

Module 2

Chapter 1 – Data Models

Introduction to Relational Model:

- The relational model uses a collection of tables to represent both data and the relationships among those data.
- Each table has multiple columns, and each column has a unique name.
- Tables are also known as relations.
- The relational model is an example of a record-based model.
- Record-based models are so named because the database is structured in fixed-format records of several types.
- Each table contains records of a particular type.
- Each record type defines a fixed number of fields, or attributes.
- The columns of the table correspond to the attributes of the record type.

What is the Relational Model?

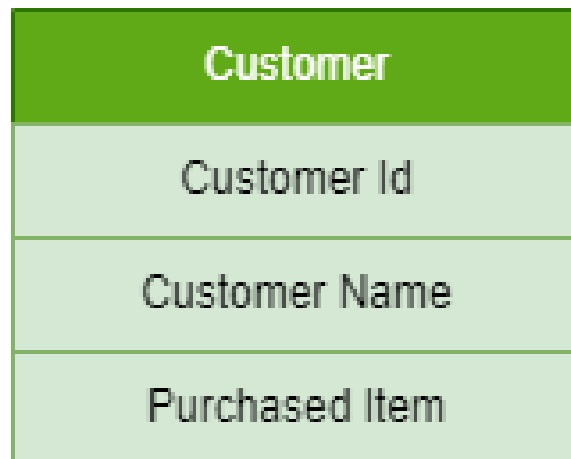
- The relational model represents how data is stored in Relational Databases. A relational database consists of a collection of tables, each of which is assigned a unique name.
- Consider a relation STUDENT with attributes ROLL_NO, NAME, ADDRESS, PHONE, and AGE shown in the table.

Table Student

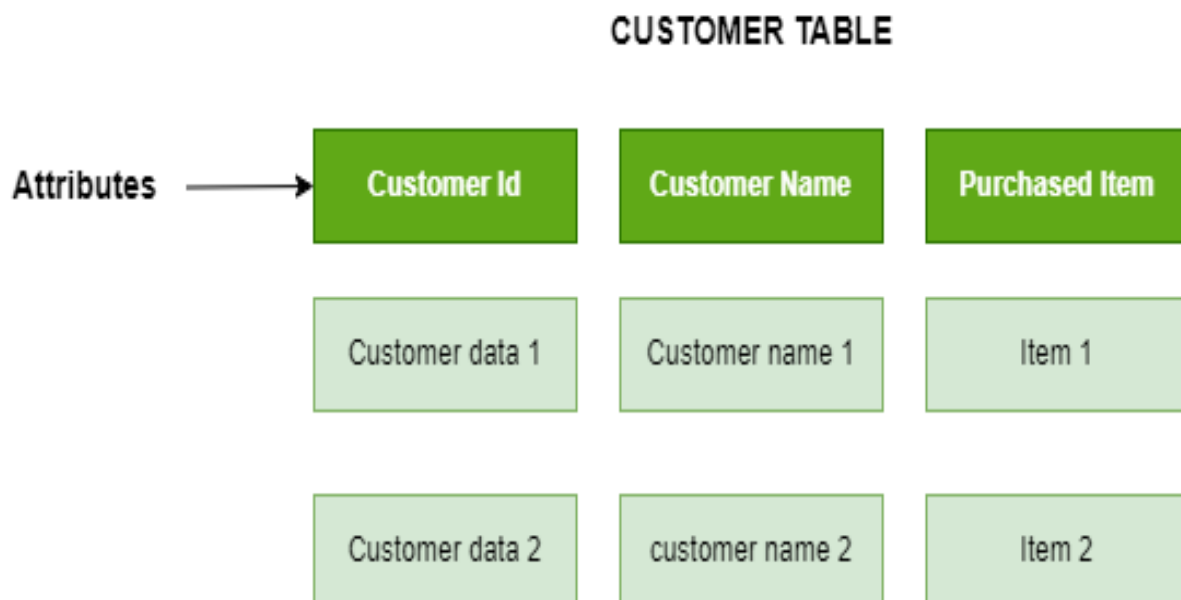
ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
2	RAMESH	GURGAON	9652431543	18
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	DELHI		18

What is Schema?

- The Skeleton of the database is created by the attributes and this skeleton is named Schema.
- Schema mentions the logical constraints like table, primary key, etc.
- The schema does not represent the data type of the attributes.
- Details of a customer:



- Schema of a customer



Database Schema:

- A database schema is a logical representation of data that shows how the data in a database should be stored logically. It shows how the data is organized and the relationship between the tables.

- Database schema contains table, field, views and relation between different keys like [primary key](#), [foreign key](#).
- Database schema provides the organization of data and the relationship between the stored data.
- Database schema defines a set of guidelines that control the database along with that it provides information about the way of accessing and modifying the data.

Types of Database Schemas:

■ There are 3 types of database schema:

1. Physical Database Schema

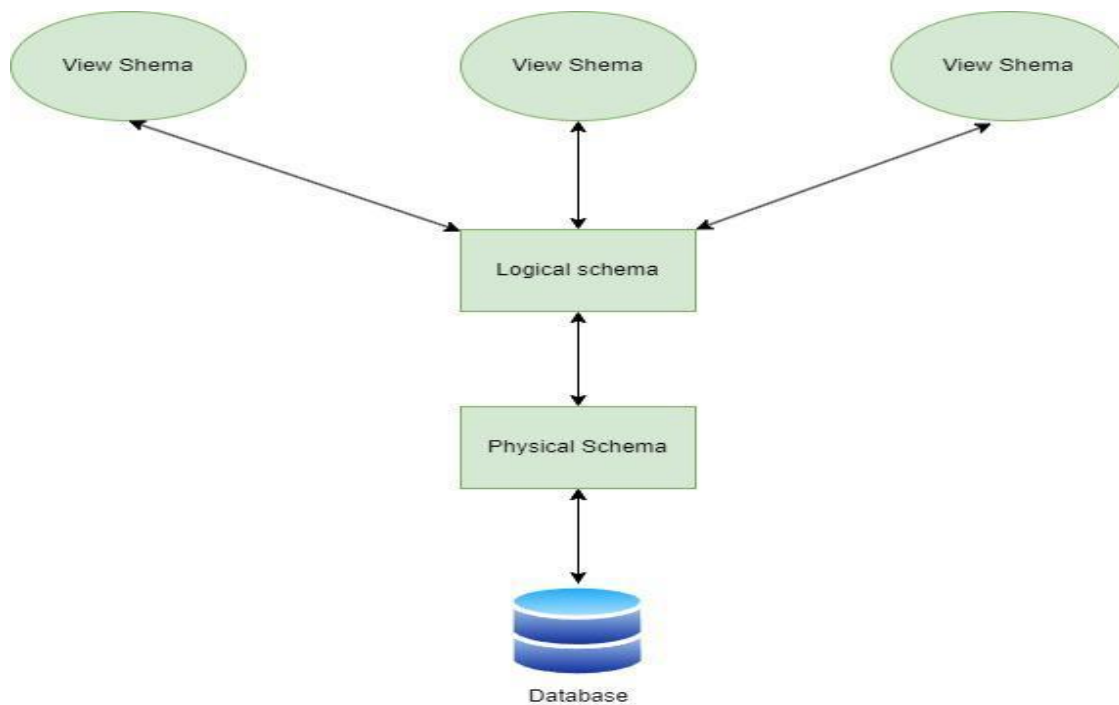
- A Physical schema defines, how the data or information is stored physically in the storage systems in the form of files & indices.
- The actual code or syntax needed to create the structure of a database, when we design a database at a physical level, it's called physical schema.
- The Database administrator chooses where and how to store the data in the different blocks of storage.

2. Logical Database Schema

- A logical database schema defines all the logical constraints that need to be applied to the stored data, and also describes tables, views, entity relationships, and integrity constraints.
- The Logical schema describes how the data is stored in the form of tables & how the attributes of a table are connected.
- Using ER modelling the relationship between the components of the data is maintained.

■ View Database Schema

- It is a view level design which is able to define the interaction between end-user and database.
- User is able to interact with the database with the help of the interface without knowing much about the stored mechanism of data in database.



Keys:

- Keys are one of the basic requirements of a relational database model. It is widely used to identify the tuples(rows) uniquely in the table. We also use keys to set up relations amongst various columns and tables of a relational database.
- Different Types of Database Keys
 - Candidate Key
 - Primary Key
 - Super Key
 - Alternate Key
 - Foreign Key
 - Composite Key

1. Candidate Key

- The minimal set of attributes that can uniquely identify a tuple is known as a candidate key.
 - It is a minimal super key.
 - It is a super key with no repeated data is called a candidate key.
 - The minimal set of attributes that can uniquely identify a record.
 - It must contain unique values.

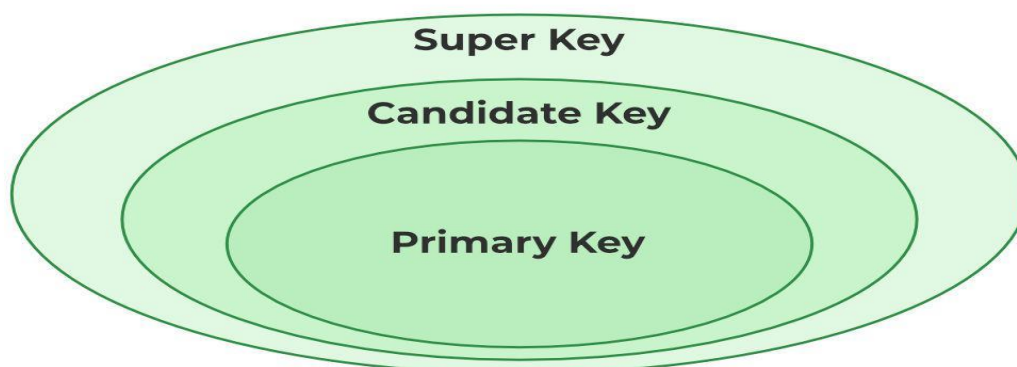
- It can contain NULL values.
- Every table must have at least a single candidate key.
- A table can have multiple candidate keys but only one primary key.
- The value of the Candidate Key is unique and may be null for a tuple.
- There can be more than one candidate key in a relationship.
- Example, STUD_NO, COURSE_NO in STUDENT relation.

2. Primary Key

- There can be more than one candidate key in relation out of which one can be chosen as the primary key.
- It is a unique key.
- It can identify only one tuple (a record) at a time.
- It has no duplicate values, it has unique values.
- It cannot be NULL.
- Primary keys are not necessarily to be a single column; more than one column can also be a primary key for a table.
- Example: Student Table->Student(stud_no, name,addr), stud_no is a primary key.

3. Super Key

- The set of attributes that can uniquely identify a tuple is known as Super Key. For Example, STUD_NO, (STUD_NO, STUD_NAME), etc.
- A super key is a group of single or multiple keys that identifies rows in a table. It supports NULL values.
- Adding zero or more attributes to the candidate key generates the super key.
- A candidate key is a super key but vice versa is not true.
- Super Key values may also be NULL.



4. Alternate Key

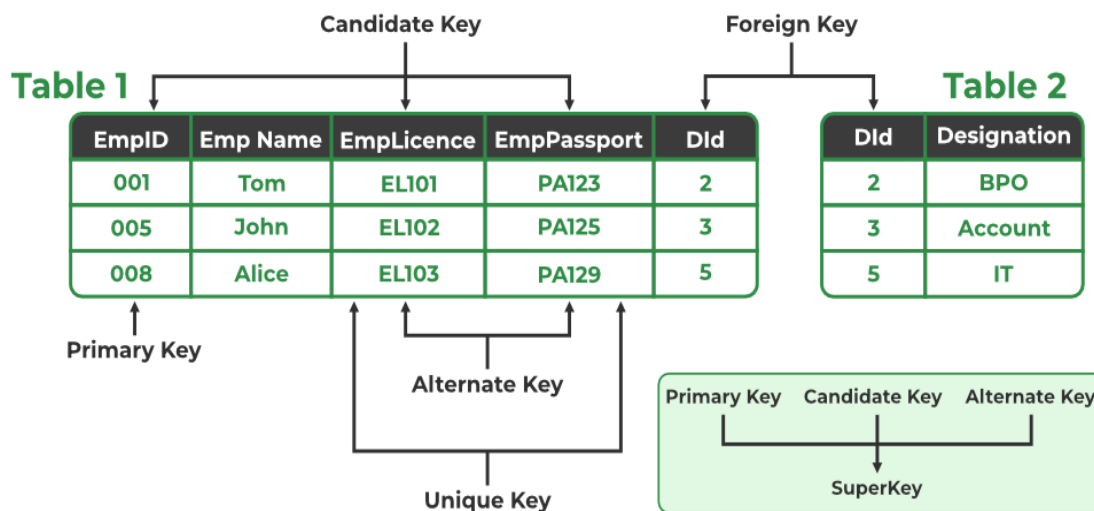
- The candidate key other than the primary key is called an [alternate key](#).
- All the keys which are not primary keys are called alternate keys.
- It is a secondary key.
- It contains two or more fields to identify two or more records.
- These values are repeated.
- Eg:- SNAME, and ADDRESS is Alternate keys

5. Foreign Key

- If an attribute can only take the values which are present as values of some other attribute, it will be a [foreign key](#) to the attribute to which it refers. The relation which is being referenced is called referenced relation and the corresponding attribute is called referenced attribute. The referenced attribute of the referenced relation should be the primary key to it.
- It is a key it acts as a primary key in one table and it acts as secondary key in another table.
- It combines two or more relations (tables) at a time.
- They act as a cross-reference between the tables.
- Example: DNO is a primary key in the DEPT table and a non-key in EMP

6. Composite Key

- Sometimes, a table might not have a single column/attribute that uniquely identifies all the records of a table. To uniquely identify rows of a table, a combination of two or more columns/attributes can be used. So, we need to find the optimal set of attributes that can uniquely identify rows in a table.
- It acts as a primary key if there is no primary key in a table
- Two or more attributes are used together to make a [composite key](#).
- Different combinations of attributes may give different accuracy in terms of identifying the rows uniquely.
- Example: Fullname + DOB
together to access the details of the student.



Relational Query Language:

- **Relational Query Language** is used by the user to communicate with the database user requests for the information from the database.
- Relational algebra breaks the user requests and instructs the DBMS to execute the requests. It is the language by which the user communicates with the database.
- Relational Algebra is a procedural query language. Relational algebra mainly provides a theoretical foundation for relational databases and [SQL](#). The main purpose of using Relational Algebra is to define operators that transform one or more input relations into an output relation.

Introduction: Conceptual Data Modeling

- Conceptual modeling is a very important phase in designing a successful database application.
- The term database application refers to a particular database and the associated programs that implement the database queries and updates.
- Here, we follow the traditional approach of concentrating on the database structures and constraints during conceptual database design

Conceptual Design:

- We present the modeling concepts of the entity–relationship (ER) model, which is a popular high-level conceptual data model.
- We also present the diagrammatic notation associated with the ER model, known as ER diagrams.

Conceptual Database Using High-Level Conceptual Data Models for Database Design:

- Figure 3.1 shows a simplified overview of the database design process.
- The first step shown is requirements collection and analysis. During this step, the database designers interview prospective database users to understand and document their data requirements. The result of this step is a concisely written set of users requirements. These requirements should be specified in as detailed and complete a form as possible.
- In parallel with specifying the data requirements, it is useful to specify the known functional requirements of the application. These consist of the user defined operations (or transactions) that will be applied to the database, including both retrievals and updates.
- In software design, it is common to use data flow diagrams, sequence diagrams, scenarios, and other techniques to specify functional requirements.
- Once the requirements have been collected and analyzed, the next step is to create a conceptual schema for the database, using a high-level conceptual data model. This step is called conceptual design.
- The conceptual schema is a concise description of the data requirements of the users and includes detailed descriptions of the entity types, relationships, and constraints; these are expressed using the concepts provided by the high-level data model
- The 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.
- This approach enables database designers to concentrate on specifying the properties of the data, without being concerned with storage and implementation details, which makes it is easier to create a good conceptual database design.
- 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; its result is a database schema in the implementation data model of the DBMS.
- The last step is the physical design phase, during which the internal storage structures, file organizations, indexes, access paths, and physical design parameters for the database files are specified.
- In parallel with these activities, application programs are designed and implemented as database transactions corresponding to the high-level transaction specifications.

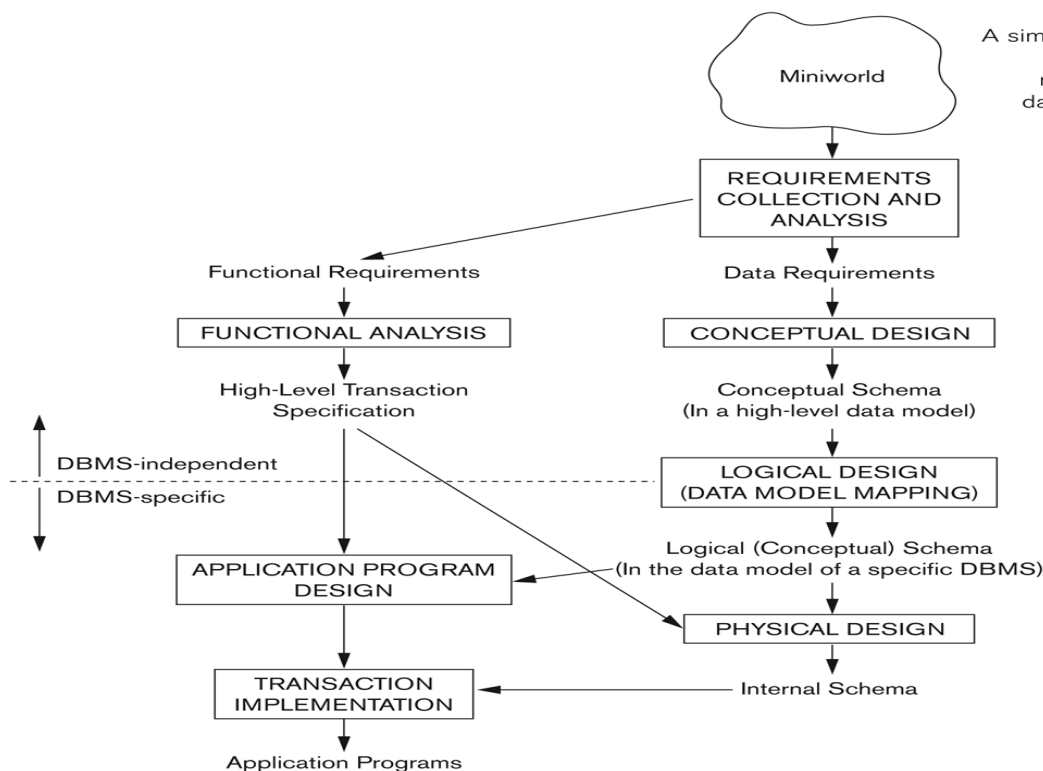


Figure 3.1
A simplified diagram to illustrate the main phases of database design.

ER Model Concepts:

■ Entities and Attributes

- Entities are specific objects or things in the mini-world that are represented in the database.

- For example the EMPLOYEE John Smith, the Research DEPARTMENT, the ProductX PROJECT

- Attributes are properties used to describe an entity.

- For example an EMPLOYEE entity may have the attributes Name, SSN, Address, Sex, BirthDate

- A specific entity will have a **value** for each of its attributes.

- For example a specific employee entity may have Name='John Smith', SSN='123456789', Address ='731, Fondren, Houston, TX', Sex='M', BirthDate='09-JAN-55'

- Each attribute has a *value set* (or data type) associated with it – e.g. integer, string, subrange, enumerated type, ...

Entities

- An entity is a "**thing**" or "**object**" in the real world.

- Entities are specific objects or things in the mini-world that are represented in the database.
- Entities are recorded in the database and must be distinguishable, i.e., easily recognized from the group.
- **For example:** A student, An employee, or bank a/c, etc. all are entities.

Entity Set:

- An entity set is a collection of similar types of entities that share the same attributes.
- **For example:** All students of a school are a entity set of Student entities.
- **It can be classified into two types:**

- **Strong Entity Set**

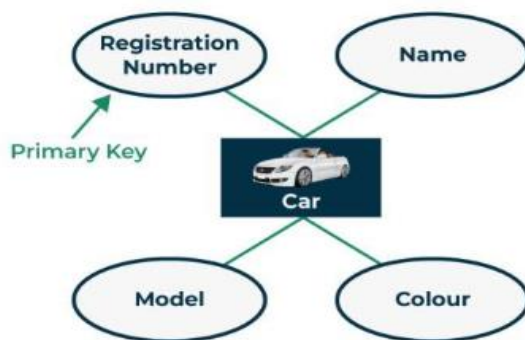
- **Weak Entity Set**

- **Strong Entity Set**

- Strong entity sets exist independently and each instance of a strong entity set has a unique primary key.

Example of Strong Entity includes:

- Car Registration Number
- Model
- Name etc.



ER Diagram of Strong Entity Set

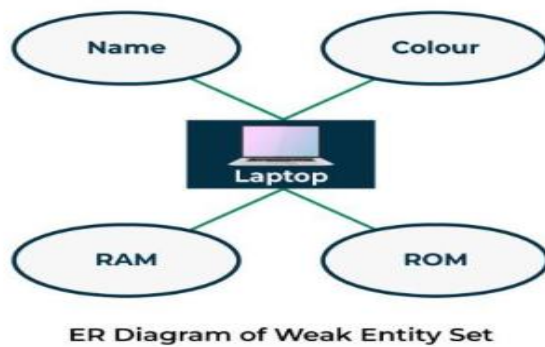
- **Weak Entity Set**

- A weak entity cannot exist on its own; it is dependent on a strong entity to identify it. A weak entity does not have a single primary key that uniquely identifies it; instead, it has a partial key.

- **Example of Weak Entity Set includes:**

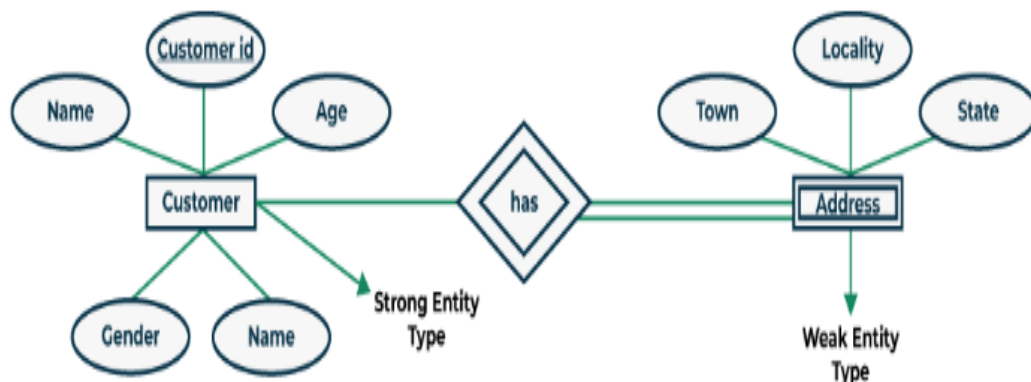
- Laptop Color

- RAM, etc.



Entity Types:

- **Strong Entity Types:** These are entities that exist independently and have a completely unique identifier.
- **Weak Entity Types:** These entities depend on another entity for his or her lifestyles and do not have a completely unique identifier on their own.
- The Example of Strong and Weak Entity Types in DMBS is:



- **Associative Entity Types:** These constitute relationships between or greater entities and might have attributes in their own.
- **Derived Entity Types:** These entities are derived from different entities through a system or calculation.
- **Multi-Valued Entity Types:** These entities will have more than one value for an characteristic.

Attribute:

- Attributes are properties used to describe an entity.
- For example an EMPLOYEE entity may have the attributes Name, SSN, Address, Sex, BirthDate

Types of Attributes

■ **Simple Attribute**

- Each entity has a single atomic value for the attribute.

For example, SSN or Sex.

■ **Composite Attribute**

- The attribute may be composed of several components. For example:
 - Address(Apt#, House#, Street, City, State, ZipCode, Country), or
 - Name(FirstName, MiddleName, LastName).
 - Composition may form a hierarchy where some components are themselves composite.

■ **Single – Valued Attribute** : The attribute have a single value for a particular entity.

For example: Age

■ **Multi-valued Attribute:**

- An entity may have multiple values for that attribute. For example, Color of a CAR or PreviousDegrees of a STUDENT.
 - Denoted as {Color} or {PreviousDegrees}.

■ **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. 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.

■ **NULL Values:** The meaning of the former type of NULL is not applicable, whereas the meaning of the latter is unknown. The unknown category of NULL can be further classified into two cases. The **first case** arises when it is known that the attribute value exists but is missing—for instance, if the Height attribute of a person is listed as NULL. The **second case** arises when it is not known whether the attribute value exists—for example, if the Home_phone attribute of a person is NULL.

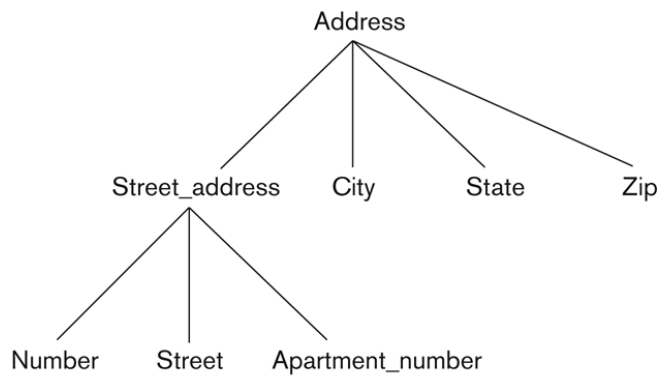
■ **Complex Attributes:** In general, composite and multi-valued attributes may be nested arbitrarily to any number of levels, although this is rare.

For example, PreviousDegrees of a STUDENT is a composite multi-valued attribute denoted by {PreviousDegrees (College, Year, Degree, Field)}

Multiple PreviousDegrees values can exist

Each has four subcomponent attributes:

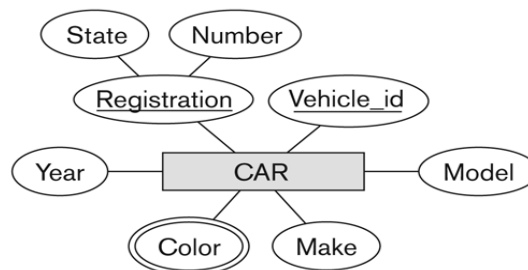
College, Year, Degree, Field

**Figure 3.4**

A hierarchy of composite attributes.

- **Key Attributes:** An attribute of an entity type for which each entity must have a unique value is called a key attribute of the entity type.
For example, SSN of EMPLOYEE.
A key attribute may be composite.
VehicleTagNumber is a key of the CAR entity type with components (Number, State).
- An entity type may have more than one key.
 - The CAR entity type may have two keys:
 - VehicleIdentificationNumber (popularly called VIN)
 - VehicleTagNumber (Number, State), aka license plate number.
- Each key is underlined

(a)

**Figure 3.7**

The CAR entity type with two key attributes, Registration and Vehicle_id. (a) ER diagram notation. (b) Entity set with three entities.

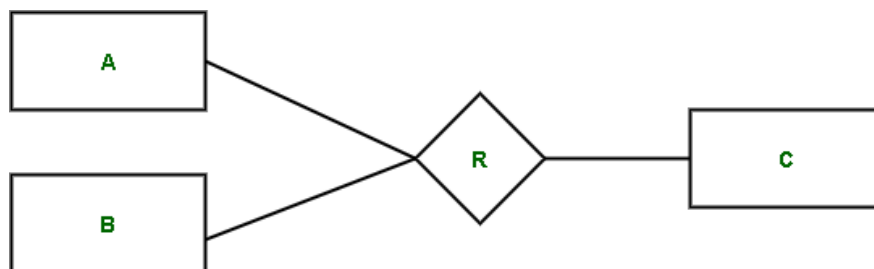
(b)

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})
⋮

Relationship :

- A relationship in a DBMS exists when the variable has a connection with the properties stored in different tables. Such relationships help the organization of entities intertwined with each other, ultimately enabling efficient data processing.
- They're exhibited usually via keys in a table, which is either columns or fields that specify a distinctive arrangement for each record.
- [Entity Relationship model \(ER model\)](#) contains entities and relationships. ER model enables us to know how these entities are associated with each other. Entities interact with other entities through associations or relationships.
- Example, Each employee *works for* one department but may *work on* several projects.
- **Diagrammatic Notation for Relationships :**
Relationship is represented by a diamond shaped box. Rectangle represents participating entities.



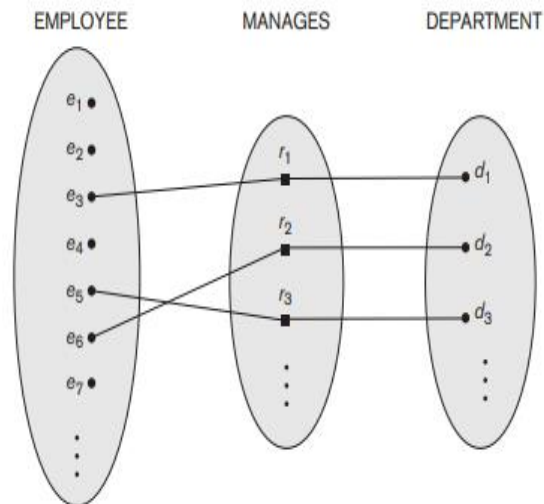
Types of Relationships in a Database:

1. One-to-One (1:1) Relationship

- In one to one relationships, a record is present in one table along with its corresponding existing relation, and the vacant relation among the records is present in another table.
- In another case, a person's employees' data consists of a record in the "personal details" table in a human resources database.

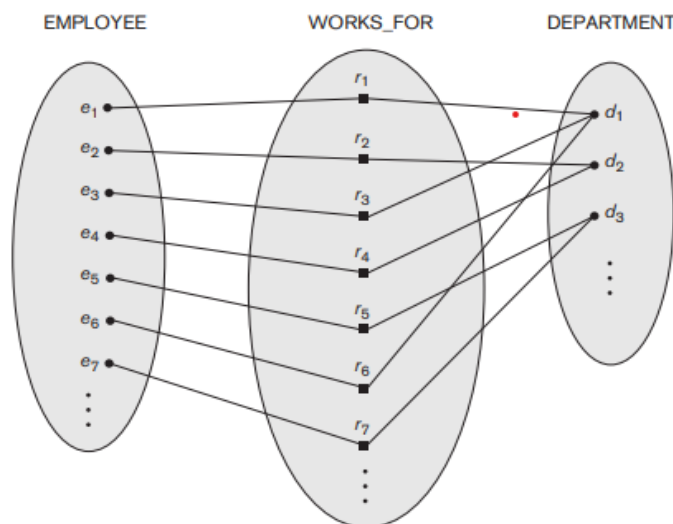
Figure 3.12

A 1:1 relationship,
MANAGES.



2. One-to-Many (1:N) Relationship

- A relationship where the items from one table can be linked to only one or many items from another table is called a one-to-many relationship.
- This connection becomes very strong in that it is particularly used to describe situations where one object can be linked to many similar or identical objects.
- For example, in an online store backend database, every customer may place multiple orders, yet the master customer record stays the same.

**Figure 3.9**

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

3. Many-to-Many (N:M) Relationship

- The duality of a [many-to-many relationship](#) is characterized by the presence of multiple records belonging to a table in association with multiple records from another table.

- In the many-to-many relationship model, a wide variety of complex relationships can be established where each entity has many related entities.
- Such a database for a music streaming service could have a table representing each track that belongs to multiple playlists, and each of them could contain multiple tracks.
- There are many people involved in each project, and every person can involve more than one project.

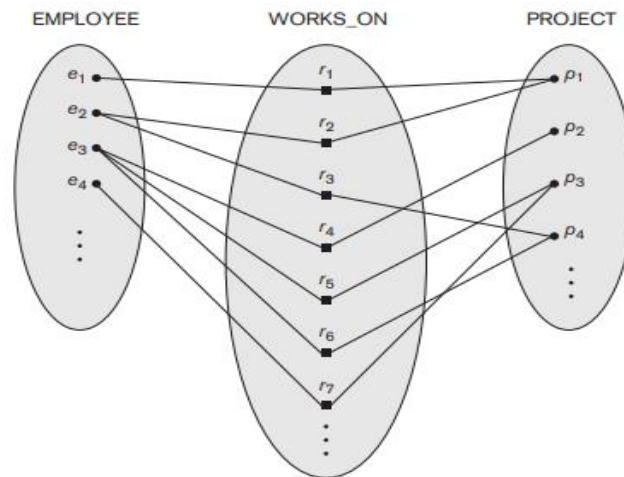
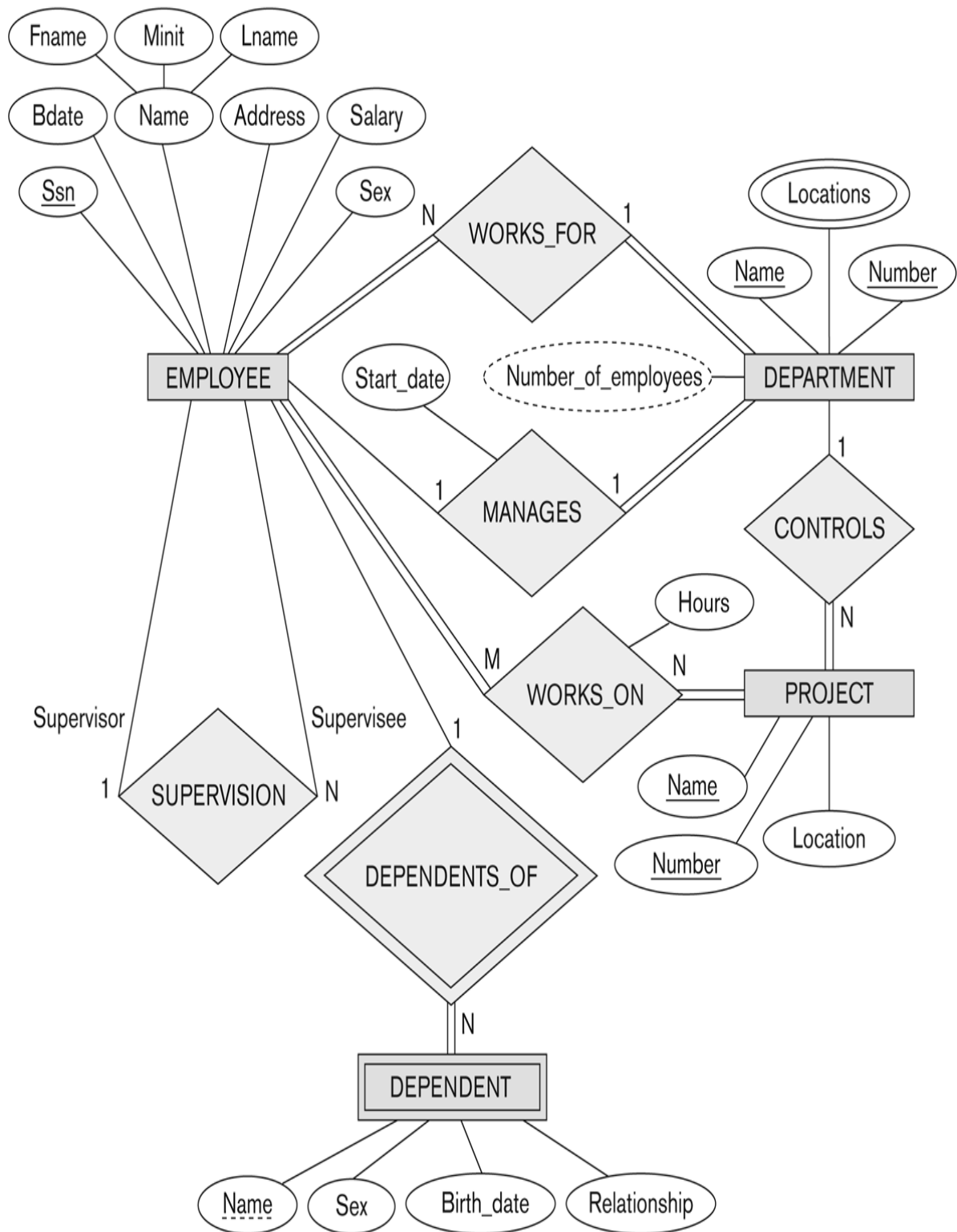


Figure 3.13
An M:N relationship,
WORKS_ON.

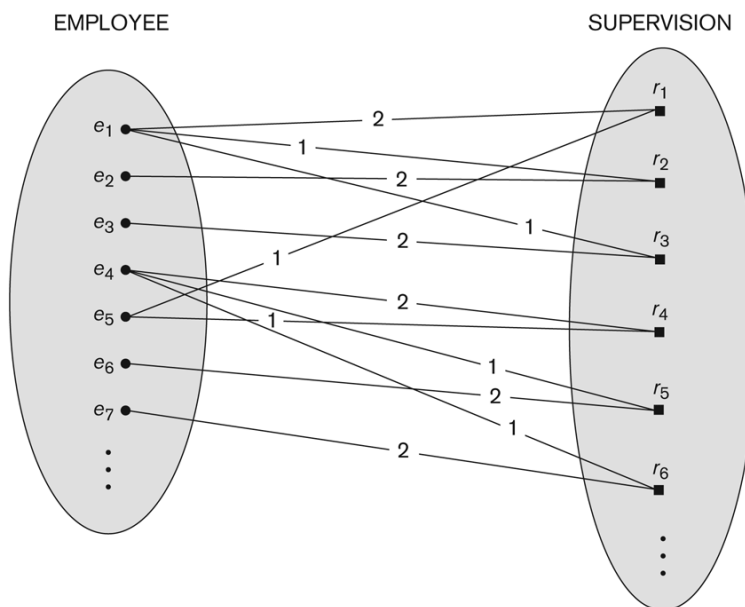
ER DIAGRAM: Example 1. Company Database

**Figure 3.2**

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

Recursive Relationship Type:

- An relationship type whose with the same participating entity type in **distinct roles**
- Example: the SUPERVISION relationship
- EMPLOYEE participates twice in two distinct roles:
 - supervisor (or boss) role
 - supervisee (or subordinate) role
- Each relationship instance relates two distinct EMPLOYEE entities:
 - One employee in *supervisor* role
 - One employee in *supervisee* role


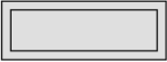
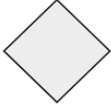




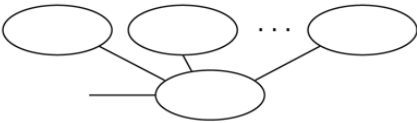

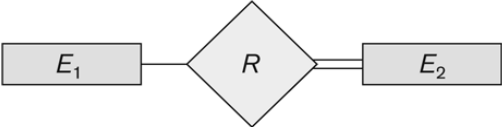

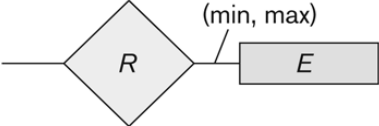
**Figure 3.11**

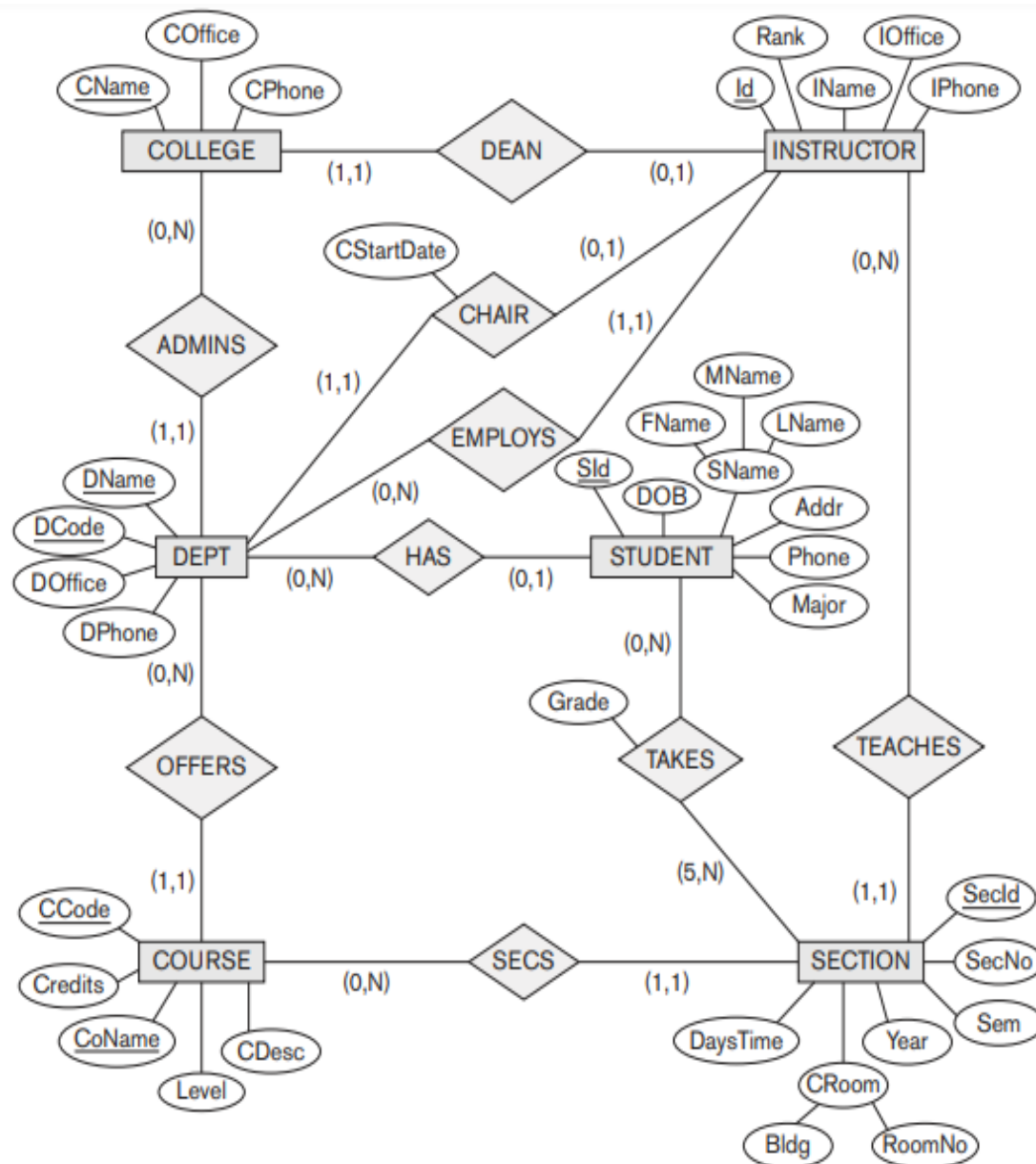
A recursive relationship SUPERVISION between EMPLOYEE in the *supervisor* role (1) and EMPLOYEE in the *subordinate* role (2).

Notation for ER diagrams

Figure 3.14

Summary of the notation for ER diagrams.

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

Example 2 : A UNIVERSITY Database**Database Design Other Models: UML class diagrams:**

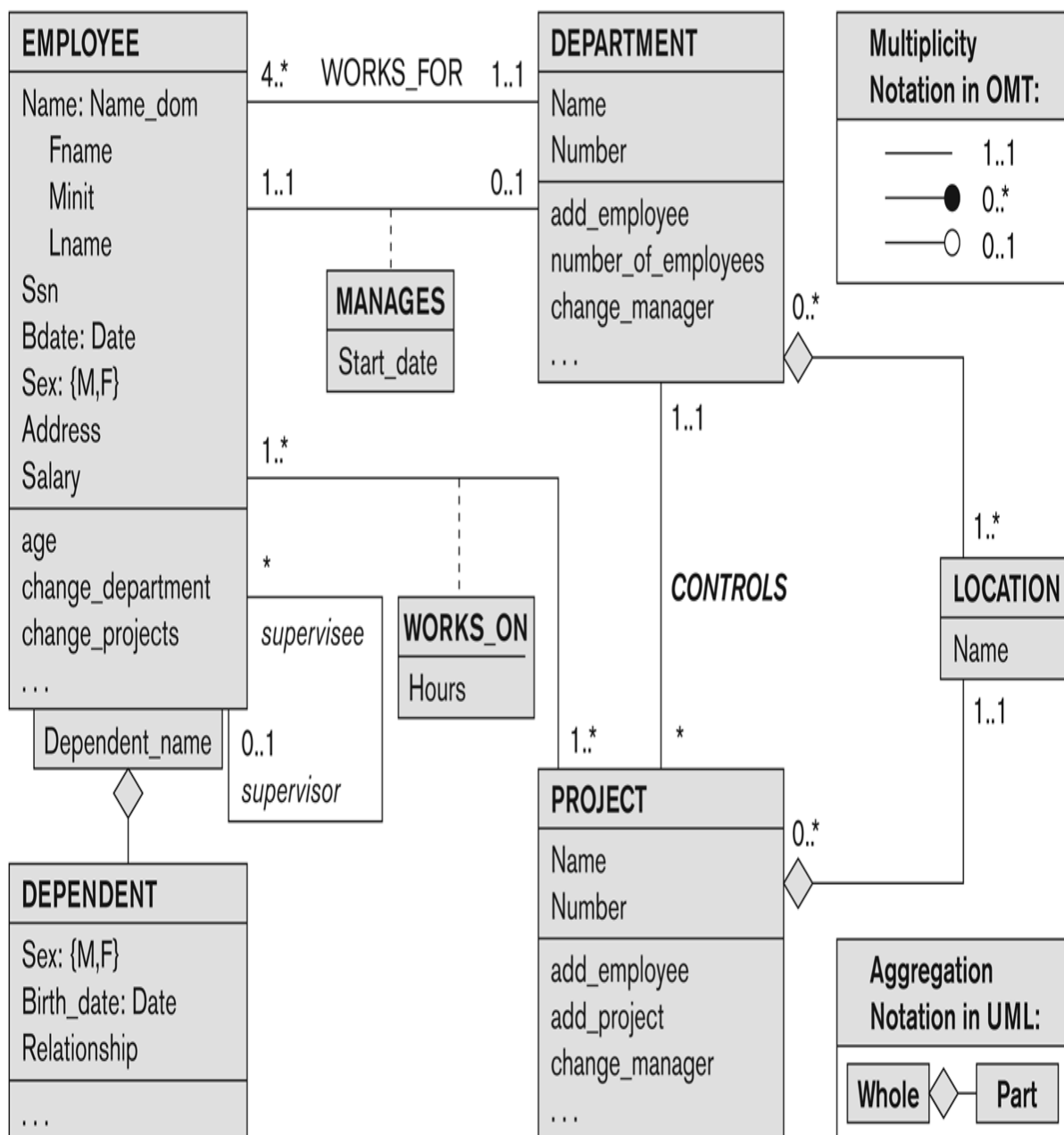
- We introduce the Unified Modeling Language (UML) notation for class diagrams, which has been proposed as a standard for conceptual object modeling.
- Represent classes (similar to entity types) as large rounded boxes with three sections:
 - Top section includes entity type (class) name
 - Second section includes attributes
 - Third section includes class operations (operations are not in basic ER model)

- Relationships (called associations) represented as lines connecting the classes
- Other UML terminology also differs from ER terminology
- Used in database design and object-oriented software design
- UML has many other types of diagrams for software design.

UML class diagram for COMPANY database schema

Figure 3.16

The COMPANY conceptual schema in UML class diagram notation.



Other alternative diagrammatic notations

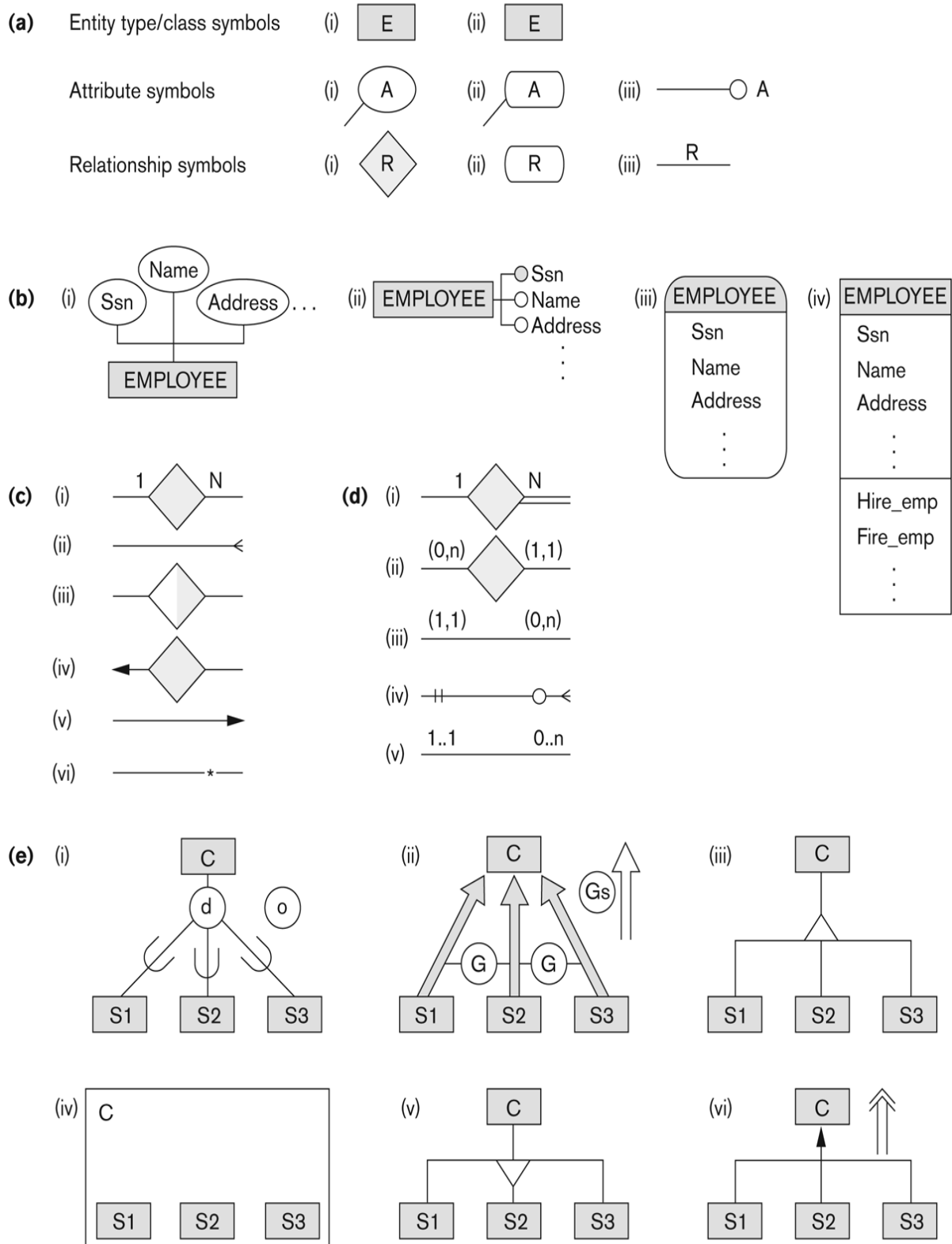


Figure A.1

Alternative notations. (a) Symbols for entity type/class, attribute, and relationship. (b) Displaying attributes. (c) Displaying cardinality ratios. (d) Various (min, max) notations. (e) Notations for displaying specialization/generalization.

Chapter 2 – Relational Algebra

Relational Operations

I. Fundamental Operators

■ These are the basic/fundamental operators used in Relational Algebra.

1. Selection(σ)
2. Projection(π)
3. Union(\cup)
4. Set Difference($-$)
5. Set Intersection(\cap)
6. Rename(ρ)
7. Cartesian Product(\times)

1. Selection(σ): It is used to select required tuples of the relations.

■ Example: For the above relation,

$\sigma(c>3)R$ will select the tuples which have c more than 3.

A	B	C	A	B	C
1	2	4	1	2	4
2	2	3	4	3	4
3	2	3			
4	3	4			

2. Projection(π): It is used to project required column data from a relation.

■ **Example:** Consider Table 1. Suppose we want columns B and C from Relation R.

$\pi(B,C)R$ will show following columns.

B	C
2	4
2	3
3	4

3. Union(U): Union operation in relational algebra is the same as union operation in [set theory](#).

■ **Example:**

French		German	
S_Name	R_Number	S_Name	R_Number
Mohan	02	Vivek	13
Vivek	13	Geeta	17
Geeta	17	Shyam	21
Ram	01	Rohan	25

- Consider the following table of Students having different Student_Name

$\pi(\text{Student_Name})\text{FRENCH} \cup \pi(\text{Student_Name})\text{GERMAN}$

Example:

Student_Name

Mohan

Vivek

Geetha

Ram

Shyam

Rohan

4. Set Difference(-): Set Difference in relational algebra is the same set difference operation as in set theory.

- Example: From the above table of FRENCH and GERMAN, Set Difference is used as follows

$\pi(\text{Student_Name})\text{FRENCH} - \pi(\text{Student_Name})\text{GERMAN}$

Example:

Student_Name

Ram

Mohan

5. Set Intersection(\cap): Set Intersection in relational algebra is the same set intersection operation in set theory.

- Example: From the above table of FRENCH and GERMAN, the Set Intersection is used as follows

$\pi(\text{Student_Name})\text{FRENCH} \cap \pi(\text{Student_Name})\text{GERMAN}$

Example:

Student_Name

Vivek

Geeta

6. Rename(ρ): Rename is a unary operation used for renaming attributes of a relation.

$\rho(a/b)R$ will rename the attribute 'b' of the relation by 'a'.

7. Cross Product(X): Cross-product between two relations. Let's say A and B, so the cross product between A X B will result in all the attributes of A followed by each attribute of B. Each record of A will pair with every record of B.

- **Example: A and B**

Name	Age	Sex
Ram	14	M
Sona	15	F
Kim	20	M

ID	Course
1	DS
2	DBMS

- **A X B**

Name	Age	Sex	ID	Course
Ram	14	M	1	DS
Ram	14	M	2	DBMS
Sona	15	F	1	DS
Sona	15	F	2	DBMS

Kim	20	M	1	DS
Kim	20	M	2	DBMS

II. Derived Operators

- These are some of the [derived operators](#), which are derived from the fundamental operators.

- Natural Join(\bowtie)

2. Conditional Join

1. Natural Join(\bowtie): Natural join is a binary operator. Natural join between two or more relations will result in a set of all combinations of tuples where they have an equal common attribute.

Example: **EMP**

Name ID Dept_Name

A	120	IT
B	125	HR
C	110	Sales
D	111	IT

- **DEPT**

Dept_Name Manager

Sales	Y
Production	Z
IT	A

- Natural join between EMP and DEPT with condition :

- **EMP.Dept_Name = DEPT.Dept_Name** or

- **EMP \bowtie DEPT**

Name	ID	Dept_Name	Manager
A	120	IT	A
C	110	Sales	Y

D 111 IT A

2. Conditional Join: Conditional join works similarly to natural join. In natural join, by default condition is equal between common attributes while in conditional join we can specify any condition such as greater than, less than, or not equal.

■ **Example:**

R

ID	Sex	Marks
1	F	45
2	F	55
3	F	60

■ **S**

ID	Sex	Marks
10	M	20
11	M	22
12	M	59

■ **Join between R and S with condition R.marks >= S.marks**

R.ID	R.Sex	R.Marks	S.ID	S.Sex	S.Marks
1	F	45	10	M	20
1	F	45	11	M	22
2	F	55	10	M	20
2	F	55	11	M	22
3	F	60	10	M	20
3	F	60	11	M	22
3	F	60	12	M	59

Relational Calculus:

- Relational calculus is a non-procedural query language.
- In the non-procedural query language, the user is concerned with the details of how to obtain the end results. The relational calculus tells what to do but never explains how to do.

- Comes in two flavors:
 - *Tuple relational calculus* (TRC) and
 - *Domain relational calculus* (DRC).
- Calculus has *variables, constants, comparison ops, logical connectives* and *quantifiers*.
 - TRC: Variables range over (i.e., get bound to) *tuples*.
 - DRC: Variables range over *domain elements* (= field values).
 - Both TRC and DRC are simple subsets of first-order logic.
- Expressions in the calculus are called *formulas*. An answer tuple is essentially an assignment of constants to variables that make the formula evaluate to *true*.

1. Tuple Relational Calculus (TRC)

- Tuple Relational Calculus (TRC) is a non-procedural query language used in relational database management systems (RDBMS) to retrieve data from tables.
- TRC is based on the concept of tuples, which are ordered sets of attribute values that represent a single row or record in a database table.
- TRC is a declarative language, meaning that it specifies what data is required from the [database](#), rather than how to retrieve it.
- TRC queries are expressed as logical formulas that describe the desired tuples.
- Syntax: The basic syntax of TRC is as follows:

$$\{ t \mid P(t) \}$$

- where t is a tuple variable and $P(t)$ is a logical formula that describes the conditions that the tuples in the result must satisfy.
- The curly braces $\{ \}$ are used to indicate that the expression is a set of tuples.
- $P(t)$ may have various conditions logically combined with OR (\vee), AND (\wedge), NOT (\neg). It also uses quantifiers:
 - $\exists t \in r (Q(t))$ = "there exists" a tuple in t in relation r such that predicate $Q(t)$ is true.
 - $\forall t \in r (Q(t))$ = $Q(t)$ is true "for all" tuples in relation r .

Example:

Table Loan

Loan number	Branch name	Amount
L33	ABC	10000
L35	DEF	15000
L49	GHI	9000
L98	DEF	65000

- Find the loan number, branch, and amount of loans greater than or equal to 10000 amount.
- $\{t \mid t \in \text{loan} \wedge t[\text{amount}] \geq 10000\}$

Resulting relation:

Loan number	Branch name	Amount
L33	ABC	10000
L35	DEF	15000
L98	DEF	65000

In the above query, $t[\text{amount}]$ is known as a tuple variable.

2. Domain Relational Calculus (DRC)

- [Domain Relational Calculus](#) is similar to Tuple Relational Calculus, where it makes a list of the attributes that are to be chosen from the relations as per the conditions.

$$\{ \langle a_1, a_2, a_3, \dots, a_n \rangle \mid P(a_1, a_2, a_3, \dots, a_n) \}$$

- where a_1, a_2, \dots, a_n are the attributes of the relation and P is the condition.

For example:

$$\{ \langle \text{article}, \text{page}, \text{subject} \rangle \mid \in \text{javatpoint} \wedge \text{subject} = \text{'database'} \}$$

Output: This query will yield the article, page, and subject from the relational javatpoint, where the subject is a database.