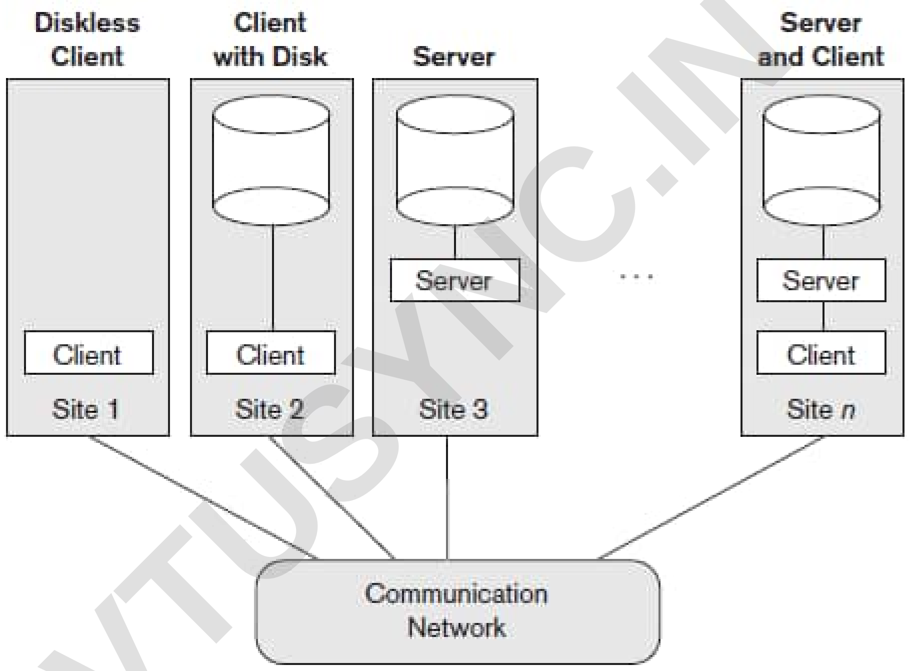


Figure: Physical two-tier client/server architecture.

A **client** in this framework is typically a user machine that provides user interface capabilities and local processing. When a client requires access to additional functionality—such as database access—that does not exist at the client, 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.

Two-Tier Client/Server Architectures for DBMSs

In relational database management systems (RDBMSs), many of which started as centralized systems, the system components that were first moved to the client side were the user interface and application programs. Because SQL provided a standard language for RDBMSs, this created a logical dividing point between client and server. Hence, the query and transaction functionality related to SQL processing remained on the server side. In such an architecture, the server is often called a **query server** or **transaction server** because it provides these two functionalities. In an RDBMS, the server is also often called an **SQL server**.

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 standard called **Open Database Connectivity** (**ODBC**) provides an **application programming interface** (**API**), which allows client-side programs to call the DBMS, as long as both client and server machines have the necessary software installed. Most DBMS vendors provide ODBC drivers for their systems. 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. This allows Java client programs to access one or more DBMSs through a standard interface.

The architectures described here are called **two-tier architectures** because the software components are distributed over two systems: client and server. The advantages of this architecture are its simplicity and seamless compatibility with existing systems. The emergence of the Web changed the roles of clients and servers, leading to the three-tier architecture.

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, as illustrated in Figure.

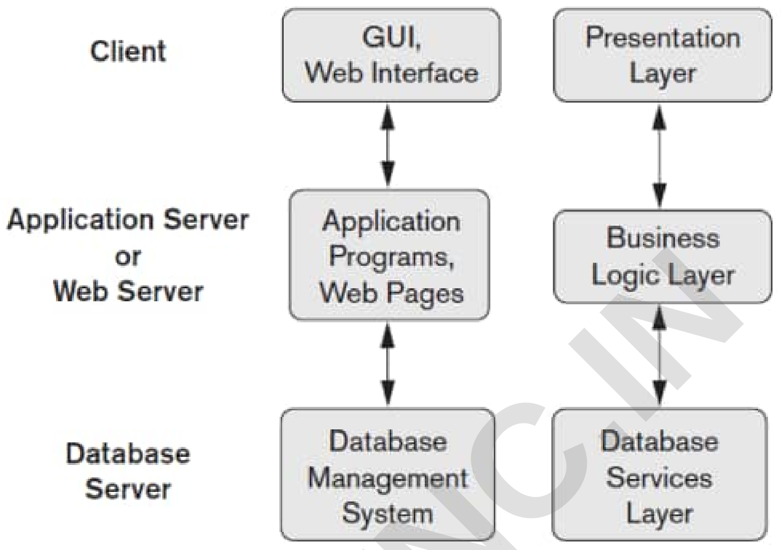


Figure: Logical three-tier client/server architecture, with a couple of commonly used nomenclatures.

This intermediate layer or **middle tier** is called the **application server** or the **Web server**, depending on the application. This server plays an intermediary role by running application programs and storing business rules (procedures or constraints) that are used to access data from the database server. It can also improve
database security by checking a client's credentials before forwarding a request to the database server. Clients contain user interfaces and Web browsers. The intermediate server accepts requests from the client, processes the request and sends database queries and commands to the database server, and then acts as a conduit for passing (partially) processed data from the database server to the clients, where it may be processed further and filtered to be presented to the users.

6. Classification of Database Management Systems (8 marks}*

Several criteria can be used to classify DBMSs.

1. **Based on data model**

The main data model used in many current commercial DBMSs is the **relational data model**, and the systems based on this model are known as **SQL systems.**

The **object data model** has been implemented in some commercial systems but has not had widespread use.

Recently, so-called **big data systems,** also known as **key-value storage systems** and **NOSQL systems,** use various data models: **document-based, graph-based, column-based,** and **key-value data models.** Many legacy applications still run on database systems based on the **hierarchical** and **network data models**.

2. Based on number of users supported by the system.

Single-user systems support only one user at a time and are mostly used with PCs.

Multiuser systems, which include the majority of DBMSs, support concurrent multiple users.

3**. Based on the number of sites over which the database is distributed.**

A DBMS is **centralized** if the data is stored at a single computer site. A centralized DBMS can support multiple users, but the DBMS and the database reside totally at a single computer site.

A **distributed** DBMS (DDBMS) can have the actual database and DBMS software distributed over many sites connected by a computer network.

4. Based on is cost.

It is difficult to propose a classification of DBMSs based on cost.

Today we have open source (free) DBMS products like MySQL and PostgreSQL that are supported by third-party vendors with additional services.

Q. List and explain the criteria of classifying the DBMS {8 marks}*

Chapter 3

Data Modeling Using the Entity– Relationship (ER) Model

Conceptual modeling is a very important phase in designing a successful database application.

The database application refers to a particular database and the associated programs that implement the database queries and updates.

These programs would provide user-friendly graphical user interfaces (GUIs) utilizing forms and menus for the end users of the application.

Entity–relationship (**ER**) **model**, which 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.

1. **Using High-Level Conceptual Data Models for Database Design {8 marks}***

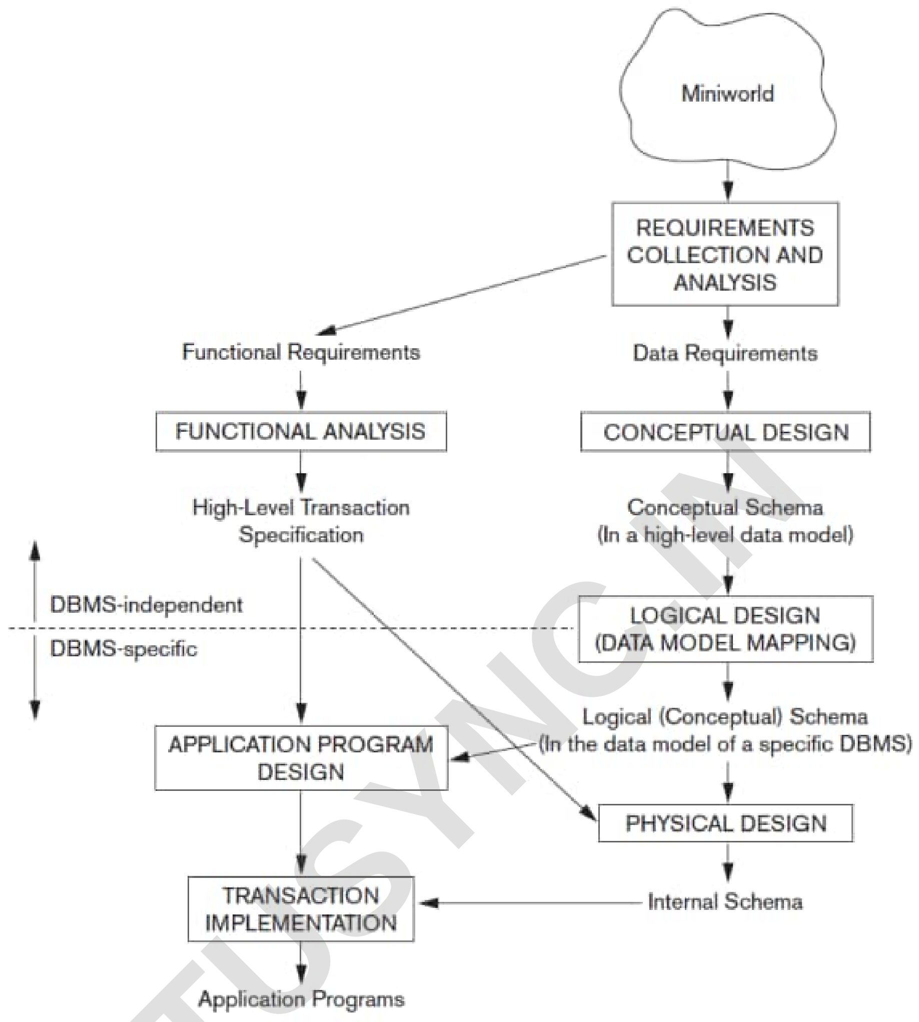

Miniworld
REQUIREMENTS COLLECTION AND ANALYSIS
Functional Requirements
Data Requirements
FUNCTIONAL ANALYSIS
CONCEPTUAL DESIGN
High-Level Transaction Specification
Conceptual Schema (In a high-level data model)
DBMS-independent
DBMS-specific
LOGICAL DESIGN (DATA MODEL MAPPING)
Logical (Conceptual) Schema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

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

The first step shown is **requirements collection and analysis**- the database designers interview prospective database users to understand and document their **data requirements**

Functional requirements - consist of the user defined **operations** (or **transactions**) that will be applied to the database, including both retrievals and updates.

Create a **conceptual schema** for the database, using a high-level conceptual data model.

The conceptual schema is transformed from the high-level data model into the implementation data model called **logical design** or **data model mapping.**

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.

2. A Sample Database Application

- 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.
- The database will store each employee's name, Social Security number, address, salary, sex (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. It is required to keep track of the current number of hours per week that an employee works on each project, as well as the direct supervisor of each employee (who is another employee).
- The database will keep track of the dependents of each employee for insurance purposes, including each dependent's first name, sex, birth date, and relationship to the employee.

3. Entity Types, Entity Sets, Attributes, and Keys {8 marks}*

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

An **entity**, which is a *thing* or *object* 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). Each entity has **attributes—**the particular properties that describe it.

For example, an EMPLOYEE entity may be described by the employee's name, age, address, salary, and job.

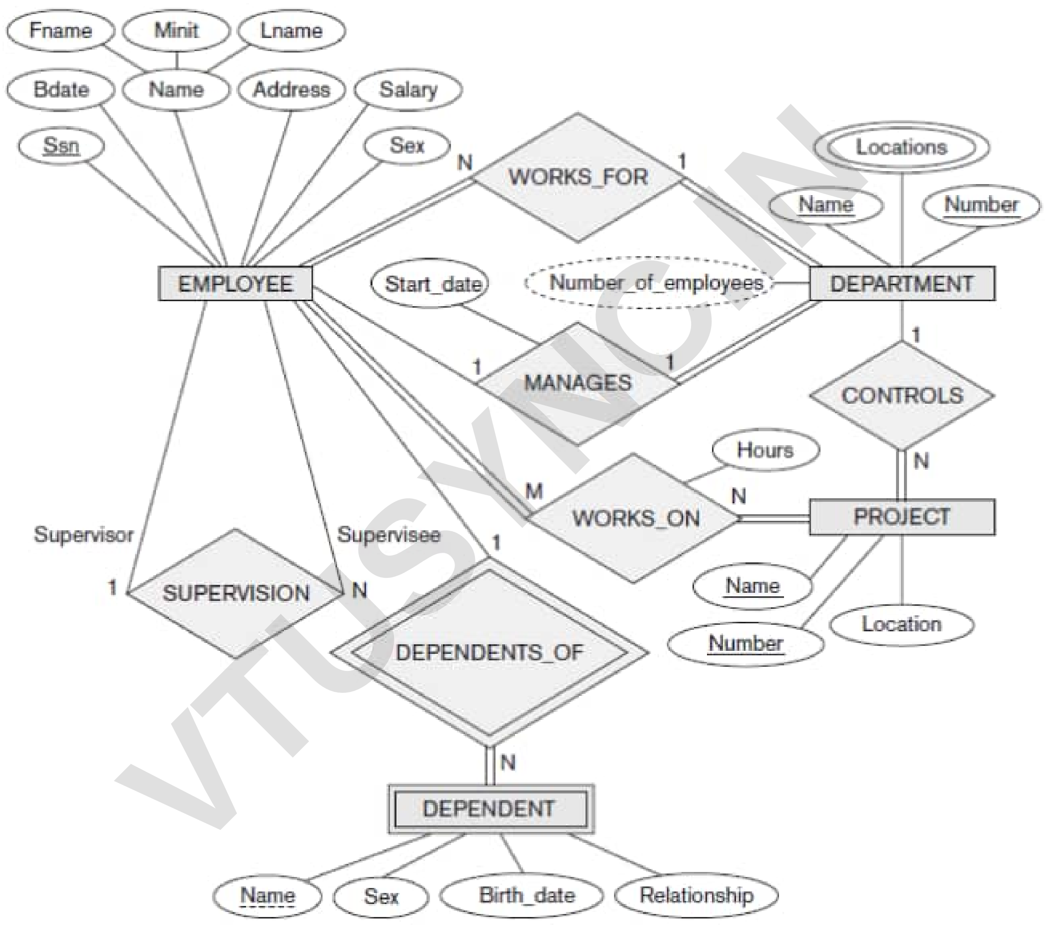


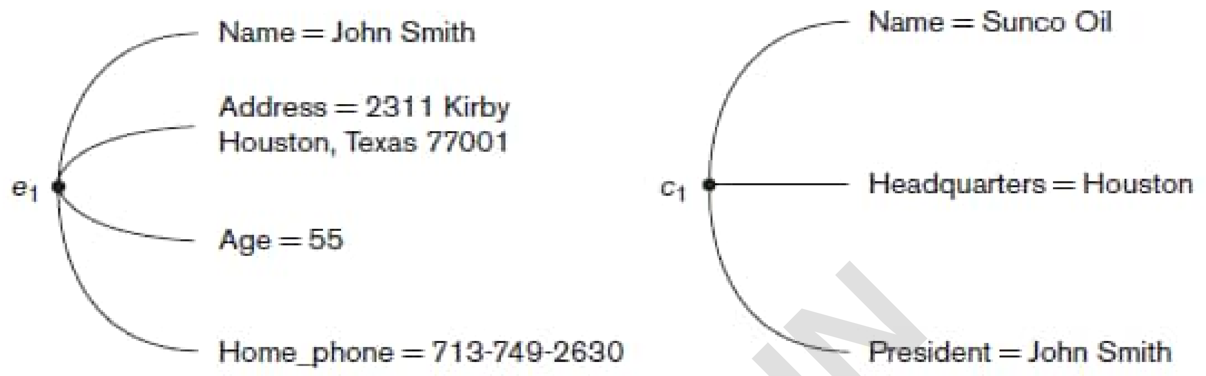
Figure: An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter.

Composite versus Simple (Atomic) Attributes.

Composite attributes can be divided into smaller subparts, which represent more basic attributes with independent meanings.

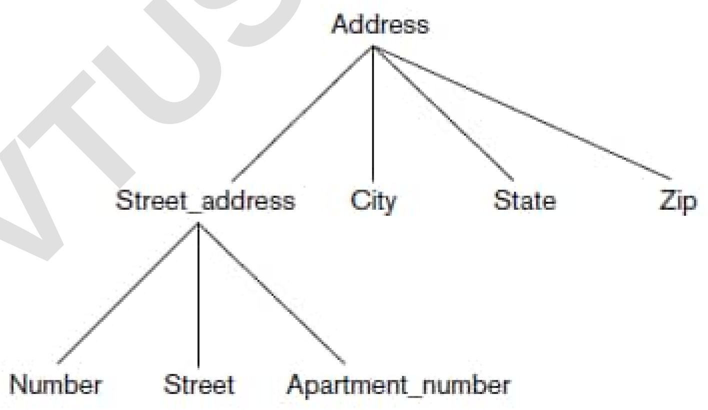
For example,

the Address attribute of the EMPLOYEE entity can be subdivided into Street_address, City, State, and Zip, with the values ‘2311 Kirby’, ‘Houston’, ‘Texas’, and ‘77001’.



The COMPANY entity *c*1 has three attributes: Name, Headquarters, and President; their values are ‘Sunco Oil’, ‘Houston’, and ‘John Smith’

The value of a composite attribute is the concatenation of the values of its component simple attributes.



Single-Valued versus Multivalued Attributes

Most attributes have a single value for a particular entity; such attributes are called **single-valued**. For example, Age is a single-valued attribute of 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 two values

Stored versus Derived Attributes.

Two (or more) attribute values are related—for example, the Age and Birth_date attributes of a person.

The Age attribute is hence called a **derived attribute** and is said to be **derivable from** the Birth_date attribute, which is called a **stored attribute**.

NULL Values.

In some cases, a particular entity may not have an applicable value for an attribute.

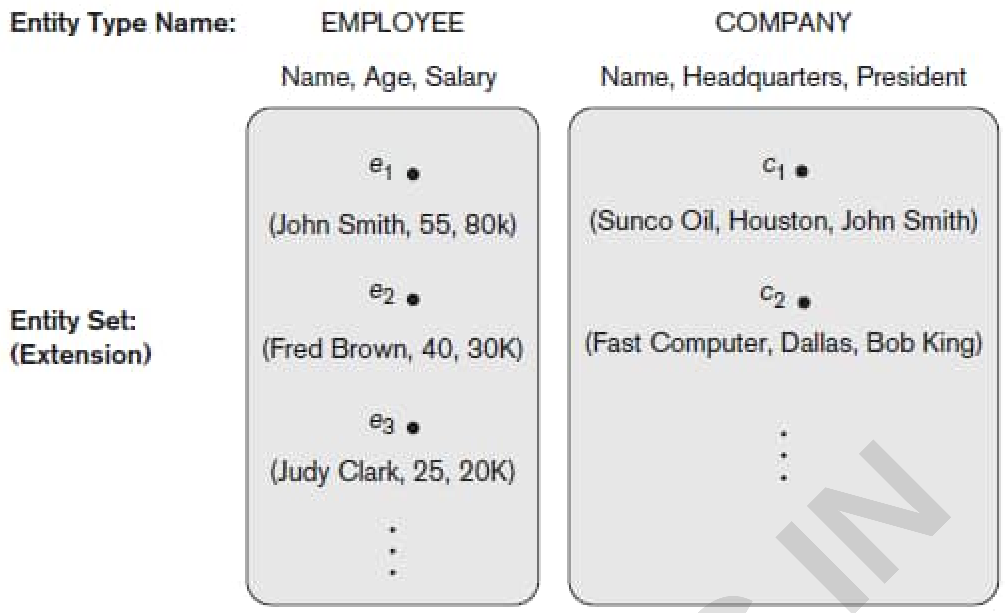
Complex Attributes

composite and multivalued attributes can be nested arbitrarily. We can represent arbitrary nesting by grouping components of a composite attribute between parentheses ( ) and separating the components with commas, and by displaying multivalued attributes between braces { }. Such attributes are called **complex attributes.**

{Address_phone({Phone(Area_code,Phone_number)},Address(Street_address (Number,Street,Apartment_number),City,State,Zip))}

A person can have more than one residence and each residence can have a single address and multiple phones.

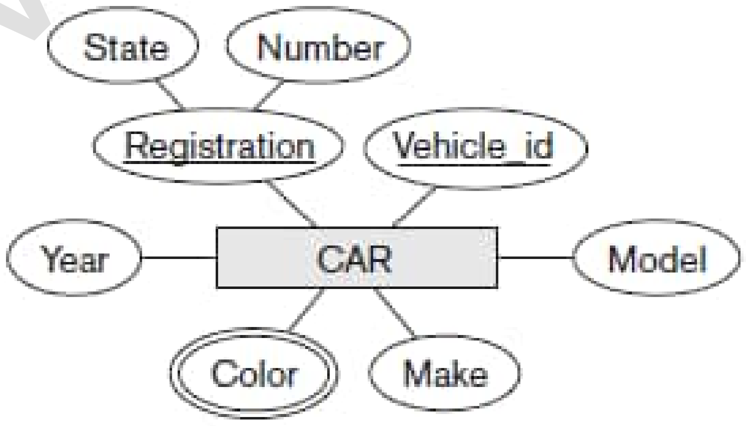
Entity Types and Entity Sets. A database usually contains groups of entities that are similar. For example, a company employing hundreds of employees may want to store similar information concerning each of the employees.

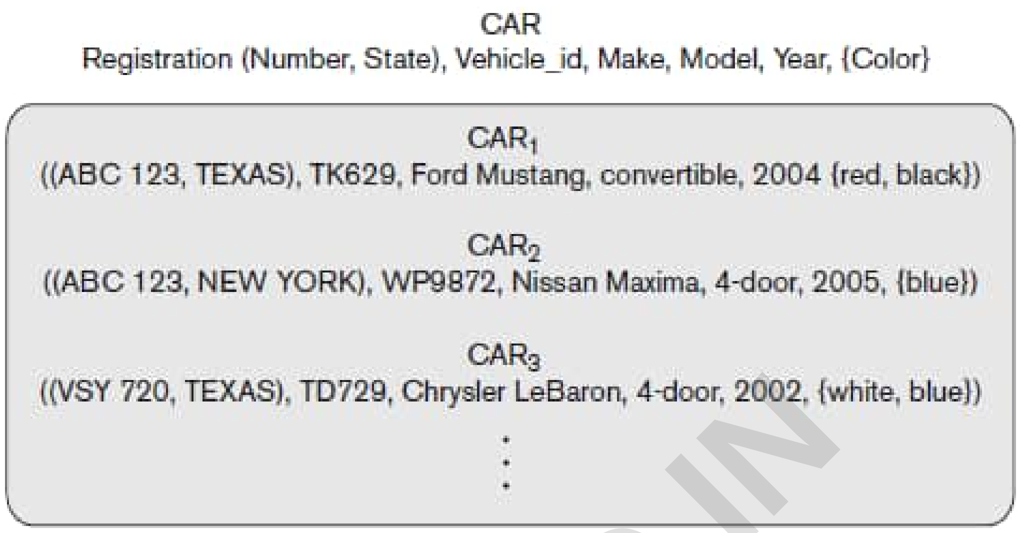


Key Attributes of an Entity Type.

An important constraint on the entities of an entity type is the **key** or **uniqueness constraint** on attributes.

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.




CAR
Registration (Number, State), Vehicle_id, Make, Model, Year, {Color}
CAR1
((ABC 123, TEXAS), TK629, Ford Mustang, convertible, 2004 {red, black})
CAR2
((ABC 123, NEW YORK), WP9872, Nissan Maxima, 4-door, 2005, {blue})
CAR3
((VSY 720, TEXAS), TD729, Chrysler LeBaron, 4-door, 2002, {white, blue})

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.

Mathematically, an attribute A of entity set E whose value set is V can be defined as a **function** from E to the power set6 $P(V)$ of V:

$$A : E \to P(V)$$

The value of attribute A for entity e as $A(e)$.

A NULL value is represented by the *empty set*.

For single-valued attributes, $A(e)$ is restricted to being a *singleton set* for each entity e in E.

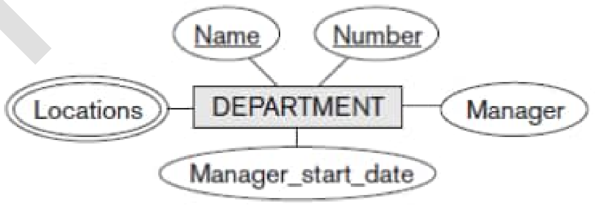
There is no restriction on multivalued attributes.

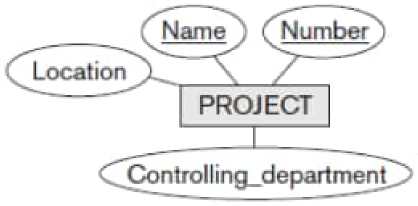
For a composite attribute A, the value set V is the power set of the Cartesian product of $P(V1)$, $P(V2), \ldots, P(Vn)$, where $V1, V2, \ldots, Vn$ are the value sets of the simple component attributes that form A:

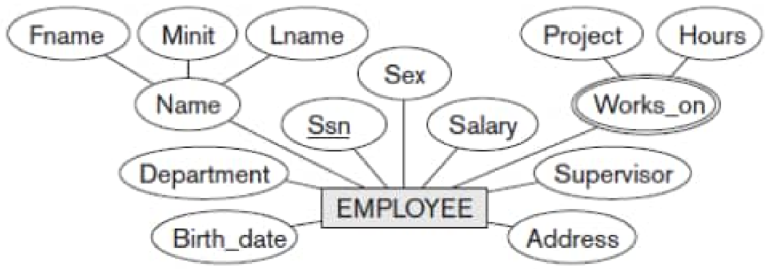
$$V = P(P(V1) \times P(V2) \times \ldots \times P(Vn))$$

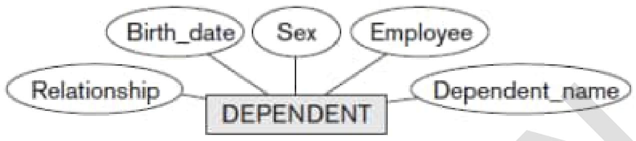
Initial Conceptual Design of the COMPANY Database

1. An entity type DEPARTMENT with attributes Name, Number, Locations, Manager, and Manager_start_date. Locations is the only multivalued attribute. We can specify that both Name and Number are (separate) key attributes because each was specified to be unique.
2. An entity type PROJECT with attributes Name, Number, Location, and Controlling_department. Both Name and Number are (separate) key attributes.
3. An entity type EMPLOYEE with attributes Name, Ssn, Sex, Address, Salary, Birth_date, Department, and Supervisor. Both Name and Address may be composite attributes; however, this was not specified in the requirements. We must go back to the users to see if any of them will refer to the individual components of Name—First_name, Middle_initial, Last_name—or of Address. Inour example, Name is modeled as a composite attribute, whereas Address is not, presumably after consultation with the users.
4. An entity type DEPENDENT with attributes Employee, Dependent_name, Sex, Birth_date, and Relationship (to the employee).









4. Relationship Types, Relationship Sets, Roles, and Structural Constraints {8 to 10 marks}*

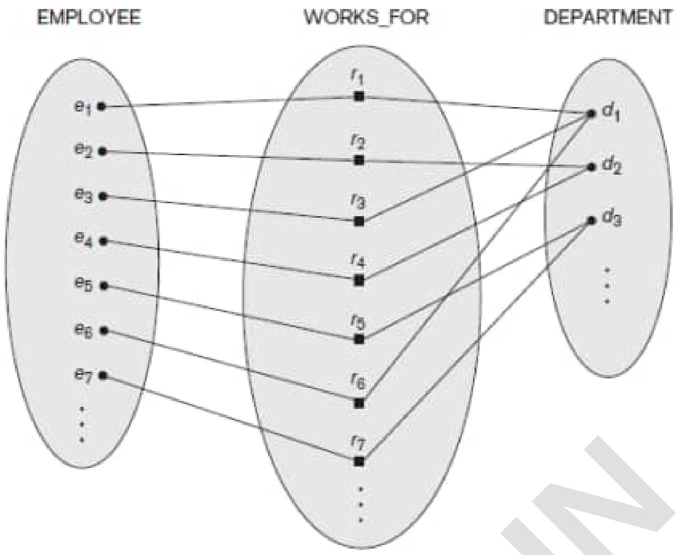
Relationship Types, Sets, and Instances

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.

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_j in r_i is a member of entity set E_j, 1 <= *j* <= *n*.

A relationship set is a mathematical relation on *E*1, *E*2, . . . , *En*;

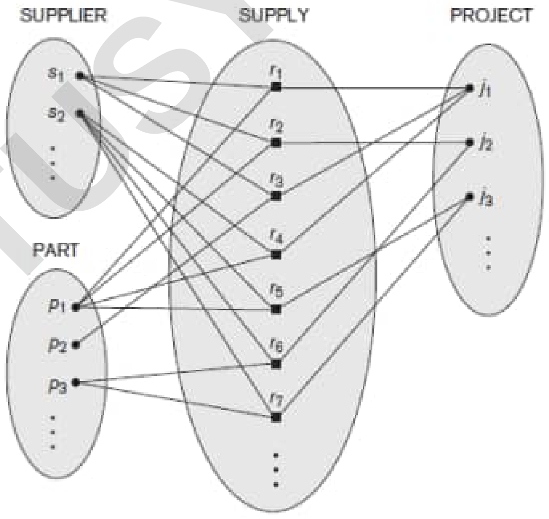
alternatively, it can be defined as a subset of the Cartesian product of the entity sets *E*1 × *E*2 × . . . × *En*. Each of the entity types *E*1, *E*2, . . . , *En* is said to **participate** in the relationship type *R*; similarly, each of the individual entities *e*1, *e*2, . . . , *en* is said to **participate** in the relationship instance *ri* = (*e*1, *e*2, . . . , *en*).



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**.



Relationships as Attributes.

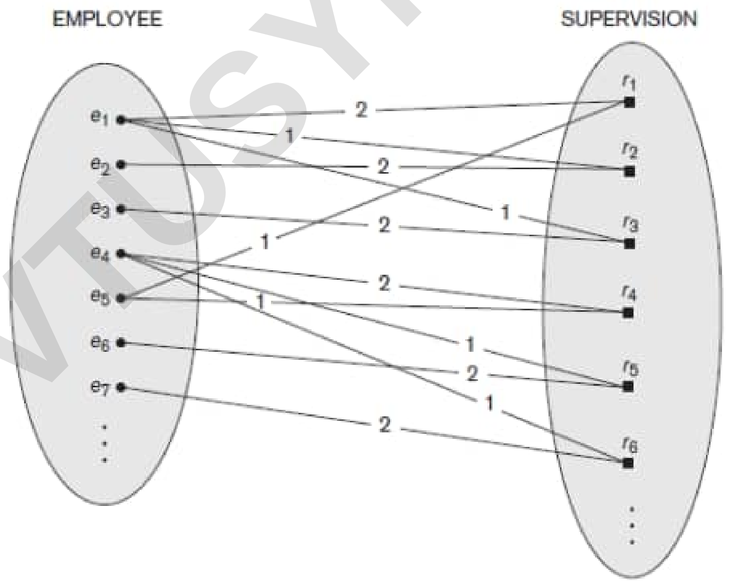
It is sometimes convenient to think of a binary relationship type in terms of attributes.

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. Hence, the value set for this Department attribute is the set of *all* DEPARTMENT entities, which is the DEPARTMENT entity set.

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 it 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*.
The *same* entity type participates more than once in a relationship type in *different roles*.
Such relationship types are called **recursive relationships** or **self-referencing relationships**.



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.

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.

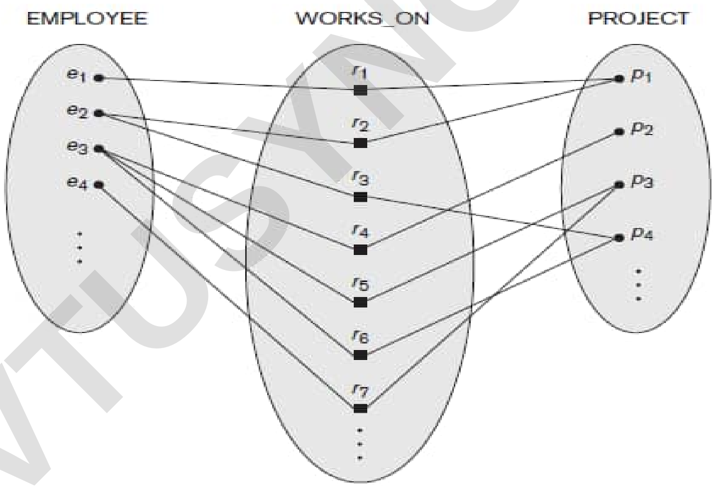
Two main types of binary relationship constraints:

***cardinality ratio* and p*articipation*.**

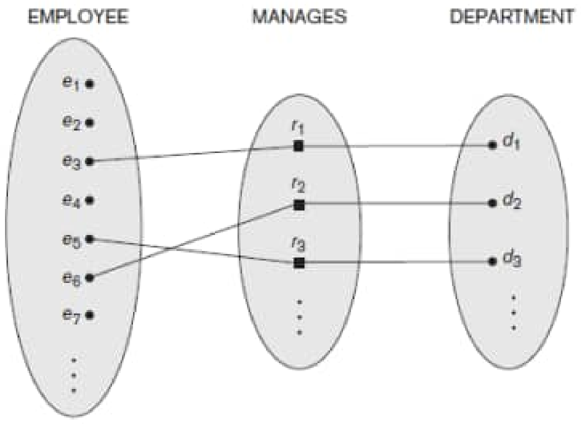
Cardinality Ratios for Binary Relationships.

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 (that is, employs) any number of employees (N)



An M:N relationship, WORKS_ON.



A 1:1 relationship, MANAGES.

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 participation and

Partial participation

The participation of EMPLOYEE in WORKS_FOR is called **total participation**

The participation of EMPLOYEE in the MANAGES relationship type is **partial participation**

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 a particular employee works on a particular project, we can include an attribute Hours for the WORKS_ON relationship type.

Another example to include the date on which a manager started managing a department via an attribute Start_date for the MANAGES relationship type.

Notice that attributes of 1:1 or 1:N relationship types can be migrated to one of the participating entity types.

For M:N (many-to-many) 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*

5. Weak Entity Types {6 marks}*

Entity types that do not have key attributes of their own are called **weak entity types**. **Regular entity types** that do have a key attribute are called **strong 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. 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.
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.*

6. ER Diagrams, Naming Conventions, and Design Issues

The participation of entity types in relationship types by displaying their entity sets and relationship sets (or extensions)—the individual entity instances in an entity set and the individual relationship instances in a relationship set. In ER diagrams the emphasis is on representing the schemas rather than the instances. This is more useful in database design because a database schema changes rarely, whereas the contents of the entity sets may change frequently. In addition, the schema is obviously easier to display, because it is much smaller.

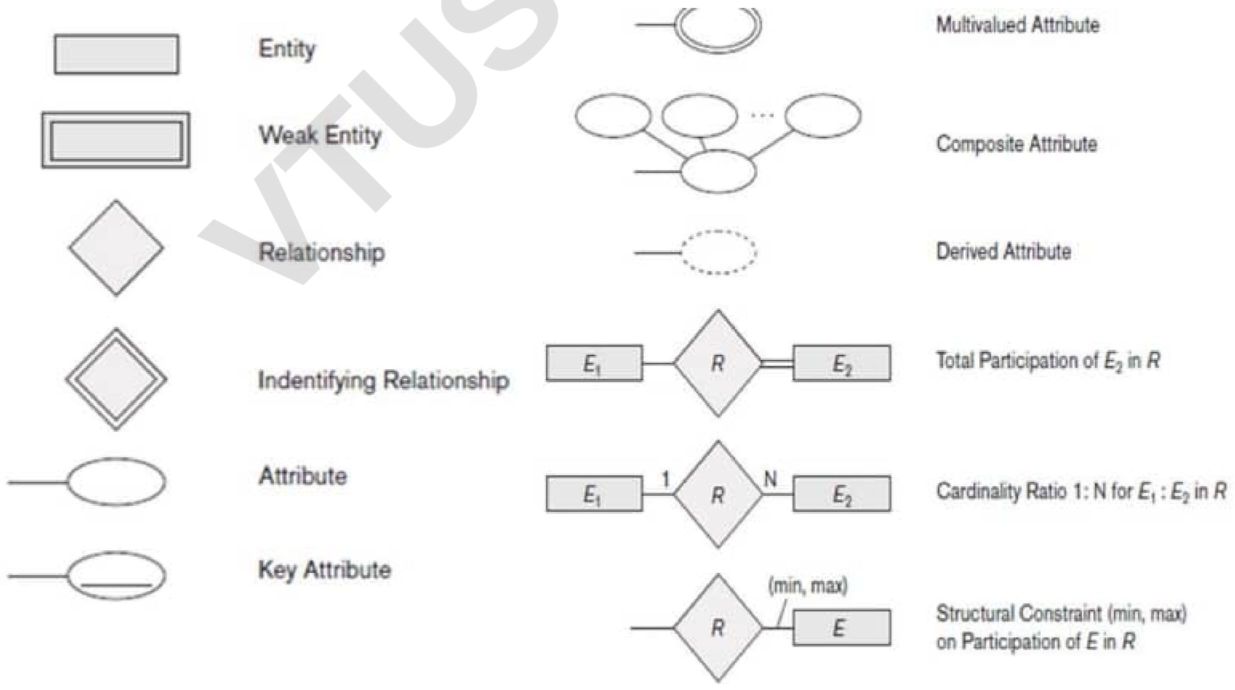

Entity
Weak Entity
Relationship
Indentifying Relationship
Attribute
Key Attribute
Multivalued Attribute
Composite Attribute
Derived Attribute
E_1
R
E_2
Total Participation of E_2 in R
1
N
Cardinality Ratio 1: N for $E_1 : E_2$ in R
(min, max)
E
Structural Constraint (min, max) on Participation of E in R

Figure: ER Diagram symbols and their notations.

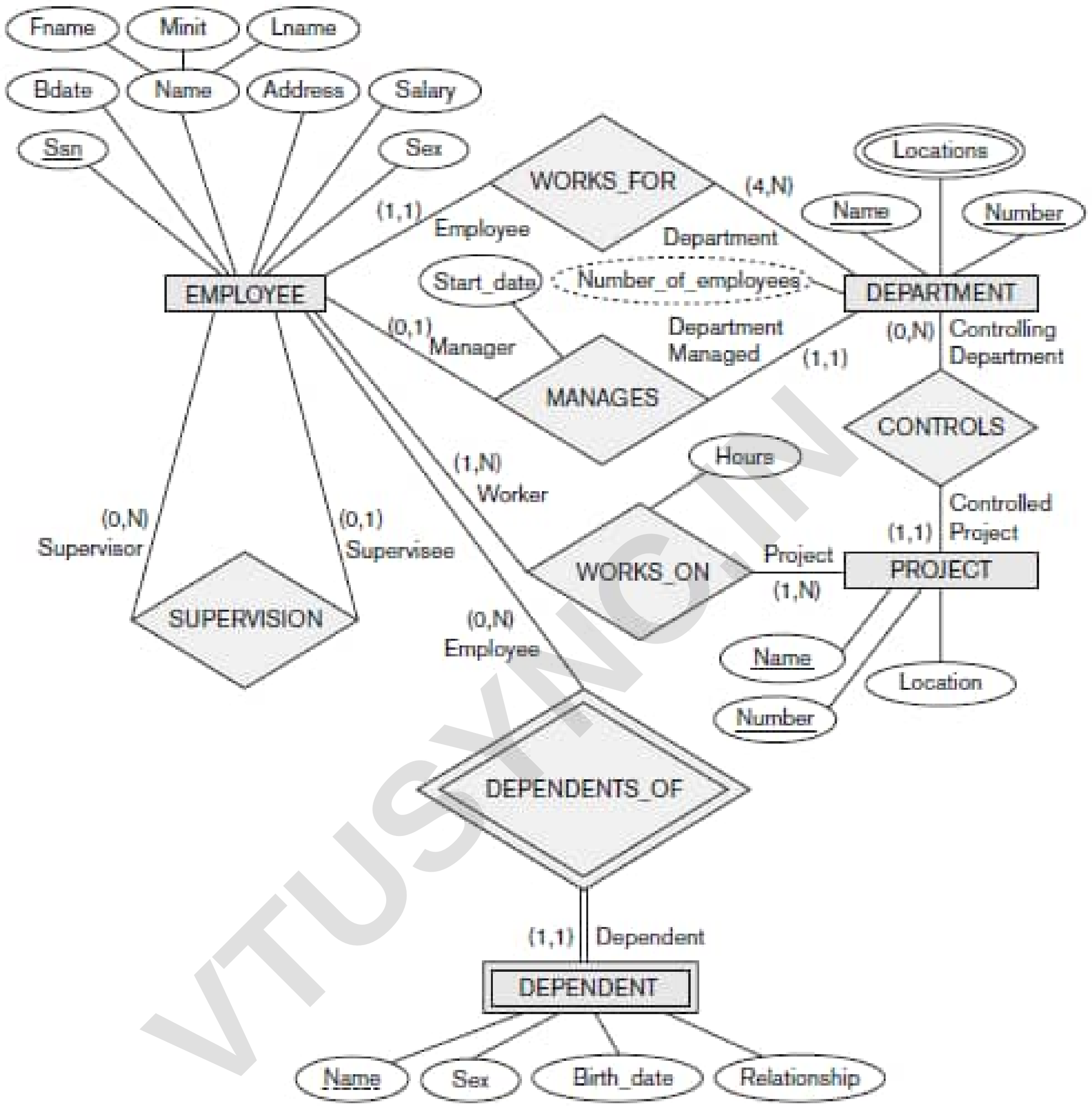


Figure: ER diagrams for the company schema, with structural constraints specified using (min, max) notation and role names.

Proper Naming of Schema Constructs

When designing a database schema, the choice of names for entity types, attributes, relationship types, and (particularly) roles is not always straightforward. One should choose names that convey, as much as possible, the meanings attached to the different constructs in the schema. We choose to 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 our ER diagrams, we will use the convention that entity type and relationship type names are in uppercase letters, attribute names have their initial letter capitalized, and role names are in 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.

Example of Other Notation:

UML Class Diagrams

The UML methodology is being used extensively in software design and has many types of diagrams for various software design purposes. We only briefly present the basics of UML class diagrams here and compare them with ER diagrams.

In UML class diagrams, a class (similar to an entity type in ER) is displayed as a box (see Figure below) that includes three sections: The top section gives the class name (similar to entity type name); the middle section includes the attributes; and the last section includes operations that can be applied to individual objects (similar to individual entities in an entity set) of the class. Operations are not specified in ER diagrams. Consider the EMPLOYEE class in Figure below. Its attributes are Name, Ssn, Bdate, Sex, Address, and Salary.

A multivalued attribute will generally be modeled as a separate class.

Relationship types are called associations in UML terminology, and relationship instances are called links. A binary association (binary relationship type) is represented as a line connecting the participating classes (entity types), and may optionally have a name.

A relationship attribute, called a link attribute, is placed in a box that is connected to the association's line by a dashed line.

In UML, there are two types of relationships: association and aggregation. Aggregation is meant to represent a relationship between a whole object and its component parts, and it has a distinct diagrammatic notation.

UML also distinguishes between unidirectional and bidirectional associations (or aggregations). In the unidirectional case, the line connecting the classes is displayed with an arrow to indicate that only one direction for accessing related objects are needed.

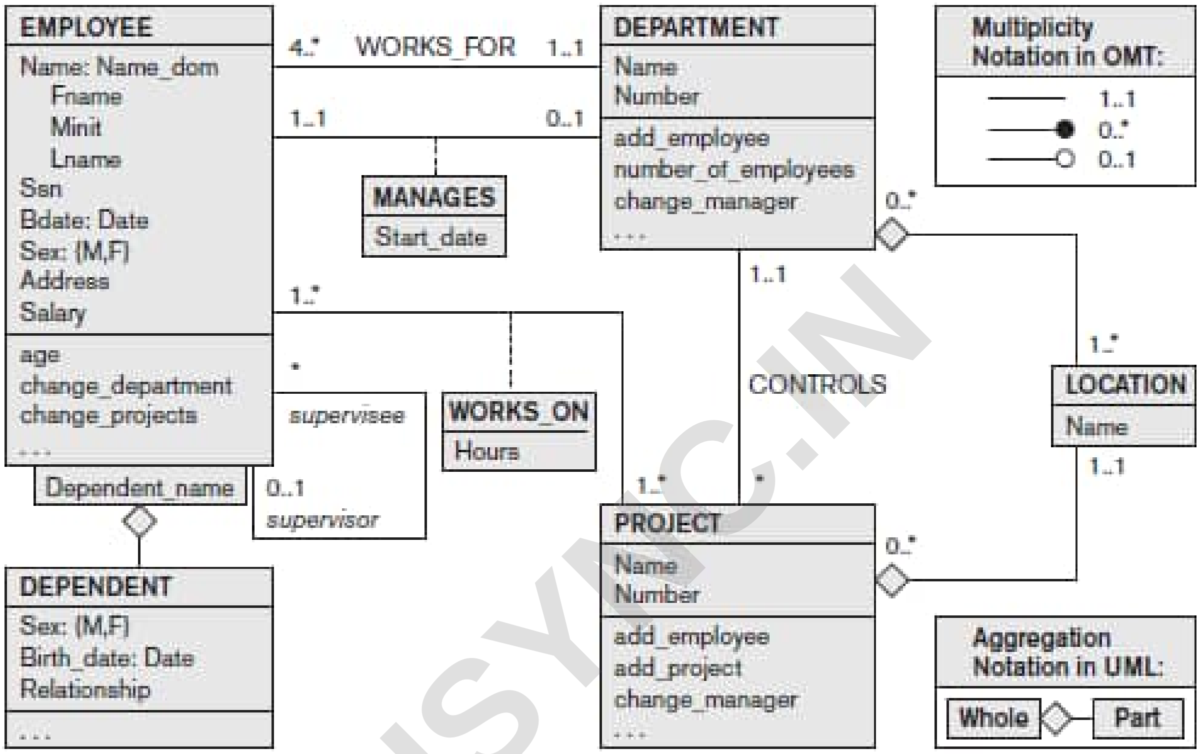


Figure: The COMPANY conceptual schema in UML class diagram notation

7. Relationship Types of Degree Higher than Two

The **degree** of a relationship type as the number of participating entity types and called a relationship type of degree two *binary* and a relationship type of degree three *ternary*.

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

The ER diagram notation for a ternary relationship type is shown in Figure below, which displays the schema for the SUPPLY relationship type that was displayed at the instance level. Recall that the relationship set of SUPPLY is a set of relationship instances (s, j, p), where the meaning is that s is a SUPPLIER who is currently supplying a PART p to a PROJECT j. In general, a relationship type R of degree n will have n edges in an ER diagram, one connecting R to each participating entity type.

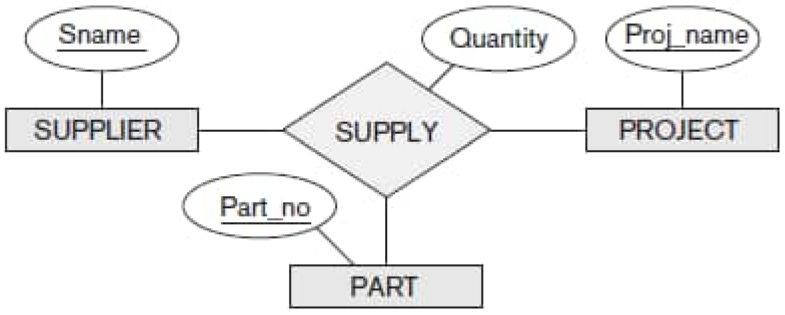


Figure: The SUPPLY relationship.

Figure below shows an ER diagram for three binary relationship types CAN_SUPPLY, USES, and SUPPLIES. In general, a ternary relationship type represents different information than do three binary relationship types. Consider the three binary relationship types CAN_SUPPLY, USES, and SUPPLIES. Suppose that 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; and SUPPLIES, between SUPPLIER and PROJECT, includes an instance (s, j) whenever supplier s supplies some part to project j.

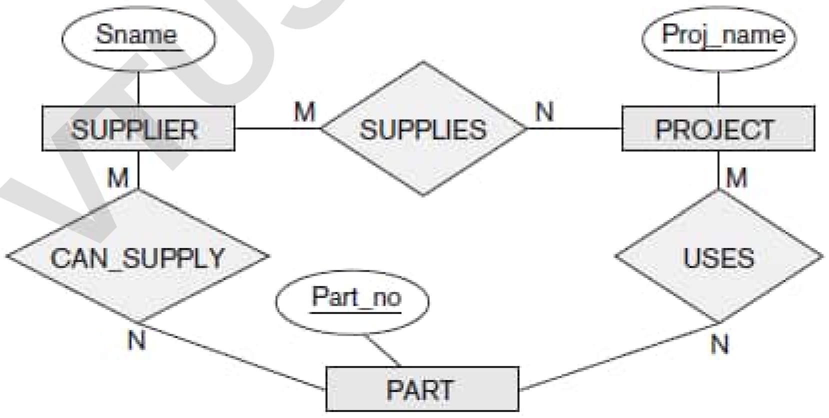


Figure: Three binary relationships not equivalent to SUPPLY.

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

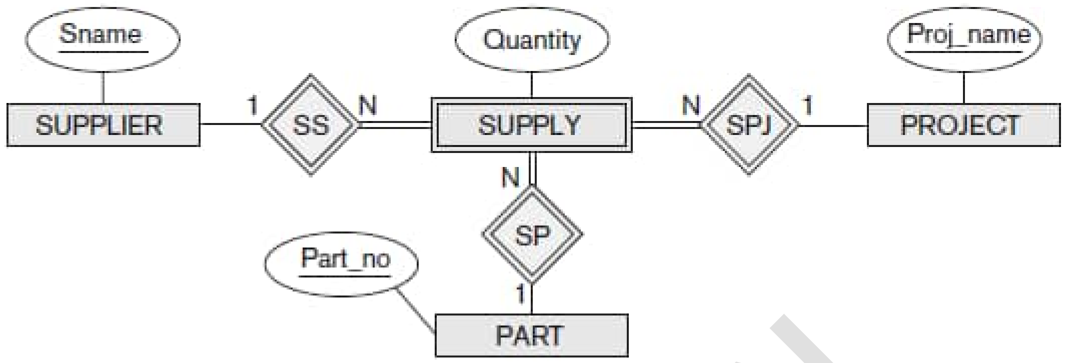


Figure: SUPPLY represented as a weak entity type.

Another example is shown in Figure below. The ternary relationship type OFFERS represents information on instructors offering courses during particular semesters; hence it includes a relationship instance (i, s, c) whenever INSTRUCTOR i offers COURSE c during SEMESTER s. The three binary relationship types shown in Figure below have the following meanings: CAN_TEACH relates a course to the instructors who can teach that course, TAUGHT_DURING relates a semester to the instructors who taught some course during that semester, and OFFERED_DURING relates a semester to the courses offered during that semester by any instructor. These ternary and binary relationships represent different information, but certain constraints should hold among the relationships.

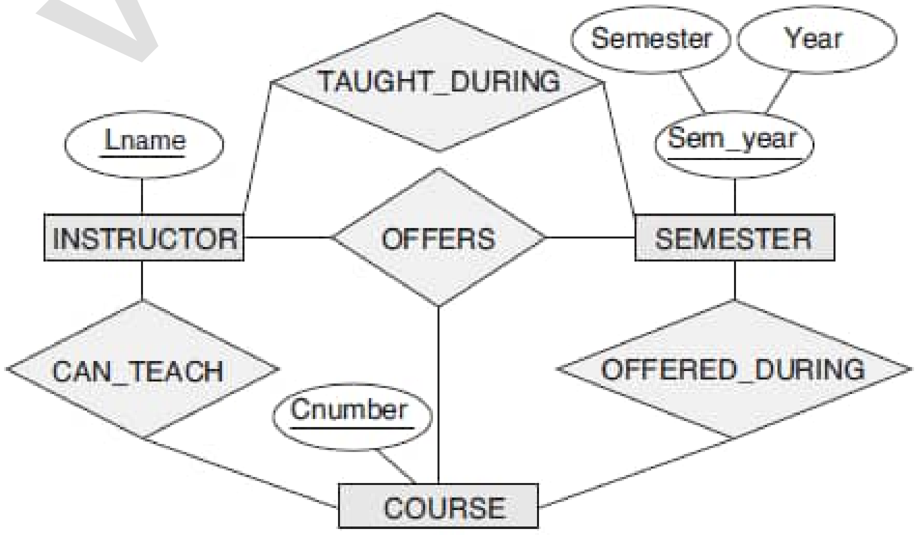


Figure: Another example of ternary versus binary relationship types.

Although in general three binary relationships cannot replace a ternary relationship, they may do so under certain additional constraints. In our example, if the CAN_TEACH relationship is 1:1 (an instructor can teach only one course, and a course can be taught by only one instructor), then the ternary relationship OFFERS can be left out because it can be inferred from the three binary relationships CAN_TEACH, TAUGHT_DURING, and OFFERED_DURING. The schema designer must analyze the meaning of each specific situation to decide which of the binary and ternary relationship types are needed.

Notice that it is possible to have a weak entity type with a ternary (or n-ary) identifying relationship type. In this case, the weak entity type can have several owner entity types. An example is shown in Figure below. This example shows part of a database that keeps track of candidates interviewing for jobs at various companies, which may be part of an employment agency database. In the requirements, a candidate can have multiple interviews with the same company (for example, with different company departments or on separate dates), but a job offer is made based on one of the interviews. Here, INTERVIEW is represented as a weak entity with two owners CANDIDATE and COMPANY, and with the partial key Dept_date. An INTERVIEW entity is uniquely identified by a candidate, a company, and the combination of the date and department of the interview.

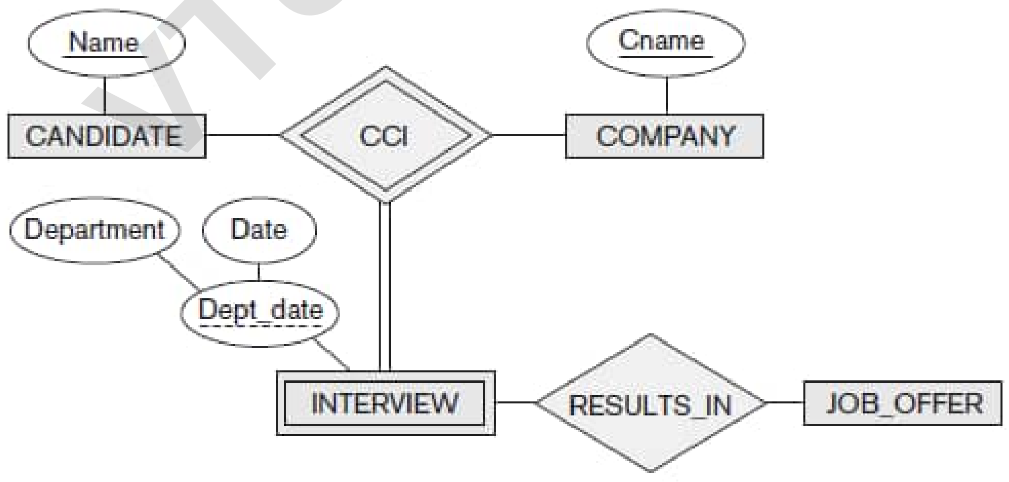


Figure: A weak entity type INTERVIEW with a ternary identifying relationship type.