UNIT-III

Entity Relationship Model: Introduction, Representation of entities, attributes, entity set, relationship, relationship set, constraints, sub classes, super class, inheritance, specialization, generalization using ER Diagrams. SQL: Creating tables with relationship, implementation of key and integrity constraints, nested queries, sub queries, grouping, aggregation, ordering, implementation of different types of joins, view(updatable and non-updatable), relational set operations.

What is ER Diagrams

Entity relationship diagram displays the relationships of entity set stored in a database. In other words, we can say that ER diagrams help you to explain the logical structure of databases. At first look, an ER diagram looks very similar to the flowchart. However, ER Diagram includes many specialized symbols, and its meanings make this model unique.

Facts about ER Diagram Model:

- ER model allows you to draw Database Design
- It is an easy to use graphical tool for modeling data
- Widely used in Database Design
- It is a GUI representation of the logical structure of a Database
- It helps you to identifies the entities which exist in a system and the relationships between those entities

Why use ER Diagrams?

Here, are prime reasons for using the ER Diagram

- Helps you to define terms related to entity relationship modeling
- Provide a preview of how all your tables should connect, what fields are going to be on each table

- Helps to describe entities, attributes, relationships
- ER diagrams are translatable into relational tables which allows you to build databases quickly
- ER diagrams can be used by database designers as a blueprint for implementing data in specific software applications
- The database designer gains a better understanding of the information to be contained in the database with the help of ERP diagram
- ERD is allowed you to communicate with the logical structure of the database to users

Components of the ER Diagram

This model is based on three basic concepts:

1. Entities
2. Attributes
3. Relationships

Entity

An **Entity** is a real-world object that are represented in database. It can be any object,place,person or class.Data are stored about such entities.

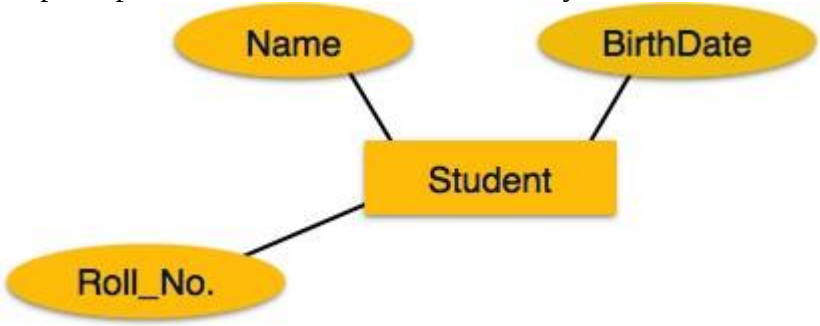
Examples of entities:

Person: Employee, Student, Patient
Place: Store, Building
Object: Machine, product, and Car
Event: Sale, Registration, Renewal
Concept: Account, Course

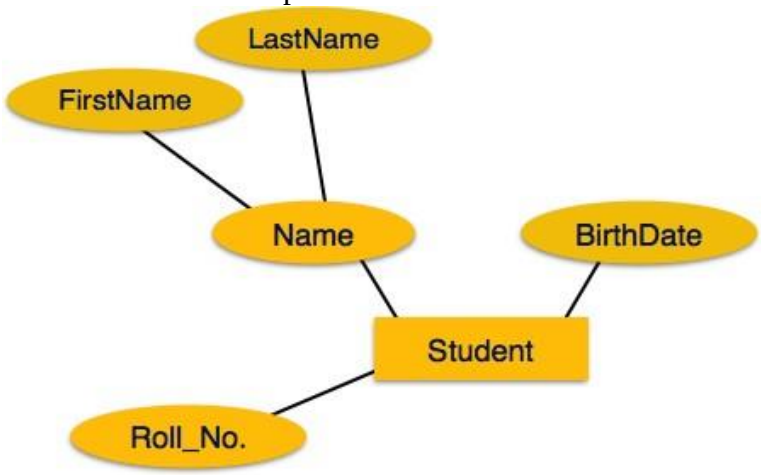
Entities are represented by means of rectangles. Rectangles are named with the entity set they represent.

Attributes

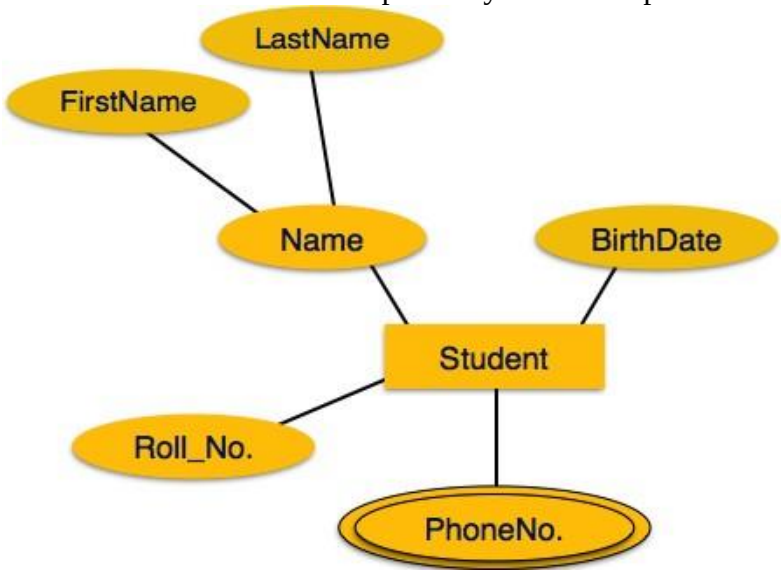
Attributes are the properties of entities. Attributes are represented by means of ellipses. Every ellipse represents one attribute and is directly connected to its entity (rectangle).



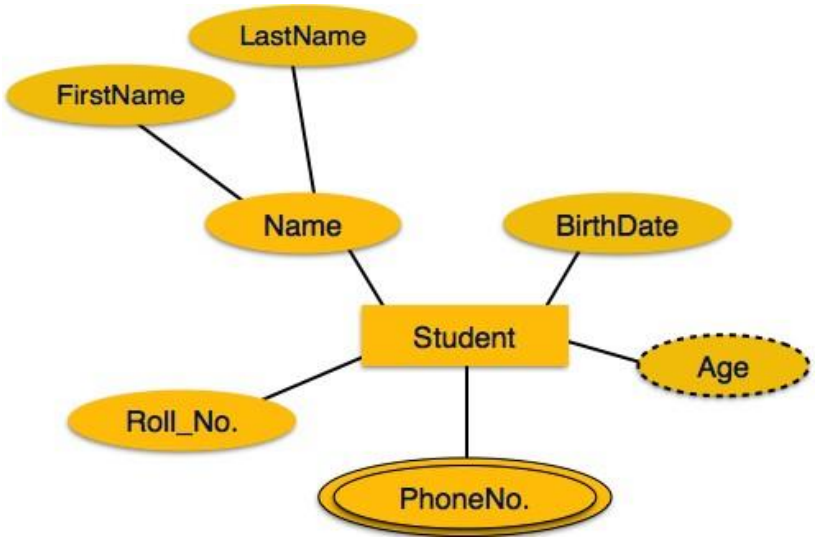
If the attributes are **composite**, they are further divided in a tree like structure. Every node is then connected to its attribute. That is, composite attributes are represented by ellipses that are connected with an ellipse.



Multivalued attributes are depicted by double ellipse.



Derived attributes are depicted by dashed ellipse.



Relationship

Relationship is nothing but an association among two or more entities. E.g., Tom works in the Chemistry department.

Example-

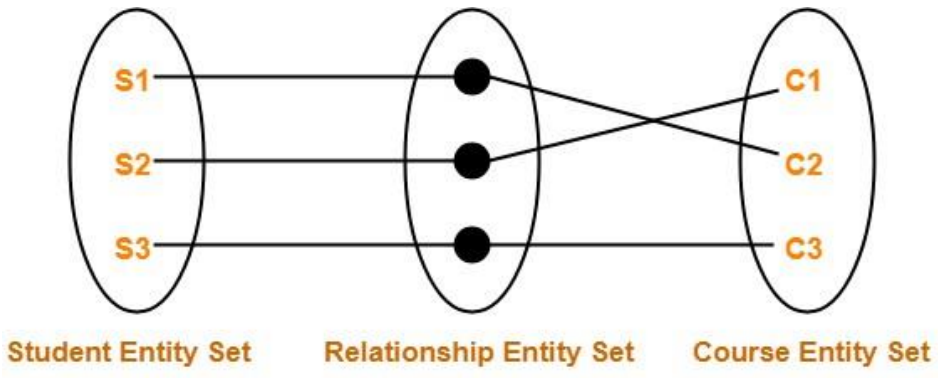
'Enrolled in' is a relationship that exists between entities **Student** and **Course**.

Entities take part in relationships. We can often identify relationships with verbs or verb phrases. Relationships are represented by diamond-shaped box. Name of the relationship is written inside the diamond-box. All the entities (rectangles) participating in a relationship, are connected to it by a line.

Relationship Set-

A relationship set is a set of relationships of same type.

Example- Set representation of above ER diagram is-



Set Representation of ER Diagram

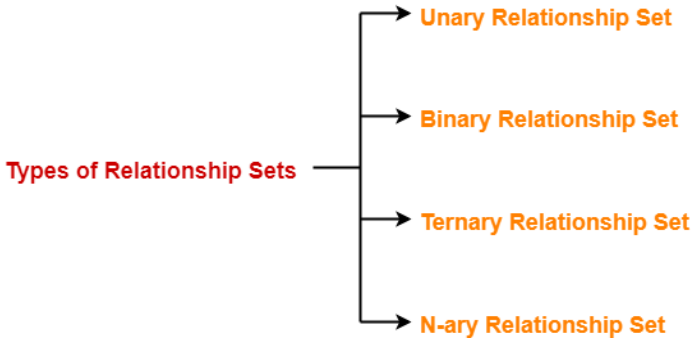
Degree of a Relationship Set-

The number of entity sets that participate in a relationship set is termed as the degree of that relationship set. Thus,

Degree of a relationship set = Number of entity sets participating in a relationship set

Types of Relationship Sets-

On the basis of degree of a relationship set, a relationship set can be classified into the following types-



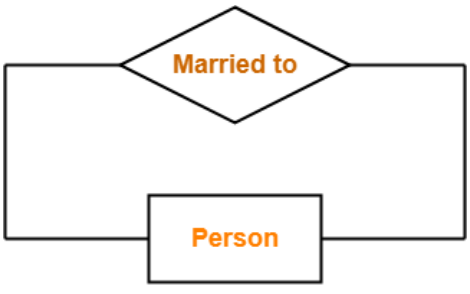
1. Unary relationship set
2. Binary relationship set
3. Ternary relationship set
4. N-ary relationship set

1. Unary Relationship Set-

Unary relationship set is a relationship set where only one entity set participates in a relationship set.

Example-

One person is married to only one person



Unary Relationship Set

2. Binary Relationship Set-

Binary relationship set is a relationship set where two entity sets participate in a relationship set.

Example-

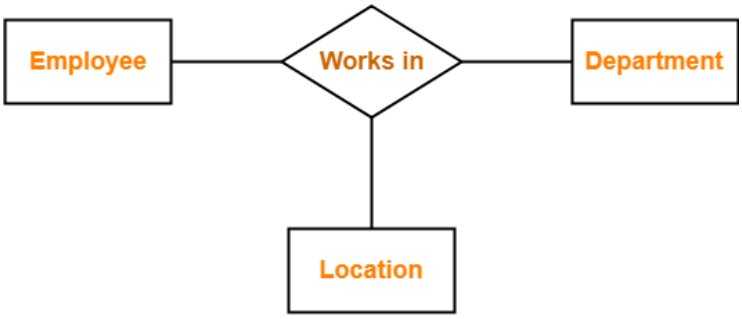
Student is enrolled in a Course

Binary Relationship Set

3. Ternary Relationship Set-

Ternary relationship set is a relationship set where three entity sets participate in a relationship set.

Example-



Ternary Relationship Set

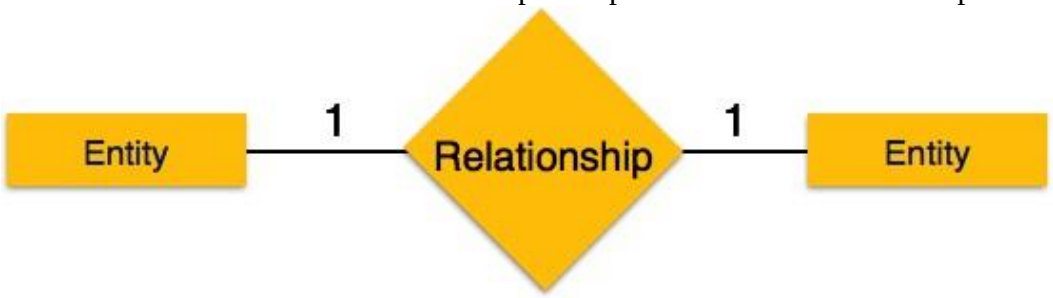
4. N-ary Relationship Set-

N-ary relationship set is a relationship set where 'n' entity sets participate in a relationship set.

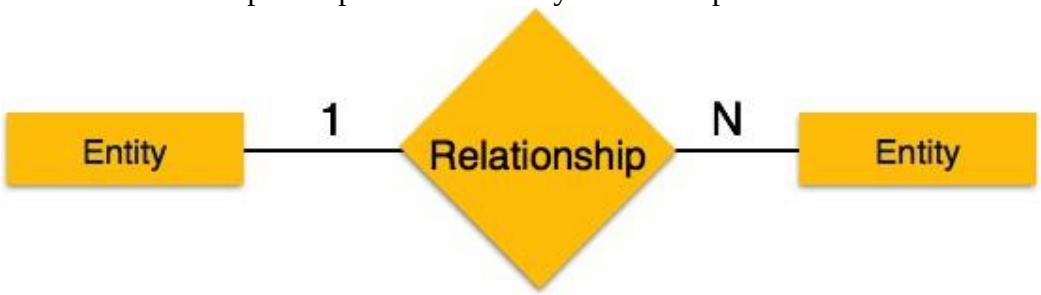
Binary Relationship and Cardinality

A relationship where two entities are participating is called a **binary relationship**. Cardinality is the number of instance of an entity from a relation that can be associated with the relation.

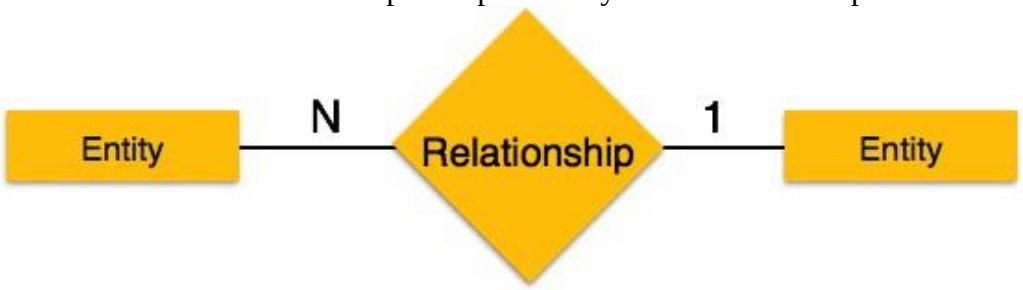
- **One-to-one** − When only one instance of an entity is associated with the relationship, it is marked as '1:1'. The following image reflects that only one instance of each entity should be associated with the relationship. It depicts one-to-one relationship.



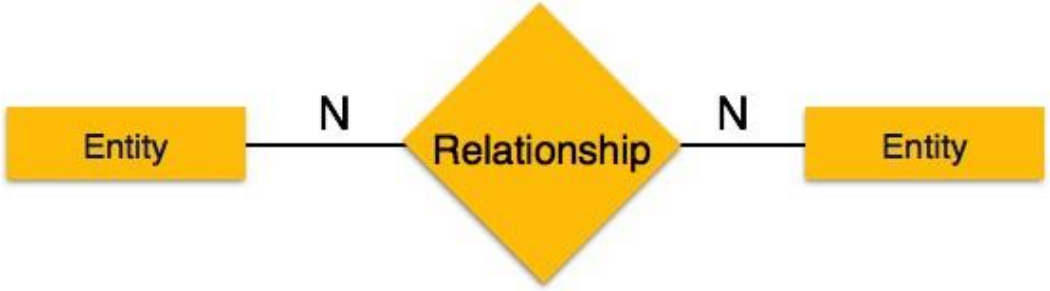
- **One-to-many** − When more than one instance of an entity is associated with a relationship, it is marked as '1:N'. The following image reflects that only one instance of entity on the left and more than one instance of an entity on the right can be associated with the relationship. It depicts one-to-many relationship.



- **Many-to-one** − When more than one instance of entity is associated with the relationship, it is marked as 'N:1'. The following image reflects that more than one instance of an entity on the left and only one instance of an entity on the right can be associated with the relationship. It depicts many-to-one relationship.

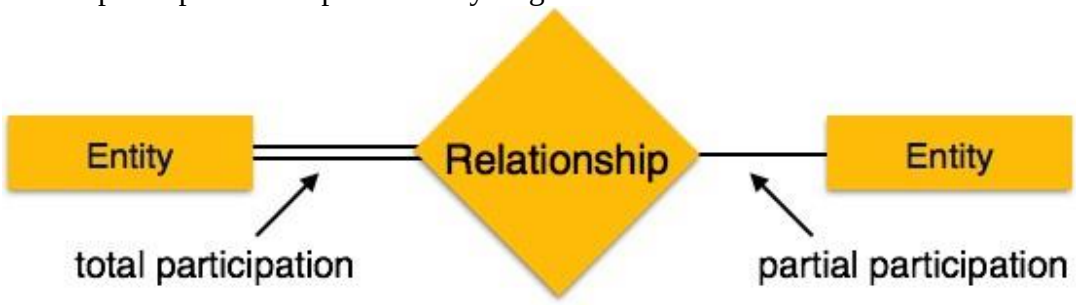


- **Many-to-many** − The following image reflects that more than one instance of an entity on the left and more than one instance of an entity on the right can be associated with the relationship. It depicts many-to-many relationship.



Participation Constraints

- **Total Participation** − Each entity is involved in the relationship. Total participation is represented by double lines.
- **Partial participation** − Not all entities are involved in the relationship. Partial participation is represented by single lines.



Single Valued attribute:
Attributes that can have single value at a particular instance of time are called single valued. A person can't have more than one age value. Therefore, age of a person is a single-values attribute.

Multi valued attributes:
A multi-valued attribute can have more than one value at one time. For example, a bank may limit the number of addresses recorded for a single customer to two Such attributes are represented by double ovals in ER diagram.

A single valued attribute can have only a single value. For example a person can have only one 'date of birth', 'age' etc. That is a single valued attributes can have only single value. But it can be simple or composite attribute.That is 'date of birth' is a composite attribute , 'age' is a simple attribute. But both are single valued attributes.

Multivalued attributes can have multiple values. For instance a person may have multiple phone numbers,multiple degrees etc.Multivalued attributes are shown by a double line connecting to the entity in the ER diagram.

Stored attribute and derived attribute:

The **main difference** between stored and derived attribute in DBMS is that **it is not possible to find the value of a stored attribute using other** attributes **while it is possible to find the value of a derived attribute using other attributes.**
Database Management System (DBMS) is a software that allows storing and managing data efficiently. It stores data in tables; these tables are also called entities. Each table has attributes. The attributes define the characteristics or properties of an entity. For example, a student table can have attributes such as id, name, age, location, etc. There is various type of attributes. Two of them are stored and derived attribute.

Stored and derived attributes

Stored attributes:

The stored attribute are such attributes which are already stored in the database and from which the value of another attribute is derived is called stored attribute. For example age of a person can be calculated from person's date of birth and present date. Difference between these two dates gives the value of age. In this case, date of birth is a stored attribute and age of the person is the derived attribute

Derived attributes:
The derived attributes are such attributes for which the value is derived or calculated from stored attributes. For example date of birth of an employee is the stored attribute but the age is the derived attributed. Derived attributes are usually created by a formula or by a summary operation on other attributes. Take another example, if we have to calculate the interest on some principal amount for a given time, and for a particular rate of interest, we can simply use the interest formula
Interest=(N*P*R)/100;
In this case, interest is the derived attribute whereas principal amount (P), time (N) and rate of interest(R) are all stored attributes.

Types of DBMS Entities and their examples

Weak entity and strong entity
Example of Entity in DBMS

Let us see an example:
<Professor>

Professor_ID	Professor_Name	Professor_City	Professor_Salary
P01	Tom	Sydney	$7000
P02	David	Brisbane	$4500
P03	Mark	Perth	$5000

Here, **Professor_Name, Professor _Address and Professor _Salary** are attributes.
Professor_ID is the primary key
Types of DBMS Entities

The following are the types of entities in DBMS:

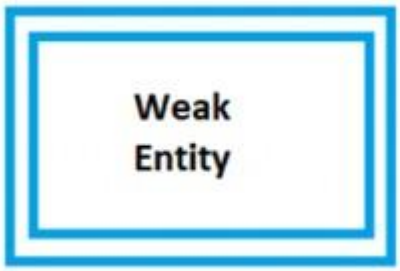
Strong Entity

The strong entity has a primary key. Weak entities are dependent on strong entity. Its existence is not dependent on any other entity.
Strong Entity is represented by a single rectangle:

Continuing our previous example, **Professor** is a strong entity here, and the primary key is **Professor_ID.**

Weak Entity

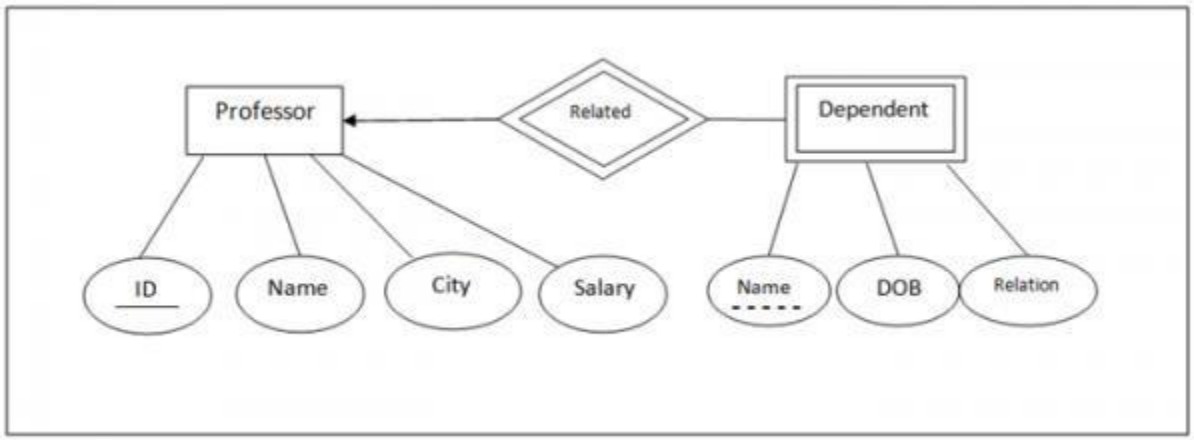
The weak entity in DBMS do not have a primary key and are dependent on the parent entity. It mainly depends on other entities.
Weak Entity is represented by double rectangle:



Continuing our previous example, **Professor** is a strong entity, and the primary key is **Professor_ID**. However, another entity is **Professor_Dependents**, which is our Weak Entity.
<Professor_Dependents>

Name	**DOB**	**Relation**

This is a weak entity since its existence is dependent on another entity **Professor**, which we saw above. A Professor has Dependents.
Example of Strong and Weak Entity
The example of strong and weak entity can be understood by the below figure.



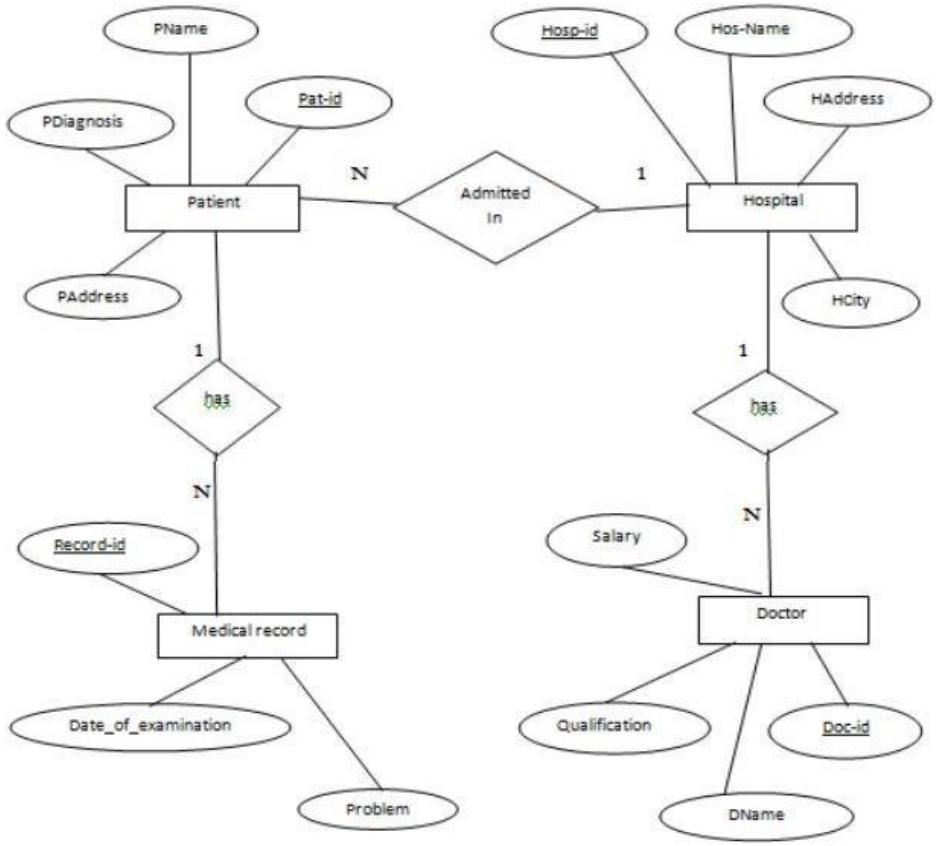
The Strong Entity is **Professor**, whereas **Dependent** is a Weak Entity.
ID is the primary key (represented with a line) and Name in **Dependent** entity is called **Partial Key** (represented with a dotted line).
The Strong Entity is **Professor**, whereas **Dependent** is a Weak Entity.
ID is the primary key (represented with a line) and Name in **Dependent** entity is called **Partial Key** (represented with a dotted line)

A member of a strong entity set is **called dominant entity and** member of weak entity set is called as **subordinate entity**.

Strong Entity Set	Weak Entity Set
it has its own primary key.	It does not save sufficient attributes to form a primary Key on its own.
It is represented by a rectangle.	It is represented by a double rectangle.
It contains a primary key represented by an underline.	It contains a Partial Key or discriminator represented by a dashed underline.
The member of strong entity set is called as dominant entity set.	The member of weak entity set is called as subordinate entity set.
The Primary Key is one of its attributes which uniquely Identifies its member.	The Primary Key of weak entity set is a combination of partial Key and Primary Key of the strong entity set.
The relationship between two strong entity set is represent by a diamond symbol.	The relationship between one strong and a weak entity set is represented by a double diamond sign. It is known as identifying relationship.
The line connecting strong entity set with the relationship is single	The line connecting weak entity set with the identifying relationship is double.
Total participation in the relationship may or may not exist.	Total participation in the identifying relationship always exists.

Draw E-R diagram for Hospital management System:

Step 1: E-R Diagram



Step 2: Converting the E-R Diagram into Tables:

Hospital	
Hosp-id	**Primary Key**
HCity	
HAddress	
Hos-Name	
Pat-id	Foreign key references to Pat-id of Patient table
Doc-id	Foreign key references to Doc-id of Doctor table

Patient	
Pat-id	**Primary Key**
PName	
PAddress	
PDiagnosis	
Record-id	Foreign key references to Record-id of Medical Record table
Hosp-id	Foreign key references to Hosp-id of Hospital table

Medical Record	
Record-id	**Primary Key**
Problem	
Date_of_examination	
Pat-id	Foreign key references to Pat-id of Patient table

Doctor	
Doc-id	**Primary Key**
DName	
Qualification	
Salary	
Hosp-id	Foreign key references to Hosp-id of Hospital table

Step 3: Mapping of Attributes

- **Simple Attributes**
  Simple Attributes which can not be divided into subparts.
  Example: Salary of Doctor

- **Composite Attributes**
  Composite Attributes which can be divided into subparts.
  Example: Patient Name, Doctor Name

Patient

First_Name
Middle_Name
Last_name

Doctor

First_Name
Middle_Name
Last_name

Step 4: Mapping of Relationships
b. Foreign Key approach
Hosp_patient

Pat-id	**Hospital table makes foreign key references to Pat-id of Patient table**
Hosp-id	Patient table makes foreign key references to Hosp-id of Hospital table

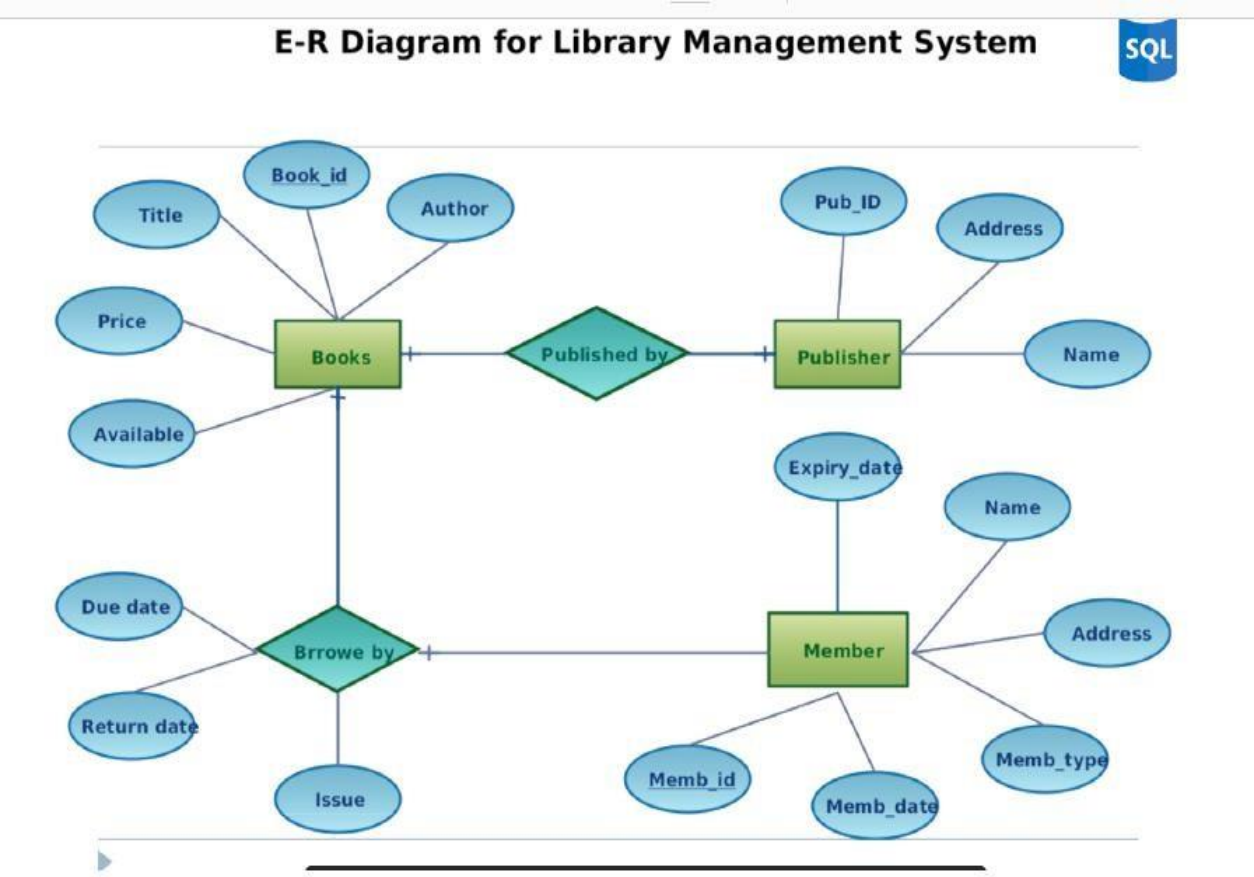
Hosp_Doctor

Hosp-id	**Doctor table makes foreign key references to Hosp-id of Hospital table**
Doc-id	Hospital table makes foreign key references to Doc-id of Doctor table

PatiPPatient_MedicalRecord

Pat-id	**Medical Record table makes foreign key references to Pat-id of Patient table**
Record-id	Patient table makes foreign key references to Record-id of Medical Record table

Step 5: Identifying the relationships
a. Hospital has a set of patients.
Therefore the relations is 1……..N.
b. Hospital has a set of doctors.
Therefore the relations is 1……..N.
c. Doctor are associated with each patient.
Therefore the relations is N……..1.
d. Each patient has record of various test and examination conducted.
Therefore the relations is 1……..N.



AGGREGATE FUNCTIONS

Aggregate Functions are all about

- Performing calculations on multiple rows of a single column of a table and returning a single value.
- The commonly used aggregated functions are

The following are the most commonly used SQL aggregate functions:

- AVG – calculates the average of a set of values.
- COUNT – counts rows in a specified table or view.
- MIN – gets the minimum value in a set of values.
- MAX – gets the maximum value in a set of values.
- SUM – calculates the sum of values.

SQL | ORDER BY

The ORDER BY statement in sql is used to sort the fetched data in either ascending or descending according to one or more columns.

- By default ORDER BY sorts the data in **ascending order.**
- We can use the keyword DESC to sort the data in descending order and the keyword ASC to sort in ascending order.

Syntax of all ways of using ORDER BY is shown below:

- **Sort according to one column:** To sort in ascending or descending order we can use the keywords ASC or DESC respectively.
  Syntax:
  SELECT * FROM table_name ORDER BY column_name ASC|DESC

Sort according to multiple columns: To sort in ascending or descending order we can use the keywords ASC or DESC respectively. To sort according to multiple columns, separate the names of columns by (,) operator.
Syntax:
SELECT * FROM table_name ORDER BY column1 ASC|DESC , column2 ASC|DESC

SELECT * FROM STUDENT ORDER BY SNO;
SELECT * FROM STUDNET ORDER BY 1 DESC;

SQL | GROUP BY

The GROUP BY Statement in SQL is used to arrange identical data into groups with the help of some functions. i.e if a particular column has same values in different rows then it will arrange these rows in a group.

Important Points:

- GROUP BY clause is used with the SELECT statement.
- In the query, GROUP BY clause is placed after the WHERE clause.
- In the query, GROUP BY clause is placed before ORDER BY clause if used any.

Syntax:

```
SELECT column1, function_name(column2)
FROM table_name
WHERE condition
GROUP BY column1, column2
ORDER BY column1, column2;
```

function_name: Name of the function used for example, SUM() , AVG().
table_name: Name of the table.
condition: Condition used.

Sample Table:

Employee

SI NO	NAME	SALARY	AGE
1	Harsh	2000	19
2	Dhanraj	3000	20
3	Ashish	1500	19
4	Harsh	3500	19
5	Ashish	1500	19

Student

SUBJECT	YEAR	NAME
English	1	Harsh
English	1	Pratik
English	1	Ramesh
English	2	Ashish
English	2	Suresh
Mathematics	1	Deepak
Mathematics	1	Sayan

Example:

- **Group By single column**: Group By single column means, to place all the rows with same value of only that particular column in one group. Consider the query as shown below:

```
SELECT NAME, SUM(SALARY) FROM Employee
GROUP BY NAME;
```

The above query will produce the below output:

NAME	SALARY
Ashish	3000
Dhanraj	3000
Harsh	5500

As you can see in the above output, the rows with duplicate NAMEs are grouped under

same NAME and their corresponding SALARY is the sum of the SALARY of duplicate rows. The SUM() function of SQL is used here to calculate the sum.

- **Group By multiple columns**: Group by multiple column is say for example, **GROUP BY column1, column2**. This means to place all the rows with same values of both the columns **column1** and **column2** in one group. Consider the below query:

```
SELECT SUBJECT, YEAR, Count(*)
FROM Student
GROUP BY SUBJECT, YEAR;
```

Output:

SUBJECT	YEAR	Count
English	1	3
English	2	2
Mathematics	1	2

As you can see in the above output the students with both same SUBJECT and YEAR are placed in same group. And those whose only SUBJECT is same but not YEAR belongs to different groups. So here we have grouped the table according to two columns or more than one column.

HAVING Clause

We know that WHERE clause is used to place conditions on columns but what if we want to place conditions on groups?

This is where HAVING clause comes into use. We can use HAVING clause to place conditions to decide which group will be the part of final result-set. Also we can not use the aggregate functions like SUM(), COUNT() etc. with WHERE clause. So we have to use HAVING clause if we want to use any of these functions in the conditions.

Syntax:

```
SELECT column1, function_name(column2)
FROM table_name
WHERE condition
GROUP BY column1, column2
HAVING condition
ORDER BY column1, column2;
```

function_name: Name of the function used for example, SUM() , AVG().
table_name: Name of the table.
condition: Condition used.

Example:

```
SELECT NAME, SUM(SALARY) FROM Employee
GROUP BY NAME
HAVING SUM(SALARY)>3000;
```

Nested Queries

Nested query is a query within another SQL query and embedded within the WHERE clause. Embedded query is called Subquery.

A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.

Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.

There are a few rules that subqueries must follow −

- Subqueries must be enclosed within parentheses.
- A subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the subquery to compare its selected columns.
- An ORDER BY command cannot be used in a subquery, although the main query can use an ORDER BY. The GROUP BY command can be used to perform the same function as the ORDER BY in a subquery.
- Subqueries that return more than one row can only be used with multiple value operators such as the IN operator.
- The SELECT list cannot include any references to values that evaluate to a BLOB, ARRAY, CLOB, or NCLOB.
- A subquery cannot be immediately enclosed in a set function.
- The BETWEEN operator cannot be used with a subquery. However, the BETWEEN operator can be used within the subquery.

Q1) Find names, salaries whose salary greater than the salary of employ ‘SMITH’;

SQL>select Ename,sal from emp where sal > (select sal from emp where ename like 'smith');

Q2) Display the Ename, Job, Sal from EMP working in the LOC 'CHICAGO' and salary is less than the salary of whose empno=7876.

```
SQL> SELECT ENAME, JOB, SAL FROM EMP WHERE DEPINO = (SELECT DEPTNO FROM DEPT WHERE LOC='CHICAGO') AND SAL < (SELECT SAL FROM EMP WHERE EMPNO=7876);
```

Q3) To retrieve the details of the employee holding the minimum salary.

```
SQL> SELECT ENAME,JOB,SAL FROM EMP WHERE SAL=(SELECT MIN(SAL) FROM E
MP;
```

Q4) finds the employees whose salary is the same as the minimum salary of the employees in some department.

```
SQL> SELECT ENAME,SAL FROM EMP WHERE SAL IN (SELECT MIN (SAL) FROM EMP
GROUP BY DEPTNO);
```

Q5) List all the employees Name and Sal working at the location 'DALLAS'.

```
SQL> SELECT ENAME, SAL, JOB FROM EMP WHERE DEPTNO IN (SELECT DEPTNO
FROM DEPT WHERE LOC ='DAllAS');
```

Correlated Subqueries

Correlated subqueries are used for row-by-row processing. Each subquery is executed once for every row of the outer query.

A correlated subquery is evaluated once for each row processed by the parent statement. The parent statement can be a **SELECT**, **UPDATE**, or **DELETE** statement.

A correlated subquery is one way of reading every row in a table and comparing values in each row against related data. It is used whenever a subquery must return a different result or set of results for each candidate row considered by the main query. In other words, you can use a correlated subquery to answer a multipart question whose answer depends on the value in each row processed by the parent statement.

Nested Sub queries Versus Correlated Sub queries :

With a normal nested subquery, the inner **SELECT** query runs first and executes once, returning values to be used by the main query. A correlated subquery, however, executes once for each candidate row considered by the outer query. In other words, the inner query is driven by the outer query.

NOTE : You can also use the **ANY** and **ALL** operator in a correlated subquery.

EXAMPLE of Correlated Subqueries : Find all the employees who earn more than the average salary in their department.

```
SQL> SELECT  Ename, sal, deptno  FROM emp out  WHERE sal >  (SELECT AVG(sal)
FROM emp inn  WHERE  inn.deptno =  out.deptno);
```

SUBQUERIES :

A Query within another query is known as sub query.

1. The result of one query is dynamically substitute in the condition of another query.
2.SQL first evaluates the sub query(or inner query) within the where clause.
3.The return value of sub query is then substituted in the condition of the outer query.
4.There is no limitation to the level of nesting queries.
5.When using relational operators, ensures that the sub query returns a single row output.

Syntax : SELECT Column list from table where column=(select column from table where---);

Note : The sub query always must be within the parenthesis.

Sub query Operators :

1.IN
2.ANY
3.ALL

1.IN : This operator defines set of values in which a value may be existed or not.

SYN : Column_Name[not] IN(value1,value2...)

SELECT * FROM EMP1 WHERE ESAL IN(SELECT MAX(ESAL) FROM EMP1);

2.ANY : >ANY >=ANY <ANY <=ANY

>=ANY: Greater than or equal to minimum salaries returned by the subquery.

<ANY: In this example it displays all rows from emp1 where the salary is less than max salary returned by a sub query salaries.

<=ANY : Less than or equal to max salary returned by a sub query salaries.

3.ALL : >ALL >=ALL <ALL <=ALL

SELECT * FROM EMP1;

ENO	ENAME	DESIG	ESAL
1	BHANU	MANAGER	15000
2	RAMU	CLERK	4500
3	HARI	MANAGER	14500
4	SIVA	CLERK	3400
5	BABU	ACCOUNTANT	5000
6	JK	ACCOUNTANT	4500
7	GG	MANAGER	17500
8	SRI	CLERK	2300
9	NANDA	ACCOUNTANT	4600
10	TTE	MANAGER	16500

SELECT * FROM EMP1 WHERE ESAL>**ANY**(SELECT ESAL FROM EMP1 WHERE DESIG='MANAGER');

ENO	ENAME	DESIG	ESAL
7	GG	MANAGER	17500
10	TTE	MANAGER	16500
1	BHANU	MANAGER	15000

SQL> SELECT * FROM EMP1 WHERE ESAL>=**ANY**(SELECT ESAL FROM EMP1 WHERE DESIG='MANAGER');

ENO	ENAME	DESIG	ESAL
7	GG	MANAGER	17500
10	TTE	MANAGER	16500
1	BHANU	MANAGER	15000
3	HARI	MANAGER	14500

SQL> SELECT * FROM EMP1 WHERE ESAL<**ANY**(SELECT ESAL FROM EMP1 WHERE DESIG='MANAGER');

ENO	ENAME	DESIG	ESAL
8	SRI	CLERK	2300
4	SIVA	CLERK	3400
2	RAMU	CLERK	4500
6	JK	ACCOUNTANT	4500
9	NANDA	ACCOUNTANT	4600
5	BABU	ACCOUNTANT	5000

```
         3 HARI            MANAGER              14500
         1 BHANU           MANAGER              15000
         10 TTE            MANAGER              16500
```

9 rows selected.

SQL> SELECT * FROM EMP1 WHERE ESAL>**ALL**(SELECT ESAL FROM EMP1 WHERE DESIG='MANAGER');

no rows selected

SQL> SELECT * FROM EMP1 WHERE ESAL>=**ALL**(SELECT ESAL FROM EMP1 WHERE DESIG='CLERK');

```
      ENO ENAME           DESIG                ESAL
---------- ---------- ---------- ----------
         1 BHANU           MANAGER              15000
         2 RAMU            CLERK                 4500
         3 HARI            MANAGER              14500
         5 BABU            ACCOUNTANT            5000
         6 JK              ACCOUNTANT            4500
         7 GG              MANAGER               17500
         9 NANDA           ACCOUNTANT            4600
        10 TTE             MANAGER               16500
```

8 rows selected.

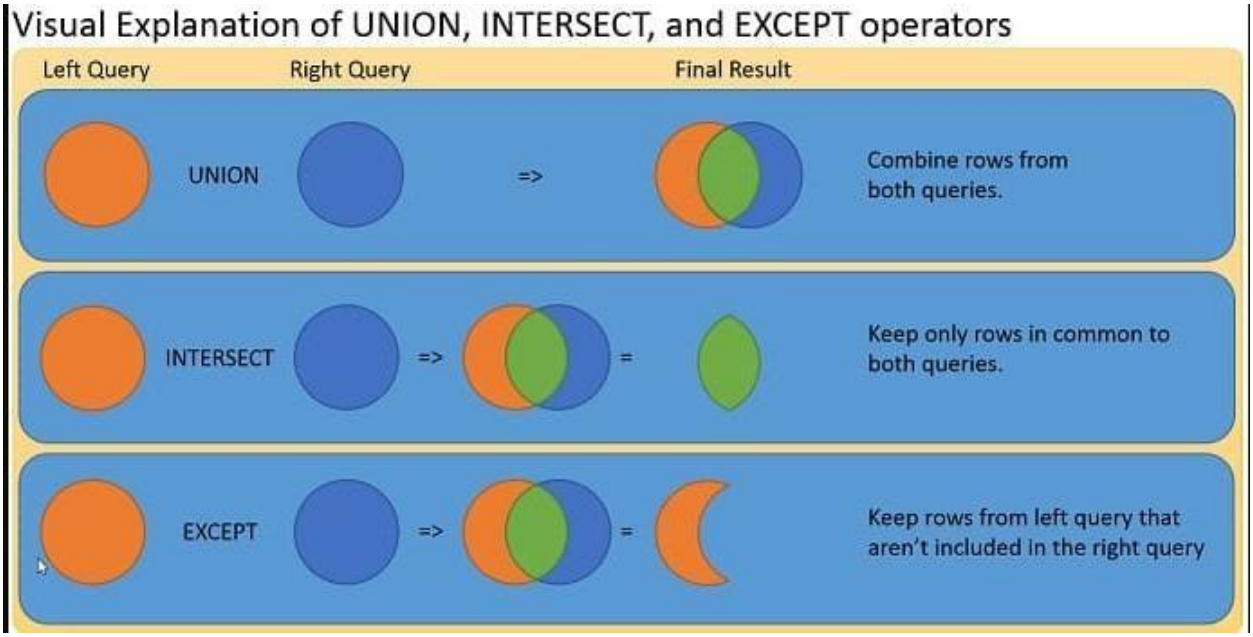
SQL> SELECT * FROM EMP1 WHERE ESAL>**ALL**(SELECT ESAL FROM EMP1 WHERE DESIG='CLERK');

```
      ENO ENAME           DESIG                ESAL
---------- ---------- ---------- ----------
         1 BHANU           MANAGER              15000
         3 HARI            MANAGER              14500
         5 BABU            ACCOUNTANT            5000
         7 GG              MANAGER              17500
         9 NANDA           ACCOUNTANT            4600
        10 TTE             MANAGER              16500
```

6 rows selected.

Union, Intersect, and Except Clauses(SET operators)

The UNION, INTERSECT, and EXCEPT clauses are used to combine or exclude like rows from two or more tables. They are useful when you need to combine the results from separate queries into one single result. They differ from a join in that entire rows are matched and, as a result, included or excluded from the combined result.



EMP_JOB

ENO	NAME	DESIG	DEPNO	SAL
1	VASU	MANAGER	10	5000
2	RAVI	CLERK	20	3000
3	KSS	CLERK	10	2500
4	VAMSI	SVISOR	30	4000
5	BOSU	CLERK	10	3000
6	RAJU	CLERK	30	2000

UNION Operator

The Union operator returns rows from both tables. If used by itself, UNION returns a distinct list of rows. Using UNION ALL, returns all rows from both tables. A UNION is useful when you want to sort results from two separate queries as one combined result. For instance if you have two tables, Vendor, and Customer, and you want a combined list of names, you can easily do so using:

```
SELECT DESIG FROM EMP_JOB
WHERE DEPTNO=20
UNION
SELECT DESIG FROM EMP_JOB
WHERE DEPNO=30;
```

20 - CLERK
30 - CLERK,SVISOR
OUTPUT - CLERK,SVISOR

UNION ALL:

```
SELECT DESIG FROM EMP_JOB
WHERE DEPTNO=20
UNION ALL
SELECT DESIG FROM EMP_JOB
WHERE DEPTNO=30
UNION ALL
SELECT DESIG FROM EMP_JOB
WHERE DEPTNO=10;
```

OUTPUT - ALL ROWS FROM DESIG

```
SELECT DESIG,SAL FROM EMP_JOB
WHERE DEPTNO=10
UNION
SELECT DESIG,SAL FROM EMP_JOB
WHERE DEPTNO=30;
```

10TH DEPT

MANAGER 5000
CLERK 2500
CLERK 3000

30TH DEPT

SVISOR 4000
CLERK 2000
CLERK 3000

OUTPUT : CLERK 3000

INTERSECT Operator

Use an intersect operator to returns rows that are in common between two tables; it returns unique rows from both the left and right query. This query is useful when you want to find results that are in common between two queries. Continuing with Vendors, and Customers, suppose you want to find vendors that are also customers. You can do so easily using:

```
SELECT DESIG FROM EMP_JOB
WHERE DEPTNO=20
INTERSECT
SELECT DESIG FROM EMP_JOB
WHERE DEPNO=30;
```

20 - CLERK
30 - CLERK,SVISOR
OUTPUT - CLERK

EXCEPT/MINUS Operator

Use the EXCEPT Operator to return only rows found in the left query. It returns unique rows from the left query that aren't in the right query's results. This is similar to MINUS command in other sql softwares. This query is useful when you're looking to find rows that are in one set but not another. For example, to create a list of all vendors that are not customers you could write:

```
SELECT DESIG FROM EMP_JOB
WHERE DEPTNO=10
MINUS
SELECT DESIG FROM EMP_JOB
WHERE DEPNO=30;
```

NULL values

The SQL **NULL** is the term used to represent a missing value. A NULL value in a table is a value in a field that appears to be blank.

A field with a NULL value is a field with no value. It is very important to understand that a NULL value is different than a zero value or a field that contains spaces.

How to Test for NULL Values?

It is not possible to test for NULL values with comparison operators, such as =, <, or <>.

We will have to use the IS NULL and IS NOT NULL operators instead.

IS NULL Syntax

```
SELECT column_names
FROM table_name
WHERE column_name IS NULL;
```

IS NOT NULL Syntax

```
SELECT column_names
FROM table_name
WHERE column_name IS NOT NULL;
```

The IS NULL Operator

The IS NULL operator is used to test for empty values (NULL values).

The following SQL lists all customers with a NULL value in the "Address" field:

Example

```
SELECT CustomerName, ContactName, Address
FROM Customers
WHERE Address IS NULL;
```

Always use IS NULL to look for NULL values.

The IS NOT NULL Operator

The IS NOT NULL operator is used to test for non-empty values (NOT NULL values). The following SQL lists all customers with a value in the "Address" field:

Example

```
SELECT CustomerName, ContactName, Address
FROM Customers
WHERE Address IS NOT NULL;
```

INTEGRITY CONSTRAINTS

The Integrity constraints are useful to prevent the invalid data entry into a table. We can add these constraints to the particular column at the time of creating table also by using ALTER TABLE command Constraints are mainly three types:

1. **Domain constraints**
   a. NOT NULL constraints
   b. CHECK constraints
2. **Entity constraints**
   a. Unique constraints
3. **Referential constraints**
   a. Primary key constraints
   b. Foreign key constrains

DOMAIN CONSTRAINTS

A. NOTNULL:

This constraint is useful to check the column values for the NULL values. It displays an error message when we enter no value to a specific column.

Create table (eno number(3) NOT NULL,Ename varchar(20));

B. CHECK:

This constraint is useful to enter the specified value into a specific column. We can add a condition along with these constraints. This constraint allows values when the condition is true. It displays an error message when the condition is false.

Create table(eno number(3) check(eno>50),Eename varchar(20));

ENTITY CONSTRAINTS

UNIQUE:

These constraints are useful to prevent duplicate values in specific columns. It displays an error message when we enter any one of the previous values in a column.

How to add constraints at the time of creating table:

```
create table emp1
(ENO number (3) constraint  SA Unique,
Ename  varchar 2 (20) constraint RA not null,
Esal number (6,2) constraint LA check (esal <= 10000),
deptno number  (4));
```

To see the user constraints list:

```
select * from user_ constraints;
```

We can also add constraints
to the exiting table by using 'ALTER' table command,
Ex:

Alter table emp 1 add constraints a unique (eno);
Alter table emp 1 modify ename constraints b not null;
Alter table emp 1 add constraint c check (esal <=10000);

we can also drop the constraints from a table by using 'ALTER' table command.
EX:

Alter table emp drop constraint c;
Alter table emp disable constraint A;;

The above example disable the constraints A. If you enable after sometime we can use same command with enable.
Ex:
Alter table Emp1 enable constraint A;

PRIMARY KEY CONSTRAINT

This constraints avoids duplication of rows and does not allow null values, when enforced in a column. As a result it is used to identify a row. A table can have only one primary key.

Example:

Create table dept (deptno number (4) constraint p primary key,
deptname varchar 2 (10));

REFERENTIAL INTIGRETY CONSTRAINTS

To establish parent child relationship between two tables having a common column, we make use of referential constraints. To implement this we should define the column in the parent table as a primary key and the same column in the child as a foreign key referring to the corresponding parent entry.

FOREIGN KEY:

A column included in the definition of referential integrity, which would refer to a referenced key [primary key].
Child table:

This table depends up on the value present in the referenced of the parent table.

Parent table:
It determines whether insertion or updating of data can be done in the child table. It would be preferred by child foreign key.

Example:

```
Create table emp (eno
number (4), ename
varchar 2 (20), esal
number (10,2),
deptno number (4) constraint f references dept (deptno ));
```

The referential integrity constraints does not use *foreignkey* keyword to identify the column that makeup the foreign key.

Simple Creation of Constraints:

```
SQL> create table emp
      (
      empno number(5) primary key,
      ename varchar2(10) not null,
      job varchar2(10),
      mgr number(4),
      hiredate date,
      sal number(7,2) check(sal>=500 and sal<=10000),
      comm number(7,2),
      deptno number(2), foreign key(deptno) references dept
      );
```

SQL Trigger

Trigger: A trigger is a stored procedure in database which automatically invokes whenever a special event in the database occurs. For example, a trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated.
The database that has a set of associated triggers is called an **Active database.**
A trigger contains 3 parts:

- Event: A change to the data base that activates the trigger.
- Condition: A query or test that is run when the trigger is activated.
- Action: A procedure that is executed when the trigger is activated and its condition is true.
- Sybtax:

```
Create or replace trigger triggername
Before/after insert/update/delet on table name
Begin
Body
End;
/
```

Example:The following trigger automatically displays a message before every insert operation on emp table.

```
create trigger T
before insert on emp
begin
  dbms_output.put_line('One new row inserted');
end;
```

Advantages of Triggers :

SQL triggers provide an alternative way to check the integrity of data.

- SQL triggers can catch errors in business logic in the database layer.
- SQL triggers provide an alternative way to run scheduled tasks. By using SQL triggers, you don't have to wait to run the scheduled tasks because the triggers are invoked automatically *before* or *after* a change is made to the data in the tables.
- SQL triggers are very useful to audit the changes of data in tables.

Disadvantages of using SQL triggers

SQL triggers only can provide an extended validation and they cannot replace all the validations. Some simple validations have to be done in the application layer.

JOINS :

Joins are used to combine the columns from different tables in to a single table.

The join operation is the mechanism that allows tables to be related to one another The join operation retrieves columns from two or more tables.

Main points of Joins :

1. Joins are used combine columns from different tables.
2. In a join the tables are listed in the FROM clause, separated by commas.
3. The condition of the query can refer to any column of any table join.
4. The condition between tables is established through the WHERE clause.

Types of Joins :

1. Inner Joins
2. Cartesian joins
3. Outer joins
4. Self joins

1.Inner Joins or Equi or Non Equi Joins :

a. Equi Joins : When tow tables are joined together using equally of values in one or more columns the make an equi join. The Equi join operator '='

```
SELECT RNO,NAME,COURSE_FEE.COURSE,TFEES,FPAID,TFEES-FPAID
FROM STUD_INFO,COURSE_FEE
WHERE STUD_INFO.COURSE=COURSE_FEE.COURSE
```

RNO	NAME	COURSE	TFEES	FPAID	TFEES-FPAID
101	VASU	DOA	6000	2000	4000
102	BHANU	ADDA	6500	1500	5000
103	SRINU	DOA	6000	1000	5000
104	SIVA	HDCA	22000	6000	16000
105	PRAKASH	DOA	6000	3000	3000
106	RAMU	PGDCA	14000	2500	11500

Note : Ambiguous columns

You need to keep in mind that each reference to a column in a join must e unambiguous. In this context unambiguous means that if the column exists in more that one table.

b. Non_Equi join :

The relationship is obtained using an operator other than equal(=) sign .

2.Outer Join :

If there are many values in one table that do not have corresponding values in the other table, in an equi join that tow will not be selected. Such rows can be forcefully selected by using the outer join symbol(+). The corresponding columns for that row will have NULLS.

```
SELECT RNO,NAME,COURSE_FEE.COURSE,TFEES,FPAID,
TFEES-FPAID FROM STUD_INFO,COURSE_FEE
WHERE STUD_INFO.COURSE(+)=COURSE_FEE.COURSE;
```

RNO	NAME	COURSE	TFEES	FPAID	TFEES-FPAID
		ADCA	16000		
102	BHANU	ADDA	6500	1500	5000
		C	1500		
		C++	2000		
101	VASU	DOA	6000	2000	4000
103	SRINU	DOA	6000	1000	5000
105	PRAKASH	DOA	6000	3000	3000
104	SIVA	HDCA	22000	6000	16000
		MS_OFFICE	1500		
106	RAMU	PGDCA	14000	2500	11500

3. Self Join

To join a table to itself means that each row of the table tis combined with itself and with every other row of the table. The Self join can be viewed as a join of two copies of the same table. The table is not actually copied.

SQL>SELECT * FROM BHANU;

ENO	ENAME	SAL	ADR
1	BHANU	6700	R.NAGAR
2	SRINU	23000	K.NAGAR
3	RAMU	4500	J.NAGAR
4	HANU	8900	R.NAGAR
5	SIVA	6789	K.NAGAR
8	HANUMAN	4567	E.NAGAR

SQL>SELECT A1.ENO,A2.ENAME,A2.SAL FROM BHANU A1,BHANU A2
WHERE A1.ENO=A2.ENO

ENO	ENAME	SAL
1	BHANU	6700
1	BHANU	6700
1	BHANU	6700
1	BHANU	6700
2	SRINU	23000
3	RAMU	4500
4	HANU	8900
5	SIVA	6789
8	HANUMAN	4567

4.Cartesian Join

You are first learning to join multiple tales. A common error is to forget to provide a join condition in the where clause. If you forgot a join condition you will notice two things

a. The query takes longer to execute
b. The query retrieved records are much longer than you expected

When nowhere clause is specified, each row of one table matches every row of the other table. This result is Cartesian Product.

```
SQL>SELECT RNO,NAME,COURSE_FEE.COURSE,TFEES,FPAID,
TFEES-FPAID FROM STUD_INFO,COURSE_FEE;
```

RNO	NAME	COURSE	TFEES	FPAID	TFEES-FPAID
101	VASU	DOA	6000	2000	4000
102	BHANU	DOA	6000	1500	4500
103	SRINU	DOA	6000	1000	5000
104	SIVA	DOA	6000	6000	0
105	PRAKASH	DOA	6000	3000	3000
106	RAMU	DOA	6000	2500	3500
101	VASU	ADCA	16000	2000	14000
102	BHANU	ADCA	16000	1500	14500
103	SRINU	ADCA	16000	1000	15000
104	SIVA	ADCA	16000	6000	10000
105	PRAKASH	ADCA	16000	3000	13000
106	RAMU	ADCA	16000	2500	13500
101	VASU	ADDA	6500	2000	4500
102	BHANU	ADDA	6500	1500	5000
103	SRINU	ADDA	6500	1000	5500

MULTIPLE TABLES:

```
SELECT RNO,NAME,STUD_INFO.COURSE,TFEES,FPAID,TFEES-FPAID,DIS,TFEES-
(TFEES*DIS/100) ACTUALFEES
FROM STUD_INFO,COURSE_FEE,FEE_DIS
WHERE STUD_INFO.COURSE=COURSE_FEE.COURSE AND
STUD_INFO.COURSE=FEE_DIS.COURSE
```

RNO	NAME	COURSE	TFEES	FPAID	TFEES-FPAID	DIS	ACTUALFEES
101	VASU	DOA	6000	2000	4000	20	4800
103	SRINU	DOA	6000	1000	5000	20	4800
105	HARI	DOA	6000	3000	3000	20	4800
104	SIVA	HDCA	22000	6000	16000	50	11000

.

Sub quries:

Single row sub query

SQL> select ename,job,sal from emp where sal=(select min(sal) from emp);

```
ENAME      JOB         SAL
---------- ---------- -----
smith      clerk        800
```

SQL> select ename,job,sal from emp where sal=(select max(sal) from emp);

```
ENAME      JOB         SAL
---------- ---------- -----
king       president   5000
```

Multiple row sub query

SQL> select ename,job,sal from emp where sal in(800,1000,1500,3000);

```
ENAME      JOB         SAL
---------- ---------- -----
smith      clerk        800
scott      analyst     3000
turner     salesman    1000
ford       analyst     3000
```

SQL> select ename,job,sal from emp where sal >any(select sal from emp where dept no=30);

```
ENAME      JOB         SAL
---------- ---------- -------------
king       president   5000
ford       analyst     3000
scott      analyst     3000
jones      manager     2975
```

```
blake      manager     2850
clark      manager     2450
allen      salesman    1600
miller     clerk       1300
ward       salesman    1250
martin     salesman    1250
adems      clerk       1100
turner     salesman    1000

12 rows selected.

SQL> select ename,job,sal from emp where sal <any(select sal from emp where dept
no=30);

ENAME      JOB         SAL
---------- ---------- -----
smith      clerk        800
james      clerk        950
turner     salesman    1000
adems      clerk       1100
ward       salesman    1250
martin     salesma ı    1250
miller     clerk       1300
allen      salesman    1600
clark      manager     2450

9 rows selected.

SQL> select ename,job,sal from emp where sal >all(select sal from emp where dept
no=30);

ENAME      JOB         SAL
---------- ---------- -----
jones      manager     2975
scott      analyst     3000
king       president   5000
ford       analyst     3000

SQL> select ename,job,sal from emp where sal <all(select sal from emp where dept
no=30);

ENAME      JOB         SAL
---------- ---------- -----
smith      clerk        800

SQL> select ename,job,sal from emp where sal =any(select sal from emp where dept
```

no=30);

```
ENAME      JOB         SAL
---------- ---------- -----
allen      salesman    1600
martin     salesman    1250
ward       salesman    1250
blake      manager     2850
turner     salesman    1000
james      clerk        950

6 rows selected.
```

Correlated sub query

```
SQL> select ename,job,sal from emp e where sal >(select min(losal) from salgrade
 s where e.sal>s.losal);

ENAME      JOB         SAL
---------- ---------- -----
clark      manager     2450
jones      manager     2975
smith      clerk        800
blake      manager     2850
ford       analyst     3000
james      clerk        950
ward       salesman    1250
allen      salesman    1600
miller     clerk       1300
king       president   5000
adems      clerk       1100
martin     salesman    1250
turner     salesman    1000
scott      analyst     3000

14 rows selected.
```

VIEWS

After a table is created and populated with data, it may become necessary to prevent all user from accessing all columns of a table, for data security reasons. The query is stored in view permanently.

View is a virtual table or logical window. An interesting fact about a view that it stored only as definition in ORACLE SYSTEM CATALOGUE. When a reference is made to a view, its definition is scanned, the base table is opened and the view created on top of the base table [view does not occupy memory space].

Some views are used only for looking at table data. Other views can be used to INSERT, UPDATE and DELETE table data as well as view data.

Features of Views :

1.View is a logical window.
2.View does not occupied any memory space.

We can use a view to perform the following tasks

A. Maintain Security
B. Rename Columns
C. Hide Complexity

TYPES OF VIEWS

1. Updatable View
2. Readable View

1.**Updatable View:** A view that is used to look at table data as well as INSERT, DELETE AND UPDATE table data.

Syntax:

CREATE[OR REPLACE][FORCE|NOFORCE] VIEW<FILE_NAME>

[COLOUMN ALIAS,------] AS QUERY;

[WITH CHECK OPTION];

Example:

CREATE OR REPLACE VIEW SAL_VIEW(ITEM_NAME,ITEM_CODE)

AS SELECT ITNAME,ITCODE FROM SALES;

Manipulation of view :

INSERT INTO SAL_VIEW VAUES('LIRIL',789);

SELECT * FROM SAL_VIEW;

UPDATE SAL_VIEW SET ITNAME='RIN'

WHERE IT_NAME='LIRIL';

WHERE CONDITION:

CREATE OR REPLACE VIEW SAL_VIEW AS SELECT * FROM SALES WHERE ITCODE>7004;

INSERT INTO SAL_VIEW

VALUES('106,'MARGO',7000,20);

SELECT * FROM SALES;

SELECT * FROM SAL_VIEW;

FORCE: Without table also we can create view by using Force.

DROP TABLE EMP;

CREATE OR REPLACE VIEW SAL_VIEW AS SELECT * FROM EMP;

ERROR : TABLE OR VIEW DOES NOT EXIST.

CREATE OR REPLACE FORCE VIEW SAL_VIEW AS SELECT * FROM EMP;

ORACLE WARNNIG :
VIEW CREATED WITH COMPLIATION ERRORS.

CREATE TABLE EMP(ENO NUMBER(3));

2.**Read Only view or Not Updatable View** : In this we can create the view by multiple columns from different tables or using Aggregated functions. If you create like this it is not possible to INSERT, DELETE AND UPDATE the View.

Multiple Columns from Multiple Tables

CREATE OR REPLACE VIEW MT
AS SELECT MASTER.ENO,MASTER.SAL,
STUDENT.NAME FROM MASTER,STUDENT;

SELECT * FROM MT;

NOTE : IN THIS WE CANNOT INSERT ANY RECORDS.

Group By Clause

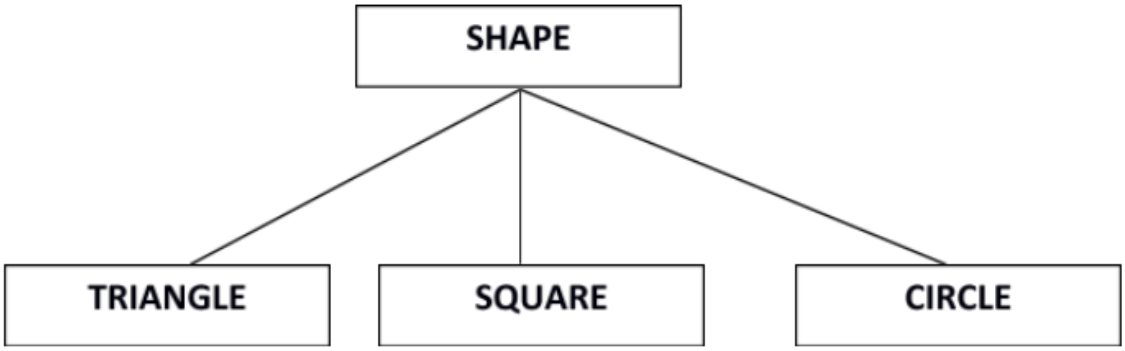
CREATE OR REPLACE VIEW SAMPLE
AS SELECT DEPTNO,SUM(SAL) FROM EMP
GROUP BY DEPTNO;

NOTE : IN THIS WE CANNOT INSERT ANY RECORDS.

Subclasses and Super class:

Super class is an entity that can be divided into further subtype.

For **example** − consider Shape super class.



Super class shape has sub groups: **Triangle, Square and Circle**.

Sub classes are the group of entities with some unique attributes. Sub class inherits the properties and attributes from super class.

Generalization is a process in which the common attributes of more than one entities form a new entity. This newly formed entity is called generalized entity.

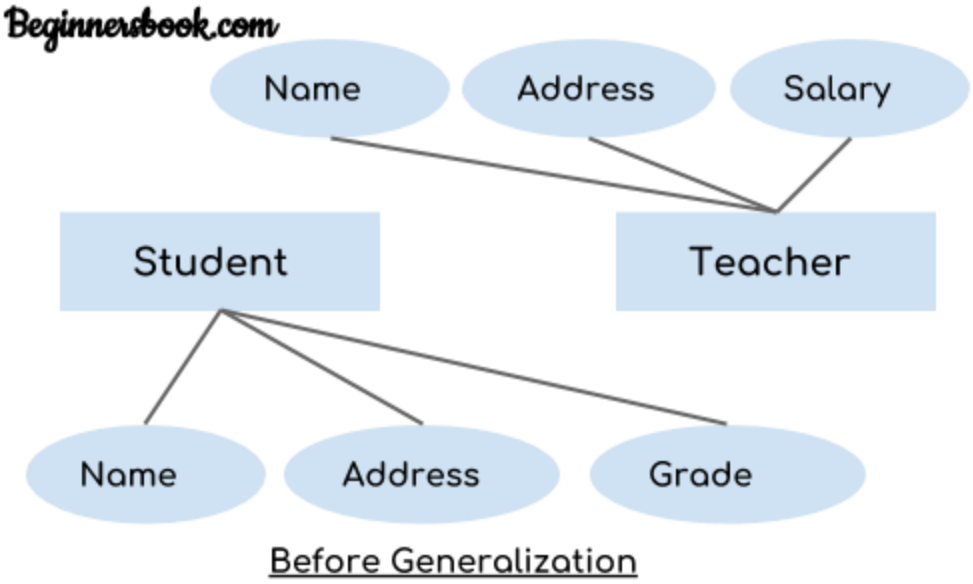
Generalization Example

Lets say we have two entities **Student and Teacher.**

Attributes of Entity Student are: Name, Address & Grade

Attributes of Entity Teacher are: Name, Address & Salary

The ER diagram before generalization looks like this:

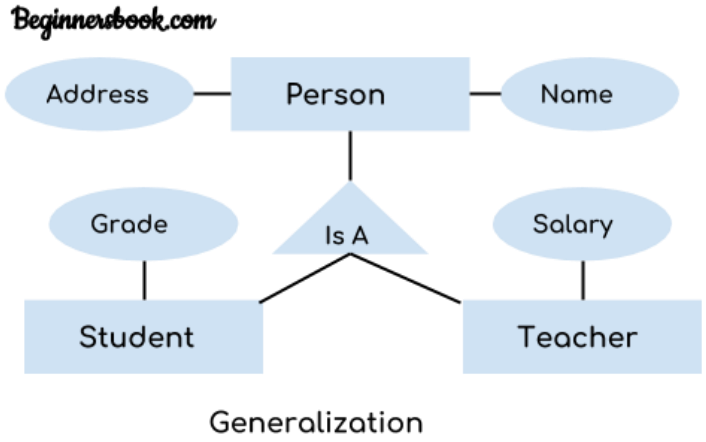


Before Generalization

These two entities have two common attributes: Name and Address, we can make a generalized entity with these common attributes. Lets have a look at the ER model after generalization.

The ER diagram after generalization:

We have created a new generalized entity Person and this entity has the common attributes of both the entities. As you can see in the following ER diagram that after the generalization process the entities Student and Teacher only has the specialized attributes Grade and Salary respectively and their common attributes (Name & Address) are now associated with a new entity Person which is in the relationship with both the entities (Student & Teacher).



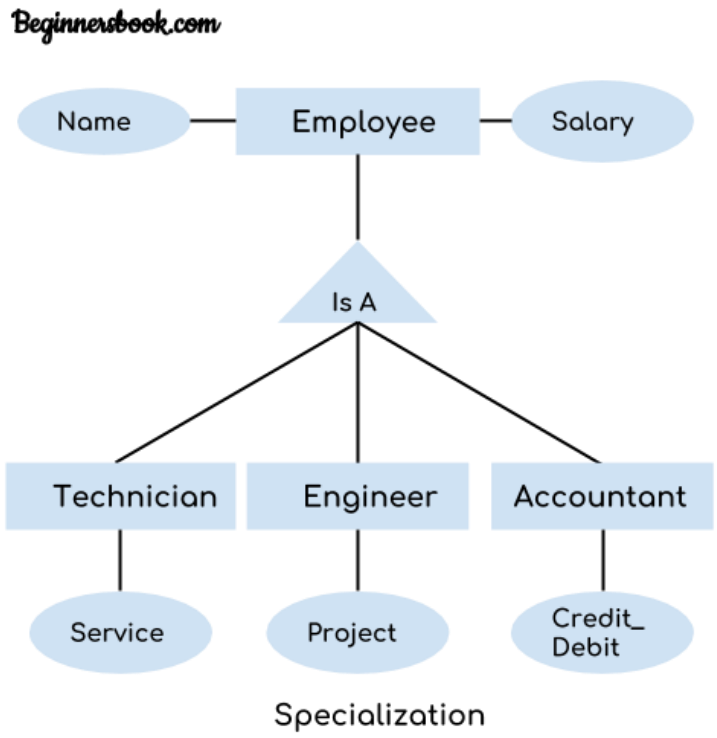
Generalization

Note:

1. Generalization uses bottom-up approach where two or more lower level entities combine together to form a higher level new entity.
2. The new generalized entity can further combine together with lower level entity to create a further higher level generalized entity.

Specialization is a process in which an entity is divided into sub-entities. You can think of it as a reverse process of generalization, in generalization two entities combine together to form a new higher level entity. Specialization is a top-down process.

The idea behind Specialization is to find the subsets of entities that have few distinguish attributes. For example – Consider an entity employee which can be further classified as sub-entities Technician, Engineer & Accountant because these sub entities have some distinguish attributes



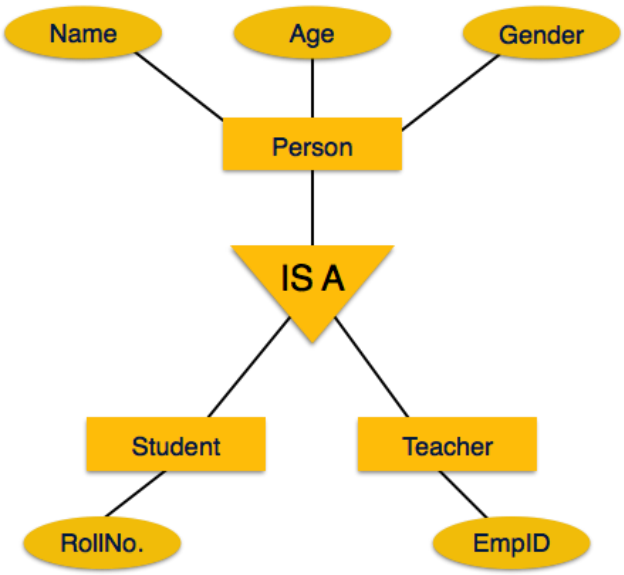
Specialization

Inheritance:

We use all the above features of ER-Model in order to create classes of objects in object-oriented programming. The details of entities are generally hidden from the user; this process known as **abstraction**.

Inheritance is an important feature of Generalization and Specialization. It allows lower-level entities to inherit the attributes of higher-level entities.

In the above diagram, we can see that we have a higher level entity “Employee” which we have divided in sub entities “Technician”, “Engineer” & “Accountant”. All of these are just an employee of a company, however their role is completely different and they have few different attributes. Just for the example, I have shown that Technician handles service requests, Engineer works on a project and Accountant handles the credit & debit details. All of these three employee types have few attributes common such as name & salary which we had left associated with the parent entity “Employee” as shown in the above diagram.



For example, the attributes of a Person class such as name, age, and gender can be inherited by lower-level entities such as Student or Teacher.

Facts about ER Diagram :

- ER model allows you to draw Database Design
- It is an easy to use graphical tool for modeling data
- Widely used in Database Design
- It is a GUI representation of the logical structure of a Database
- It helps you to identifies the entities which exist in a system and the relationships between those entities
- ER modeling, which aims at **conceptual representation** of the business requirements.
- Entities are classified as independent or dependent entities.
- An **independent entity** or **strong entity** is one that does not rely on another for identification.
- A dependent entity or weak entity is one that relies on another for identification.
- A **weak entity** set is represented by a doubly outlined box
- its relationship is represented by a doubly outlined diamond.
- The discriminator of a weak entity set is underlined with a dashed line**.**
- Attributes are the properties or descriptors of an entity. e.g., the entity course contains ID, name, credits, and faculty attributes. Attributes are represented by ellipses.
- The logically-grouped attributes are called compound attributes.
- An attribute can be **single valued or multi-valued**.
- A **multi-valued** attribute is represented by a **double ellipse**.
- The attribute which is used to uniquely identity an entity is called **a key attribute** or an

1. Identifier and it is indicated by **an underline**.

- **Derived attributes** are the attributes whose values are derived from other attributes. They are indicated by a **dotted ellipse**.

Notations of ER Model :