Module 3

Relationship Model and Relational Algebra

Topics

- Introduction to Relational Model
- Mapping the ER and EER model to Relational Model
- Relational Schema Design
- Introduction to Relational Algebra
- Relational Algebra - Operators
- Relational Algebra – Queries

• Introduction to Relational Model

The relational Model was proposed by E.F. Codd to model data in the form of relations or tables. After designing the conceptual model of the Database using ER diagram, we need to convert the conceptual model into a relational model which can be implemented using any RDBMS language like Oracle SQL, MySQL, etc. So we will see what the Relational Model is.

What is the Relational Model?
The relational model represents how data is stored in Relational Databases. A relational database stores data in the form of relations (tables). Consider a relation STUDENT with attributes ROLL_NO, NAME, ADDRESS, PHONE, and AGE shown in Table 1.

STUDENT

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

IMPORTANT TERMINOLOGIES

- **Attribute:** Attributes are the properties that define a relation. e.g.; **ROLL_NO**, **NAME**
- **Relation Schema:** A relation schema represents the name of the relation with its attributes. e.g.; STUDENT (ROLL_NO, NAME, ADDRESS, PHONE, and AGE) is the relation schema for STUDENT. If a schema has more than 1 relation, it is called Relational Schema.
- **Tuple:** Each row in the relation is known as a tuple. The above relation contains 4 tuples, one of which is shown as:

1	RAM	DELHI	9455123451	18

- **Relation Instance:** The set of tuples of a relation at a particular instance of time is called a relation instance. Table 1 shows the relation instance of STUDENT at a particular time. It can change whenever there is an insertion, deletion, or update in the database.

- **Degree:** The number of attributes in the relation is known as the degree of the relation. The **STUDENT** relation defined above has degree 5.
- **Cardinality:** The number of tuples in a relation is known as cardinality. The **STUDENT** relation defined above has cardinality 4.
- **Column:** The column represents the set of values for a particular attribute. The column **ROLL_NO** is extracted from the relation STUDENT.

ROLL_NO
1
2
3
4

- **NULL Values:** The value which is not known or unavailable is called a NULL value. It is represented by blank space. e.g.; PHONE of STUDENT having ROLL_NO 4 is NULL.
-

Constraints in Relational Model

While designing the Relational Model, we define some conditions which must hold for data present in the database are called Constraints. These constraints are checked before performing any operation (insertion, deletion, and updation) in the database. If there is a violation of any of the constraints, the operation will fail.

Domain Constraints: These are attribute-level constraints. An attribute can only take values that lie inside the domain range. e.g; If a constraint AGE>0 is applied to STUDENT relation, inserting a negative value of AGE will result in failure.

Key Integrity: Every relation in the database should have at least one set of attributes that defines a tuple uniquely. Those set of attributes is called keys. e.g.; ROLL_NO in STUDENT is a key. No two students can have the same roll number. So a key has two properties:

- It should be unique for all tuples.
- It can’t have NULL values.

Referential Integrity: When one attribute of a relation can only take values from another attribute of the same relation or any other relation, it is called referential integrity. Let us suppose we have 2 relations

STUDENT

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

BRANCH

BRANCH_CODE	BRANCH_NAME
CS	COMPUTER SCIENCE
IT	INFORMATION TECHNOLOGY
ECE	ELECTRONICS AND COMMUNICATION ENGINEERING
CV	CIVIL ENGINEERING

BRANCH_CODE of STUDENT can only take the values which are present in BRANCH_CODE of BRANCH which is called referential integrity constraint. The relation which is referencing another relation is called REFERENCING RELATION (STUDENT in this case) and the relation to which other relations refer is called REFERENCED RELATION (BRANCH in this case).

Advantages:

- Simple model
- It is Flexible
- It is Secure
- Data accuracy
- Data integrity
- Operations can be applied easily

Disadvantage:

- Not good for large database
- Relation between tables become difficult some time

- ## Mapping ER diagrams into relational schemas

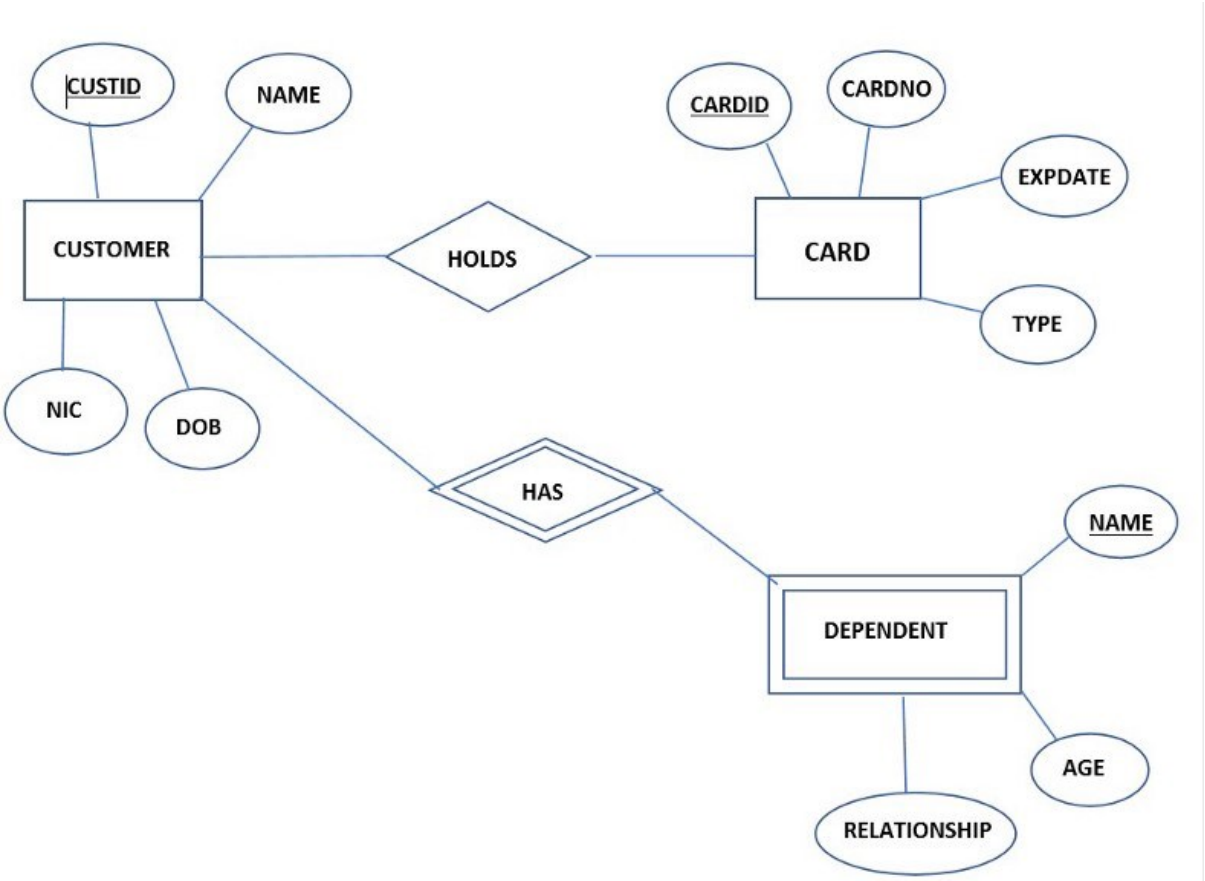
Follow the steps one by one to get it done.

1. ***Mapping strong entities.***
2. ***Mapping weak entities.***
3. ***Map binary one-to-one relations.***
4. ***Map binary one-to-many relations***
5. ***Map binary many-to-many relations.***
6. ***Map multivalued attributes.***
7. ***Map N-ary relations***

Let's go deep with the examples.

1. Mapping strong entities.

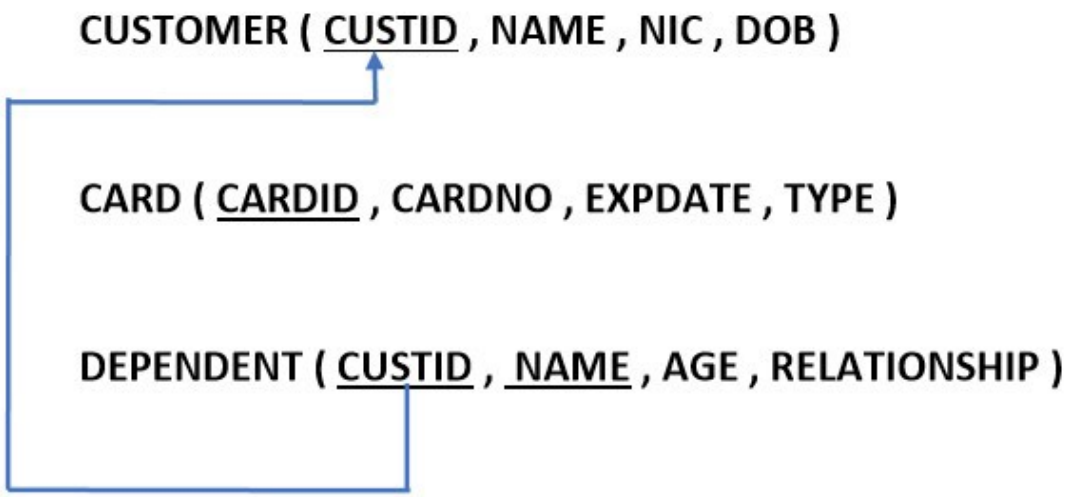
2. Mapping weak entities.



Above it shows an ER diagram with its relationships. You can see there are two strong entities with relationships and a weak entity with a weak relationship.

When you going to make a relational schema first you have to identify all entities with their attributes. You have to write attributes in brackets as shown below. Definitely you have to underline the primary keys. In the above **DEPENDENT** is a weak entity. To make it strong go through the weak relationship and identify the entity which connects with this. Then you have written that entity's primary key inside the weak entity bracket.

Then you have to map the primary key to where you took from as shown below.

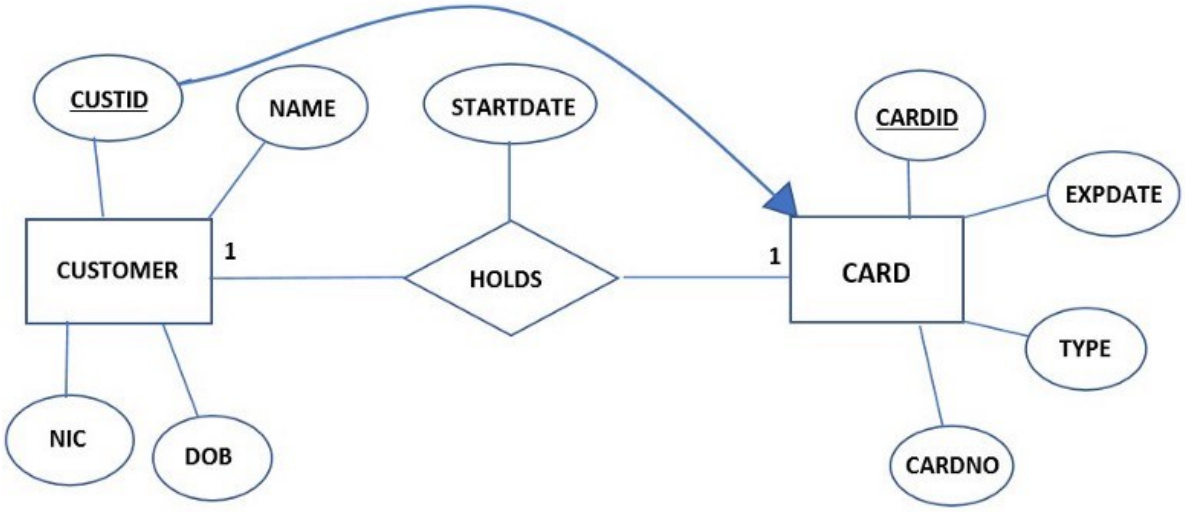


3. Map binary one to one relations.

Let's assume that the relationship between CUSTOMER and CARD is one to one.

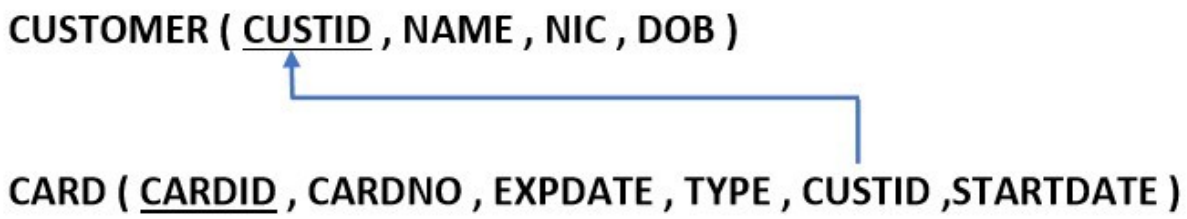
There are three occasions where one to one relations take place according to the participation constraints.

I. Both sides have partial participation.



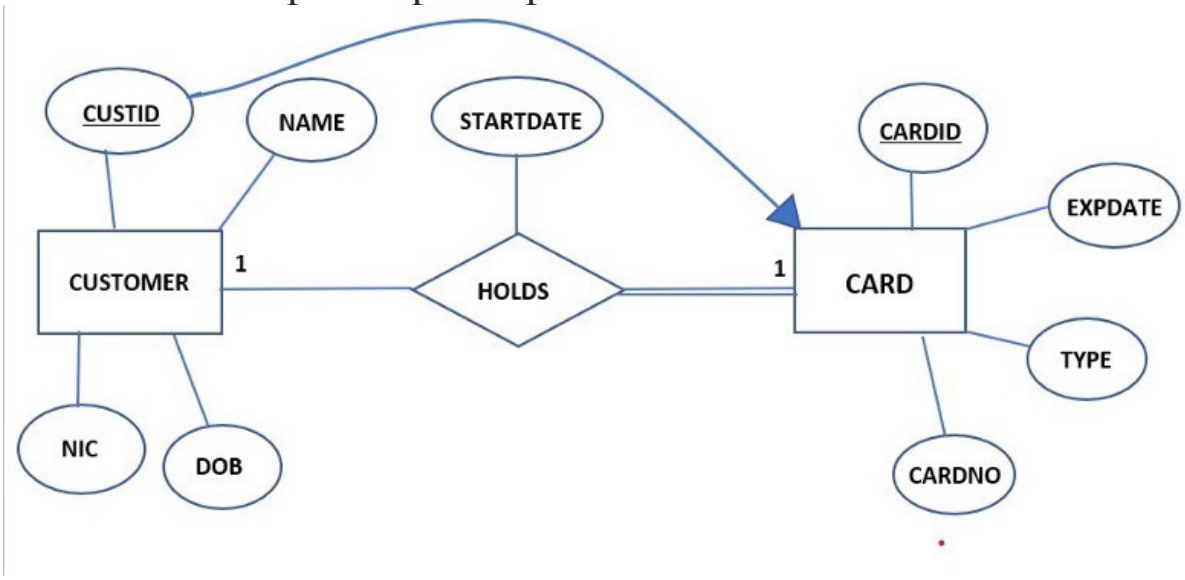
When both sides have partial participation you can send any of the entity's primary key to others. At the same time, if there are attributes in the relationship between those two entities, it is also sent to other entity as shown above.

So, now let us see how we write the relational schema.



Here you can see I have written **CUSTID** and **STARTDATE** inside the **CARD** table. Now you have to map **CUSTID** from where it comes. That's it.😁

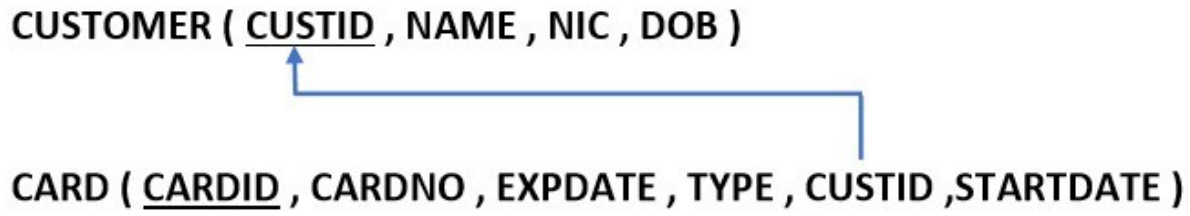
II. One side has partial participation.



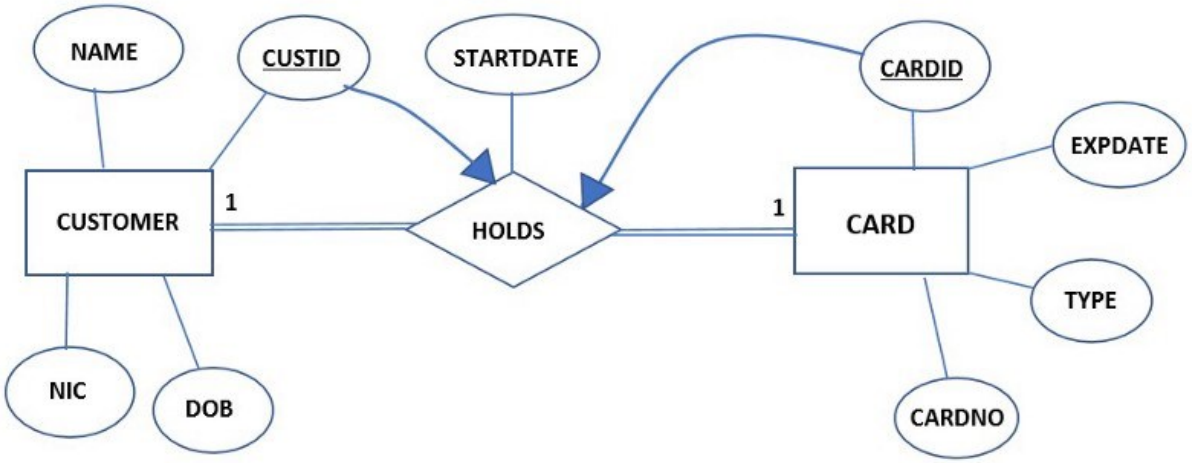
You can see between the relationship and **CARD** entity it has total participation.

When there is total participation definitely the primary of others comes to this. And also if there are attributes in the relationship it also comes to total participation side.

Then you have to map them as below.



III. Both sides have total participation



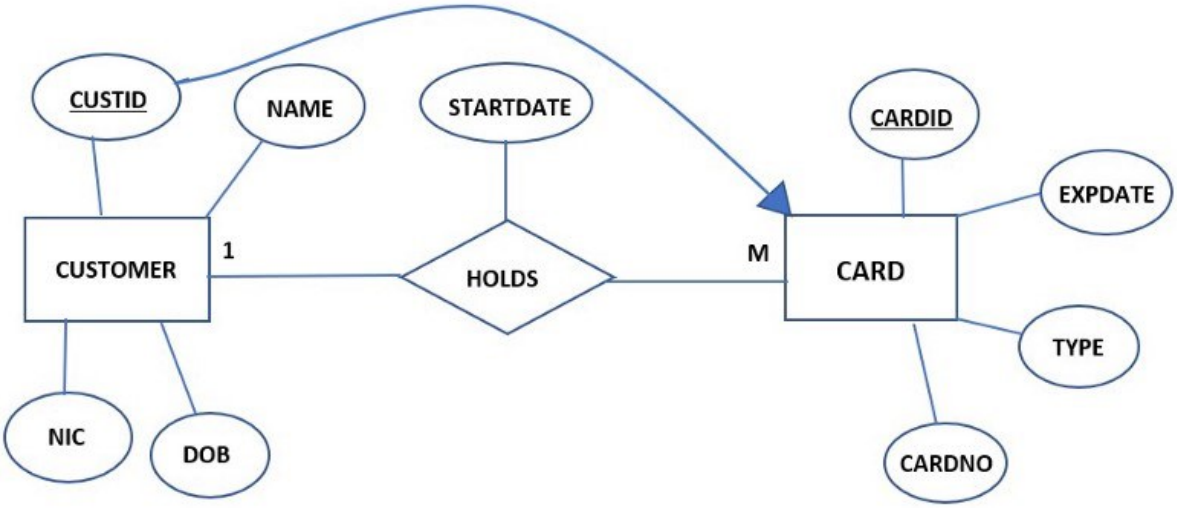
If both sides have total participation you need to make a new relationship with a suitable name and merge entities and the relationship.

Following it shows how we should write the relation.

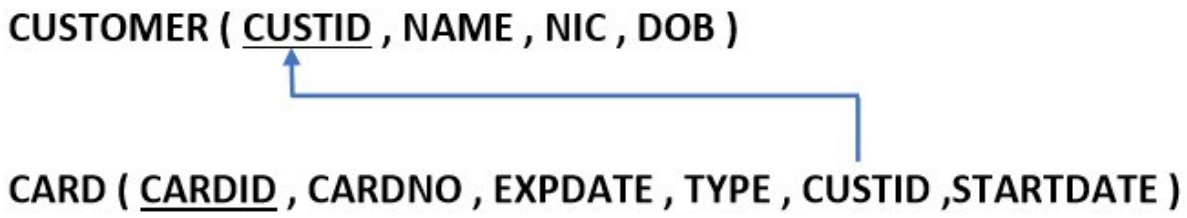
CUST_HOLD(CUSTID , NAME , NIC , DOB , CARDID , CARDNO , EXPDATE , TYPE ,STARTDATE)

Now let us see how to map one to many relations.

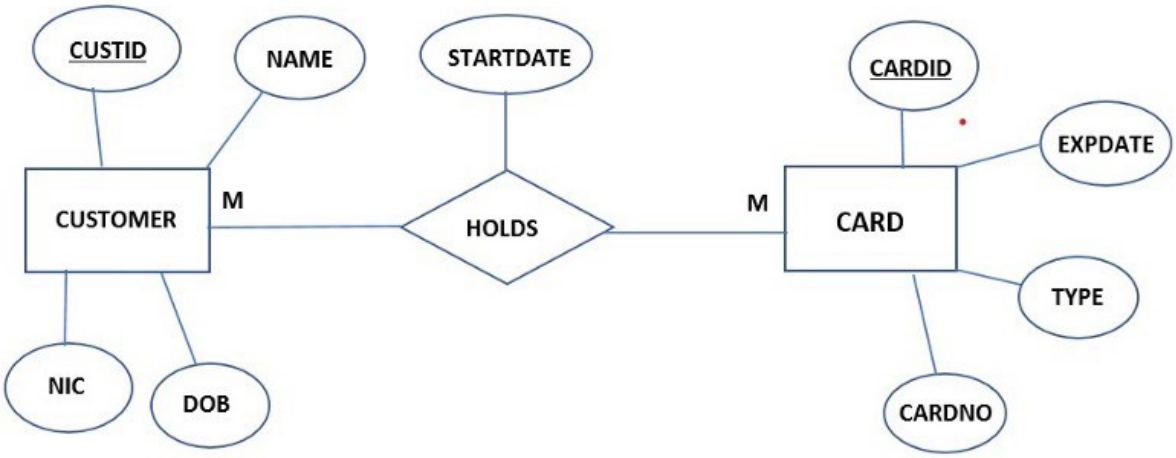
4. Map binary one-to-many relations



If it is one-to-many, always to the many side other entities' primary keys and the attributes in the relationship go to the *many* side. No matter about participation. And then you have to map the primary key.

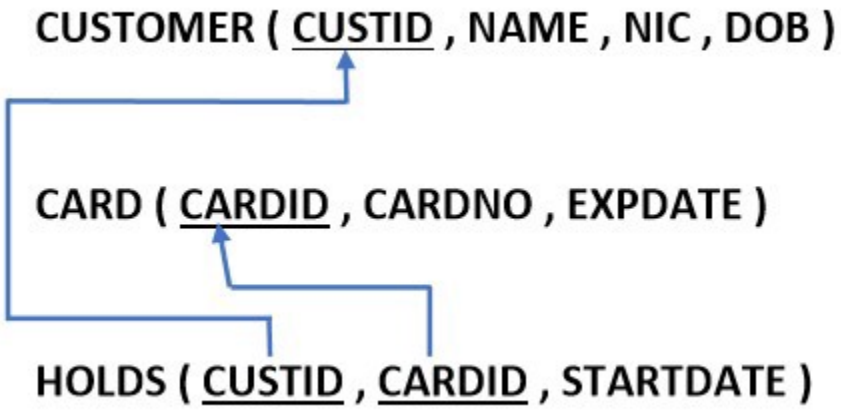


5. Map binary many to many relations.

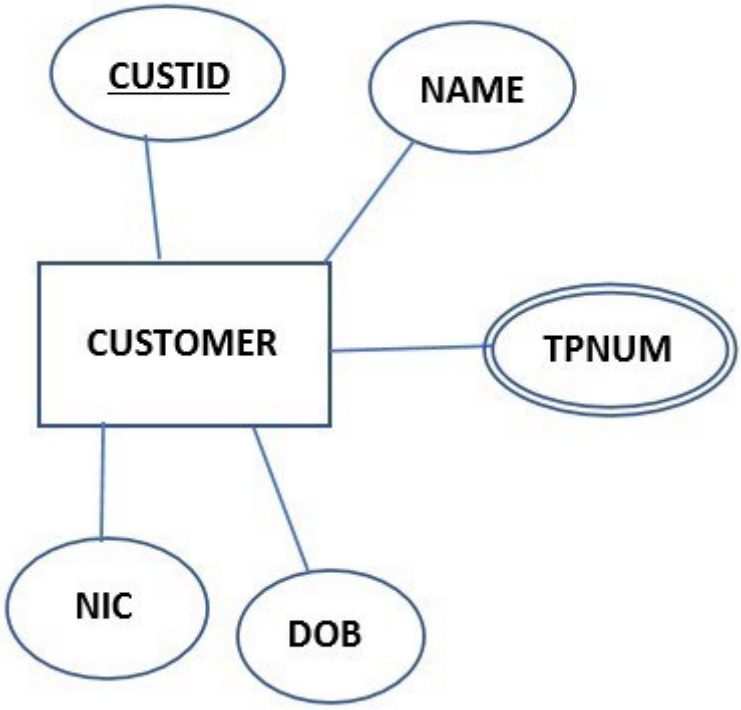


If it is many to many relations you should always make a new relationship with the name of the relationship between the entities.

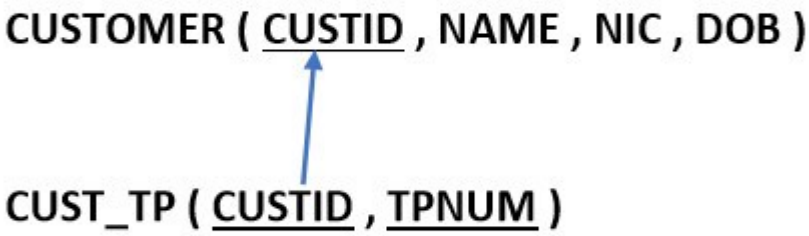
And there you should write both primary keys of the entities and attributes in the relationship and map them to the initials as shown below.



6. Map multivalued attributes.



If there are multivalued attributes you have to make a new relationship with a suitable name and write the primary key of the entity which belongs to the multivalued attribute and also the name of the multivalued attribute as shown below.

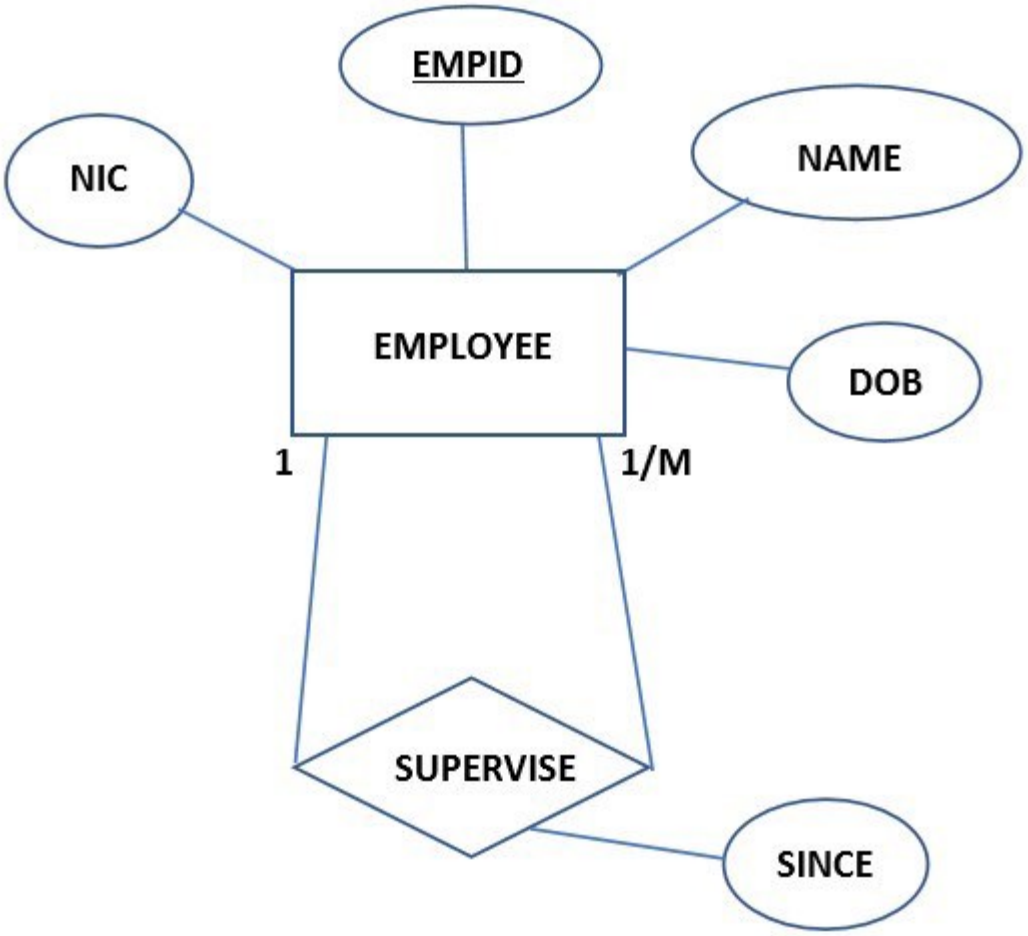


7. Map N-ary relations.

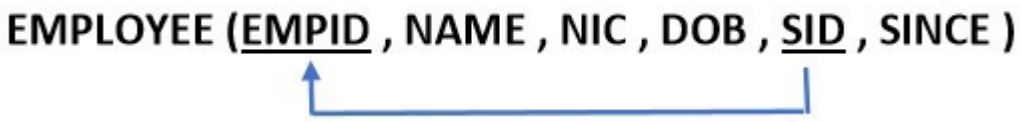
First, let us consider unary relationships.

We categorized them into two.

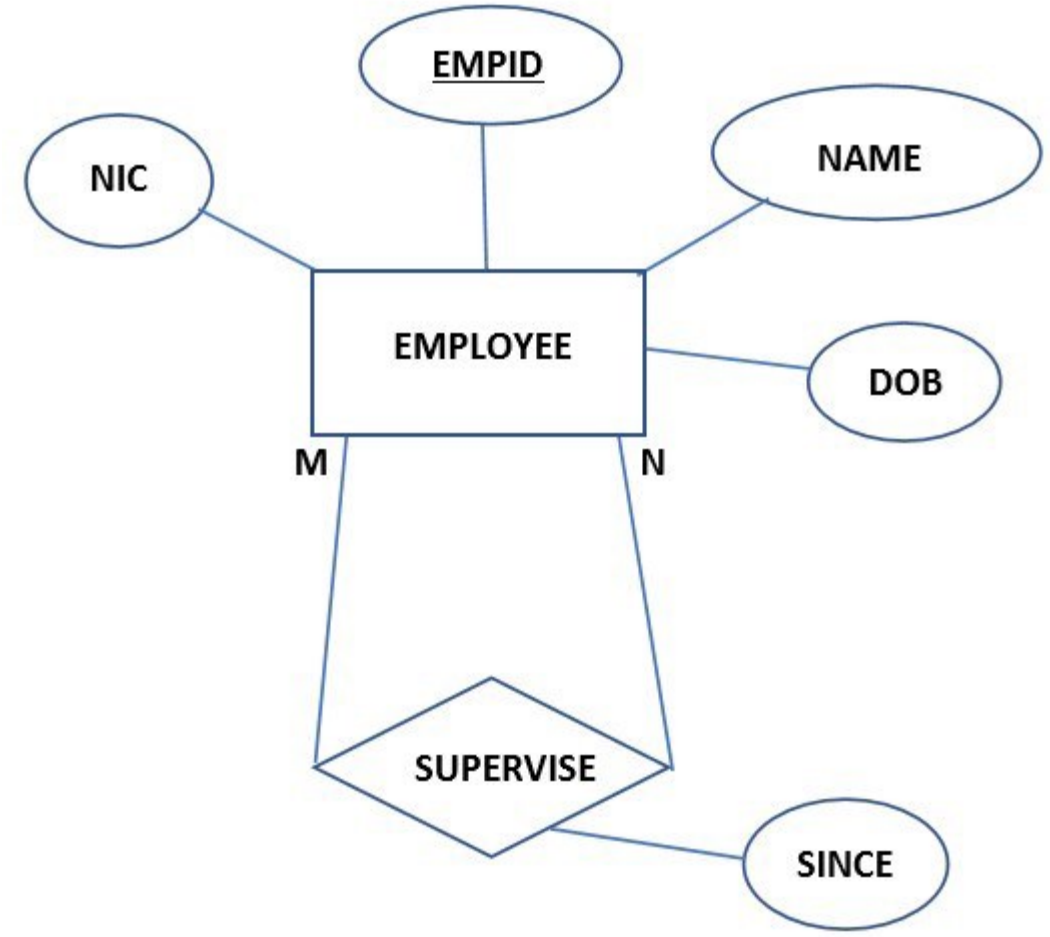
I. one-to-one and one to many relations.



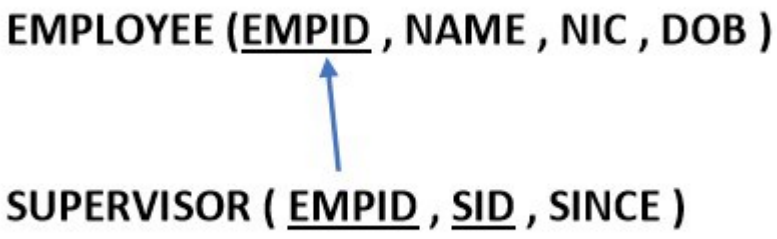
If it is unary and one to one or one to many relations you do not need to make a new relationship you just want to add a new primary key to the current entity as shown below and map it to the initial. For example, in the above diagram, the employee is supervised by the supervisor. Therefore we need to make a new primary key as **SID** and map it to **EMPID**. Because of **SID** also an **EMPID**.



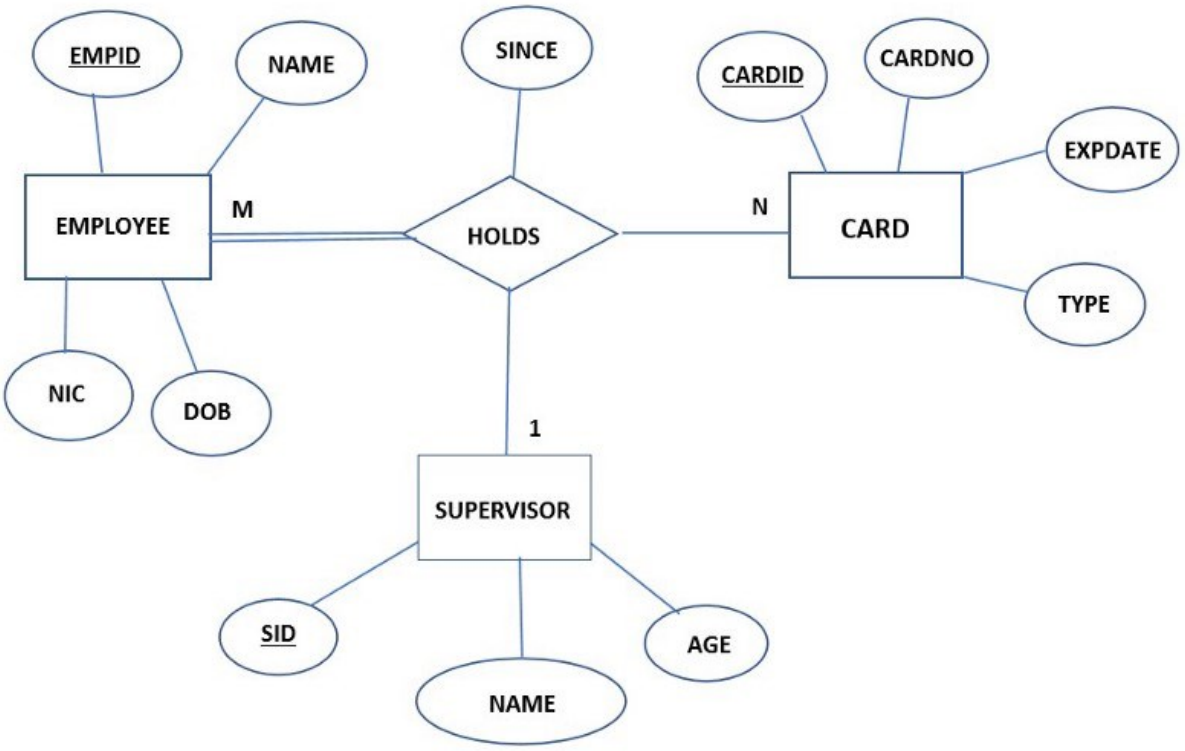
II. many-to-many relations.



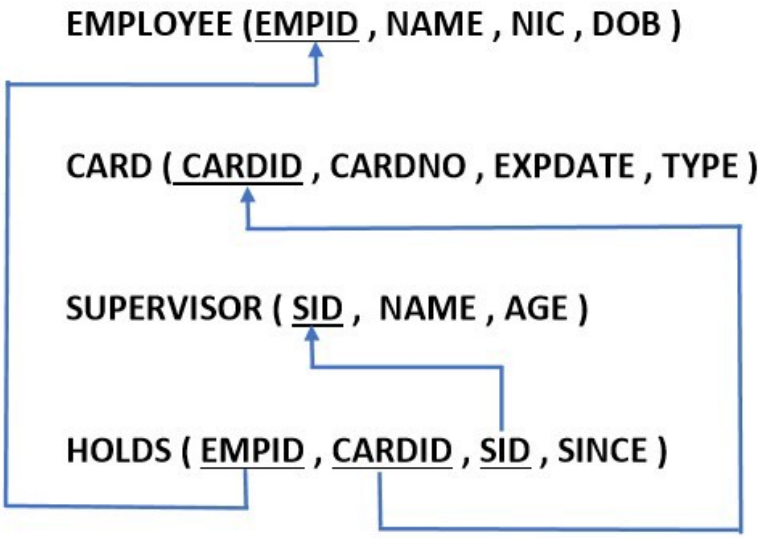
If it is unary and many to many relations you need to make a new relationship with a suitable name. Then you have to give it a proper primary key and it should map to where it comes as shown below.



Now let us see how to map relations with more than two entities.



If there are more than three entities for a relationship you have to make a new relation table and put all primary keys of connected entities and all attributes in the relationship. And in the end, you have to map them as shown below.



- # Relational Schema Design

Relation schema defines the design and structure of the relation like it consists of the relation name, set of attributes/field names/column names. every attribute would have an associated domain.

There is a student named Geeks, she is pursuing B.Tech, in the 4th year, and belongs to IT department (department no. 1) and has roll number 1601347 She is proctored by Mrs. S Mohanty. If we want to represent this using databases we would have to create a student table with name, sex, degree, year, department, department number, roll number and proctor (adviser) as the attributes.

```
student (rollNo, name, degree, year, sex, deptNo, advisor)
```

Note –
If we create a database, details of other students can also be recorded. Similarly, we have the IT Department, with department Id 1, having Mrs. Sujata Chakravarty as the head of department. And we can call the department on the number 0657 228662 .

This and other departments can be represented by the department table, having department ID, name, hod and phone as attributes.

```
department (deptId, name, hod, phone)
```

The course that a student has selected has a courseid, course name, credit and department number.

```
course (coursId, ename, credits, deptNo)
```

The professor would have an employee Id, name, sex, department no. and phone number.

```
professor (empId, name, sex, startYear, deptNo, phone)
```

We can have another table named enrollment, which has roll no, courseId, semester, year and grade as the attributes.

```
enrollment (rollNo, coursId, sem, year, grade)
```

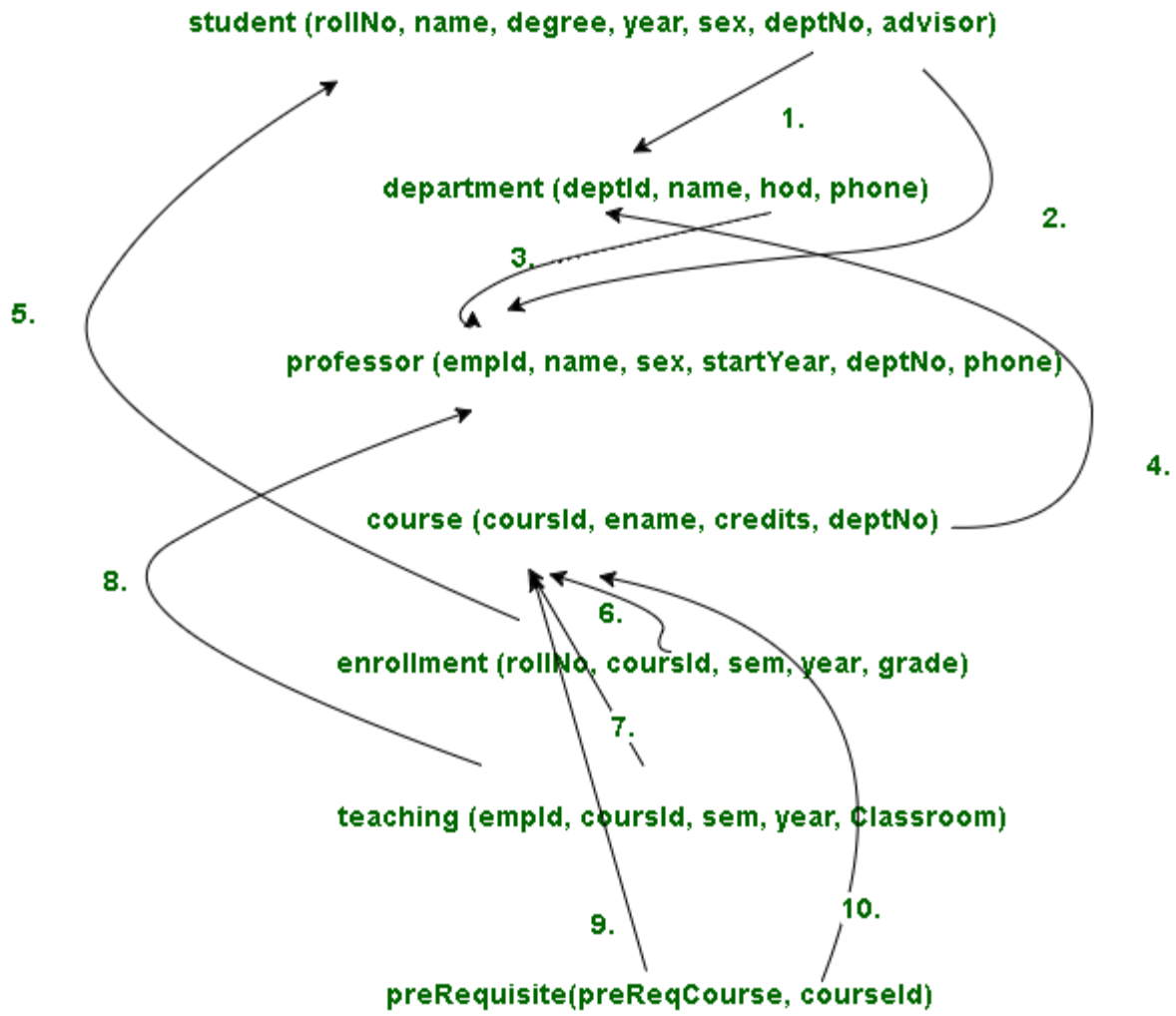
Teaching can be another table, having employee id, course id, semester, year and classroom as attributes.

```
teaching (empId, coursed, sem, year, Classroom)
```

When we start courses, there are some courses which another course that needs to be completed before starting the current course, so this can be represented by the Prerequisite table having prerequisite course and course id attributes.

`prerequisite (preReqCourse, courseId)`

The relations between them is represented through arrows in the following **Relation diagram**,



1. This represents that the deptNo in student table table is same as deptId used in department table. deptNo in student table is a foreign key. It refers to deptId in department table.

2. This represents that the advisor in student table is a foreign key. It refers to empId in professor table.

3. This represents that the hod in department table is a foreign key. It refers to empId in professor table.

4. This represents that the deptNo in course table table is same as deptId used in department table. deptNo in student table is a

foreign key. It refers to deptId in department table.

5. This represents that the rollNo in enrollment table is same as rollNo used in student table.

6. This represents that the courseId in enrollment table is same as courseId used in course table.

7. This represents that the courseId in teaching table is same as courseId used in course table.

8. This represents that the empId in teaching table is same as empId used in professor table.

9. This represents that preReqCourse in prerequisite table is a foreign key. It refers to courseId in course table.

10. This represents that the deptNo in student table is same as deptId used in department table.

Note –
startYear in professor table is same as year in student table

- # Introduction to Relational Algebra

  Relational algebra refers to a procedural query language that takes relation instances as input and returns relation instances as output. It performs queries with the help of operators. A binary or unary operator can be used. They take in relations as input and produce relations as output. Recursive relational algebra is applied to a relationship, and intermediate outcomes are also considered relations.

- # Relational Algebra – Operators

  Fundamental operations of Relational Algebra:

Unary Relational Operations

- SELECT (symbol: σ)
- PROJECT (symbol: π)
- RENAME (symbol: ρ)

Relational Algebra Operations From Set Theory

- UNION (υ)
- INTERSECTION ( ),

- DIFFERENCE (-)
- CARTESIAN PRODUCT (x)

Binary Relational Operations

- JOIN
- DIVISION

SELECT (σ)

The SELECT operation is used for selecting a subset of the tuples according to a given selection condition. Sigma(σ)Symbol denotes it. It is used as an expression to choose tuples which meet the selection condition. Select operator selects tuples that satisfy a given predicate.

$\sigma_p(r)$

σ is the predicate

r stands for relation which is the name of the table

p is prepositional logic

Example 1

$$\sigma_{\text{topic = "Database"}}(\text{Tutorials})$$

Output – Selects tuples from Tutorials where topic = 'Database'.

Example 2

$$\sigma_{\text{topic = "Database" and author = "guru99"}}(\text{Tutorials})$$

Output – Selects tuples from Tutorials where the topic is 'Database' and 'author' is guru99.

Example 3

$$\sigma_{\text{sales > 50000}}(\text{Customers})$$

Output – Selects tuples from Customers where sales is greater than 50000

Projection(π)

The projection eliminates all attributes of the input relation but those mentioned in the projection list. The projection method defines a relation that contains a vertical subset of Relation.

This helps to extract the values of specified attributes to eliminates duplicate values. (pi) symbol is used to choose attributes from a relation. This operator helps you to keep specific columns from a relation and discards the other columns.

Example of Projection:

Consider the following table

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive
4	Alibaba	Active

Here, the projection of CustomerName and status will give

$\Pi_{\text{CustomerName, Status}}(\text{Customers})$

CustomerName	Status
Google	Active
Amazon	Active
Apple	Inactive
Alibaba	Active

Rename (ρ)

Rename is a unary operation used for renaming attributes of a relation.

ρ (a/b)R will rename the attribute ‘b’ of relation by ‘a’.

Union operation (υ)

UNION is symbolized by ∪ symbol. It includes all tuples that are in tables A or in B. It also eliminates duplicate tuples. So, set A UNION set B would be expressed as:

The result <- A ∪ B

For a union operation to be valid, the following conditions must hold –

- R and S must be the same number of attributes.
- Attribute domains need to be compatible.
- Duplicate tuples should be automatically removed.

Example

Consider the following tables.

Table A		Table B	
column 1	**column 2**	**column 1**	**column 2**
1	1	1	1
1	2	1	3

A ∪ B gives

Table A ∪ B	
column 1	**column 2**
1	1
1	2
1	3

Set Difference (-)

– Symbol denotes it. The result of A – B, is a relation which includes all tuples that are in A but not in B.

- The attribute name of A has to match with the attribute name in B.
- The two-operand relations A and B should be either compatible or Union compatible.
- It should be defined relation consisting of the tuples that are in relation A, but not in B.

Example

`A-B`

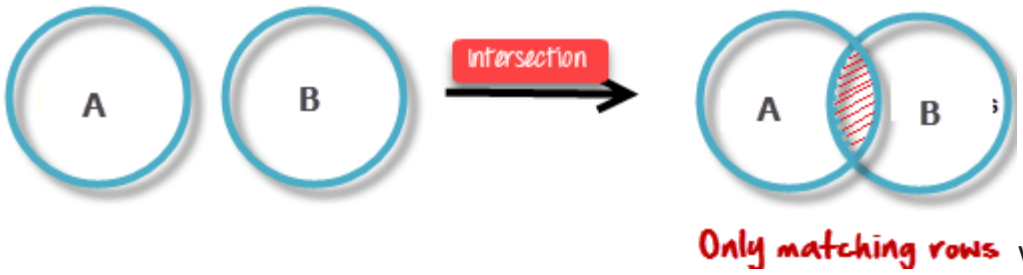
Table A – B	
column 1	**column 2**
1	2

Intersection

An intersection is defined by the symbol ∩

A ∩ B

Defines a relation consisting of a set of all tuple that are in both A and B. However, A and B must be union-compatible.



Visual Definition of Intersection

Example:

A ∩ B

Table A ∩ B	
column 1	**column 2**
1	1

Cartesian Product(X) in DBMS

Cartesian Product in DBMS is an operation used to merge columns from two relations. Generally, a cartesian product is never a meaningful operation when it performs alone. However, it becomes meaningful when it is followed by other operations. It is also called Cross Product or Cross Join.

Example – Cartesian product

$\sigma_{column\ 2 = '1'}(A \times B)$

Output – The above example shows all rows from relation A and B whose column 2 has value 1

σ column 2 = '1' (A X B)	
column 1	**column 2**
1	1
1	1

Join Operations

Join operation is essentially a cartesian product followed by a selection criterion.

Join operation denoted by ⋈.

JOIN operation also allows joining variously related tuples from different relations.

Types of JOIN:

Various forms of join operation are:

Inner Joins:

- Theta join
- EQUI join
- Natural join

Outer join:

- Left Outer Join
- Right Outer Join
- Full Outer Join

Inner Join:

In an inner join, only those tuples that satisfy the matching criteria are included, while the rest are excluded. Let's study various types of Inner Joins:

Theta Join:

The general case of JOIN operation is called a Theta join. It is denoted by symbol **θ**

Example

$A \bowtie_\theta B$

Theta join can use any conditions in the selection criteria.

For example:

$A \bowtie_{A.column\ 2 > B.column\ 2} (B)$

A ⋈ A.column 2 > B.column 2 (B)	
column 1	**column 2**
1	2

EQUI join:

When a theta join uses only equivalence condition, it becomes a equi join.

For example:

$A \bowtie_{A.column\ 2 = B.column\ 2} (B)$

A ⋈ A.column 2 = B.column 2 (B)	
column 1	**column 2**
1	1

EQUI join is the most difficult operations to implement efficiently using SQL in an RDBMS and one reason why RDBMS have essential performance problems.

NATURAL JOIN (⋈)

Natural join can only be performed if there is a common attribute (column) between the relations. The name and type of the attribute must be same.

Example

Consider the following two tables

C	
Num	**Square**
2	4
3	9

D	
Num	**Cube**
2	8
3	27

$C \bowtie D$

C ⋈ D		
Num	**Square**	**Cube**
2	4	8
3	9	27

OUTER JOIN

In an outer join, along with tuples that satisfy the matching criteria, we also include some or all tuples that do not match the criteria.

Left Outer Join(A ⟕ B)

In the left outer join, operation allows keeping all tuple in the left relation. However, if there is no matching tuple is found in right relation, then the attributes of right relation in the join result are filled with null values.

Consider the following 2 Tables

A	
Num	**Square**
2	4
3	9
4	16

B	
Num	**Cube**
2	8
3	18
5	75

A ⋈ B

A ⋈ B		
Num	**Square**	**Cube**
2	4	8
3	9	18
4	16	–

Right Outer Join: (A ⋈ B)

In the right outer join, operation allows keeping all tuple in the right relation. However, if there is no matching tuple is found in the left relation, then the attributes of the left relation in the join result are filled with null values.

A ⟕ B

A ⋈ B		
Num	**Cube**	**Square**
2	8	4
3	18	9
5	75	–

Full Outer Join: (A ⟗ B)

In a full outer join, all tuples from both relations are included in the result, irrespective of the matching condition.

A ⟗ B

A ⋈ B		
Num	**Cube**	**Square**
2	4	8
3	9	18
4	16	–
5	–	75

Summary

Operation(Symbols)	Purpose
Select(σ)	The SELECT operation is used for selecting a subset of the tuples according to a given selection condition
Projection(π)	The projection eliminates all attributes of the input relation but those mentioned in the projection list.

Union Operation(∪)	UNION is symbolized by symbol. It includes all tuples that are in tables A or in B.
Set Difference(-)	– Symbol denotes it. The result of A – B, is a relation which includes all tuples that are in A but not in B.
Intersection(∩)	Intersection defines a relation consisting of a set of all tuple that are in both A and B.
Cartesian Product(X)	Cartesian operation is helpful to merge columns from two relations.
Inner Join	Inner join, includes only those tuples that satisfy the matching criteria.
Theta Join(θ)	The general case of JOIN operation is called a Theta join. It is denoted by symbol θ.
EQUI Join	When a theta join uses only equivalence condition, it becomes a equi join.
Natural Join(⋈)	Natural join can only be performed if there is a common attribute (column) between the relations.
Outer Join	In an outer join, along with tuples that satisfy the matching criteria.
Left Outer Join(⟕)	In the left outer join, operation allows keeping all tuple in the left relation.
Right Outer join(⟖)	In the right outer join, operation allows keeping all tuple in the right relation.
Full Outer Join(⟗)	In a full outer join, all tuples from both relations are included in the result irrespective of the matching condition.