Database Management Systems(BCS403)

MODULE 1

Module – 1 Introduction to Databases

Contents:

Introduction

- Characteristics of database approach
- Advantages of using the DBMS approach
- History of database applications

Overview of Database Languages and Architectures:

- Data Models.
- Schemas.
- Instances.
- Three schema architecture.
- Data independence.
- Database languages and interfaces.
- The Database System environment.

Module – 1 Introduction to Databases

Contents:

Conceptual Data Modelling using Entities and Relationships:

- Entity types.
- Entity sets.
- Attributes.
- Roles and Structural Constraints,
- Weak entity types,
- ER diagrams and Examples
- Entity Specialization and Generalization

Module – 1 Introduction to Databases

Introduction

A **database** is a collection of related data. The common use of the term database is usually more restricted.

A database has the following implicit properties:

1. A database represents some aspect of the real world, sometimes called the **mini-world or the universe of discourse** (UoD). Changes to the mini-world are reflected in the database.
2. 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.
3. 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.

Module – 1 Introduction to Databases

Introduction

A database can be of **any size and complexity**.

For example,

Case 1: The list of **names and addresses** referred to earlier may consist of only a few hundred records, each with a simple structure.

Case 2: 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.

Case 3: A database of even greater size and complexity would be maintained by a **social media company such as Facebook**, which has more than a billion users. The database has to maintain information on which users are related to one another as friends, the postings of each user, which users are allowed to see each posting.

Module – 1 Introduction to Databases

Introduction

Case 4: An example of a **large commercial database** is Amazon.com. It contains data for over 60 million active users, and millions of books, CDs, videos, DVDs, games, electronics, apparel, and other items.

Note:

- A **database** may be **generated and maintained manually** or it may be **computerized**.
- **For example**, a library card catalog is a database that may be created and maintained manually.
- A **computerized database** may be created and maintained either by a group of application programs written specifically for that task or by a database management system.

Module – 1 Introduction to Databases

Introduction

- **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 **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.
- **Meta-data**: The database definition or descriptive information is also stored by the DBMS in the form of a database catalog or dictionary; it is called **meta-data**.
- **Manipulating a database** includes functions such as querying the database to retrieve specific data, updating the database to reflect changes in the mini-world, and generating reports from the data.

Module – 1 Introduction to Databases

Introduction

- **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.
- **Database system:** The database and DBMS software together called as database system.

Module – 1 Introduction to Databases

Introduction

Important note:

- There is no absolute reason or necessary to use general-purpose DBMS software to implement a computerized database.
- It is possible to **write a customized set of programs** to create and maintain the database, in effect creating a **special-purpose DBMS software for a specific application**, such as airlines reservations.

Module – 1 Introduction to Databases

Introduction

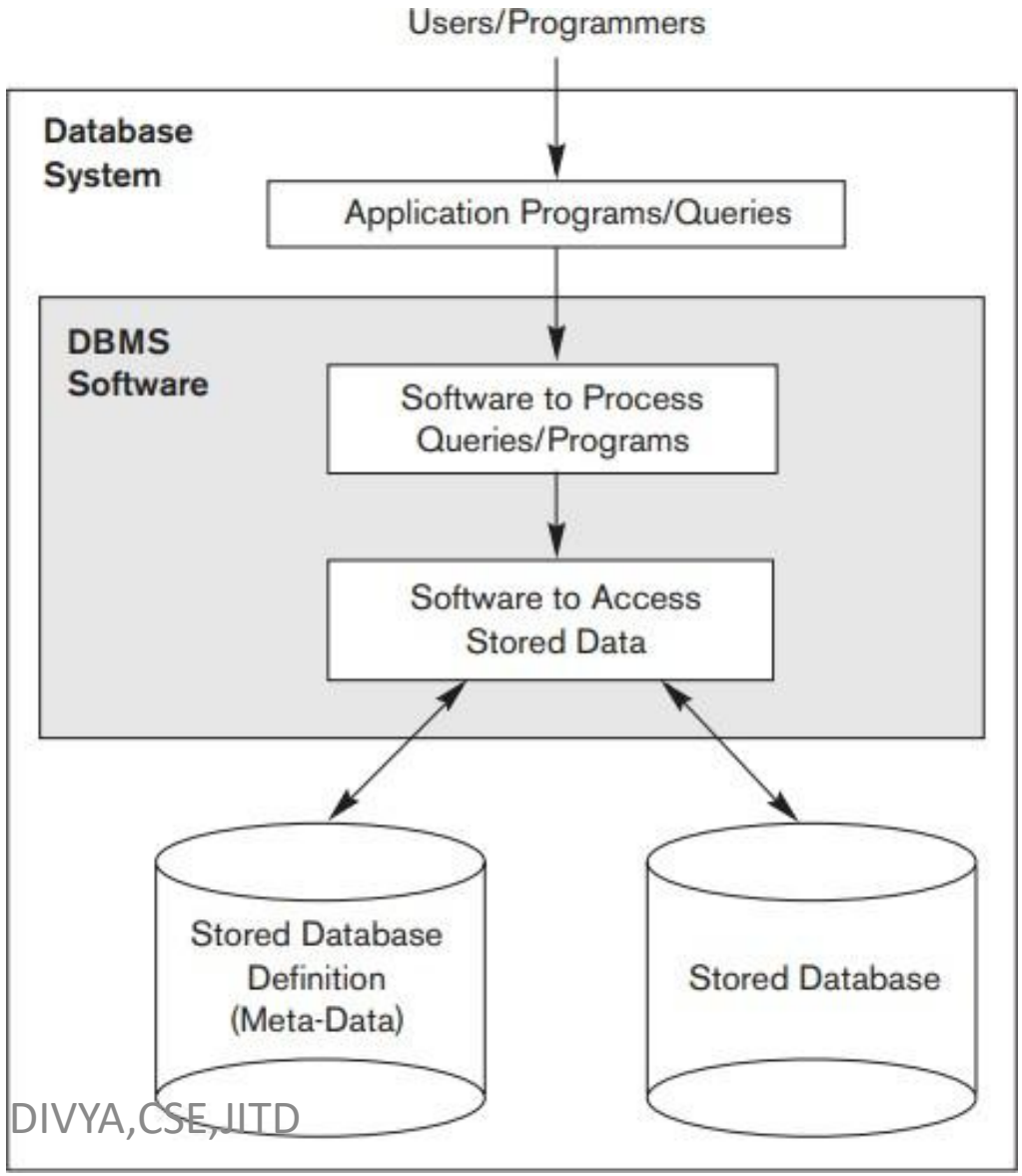


Figure: **A simplified database system environment.**

Module – 1 Introduction to Databases

Introduction

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

Module – 1 Introduction to Databases

Introduction

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

Module – 1 Introduction to Databases

Introduction

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

Module – 1 Introduction to Databases

Introduction

- The **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 (**Entity-Relationship model**)** 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 mini-world.

Module – 1 Introduction to Databases

Characteristics of the Database Approach

- A number of characteristics 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.
- **For example,** consider a college information system, one user, the **grade reporting office**, may keep files on students and their grades. Programs to **print a student's transcript** and to **enter new grades** are implemented as part of the application.
- A second user, the **accounting office**, may keep track of students' fees and their payments.
- This approach causes redundancy in defining and storing data results in wasted storage space and in redundant efforts to maintain common up-to-date data.

Module – 1 Introduction to Databases

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.

Self-describing nature of a database system

- 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(meta-data), which contains information such as the structure of each file, the type and storage format of each data item, and various constraints on the data.

Module – 1 Introduction to Databases

Characteristics of the Database Approach

- The DBMS software **must work equally** well with any number of database applications.
- **For example,** a university database, a banking database, or a company database—as long as the database definition is stored in the catalog.
- In **traditional file processing**, data definition is typically part of the application programs themselves. Hence, these programs are constrained to work with only one specific database, whose structure is declared in the application programs.

Module – 1 Introduction to Databases

Characteristics of the Database Approach

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.

Module – 1 Introduction to Databases

Characteristics of the Database Approach

Insulation between programs and data, and data abstraction

- In traditional file processing, the structure of data files is embedded in the application programs, so any changes to the structure of a file may require changing all programs that access that file.
- DBMS access programs do not require such changes in most cases. The structure of data files is stored in the DBMS catalog separately from the access programs. **This property is called as program-data independence.**
- **For example**, a file access program may be written in such a way that it can access only STUDENT records. If we want to add another piece of data to each STUDENT record, say the Birth_date**, such a program will no longer work and must be changed.**

Module – 1 Introduction to Databases

Characteristics of the Database Approach

Insulation between programs and data, and data abstraction

- In a DBMS environment, we only need to change the description of STUDENT records in the catalog to reflect the inclusion of the new data item Birth_date; no programs are changed.

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.
- **For example**, one user of the university database may be interested only in accessing and printing the transcript of each student as shown in figure below.

Module – 1 Introduction to Databases

Characteristics of the Database Approach

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

- A second user, who is interested only in checking that students have taken all the prerequisites of each course for which the student registers, as shown in Figure below.

Module – 1 Introduction to Databases

Characteristics of the Database Approach

COURSE_PREREQUISITES

Course_name	Course_number	Prerequisites
Database	CS3380	CS3320
		MATH2410
Data Structures	CS3320	CS1310

Sharing of data and multiuser transaction processing

- A multiuser DBMS allows multiple users to access the database at the same time.
- The DBMS must include concurrency control software to ensure that several users trying to update the same data in a controlled manner so that the result of the updates is correct.

Module – 1 Introduction to Databases

Characteristics of the Database Approach

- **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.**

Module – 1 Introduction to Databases

Actors on the Scene

Database Administrators

- In a database environment, the primary resource is the database itself, and the secondary resource is the DBMS and related software.
- 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.

Module – 1 Introduction to Databases

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.

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.

Module – 1 Introduction to Databases

There are several categories of end users:

Casual end users occasionally access the database, but they may need different information each time. They use a sophisticated database query interface to specify their requests and are typically middle- or high-level managers or other occasional browsers.

Naive or parametric end users make up a sizable portion of database end users. Their main job function revolves around constantly querying and updating the database, using standard types of queries and updates called **canned transactions** that have been carefully programmed and tested.

A few examples are:

- Bank customers and tellers check account balances and post withdrawals and deposits.
- Reservation agents or customers for airlines, hotels, and car rental companies check availability for a given request and make reservations.

Module – 1 Introduction to Databases

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. **An example** is the user of a financial software package that stores a variety of personal financial data.

System Analysts and Application Programmers

- 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. Such analysts and programmers commonly referred to as software developers or software engineers.

Module – 1 Introduction to Databases

Workers behind the Scene

Those who work to maintain the database system environment but who are not actively interested in the database contents as part of their daily job.

- **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. Tools are optional packages that are often purchased separately.
- **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.

Module – 1 Introduction to Databases

Advantages of Using the DBMS Approach

- Controlling Redundancy.
- Restricting Unauthorized Access.
- Providing Persistent Storage for Program Objects.
- Providing Storage Structures and Search Techniques for Efficient Query Processing.
- Providing Backup and Recovery.
- Providing Multiple User Interfaces.
- Representing Complex Relationships among Data.
- Enforcing Integrity Constraints.
- Permitting Inferencing and Actions Using Rules and Triggers.

Module – 1 Introduction to Databases

Advantages of Using the DBMS Approach

Controlling Redundancy

- In **traditional file processing** system, every user group maintains its own files for handling its data-processing applications.
- This approach causes redundancy means all users may have same set of data.
- Redundancy may causes data inconsistency and consumes more memory.
- In the **database approach**, the views of different user groups are integrated during database design.
- We should have a database design that stores each logical data item such as a student's name or birth date in **only one place in the database**. This is known as data normalization.
- This approach **reduces the redundancy**.

Module – 1 Introduction to Databases

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 such as salaries and bonuses 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.
- A DBMS should provide a security and authorization subsystem, which the DBA uses to create accounts and to specify account restrictions.

Providing Persistent Storage for Program Objects

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

Module – 1 Introduction to Databases

- Object-oriented database systems typically offer data structure compatibility with one or more object-oriented programming languages.

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

Module – 1 Introduction to Databases

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.
- 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.

Providing Multiple User Interfaces

- Many types of users with varying levels of technical knowledge use a database, a 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.

Module – 1 Introduction to Databases

Representing Complex Relationships among Data

- A database may include numerous varieties of data that are interrelated in many ways.
- Consider the university database example shown above, the record for 'Brown' in the STUDENT file is related to four records in the GRADE_REPORT file.

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.
- The simplest type of integrity constraint involves specifying a data type for each data item, similarly DBMS provide referential integrity, key or uniqueness constraint, etc.,

Module – 1 Introduction to Databases

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**.
- A trigger is a form of a rule activated by updates to the table, which results in performing some additional operations to some other tables, sending messages, and so on.

Module – 1 Introduction to Databases

Additional Implications of Using the Database Approach

- Potential for Enforcing Standards.
- Reduced Application Development Time.
- Flexibility.
- Availability of Up-to-Date Information.
- Economies of Scale.

Potential for Enforcing Standards:

- The database approach permits the DBA to define and enforce standards among database users in a large organization.
- This 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, terminology, and so on.

Module – 1 Introduction to Databases

Additional Implications of Using the Database Approach

Reduced Application Development Time:

- A prime selling feature of the database approach is that developing a new application such as the retrieval of certain data from the database for printing a new report takes very little time.
- Designing and implementing a large multiuser database from scratch may take more time than writing a single specialized file application.

Flexibility:

- Some time it may be necessary to change the structure of a database as requirements change.
- For example, it may be necessary to add a file(table) to the database or to extend the data elements(column) in an existing file(table).
- Modern DBMSs allow certain types of evolutionary changes to the structure of the database without affecting the stored data and the existing application programs.

Module – 1 Introduction to Databases

Additional Implications of Using the Database Approach

Availability of Up-to-Date Information:

- A DBMS makes the database available to all users.
- As soon as one user's update is applied to the database, all other users can immediately see this update.
- This availability of up-to-date information is essential for many transaction-processing applications, such as reservation systems or banking databases.

Economies of Scale:

- The DBMS approach permits **consolidation of** data and applications, thus reducing the amount of wasteful overlap between activities of data-processing personnel in different projects or departments as well as redundancies among applications.

Module – 1 Introduction to Databases

Additional Implications of Using the Database Approach

- This enables the whole organization to invest in more powerful processors, storage devices, or networking gear, rather than having each department purchase its own (lower performance) equipment.
- This reduces overall costs of operation and management.

A Brief History of Database Applications

Early Database Applications Using Hierarchical and Network Systems:

- The main types of early systems were based on three main paradigms: hierarchical systems, network model–based systems, and inverted file systems.
- Most of these database systems were implemented on large and expensive mainframe computers starting in the mid-1960s and continuing through the 1970s and 1980s.

Module – 1 Introduction to Databases

- One of the main problems with early database systems was the intermixing of conceptual relationships with the physical storage and placement of records on disk.
- Hence, these systems did not provide sufficient data abstraction and program-data independence capabilities.
- These systems were provided only programming language interfaces.
- This made it time-consuming and expensive to implement new queries and transactions, since new programs had to be written, tested, and debugged.

Providing Data Abstraction and Application Flexibility with Relational Databases:

- Relational databases were originally proposed to separate the physical storage of data from its conceptual representation and to provide a mathematical foundation for data representation and querying.

Module – 1 Introduction to Databases

- The relational data model also introduced high-level query languages that provided an alternative to programming language interfaces, making it much faster to write new queries.
- Relational database management systems (RDBMS) introduced in the early 1980s were quite slow, since they did not use physical storage pointers or record placement to access related data records.
- With the development of new storage and indexing techniques and better query processing and optimization, their performance improved.

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).

Module – 1 Introduction to Databases

- The complexity of the model and the lack of an early standard contributed to their limited use.
- 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

- Starting in the 1990s, electronic commerce (e-commerce) emerged as a major application on the Web.
- Much of the critical information on e-commerce Web pages is dynamically extracted data from DBMSs, such as flight information, product prices, and product availability.
- 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.

Module – 1 Introduction to Databases

Extending Database Capabilities for New Applications

- Database systems now offer extensions to better support the specialized requirements for some of advanced applications.

The following are some examples of these applications:

- **Scientific applications** that store large amounts of data resulting from scientific experiments in areas such as high-energy physics, the mapping of the human genome, and the discovery of protein structures.
- **Storage and retrieval of images**, including scanned news or personal photographs, satellite photographic images, and images from medical procedures such as x-rays and MRI (magnetic resonance imaging) tests.
- **Storage and retrieval of videos**, such as movies, and video clips from news or personal digital cameras.

Module – 1 Introduction to Databases

- **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

- 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.

Module – 1 Introduction to Databases

- 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 non-traditional types of data, such as social media posts and tweets.
- Some of the requirements of these new systems were not compatible with SQL relational DBMSs (SQL is the standard data model and language for relational databases).
- NoSQL databases were introduced, the term NOSQL is generally interpreted as Not Only SQL, meaning that the database servers stores structured data and also unstructured data.

Module – 1 Introduction to Databases

Overview of Database System Languages and Architecture

Module – 1 Introduction to Databases

Overview

- The architecture of DBMS packages has evolved from the early monolithic systems, where the whole DBMS software package was one tightly integrated system, to the modern DBMS packages that are modular in design, with a **client/server system architecture**.
- The current cloud computing environments consist of thousands of large servers managing so-called **big data** for **users on the Web**.
- In a basic **client/server DBMS architecture**, the system functionality is distributed between two types of modules.
- A **client module** is typically designed so that it will run on a mobile device, user workstation, or personal computer (PC).
- Typically, **application programs and user interfaces** that access the database run in the client module.

Module – 1 Introduction to Databases

Overview

- Hence, the **client module** handles user interaction and provides the user-friendly interfaces such as apps for mobile devices, or forms- or menu-based GUIs (graphical user interfaces) for PCs.
- The other kind of module, called a **server module**, typically handles data storage, access, search, and other functions.

Data Models, Schemas, and Instances

A data model:

- A collection of concepts that can be used to **describe the structure of a database** which provides the necessary means to achieve the data abstraction.
- Structure of a database means the data types, relationships, and constraints that apply to the data.
- Most data models also include a set of basic operations for specifying retrievals and updates on the database.

Module – 1 Introduction to Databases

Data Models, Schemas, and Instances

- Also data model include the concepts to specify the **dynamic aspect or behavior** of a database application.
- This allows the database designer to specify a **set of valid user-defined operations** that are allowed on the database objects.
- An example of a user-defined operation could be **COMPUTE_GPA**(grade point average), which can be applied to a STUDENT object.
- Generic operations to insert, delete, modify, or retrieve any kind of object are often included in the basic data model operations.

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.

Module – 1 Introduction to Databases

Data Models, Schemas, and Instances

Categories of Data Models

1. High-level or conceptual data models.
2. Low-level or physical data models.
3. Representational (or implementation) data models.

High-level or conceptual data models

- Provide concepts that are close to the way many users perceive(understand) data.
- 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.
- An **attribute** represents some property which describes an entity, such as the employee's name or salary.

Module – 1 Introduction to Databases

Data Models, Schemas, and Instances

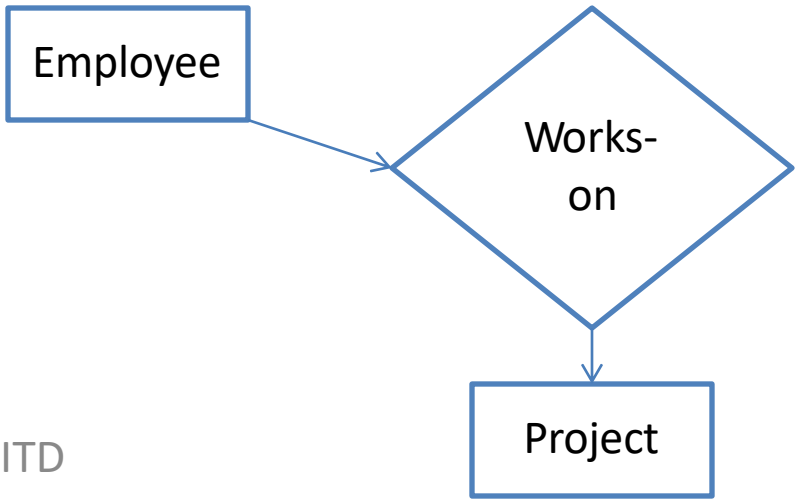
- A **relationship** among two or more entities **represents an association** among the entities, for example, a works-on relationship between an employee and a project.

Employee

EMP_ID	EMP_Name	EMP_DOJ	EMP_DESG	Works_On
				8450
				8455

Project

PROJ_ID	PROJ_Name	PROJ_Loc
8450		
8451		
8453		
8455		



Module – 1 Introduction to Databases

Data Models, Schemas, and Instances

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.
- Concepts provided by physical data models are generally **meant for computer specialists**, not for end users.
- 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.

Module – 1 Introduction to Databases

Data Models, Schemas, and Instances

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.
- 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—that have been widely used in the past.

Module – 1 Introduction to Databases

Data Models, Schemas, and Instances

- Another class of data models is known as **self-describing data models.**
- The data storage in systems based on these models combines the description of the data with the data values themselves.
- These models include XML as well as many of the key-value stores and NOSQL systems that were recently created for managing big data.
- In traditional DBMSs, the description (schema) is separated from the data.

Module – 1 Introduction to Databases

Schemas, Instances, and Database State

Schema:

- 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.
- The schema diagram shown below displays the structure of each record type but not the actual instances of records.

Module – 1 Introduction to Databases

Schemas, Instances, and Database State

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 :Schema diagram for the university database.

Module – 1 Introduction to Databases

Schemas, Instances, and Database State

- The above schema diagram shows neither the data type of each data item nor the relationships among the various files. Many types of constraints are not represented in schema diagrams.
- Constraints are quite difficult to represent diagrammatically.
- **State**: 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.
- Every time we **insert or delete** a record or **change the value** of a data item in a record, we change one state of the database into another state.
- When we **define a new database**, we specify its database schema only to the DBMS. At this point, the corresponding database state is the empty state with no data**.**

Module – 1 Introduction to Databases

Schemas, Instances, and Database State

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

```
MySQL  localhost:33060+ ssl  world  SQL > desc city;
+-------------+----------+------+-----+---------+----------------+
| Field       | Type     | Null | Key | Default | Extra          |
+-------------+----------+------+-----+---------+----------------+
| ID          | int      | NO   | PRI | NULL    | auto_increment |
| Name        | char(35) | NO   |     |         |                |
| CountryCode | char(3)  | NO   | MUL |         |                |
| District    | char(20) | NO   |     |         |                |
| Population  | int      | NO   |     | 0       |                |
+-------------+----------+------+-----+---------+----------------+
5 rows in set (0.0025 sec)
```

Module – 1 Introduction to Databases

Three-Schema Architecture and Data Independence

1. Use of a catalog to store the database description (schema) so as to make it self-describing.
2. Insulation of programs and data (program-data and program-operation independence).
3. Support of multiple user views.

- **Three-schema architecture,** that was proposed to help achieve and visualize these characteristics.
- The **goal of the three-schema arc**hitecture is to separate the user applications from the physical database.

Internal schema

- 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.

Module – 1 Introduction to Databases

Three-Schema Architecture and Data Independence

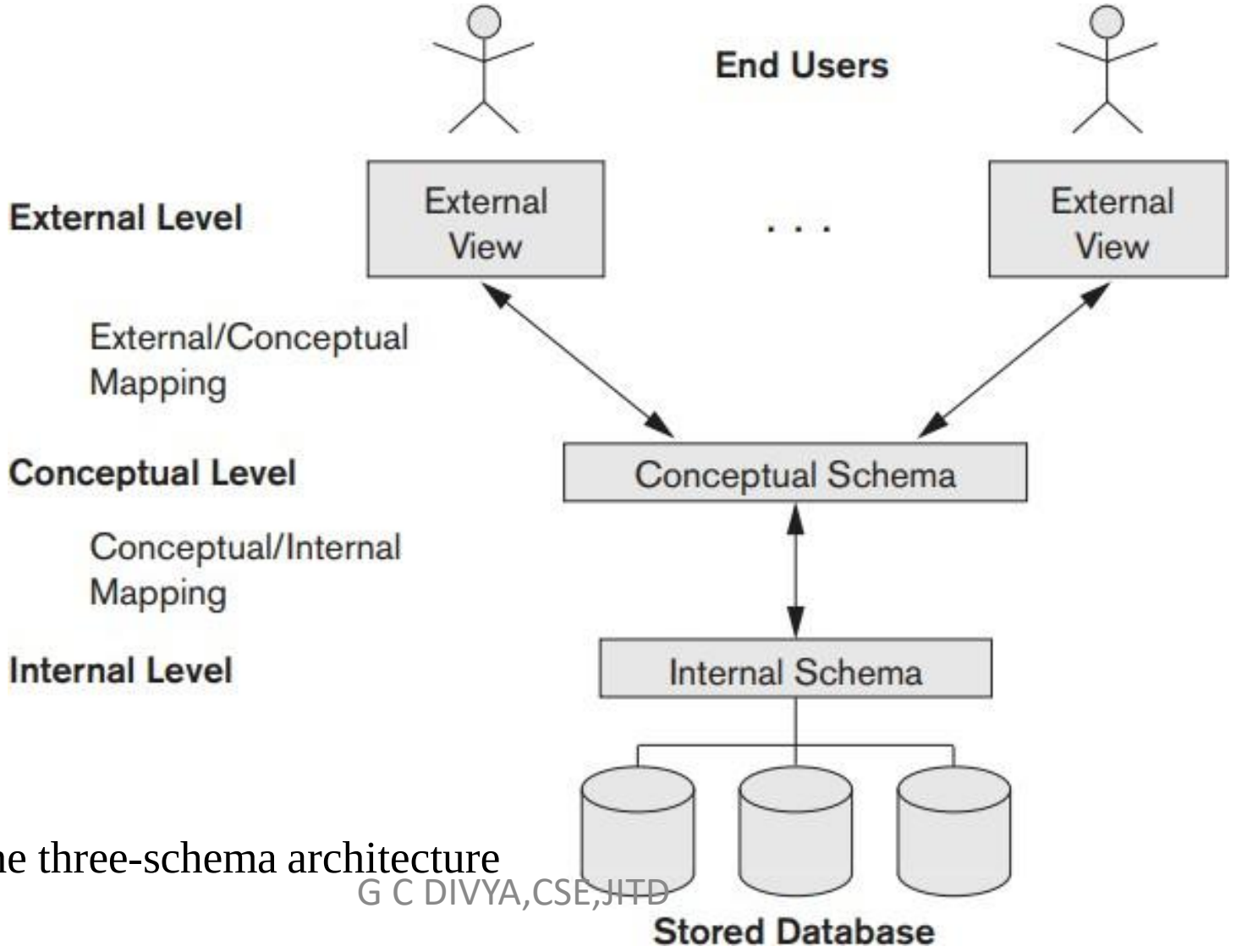


Figure: The three-schema architecture

Module – 1 Introduction to Databases

Three-Schema Architecture and Data Independence

Conceptual schema

- 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.
- This implementation conceptual schema is often based on a conceptual schema design in a high-level data model.

Module – 1 Introduction to Databases

Three-Schema Architecture and Data Independence

External schema

- 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 conceptual data model.

NOTE: In most DBMSs that support user views, external schemas are specified in the same data model that describes the conceptual-level information (**for example**, a relational DBMS like Oracle or SQL-Server uses SQL for this).

Module – 1 Introduction to Databases

Data Independence

Data independence, which 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.

There are two types of data independence:

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 (by adding a record type or data item), to change constraints, or to reduce the database (by removing a record type or data item).
- In the last case, external schemas that refer only to the remaining data should not be affected.

Module – 1 Introduction to Databases

Data Independence

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.

Note: If the same data as before remains in the database, we should not have to change the conceptual schema.

Module – 1 Introduction to Databases

Database Languages and Interfaces

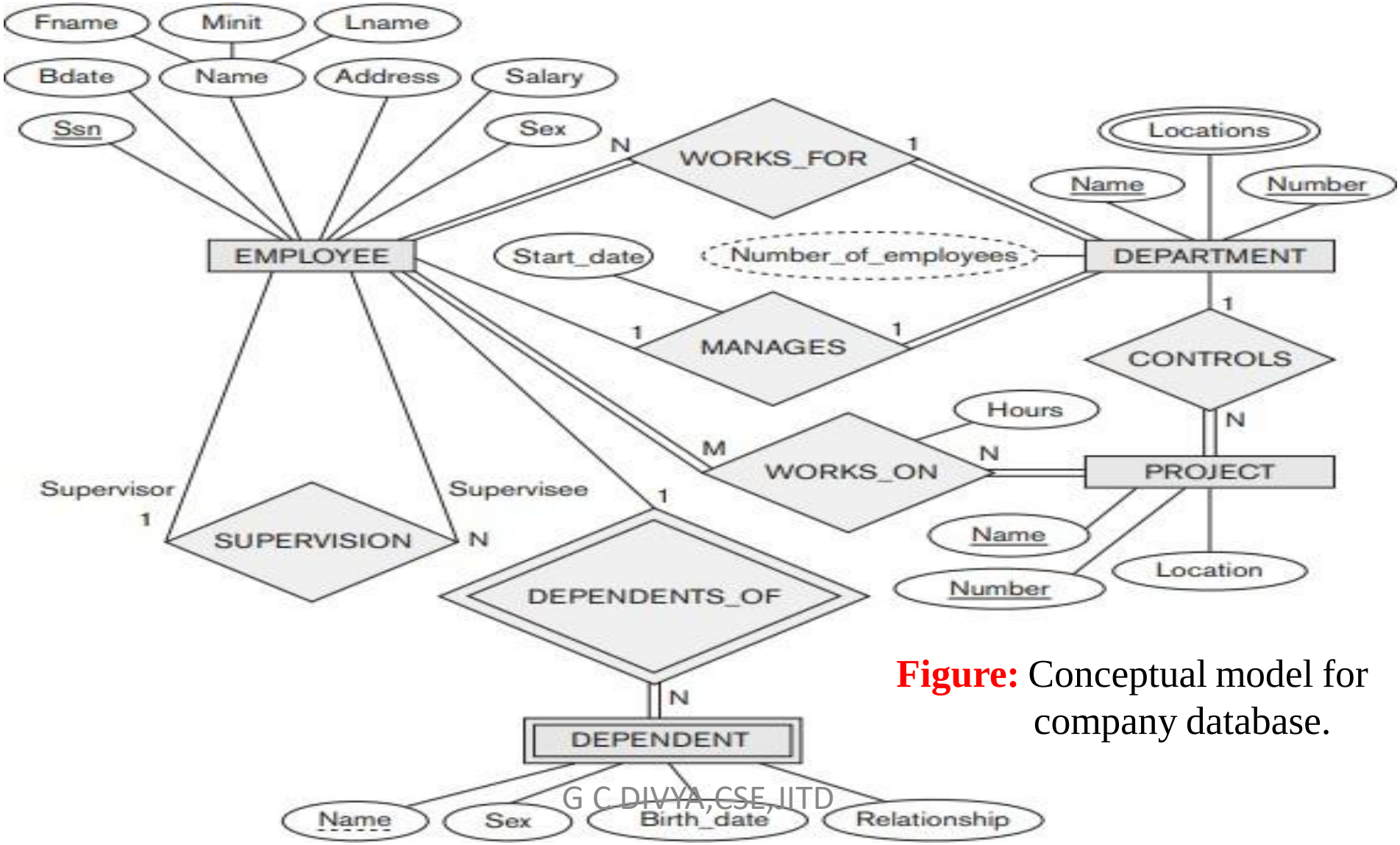


Figure: Conceptual model for company database.

Module – 1 Introduction to Databases

Database Languages and Interfaces

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

DBMS Languages

- Once the design(conceptual) 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)

- In many DBMSs **where no strict separation of levels is maintained**, **DDL** is used by the DBA and by database designers to define both schemas.
- 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.

Module – 1 Introduction to Databases

Database Languages and Interfaces

- In DBMSs **where a clear separation is maintained** between the conceptual and internal levels, the DDL is used to specify the conceptual schema only.
- Another language called 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.
- In **most relational DBMSs today**, there is no specific language that performs the role of SDL.
- Instead, the internal schema is specified by a combination of functions, parameters, and specifications related to storage of files.
- These permit the DBA staff to control indexing choices and mapping of data to storage.

Module – 1 Introduction to Databases

Database Languages and Interfaces

Example - 1: CREATE TABLE geeksforgeeks (article_title varchar(65000) ENGINE = MEMORY;

- In the above example statement the table is stored in memory(HEAP) and this is supported in MySQL DBMS.

Example - 2: CREATE TABLE f (x int, y varchar(25));

- In the above example statement, Storage Definition Language (SDL) defines storage of row **int, varchar(25)**.
- For a **true three-schema architecture**, we would need a third language, the 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.
- In relational DBMSs, **SQL is used** in the role of VDL to define user or application views as results of predefined queries.

Module – 1 Introduction to Databases

Database Languages and Interfaces

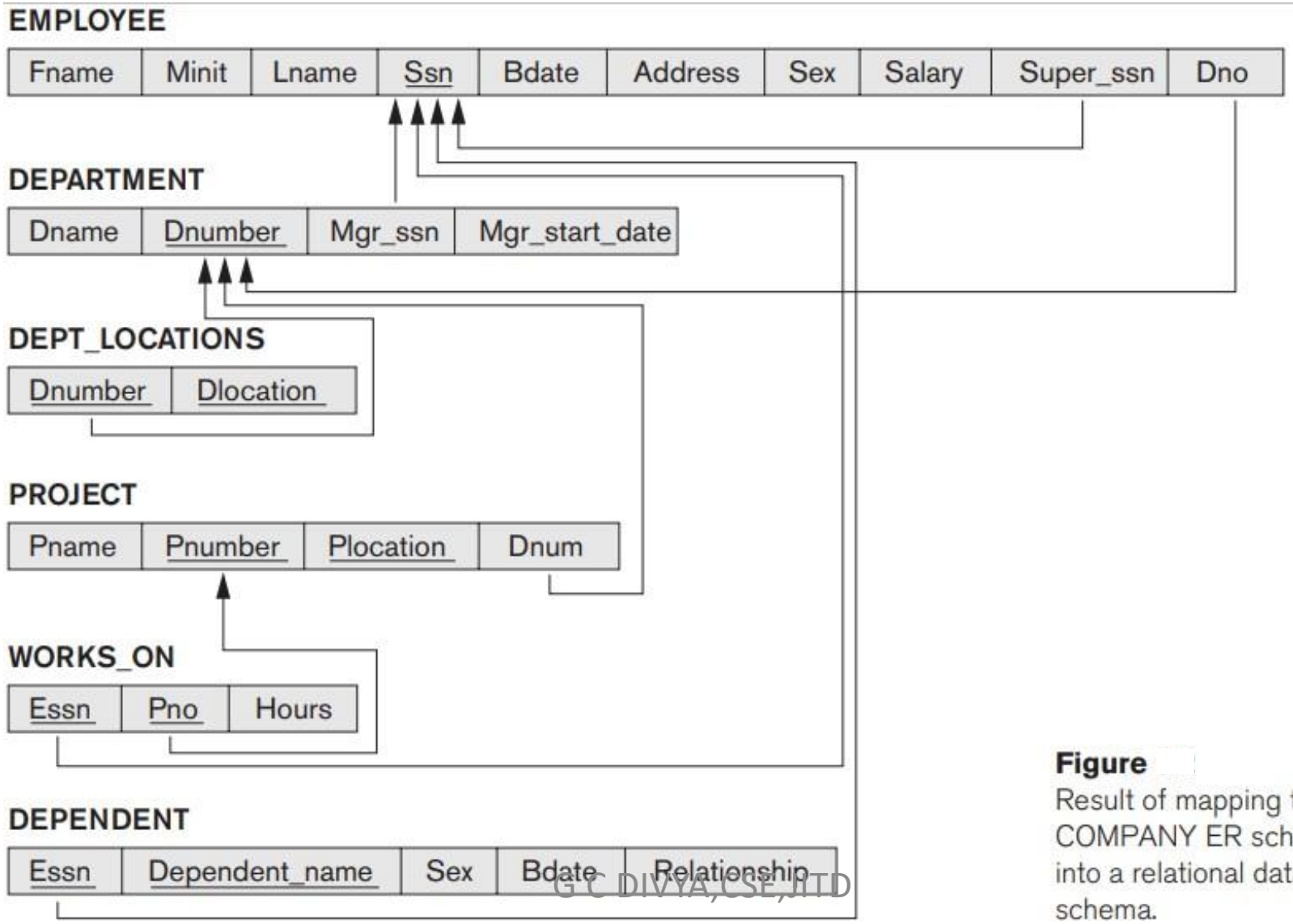


Figure
Result of mapping the COMPANY ER schema into a relational database schema.

Module – 1 Introduction to Databases

Database Languages and Interfaces

Data Manipulation Language (DML)

- Once the database schemas are compiled, next is to populate the database with data, this means manipulating the database.
- Typical manipulations include retrieval, insertion, deletion, and modification of the data.
- The DBMS provides a set of operations or a language called the data manipulation language (**DML**) for these purposes.

There are two main types of DMLs

1. A high-level or nonprocedural DML
2. A low-level or procedural DML

Module – 1 Introduction to Databases

Database Languages and Interfaces

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

Module – 1 Introduction to Databases

Database Languages and Interfaces

A 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.
- Therefore, it needs to use programming language constructs, such as looping, to retrieve and process each record from a set of records.
- Low-level DMLs are also called record-at-a-time DMLs.

NOTE: 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.**

Module – 1 Introduction to Databases

Database Languages and Interfaces

DBMS Interfaces

- 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

Module – 1 Introduction to Databases

Database Languages and Interfaces

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.
- SQL*Forms is a form-based language that specifies queries using a form designed in conjunction with the relational database schema.

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

Module – 1 Introduction to Databases

Database Languages and Interfaces

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, that are used to interpret the request.

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).

Module – 1 Introduction to Databases

Database Languages and Interfaces

- Not yet common in structured relational databases, although a research area called **keyword-based querying.**

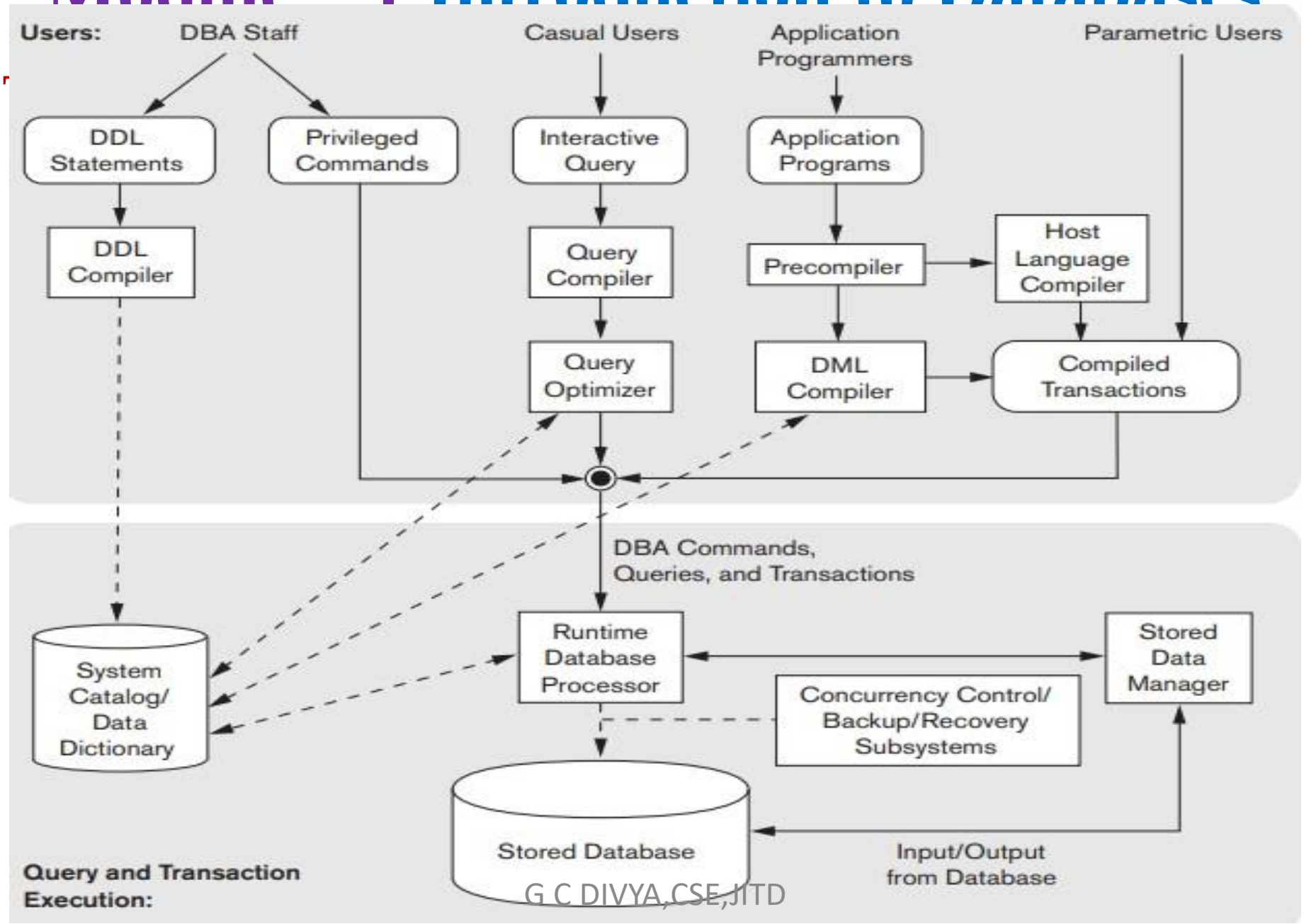
Speech Input and Output.

- 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 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.

Module – 1 Introduction to Databases



Module – 1 Introduction to Databases

The Database System Environment

- 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.
- 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.
- The DBA staff works on defining the database and tuning it by making changes to its definition using the **DDL** and other **privileged commands.**
- The **DDL compiler** processes schema definitions, specified in the DDL, and stores descriptions of the schemas (meta-data) in the DBMS catalog.
- **Casual users and persons** with occasional need for information from the database interact using the interactive query interface.

Module – 1 Introduction to Databases

The Database System Environment

- These queries are parsed and validated for correctness of the query syntax, the names of files and data elements by a **query compiler** that compiles them into an internal form.
- The **query optimizer** optimizes the query by rearranging and possible reordering of operations, elimination of redundancies, and use of efficient search algorithms during execution and makes calls on the **runtime processor**.
- Application programmers write programs in host languages such as Java, C, or C++ that are submitted to a **pre-compiler**.
- The pre-compiler extracts DML commands from an application program written in a host programming language.
- These commands are sent to the DML compiler for compilation into object code for database access.

Module – 1 Introduction to Databases

The Database System Environment

The runtime **database processor** executes

1. the privileged commands,
2. the executable query plans, and
3. the canned transactions with runtime parameters.

- It works with the system catalog and may update it with statistics.
- It also works with the stored data manager, which in turn uses basic operating system services for carrying out low-level input/output (read/write) operations between the disk and main memory.
- **Concurrency Control and Backup and Recovery** systems are integrated into the working of the runtime database processor for purposes of transaction management.

Module – 1 Introduction to Databases

References:

1. Difference Between SQL and MySQL – javatpoint
2. database theory - What is a Storage Definition Language (SDL)? - Database Administrators Stack Exchange
3. Difference between MySQL and Oracle - javatpoint

Module – 1 Introduction to Databases

Conceptual Data Modelling using Entities and Relationships

Module – 1 Introduction to Databases

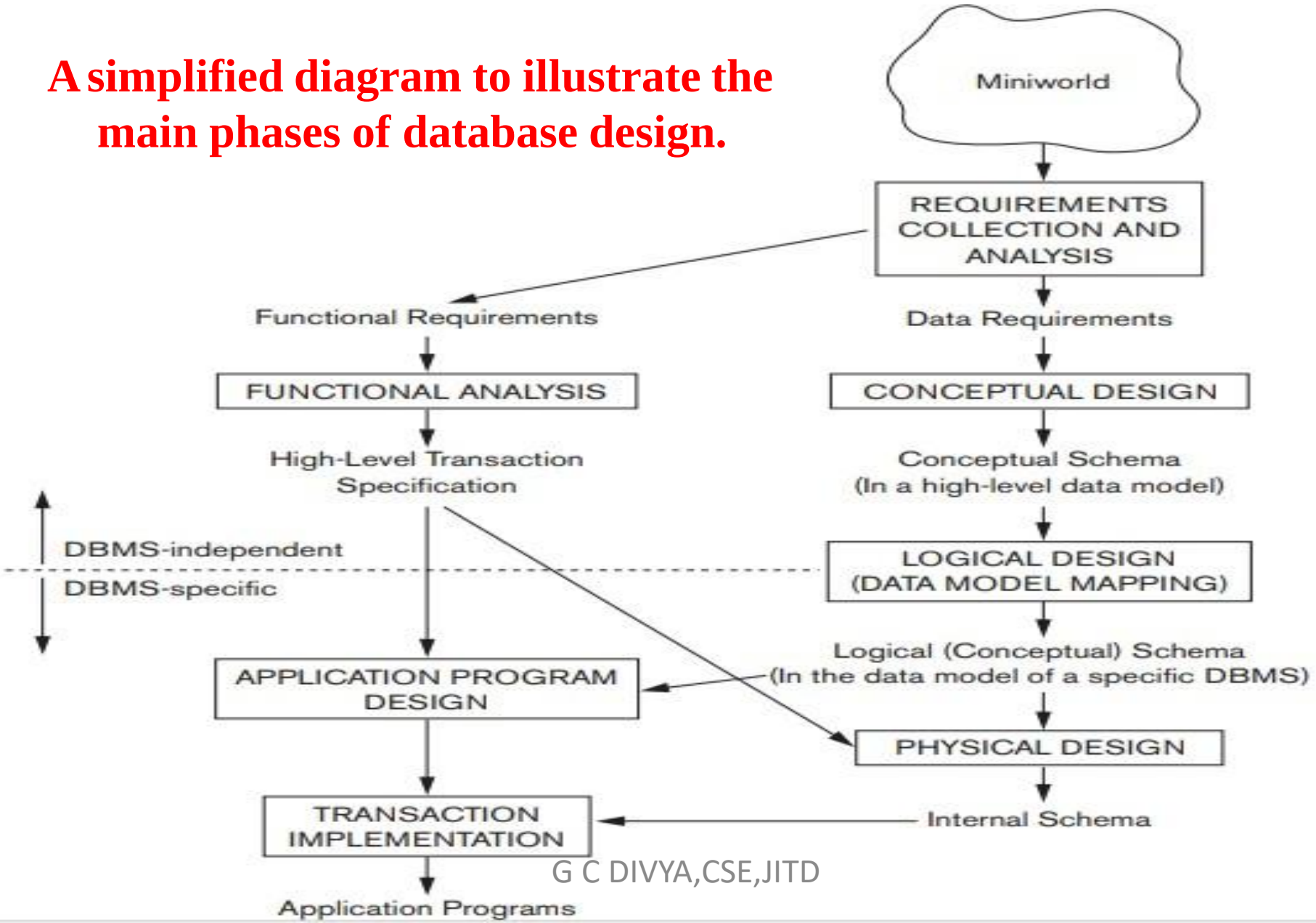
- Entity–relationship (ER) model is a popular high-level conceptual data model.
- This model and its variations are frequently used for the conceptual design of database applications, and many database design tools employ its concepts.

Using High-Level Conceptual Data Models for Database Design

- The first step is requirements collection and analysis.
- During this step, the database designers interview prospective database users to understand and document their data requirements.
- 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.

Module – 1 Introduction to Databases

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



Module – 1 Introduction to Databases

- 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.
- The high-level conceptual schema can also be **used as a reference to ensure** that all users' data requirements are met and that the requirements do not conflict.
- 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 (SQL)** 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**.

Module – 1 Introduction to Databases

- 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 high-level transaction specifications.

A Sample Database Application

- Database application, called COMPANY.
- The COMPANY database keeps track of a company's employees, departments, and projects.

Data Requirements

- The company is organized into **departments**. Each department has a unique name, a unique number, and a particular employee who manages the department.

Module – 1 Introduction to Databases

- A department controls a **number of projects**, each **project** has a unique name, a unique number, and a single location.
- The database will store each employee's name, Social Security number,2 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.
- 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.
- **Figure shown below** is the schema for this database application displayed by means of the graphical notation known as **ER diagrams.**

Module – 1 Introduction to Databases

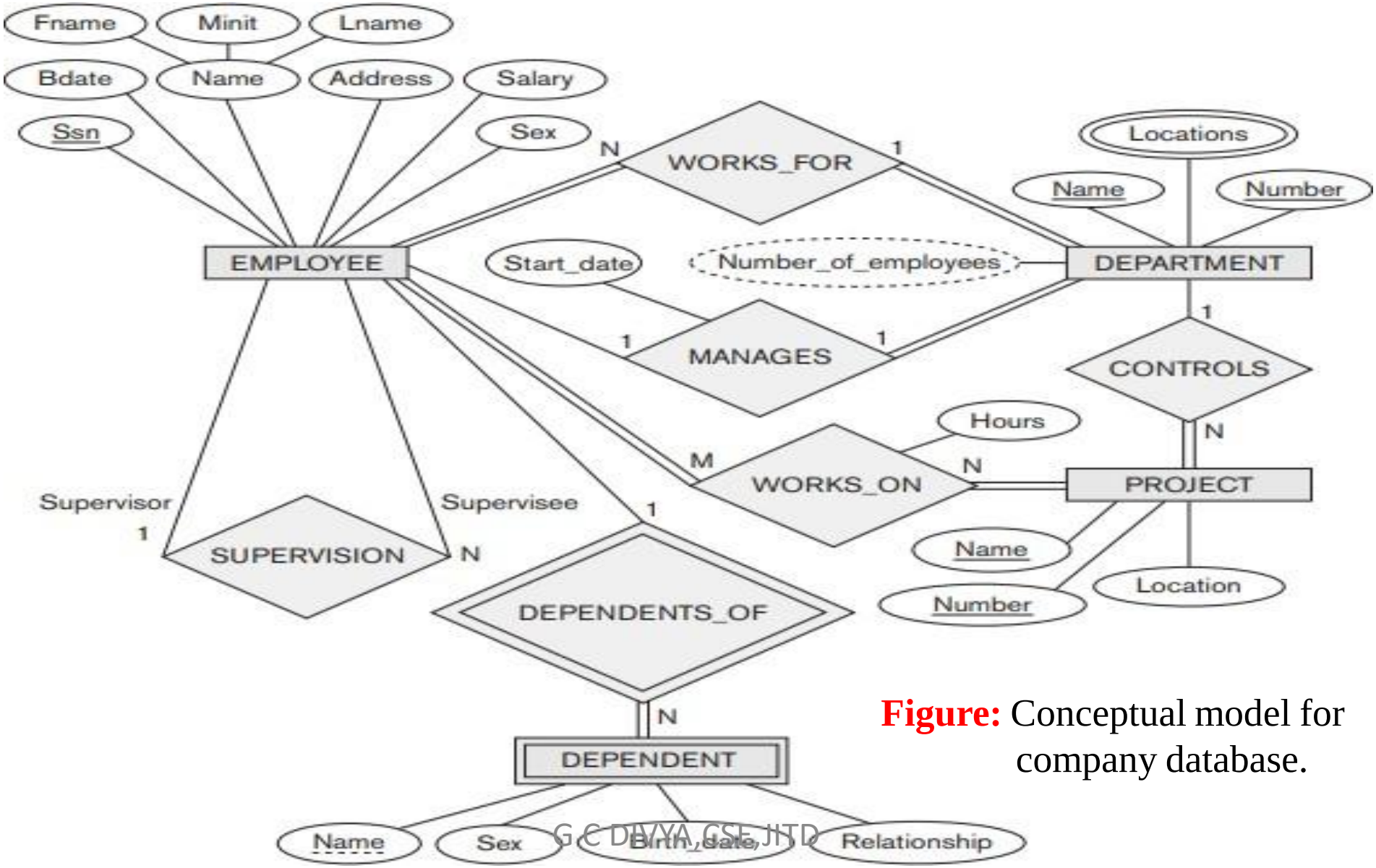


Figure: Conceptual model for company database.

Module – 1 Introduction to Databases

Entity Types, Entity Sets, Attributes, and Keys

Entity:

- 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).

Attributes:

- 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.
- A CAR entity may be described by the car's model, color, etc.,
- Every entity will have value for each of its attributes.

Module – 1 Introduction to Databases

Entity Types, Entity Sets, Attributes, and Keys

Several types of attributes occur in the ER model:

- simple versus composite
- single valued versus multivalued
- stored versus derived.

simple versus composite

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

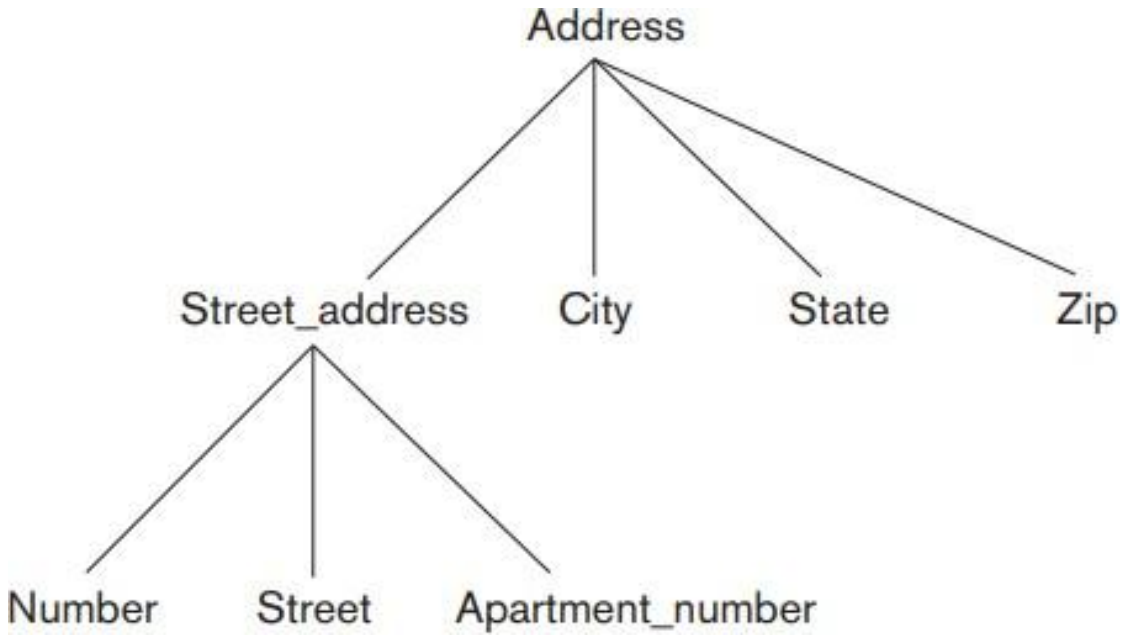
For example, the **Address attribute** of the EMPLOYEE entity shown in figure above can be subdivided into Street_address, City, State, and Zip_code.

Attributes that are not divisible are called **simple or atomic attributes**.

Module – 1 Introduction to Databases

Entity Types, Entity Sets, Attributes, and Keys

Composite attributes can form a hierarchy; for example, Street_address can be further subdivided into three simple component attributes: Number, Street, and Apartment_number.



Module – 1 Introduction to Databases

Entity Types, Entity Sets, Attributes, and Keys

Single-Valued versus Multivalued Attributes.

- The attribute which has single value for a particular entity; such attribute is called **single-valued**.
- **For example**, Age is a single-valued attribute of an entity person.
- In some cases an attribute can have a **set of values** for the same entity such attribute is called **multi-valued**.
- **For example**, a Colors attribute for a car entity, or a College_degrees attribute for a person entity can have more than one value.
- A multivalued attribute may have lower and upper bounds to constrain the number of values allowed for each individual entity.

Module – 1 Introduction to Databases

Entity Types, Entity Sets, Attributes, and Keys

Stored versus Derived Attributes

- In some cases, two (or more) attribute values are related.
- **For example**, the Age and Birth_date attributes of a person are related.
- For a particular **person entity**, the value of Age can be determined from the current (today's) date and the value of that person's Birth_date.
- 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**.

Module – 1 Introduction to Databases

Entity Types, Entity Sets, Attributes, and Keys

NULL Values.

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

Complex Attributes.

- In general, composite and multivalued attributes can be nested arbitrarily.

Module – 1 Introduction to Databases

Entity Types, Entity Sets, Attributes, and Keys

- 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))}

Entity Types and Entity Sets

- 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.

Module – 1 Introduction to Databases

Entity Types, Entity Sets, Attributes, and Keys

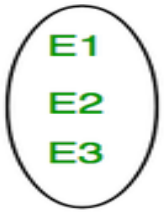
- The collection of all entities of a particular entity type in the database at any point in time is called an **entity set** or **entity collection.**
- An **entity type** is represented in ER diagrams as a rectangular box enclosing the entity type name.
- **Attribute names** are enclosed in ovals and 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.
- 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**.

Entity type & Entity Set , Value Sets (Domains) of Attributes

- For an entity type called Student, there will exist students and their respective attributes such as studentID, name and age.

Entity Type

- An Entity set is a comprehensive representation of all entities of the same type at a specific

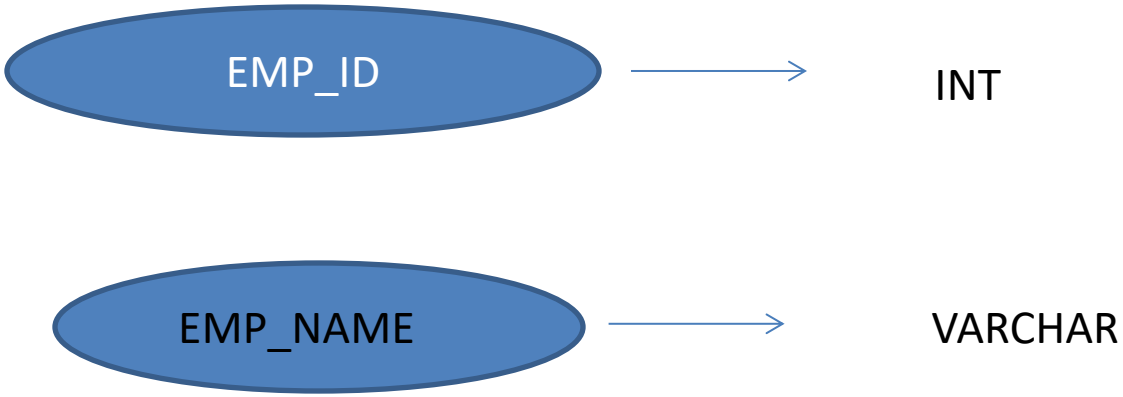


Entity Set

Cont.,

- Domain is a set of values that can be assigned to an attribute.

EX:



Module – 1 Introduction to Databases

Entity Types, Entity Sets, Attributes, and Keys

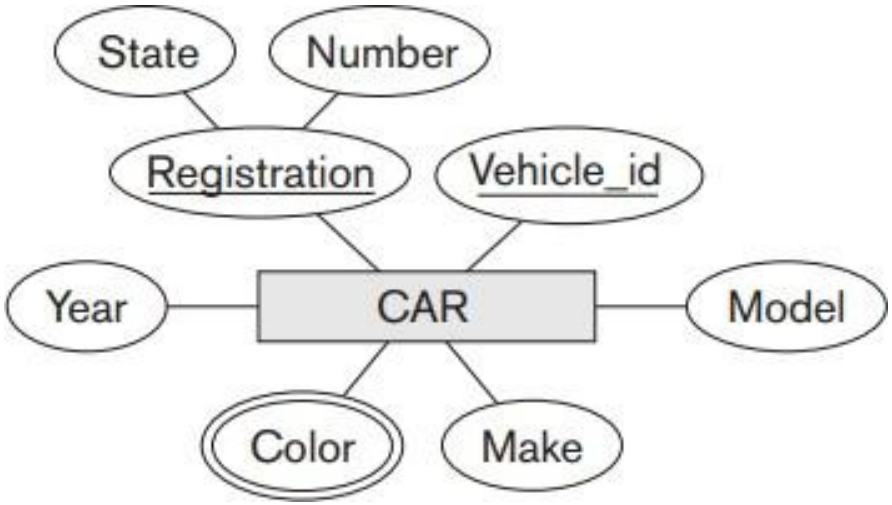
Key Attributes of an Entity Type.

- 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.
- 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.

Module – 1 Introduction to Databases

Entity Types, Entity Sets, Attributes, and Keys

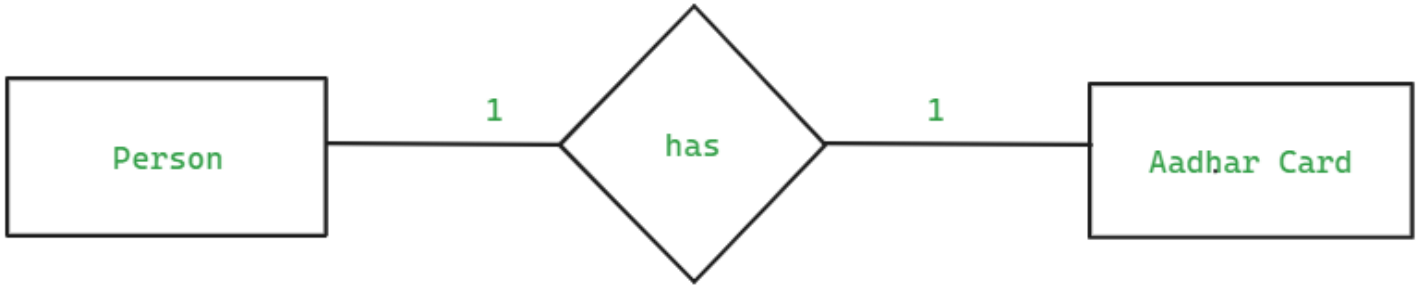
Key Attributes of an Entity Type



Reg. No	Vehicle_Id	Make	Color	Year	Model
KA-21-1234	SZ09823456	Maruti Swift	{red,black}	2019	XZI
KA-25-1234	SZ09824467	Maruti Swift	{red,black}	2020	VXI
KA-13-9269	TA12308729	Tata Safari	white	2018	XZA+

Relationship Types, Relationship Sets, Roles, and Structural Constraints

Relationship Types A **relationship type** R among n **entity types** $E_1, E_2, \ldots, E_n$ defines a **set of associations** or a **relationship set** among entities from the entity types.



Module – 1 Introduction to Databases

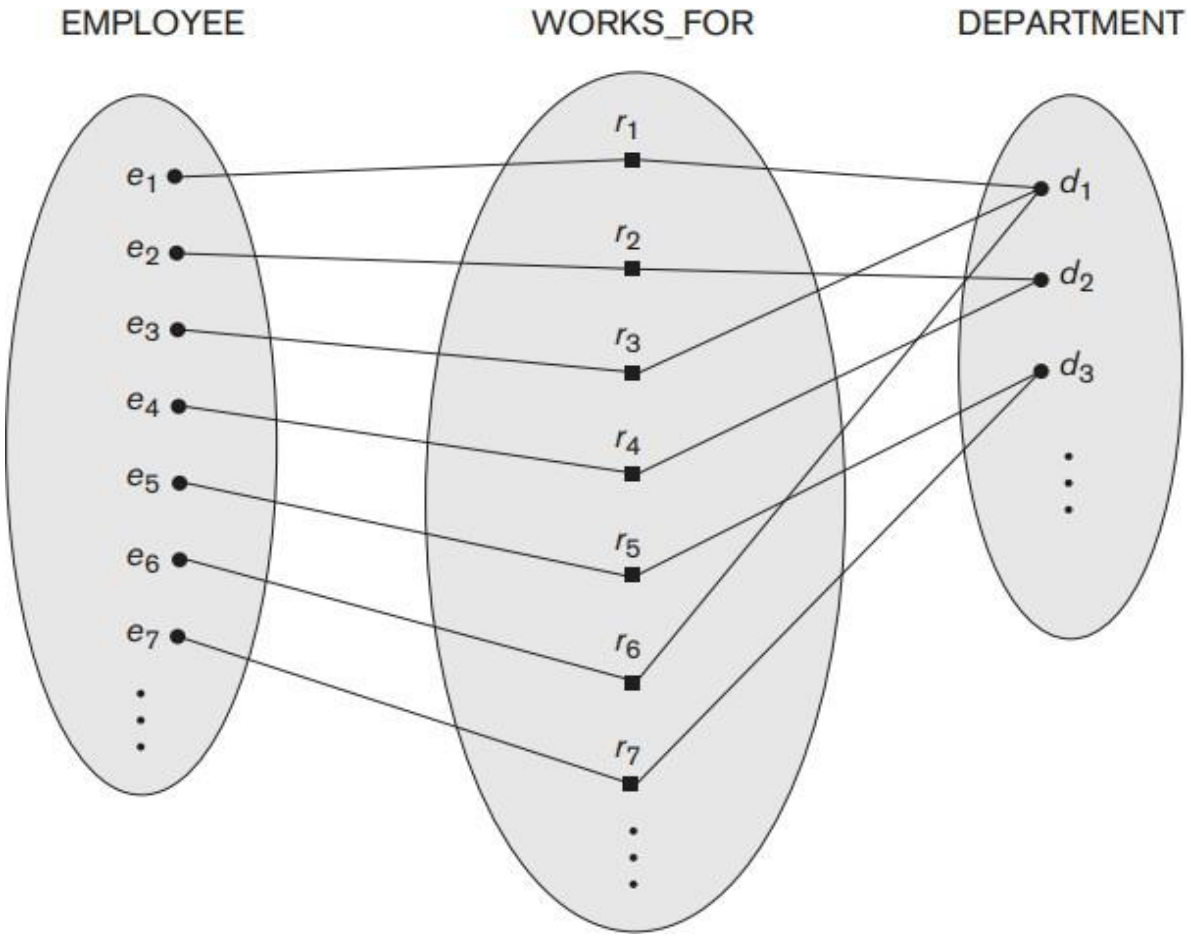


Figure
Some instances in the WORKS_FOR relationship set, which represents a relationship type WORKS_FOR between EMPLOYEE and DEPARTMENT.

Module – 1 Introduction to Databases

Relationship Degree

- The degree of a relationship type is the **number of participating entity types**.
- Hence, in the above example, the WORKS_FOR relationship is of degree two.
- A relationship type of degree two is **called binary**.
- A relationship type of degree three is **called ternary**.
- An **example of a ternary relationship** is SUPPLY, where each relationship instance r_i associates three entities they are **supplier** s, **part** p, and **project** j, whenever s supplies part p to project j.

Module – 1 Introduction to Databases

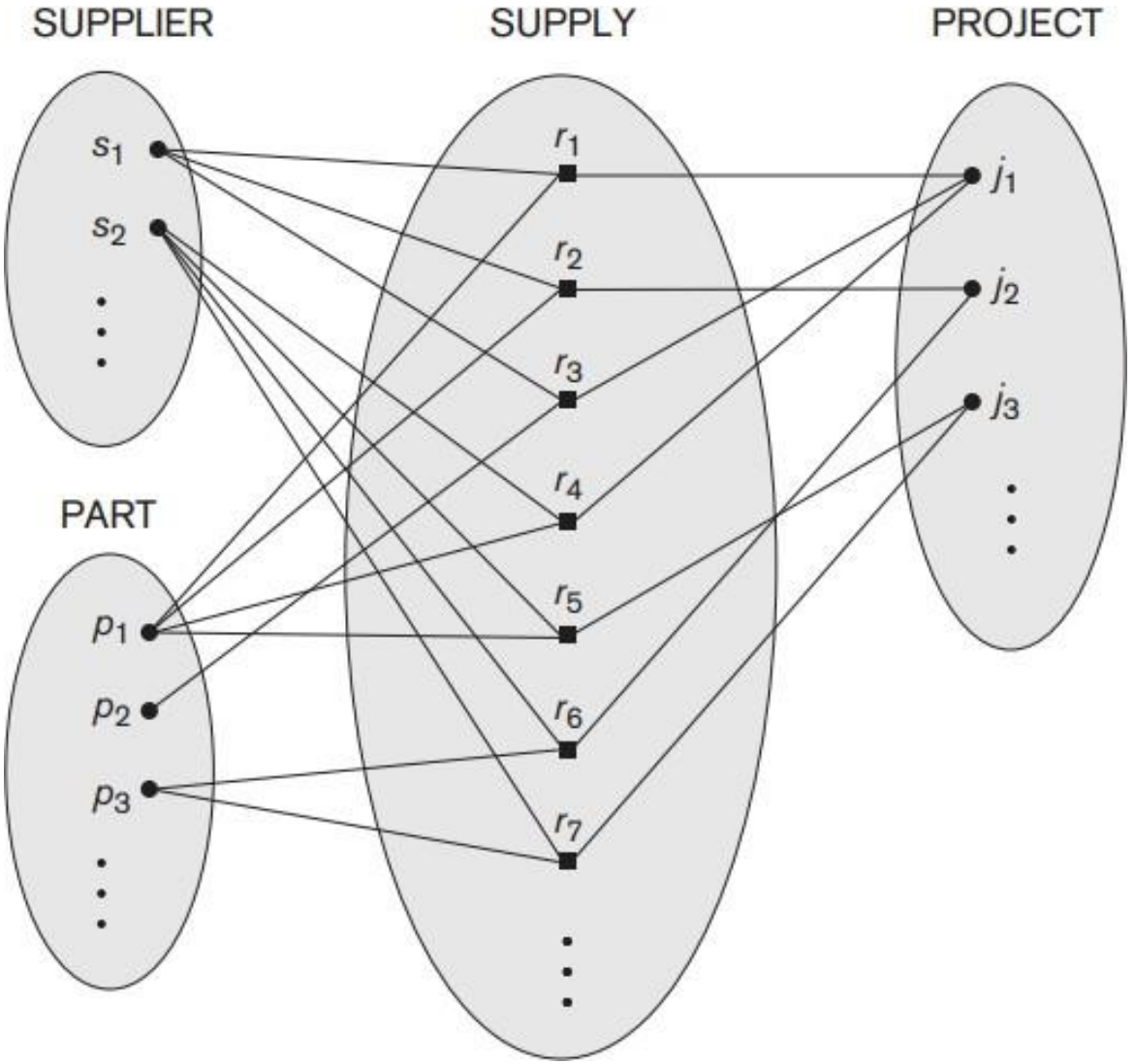


Figure
Some relationship instances in the SUPPLY **ternary relationship** set.

Module – 1 Introduction to Databases

Relationship Types, Relationship Sets, Roles, and Structural Constraints

Role Names and Recursive Relationships

Role Names:

- 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.

Module – 1 Introduction to Databases

Relationship Types, Relationship Sets, Roles, and Structural Constraints

Recursive Relationships

- 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** or **self-referencing relationships**.
- Example: **HOD** is an employee and as well as plays the role of HOD

Module – 1 Introduction to Databases

Relationship Types, Relationship Sets, Roles, and Structural Constraints

Recursive Relationships

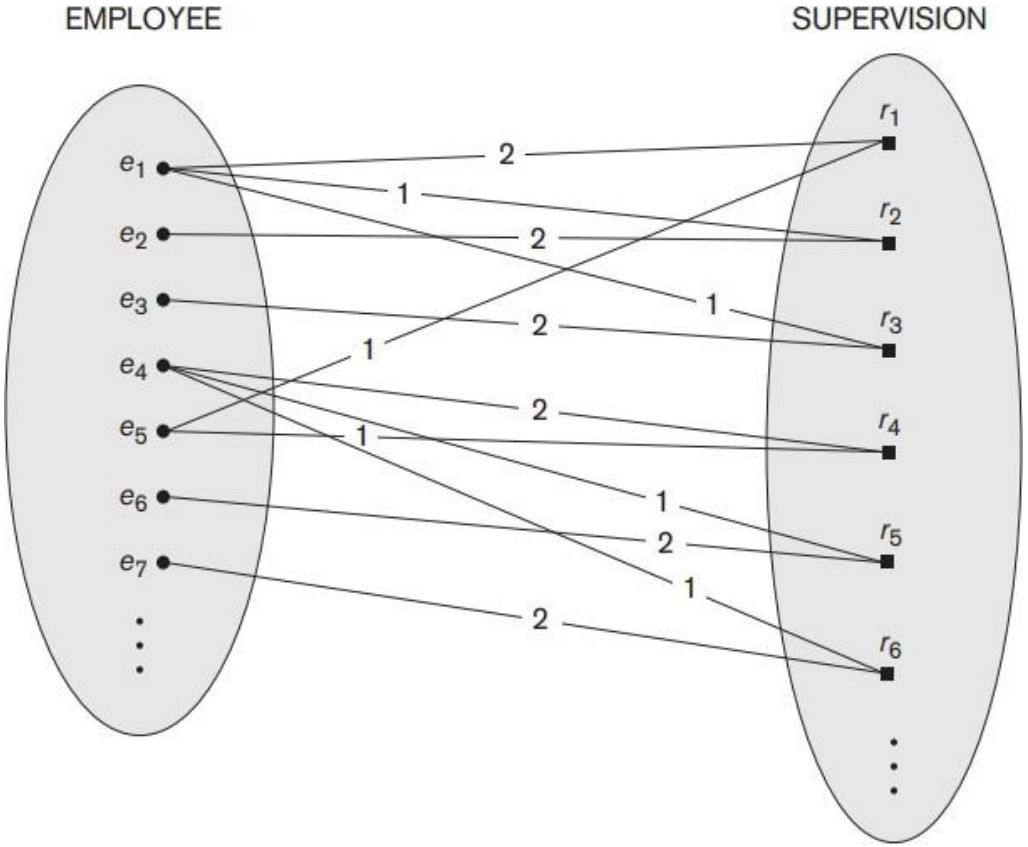


Figure: A recursive relationship SUPERVISION between EMPLOYEE in the supervisor role (1) and EMPLOYEE in the subordinate role (2).

Module – 1 Introduction to Databases

Relationship Types, Relationship Sets, Roles, and Structural Constraints

Constraints on Binary Relationship Types

- We can impose constraints on relationship types that limit the possible combinations of entities that may participate in the corresponding relationship set.
- If the company has a rule that each employee must work for exactly one department, then we can **describe this constraint in the schema**.
- There are two main types of **binary relationship constraints**:
  1. Cardinality ratio constraint.
  2. Participation constraint.

Module – 1 Introduction to Databases

Relationship Types, Relationship Sets, Roles, and Structural Constraints

Cardinality Ratios for Binary Relationships

Cardinality Ratio (specifies *maximum* number of relationship instances an entity can participate in)

a) One-to-one (1:1)

b) One-to-many (1:N) or Many-to-one (N:1)

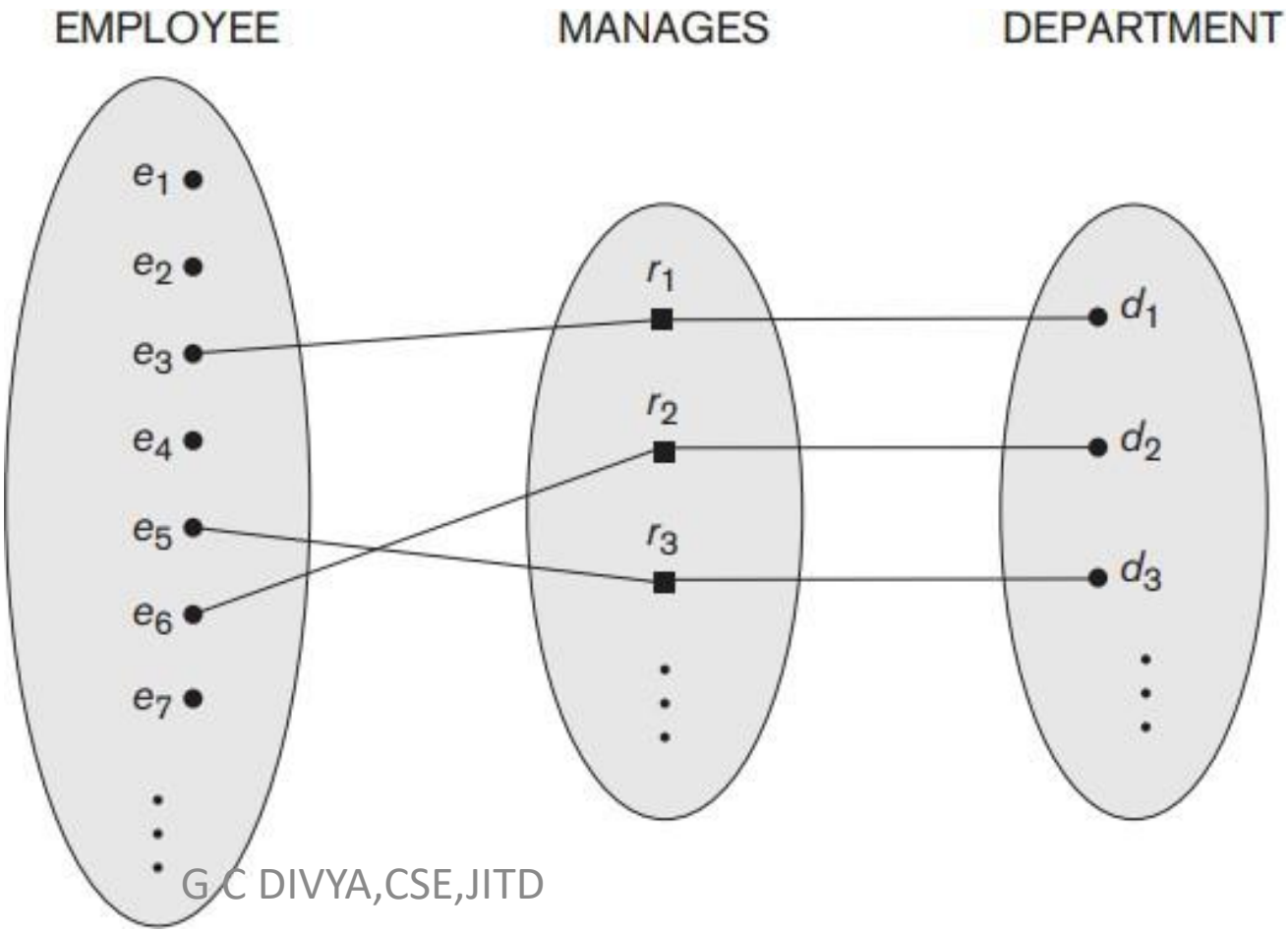
c) Many-to-many (M:N)

Module – 1 Introduction to Databases

Relationship Types, Relationship Sets, Roles, and Structural Constraints

Cardinality Ratios for Binary Relationships

Figure: An example of 1:1 cardinality ratio for relationship type MANAGES.

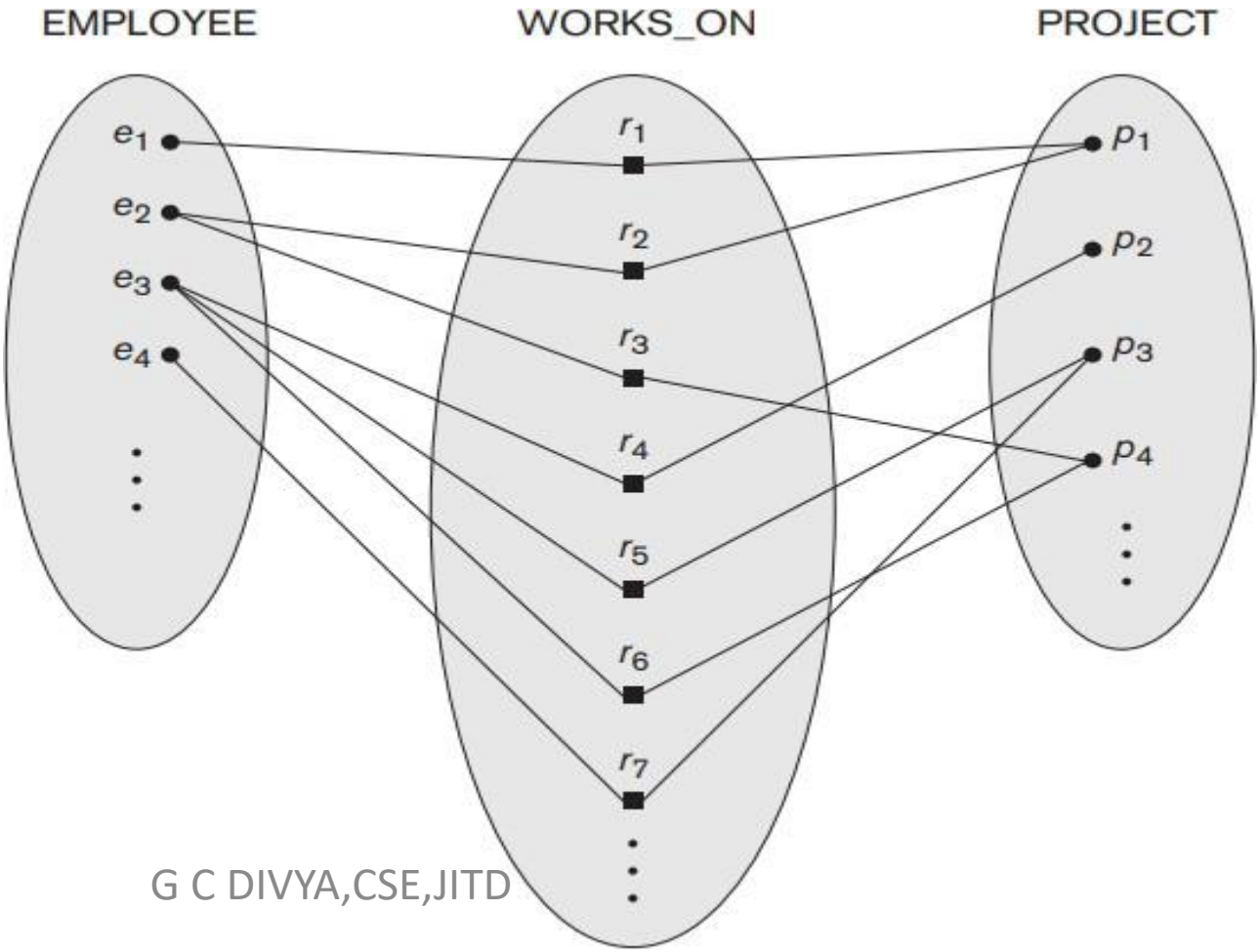


Module – 1 Introduction to Databases

Relationship Types, Relationship Sets, Roles, and Structural Constraints

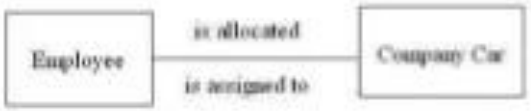
Cardinality Ratios for Binary Relationships

Figure: An example of M:N cardinality ratio for relationship type WORKS_ON.

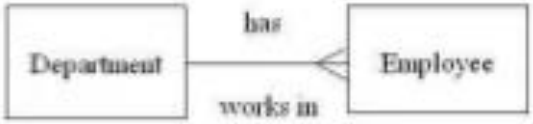


There are three degrees of Cardinality, known as:

- **One-to-One** (1:1):One occurrence of an entity relates to only one occurrence in another entity.



- **One-to-Many** (1:M): One occurrence in an entity relates to many occurrences in another entity.



- **Many-to-Many** (M:N) Many occurrences in an entity relate to many occurrences in another entity.

Module – 1 Introduction to Databases

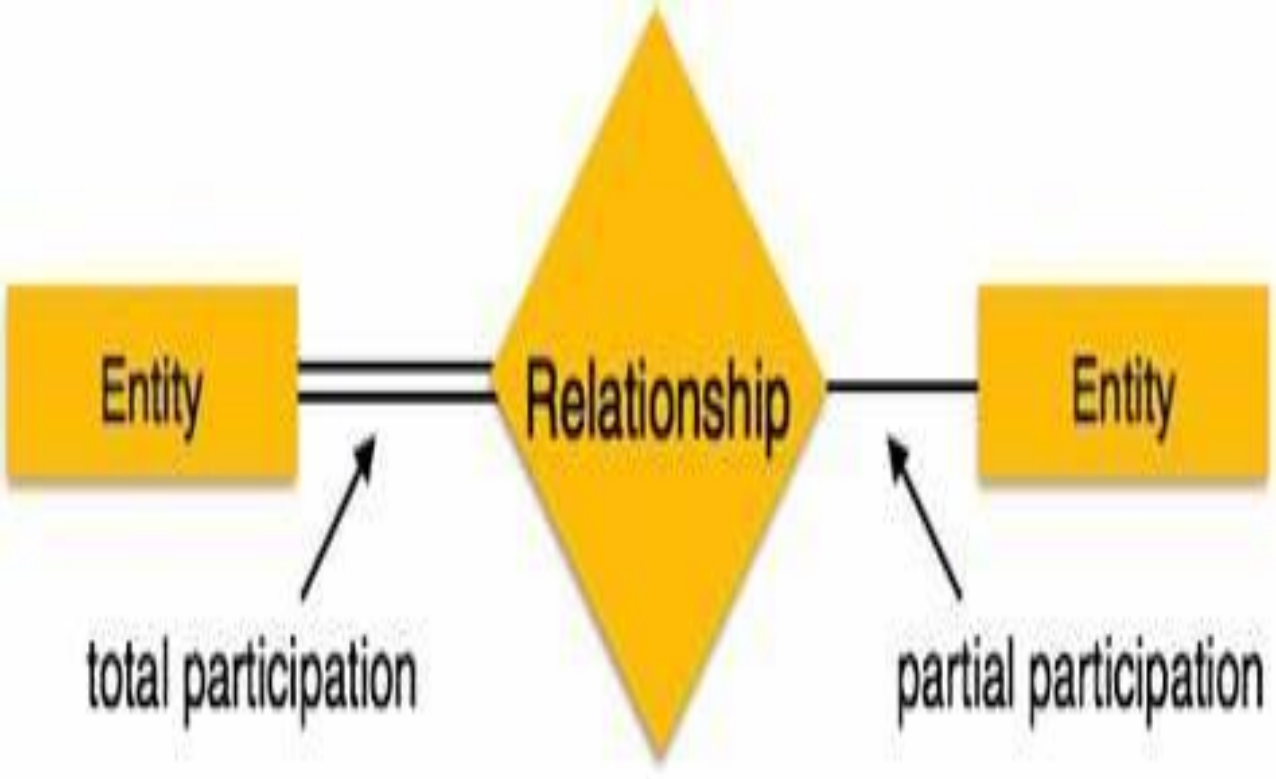
Participation Constraints

- 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**.

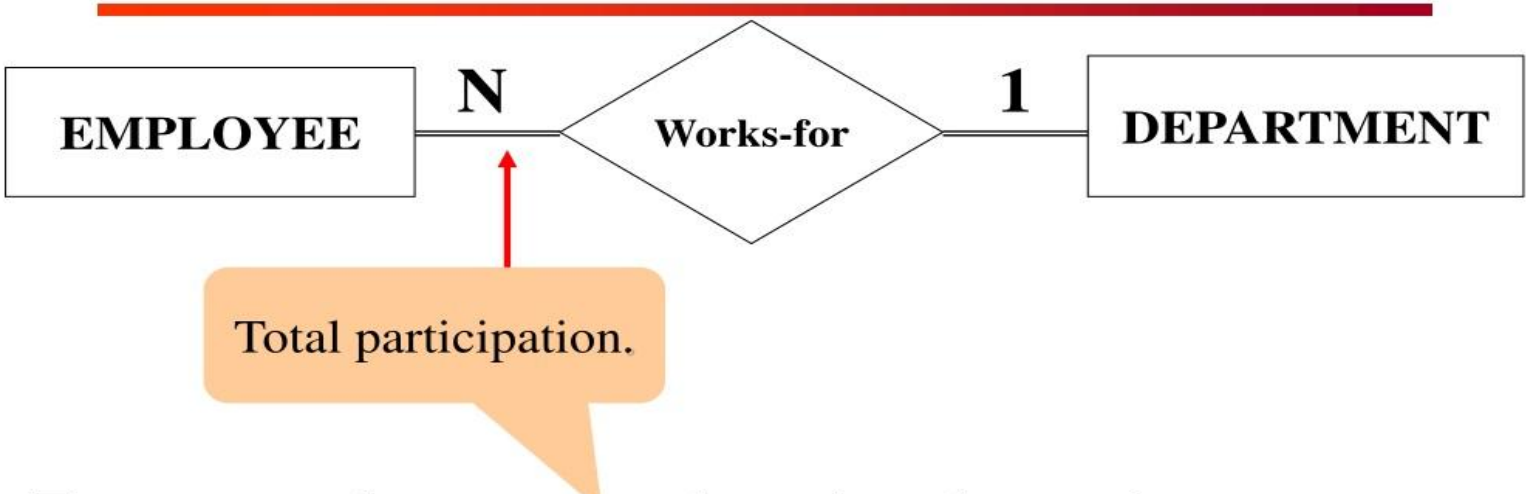
There are **two types of participation constraints**

1. Total participation constraint.
2. Partial participation constraint.

Representation



Total participation



Every employee **must** be related to a department via WORKS-FOR relationship**.** A department **must have** at least one employee.

Total & Partial participation

A professor may manage a department (*partial participation*), but a department must be managed by a professor (*total participation*).

Module – 1 Introduction to Databases

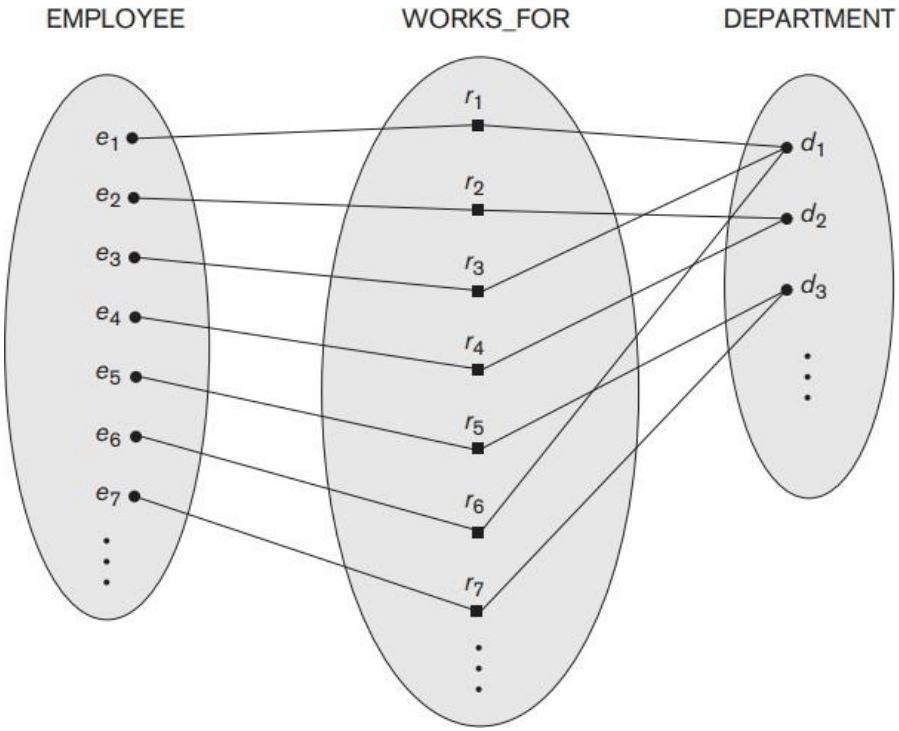


Figure: total participation

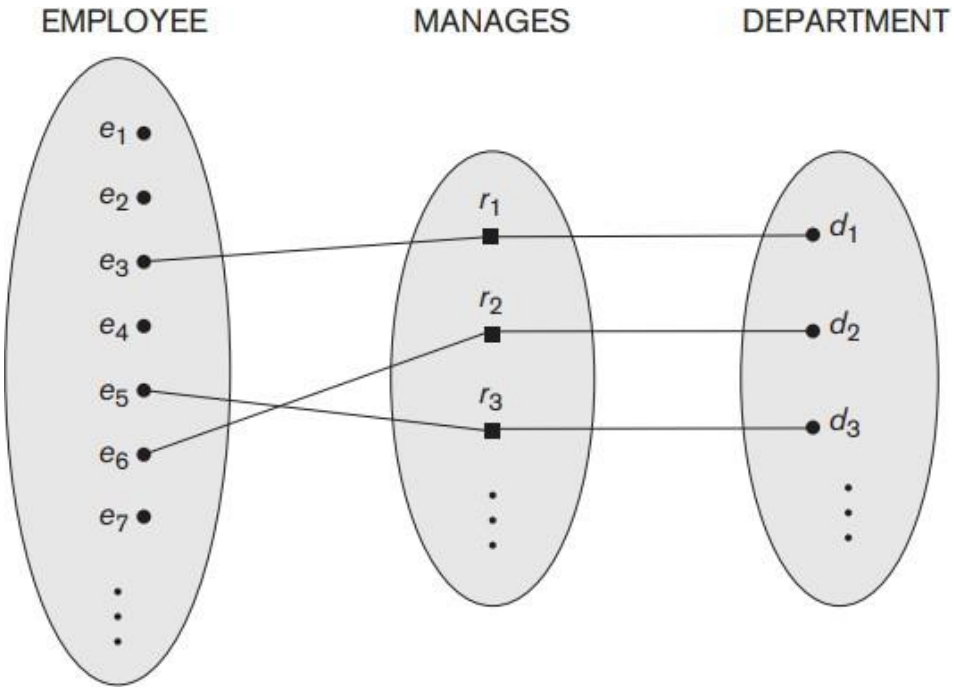


Figure: partial participation

Module – 1 Introduction to Databases

Relationship Types, Relationship Sets, Roles, and Structural Constraints

- The cardinality ratio and participation constraints, together referred to as the **structural constraints** of a relationship type.

NOTE:

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

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.

Module – 1 Introduction to Databases

Weak Entity Types

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

In entity-relationship (ER) diagrams, a weak entity is an entity that cannot be uniquely identified by its own attributes alone; it depends on the existence of another entity, called the "owner" or "parent" entity.

- 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.

Module – 1 Introduction to Databases

- The partial key attribute is underlined with a dashed or dotted line.

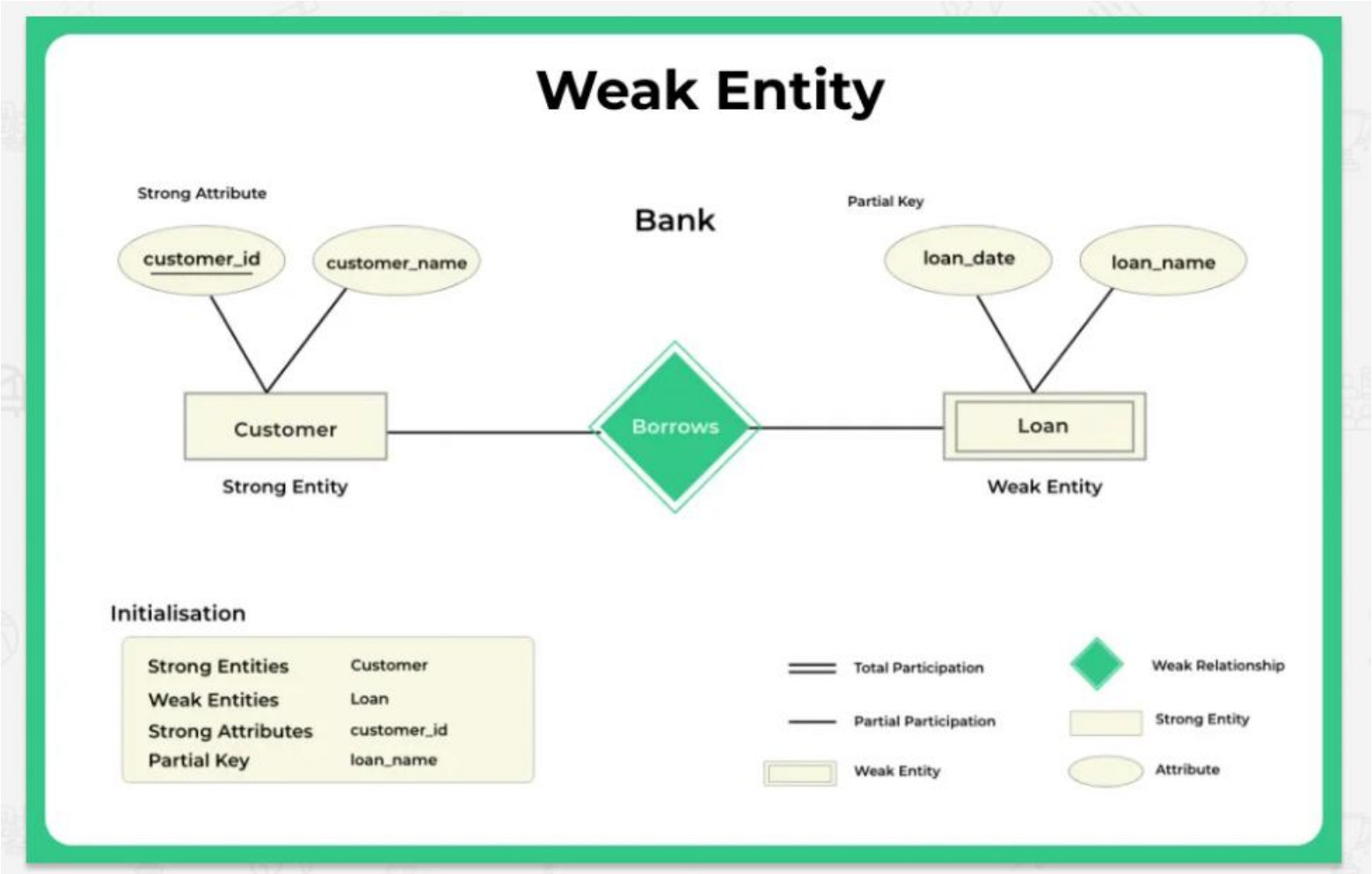
Weak Entity Types

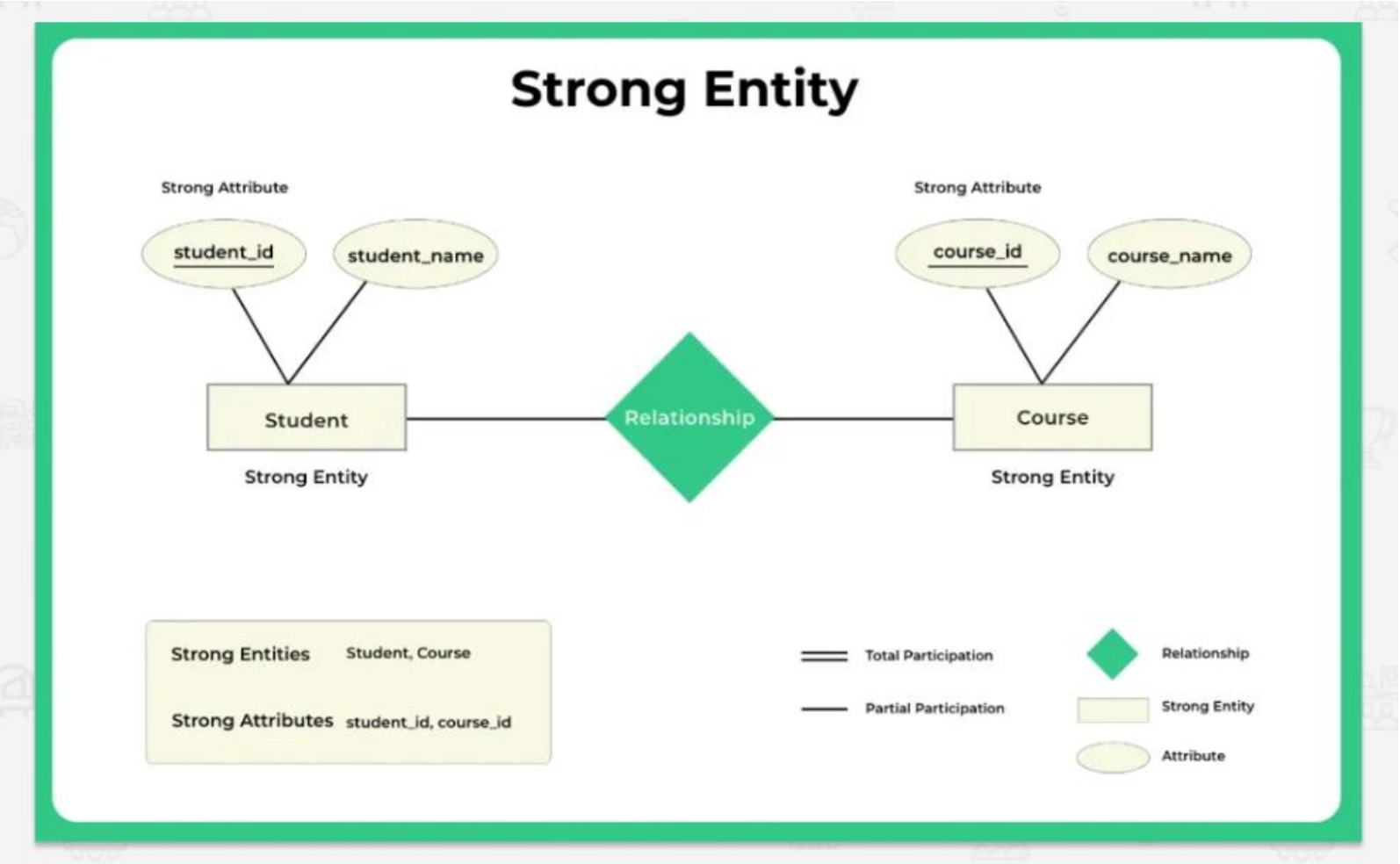
- A weak entity is an entity set that does not have sufficient attributes for Unique Identification of its records.

Example 1 – A loan entity can not be created for a customer if the customer doesn't exist

Example 2 – A dependents list entity can not be created if the employee doesn't exist

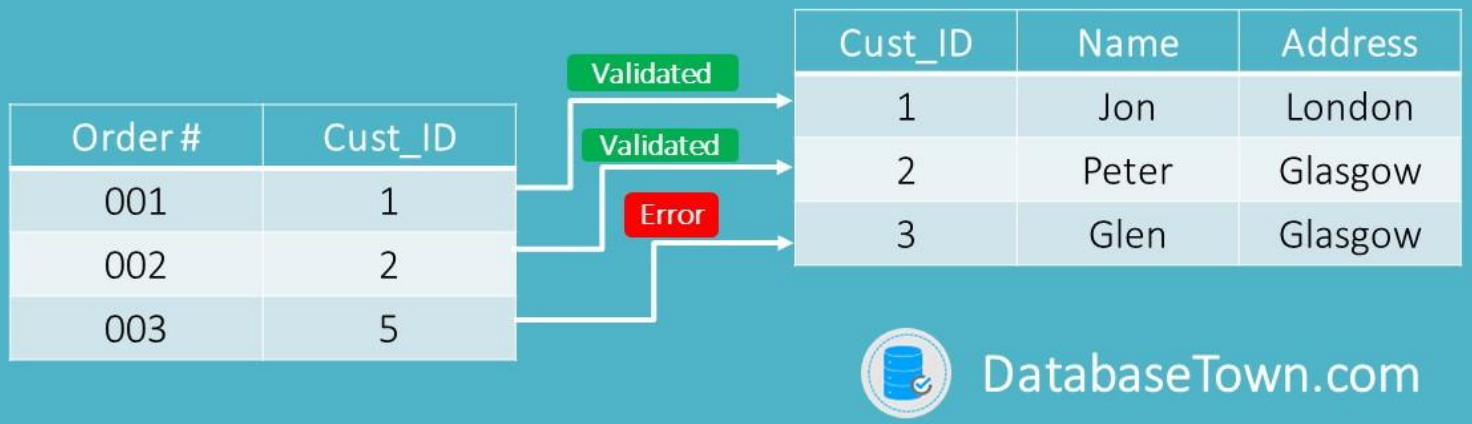
- A **double rectangle** is used for representing a **weak entity set**
- The **double diamond** symbol is used for representing the **relationship between a strong entity and a weak entity** which is known as identifying relationship

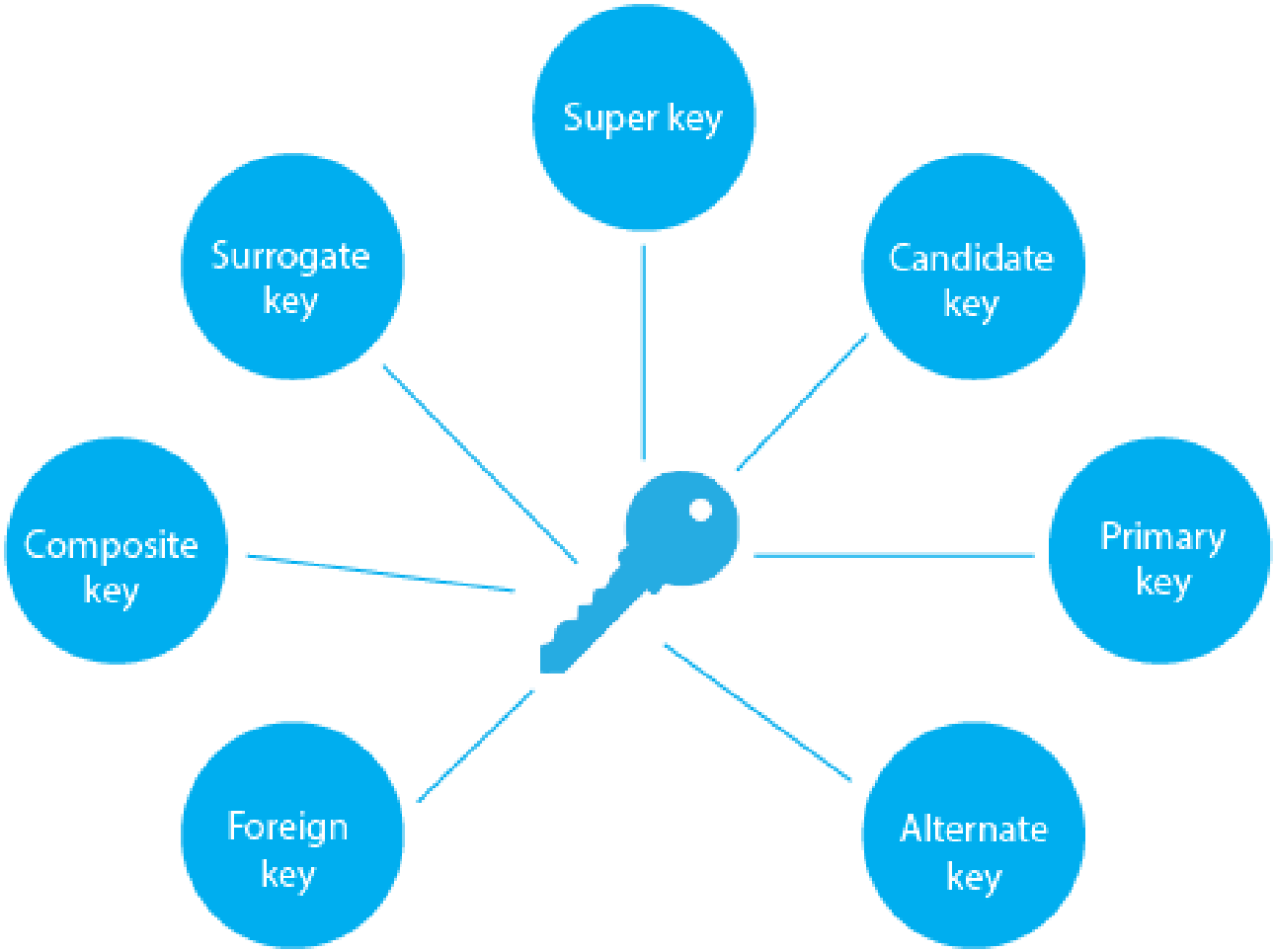

Weak Entity
Strong Attribute
customer_id
customer_name
Bank
Partial Key
loan_date
loan_name
Customer
Borrows
Loan
Strong Entity
Weak Entity
Initialisation
Strong Entities
Customer
Weak Entities
Loan
Strong Attributes
customer_id
Partial Key
loan_name
Total Participation
Partial Participation
Weak Entity
Weak Relationship
Strong Entity
Attribute


Strong Entity
Strong Attribute
student_id
student_name
Strong Attribute
course_id
course_name
Student
Relationship
Course
Strong Entity
Strong Entity
Strong Entities Student, Course
Strong Attributes student_id, course_id
Total Participation
Partial Participation
Relationship
Strong Entity
Attribute

What is Referential Integrity?

Referential integrity is a property of a database that ensures the consistency and correctness of data in a table. It is typically enforced through the use of foreign keys, which are fields in a table that reference the primary key of another table.




Super key
Surrogate key
Candidate key
Composite key
Primary key
Foreign key
Alternate key

A superkey is a group of single or multiple keys which identifies rows in a table. A Super key may have additional attributes that are not needed for unique identification.

Example:

EmpSSN	EmpNum	Empname
9812345098	AB05	Shown
9876512345	AB06	Roslyn
199937890	AB07	James

PRIMARY KEY in DBMS is a column or group of columns in a table that uniquely identify every row in that table. The Primary Key can't be a duplicate meaning the same value can't appear more than once in the table. A table cannot have more than one primary key.

Rules for defining Primary key:

- Two rows can't have the same primary key value
- It must for every row to have a primary key value.
- The primary key field cannot be null.
- The value in a primary key column can never be modified or updated if any foreign key refers to that primary key.

Example:

In the following example, `StudID` is a Primary Key.

StudID	Roll No	First Name	LastName	Email
1	11	Tom	Price	abc@gmail.com
2	12	Nick	Wright	xyz@gmail.com

ALTERNATE KEYS is a column or group of columns in a table that uniquely identify every row in that table. A table can have multiple choices for a primary key but only one can be set as the primary key. All the keys which are not primary key are called an Alternate Key.

Example:

In this table, StudID, Roll No, Email are qualified to become a primary key. But since StudID is the primary key, Roll No, Email becomes the alternative key.

StudID	Roll No	First Name	LastName	Email
1	11	Tom	Price	abc@gmail.com
2	12	Nick	Wright	xyz@gmail.com
3	13	Dana	Natan	mno@yahoo.com

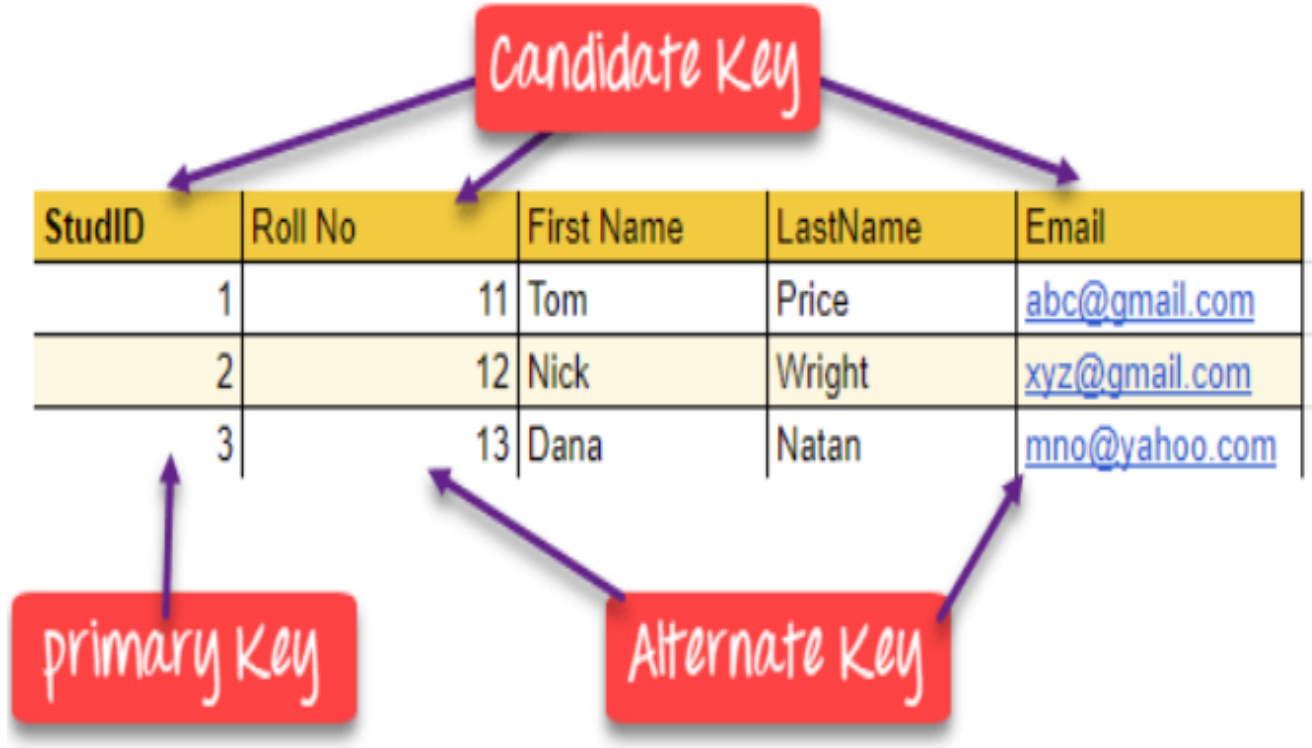
CANDIDATE KEY in SQL is a set of attributes that uniquely identify tuples in a table. Candidate Key is a super key with no repeated attributes. The Primary key should be selected from the candidate keys. Every table must have at least a single candidate key. A table can have multiple candidate keys but only a single primary key.

Properties of Candidate key:

- It must contain unique values
- Candidate key in SQL may have multiple attributes
- Must not contain null values
- It should contain minimum fields to ensure uniqueness
- Uniquely identify each record in a table

Candidate key Example: In the given table Stud ID, Roll No, and email are candidate keys which help us to uniquely identify the student record in the table.

StudID	Roll No	First Name	LastName	Email
1	11	Tom	Price	abc@gmail.com
2	12	Nick	Wright	xyz@gmail.com



Candidate Key in DBMS

FOREIGN KEY is a column that creates a relationship between two tables. The purpose of Foreign keys is to maintain data integrity and allow navigation between two different instances of an entity. It acts as a cross-reference between two tables as it references the primary key of another table.

Example:

DeptCode	DeptName
001	Science
002	English
005	Computer

Teacher ID	Fname	Lname
B002	David	Warner
B017	Sara	Joseph
B009	Mike	Brunton

In this key in dbms example, we have two table, teach and department in a school. However, there is no way to see which search work in which department.

In this table, adding the foreign key in Deptcode to the Teacher name, we can create a relationship between the two tables.

Teacher ID	DeptCode	Fname	Lname
B002	002	David	Warner
B017	002	Sara	Joseph
B009	001	Mike	Brunton

This concept is also known as Referential Integrity.

What is the Compound key?

COMPOUND KEY has two or more attributes that allow you to uniquely recognize a specific record. It is possible that each column may not be unique by itself within the database. However, when combined with the other column or columns the combination of composite keys become unique. The purpose of the compound key in database is to uniquely identify each record in the table.

Example:

OrderNo	PorductID	Product Name	Quantity
B005	JAP102459	Mouse	5
B005	DKT321573	USB	10
B005	OMG446789	LCD Monitor	20
B004	DKT321573	USB	15
B002	OMG446789	Laser Printer	3

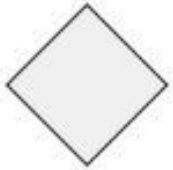
SURROGATE KEYS is An artificial key which aims to uniquely identify each record is called a surrogate key. This kind of partial key in dbms is unique because it is created when you don't have any natural primary key. They do not lend any meaning to the data in the table. Surrogate key in DBMS is usually an integer. A surrogate key is a value generated right before the record is inserted into a table.

Fname	Lastname	Start Time	End Time
Anne	Smith	09:00	18:00
Jack	Francis	08:00	17:00
Anna	McLean	11:00	20:00
Shown	Willam	14:00	23:00

Above, given example, shown shift timings of the different employee. In this example, a surrogate key is needed to uniquely identify each employee.

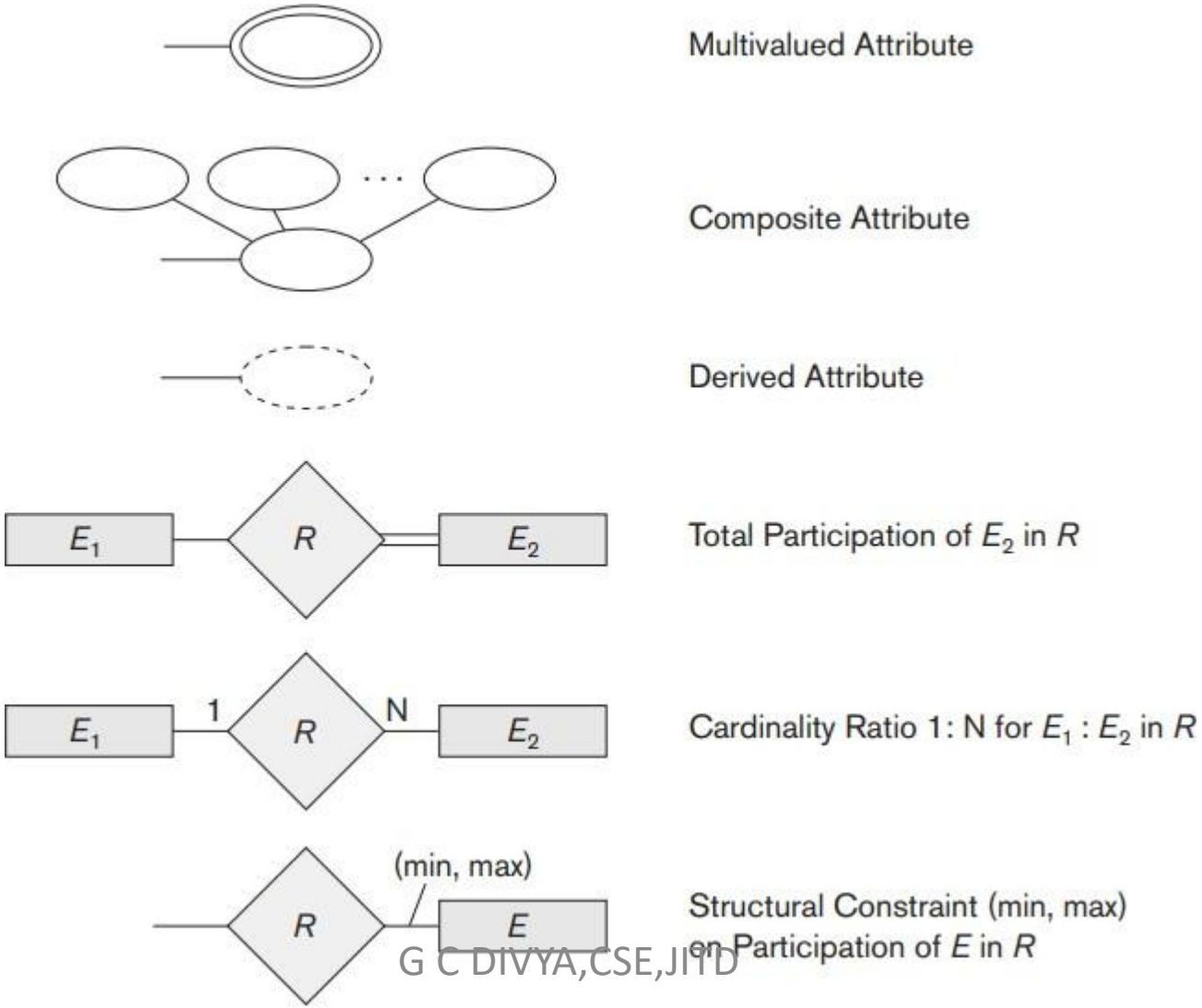
Module – 1 Introduction to Databases

Summary of the different notations for ER diagrams

Symbol	Meaning
	Entity
	Weak Entity
	Relationship
	Indentifying Relationship
	Attribute
	Key Attribute

Module – 1 Introduction to Databases

Summary of the different notations for ER diagrams



Module – 1 Introduction to Databases

ER diagram of company database

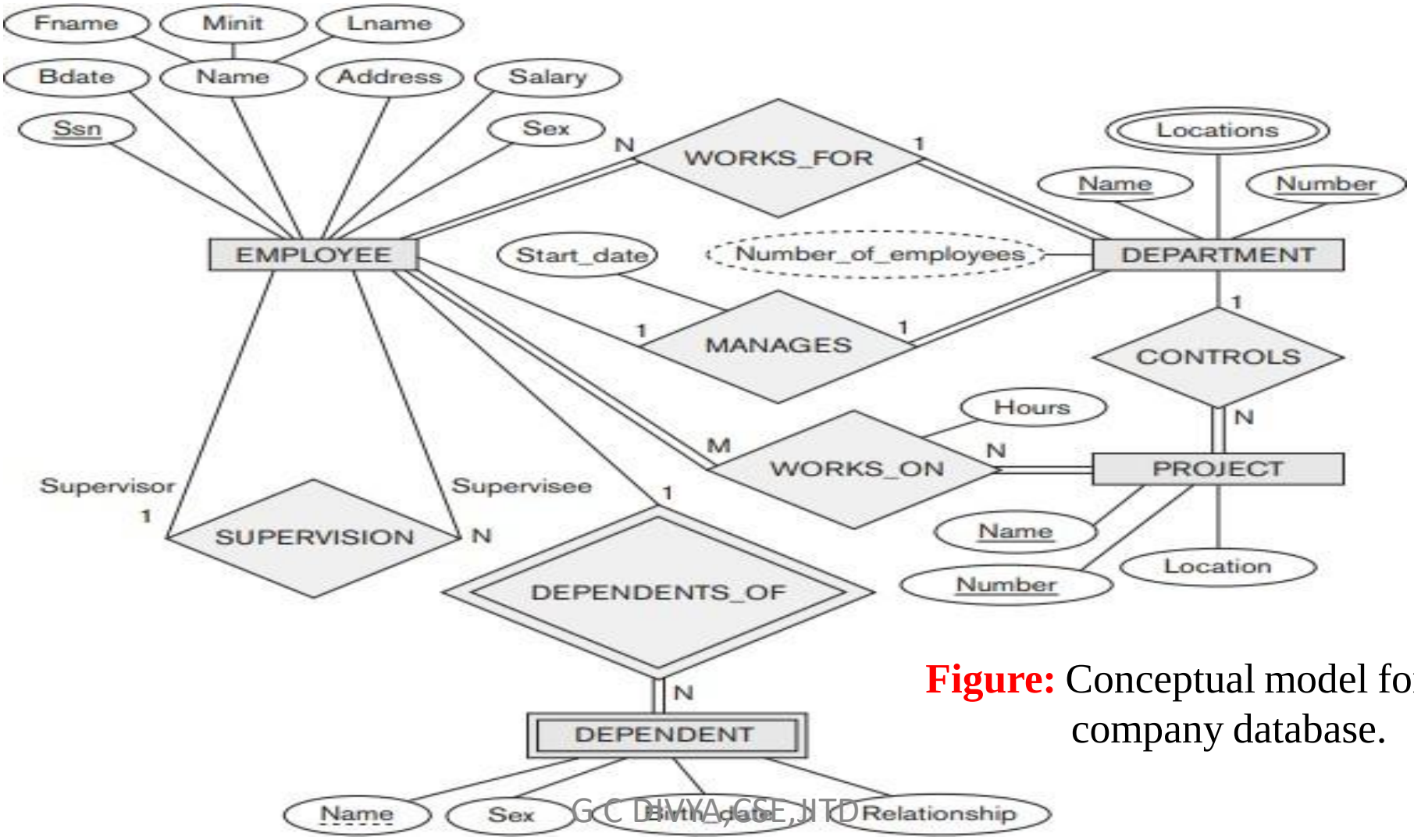


Figure: Conceptual model for company database.

Module – 1 Introduction to Databases

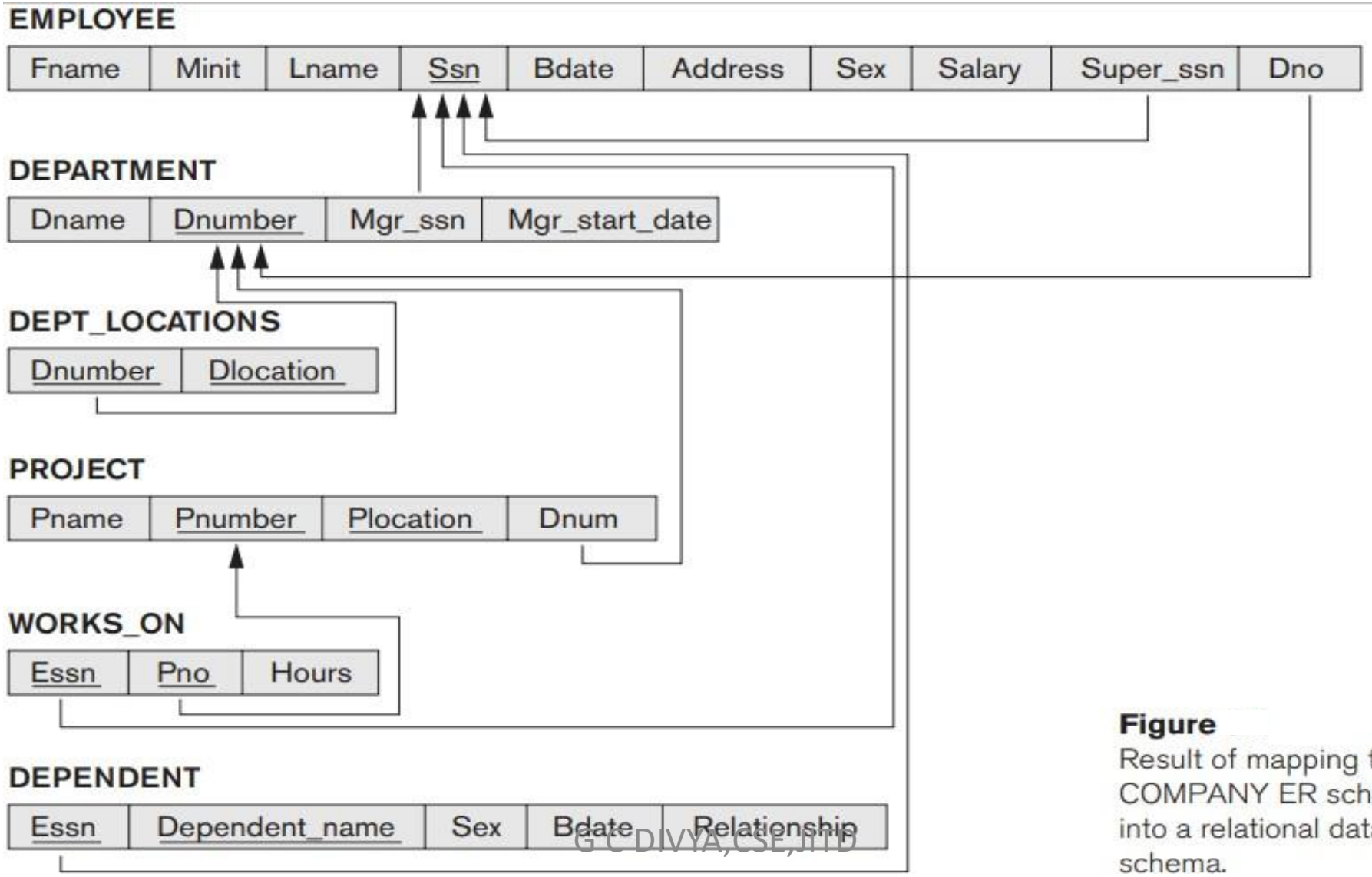


Figure
Result of mapping the COMPANY ER schema into a relational database schema.

Module – 1 Introduction to Databases

Alternative Notations for 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 $\max \geq 1$.
- In this method, **min = 0** implies partial participation, whereas **min > 0** implies total participation.
- **The figure given below shows** the ER diagrams for the company schema, with structural constraints specified using (min, max) notation and role names.

Module – 1 Introduction to Databases

Alternative Notations for ER Diagrams

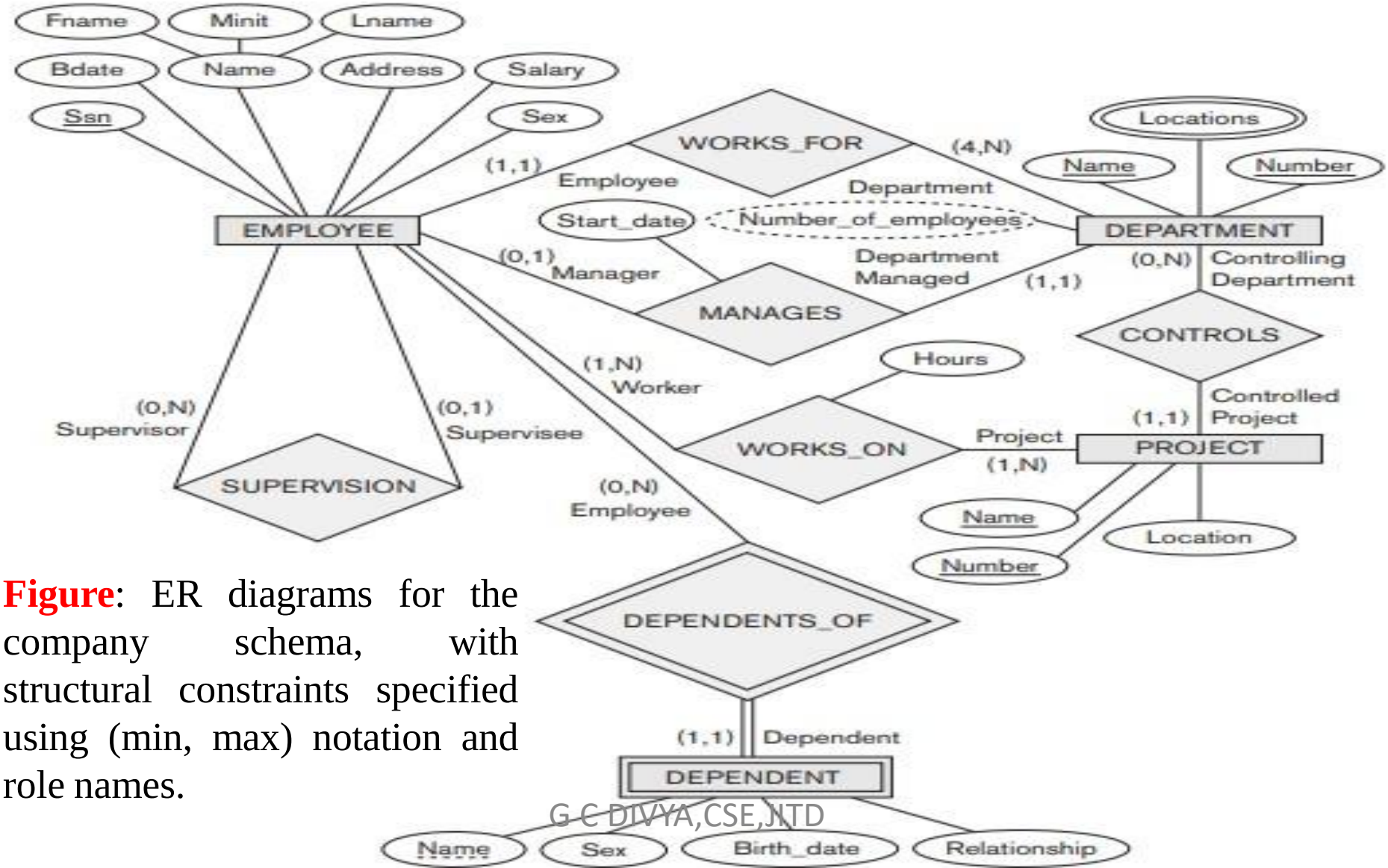


Figure: ER diagrams for the company schema, with structural constraints specified using (min, max) notation and role names.

Module – 1 Introduction to Databases

ER Diagram Naming Conventions

When designing a database schema, you are suggested to use the following naming conventions.

One should choose names that convey, as much as possible, the meanings of the different constructs(entity type) 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.

The example ER diagrams used here, used 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.

Module – 1 Introduction to Databases

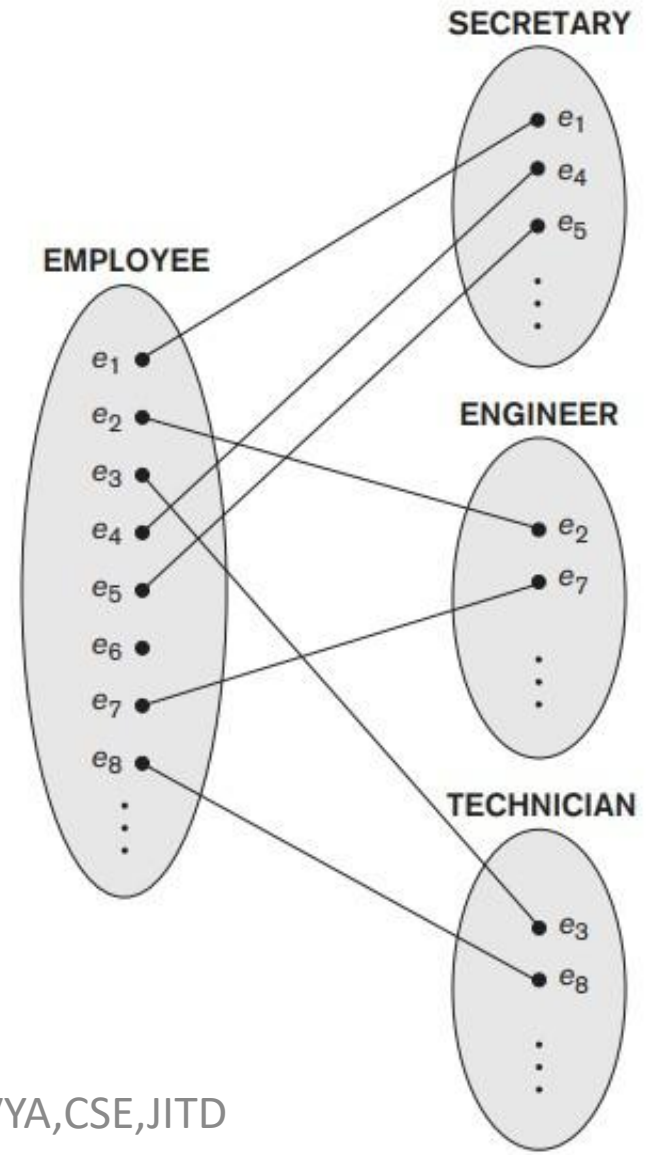
Specialization

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

For example, the set of subclasses {SECRETARY, ENGINEER, TECHNICIAN} is a specialization of the superclass **EMPLOYEE** that distinguishes among employee entities based on the **job type** of each employee.

Module – 1 Introduction to Databases

Figure: Instances of a specialization.



Module – 1 Introduction to Databases

Introduction

Generalization

We can think generalization as a reverse process of abstraction in which we suppress the differences among several entity types, identify their common features, and generalize them into a single superclass of which the original entity types are special subclasses.

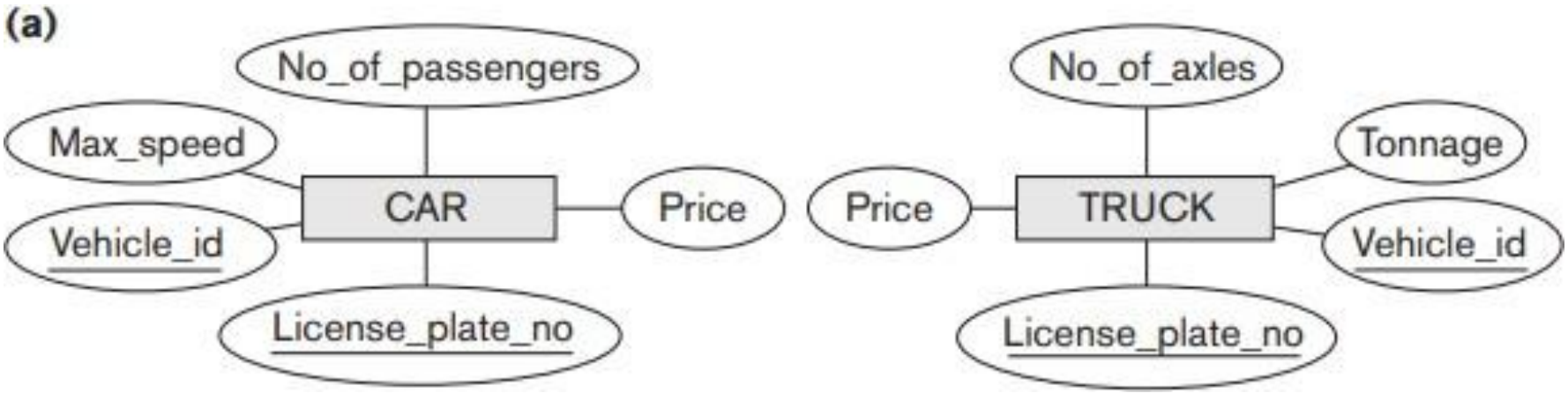


Figure: (a) Two entity types, CAR and TRUCK

Module – 1 Introduction to Databases

Introduction

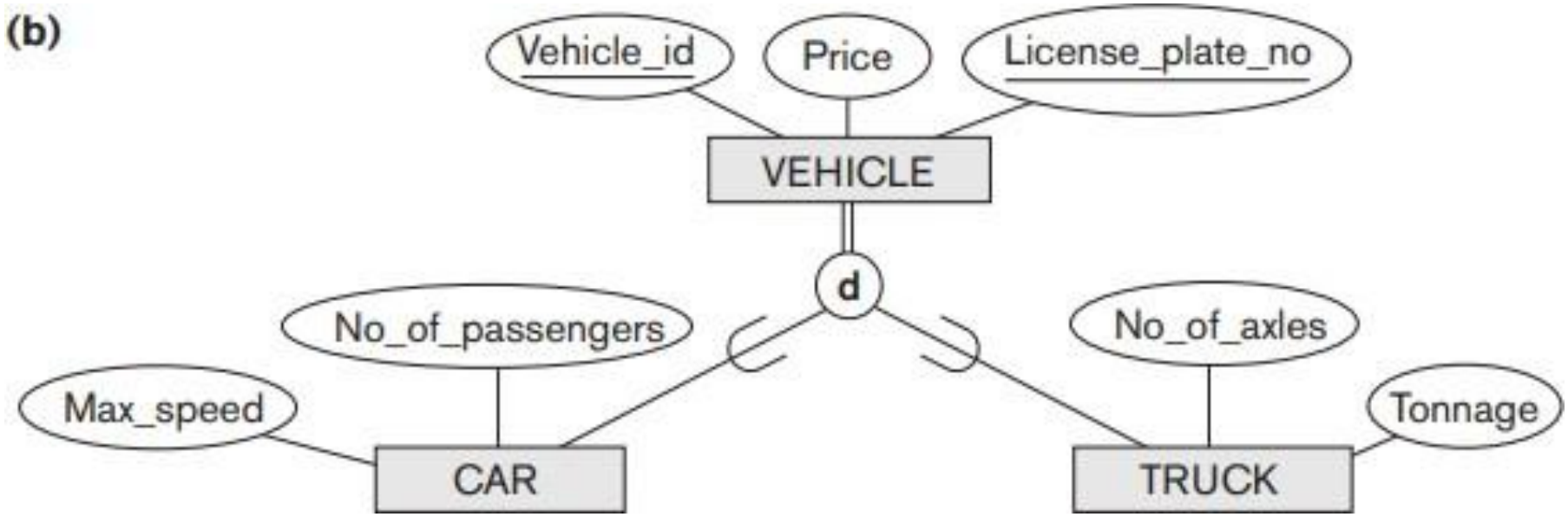


Figure: (b) Generalizing CAR and TRUCK into the superclass VEHICLE.

Note: Functional dependency and a formal constraint among attributes that is the **main tool for formally measuring the appropriateness** of attribute groupings into relation schemas.

Module – 1 Introduction to Databases

Source

Text books

1. Fundamentals of Database Systems, Ramez Elmasri and Shamkant B. Navathe, 7th Edition, 2017, Pearson.
2. Database management systems, Ramakrishnan, and Gehrke, 3rd Edition, 2014, McGraw Hill