

Conceptual Design

- 🔊 E-R model: entities, attributes and its types, Relationship, Relationship sets, Generalization, Specialization, Aggregation
- 🔊 Relational Model: Relation, Domain, Tuple, Degree, cardinality
- 🔊 Relational Algebra operations: Select, Project, Cartesian Product, Union, Set difference, join

🔊 Entity-relationship (E-R) model

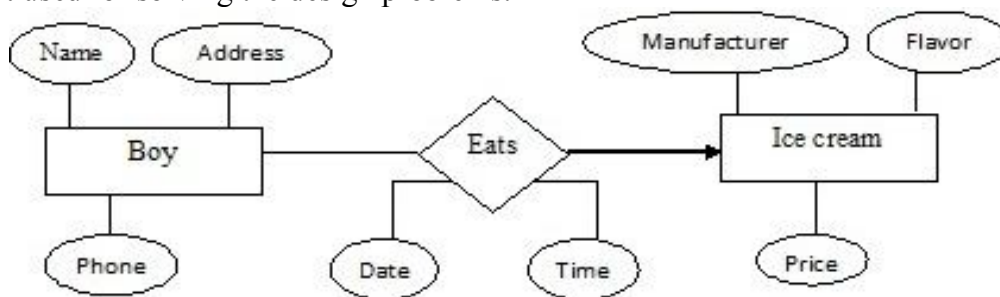
ER Model is a high-level data model, developed by Chen in 1976. This model defines the entities, relationships and their association with attributes for a specified system. It is useful in developing a conceptual design for the database & is very simple and easy to design logical view of data. The E-R model is mainly used for communication between database designers and end users during the analysis phase of database development.

Importance of ER Model:

- ER Model is plain and simple for designing the structure.
- It saves time.
- Without ER diagrams you cannot make a database structure & write production code.
- It displays the clear picture of the database structure.

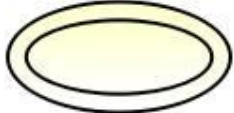
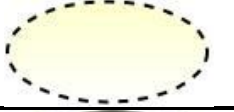
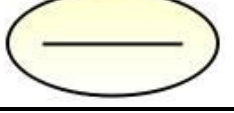
🔊 ER Diagrams

- ERD stands for Entity Relationship diagram.
- It is a graphical representation of an information system.
- ER diagram shows the relationship between objects, places, people, events etc. within system.
- It is a data modeling technique which helps in defining the business process.
- It used for solving the design problems.



➤ Notations/Symbols used in E-R diagrams

Notations	Representation	Description
	Rectangle	It represents the Entity.
	Ellipse	It represents the Attribute.
	Diamond	It represents the Relationship.
	Line	It represents the link between attribute and entity set to relationship set.
	Double Rectangle	It represents the weak entity.
	Composite Attribute	It represents composite attribute which can be divided into subparts.

	Multi valued Attribute	It represents multi valued attribute which can have many values for a particular entity. For eg. Mobile Number.
	Derived Attribute	It represents the derived attribute which can be derived from the value of related attribute.
	Key Attribute	It represents key attribute of an entity which have a unique value in a table. eg. Employee → EmpId (Employee Id is Unique).

Entities and Entity Sets

- An entity is a real world object that exists and it is distinguishable from other entities. A database can be modeled as a collection of entities. Example-Person, Bank, company, Department, Customer, Employee, Student, event, plant etc.
- An entity has a set of properties and some properties have the values. It may uniquely identify an entity. Example, a *Customer* with *Customer_id* property with value *C101* uniquely identifies that person.
- An entity may be concrete (real) such as Customer, book or it may be abstract (Conceptual) such as a loan, or a holiday.
- A set of entities of the same type that share the same properties or attributes is called as **Entity set**. The set of all customers at a given bank can be defined as the entity set *Customer*.
Two entity sets:
 - *Customer*, with properties *Customer_id*, *Customer_name*, *City*
 - *Account* with properties *Account_no* and *Balance*.

Attributes and its types

All the entities in the data model have associated with it a set of properties of entities is called as attributes. Example: *Customer* has *Cust_id*, *Cust_name* and *address*.

There are types of attributes has been classified Such as

Simple attribute:

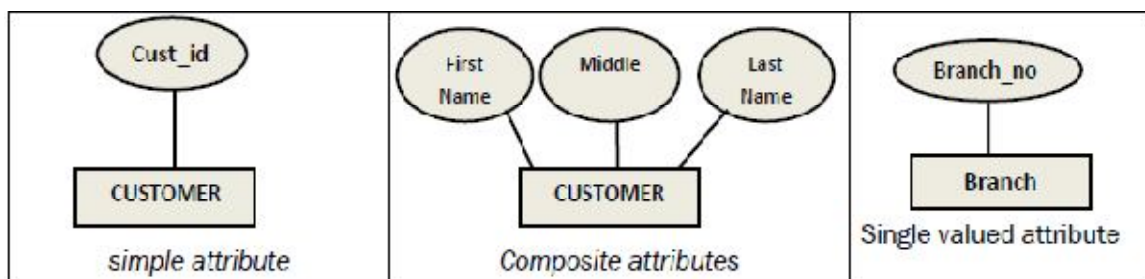
- A simple attribute is an attribute collected from a single component with an independent existence. Simple attributes cannot be further subdivided.
- Example: Customer's *Cust_id*, Students *Roll_No* etc.

Composite attribute:

- Composite attributes are composed of more than one simple attribute; each attribute with its own independent existence is called a composite attribute.
- Example: A *Cust_name* attributes of a *Customer* entity that can be further divided in to subparts i.e. *first name*, *middle name* and *last name*.

Single-valued attribute:

- Single valued attribute is an attribute which as single value (atomic) for each entity OR An attribute that holds a single value for each single entity type
- Example: Each *branch* has only single valued attributes is known as *branch_no*

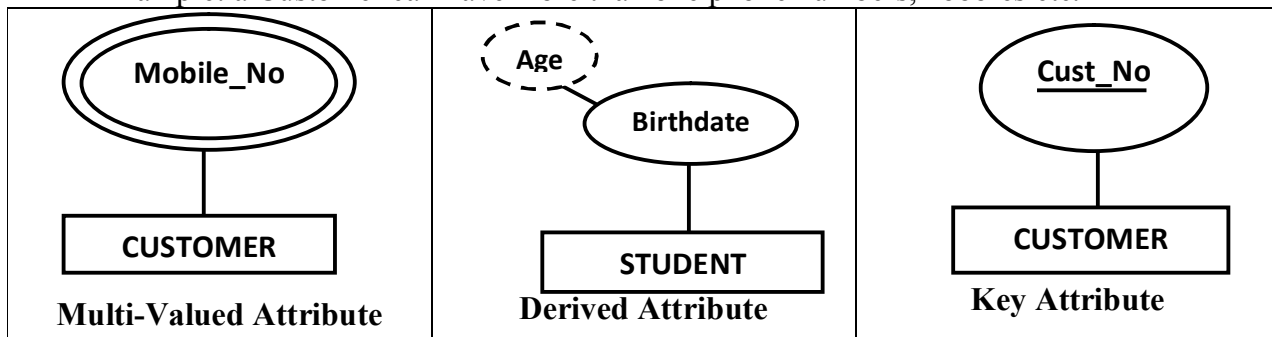


Multi-valued attribute:

- Multi-value attribute may contain more than one value. OR An attribute that holds multiple

values for each single entity type

- Example: a Customer can have more than one phone numbers, hobbies etc.



⊙ **Derived attribute:**

- In Derived attributes the value is derived from the other stored attribute. Derived attributes are attributes, which do not exist physical in the database, but their values are derived from other attributes presented in the database.
- Example: Age can be derived from Date of Birth (DOB).

⊙ **Stored attribute:**

- An attribute whose value cannot be derived from the values of other attributes is called a stored attribute. Example: Date of Birth (DOB).

⊙ **Null attribute**

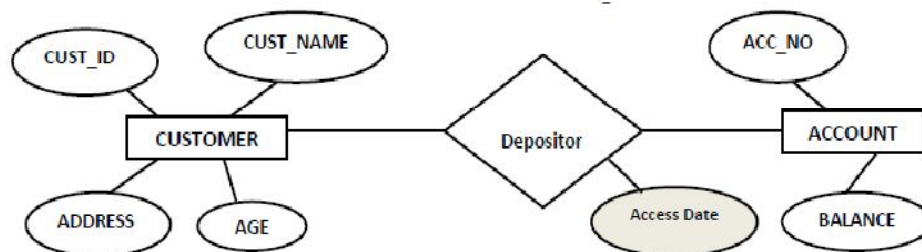
- The attribute which takes NULL value when an entity does not have the value for it. The Null attribute is an attribute whose value is unknown, unassigned, and missing information.

⊙ **Key Attributes**

- This attribute has the unique value for an entity which is used to identify a given row in the table; it is called a key attribute of an entity.
- Example: Cust_id is a key attribute which has a unique value which is used to identify a given row in the table.

⊙ **Descriptive attributes**

- A relationship may also have attributes called descriptive attributes.
- Consider a relationship set *Depositor* with entity sets *Customer* and *Account*. We can associate the attribute *access-date* to the *depositor* relationship to specify the most recent date on which a customer accessed an account as shown in Fig.



✚ **Relationship, Relationship sets, Degree and Cardinality**

➤ **Relationships**

The association among entities is called a relationship. A set of meaningful relationships between several entities. We use the diamond symbol to indicate relationships among several entities; it could be read from left to right.

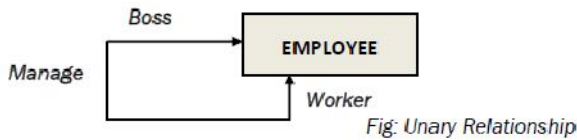
Example: Employee entity has a relationship works for with department.



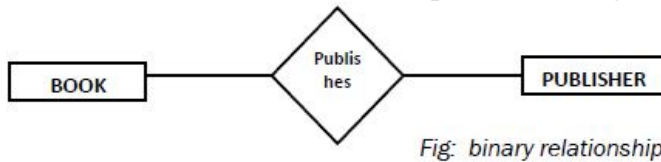
➤ **Degree of relationship :** Total number of entity sets participating in a relationship set is known

Types of relationships

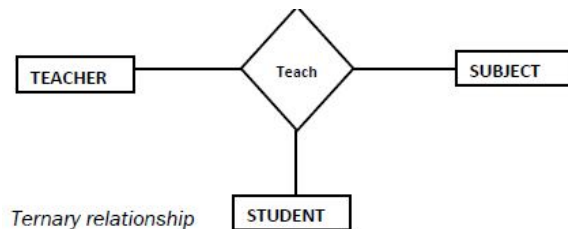
i) Unary relationship: A unary relationship exists when an association is maintained within a single entity. Example, boss and worker distinguish the two employees participating in the manage association as shown in following Fig.



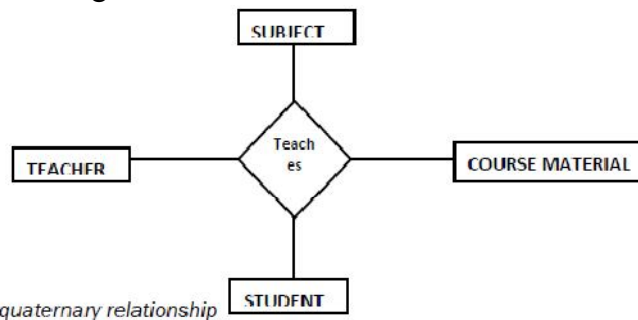
ii) Binary relationship: A binary relationship exists when two entities are associated. Example: the Book-Publisher relationship shown in Fig.



iii) Ternary relationship: A ternary relationship exists when there are three entities associated. For example, the entities Teacher, Subject and Student are related using a ternary relationship called 'Teaches' as shown in Fig.

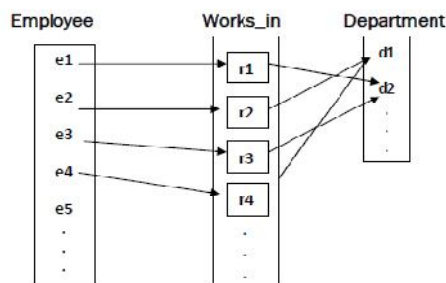


iv) Quaternary relationship: A quaternary relationship exists when there are four entities associated. Example: 'studies' where four entities are involved Student, Teacher, Subject and Course-material. It is shown in Fig.



➤ **Relationship Set:** A relationship set is a set of relationships of the same type.

Consider an example, employees work in different departments. Then relationship exists between employees and departments because each employee must belong to some department. Relation of all employees with department when combined makes the relationship set because each employee has same kind of relation with departments.



Here, Employee and Department are two entity sets, *r* stands for relationship between Employee and Department. Works_in is the relationship set as shown in Figure.

➤ **Mapping Constraints**

There are *two types* of mapping constraints:

(a) Mapping cardinalities

(b) Participation constraints

(a) Mapping Cardinalities (Cardinality Ratios)

- The numbers of entities of an entity set that are associated with entities of another entity set through a relationship set.
- Cardinalities indicates that a specific number of entity occurrence of related entity. Types of Relationship Mapping

Following are the types of Relationship Mapping,

1. One - to - One Relationship
2. One - to - Many Relationships
3. Many - to - One Relationships
4. Many - to - Many Relationships

1. One - to - One Relationship

- In One - to - One Relationship, one entity is related with only one other entity.
- One row in a table is linked with only one row in another table and vice versa.

Example: A Country can have only one Capital City.

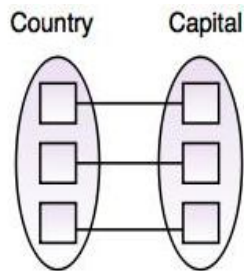


Fig. One to One Mapping

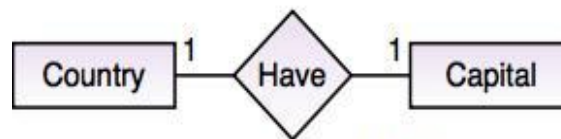


Fig. Representation in ER Diagram

2. One - to - Many Relationship

- In One - to - Many Relationship, one entity is related to many other entities.
- One row in a table A is linked to many rows in a table B, but one row in a table B is linked to only one row in table A. **Example:** One Department has many Employees.

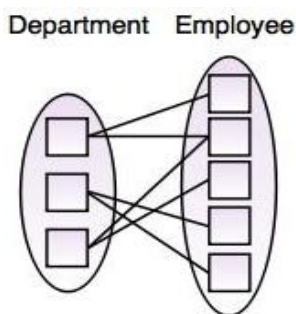


Fig. One to Many Mapping

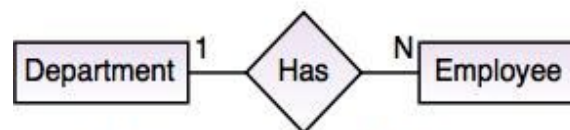


Fig. Representation in ER Diagram

3. Many - to - One Relationship

- In Many - to - One Relationship, many entities can be related with only one other entity.
- **For example:** No. of Employee works for Department.
- Multiple rows in Employee table are related with only one row in Department table.

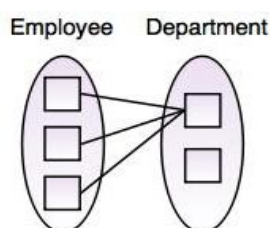


Fig. Many to One Mapping

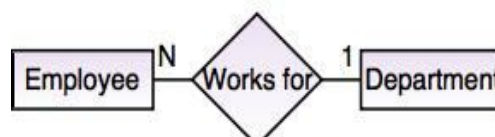


Fig. Representation in ER Diagram

4. Many - to - Many Relationship

- In Many - to - Many Relationship, many entities are related with the multiple other entities.
- This relationship is a type of cardinality which refers the relation between two entities.

Example: Various Books in a Library are issued by many Students.

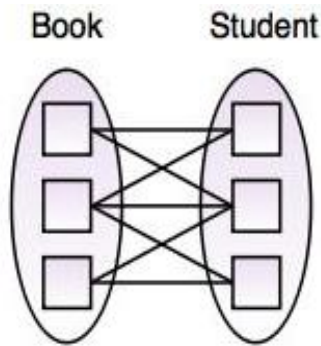


Fig. Many to Many Mapping

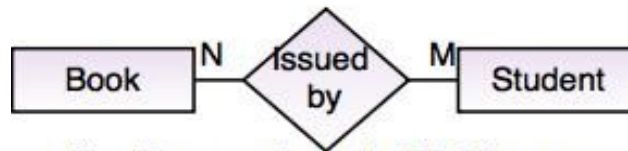


Fig. Representation in ER Diagram

(b) Participation Constraints

Participation concerns with the involvement of entities in a relationship. It specifies whether the existence of an entity depends on another entity. There are two types of Participation Constraints –

1. Total Participation
2. Partial Participation

1. Total Participation

- In Total Participation, every entity in the set is involved in some association of the relationship.
- It is indicated by a double line (====) between entity and relationship.



Fig. Total Participation

Example: Every Department must have a Manager.

2. Partial Participation

- In Partial Participation, not all entities in the set are involved in association of the relationship.
- It is indicated by a single line (——) between entity and relationship.



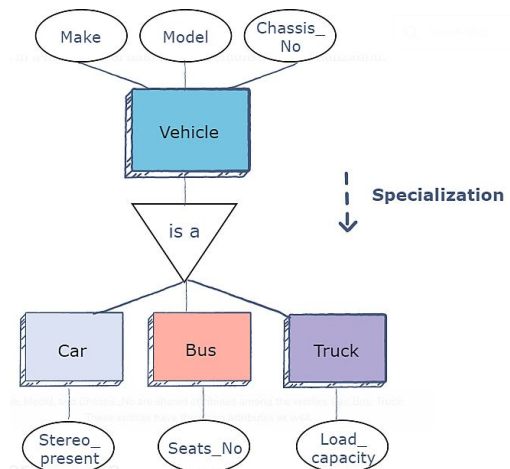
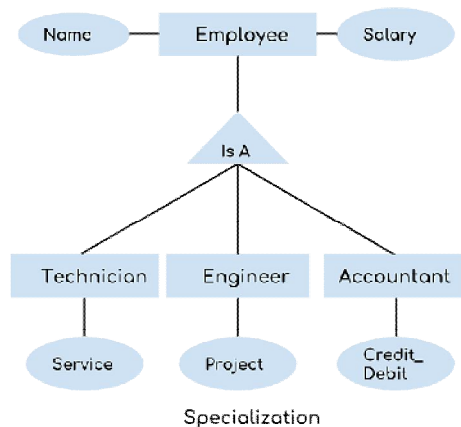
Fig. Partial Participation

Extended 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 the ER model.
- EER creates a design more accurate to database schemas.
- It reflects the data properties and constraints more precisely.
- Diagrammatic technique helps for displaying the EER schema.
- It includes the concept of specialization and generalization.
- It is used to represent a collection of objects that is union of objects of different of different entity types.

➤ Specialization

Specialization is a top-down approach in which a higher-level entity is divided into multiple *specialized* lower-level entities. In addition to sharing the attributes of the higher-level entity, these lower-level entities have *specific* attributes of their own. Specialization is used to find subsets of an entity that has a few different or additional attributes.

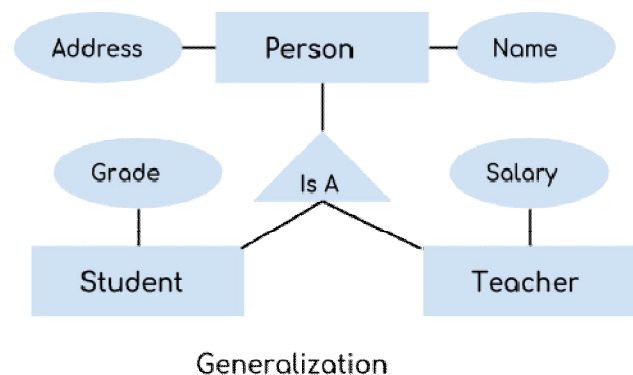
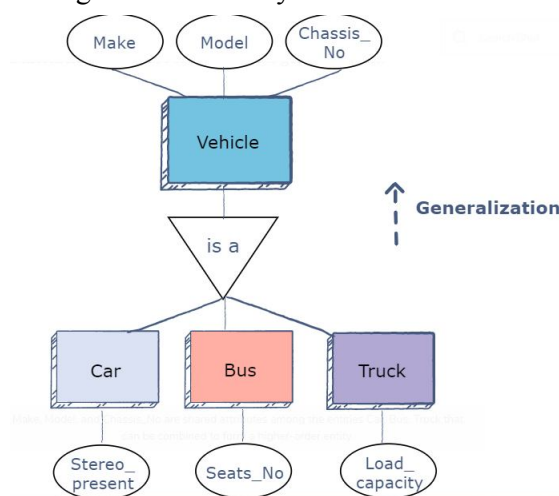


Example – Consider an entity employee which can be further classified as sub-entities Technician, Engineer & Accountant because these sub entities have some distinguish attributes. We have a higher level entity “Employee” which we have divided in sub entities “Technician”, “Engineer” & “Accountant”. 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.

➤ Generalization

Generalization is a bottom-up approach in which multiple lower-level entities are combined to form a single higher-level entity. Generalization is usually used to find common attributes among entities to form a generalized entity. It can also be thought of as the opposite of specialization.

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.

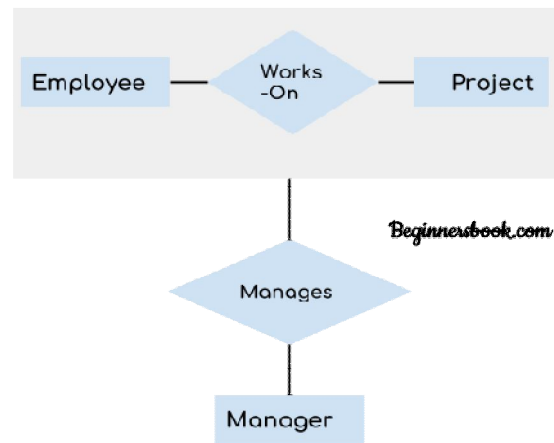


Example: Generalization process the entities Student and Teacher only has the specialized attributes Grade and Salary respectively and their common attributes (Name & Address) are now associated with a new entity Person which is in the relationship with both the entities (Student & Teacher).

➤ Aggregation

In aggregation, the relation between two entities is treated as a single entity. In aggregation, relationship with its corresponding entities is aggregated into a higher level entity.

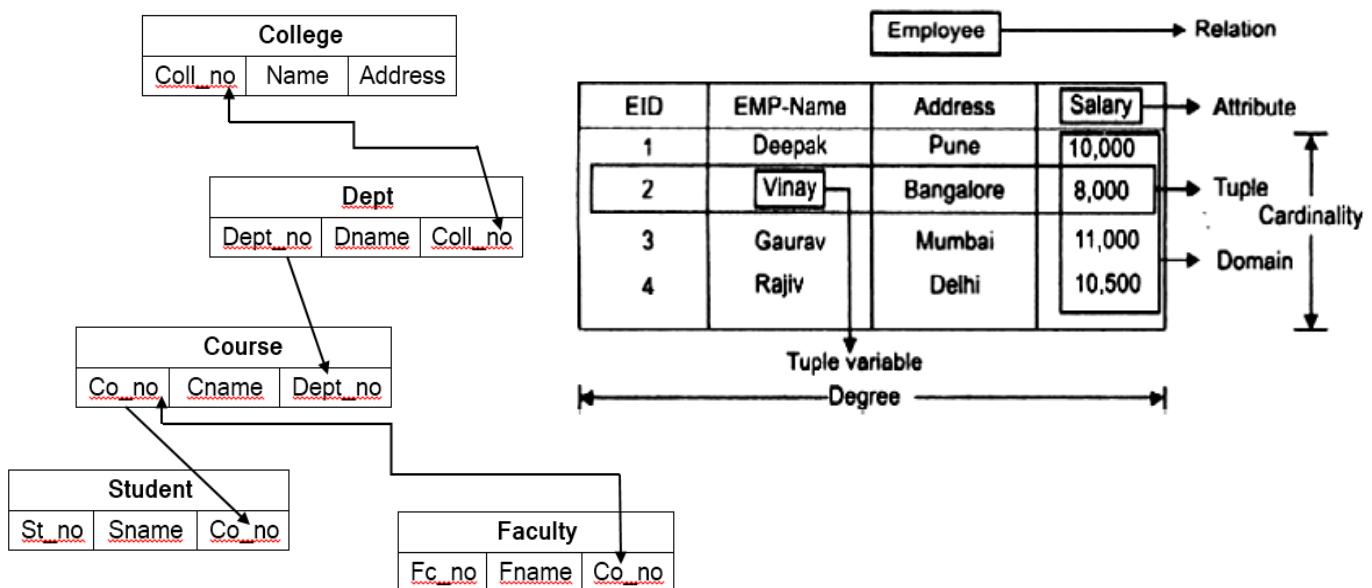
Example: A manager not only manages the employee working as well as manages the project. In such scenario if entity “Manager” makes a “manages” relationship with either “Employee” or “Project” entity



In these cases the relationship of two entities acts as one entity. In our example, the relationship “Works-On” between “Employee” & “Project” acts as one entity that has a relationship “Manages” with the entity “Manager”.

Relational Model

The relational database was developed by E. F. Codd at IBM in 1970. The relational model represents data is organized in rows and column structure i.e., two-dimensional tables and the relationship is maintained by storing a common field. All the information is stored in the form of row and columns. The basic structure of a relational model is tables. So, the tables are also called *relations* in the relational model. Each relation have a group of column with unique names and rows, where column represents attribute of an entity and rows represents records.



- ◆ **Attribute /field:** An attribute is a descriptive property or characteristics of an entity i.e. *Columns in a table are called attributes*. Example- EID Emp_name, Address etc.
- ◆ **Tuple:** Each row in the table is called tuple. A row contains all the information about any instance of the object.
- ◆ **Degree:** The number of column/attributes in a relation is called as degree. Example: In the employee Relation Degree is 4
- ◆ **Cardinality:** The number of rows (tuples) in a table is called cardinality. Example: In the employee Relation cardinality is 4
- ◆ **Domain:** Domain is the set of all permitted values or information for one or more attribute. Example: In relation Employee, domain attribute salary is 10000, 8000, 11000.

Properties of Relation:

- 1. Table names in the database should be unique

- Attribute names should be unique.
- Attributes & Tuples are unordered
- Each record in the table is unique i.e. no duplicate rows.
- There is only one (atomic) value for each attribute of a tuple.

◆ Advantages

- Use of SQL language to access data.
- Easier database design implementation and management
- Improves conceptual simplicity.
- Structural Independence-Changes in database structure without changing way to access the data
- Data independence.
- Multi-level relationships between data.

◆ Disadvantages

- Data anomalies.
- People need training if they want to use the system effectively and efficiently.
- Slower processing than hierarchical and network models.

E-R MODEL	RELATIONAL MODE
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.
Entity Relationship Model describes data as Entity set, Relationship set and Attribute.	Relational Model describes data in a table as Domain, Attributes, Tuples.
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.
E-R Model describes Mapping Cardinalities.	Relational Model does not describe mapping cardinalities.

🔊 Relational Algebra operations:

The relational algebra is a theoretical procedural query language associated with the relational model. It consists of a set of operations that take one or two relations (tables) as input and produce a new relation, on the request of the user to retrieve the specific information, as the output.

The relational algebra is very important due to many reasons.

- It provides a basic operation for relational model
- It is used as basis for implementing and optimizing queries in RDBMS's.
- The basic concepts of relational algebra are incorporated into the SQL language

The relational algebra uses various logical operator [\wedge (and), \vee (or), \neg (not)] and comparison operators ($<$, \leq , $=$, $*$, \geq , $>$) to construct composite and complex queries.

Query languages are of two types,

1. **Procedural language:** the user has to describe the specific procedure to retrieve the information from the database. Example: The Relational Algebra is a procedural language.
2. **Non-procedural language:** the user retrieves the information from the database without describing the specific procedure to retrieve it. Example: Tuple Relational Calculus and Domain Relational Calculus are non-procedural languages.

◆ Operations in Relational Algebra

Relational Algebra Fundamental operations are divided into two groups

► Basic set-oriented operations (Simple operations)

- Union (binary operations)
- Intersection (binary operations)
- Set difference (binary operations)
- Cartesian product (binary operations)

► Relational-oriented operations (Special operations)

- Selection (unary operations)

- Projection (unary operations)
- Joins (binary operations)
- Division (binary operations)
- Rename (binary operations)
- Assignment (binary operations)

The Selection, Projection and Rename operations are called **unary operations** because they operate only on one relation.

The other operations operate on pairs of relations are called **binary operations**

Basic set-oriented operations (Binary)		
Operation	Keyword	Symbol
Union	UNION	U
Intersection	INTERSECT	∩
Cartesian product	X	X
Set-Difference	MINUS	−

Relational-oriented operations (Unary)		
Operation	Keyword	Symbol
Selection	SELECT	σ
Projection	PROJECT	Π
Rename	RENAME	ρ
Assignment	←	←

Relational-oriented operations (Binary)		
Operation	Keyword	Symbol
Join	JOIN	⋈
Left Outer Join	LEFT OUTER JOIN	⋈ _L
Right Outer Join	RIGHT OUTER JOIN	⋈ _R
Full Outer Join	FULL OUTER JOIN	⋈ _F
Division	DIVIDE	⋈ _÷

Employee

EID	Name	Salary	Join year
1	John	16000	1994
2	Ramesh	5000	2000
3	Smith	8000	2001
4	Jack	6000	2000
5	Nilesh	15000	1998

Student

SID	Name	Fees
101	Smith	1000
102	Vijay	950
103	Gaurav	2000
104	Nilesh	1500
105	John	950

Basic set-oriented operations

➤ Union operation (U):

The union operation is a binary operation that is used to find union of tables. It combines the similar columns from two tables into one resultant table. Duplicate values are removed. Here relations are considered as sets. It is denoted by (U). A union operation $R \cup S$ to be valid, we require that two necessary conditions must be satisfied:

- Both relations have same number of attribute i.e. Degree of relations are same
- Data types of attributes in both the relations should be same.

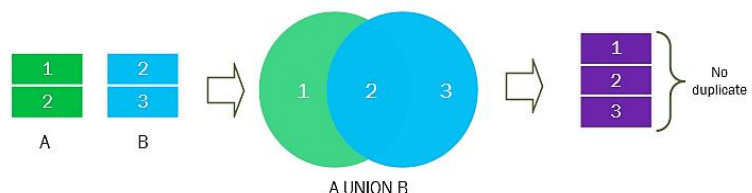
Above conditions, If R and S are two relations, which are union well-matched, resulting tuples of relation either in R or S or both. There are no duplicate tuples. **union operation: $R \cup S = S \cup R$**

Examples: If you want to find the names of all employees and names of all students jointly then

Π Name (Employee) U Π Name (Student)

Select name from employee UNION Select name from student;

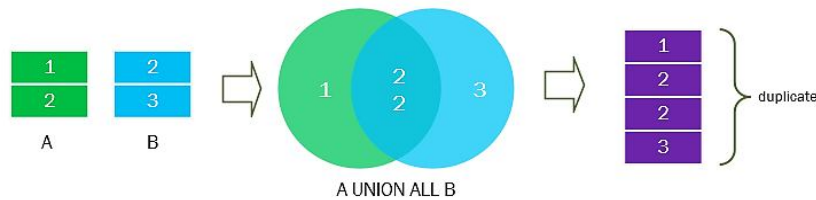
Name
John
Ramesh
Smith
Jack
Nilesh
Vijay
Gaurav



➤ UNION ALL

This operation is also similar to UNION, but it does not eliminate the duplicate records. It shows all the records from both the tables. All other features are same as UNION.

Select name from employee UNION ALL Select name from student;



➤ Intersection operation (\cap)

MySQL does not support the INTERSECT operator. Intersection is used to find common tuples between two tables. It is denoted by (\cap). Rules of Union operations are also applicable to intersection operation i.e. same degree and same domain. If R and S are two relations and we take intersection of these two relations then the resulting relation would be the set of tuples, which are in both R and S. $R \cap S = S \cap R$

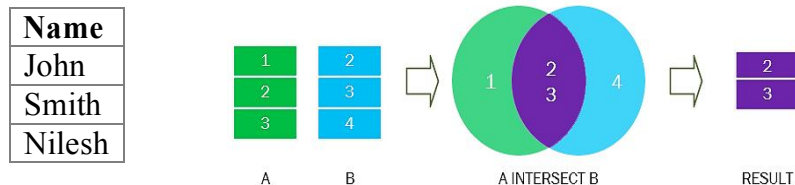
▪ Emulate INTERSECT using DISTINCT and INNER JOIN clause

Examples: find all the employees from Relation Employee those are also students.

Π Name (Employee) \cap Π Name (Student)

Select name from employee inner join student using(name);

The INNER JOIN clause returns rows from both left and right tables.



➤ Set-difference operation ($-$):

Set-difference operation is a binary operation which is used find tuples that are present in first table but not into other second table. It is denoted by ($-$). It removes the common tuples of two relations and produces a new relation having rest of the tuples of first relation. **MySQL does not support the MINUS operator.** You can use the LEFT JOIN clause to return the same result as MINUS operator.

Examples: If you want the names of those employees that are not students, then the

Π Name (Employee) $-$ Π Name (Student)

Select name from employee left join student using(name) where student.name is null;

Name
Ramesh
Jack

➤ Cartesian product operation (X):

Cartesian product is a binary operation which is used combine information of any two relations. It is denoted by (X). Suppose a relation R1 is having m tuples and other relation R2 is having n tuples then $R1 \times R2$ has $m \times n$ tuples. The Cartesian product needs not to be union compatible. It means they can be of different degree.

Select * from Paper, Student;

Paper		Student		Paper X Student			
Paper_Code	Paper_Name	Stud_ID	Stud_Name	Paper_Code	Paper_Name	Stud_ID	Stud_Name
1P	DBMS	1S	John	1P	DBMS	1S	John
2P	Java	2S	Smith	1P	DBMS	1S	John
3P	Networking			2P	Java	1S	John
				2P	Java	1S	John
				2P	Java	2S	Smith
				3P	Networking	2S	Smith
				3P	Networking	2S	Smith

➤ Selection or Restriction operation (σ):

- It is used to select certain rows or tuples of a table, so it performs on the table horizontally. It

- This command works on a single table and takes rows that meet a specified condition, copying them into a new table.
- As a result of this operation a new table is formed, without changing the original table i.e. Degree of table.
- Syntax: Selection condition (relation_name)
 σ <selection condition> (R)
- Selection condition is the Boolean expression on the attributes of relation R. The selection condition is represented as
 σ <Attribute name><comparison operator><constant value> or
 σ <Attribute name><comparison operator><attribute name> Where,
 - < Attribute name > is the name of an attribute (column) of the relation R.
 - < Comparison operator > is one of the comparison operators (=, #, <, <=, >, >=).
 - < Constant value > is the constant value from the attribute domain.

Clauses are connected by Boolean operators AND, OR and NOT.

Examples: Display all employees having salary greater than 9,000 from Employee.

σ Salary > 9000 (Employee)

Select * from employee where salary > 9000;

EID	Name	Salary	Joinyear
1	John	16000	1994
5	Nilesh	15000	1998

1) Display employees joined in year 2000.

σ Joinyear=2000 (Employee)

Select * from employee where joinyear=2000;

EID	Name	Salary	Joinyear
2	Ramesh	5000	2000
4	Jack	6000	2000

2) Select the tuples for all employees whose joining year is 1998 or salary is greater than 10000.

σ (Joinyear=1998) OR (Salary>10000) (Employee)

Select * from employee where Joinyear=1998 OR Salary>10000;

EID	Name	Salary	Joinyear
5	Nilesh	15000	1998

Select the tuples for all employees whose joining year is 1994 and salary is greater than 10000.

σ (Joinyear=1994) AND (Salary>10000) (Employee)

Select * from employee where Joinyear=1994 AND Salary>10000;

EID	Name	Salary	Joinyear
1E	John	16000	1994

3) Display employee have EID is 3E.

σ EID='3E' (Employee)

EID	Name	Salary	Joinyear
3E	Smith	8000	2001

➤ The Project Operation (Π)

- The project operation selects certain columns (Attribute) from a table while remove others. It is denoted by sigma(Π).
- Create a new table by selecting only specified attributes of the existing table.
- Remove duplicate tuples in the newly formed relation.
- representation of Projection operation : Π <Attribute list >(R)

Examples: Display all employee Names with Salary and Joinyear

Π Name, Salary, Joinyear (Employee)

Select Name, Salary, Joinyear from employee

Name	Salary	Joinyear
John	16000	1994
Ramesh	5000	2000
Smith	8000	2001
Jack	6000	2000
Nilesh	15000	1998

➤ Composition of Select and Project Operations

The relational operations select and project can be combined to form a complex query.

Display the Name of Employee having Salary greater than 8000.

Π Name (σ Salary > 8000 (Employee)) or σ Salary > 8000 (Π Name (Employee))

Name
John
Smith
Nilesh

Display the EID and Name of Employee having Joining year is 2000.

Π EID, Name (σ Joinyear='2000' (Employee))

EID	Name
2	Ramesh
4	Jack

➤ The Rename Operation (ρ)

- This is a unary operator which changes attribute names for a relation without changing any values.
- The rename operation provides database designers to rename the output relation. Notation: ρ OldName \rightarrow NewName(r)
- Example, ρ Name \rightarrow Emp_name(Empolyee)
- If there are two or more attributes involved in a renaming operation, then ordering is meaningful: e.g., ρ Eid,Salary \rightarrow Emp_id,Payment(Employees)

✚ Mechanisms for joining relations (Join Operation) -inner joins, outer joins and its types

- The process of linking tables is called joining. MySQL handles queries across more than one table through the use of JOINS.
- JOINS are clauses in MySQL statements that link two tables together, usually based on the keys that define the relationship between those two tables.
- The purpose of Join statement is to combine data or rows from two or more tables based on a common field between them.
- MySQL can get data from several related tables by performing either a physical or virtual join on the tables.
- The WHERE Clause is mostly used to perform the JOIN function with two or more tables have columns.

Join Syntax: SELECT [table_name.column_name1, table_name.column_name2,...] FROM table_name1 join_type JOIN table_name2 ON (join_condition);

Types of SQL Joins:

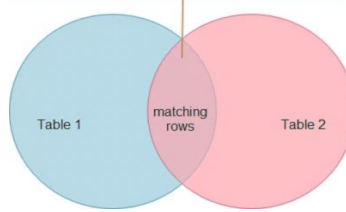
- Inner Join(Equi join)
- Outer Join
 - Left Outer Join
 - Right Outer Join
 - Full Outer Join
- Self-join
- Non-equi join

- Natural join

⊙ **Inner join:** The INNER JOIN is the most common type of join. It returns only those rows that are a match (common) in both joined tables. i.e. it gives Intersection of two tables.

Syntax: SELECT col1,col2 FROM table1 INNER JOIN table2 ON table1.column_name = table2.column_name;

Rows in Table 1 that have matching rows in Table 2



emp

emp_id	emp_name	hire_date	dept_id
1	Eshan	01-May-2001	4
2	Tony	15-July-2002	1
3	Sara	18-Oct-2005	5
4	Ravi	03-Jan-2007	3
5	Mahesh	24-June-2008	null

dept

dept_id	dept_name
1	Administration
2	Customer Service
3	Finance
4	HR
5	Sales

Retrieves the employee's id, name, hiring date and their department by joining the 'emp' and 'dept' tables together using the common 'dept_id' column. It removes those employees who are not assigned to any department.

- **On Clause:** SELECT e.emp_id, e.emp_name, e.hire_date, d.dept_name FROM emp e INNER JOIN dept d ON e.dept_id=d.dept_id ORDER BY emp_id;
- **Using Clause:** SELECT e.emp_id, e.emp_name, e.hire_date, d.dept_name FROM emp e JOIN dept d USING(dept_id);
- **Where Clause:** SELECT e.emp_id, e.emp_name, e.hire_date, d.dept_name FROM emp e, dept d where e.dept_id=d.dept_id ORDER BY emp_id;

The result set contains only those employees whose 'dept_id' value is present and that value also exists in the 'dept_id' column of the dept table. Equality operator (=) is known as EQUI JOIN.

emp_id	emp_name	hire_date	dept_name
1	Eshan	01-May-2001	HR
2	Tony	15-July-2002	Administration
3	Sara	18-Oct-2005	Sales
4	Ravi	03-Jan-2007	Finance

An inner join connects two tables on a column with the same data type. Only the rows where the column values match are returned; unmatched rows are not returned.

⊙ **Non-Equi Join :** When a joining condition contains any operator other than equality operator like > < >= <= then the join is known as NON-EQUI JOIN

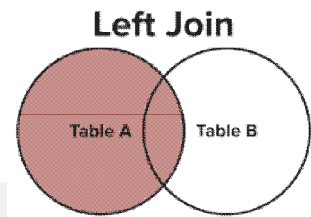
Example: `SELECT e.emp_id, e.emp_name, e.hire_date, d.dept_name FROM emp e ,dept d where e.dept_id > d.dept_id ORDER BY emp_id;`

- **Outer join:** Outer join returns all matching rows and non-matching rows from both tables. The outer join operator in Oracle (+) is used on one side of the join condition only.

☉ **Left outer join:** A LEFT JOIN return all rows from the “left” table, and matching rows from the “right” table. If there are no matches in the right table, return Null values for those columns.

Syntax: `SELECT column_names FROM table_name1 LEFT JOIN table_name2 ON table_name1.column_name=table_name2.column_name;`

Example: `SELECT e.emp_id, e.emp_name, e.hire_date, d.dept_name FROM emp e LEFT JOIN dept d ON e.dept_id=d.dept_id ORDER BY emp_id;`



emp_id	emp_name	hire_date	dept_name
1	Eshan	01-May-2001	HR
2	Tony	15-July-2002	Administration
3	Sara	18-Oct-2005	Sales
4	Ravi	03-Jan-2007	Finance
5	Mahesh	24-June-2008	null

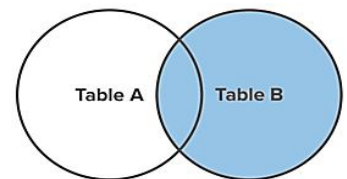
If there is a row but no match in then the row contains NULL values

in the left table the right table, associated result

- ☉ **Right outer join:** A right outer join return all rows from the “right” table, and matching rows from the “left” table. If there are no matches in the left table, return Null values for those columns.

Syntax: `SELECT column_names FROM table_name1 RIGHT JOIN table_name2 ON table_name1.column_name=table_name2.column_name;`

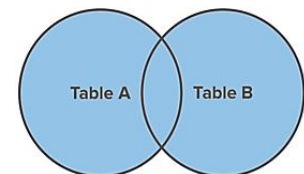
Example: `SELECT e.emp_id, e.emp_name, e.hire_date, d.dept_name FROM emp e RIGHT JOIN dept d ON e.dept_id=d.dept_id;`



emp_id	emp_name	hire_date	dept_name
2	Tony	15-July-2002	Administration
NULL	NULL	NULL	Customer Service
4	Ravi	03-Jan-2007	Finance
3	Sara	18-Oct-2005	Sales

Retrieve the names of all departments as well as the details of employees who're working in that department. But in some department no employee is working currently.

- ☉ **Full Join:** This will display the all matching and the non-matching records from both tables. SQL full outer join is used to combine the result of both left and right outer join and returns all rows it's matched or unmatched from the both tables.



Syntax: `SELECT column_names FROM table_name1 FULL JOIN table_name2 ON table_name1.column_name = table_name2.column_name;`

Example: `SELECT e.emp_id, e.emp_name, e.hire_date, d.dept_name FROM emp e FULL JOIN dept d ON e.dept_id=d.dept_id ORDER BY emp_name;`

Some databases, such as Oracle, MySQL do not support full joins. In that case you can use the UNION ALL operator to combine the LEFT JOIN and RIGHT JOIN as follows:

`SELECT e.emp_id, e.emp_name, e.hire_date, d.dept_name FROM emp e LEFT JOIN dept d ON e.dept_id = d.dept_id UNION ALL SELECT e.emp_id, e.emp_name, e.hire_date, d.dept_name FROM emp e RIGHT JOIN dept d ON e.dept_id = d.dept_id ORDER BY emp_name;`

emp_id	emp_name	hire_date	dept_name
--------	----------	-----------	-----------

1	Eshan	01-May-2001	HR
1	Eshan	01-May-2001	HR
5	Mahesh	24-June-2008	null
4	Ravi	03-Jan-2007	Finance
4	Ravi	03-Jan-2007	Finance
3	Sara	18-Oct-2005	Sales
3	Sara	18-Oct-2005	Sales
2	Tony	15-July-2002	Administration
2	Tony	15-July-2002	Administration

⊙ **Self join** :Self-Join is a specific type of Join. In Self Join, a table is joined with itself (Unary relationship). A self-join simply specifies that each rows of a table is combined with itself and every other row of the table.

`SELECT a.emp_name, b.dept_id, a.emp_id FROM emp a, emp b WHERE a.emp_id > emp_id;`

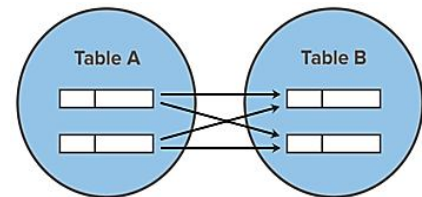
⊙ **Natural join**: The natural join is a type of Equi Join and is structured in such a way that, columns with same name of associated tables will appear once only.

Syntax: `SELECT * FROM table1 NATURAL JOIN table2;`

`SELECT * FROM emp NATURAL JOIN dept; or SELECT * FROM emp, dept;`

⊙ **Cross Join (Cartesian Products)**: If you don't specify a join condition when joining two tables, database system combines each row from the first table with each row from the second table. This type of join is called a cross join or a Cartesian product.

If there are "x" rows in table1 and "y" rows in table2 then the cross join result set have x*y rows. It normally happens when no matching join columns are specified. If a join condition is not specified, Oracle performs a Cartesian product.



Syntax: `SELECT * FROM [TABLE1] CROSS JOIN [TABLE2];`

OR

`SELECT * FROM [TABLE_NAME1], [TABLE_NAME2] ;`

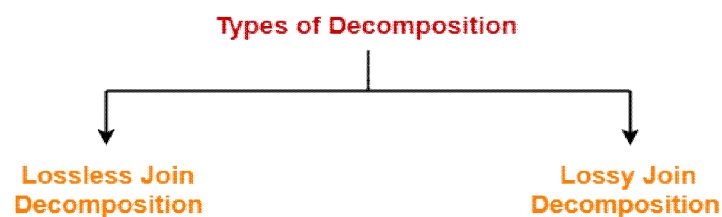
Example: `SELECT e.emp_id, e.emp_name, e.hire_date, d.dept_name FROM emp e CROSS JOIN dept d;`

OR

`SELECT * FROM emp, dept;`

⊙ Relational Decomposition

- When a relation in the relational model is not in appropriate normal form then the decomposition of a relation is required.
- Decomposition is the process of dividing given table (relation R) into two or more tables (relation R1,R2).
- If the relation has no proper decomposition, then it may lead to problems like loss of information.
- Decomposition is used to eliminate some of the problems of bad design like anomalies, inconsistencies, and redundancy.



- "The decomposition of relation R into R1 and R2 is **lossy** when the join of R1 and R2 does not produce the original relation as in R." This is Lossy join decomposition is also known as **careless decomposition**.
- One of the disadvantages some information is lost during retrieval of original table.
- **Example: Student Table**

Roll_no	Sname	Dept
111	parimal	COMPUTER
222	parimal	ELECTRICAL

This relation is decomposed into two relation no_name and name_dept:

```
create table Stu_name as select roll_no, sname from student;
create table Name_dept as select sname, dept from student;
```

Stu_name

Roll_no	Sname
111	parimal
222	parimal

Name_dept

Sname	Dept
parimal	COMPUTER
parimal	ELECTRICAL

In lossy decomposition, Extra tuples are generated when a natural join is applied to the relations in the decomposition.

```
select * from Stu_name natural join Name_dept;
```

Roll_no	Sname	Dept
111	parimal	COMPUTER
111	parimal	ELECTRICAL
222	parimal	COMPUTER
222	parimal	ELECTRICAL

Extra Tuples

The above decomposition is a bad decomposition or Lossy decomposition. Extra tuples make the identification of the original tuples difficult.

➤ Lossless Join:

- "The decomposition of relation R into sub relations R1, R2.....Rn is lossless when the join of the sub relations results in the same relation R". This decomposition is called lossless join decomposition.

$$R_1 \bowtie R_2 \bowtie R_3 \dots \dots \bowtie R_n = R \quad \text{Where } \bowtie \text{ is a natural join operator}$$

- A relational table is decomposed into two or more smaller tables, in such a way the designer can capture the accurate content of the original table by joining the decomposed parts. This is called lossless-join or non-additive join decomposition.
- Consider **STUDENT** table with three attribute roll_no ,sname and department.

Roll_no	Sname	Dept
111	parimal	COMPUTER
222	parimal	ELECTRICAL

This relation is decomposed into two relations Stu_name and Stu_dept:

```
create table Stu_name as select roll_no, sname from student;
create table Stu_dept as select roll_no, dept from student;
```

Stu_name table

Roll_no	Sname
111	parimal
222	parimal

Stu_dept table

Roll_no	Dept
111	COMPUTER
222	ELECTRICAL

Now, when these two relations are joined on the common column 'roll_no', the resultant relation will

```
select * from Stu_name natural join Stu_dept;
```

Roll_no	Sname	Dept
---------	-------	------

111	parimal	COMPUTER
222	parimal	ELECTRICAL

In lossless decomposition, no any extra tuples are generated when a natural joined is applied to the relations in the decomposition.