database

2024-2025

2025-2026

1.Database (dB)

Database is a collection of interrelated data stored together without harmful or unnecessary redundancy to serve multiple applications.

▶ 2.Database management system (DBMS)

(DBMSs) are specially designed applications that interact with the user, other applications, and the database itself to capture and analyze data. A general-purpose **(DBMS)** is a software system designed to allow the definition, creation, querying, update, and administration of DB.

- ▶ Well-known DBMSs include:
- MySQL
- Microsoft SQL Server
- Oracle
- FoxPro
- IBM DB2

- ▶ A database is not generally portable across different DBMS, but different DBMSs can by using standards such as SQL (Structure Query Language) and ODBC (Open Database Connectivity) or JDBC (The Java Database Connectivity) to allow a single application to work with more than one database.

Database Administrator DBA (DBA)

- is a person responsible for the performance, integrity and security of a database. He will also be involved in the planning and developing of the DB, as well as troubleshooting any issues on behalf of the users.

3.Purpose of Database Systems: (Why DB?)

- From the centralized control there are some advantages such as:

1- Redundancy can be reduced:

- In non-database systems each application has its own private files. That fact can often lead to considerable redundancy in stored data, which resultant waste in storage space.

2- Inconsistency can be avoided:

- If we have redundant data in one file, at such time the DB is said to be inconsistent. (If the redundancy is removed or controlled then the DB is never inconsistent as seen by the user).

3- The data can be shared:

- We mean that individual pieces of data in the DB may be shared among several different users and applications, in that each of those users may have access to the same piece of data.

4- Standards can be enforced:

- The database administrator (DBA) can ensure that all applicable standards are observed in the representation of the data.

5- Security restriction can be applied:

- The DBA can ensure that the only means of access to the DB is through the proper channels.
- The DBA can define security checks to be carried out whenever access is attempted to sensitive data.

6- Integrity can be maintained:

- Integrity means ensuring that the data in the DB is accurate. Inconsistency between two entries that represent the same fact is an example of integrity (Even if there is no redundancy however, the DB may still contain incorrect information). The DBA can define integrity checks to be carried out whenever any update operation is attempted.

7- Data Independence:

- It is the immunity of application to change in storage structure and access strategy, which implies that the application concerned do not depend on any one particular storage structure and access strategy.

4.1.Classification Based on Data Model

A Database model defines the logical design and structure of a database. It defines how data will be stored, accessed, and updated in a database management system.

- **Types of DB models:**

- Network
- Hierarchical
- Relational
- Entity-Relationship
- Extended Relational
- Object-oriented
- Object-relational
- Semi-structured (XML/ Extensible Markup Language)
- NoSQL

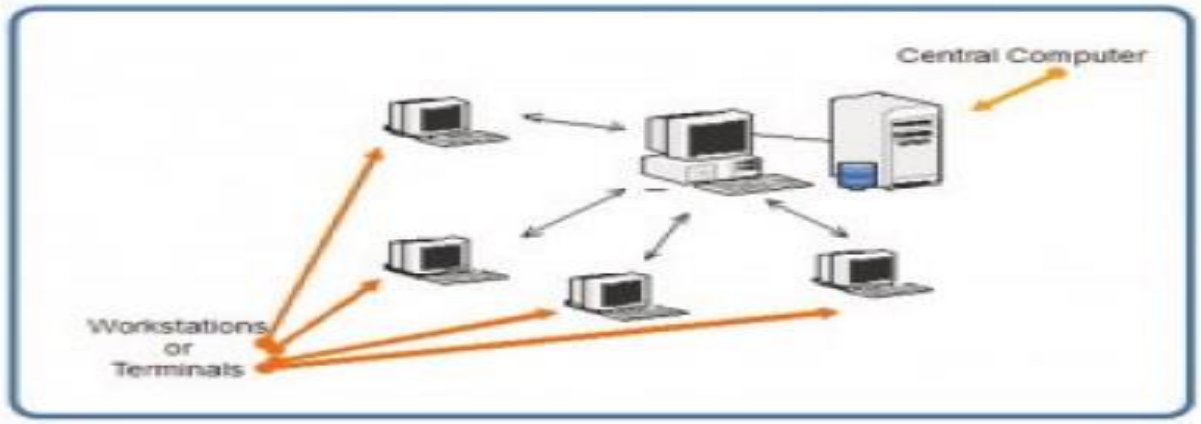
4.2.Classification Based on User Numbers

- It can be a single-user database system, which supports one user at a time, or a multiuser database system, which supports multiple users concurrently

4.3.Classification Based on Database Distribution

► There are four main distribution systems for database systems and these, in turn, can be used to classify the DBMS.

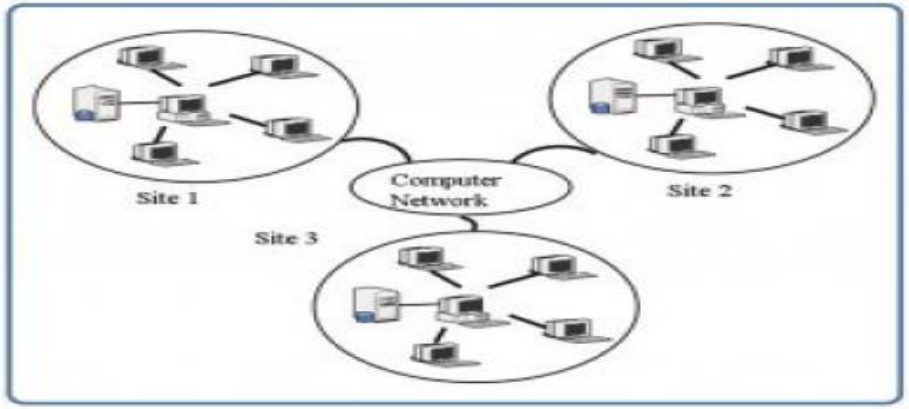
4.3.1Centralized systems: With a centralized database system, the DBMS and database are stored at a single site that is used by several other systems too



4.3.2 Distributed database system: In a distributed database system, the actual database and the DBMS software are distributed from various sites that are connected by a computer network, A distributed database system allows applications to access data from local and remote databases

- **Homogeneous distributed database systems:** Homogeneous distributed database systems use the same DBMS software from multiple sites. Data exchange between these various sites can be handled easily.

- **Heterogeneous distributed database systems:** In a heterogeneous distributed database system, different sites might use different DBMS software, but there is additional common software to support data exchange between these sites.



6. Relational Model

- From a historical perspective, the relational data model is relatively new. The first database systems were based on either the network model or the hierarchical model. Those two older models are tied more closely to the underlying implementation of the database than is the relational model.

- In the years following the introduction of the relational model, a substantial theory has developed for relational databases. This theory assists in the design of relational databases and in the efficient processing of user requests for information from the database.

- The relational model has established itself as the primary data model for commercial data processing applications. The relational model has established itself as the primary data model for commercial data processing applications.

6.1.Structure of Relational Databases

- In this model, data is organized in two-dimensional tables and the relationship is maintained by storing a common field.
- This model was introduced by **E.F Codd** in 1970, and since then it has been the most widely used database model.
- The basic structure of data in the relational model is **tables**. All the information related to a particular type is stored in **rows** of that table.
- Hence, tables are also known as relations in the relational model

Relationships :there are three types of relationships:

- 1:1 (unary relationship).
- 1:M (multiple relationship).
- N:M (complex relationship).

- **Database schema** :is a formal description of all the database relations and all the relationships existing between them.

- **Database Integrity** : means ensuring that the data in the DB is accurate. Inconsistency between two entries that represent the same fact is an example of integrity (Even if there is no redundancy however, the DB may still contain incorrect information). The DBA can define integrity checks to be carried out whenever any update operation is attempted.

In a relational data model, **data integrity** can be achieved using **integrity rules** or **constraints**

- **A Null** value is a special symbol, independent of data type, which means either unknown or inapplicable. It does not mean zero or blank.

Null Can represent:-

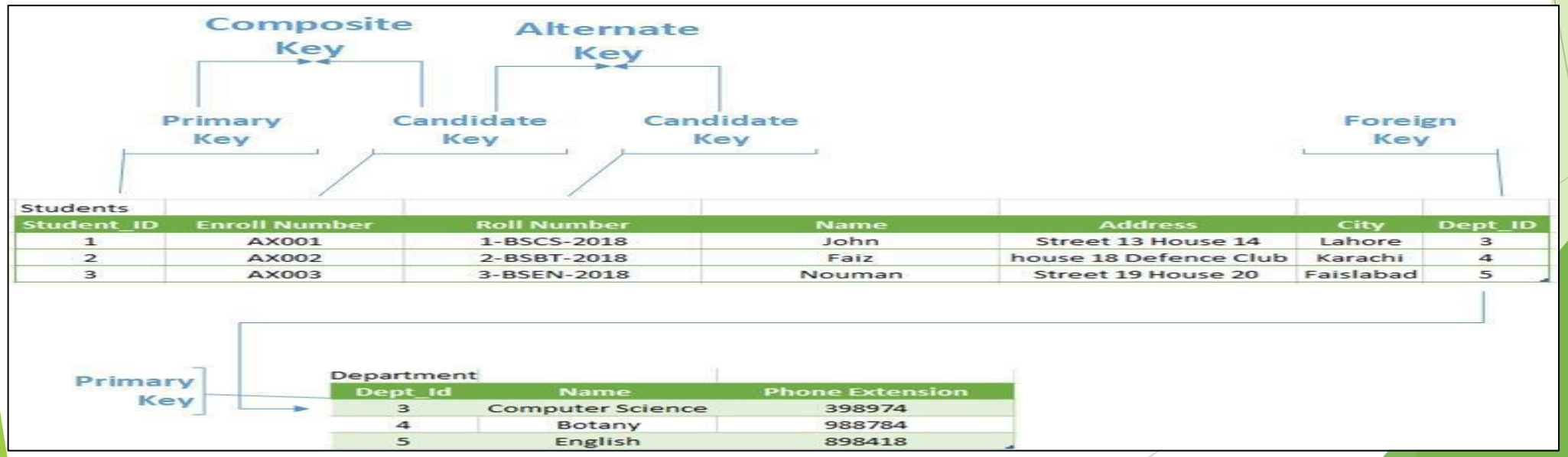
- An unknown attribute value.
- A known, but missing, attribute value.
- A not “applicable” condition.

- **ENTITY**: An entity is a person, place, thing, or event for which data is collected and maintained. For example student.

- **FIELD:** A field, also called an attribute, is a single characteristic or fact about an entity in the example shown in Fig(3) the student entity has seven fields that store the student_ID, enroll number, roll number, name, Address, City, dept_Id.

A common field is an attribute that appears in more than one entity. Common field can be used to link entities in various types of relationships.

- **RECORD:** A record, also called a tuple, is a set of related fields that describes one instance, or occurrence of an entity, such as one student, one department or one product. A record might have one or dozens of fields, depending on what information is needed.



Students

Student_ID	Enroll Number	Roll Number	Name	Address	City	Dept_ID
1	AX001	1-BSCS-2018	John	Street 13 House 14	Lahore	3
2	AX002	2-BSBT-2018	Faiz	house 18 Defence Club	Karachi	4
3	AX003	3-BSEN-2018	Nouman	Street 19 House 20	Faislabad	5

Department

Dept_Id	Name	Phone Extension
3	Computer Science	398974
4	Botany	988784
5	English	898418

Key Fields

During the systems design, you use key fields to organize, access, and maintain data structures. The four types of keys are primary keys, candidate keys, foreign keys, and secondary keys.

1. Primary Key:

A primary key is a field or combination of fields that uniquely and minimally identifies a particular member of an entity; a table can have only one primary key.

A combination key also can be called a composite key, a concatenated key, or a multi-valued key.

2. Candidate Key:

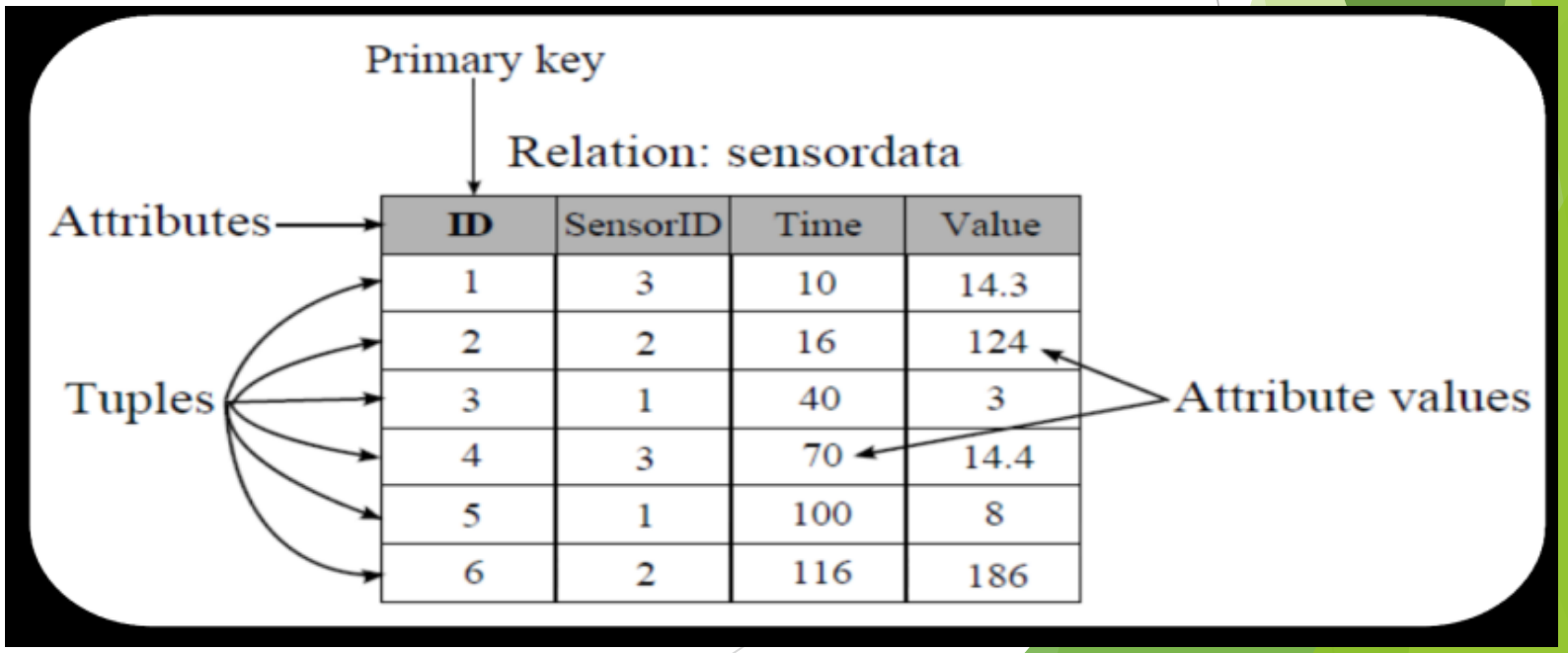
Sometimes you have a choice of fields or field combinations to use as the primary key. Any field that could serve as a primary key is called a candidate key.

3. Foreign Key:

A foreign key is a field in one table that must match a primary key value in another table in order to establish the relationship between the two tables. Unlike a primary key, a foreign key need not be unique.

4. Secondary Key:

A secondary key is a field or combination of fields that can be used to access or retrieve records. Secondary key values are not unique. The need for a secondary key arises because a table can have only one primary key.



Relational dB constraint

Those rules are general, specified at the database schema level, and they must be respected by each schema instance. If we want to have a correct relational database definition, we have to declare such constraints.

- Relational data model constraints are:
  - Entity integrity constraint
  - Referential integrity constraint
  - Semantic integrity constraints

1. Entity Integrity Constraint

The entity integrity constraint says that no attribute participating in the primary key of a relation is allowed to accept null values.

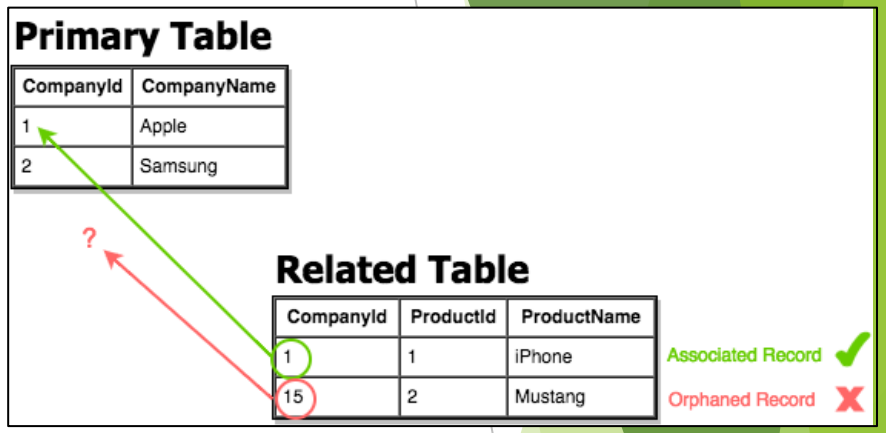
EMPLOYEE

EMP_ID	EMP_NAME	SALARY
123	Jack	30000
142	Harry	60000
164	John	20000
	Jackson	27000

Not allowed as primary key can't contain a NULL value

2. Referential Integrity Constraint

The referential integrity constraint says that if a relation R2 includes a foreign key FK matching the primary key PK of other relation R1, then every value of FK in R2 must either be equal to the value of PK in some tuple of R1 or be wholly null.



3. Semantic Integrity Constraint

A semantic integrity constraint refers to the correctness of the meaning of the data. For example, the street number attribute value from the OWNERS relation must be positive, because the real-world street numbers are positive.

Semantic integrity constraints must be specified typically by a database administrator and must be maintained in the system catalog or dictionary.

Relational DBMS permits several types of semantic integrity constraints such as :

- **domain constraint**
- **null constraint**
- **unique constraint**
- **check constraint**

☒ Domain constrain

A domain constraint implies that a particular attribute of a relation is defined on a particular domain. For example, the Street attribute domain of OWNERS relation is CHAR(20), because streets have names in general and Number attribute domain is NUMERIC, because street numbers are numeric values.

There are some particular forms of domain constraints, namely **format constraints and range constraints.**

A **format constraint** might specify something like a data value pattern.

For example, the PhoneNumber attribute values must be of this type XXXX-XXXXXX where X represents an a digits first 4 X is the key and the last 6 is phone number .

A range constraint requires that values of the attribute lie within the range values. For example, the FabricationYear attribute values might range between 1950 and 2010.

☒ Null constraint

A null constraint specifies that attribute values cannot be null. On every tuple, from every relation instance, that attribute must have a value which exists in the underlying attribute domain.

For example, FirstName and LastName attributes values cannot be null, this means that a car owner must have a name.

☒ Unique constraint

A unique constraint specifies that attribute values must be different. It is not possible to have two tuples in a relation with the same values for that attribute.

For example, in the CARS relation the SerialNumber attribute values must be unique, because it is not possible to have two cars and only one engine.

A unique constraint is usually specified with an attribute name followed by the word Unique. Note that NULL is a valid unique value.

☒ Check Constraint

A check constraint is a more complicated type of restriction that evaluates a Boolean expression to see if certain data should be allowed. If the expression evaluates to true, the data is allowed. For example, in a Salespeople table you could place the constraint on salary field that Salary > 0 which mean that the field‟s value must be positive

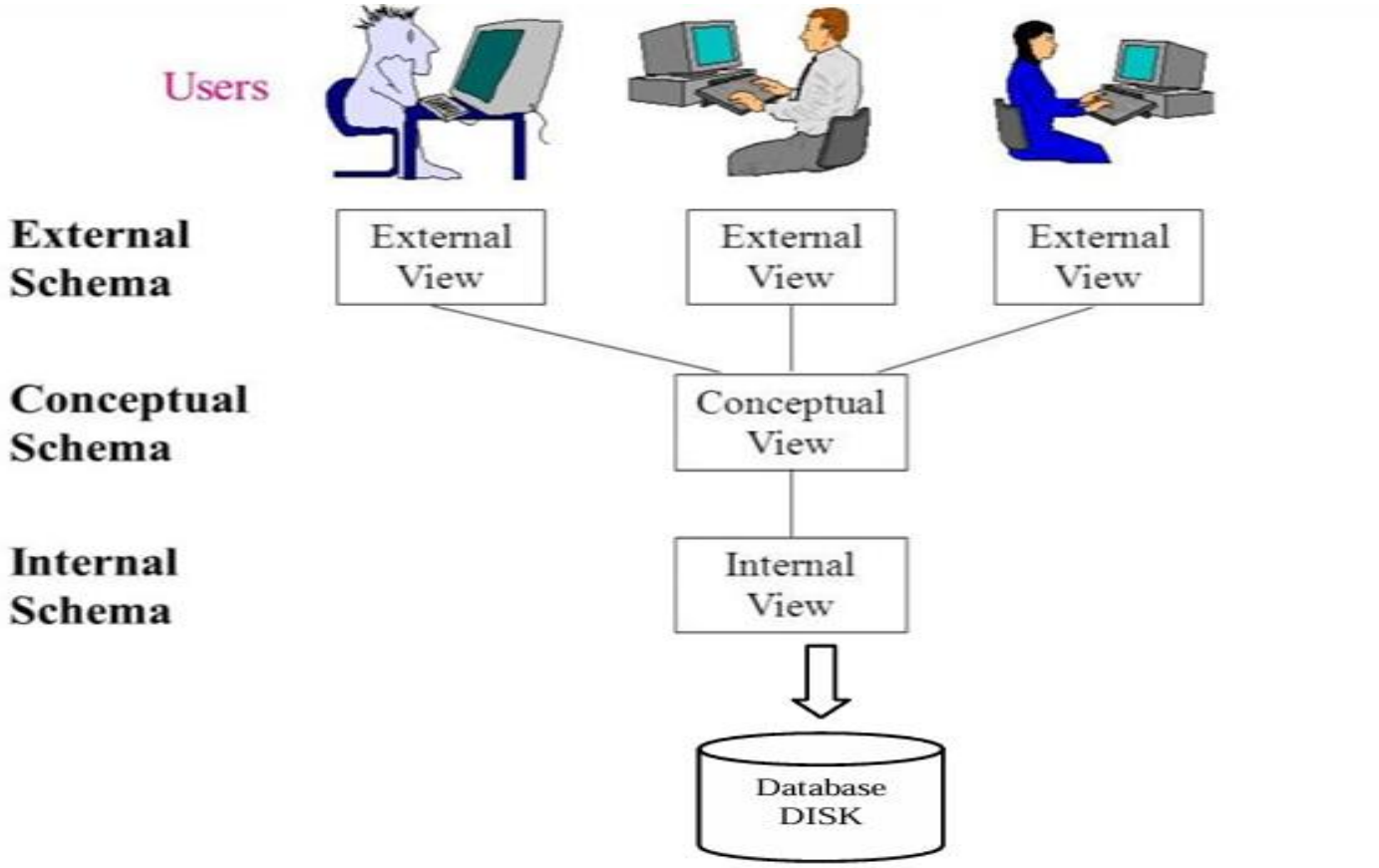
Data Abstraction

- For the system to be usable, it must retrieve data efficiently. The need for efficiency has led designers to use complex data structures to represent data in the database. Since many database-system users are not computer trained, developers hide the complexity from users through several levels of abstraction, to simplify users' interactions with the system:

- • **Physical level (or Internal View / Schema):** The lowest level of abstraction describes how the data are actually stored. The physical level describes complex low-level data structures in detail.
- • **Logical level (or Conceptual View / Schema):** The next-higher level of abstraction describes what data are stored in the database, and what relationships exist among those data. The logical level thus describes the entire database in terms of a small number of relatively simple structures. Although implementation of the simple structures at the logical level may involve complex physical-level structures, the user of the logical level does not need to be aware of this complexity. This is referred to as physical data independence. Database administrators, who must decide what information to keep in the database, use the logical level of abstraction.

• **View level (or External View / Schema):** The highest level of abstraction describes only part of the entire database. Even though the logical level uses simpler structures, complexity remains because of the variety of information stored in a large database. Many users of the database system do not need all this information; instead, they need to access only a part of the database. The view level of abstraction exists to simplify their interaction with the system. The system may provide many views for the same database

.



An analogy to the concept of data types in programming languages may clarify the distinction among levels of abstraction. Many high-level programming languages support the notion of a structured type. For example, we may describe a record as follows:

```
type instructor = record
    ID : char (5);
    name : char (20);
    dept name : char (20);
    salary : numeric (8,2);
end;
```

This code defines a new record type called instructor with four fields. Each field has a name and a type associated with it. A university organization may have several such record types, including

department, with fields dept_name, building, and budget

course, with fields course_id, title, dept_name, and credits

student, with fields ID, name, dept_name, and tot_cred

At the physical level, an instructor, department, or student record can be described as a block of consecutive storage locations. The compiler hides this level of detail from programmers. Similarly, the database system hides many of the lowest-level storage details from database programmers.

Database administrators, on the other hand, may be aware of certain details of the physical organization of the data.

At the logical level, each such record is described by a type definition, as in the previous code segment, and the interrelationship of these record types is defined as well. Programmers using a programming language work at this level of abstraction. Similarly, database administrators usually work at this level of abstraction.

Finally, at the view level, computer users see a set of application programs that hide details of the data types.

At the view level, several views of the database are defined, and a database user sees some or all of these views. In addition to hiding details of the logical level of the database, the views also provide a security mechanism to prevent users from accessing certain parts of the database. For example, clerks in the university registrar office can see only that part of the database that has information about students; they cannot access information about salaries of instructors.

Database systems have **several schemes**, partitioned according to the levels of abstraction. At the lowest level is the *physical scheme*; at the intermediate level, the *conceptual scheme*; at the highest level, a *subscheme*. In general, database systems support one physical scheme, one conceptual scheme, and several subschemes.

Data Independence

The ability to modify a scheme definition in one level without affecting a scheme definition in the next higher level is called *data independence*. There are two levels of data independence:

- **Physical data independence** is the ability to modify the physical scheme without causing application programs to be rewritten. Modifications at the physical level are occasionally necessary in order to improve performance.
- **Logical data independence** is the ability to modify the conceptual scheme without causing application programs to be rewritten. Modifications at the conceptual level are necessary whenever the logical structure of the database is altered.

- Logical data independence is more difficult to achieve than physical data independence since application programs are heavily dependent on the logical structure of the data they access.

- In general, the concept of data independence hides implementation details from the users. This allows users to concentrate on the general structure rather than low-level implementation details.

Transaction and ACID properties

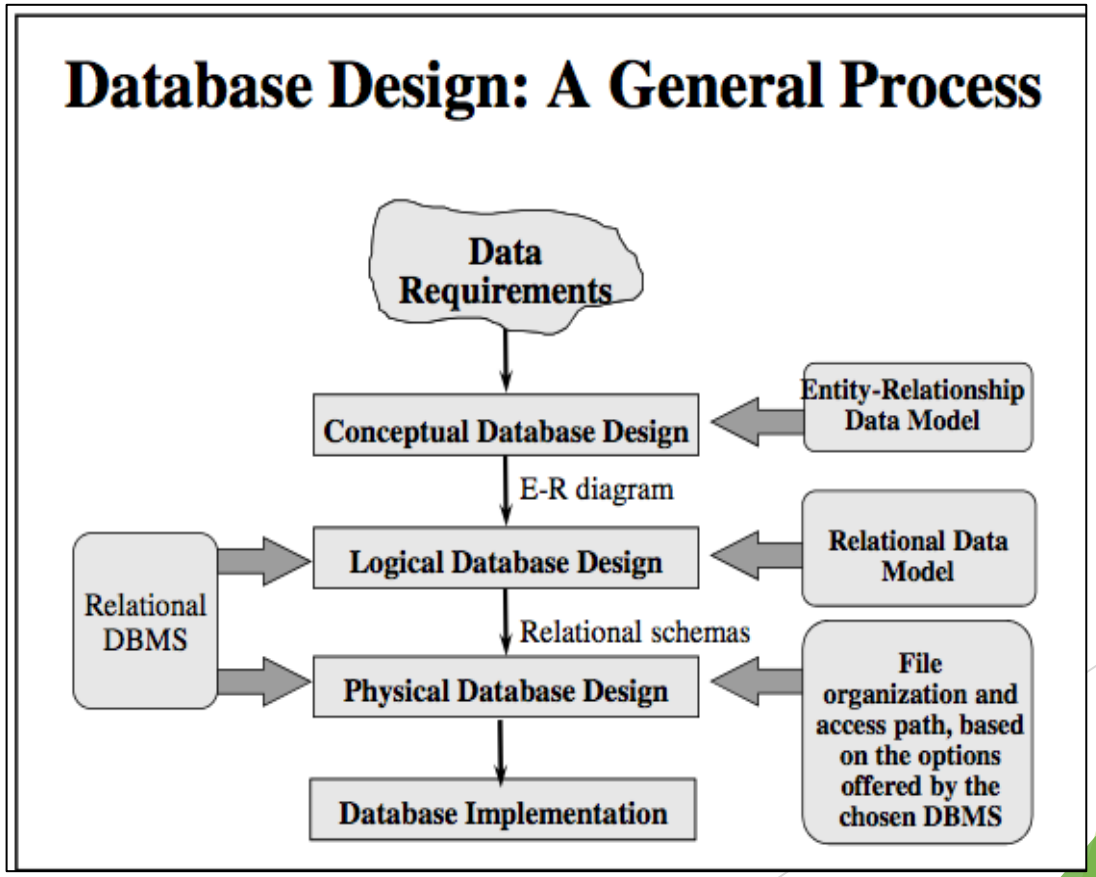
- Transactions are a sequence of queries and updates that together carry out a task. Transactions can be committed, or rolled back; when a transaction is rolled back, the effects of all updates performed by the transaction are undone**.**
- ACID is an acronym describing four features that an effective transaction system should provide. ACID stands for Atomicity, Consistency, Isolation, and Durability.
-
- - **Atomicity** means transactions are atomic. The operations in a transaction either all happen or none of them happen.
- - **Consistency** means the transaction ensures that the database is in a consistent state before and after the transaction.
- In other words, if the operations within the transaction would violate the database‟s rules, the transaction is rolled back.

- **Isolation** means the transaction isolates the details of the transaction from everyone except the person making the transaction.
Suppose you start a transaction, remove $100 from Alice‟s account, and add $100 to Bob‟s account. Another person cannot peek at the database while you‟re in the middle of this process and see a state where neither Alice nor Bob has the $100. Anyone who looks in the database sees the $100 somewhere, either in Alice‟s account before the transaction or in Bob‟s account afterwards.

Durability means that once a transaction is committed, it will not disappear later. If the power fails, when the database restarts, the effects of this transaction will still be there.

7. Database design

- Database Design and Application Development: How can a user describe a real-world enterprise (e.g., a university) in terms of the data stored in a DBMS? What factors must be considered in deciding how to organize the stored data?



1.1 Database Design Process

- The database design process can be divided into six steps. The E-R model is most relevant to the first three steps.

1. Requirements Analysis:
   The very first step in designing a DB application is to understand what data is to be stored in the DB, what applications must be built on top of it, and what operations are most frequent and subject to performance requirements.

2. Conceptual Database Design: The information gathered in the requirements analysis step is used to develop a high-level description of the data to be stored in the DB, along with the constraints known to hold over this data. This step is often carried out using the E-R model.

3. Logical Database Design: We must choose a DBMS to implement our DB design, and convert the conceptual database design into a DB schema in the data model of the chosen DBMS.

4. Schema Refinement: The fourth step with DB's design is to analyze the collection of relations in our relational DB schema to identify potential problems, and to refine it (Normalization).

5. Physical Database Design: It describes the details of how data is stored. This step may simply involve building indexes on some tables and clustering some tables

6. Application and Security Design: Any software project that involves a DBMS must consider aspects of the application that go beyond the database itself. We must identify the entities (e.g., users, user groups, departments) and processes involved in the application. We must describe the role of each entity in every process that is reflected in some application task, as part of a complete workflow for that task.

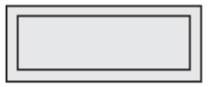
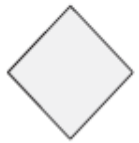
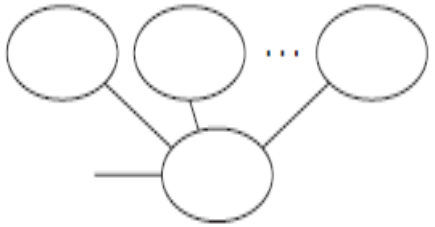
8.The Entity Relationship Mode

ER data model has existed for over 35 years. It is well suited to data modelling for use with DB because it is fairly abstract and is easy to discuss and explain. ER models are readily translated to relations. ER models, also called an ER schema, are represented by

ER diagrams. ER modelling is based on two concepts:

1. Entities, defined as tables that hold specific information (data)
2. Relationships, defined as the associations or interactions between entities

Table(1) Entity Relationship Diagram Symbols

Symbol	Meaning
	Entity
	Weak Entity
	Relationship
	Indentifying Relationship
	Attribute
	Key Attribute
	Multivalued Attribute
	Composite Attribute
	Derived Attribute

- **Entity**
- An entity is an object in the real world with an independent existence that can be differentiated from other objects. An entity might be
  - An object with physical existence (e.g., a lecturer, a student, a car)
  - An object with conceptual existence (e.g., a course, a job, a position)

- Entities can be classified based on their strength into:-

-**Weak entity:** an entity is considered **weak** if its tables are existence dependent.
- That is, it cannot exist without a relationship with another entity
- Its primary key is derived from the primary key of the parent entity

- Strong entity: an entity is considered **strong** if it can exist a part from all of its related entities.

- A table without a foreign key or a table that contains a foreign key that can contain nulls is a strong entity

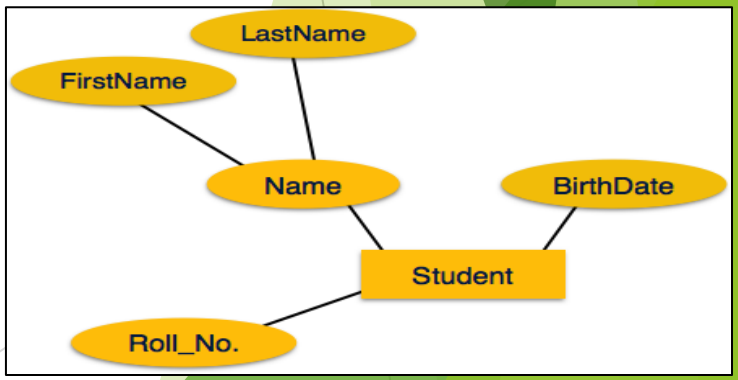
Entity and attributes

- 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 particular entity will have a value for each of its attributes.
- **Types Of Attributes**

Several types of attributes occur in the ER model: **simple** versus **composite**, **single** valued **versus multivalued**, and **stored** versus **derived.**

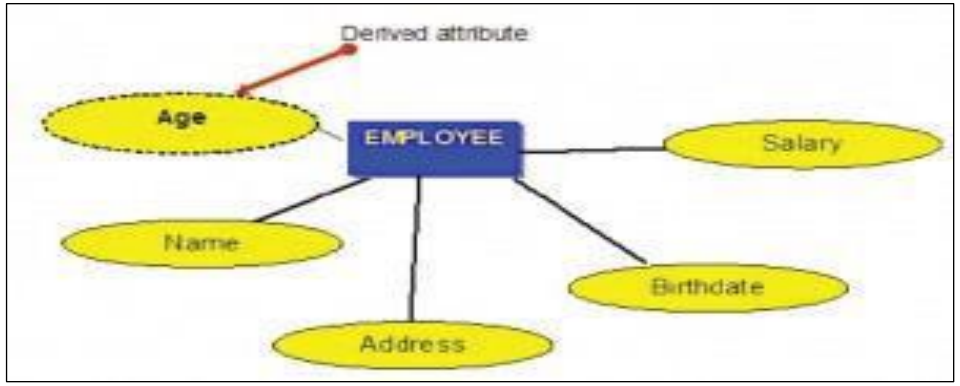
1. Simple attributes: Are those drawn from the atomic value domains; they are also called single-valued attributes. In the COMPANY database, an example of this would be: Name = {John} ; Age = {23}

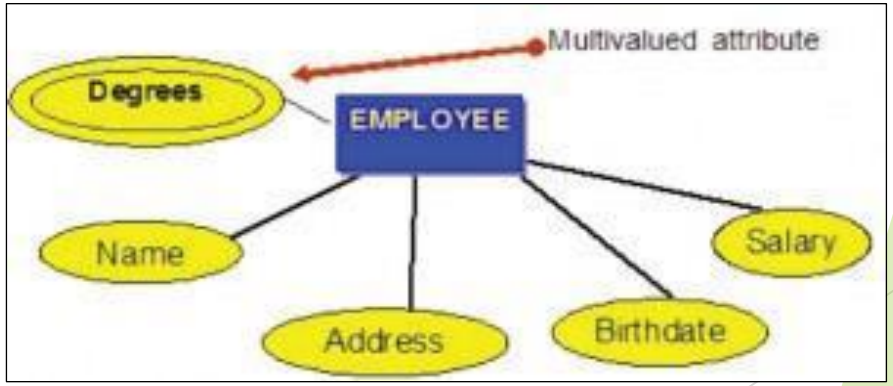
2. Composite attributes: Are those that consist of a hierarchy of attributes. Figure 3 Address may consist of Number, Street and Suburb. So this would be written as → Address = {59 + 'Meek Street' + 'Kingsford'}



3. Multivalued attributes: Are attributes that have a set of values for each entity. For ex: the degrees of an employee: BSc, MIT, PhD.

4. Derived attributes: Are attributes that contain values calculated from other attributes. Age can be derived from the attribute Birthdate. In this situation, Birthdate is called a stored attribute, which is physically saved to the database.



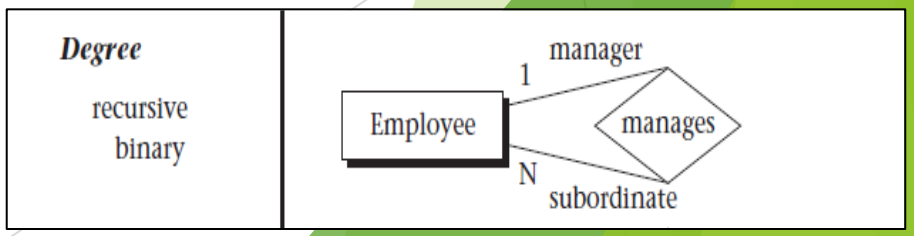


Degree of a Relationship

- The degree of a relationship is the number of entities associated in the relationship. **Binary** and **ternary** relationships are special cases where the degrees are 2 and 3, respectively.

- The **binary** relationship, an association between two entities, is by far the most common type in the natural world. In fact, many modeling systems use only this type. we see many examples of the association of two entities in different ways: Department and Division, Department and Employee, Employee and Project, and so on.

- A binary recursive relationship (for example, “manages”) relates a particular Employee to another Employee by management. It is called **recursive** is one in which the same entity participates more than once in the relationship; because the entity relates only to another instance of its own type.
- The binary recursive relationship construct is a diamond with both connections to the same entity. A ternary relationship is an association among three entities.

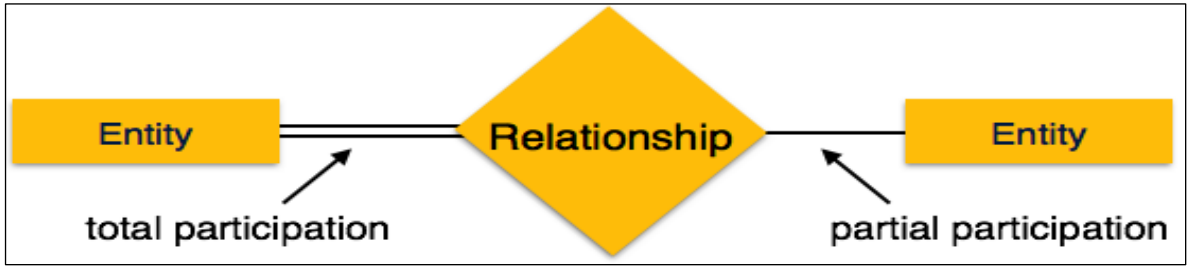


Participation

- Participation refers to whether an entity *must* participate in a relationship with another entity to exist. can be total or partial (optional):

- **Total participation** is where an entity must participate in a relationship to exist. For example, an employee must work for at least one department to exist as an employee.

- **Partial (optional) participation** is where the entity can exist without participating in a relationship with another entity . For example, the entity course may exist within an organization, even though it has no current students.



9.How to Convert ER Diagram to Relational Database

- We will be following the simple rules:

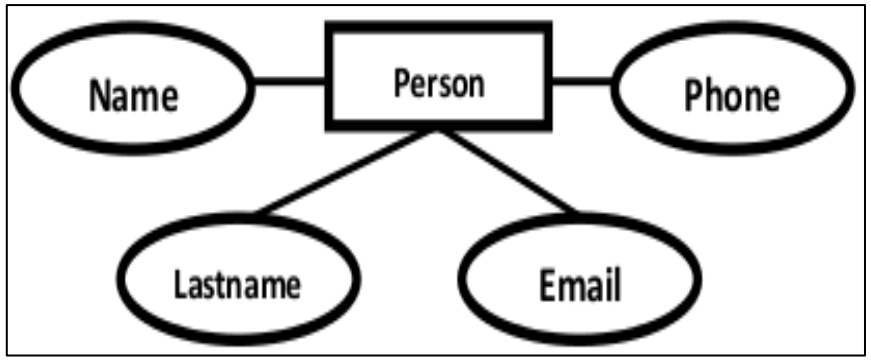
1. **Entities and Simple Attributes:**

- An entity type within ER diagram is turned into a table. You may preferably keep the same name for the entity or give it a sensible name but avoid DBMS reserved words as well as avoid the use of special characters.
- Each attribute turns into a column (attribute) in the table. The key attribute of the entity is the primary key of the table which is usually underlined. It can be composite if required but can never be null.
- It is highly recommended that every table **should start with its primary key** attribute conventionally named as TablenameID.

- Taking the following simple ER diagram:

The initial relational schema is expressed in the following format writing the table names with the attributes list inside a parentheses as shown below for Persons and Phones are Tables. name, lastname, are Table Columns (Attributes).

- Persons (**personid**, name, lastname, email)



2. Multi-Valued Attributes

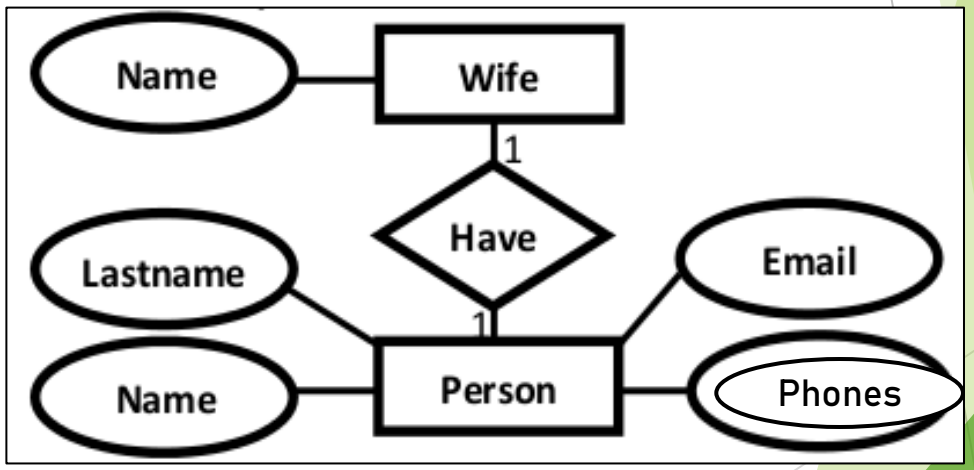
A multi-valued attribute is usually represented with a double-line oval.

If you have a multi-valued attribute, take the attribute and turn it into a new entity or table of its own. Then make a 1: N relationship between the new entity and the existing one. In simple words:

- Create a table for the attribute.
- Add the primary (id) column of the parent entity as a foreign key within the new table

Persons (personid, name, lastname, email)

Phones (phoneid , ***personid***, phone)



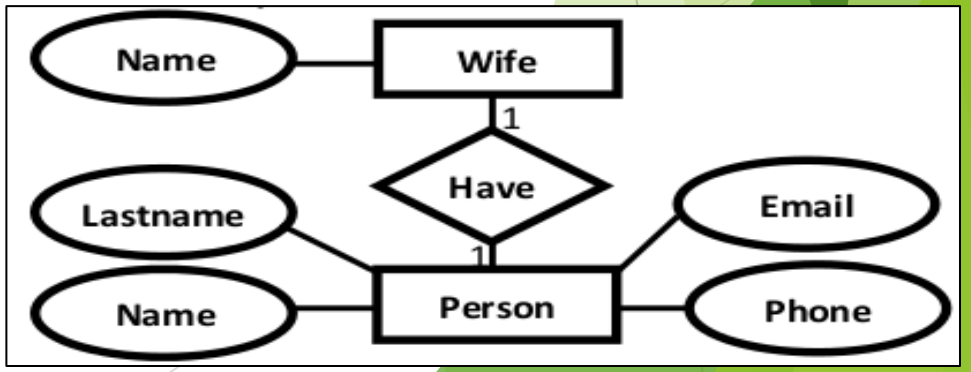
3. 1:1 Relationships

- To keep it simple and even for better performances at data retrieval, I would personally recommend using attributes to represent such relationship. For instance, let us consider the case where the Person has or optionally has one wife. You can place the primary key of the wife within the table of the Persons which we call in this case Foreign key as shown below.

Persons (personid, name, lastname, email , phone, ***wifeid***)
Wife (wifeid , name)

- Or vice versa to put the personid as a foreign key within the Wife table as shown below:

Persons (personid , name, lastname, email, phone)
Wife (wifeid , name , ***personid)***



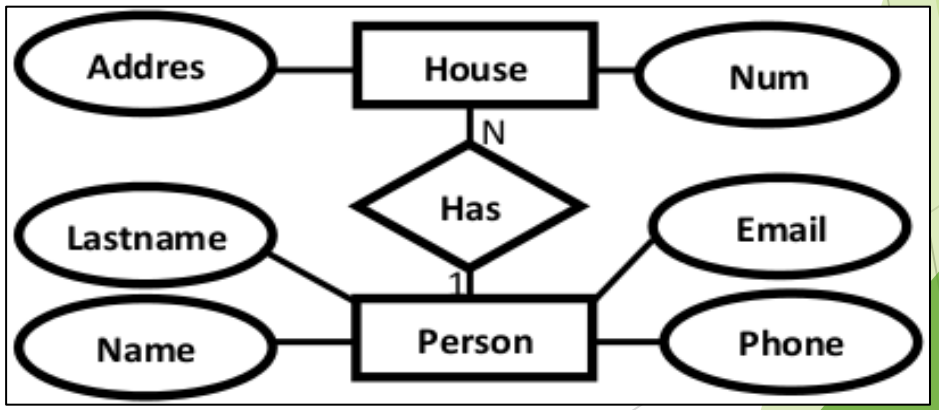
4. 1: N Relationships

- This is the tricky part! For simplicity, use attributes in the same way as 1:1 relationship but we have only one choice as opposed to two choices. For instance, the Person can have a **House** from zero to many, but a **House** can have only one **Person**. To represent such relationship the **personid**as the Parent node must be placed within the Child table as a foreign key but not the other way around as shown next:

- It should convert to:

Persons (personid , name, lastname, email)

House (houseid , num , address, ***personid***)



5. N:N Relationships

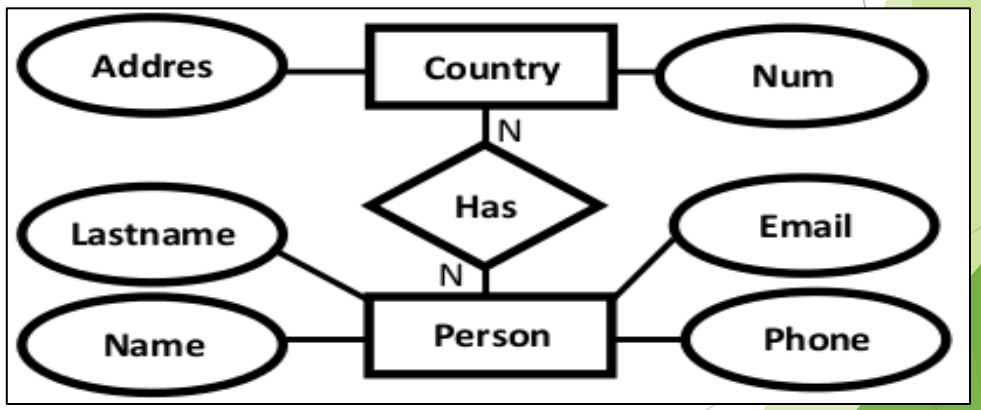
- We normally use tables to express such type of relationship. It is the same for N − ary relationship of ER diagrams. For instance, The Person can live or work in many countries. Also, a country can have many people. To express this relationship within a relational schema we use a separate table as shown below:

- it should convert into :

Persons(personid , name, lastname, email)

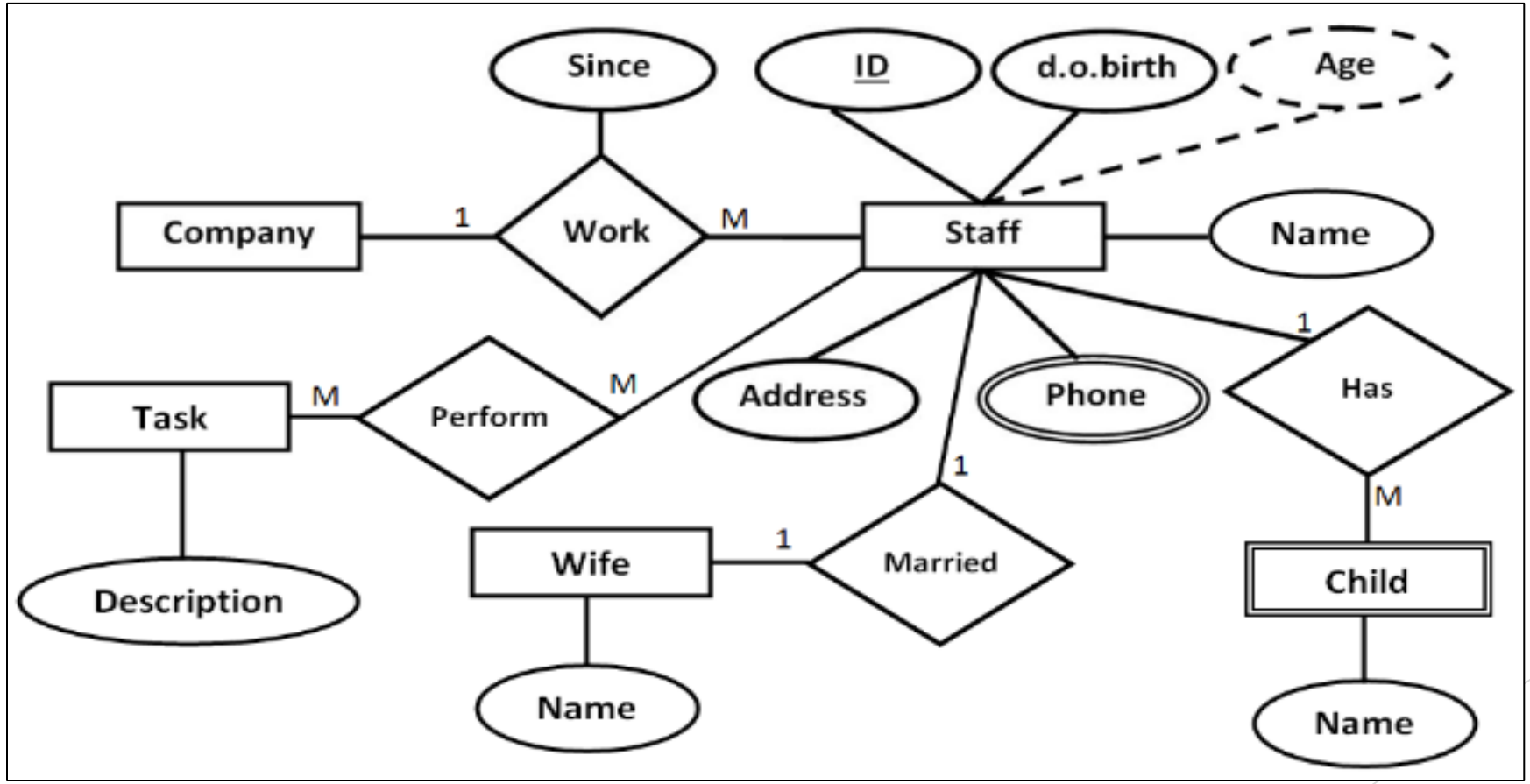
Countries (countryid , name, code)

HasRelat (hasrelatid , **personid , countryid**)



Case Study

- We will be producing the relational schema for the following ER diagram:



10. Relational Algebra

- Relational algebra a formal query language for asking questions. Is a set of operators to manipulate relations. Each operator of the relational algebra takes either one or two relations as its input and produces a new relation as its output

- Codd defined 8 such operators, two groups of 4 each:
  - The traditional set operations: union, intersection, difference and Cartesian product
  - The special relational operations: select, project, join and divide.

1. Selection: σ

- The *selection* operator extracts certain *rows* from the table and discards the others.

Retrieved tuples must satisfy a given filtering condition.

Syntax: *Result* = σ ***c*** (*R*)

- Example : find the movies made by Hanson after 1997

Movies

title	director	myear	rating
Fargo	Coen	1996	8.2
Raising Arizona	Coen	1987	7.6
Spiderman	Raimi	2002	7.4
Wonder Boys	Hanson	2000	7.6

$\sigma_{myear>1997 \wedge director='Hanson'}(\text{Movies})$

title	director	myear	rating
Wonder Boys	Hanson	2000	7.6

Selection Condition

- **Selection condition is a Boolean combination of terms; A term is one of the following forms:-**

1. **attribut op constant op ∈ {=, ≠,<,≤,>, ≥}**
2. **attribute1 op attribute2**
3. **term1 ∧ term2**
4. **term1 ∨ term2**
5. **¬ term1**
6. **(term1)**

- **Operator precedence: (), op, ¬, ∧, ∨**

Example: Find movies made after 1997

Movies

title	director	myear	rating
Fargo	Coen	1996	8.2
Raising Arizona	Coen	1987	7.6
Spiderman	Raimi	2002	7.4
Wonder Boys	Hanson	2000	7.6

$\sigma_{myear>1997}(\text{Movies})$

title	director	myear	rating
Spiderman	Raimi	2002	7.4
Wonder Boys	Hanson	2000	7.6

2 Projection π

- The project operator extracts certain columns from the table and discards the other columns; Duplicate tuples in the resulting table are eliminated

$$\pi_{fields}(Input)$$

Allows us to extract columns from a relation

- **Example: Find all movies and their ratings**

Movies

title	director	myear	rating
Fargo	Coen	1996	8.2
Raising Arizona	Coen	1987	7.6
Spiderman	Raimi	2002	7.4
Wonder Boys	Hanson	2000	7.6

$$\pi_{title,\ rating}(\text{Movies})$$

title	rating
Fargo	8.2
Raising Arizona	7.6
Spiderman	7.4
Wonder Boys	7.6

Set Operations

- **Union: R ∪ S** returns a relation containing all tuples that occur in R or S (or both). Remove the duplicated tuples
- **Intersection: R ∩ S** returns a relation containing all tuples that occur in both R and S
- **Set-difference: R − S** returns a relation containing all tuples in R but not in S

Two relations are **union compatible** if :-

- They have the same number of fields

- corresponding fields, have the same domains

- **union (∪), intersection (∩), and set-difference (−) operators require input relations to be union compatible**

Actors

actor	ayear
Cage	1964
Hanks	1956
Maguire	1975
McDormand	1957

$\pi_{actor}(\textbf{Actors})$

actor
Cage
Hanks
Maguire
McDormand

Directors

director	dyear
Coen	1954
Hanson	1945
Raimi	1959

$\pi_{director}(\textbf{Directors})$

director
Coen
Raimi
Hanson

$\pi_{actor}(Actors) \cup \pi_{director}(Directors)$

actor
Cage
Hanks
Maguire
McDormand
Coen
Raimi
Hanson

Union Example:

Find all actors & directors

$\pi_{actor}(Actors) \cup \pi_{director}(Directors)$

R1

Name	Age	Sex
A	20	M
C	21	M
B	21	F

R2

Name	Age	Sex
D	20	F
A	20	M
E	21	F

R3= R1 ∩ R2

Name	Age	Sex
A	20	M

R1

Name	Age	Sex
A	20	M
C	21	M
B	21	F

R2

Name	Age	Sex
D	20	F
A	20	M
E	21	F

R3= R1 U R2

Name	Age	Sex
A	20	M
C	21	M
B	21	F
D	20	F
E	21	F

R1

Name	Age	Sex
A	20	M
C	21	M
B	21	F

R2

Name	Age	Sex
D	20	F
A	20	M
E	21	F

R3= R1 - R2

Name	Age	Sex
C	21	M
B	21	F

R3= R2 – R1

Name	Age	Sex
D	20	F
E	21	F

Acts

actor	title
Cage	Raising Arizona
Maguire	Spiderman
Maguire	Wonder Boys
McDormand	Fargo
McDormand	Raising Arizona
McDormand	Wonder Boys

Movies

title	director	myear	rating
Fargo	Coen	1996	8.2
Raising Arizona	Coen	1987	7.6
Spiderman	Raimi	2002	7.4
Wonder Boys	Hanson	2000	7.6

e_1

title
Fargo
Raising Arizona
Wonder Boys

e_2

title
Fargo
Raising Arizona

$e_1 \cap e_2$

title
Fargo
Raising Arizona

Intersection Example:

Find Coen's movies with McDormand

$e_1 = \pi_{title}(\sigma_{actor='McDormand'}(Acts))$

$e_2 = \pi_{title}(\sigma_{director='Coen'}(Movies))$

$\mathbf{result} = e_1 \cap e_2$

3. Cross-Product:

- Consider R(A,B,C) and S(X, Y), Cross-product: R × S returns a relation with attribute list (A,B,C,X, Y) defined as follows:

$R \times S = \{(a, b, c, x, y) \mid (a, b, c) \in R, (x, y) \in S\}$

- Cross-product operation is also known as Cartesian product
- Fields of the same name are may renamed

R1

Name	Age	Sex
A	20	M
C	21	M

R2

Name	Age	Sex
D	20	F
E	21	F

R3= R1 X R2

Name	Age	Sex	Name	Age	Sex
A	20	M	D	20	F
C	21	M	D	20	F
A	20	M	E	21	F
C	21	M	E	21	F

4 .Join

Join can be defined as cross-product followed by selection and projection.

- We have Several variants of join:-
- **Condition joins**

Condition Join: $R \bowtie_c S$

- **Condition join** = Cross-product followed by selection

$$R \bowtie_c S = \sigma_c(R \times S)$$

- **Example**: Find (director,actor) pairs where the director is younger than the actor
- **Answer**: $\pi_{director,actor}$(Directors $\bowtie_{dyear>ayear}$ Actors)

Directors

director	dyear
Coen	1954
Hanson	1945
Raimi	1959

Actors

actor	ayear
Cage	1964
Hanks	1956
Maguire	1975
McDormand	1957

e_1 = Directors $\bowtie_{dyear>ayear}$ Actors

director	dyear	actor	ayear
Raimi	1959	Hanks	1956
Raimi	1959	McDormand	1957

$\pi_{director,actor}(e_1)$

director	actor
Raimi	Hanks
Raimi	McDormand

Equi join

A special case of condition join where the condition **c** contains only **equalities Condition**.

Result schema similar to cross-product, but only one copy of fields for which equality is specified

- **Example**: Find actors who have acted in some Coen's movie
- $\pi_{actor}(\sigma_{director='Coen'} (\text{Acts} \bowtie_{Acts.title = Movies.title} \text{Movies}))$

$e_1 = \text{Acts} \bowtie_{Acts.title = Movies.title} \text{Movies}$

actor	title	director	myear	rating
Cage	Raising Arizona	Coen	1987	7.6
Maguire	Spiderman	Raimi	2002	7.4
Maguire	Wonder Boys	Hanson	2000	7.6
McDormand	Fargo	Coen	1996	8.2
McDormand	Raising Arizona	Coen	1987	7.6
McDormand	Wonder Boys	Hanson	2000	7.6

$\pi_{actor}(\sigma_{director='Coen'}((e_1))$

actor
Cage
McDormand

Movies

title	director	myear	rating
Fargo	Coen	1996	8.2
Raising Arizona	Coen	1987	7.6
Spiderman	Raimi	2002	7.4
Wonder Boys	Hanson	2000	7.6

Acts

actor	title
Cage	Raising Arizona
Maguire	Spiderman
Maguire	Wonder Boys
McDormand	Fargo
McDormand	Raising Arizona
McDormand	Wonder Boys

Natural Join

Natural Join is Equijoin on all common fields. It is an equijoin in which equalities are specified on all fields having the same name in R and S.

- We can then omit the join condition.
- Result is guaranteed not to have two fields with the same name.
- If no fields in common, then natural join is simply cross product

Example: Condition, Equi-, Natural Joins

R

A	B	X
0	6	x_1
1	9	x_2
2	7	x_3

S

A	B	Y
0	8	y_1
1	5	y_2
2	7	y_3

- $R \bowtie_{A=A' \wedge B<B'} \rho_{S'(A',B',Y)}(S)$

A	B	X	A'	B'	Y
0	6	x_1	0	8	y_1

- $R \bowtie_{A=A'} \rho_{S'(A',B',Y)}(S)$

A	B	X	B'	Y
0	6	x_1	8	y_1
1	9	x_2	5	y_2
2	7	x_3	7	y_3

- $R \bowtie S$

A	B	X	Y
2	7	x_3	y_3

5 Division

- Suppose A has two groups of fields <x,y> y fields are same fields in terms of domain as B. A/B = <x> such as for every y value in a tuple of B there is <x,y> in A.
- Another way to understand division is as follows. For each x value in (the first column of) A, consider the set of y values that appear in (the second field of) tuples of A with that x value. If this set contains (all y values in) B, the x value is in the result of A/B

sno	pno
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

A

pno
p2

B1

pno
p2
p4

B2

pno
p1
p2
p4

B3

sno
s1
s2
s3
s4

A/B1

sno
s1
s4

A/B2

sno
s1

A/B3