

## Database

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

### 1. 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, dBASE
- 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.

### 2. Classification of DBMS

Database management systems can be classified based on several criteria, such as the data model, user numbers and database distribution.

#### 2.1 Classification Based on Data Model

A database model is a type of data model that determines the logical structure of a database and fundamentally determines in which manner data can be stored, organized, and manipulated.

**Types of DB models:**

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

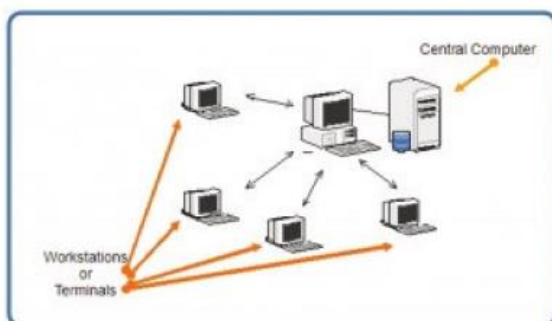
**2.2 Classification Based on User Numbers**

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.

**2.3 Classification Based on Database Distribution**

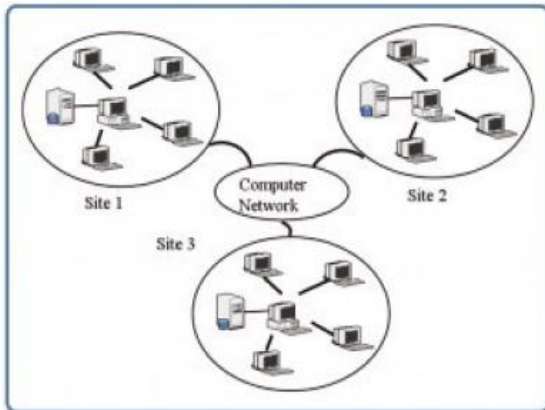
There are four main distribution systems for database systems and these, in turn, can be used to classify the DBMS.

**2.3.1 Centralized systems:** With a centralized database system, the DBMS and database are stored at a single site that is used by several other systems too. Fig(1)



Fig(1) centralized DB

**2.3.2 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 as shown in Fig(2)

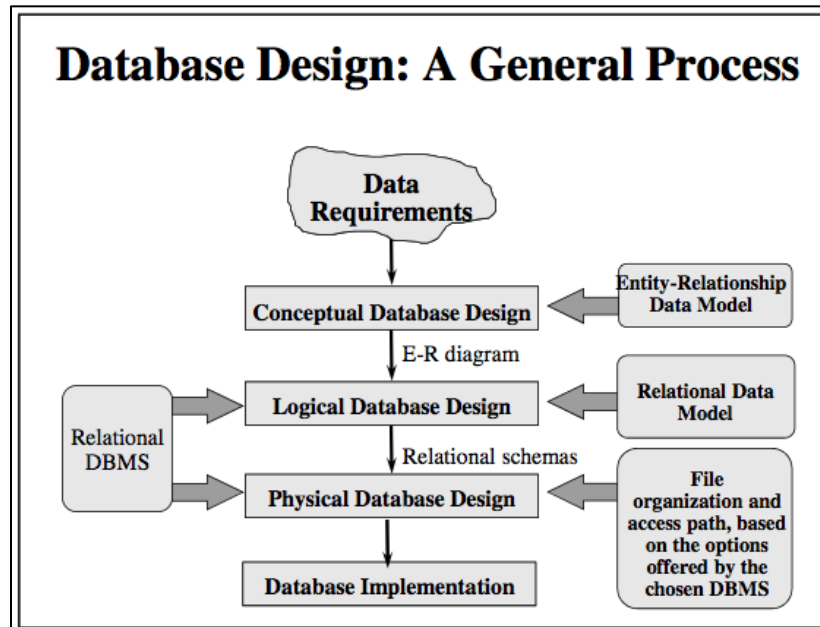


Fig(2) distributed DB

- **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.

### 3. 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?



### 3.1 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.

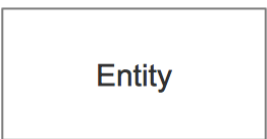
4. Schema Refinement: The fourth step with DB's design is to analyze the collection of relations in our relational DB schema to identify potential problems, and to refine it (Normalization).
5. Physical Database Design: It describes the details of how data is stored. This step may simply involve building 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.

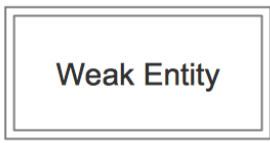
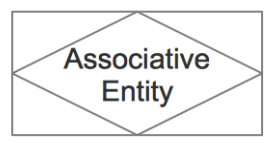

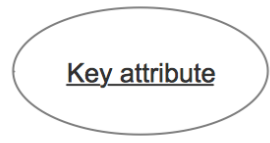



#### 4. The Entity Relationship Mode

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	Shape Name	Symbol Description
<b>Entities</b>		
	Entity	An entity is an object in the real world with an independent existence that can be differentiated from other objects. An entity is represented by a rectangle which contains the entity's name.

	Weak Entity	An entity that cannot be uniquely identified by its attributes alone. The existence of a weak entity is dependent upon another entity called the owner entity.
	Associative Entity	An entity used in a many-to-many relationship (represents an extra table). All relationships for the associative entity should be many
<b>Attributes</b>		
	Attribute	An attribute is a particular property that describes the entity. each attribute is represented by an oval containing attribute's name
	Key attribute	An attribute that uniquely identifies a particular entity. The name of a key attribute is underscored.
	Multivalued attribute	An attribute that can have many values (there are many distinct values entered for it in the same column of the table). Multivalued attribute is depicted by a dual oval.
	Derived attribute	An attribute whose value is calculated (derived) from other attributes. The derived attribute may not be physically stored in the database. This attribute is represented by dashed oval.
<b>Relationships</b>		
	Relationship	A relationship among two or more entities

## 4.1 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:-

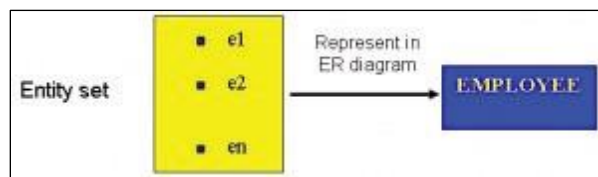
**-Weak entity:** an entity is considered **weak** if its tables are existence dependent.

- 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

An **entity set** is a collection of entities of an entity type at a particular point of time. In an entity relationship diagram (ERD), an entity type is represented by a name in a box.



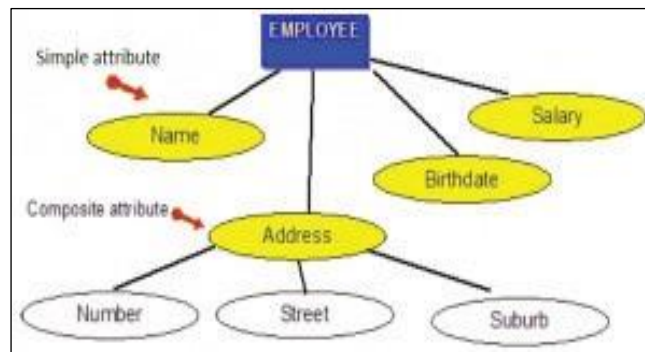
## 4.2 Entity and attributes

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 may have values for each of its attributes.

### Types Of Attributes

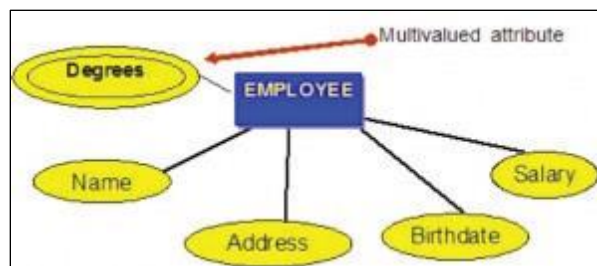
Several types of attributes occur in the ER model: **simple** versus **composite**, **single** valued versus **multivalued**, and **stored** versus **derived**.

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' }



Fig(3) Address is a Composite attribute

3. **Multivalued attributes:** Are attributes that have a set of values for each entity. See Figure 4, are the degrees of an employee: BSc, MIT, PhD.



Fig(4) Degree is a multi-valued attribute

4. **Derived attributes:** Are attributes that contain values calculated from other attributes. Figure 5 Age can be derived from the attribute Birthdate. In this situation, Birthdate is called a stored attribute, which is physically saved to the database



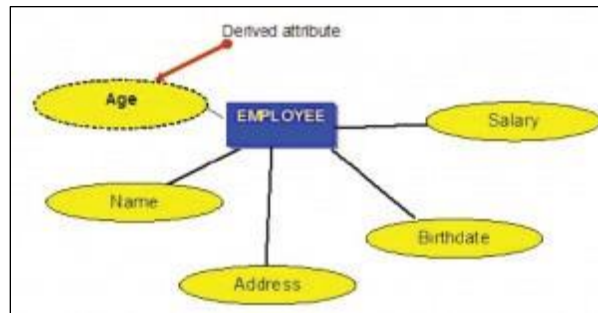


Fig (5) Age is a derived attribute, name, birthday, salary, address are stored attributes

### 4.3 NULL Values

A null is a special symbol, independent of data type, which means either unknown or inapplicable. It does not mean zero or blank. Features of null include:

- No data entry
- Not permitted in the primary key
- Should be avoided in other attributes

Null Can represent:-

- An unknown attribute value.
  - A known, but missing, attribute value.
  - A not “applicable” condition.
- Can create problems when functions such as COUNT, AVERAGE and SUM are used

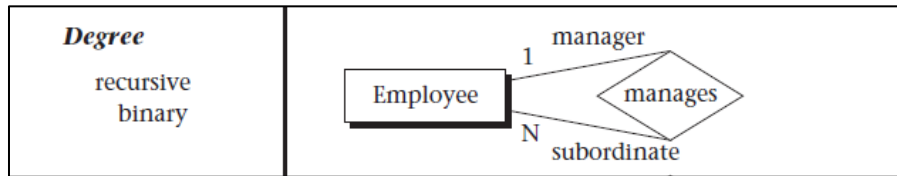
### 4.4 Relationship

Relationships are the glue that holds the tables together. They are used to connect related information between tables.

**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.

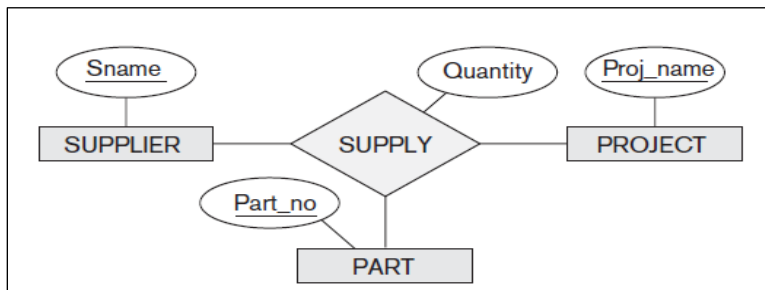
The **binary** relationship, an association between two entities, is by far the most common type in the natural world. In fact, many modeling systems use only this type.

**Unary relationship (recursive):** A unary relationship, also called recursive, is one in which a relationship exists between occurrences of the same entity set. In this relationship, the primary and foreign keys are the same, but they represent two entities with different roles fig(7).



Fig(7) recursive relationship

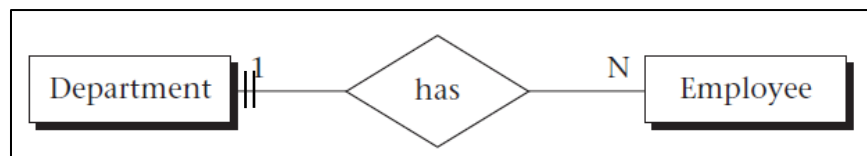
**Ternary Relationships:** A ternary relationship is a relationship type that involves many to many relationships between three tables. Refer to Figure (8) for an example of mapping a ternary relationship type. Note n-ary means multiple tables in relationship.



Fig(8) A ternary relationship

**Connectivity of a Relationship** The connectivity of a relationship describes a constraint on the connection of the associated entity occurrences in the relationship. Values for connectivity are either “one” or “many.”




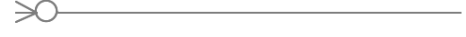
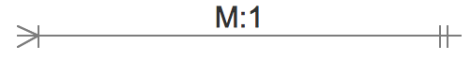
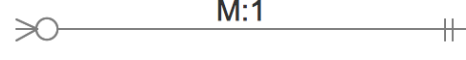
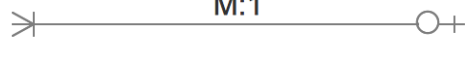


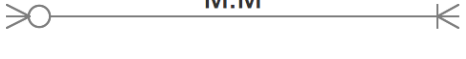



Example: for a relationship between the entities Department and Employee, a connectivity of one for Department and many for Employee means that there is at most one entity occurrence of Department associated with many occurrences of Employee.



The actual count of elements associated with the connectivity is called the **cardinality of the relationship**.

## Relationship Connectivity

Table (2) Relational Symbol and Meaning

Symbol	Meaning
<b>Relationships (Cardinality and Modality)</b>	
	Zero or One
	One or More
	One and only One
	Zero or More
<b>Many - to - One</b>	
	a one through many notation on one side of a relationship and a one and only one on the other
	a zero through many notation on one side of a relationship and a one and only one on the other
	a one through many notation on one side of a relationship and a zero or one notation on the other
	a zero through many notation on one side of a relationship and a zero or one notation on the other
<b>Many - to - Many</b>	
	a zero through many on both sides of a relationship
	a zero through many on one side and a one through many on the other
	a one through many on both sides of a relationship
	a one and only one notation on one side of a relationship and a zero or one on the other
	a one and only one notation on both sides

## 4.5 How to Convert ER Diagram to Relational Database

We will be following the simple rules:

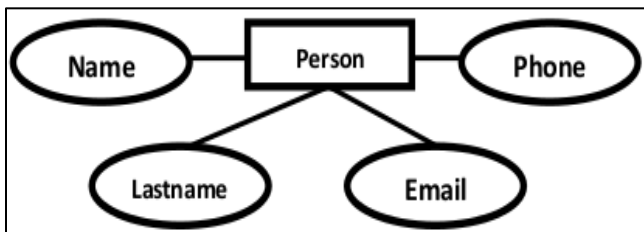
### 1. Entities and Simple Attributes:

- 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.

Example:

Persons ( personid, name, lastname, email)

Note the phone attribute not included.



### 2. Multi-Valued Attributes

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

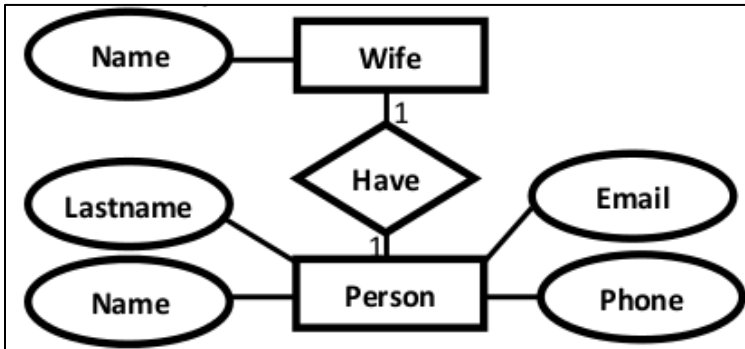
If you have a multi-valued attribute, take the attribute and turn it into a new entity or table of its own. Then make a 1: N relationship between the new entity and the existing one. In simple words:

- 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 , **wifeid** )

Wife ( wifeid , name )

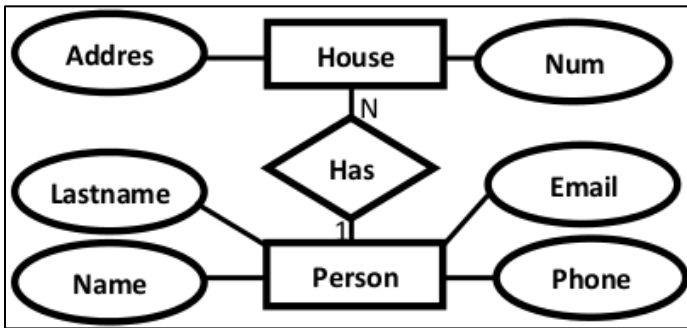
Or vice versa to put the personid as a foreign key within the Wife table as shown below:

Persons ( personid, name, lastname, email )

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:



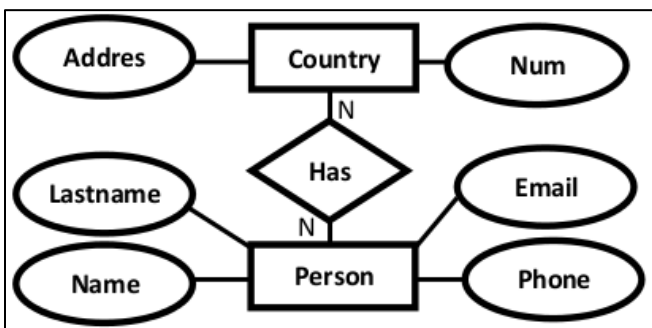
It should convert to:

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 :

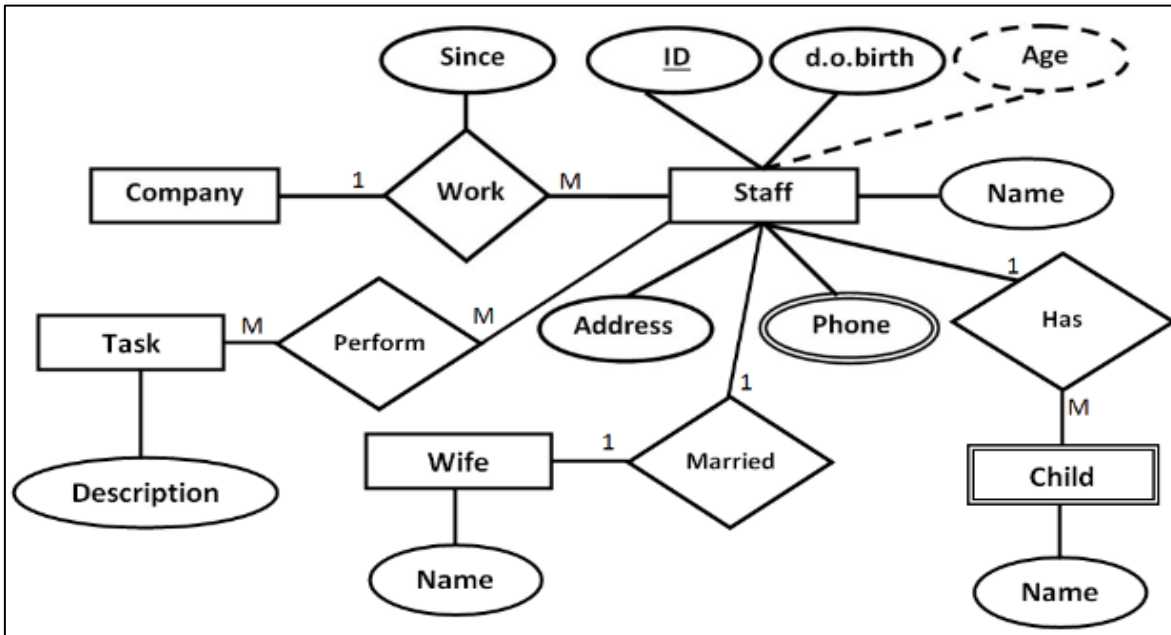
Persons( personid , name, lastname, email )

Countries ( countryid , name, code)

HasRelat ( personid + countryid )

## Case Study

Convert the E-R diagram into relational database



The relational schema for the ER Diagram is given below as:

Company( CompanyID )

Staff( ID , dob , name, address , **companyID**, **WifeID** )

Child( ChildID , name , **ID** )

Wife ( WifeID , name )

Phone(PhoneID , phoneNumber , **ID**)

Task ( TaskID , description)

Perform(**ID** + **TaskID**)

#### 4.6 Key Differences between E-R Model and Relational Model

BASIS FOR COMPARISON	E-R MODEL	RELATIONAL MODEL
Basic	It represents the collection of objects called entities and relation between those entities.	It represents the collection of Tables and the relation between those tables.
Describe	Entity Relationship Model describe data as Entity set, Relationship set and Attribute.	Relational Model describes data in a table as Domain, Attributes, Tuples.
Relationship	E-R Model is easier to understand the relationship between entities.	Comparatively, it is less easy to derive a relation between tables in Relational Model.
Mapping	E-R Model describes Mapping Cardinalities.	Relational Model does not describe mapping cardinalities.

### 5. Enhanced Entity Relationship Model (EER Model)

EER is a high-level data model that incorporates the extensions to the original ER model. It includes all modeling concepts of basic ER and additional:

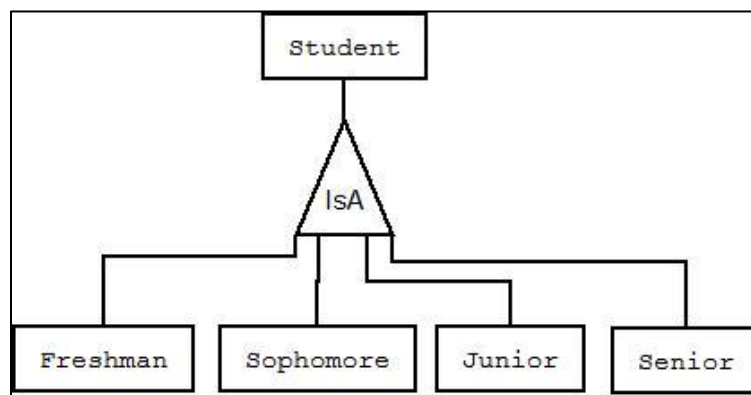
- Sub Class and Super Class
- Specialization and Generalization
- Union or Category
- Aggregation



## 5.1 Sub class-super class

One entity type might be a subtype of another; very similar to subclasses in OO programming

- Example:
  - EMPLOYEE may be further grouped into SECRETARY, ENGINEER, MANAGER, TECHNICIAN....
  - VEHICLE may be grouped into CAR, TRUCK, VAN, ...
  - Each of these groupings is a subset of EMPLOYEE entities and is called a **subclass** of EMPLOYEE
  - EMPLOYEE is the **superclass** for each of these subclasses
  - These are called **superclass/subclass relationships**.
  - These are also called IS-A relationships (SECRETARY IS-A EMPLOYEE, TECHNICIAN IS-A EMPLOYEE, ...). -

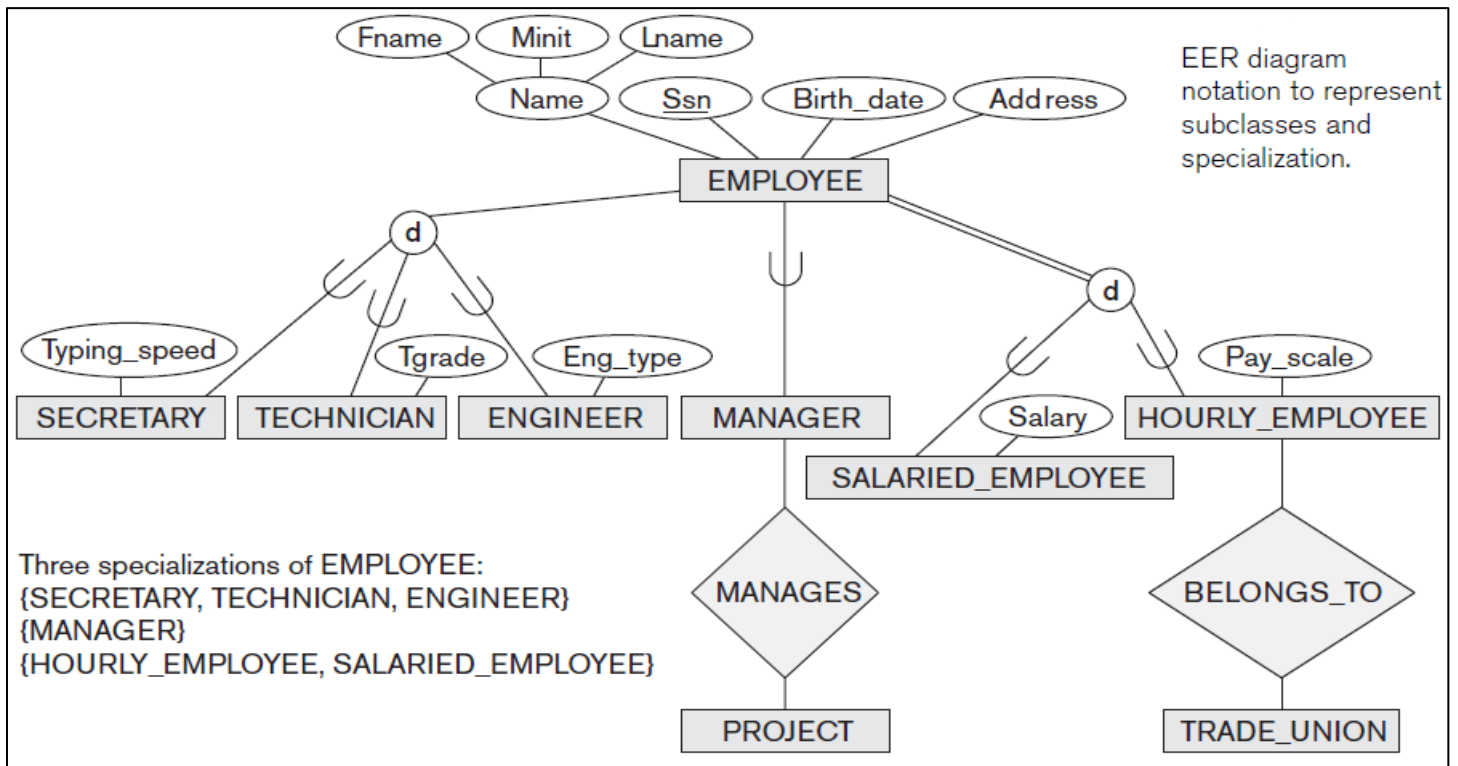


- An entity that is member of a subclass **inherits** all attributes of the entity as a member of the superclass
- It also inherits all relationships

## 5.2 Specialization

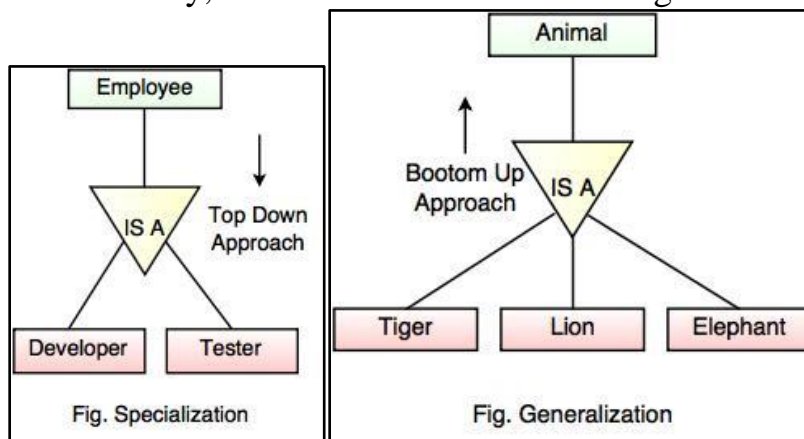
- Is the process of defining a set of subclasses of a superclass
- The set of subclasses is based upon some distinguishing characteristics of the entities in the superclass
- Example: {SECRETARY, ENGINEER, TECHNICIAN} is a specialization of EMPLOYEE based upon job type.
- May have several specializations of the same superclass
- Example: Another specialization of EMPLOYEE based in method of pay is {SALARIED\_EMPLOYEE, HOURLY\_EMPLOYEE}.
- Superclass/subclass relationships and specialization can be diagrammatically represented in EER diagrams
- Attributes of a subclass are called specific attributes. For example, Typing Speed of SECRETARY

- The subclass can participate in specific relationship types. For example, BELONGS\_TO of HOURLY\_EMPLOYEE.



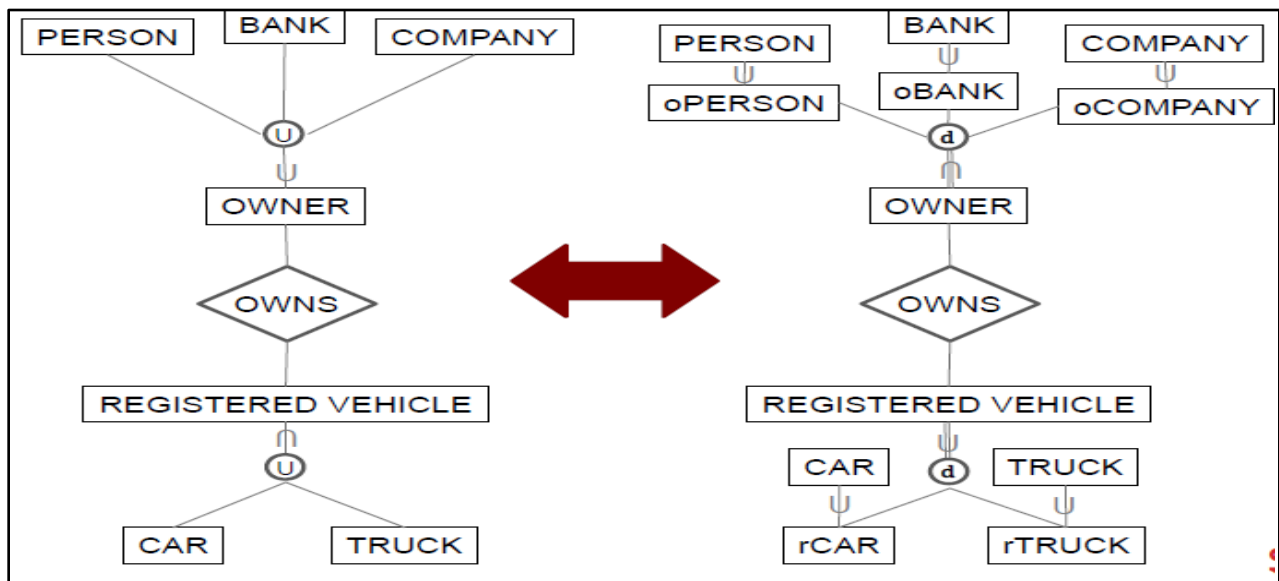
### 5.3 Generalization

- The reverse of the specialization process
- Several classes with common features are generalized into a superclass; original classes become its subclasses
- Example: CAR, TRUCK generalized into VEHICLE; both CAR, TRUCK become subclasses of the superclass VEHICLE.
- We can view {CAR, TRUCK} as a specialization of VEHICLE
- Alternatively, we can view VEHICLE as a generalization of CAR and TRUCK



### 5.4 Category or Union

- Category represents a single super class or sub class relationship with more than one super class.
- **For example** Car booking, Car owner can be a person, a bank (holds a possession on a Car) or a company. Category (sub class) → Owner is a subset of the union of the three super classes → Company, Bank, and Person. A Category member must exist in at least one of its super classes.



REWRITING UNION AS SPECIALIZATION

### 5.5 Aggregation

- Aggregation is a process that represent a relationship between a whole object and its component parts.
- It abstracts a relationship between objects and viewing the relationship as an object.
- It is a process when two entity is treated as a single entity.

In the following example, the relation between College and Course is acting as an Entity in Relation with Student.

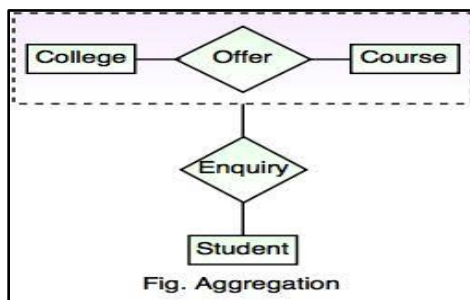
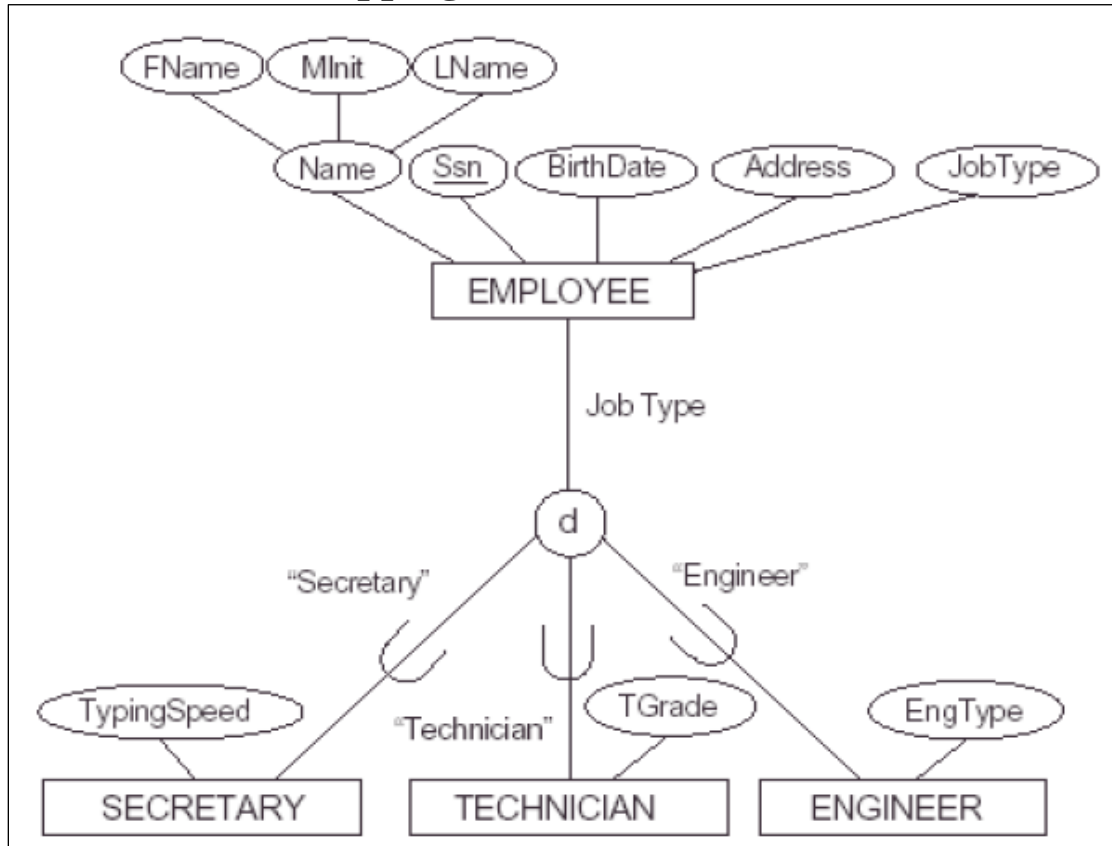


Fig. Aggregation

## 5.6 EER to Relational Mapping



Fig( 9) ER schema diagram specialization on job title

To convert each **super-class/subclass** relationship into a relational schema you must use one of the four options available.

Let  $C$  be the super-class,  $K$  its primary key and  $A_1, A_2, \dots, A_n$  its remaining attributes and let  $S_1, S_2, \dots, S_m$  be the sub-classes.

### Option A (multiple relation option):

- Create a relation  $L$  for  $C$  with attributes

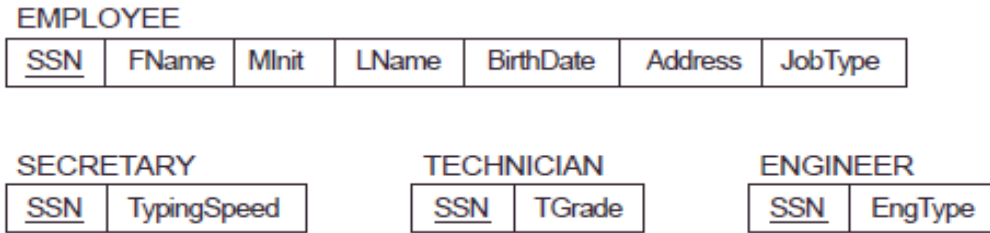
$$(L) = \{K, A_1, A_2, \dots, A_n\} \text{ and } PK(L) = K.$$

- Create a relation  $L_i$  for each subclass  $S_i$ ,  $1 < i < m$ , with the attributes

$$(L_i) = \{K\} \cup \{\text{attributes of } S_i\} \text{ and } PK(L_i) = K.$$

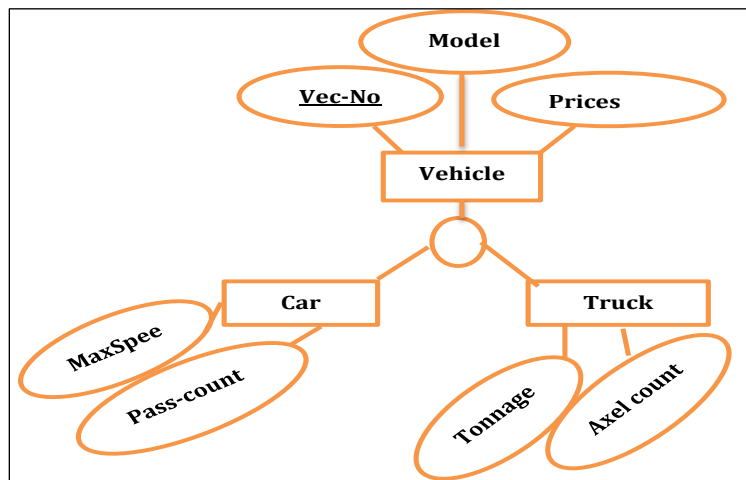
- This option works for any constraints: disjoint or overlapping; total or partial.

Mapping the EER diagram on fig(9) using option A



**Option B (multiple relation option):**

- Create a relation  $L_i$  for each subclass  $S_i$ ,  $1 < i < m$ , with  $(L_i) = \{\text{attributes of } S_i\} \cup \{K, A_1, A_2, \dots, A_n\}$   $PK(L_i) = K$
- This option works well only for disjoint and total constraints.
- If not disjoint, redundant values for inherited attributes.



Fig(10)

**Car**

MaxSpeed	Pass-count	Model	Price	<u>Vec-no</u>
----------	------------	-------	-------	---------------

**Truck**

Tonnage	Axel count	Model	Price	<u>Vec-no</u>
---------	------------	-------	-------	---------------

**Option c (Single Relation Option)**

- Create a single relation  $L$  with attributes  $(L) = \{K, A_1, \dots, A_n\} \cup \{\text{attributes of } S_1\} \cup \dots \cup \{\text{attributes of } S_m\} \cup \{T\}$  and  $PK(L)=K$

- This option is for specialization whose subclasses are DISJOINT, and T is a **type** attribute that indicates the subclass to which each tuple belongs, if any. This option may generate a large number of null values.
- Not recommended if many specific attributes are defined in subclasses (will result in many null values!)

**Employee**

<u>Ssn</u>	Fname	Minit	Lname	birthdate	Address	JobType	typpigSpeed	TGrad	EngType
------------	-------	-------	-------	-----------	---------	---------	-------------	-------	---------

**Option d (Single Relation Option)**

- Create a single relation schema L with attributes  
 $(L) = \{K, A1, \dots, An\} \cup \{\text{attributes of } S1\} \cup \dots \cup \{\text{attributes of } Sm\} \cup \{T1, \dots, Tn\}$  and  
 $PK(L)=K$
- This option is for specialization whose subclasses are overlapping, and each  $Ti, 1 < i < m$ , is a Boolean attribute indicating whether a tuple belongs to subclass  $Si$ .
- This option could be used for disjoint subclasses too.

**Vehicle**

Vec_no	Model	Price	Maxspee	Pass-count	carF	tonnage	Axel count	truckF
--------	-------	-------	---------	------------	------	---------	------------	--------

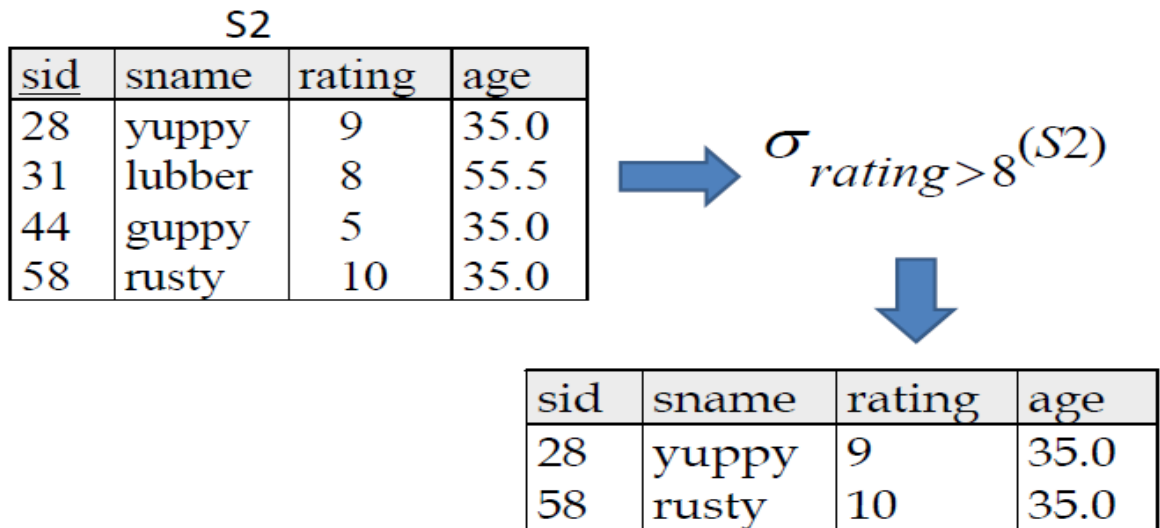
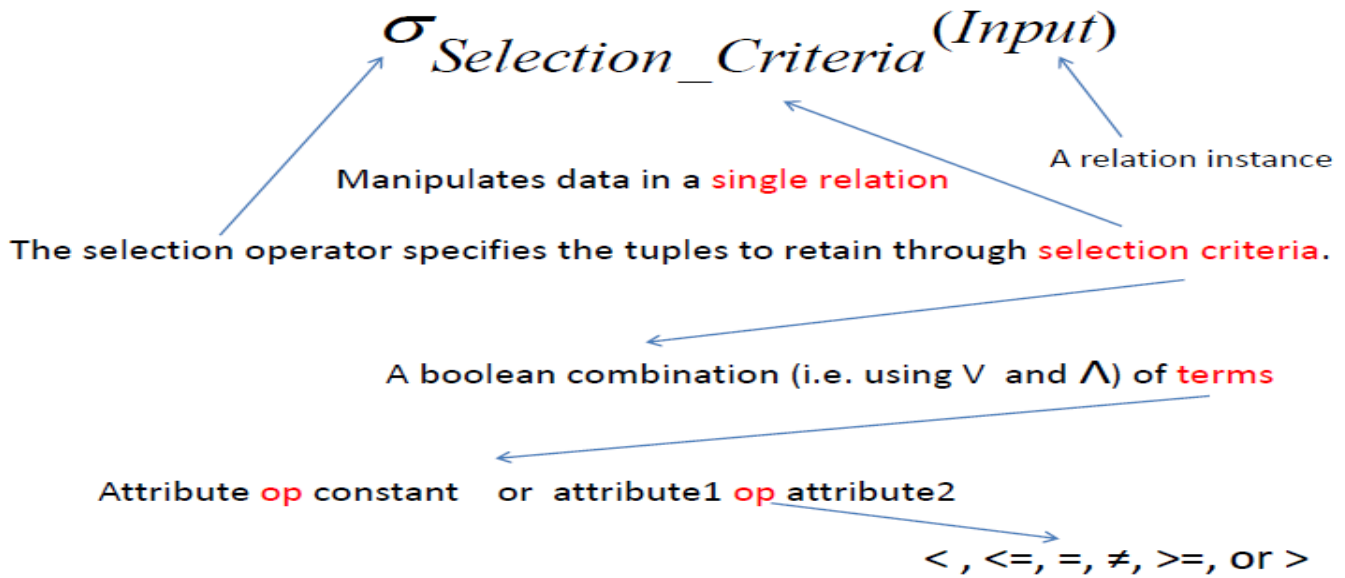
**6. Relational algebra**

Relational algebra 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.

**6.1 Selection:** picking certain rows.



## 6.2 Projection:

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

Allows us to extract **columns** from a **relation**

Example:

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

$$\Rightarrow \pi_{age}(S2) \Rightarrow$$

age
35.0
55.5

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

$$\Rightarrow \pi_{sname, rating}(\sigma_{rating > 8}(S2))$$

sname	rating
yuppy	9
rusty	10



## Set Operations

- Takes as input two relation instances, four standard operations
  - Union
  - Intersection
  - Difference
  - Cross-product

**Note:** Union, intersection, and difference require the two input set to be **union compatible** – They have the same number of fields – corresponding fields, taken in order from left to right, have the same domains

### 6.3 Union

**$R \cup S$**  returns relation instance containing all tuples that occur in either relation instance R or S, or both.

- R and S must be union compatible.
- Schema of the result is defined to be that of R.
- Remove the duplicated tuples.

R1			R2		
Name	Age	Sex	Name	Age	Sex
A	20	M	D	20	F
C	21	M	A	20	M
B	21	F	E	21	F

$R3 = R1 \cup R2$		
Name	Age	Sex
A	20	M
C	21	M
B	21	F
D	20	F
E	21	F

### 6.4 Intersection

**$R \cap S$** : returns a relation instance containing all tuples that occur in both R and S.

- R and S must be union compatible.
- Schema of the result is that of R.

R1			R2		
Name	Age	Sex	Name	Age	Sex
A	20	M	D	20	F
C	21	M	A	20	M
B	21	F	E	21	F

$R3 = R1 \cap R2$

Name	Age	Sex
A	20	M

### 6.5 Difference

**R – S:** returns a relation instance containing all tuples that occur in R but not in S.

- R and S must be union-compatible.
- Scheme of the result is the schema of R.

R1			R2		
Name	Age	Sex	Name	Age	Sex
A	20	M	D	20	F
C	21	M	A	20	M
B	21	F	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

### 6.6 Cross-Product

**R x S:** Returns a relation instance whose scheme contains:

- All the fields of R (in the same order as they appear in R)
- All the fields of S (in the same order as they appear in S)
- The result contains one tuple  $\langle r,s \rangle$  for each pair with  $r \in R$  and  $s \in S$
- Basically, it is the **Cartesian product**.
- **Fields of the same name are may renamed**

R1			R2		
Name	Age	Sex	Name	Age	Sex
A	20	M	D	20	F
C	21	M	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

### 6.7 Join

Join can be defined as cross-product followed by selection and projection. We have several variants of join.

- Condition joins
- Equijoin
- Natural join

#### 6.7.1 Condition Joins:

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

Example:  $S1 \bowtie_{S1.sid < R1.sid} R1$

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	58	103	11/12/96



### 6.7.2 Equijoin

Condition consists only of **equalities** connected by  $\wedge$

- Redundancy in retaining both attributes in result
- So, an additional projection is applied to remove the second attribute.

**Example:**

$$S1 \bowtie_{Rsid = Sid} R1$$

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96



sid	sname	rating	age	bid	day
22	dustin	7	45.0	101	10/10/96
58	rusty	10	35.0	103	11/12/96

### 6.7.3 Natural Join

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.

Relations r, s:

A	B	C	D
$\alpha$	1	$\alpha$	a
$\beta$	2	$\gamma$	a
$\gamma$	4	$\beta$	b
$\alpha$	1	$\gamma$	a
$\delta$	2	$\beta$	b

r

B	D	E
1	a	$\alpha$
3	a	$\beta$
1	a	$\gamma$
2	b	$\delta$
3	b	$\epsilon$

s

$r \bowtie s$

A	B	C	D	E
$\alpha$	1	$\alpha$	a	$\alpha$
$\alpha$	1	$\alpha$	a	$\gamma$
$\alpha$	1	$\gamma$	a	$\alpha$
$\alpha$	1	$\gamma$	a	$\gamma$
$\delta$	2	$\beta$	b	$\delta$

### 6.8 Division

Suppose A has two groups of fields <x,y> y fields are same fields in terms of domain as B.

$A/B = \langle x \rangle$  such as for every y value in a tuple of B there is <x,y> in A.

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

### Combining Operations:

Since the result of a relational algebra operation is also a relation, it can act as input to another Algebra operation.

- For instance, to compute the exercises solved by student 102:

$$\pi_{CAT, ENO}(\sigma_{SID=102}(RESULTS))$$

- An intermediate result can be stored in a temporary relation.

$$S102 := \sigma_{SID=102}(RESULTS);$$

$$\pi_{CAT, ENO}(S102)$$

### 7. Relational calculus

*Relational calculus* represents an alternative to relational algebra as a candidate for the manipulative part of the relational data model. The difference between the two is as follows:

BASIS FOR COMPARISON	RELATIONAL ALGEBRA	RELATIONAL CALCULUS
Basic language	Relational Algebra is a Procedural	Relational Claculus is Declarative language
States	Relational Algebra states how to obtain the result.	Relational Calculus states what result we have to obtain.
Order	Relational Algebra describes the order in which operations have to be performed	Relational Calculus does not specify the order of operations
Domain	Relational Algebra is not domain dependent.	Relation Claculus can be domain dependent.
Related	It is close to a programming language.	It is close to the natural language.

For example, consider the query “*Get owners’ complete names and cities for owners who own a red car.*”

An algebraic version of this query could be:

- Join relation **OWNERS** with relation **CARS** on **IdentificationNumber** attributes
- Select from the resulting relation only those tuples with car **Colour = "RED"**
- Project the result of that restriction on owner **FirstName, LastName** and **City**

A calculus formulation, by contrast, might look like:

- Get **FirstName, LastName** and **City** for cars owners such that there exists a car with the same **IdentificationNumber** and with **RED** color.