UNIT- II

Entity - Relationship Model- Conceptual Data Models for Database Design Entity Relationship Models, Concept of Entity, Entity sets, Relationship Sets, Attributes, Domains, Constraints, Keys Strong and Weak entities, Concepts of EER.

Relational data model Relations, Domains and attributes, Tuples, Keys, Integrity Rules, Relational algebra and Operations, Relational calculus and domain calculus, Relational Database Design Using ER to Relational mapping

ENTITY- RELATIONSHIP MODEL.

→ The E-R model defines the conceptual view of a database. It works around real-world entities and the associations among them.

→ An E-R model describes the structure of a database with the help of a diagram, which is known as Entity - Relationship diagram (E-R diagram).

Entity

→ An entity is a thing or object in the real world that is distinguishable from all other objects.

→ For example, In a school database, students, teachers, classes, and courses offered can be considered as entities.

→ An entity is represented as rectangle in an ER diagram.

Attributes.

→ Entities are represented by means of their properties, called attributes. All attributes have values. For example, a student entity may have name, class, and age as attributes.

→ An entity has a set of properties, and the values for some set of properties may uniquely identify an entity. For instance, a person may have a 'person-id' property whose value uniquely identifies that person.

Domain

→ For each attribute, there is a set of permitted values, called the <u>domain</u>, or <u>value set</u> of that attribute.

→ For example, a student's name cannot be a numeric value. It has to be alphabetic. A student's age cannot be negative, etc.

Entity Set

→ An entity set is a set of entities of the same type that share the same properties or attributes. So a set of one or more entities of student entity type is an Entity set.

→ For example, the set of all persons who are customers at a given bank can be defined as the entity set "customer". The ~~ets~~ customer entities share the same attributes such as customer_id, customer_name and customer_address, etc.

→ A database thus includes a collection of entity sets, each of which contains any number of entities of the same type.

→ The individual entities that constitute a set are said to be extensions of the entity set. Thus all the individual bank customers are the extensions of the entity set customer.

→ Entity sets do not need to be disjoint. For example, it is possible to define the entity set of all employees

of a bank (employee) and the entity set of all customers of the bank (customer). A person entity may be an employee entity, a customer entity, both, or neither.

Example :-

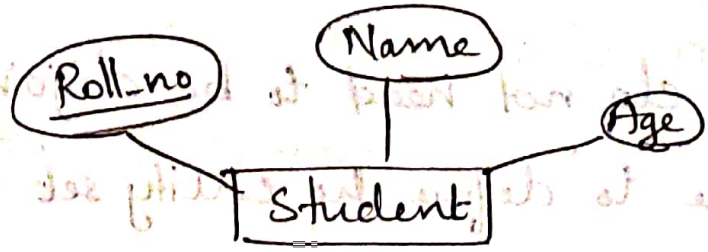
Consider the following Student table

Roll no	Name	Age
1	Akshay	20
2	Rahul	19
3	Pooja	20
4	Arathi	19

> student entity

~~The complete table is referred to as "student entity set" and each row represents an "entity".~~

The above table may be represented as ER diagram as



~~→ An entity set is represented in E R diagrams as a rectangular box enclosing the entity set name. Attribute names are enclosed in ovals and are attached to their entity set by straight line.~~

Entity type :-

→ It is collection of entity having common attribute. As in student table each row is an entity and have common attributes. so STUDENT is an entity type which contains entities having attributes Rollno, name and Age Also each entity type in a database is described by a name and a list of attribute. So we may say a table is an entity type.

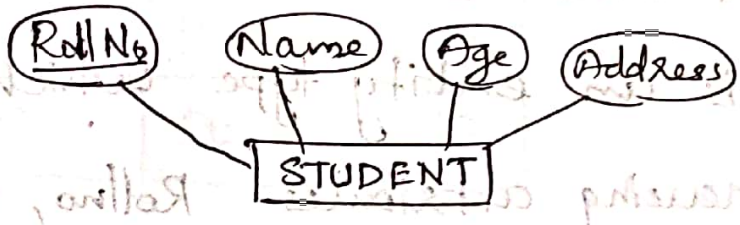
→ In the above example, '1, Akshay, 20' is an entity of entity type Student.

→ Set of one or more entities of student entity type is an Entity set.

Types of attributes:-

1) Key attribute:-

→ A key attribute can uniquely identify an entity from an entity set. For example Student rollnumber can uniquely identify a student from a set of students. Key attribute is represented by oval same as other attributes however the text of key attribute is underlined.



→ A key is a minimal set of attributes whose values uniquely identify an entity in an entity set.

→ For example, the customer_id attribute of the entity set customer is sufficient to distinguish one customer entity from another. Thus 'customer_id' is a key, But the customer_name attribute of customer is not a key, because several people

might have the same name.

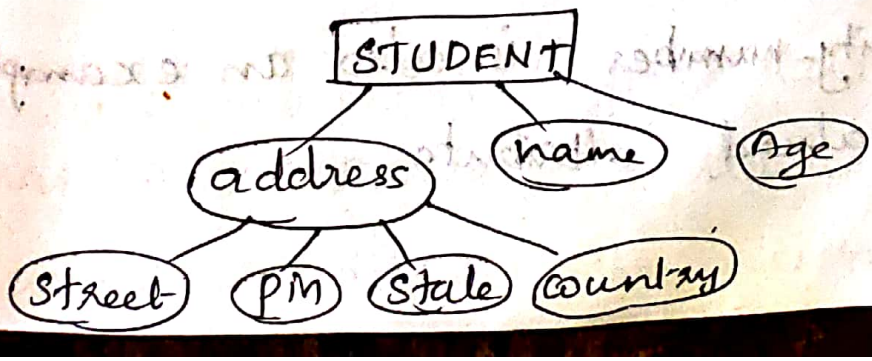
2). Simple attribute:-

→ simple attributes are atomic values, which cannot be divided further. For example, a student's phone number is an atomic value of 10 digits.

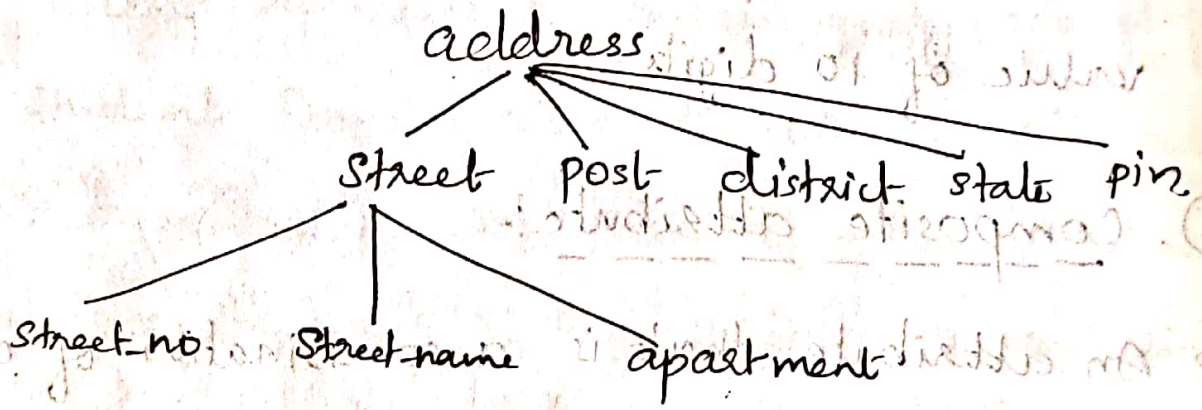
3). Composite attribute:-

→ An attribute that is a combination of other attributes is known as composite attribute. Composite attributes can be divided into smaller subparts.

→ For example, In student entity, the student address is a composite attribute as an address is composed of other attributes such as street, pincode, state, country, etc..



→ composite attribute can form a hierarchy. For example, in the composite attribute address, its component attribute 'street' can be further divided into street_number, street_name and apartment_number.



4). Single-valued attribute:-

→ Most attributes have a single value for a particular entity. such attributes are called single-valued attributes.

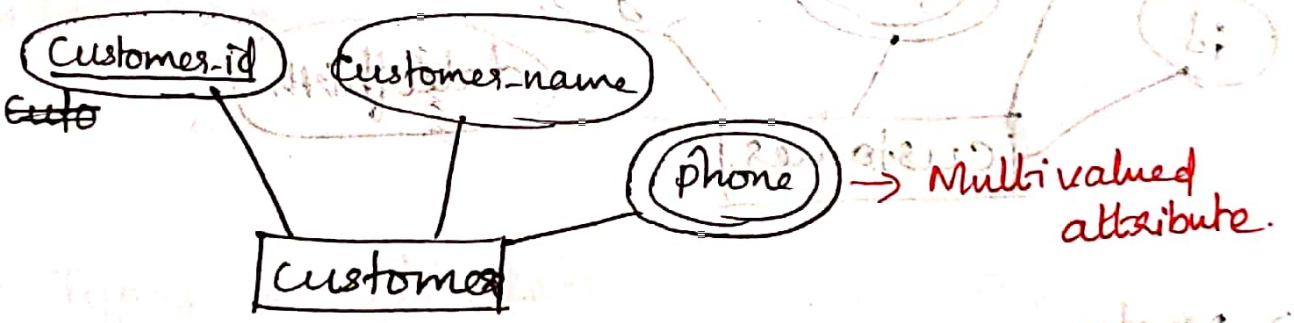
→ For example, age is a single valued attribute of a person.

→ social-security-number is also an example for single valued attribute.

5). Multi-valued attribute:-

→ Multivalued attributes may contain more than one values. For example, a person can have more than one phone numbers, email addresses, etc.

→ It is represented with double ovals in an ER diagram.



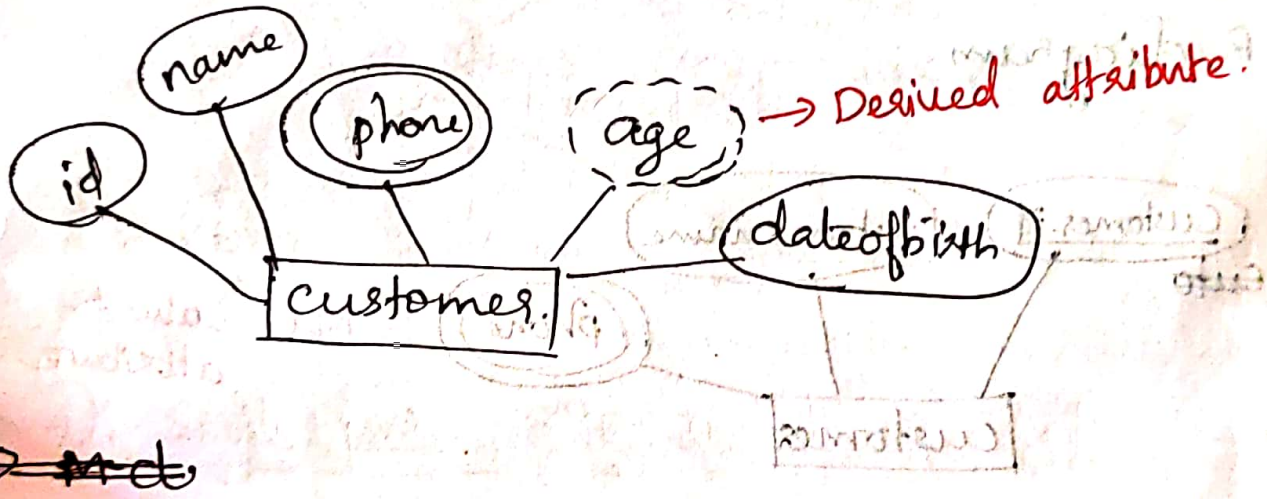
6). Stored and Derived Attributes:-

→ Derived attributes are the attributes whose values can be derived from the values of the other related attributes or entities.

→ For example, suppose that the customer entity set has an attribute age, which indicates the customer's age. If the customer entity

set also has an attribute date-of-birth, we can calculate the age from the date-of-birth and the current date. Thus age is a derived attribute.

→ In this case the date-of-birth attribute is stored or base attribute. The value of a derived attribute is not stored, but is computed when required.



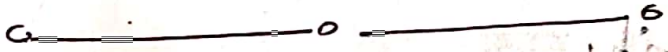
→ Derived attribute is represented by clashed oval in an ER diagram.

Null Values.

→ In some cases a particular entity may not have an applicable value for an attribute. For example, the 'apartment number' attribute of an address applies only to the addresses

that are in apartment buildings and not to other types of residences. Similarly, a college degree attribute ~~only~~ applies only to persons with college degrees.

→ For such situations, a special value called NULL is created. The null value may indicate 'not applicable' - that is, the value does not exist for the entity. For example, one may have no middle name



Types of Entity sets.

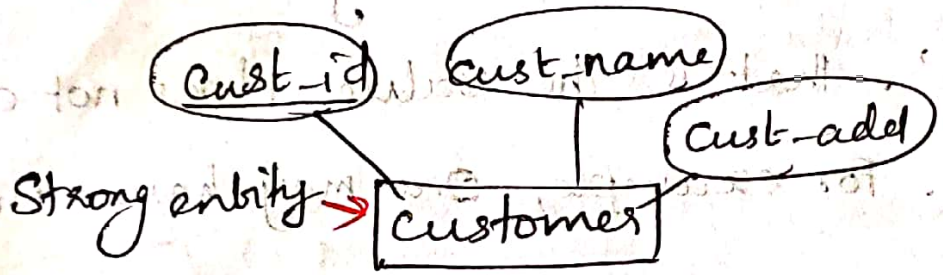
1) Strong entity.
2) Weak entity.

Strong entity:-

→ The strong entity is the one whose existence does not depend on the existence of any other entity in a schema.

→ It is denoted by a single rectangle.

→ A strong entity always has the primary key in the set of attributes that describes the strong entity. It indicates that each entity in a strong entity set can be uniquely identified.



Weak entity

→ An entity set may not have sufficient attributes to form a primary key - such an entity is termed as a weak entity set.

→ A weak entity is denoted by double rectangle

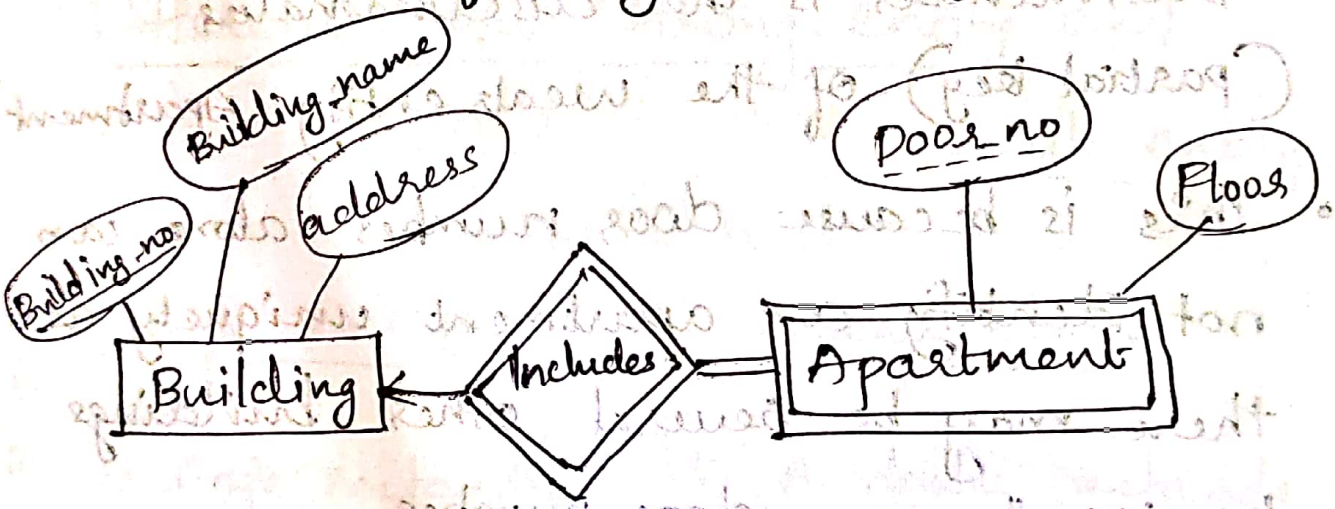
→ Weak entity do not have the primary key instead it has a partial key that uniquely discriminates the weak entities. The primary key of a weak entity is a composite key formed from the primary key of the strong

entity and partial key of the weak entity

→ The relationship between a weak entity and a strong entity is always denoted with identifying relationship. ie, double diamond

Example:-

Consider the following E-R diagram



→ A double diamond symbol is used for representing the relationship that exists between the strong and weak entity sets and this relationship is known as identifying relationship

→ A double line is used for representing the connection of the weak entity with the identifying relationship.

→ In the above ER Diagram

- One strong entity "Building" and one weak entity "Apartment" are related to each other
- Strong entity "Building" has building number as its primary key
- Door number is the discriminator (partial key) of the weak entity "Apartment"
- This is because door number alone can not identify an apartment uniquely as there may be several other buildings having the same door number.
- It suggests that there might exist some buildings which has no apartment.

→ To uniquely identify any apartment,

- First, building number is required to identify the particular building.
- Secondly, door number of the apartment is required to uniquely identify the apartment.

Thus,
Primary key of Apartment

= Primary key of Building + its own discriminator.

= Building number + Door number.

Difference b/w strong entity and weak entity

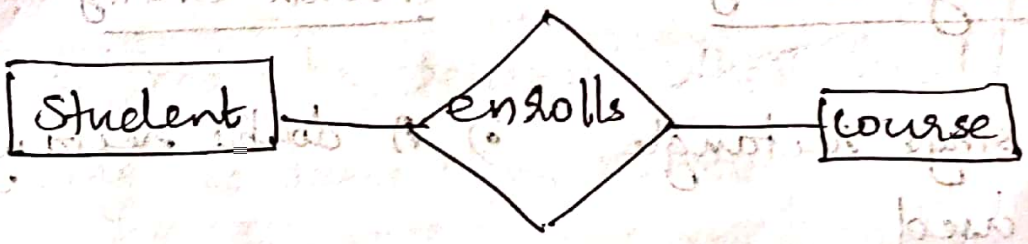
Strong entity	Weak entity
A single rectangle is used	A double rectangle is used
It contains sufficient attributes to form its primary key	It does not contain
A diamond symbol is used for relationship.	A double diamond symbol is used
A single line is used for connection	A double line is used for connection.

Relationship and Relationship sets.

Relationship

→ The association among entities is called a relationship. For example, an employee works_at a department, a student enrolls in a course. Here 'works_at' and 'Enrolls' are called relationship.

eg:-



Relationship set

→ A set of relationships of similar type is called a relationship set. Like entities, a relationship too can have attributes. These attributes are called descriptive attribute.

eg:-

STUDENT

Roll	Name	Age
1	A	20
2	B	21
3	C	20
4	D	22

BOOK

Book No	Author	Subject
B_1	E	Physics
B_2	F	Maths
B_3	G	Chemistry
B_4	H	Biology
B_5	I	History
B_6	J	Computer

→ Here the relationship is 'issue'.

issue (Roll, BookNo)

issue

Roll	Book No
1	B_2
2	B_6
3	B_4
4	B_1

Here '1, B_2' is a relationship, '2, B_6' is another relationship, and so on.

→ All the relationships are alike in nature. So they form a relationship set.

→ Here we have to keep the 'date of issue'.

→ Set of attributes in relationship - R will be

Primary key (E_1) ∪ primary key (E_2) ∪ . . . .

Primary key (E_n)

- If the relationship ~~have~~ has the descriptive attributes associated with it and if these attributes are $\{a_1, a_2, \ldots a_n\}$ then the relationship will be

Primary key (E_1) ∪ . . . Primary key (E_n) ∪

$\{a_1, a_2 \ldots a_n\}$.

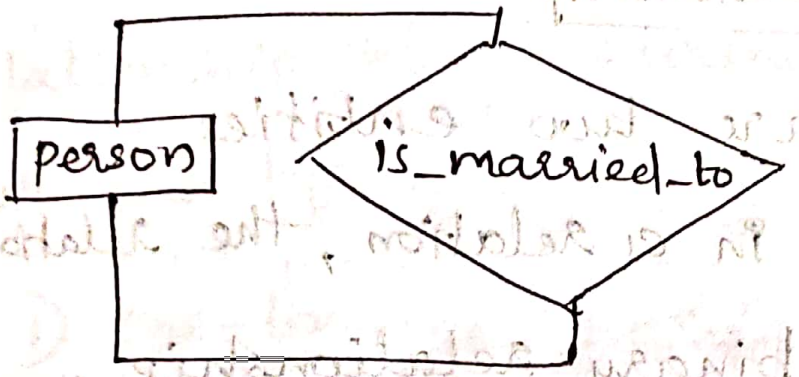
So here issue (Roll, Book No, access_date).

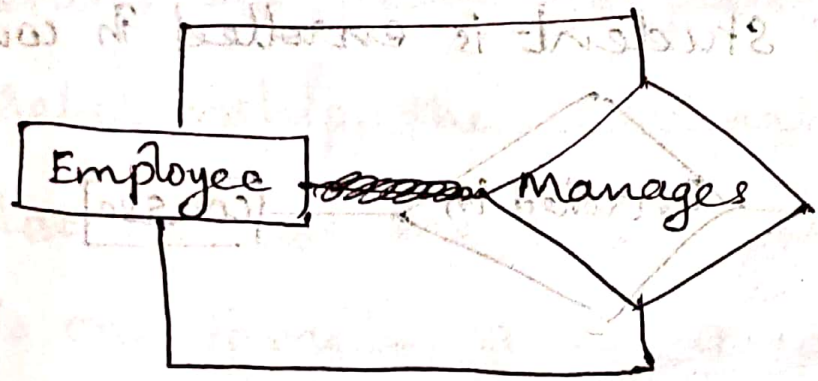
Degree of relationship.

→ The number of participating entities in a relationship, defines the degree of the relationship.

1) Unary relationships

→ A unary relationship exists when an association is maintained within a single entity.





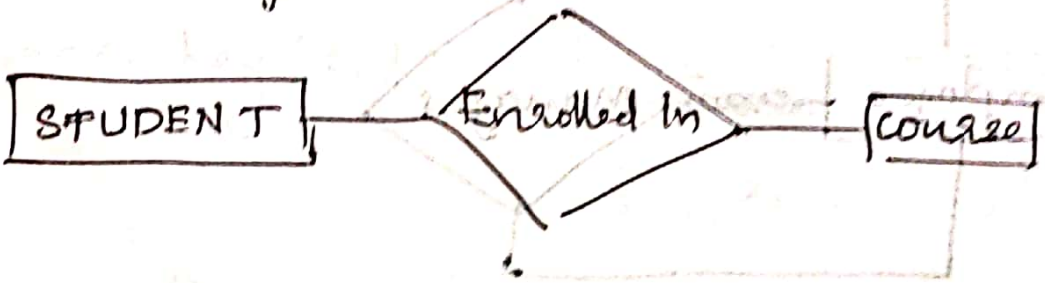
In the case of the Unary relationship shown in the figure an employee within the EMPLOYEE entity is the manages for one or more employees within that entity. In this case, the existence of "manages" relationship means that EMPLOYEE requires another

EMPLOYEE to be the manager - that is EMPLOYEE has a relationship with itself. Such a relationship is known as a recursive relationship.

2). Binary relationship

→ When there are two entities participating in a relation, the relationship is called as binary relationship.

For example, Student is enrolled in course



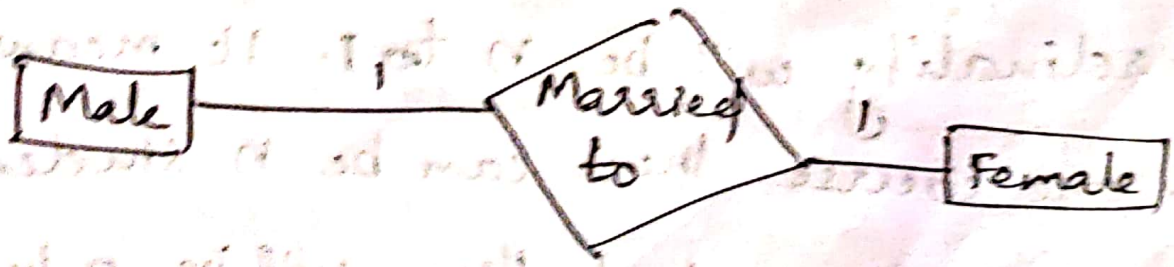
3). n-ary relationship

→ When there are 'n' entities set participating in a relation, the relationship is called as n-ary relationship.

Mapping Cardinalities.

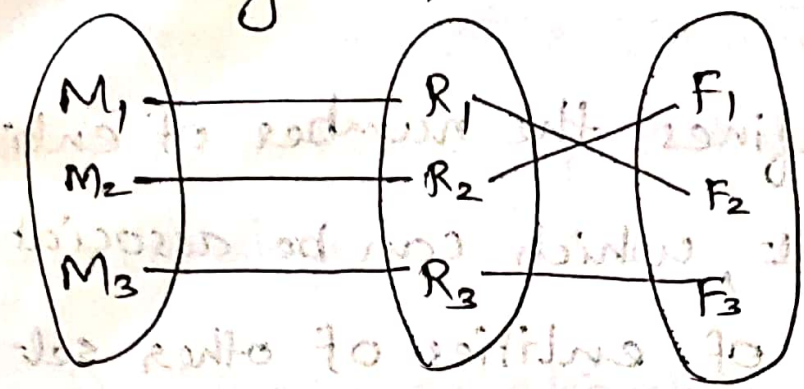
→ Cardinality defines the number of entities in one entity set, which can be associated with the number of entities of other set via relationship set. Cardinality can be of different types:-

1) One-to-one :- When each entity in each entity set can take part only once in the relationship, the cardinality is one-to-one. Let us assume that a male can marry to one female and a female can marry to one male. So the relationship will be one-to-one.

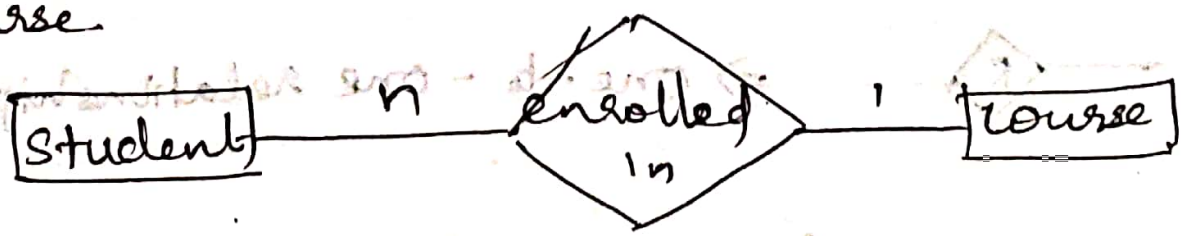


⇒ One-to-one relationship

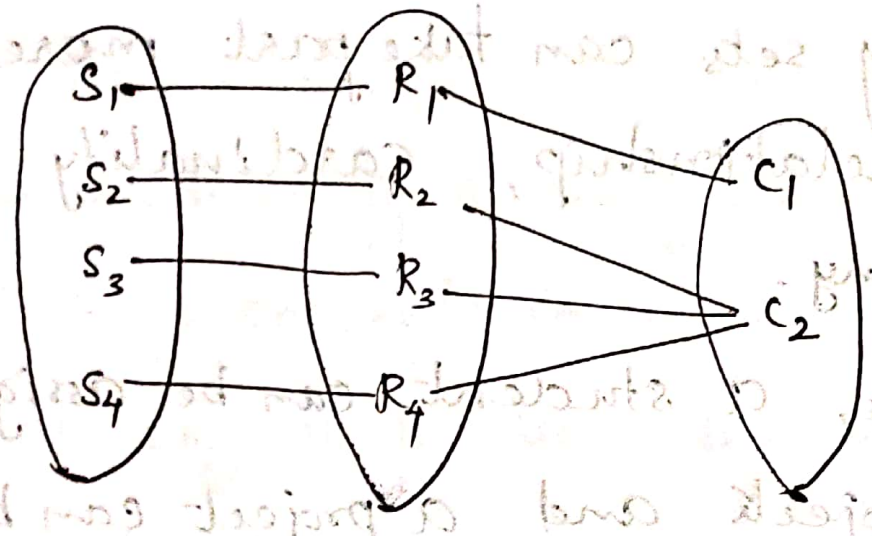
Using sets, it can be represented as,



2). Many-to-one :- When entities in one entity set can take part only once in the relationship set and entities in other entity set can take part more than once in the relationship set, cardinality is many to one. Let us assume that a student can take only one course but one course can be taken by many students. So the cardinality will be n to 1. It means that for one course there can be n students but for one student, there will be only one course.

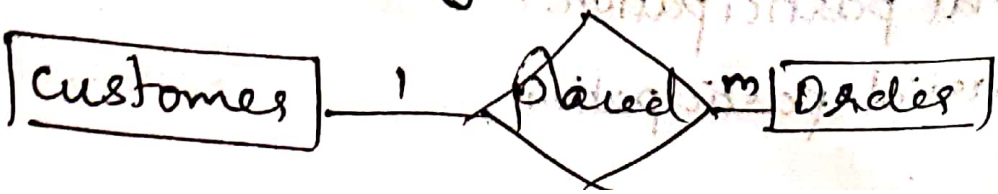


Using sets, it can be represented as,



In this case, each student is taking only one course but 1 course has been taken by many students.

3. One-to-Many :- When a single instance of one entity is associated with more than one instances of another entity, then it is called one-to-many relationship. For example, a customer can place many orders, but a order cannot be placed by many customers.



4) Many-to-many :- when entities in all entity sets can take part more than once in the relationship, cardinality is many to many.

For example, a student can be assigned to many projects and a project can be assigned to many students.

Participation Constraints

→ The participation constraint specify whether the existence of an entity depends on its being related to another entity via the relationship set.

→ How an entity participate in a relationship.

→ Two types of participation constraints:-

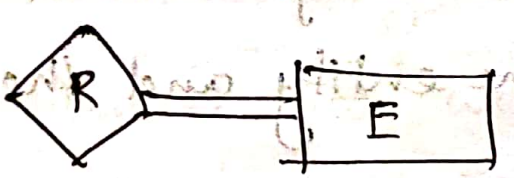
1) Total participation.
2) Partial participation.

Total Participation.

→ A total participation of an entity set represents that each entity in entity set must have at least one relationship in a relationship set.

→ Thats why it is also called as mandatory participation.

→ Total participation is represented using a double line b/w the entity set and the relationship



→ Each entity in the entity set must participate in the relationship.

eg:-

- It specifies that each student must be enrolled in atleast one course.

- Double line between the entity set 'student' and relationship 'enrolled in' signifies total participation.

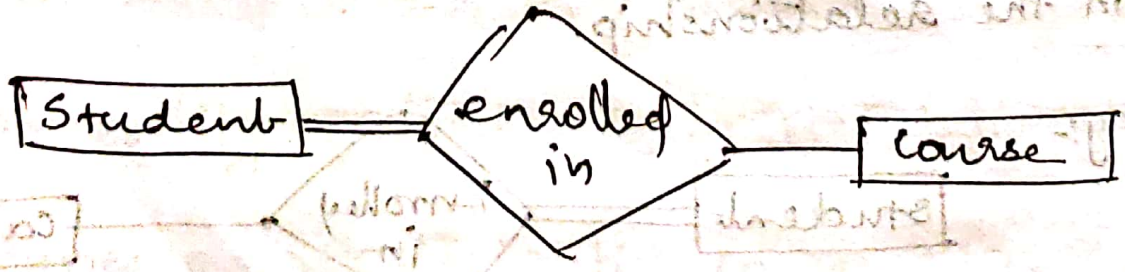
Partial participation

→ Not all entities are involved in the relationship.

→ The entity may or may not participate in the relationship. That's why it is also called as Optional participation.

→ Partial participation is represented using a single line b/w the entity and the relationship.

eg:-



- single line b/w entity 'course' and relationship 'enrolled in' signifies partial participation.

- It specifies that there might exist some courses for which no enrollments are made

---

THE ENHANCED E-R MODEL
(EXTENDED E-R MODEL)

→ As the complexity of data increased in the late 1980's, it became more and more difficult to use the traditional ER model for database modelling. Hence some improvements or enhancements were made to the existing ER model to make it able to handle the complex applications better.

→ Extended E-R features :-

1) Generalization :-

- Generalization is the process of generalizing the entities which contain the properties of all the generalized entities.

- It is a bottom up approach
- Generalization is the reverse process of Specialization.
- It defines a general entity type from a set of specialized entity type.

For example:-

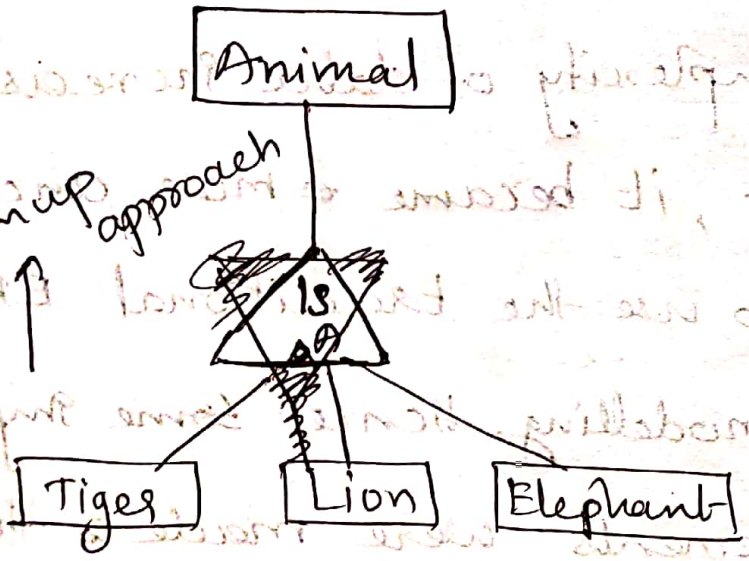
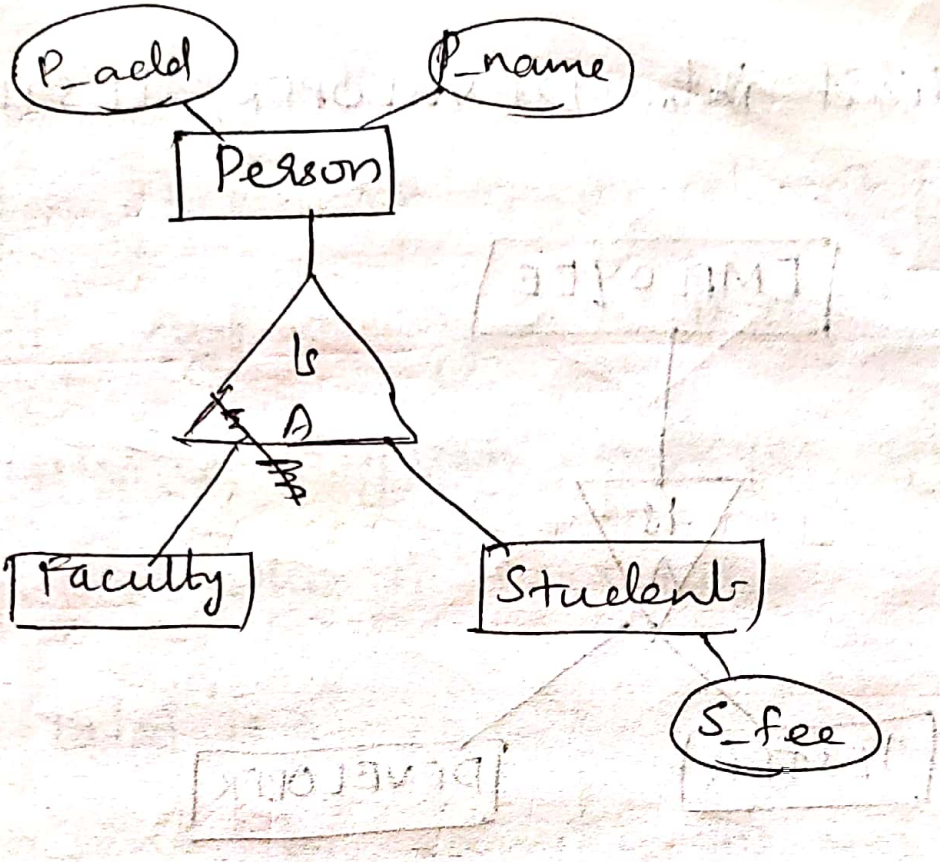


fig:- Generalization.

In the above example Tiger, Lion, Elephant can all be generalized as Animals.

→ Generalization is a bottom up approach in which two or more entities can be generalized to a higher level entity.



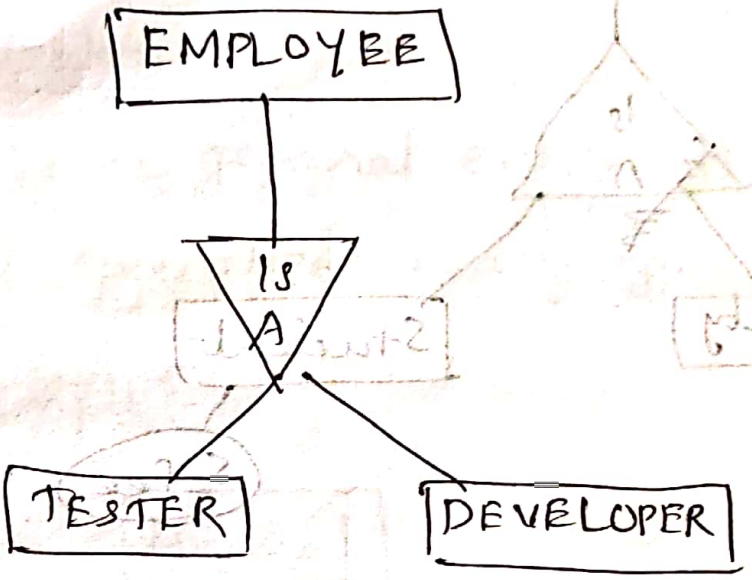
→ For example, student and faculty can be generalized to a higher level entity called person as shown in figure

2). Specialization :-

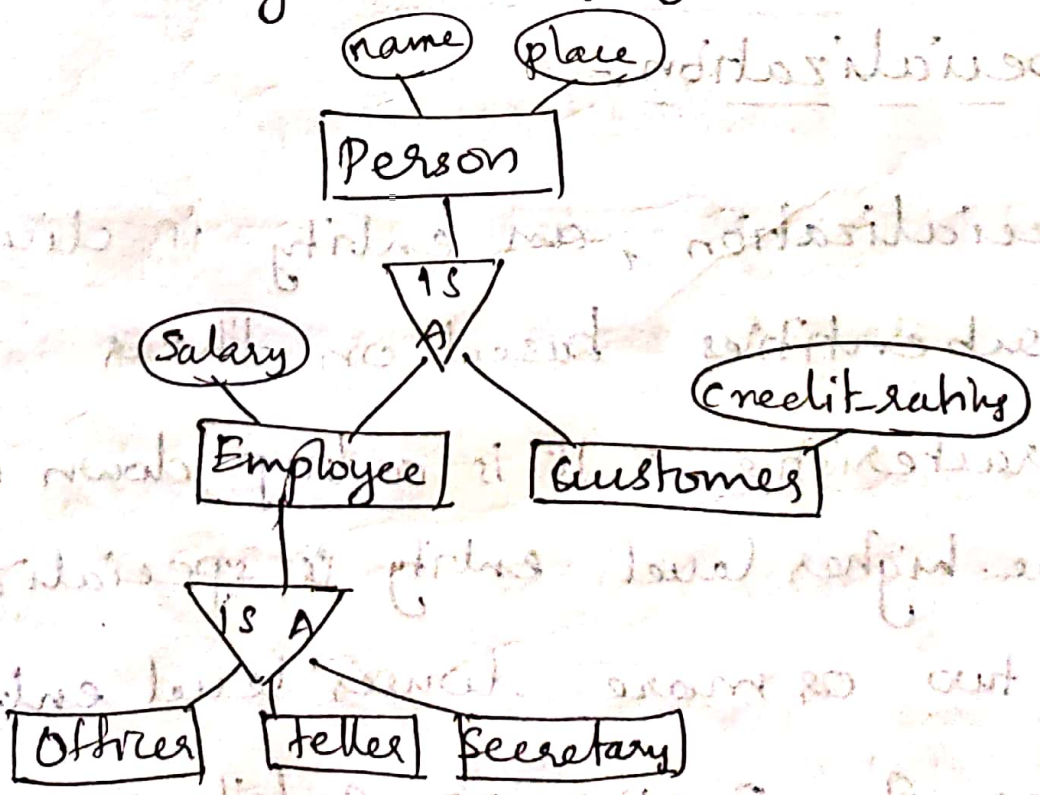
→ In specialization, an entity is divided into sub entities based on their characteristics. It is a top down approach where higher level entity is specialized into two or more lower level entities.

→ For example, EMPLOYEE entity can be

Specialized into DEVELOPER, TESTER, etc



→ The specialization of person allows us to distinguish among persons according to whether they are employee or customers.



→ When more than one specialization is formed on an entity set, a particular entity may belong to multiple specializations. For instance, a given employee may be a temporary employee who is secretary.

→ The label 'is A' stands for 'is a' and represents, for example, the ~~cust~~ customer is a person. The is A ~~rep~~ relationship may also be referred to as a superclass-subclass relationship.

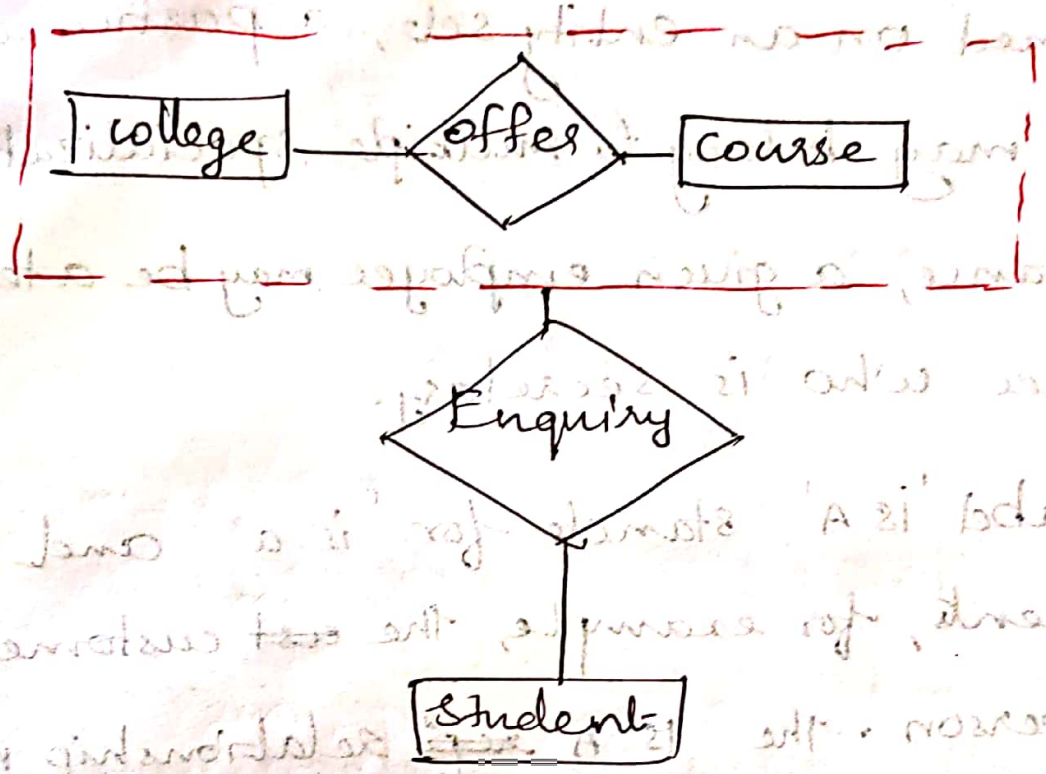
3). Aggregation :-

→ Aggregation is a process that represent a relationship between a whole object and its component parts.

→ It abstracts a relationship between objects and viewing the relationship as an object.

→ It is a process when two entity is treated

as a single entity

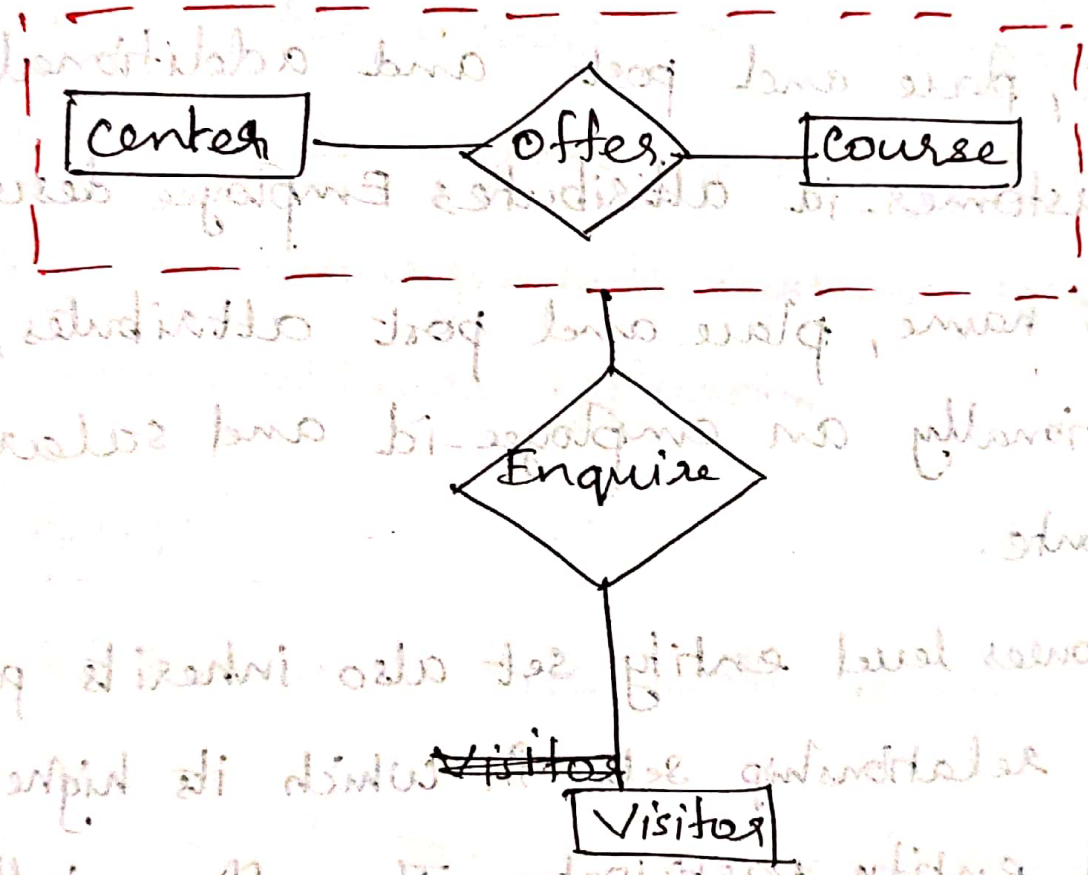


→ In the above example, the relation between college and course is acting as an entity in relation with student.

→ Aggregation is a process when relation between two entities is treated as a single entity.

→ In the diagram below, the relationship b/w center and course together, is acting as an Entity, which is in relationship with another entity Visitor. Now in real world

If a visitor or a student visits a coaching center, he/she will never enquire about the center only or just about the course, rather he/she will ask ~~eq~~ enquire about both.



4). Attribute Inheritance :-

→ A crucial property of the higher and lower level entities created by specialization and generalization is attribute inheritance

→ The attributes of higher level entity sets

are said to be inherited by the lower level entity sets.

→ For example, Customer and Employee inherit the attribute of person. Thus customer is described by its attributes name, place and post and additionally a customer_id attributes Employee described by its name, place and post attributes, and additionally an employee_id and salary attribute.

→ A lower level entity set also inherits participation in the relationship sets in which its higher level entity participates. The officer, teller, and secretary entity sets can participate in the works_for relationship sets since the higher level Employee participates in the works_for relationship.

RELATIONAL DATA MODEL:

→ The relational model for database management is a database model based on predicate logic and set theory

→ It was formulated and proposed in 1969 by Edgar Codd as a way to make database management systems more independent of any particular application.

Relation.

→ A relation is a table with columns and rows.

→ In relational data model, relations are saved in the format of tables. This format stores the relation among entities.

→ A table has rows and columns, where rows represents records and columns represent the attributes.

eg:-

STUDENT

Roll no	Name	Phone
1	Alex	2491255
2	John	421410
3	Arjan	241426
4	Kevin	421530

- Attribute :- An attribute is a named column of a relation. • Attributes are the properties that define a relation

  eg :- Rollno, Name, phone

- Relation schema :- A relation schema represents name of the relation with its attributes.

  eg :- STUDENT (Rollno, Name, phone).

- Tuple :- Each row in the relation is known as tuple. The above relation contains 4 tuples.

•) Relation Instance :- The set of tuples of a relation at a particular instance of time is called as relation instance.

The above table shows the relation instance of STUDENT at a particular time. It can change whenever there is insertion, deletion or updation in the database.

•) Degree :- The number of attributes in the relation is known as degree of the relation. The STUDENT relation defined above has degree 3.

•) Cardinality :- The number of tuples in a relation is known as cardinality. The STUDENT relation defined above has cardinality 4.

•) Domain :- A domain is the set of allowable values for one or more attributes.

⇒ For each attribute, there is a set of permitted values, called domain of the attribute.

INTEGRITY CONSTRAINTS

⇒ Relational Integrity constraints is referred to conditions which must be present for a valid relation.

~~⇒ An integrity constraint~~ ~~constraint~~

→ Integrity constraints are used to ensure accuracy and consistency of data in a relational database.

- Constraints on databases can ~~be~~ generally be divided into three main categories.

  1) Inherent - Model based constraints :-
  2) Schema based constraints
  3) Application based constraints.

- Inherent Model based constraints :- These are the constraints that are inherent in the data model. For example, in relational model, no two tuples in a relation can be duplicate.

1) schema based constraints :- can be expressed using DDL (Data Definition Language); this kind is the focus of this section.

2) Application based constraints :- These are the constraints that cannot be directly expressed in the schemas of the data model, and hence must be expressed and enforced by the application programs.

*). Elaborating upon schema based constraints :-

- Domain constraints
- Key constraints
- Entity Integrity constraints
- Referential integrity constraints.

- Domain constraints :-

→ Domain constraints are the most elementary form of integrity constraint.

→ It specifies that the value taken by the attribute must be the atomic value from its domain.

→ Attributes have specific values in real world scenario. For example, age can only be a positive integer. The same constraints have been tried to employ on the attributes of a relation. Every attribute is bound to have a specific range of values. For example, age cannot be less than zero and telephone numbers cannot contain a digit outside 0-9.

→ Domain constraints are user defined datatype and we can define them like this.

Domain constraint = datatype + constraints (NOT NULL / UNIQUE / PRIMARY KEY / FOREIGN KEY / CHECK / DEFAULT).

→ Example:-

I want to create a table "student_info" with 'stu_id' field having value greater than 100.

create domain id_value int
constraint id_test
check (value > 100);

create table student_info (
stud_id id_value PRIMARY KEY,
stu_name varchar(30), stu_age int);

2) Key constraints :-

→ Every relation in the database should have at least one set of attributes which defines a tuple uniquely. Those set of attributes is called key. eg :- ROLL_NO in STUDENT is a key. No two students can have same roll number.

→ Key constraints force that, in a relation with a key attribute, no two tuples can have identical values for key attributes.

Super key :- The set of attributes which can

uniquely identify a tuple is known as super key. For example,

STUDENT (Stud_no, Stud_name, age, phone)

Stud_no
(Stud_no, Stud_name)
(Stud_no, age)
:
phone.

} super keys

Candidate key :- Candidate keys are defined as the minimal set of fields, which can uniquely identify each record in a table. There can be more than one Candidate key.

→ In the above example 'Stud_no' and 'phone' both are candidate keys for relation STUDENT

→ A candidate key can never be NULL or empty. And its value should be unique.

→ A candidate key can be a combination of more than one columns (attributes)

Primary key :- The primary key is selected

from one of the candidate keys, and becomes the identifying key of a table. It can uniquely identify any data row of the table.

→ Primary key is a candidate key that is most appropriate to become the main key for any table.

→ For the table STUDENT we can make 'stud_id' column as the primary key.

Composite key :- Key that consists of two or more attributes that uniquely identify any record in a table is called composite key.

composite key.

Stude_id	Subject_id	marks	exam_name

In the above picture, we have a score table which stores the marks scored by a student in a particular subject.

In this table, 'Stud_id' and 'subject_id' together will form the primary key, hence it is a composite key.

• Entity Integrity constraints:-

It states that no primary key value can be null. This is because the primary key value is used to identify individual tuples in a relation; having null values in the primary key implies that we cannot identify some tuples.

• Referential Integrity constraints:-

→ Referential Integrity constraint works on the concept of Foreign keys.

Foreign key:- Foreign keys are the column of the table which is used to point to the primary key of another table.

→ They act as a cross reference between tables.

eg:-

EMPLOYEE

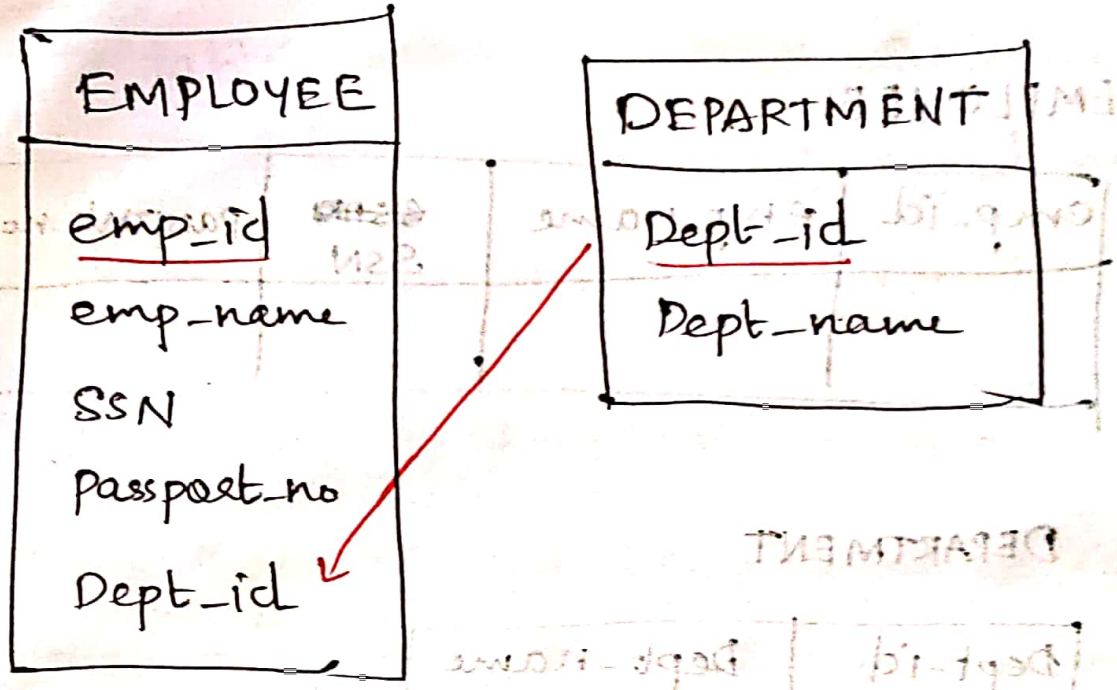
emp_id	emp_name	~~SSN~~ SSN	passport_no

DEPARTMENT

Dept-id	Dept_name

In a company, every employee works in a specific department, and employee and department are two different entities. So we can't store the information of the department in the employee table. That's why we link these two tables through the primary key of one table.

- We add the primary key of the department table, 'Dept-id' as a new attribute in the EMPLOYEE table.
- Now in the EMPLOYEE table, Dept-id is the foreign key and both the tables are related.



→ Referential Integrity constraint states that if a relation refers to a key attribute of a different or same relation, then that key element must exists

→ The relation which is referencing to other relation is called REFERENCING RELATION (EMPLOYEE in this case) and the relation to which other relations refers is called REFERENCED RELATION (DEPARTMENT in this case).

→ 'Dept_id' of Employee can only take

the values which are present in 'Dept_id' of Department relation, which is called referential integrity constraint.

→ The following two important results emerges out due to referential integrity constraint.

- We can not insert a record into a referencing relation if the corresponding record does not exist in the referenced relation
- We cannot delete or update a record of the referenced relation if the corresponding record exists in the referencing relation.

eg:-

Consider the following two relations - 'student' and 'department'. Here relation 'student' references the relation 'Department'.

Student (Stu_id, Name, Dept_no)

Referencing

Foreign key.

Department (Dept_no, Dept_name)

Referenced

STUDENT

Stu_id	Name	Dept_no
S001	Akshay	D10
S002	Abhishek	D10
S003	Shashank	D11
S004	Rahul	D14

DEPARTMENT

Dept_no	Dept_name
D10	A
D11	B
D12	C
D13	D

Here,

- The relation 'student' does not satisfy the referential integrity constraint.
- This is because in relation 'Department' no values of primary key specifies dept_no

- Thus referential Integrity constraint is violated.

→ To ensure the correctness of the database, it is important to handle the violation of referential Integrity constraints properly.

RELATIONAL ALGEBRA AND OPERATIONS.

→ The relational algebra is a procedural query language. It consists of a set of operations that take one or two relations as input and produces a new relation as the result.

→ Relational algebra is a collection of operations on relations.

Fundamental Operations:-

- SELECT
- PROJECT
- UNION
- SET DIFFERENCE

- ~~cost~~ CARTESIAN PRODUCT
- RENAME.

→ The select, project and Rename operations are called Unary operations, because they operate on one relation. The other three fundamental operations Union, set difference, and cartesian product operate on pairs of relations, and are therefore, called binary operations.

SELECT operation.

→ The select operation selects tuple that satisfy a given ~~procedure~~ predicate (condition).

Notation :- $\sigma_p(r)$

$\sigma \Rightarrow$ Used for selection

$r \Rightarrow$ Used for relation.

$P \Rightarrow$ Used as a propositional logic formula which may use connectors like : AND, OR and NOT. These relational can use as relational

operators like $=, \neq, \geq, <, >, \leq$.

→ Let's take an example of the student table fetch data for students with 'age' more than 17.

$\sigma_{age > 17}$ (STUDENT)

This will fetch the tuples (rows) from table STUDENT for which age will be greater than 17.

→ You can also use 'and', 'or', etc. operators to specify two conditions.

for example,

$\sigma_{age > 17 \text{ and } gender = 'male'}$ (STUDENT).

This will return tuples (rows) from table STUDENT with information of male students of age more than 17.

PROJECT operation.

→ Project operation is used to project only a certain set of attributes of a relation. In simple words, if you want to see only the names of

⇒ all of the students in the student table, then you can use project operation.

⇒ It will only project or show the columns or attributes asked for, and will also remove duplicate data from the columns.

Syntax:

$$\Pi_{A_1, A_2, \ldots}(r)$$

Where $A_1, A_2, \ldots$ etc are attribute names (column names)

For example

$$\Pi_{Name, Age}(Student)$$

Above statement will show us only the Name and Age columns for all the rows of data in student table.

eg:-

R

A	B	C
1	2	4
2	2	3
3	2	3
4	3	4

$\Pi_{B, C}(R)$

B	C
2	4
2	3
3	4

RENAME operation

→ Rename operator is used to given another name to relation.

Syntax:-

$$\rho\,(\text{Relation}_2, \text{Relation}_1)$$

For example, we can use the rename operator to rename STUDENT relation to STUDENT1,

$$\rho\,(\text{STUDENT1}, \text{STUDENT})$$

→ The Rename operation allows us to rename the output relation.

UNION operation.

→ Suppose there are two tuples R and S. The union operation contains all the tuples that are either in R or S or both in R & S.

→ It eliminates the duplicate tuples. It is denoted by ∪.

Notation:- $R \cup S$.

→ The two relation instances are said to be union compatible if the following conditions hold:

- They have the same no. of attributes.
- Corresponding attributes taken in the order from left to right, have the same domains.

→ As an example, consider a query to find the names of all bank customers who have either an account or a loan or both. Assume that borrower and depositor relations have the following schemas.

Borrower (customer_name, loan_number)

Depositor (customer_name, Account_number).

Borrower

customer_name	loan_number
Ajith	HL 101
John	HL 102
Sandhya	HL 205
Adam	HL 231

Depositor

Customer_name	Account_number.
Ajith	52453340
Sreedevi	27443648
Sandhya	9544537
Rajan	75473694

To find the names of all customers with a loan in the bank, we use the projection operation on borrower.

$$\Pi_{customer\_name}(Borrower)$$

And to find the names of all customers with an account in the bank, we use the projection operation on depositor.

$$\Pi_{customer\_name}(Depositor)$$

To answer the query we need Union of these two sets.

$$\Pi_{customer\_name}(Borrower) \cup \Pi_{customer\_name}(Depositor)$$

The result of this query appears as shown below:-

Customer-name
Ajith
John
Sandhya
Adam
Sreedevi
Rajan

SET DIFFERENCE Operation

→ This operation is used to find data present in one relation and not present in the second relation.

→ This operation is also applicable on two relations, just like union operation.

→ The set difference operation (-) between two relations R and s, denoted by R-s, returns a relation instance containing all tuples that occur in relation instance R

but not in relation instance s.

→ For example, we can find all customers of the bank who have an account but not a loan

$\Pi_{customer\_name}(Depositor) - \Pi_{customer\_name}(Borrower)$

The result of this query appears as shown below:-

Customer_name
~~John~~ Sreedevi
~~Adam~~ Rajan.

CARTESIAN PRODUCT

→ Cartesian product or cross product between two relations let say A and B, so cross product b/w A × B will results all the B attributes of A followed by each attribute of B. Each record of A will pairs with every record of B.

→ Example :-

A

Name	Age	Sex
Ram	14	M
Sona	15	F
Arun	19	M

B

Id	course
1	DS
2	DBMS

A X B

Name	Age	Sex	Id	course
Ram	14	M	1	DS
Ram	14	M	2	DBMS
Sona	15	F	1	DS
Sona	15	F	2	DBMS
Arun	19	M	1	DS
Arun	19	M	2	DBMS

Note :- If A has 'n' tuples and B has 'm' tuples, then A X B will have 'n*m' tuples.

Additional Operations.

8). The SET-INTERSECTION operation:-

The intersection operation ($\cap$) between two relations R and S, denoted by $R \cap S$, returns a relation instance containing all tuples that occur both in relation instances R and S.

As an example, suppose that we wish to find all bank customers who have both an account and a loan. Using set-intersection, we can write

$\Pi_{customer\_name}(borrower) \cap \Pi_{customer\_name}(Depositor)$

Borrower

Customer_name	loan_number
Ajith	HL 101
John	HL 205
Sandhya	HL 143
Adam	HL 231

Depositor

Customer_name	account_number
Ajith	98442233
Sreedevi	11253418
Sandhya	97332745
Rajan	27364342

The result of this query would be

Customer_name
Ajith
Sandhya

JOIN Operation:-

→ The join operation, denoted by the join symbol ⋈, is one of the most useful operations in relational algebra and the most commonly used way to combine information from two or more relations.

→ Although, a join can be defined as cartesian product followed by selections and projections. Joins arise much more frequently in practice than plain cartesian products.

Natural Join. (⋈)

→ Natural join is a binary operator. Natural join between two or more relations will results set of all combination of tuples where they have equal common attribute.

eg:-

Emp

Name	Id	Dept_name
A	120	IT
B	125	HR
C	110	Sale
D	111	IT

Dep

Dept_name	Manager
Sale	Y
Prod	Z
IT	A

Emp ⋈ Dep

Name	Id	Dept_name	Manager
A	120	IT	A
C	110	Sale	Y
D	111	111	A

Conditional Join. ($\bowtie_c$)

→ Conditional join works similar to natural join. In natural join, by default condition is equal between common attributes. While in conditional join we can specify the any condition.

eg:-

R

Id	Sex	Marks
1	F	45
2	F	55
3	F	60

S

Id	sex	Marks
10	M	20
11	M	22
12	M	59

Join between R and S with condition

R.marks >= S.marks.

$R \bowtie_c$ R.marks >= S.marks 'S

R.Id	R.Sex	R.Marks	S.Id	S.Sex	S.Marks
1	F	45	10	M	20
1	F	45	11	M	22
2	F	55	10	M	20
2	F	55	11	M	22
3	F	60	10	M	20
3	F	60	11	M	22
3	F	60	12	M	59

Theta join (Θ)

Theta join combines tuples from different relations provided they satisfy the theta condition. The join condition is denoted by the symbol Θ.

Notation:- $R_1 \bowtie_\Theta R_2$

→ R_1 and R_2 are relations having attributes $(A_1, A_2, \ldots A_n)$ and $(B_1, B_2, \ldots B_n)$ such that

the attributes don't have anything in common, ie $R_1 \cap R_2 = \phi$.

Student

Sid	Name	Std
101	Alex	10
102	Maria	11

Subjects

Class	subject
10	Maths
10	English
11	Music
11	Sports.

Student $\bowtie_{\text{Student.Std} = \text{subject.class}}$ subject

sid	Name	std	class	subject
101	Alex	10	10	Maths
101	Alex	10	10	English
102	Maria	11	11	Music
102	Maria	11	11	sports.

Equi join:-

→ Equi join is a special case of conditional join

→ when theta join uses only equality comparison operator, it is said to be equi join.

-The above example corresponds to equi join.

OUTER JOINS

→ Theta join, Equi-join and natural join are called inner joins; An inner join includes only those tuples with matching attributes and the rest are discarded in the resulting relation. Therefore we need to use outer joins to include all the tuples from the participating relations in the resulting relation. There are three kinds of outer joins:-

- Left outer join
- Right outer join
- Full outer join.

Left outer join (R ⟕ S)

→ All the tuples from the left relation, R, are included in the resulting relation. If there are tuples in R without any matching tuples in the right relation S, then the S-attributes of the resulting relation are made NULL.

eg:-

Course

A	B
100	Database
101	Mechanics
102	Electronics

Hod

C	D
100	Alex
102	Mariya
104	Mira

Course ⟕ Hod

A	B	C	D
100	Database	100	Alex
101	Mechanics	—	—
102	Electronics	102	Mariya

Right outer join (R ⟖ S)

→ All the tuples from the right relation, S are included in the resulting relation. If there are tuples in S without any matching tuple in R, then the R- attributes of resulting relation are made NULL.

Course ⟖ Hod

A	B	C	D
100	Database	100	Alex
102	Electronics	102	Maaiya
_	_	104	Mira

Full outer join (R ⟗ S)

→ All the tuples from both participating relations are included in the resulting relation. If there are no matching tuple for both relations,

their respective unmatched attributes are made NULL.

course ~~⋈~~ Hod

A	B	C	D
100	Database	100	Alex
101	Mechanics	–	–
102	Electronics	102	Mariya
–	–	104	Mira

The division operator

→ Division operator $A \div B$ can be applied if and only if:

- Attributes of B is proper subset of attributes of A.
- The relation returned by division operator will have attributes = (All attributes of A ÷ All ~~attr~~ attributes of B)

- the relation returned by division operator will return those tuples from relation A which are associated to every B's tuple

eg:-

STUDENT_SPORTS

Rollno	Sports
1	Badminton
2	Cricket
2	Badminton
4	Badminton

ALL_SPORTS

SPORTS
Badminton
Cricket

Consider the relation STUDENT_SPORTS and ALL_SPORTS given in tables above

To apply division operator as

STUDENT_SPORTS ÷ ALL_SPORTS

- The operation is valid as attributes in ALL_SPORTS is a proper subset of attributes in STUDENT_SPORTS.
- The attributes in resulting relation will have attributes {ROLL_NO, SPORTS} − {SPORTS}

= ~~Roo~~ ROLL_NO.

- The tuples in resulting relation will have those ROLL_NO which are associated with all B's tuples {Badminton, Cricket}.

ROLL_NO 1 and 4 are associated to Badminton only ROLL_NO. 2 is associated to all tuples of B. So the resulting relation will be.

ROLL_NO
2

RELATIONAL CALCULUS.

→ Contrary to relational algebra which is procedural query language to fetch data and which also explains how it is done:

→ Relational calculus is non-procedural query language and has no description about how the query will work or the data will be fetched. It only focuses on what to do, and not on how to do it.

→ The relational calculus has had a big influence on the design of commercial query languages such as SQL and QBE (Query By ~~Ex~~ Example).

→ Relational calculus exists in two forms:-

1. Tuple Relational calculus (TRC)
2. Domain Relational calculus (DRC).

Tuple Relational Calculus (TRC)

→ In tuple relational calculus, we work on filtering tuples based on the given condition.

→ In this form of relational calculus, we define a tuple variable, specify the table (relation) name on which the tuple is to be searched for, along with a condition.

→ ~~we can also specify column name using '.' (dot) operator, with the tuple variable to only get a certain attribute (column) in result.~~

Eg:- STUDENT (Roll-no, Name, Department_no, sex)

Query:- Find Roll no and name of students in department no 2.

~~{ t : Roll no, t.Name | Student~~

~~STUDENT (t) ∧ t. Department no := 7 }~~

→ General format :- $\{t \mid P(t)\}$

It is set of all tuples t, such that predicate P is true for t

eg :- $\{t \mid t \in student \wedge t[age] \geq 15\}$

List all student details who have age ≥ 15.

eg :- Loan (loan_number, branch_name, amount)

Query :- Find the branch_name, loan_number, and amount for loan over $1500.

$\{t \mid t \in loan \wedge t[amount] > 1500\}$

Query :- Find the loan numbers for each loan of an amount greater than $1500.

$\{t \mid \exists L \in Loan\ (t[loan\_number] = L[loan\_number] \wedge L[amount] > 1500\}$.

→ $\{t \mid P(t)\}$

t ⇒ resulting tuples

P(t) ⇒ known as predicate and these are the conditions that are used to fetch t.

P(t) may have various conditions logically combined with OR (∨), AND (∧), NOT (¬).

→ It also uses quantifiers:

$\exists t \in r\ (Q(t))$ = "there exists" a tuple in t in relation r such that predicate Q(t) is true

$\forall t \in r\ (Q(t))$ = Q(t) is true "for all" tuples in relation r.

Eg:-

Customer

Customer_name	Street	City
Saurabh	A7	Patiala
Mehak	B6	Jalandhar
Sumiti	D9	Ludhiana
Ria	A5	Patiala

Branch

Branch_name	Branch_city
ABC	Patiala
DEF	Ludhiana
GHI	Jalandhar.

Account

Account_no	Branch_name	Balance
1111	ABC	50000
1112	DEF	10000
1113	GHI	9000
1114	ABC	7000

Loan

Loan_no	Branch_name	Amount
L33	ABC	10000
L35	DEF	15000
L49	GHI	9000
L98	DEF	65000

Borrower.

Customer_name	loan_no
Saurabh	L33
Mehak	L49
Ria	L98

Depositor.

Customer_name	Account_no
Saurabh	1111
Mehak	1113
Sumiti	1114

query-1:- Find the loan number, branch, amount of loans of greater than or equal to 10000.

$\{t \mid t \in Loan \wedge t[amount] >= 10000\}$

Resulting relation:

Loan_no	Branch_name	Amount
L33	ABC	10000
L35	DEF	15000
L98	DEF	65000

In the above query, t [amount] is known as tuple variable

query-2:- Find the loan number for each loan of an amount greater or equal to 10000.

$\{t \mid \exists s \in Loan (t[loan\_no] = s[loan\_no] \wedge s[amount] >= 10000)\}$.

Resulting relation:-

loan_no
L33
L35
L98

query-3:- Find the names of all customers who have a loan and an account at the bank.

$$\{t \mid \exists s \in \text{Borrower}(t[\text{customer\_name}] = s[\text{customer\_name}]) \wedge \exists u \in \text{depositor}(t[\text{customer\_name}] = u[\text{customer\_name}])\}$$

Resulting relation:-

Customer_name
Saurabh
Mehak

query-4:- Find the names of all customers having a loan at the "ABC" branch.

$$\{t \mid \exists s \in \text{borrower}(t[\text{customer\_name}] = s[\text{customer\_name}] \wedge \exists u \in \text{Loan}(u[\text{branch\_name}] = \text{"ABC"} \wedge u[\text{loan\_no}] = s[\text{loan\_no}]))\}$$

Resulting relation:-

Customer_name
Saurabh

Domain Relational Calculus (DRC)

→ The second form of relational calculus is known as domain relational calculus. In domain relational calculus, filtering variable uses the domain of attributes.

→ Domain relational calculus uses the same operators as tuple calculus. It uses logical connectives (∧), (∨), and ¬(not)

→ It uses Existential (∃) and Universal Quantifiers (∀) to bind the variable

Notation:-

$$\{a_1, a_2, \ldots a_n / P(a_1, a_2, \ldots a_n)\}$$

Where $a_1, a_2, \ldots a_n$ are attributes

P stands for formula built by inner attributes.

eg:-

$$\{\langle r, n, a\rangle / \langle r, n, a\rangle \in \text{students} \wedge a \geq 15\}.$$

~~Loan (loan_no, branch_name, amount)~~

- branch (branch_name, branch_city, assets)
- customer (customer_name, customer_street, customer_city)
- loan (loan_no, branch_name, amount)
- borrower (customer_name, loan_number)
- account (account_no, branch_name, balance)
- depositor (customer_name, account_number)

query-1:- Find the loan number, branch_name and amount for loans over $1500.

$$\{\langle l, b, a\rangle \mid \langle l, b, a\rangle \in \text{loan} \wedge a > 1500\}$$

query-2:- Find all loan numbers for loans with an amount greater than $1500.

$$\{ \langle l \rangle \mid \exists\, b, a\ (\langle l, b, a \rangle \in loan \wedge a > 1500) \}$$

query-3:- Find the names of all customers who have a loan from the pune branch and find the loan amount.

$$\{ \langle c, a \rangle \mid \exists\, l\ (\langle c, l \rangle \in borrower \wedge \exists\, b\ (\langle l, b, a \rangle \in loan \wedge b = \text{"pune"})) \}$$

query-4:- Find the names of all customers who have a loan, an account or both at Pune branch.

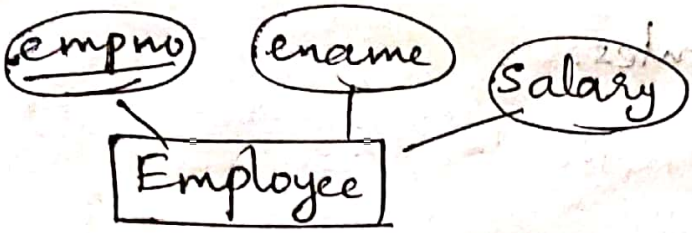
$$\{ \langle c \rangle \mid \exists\, l\ (\langle c, l \rangle \in borrower \wedge \exists\, b, a\ (\langle l, b, a \rangle \in loan \wedge b = \text{"pune"})) \vee \exists\, a\ (\langle c, a \rangle \in depositor \wedge \exists\, b, n\ (\langle a, b, n \rangle \in account \wedge b = \text{"pune"})) \}$$

RELATIONAL DATABASE DESIGN USING E-R TO RELATIONAL MAPPING.

→ ER Model, when conceptualized into diagrams, gives a good overview of entity-relationship, which is easier to understand. ER diagrams can be mapped to relational schema, that is it is possible to create relational schema using ER diagram. We can not import all the ER constraints into relational model, but an approximate schema can be generated.

→ There are several processes and algorithms available to convert ER diagrams into Relational schema. Some of them are automated and some of them are manual.

- eg:-

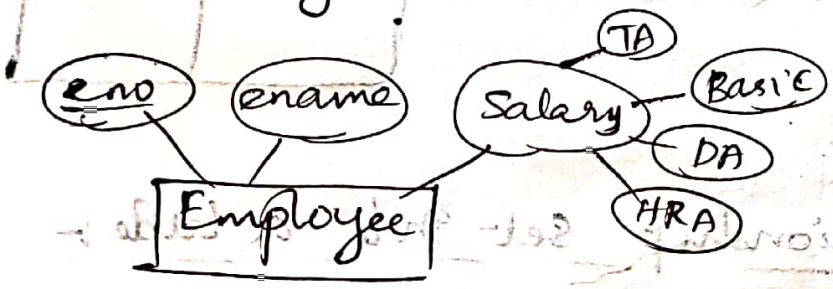


Employee

empno	ename	salary

- Entity set mapped with a relation.

2). Entity set with composite attribute :-



Employee

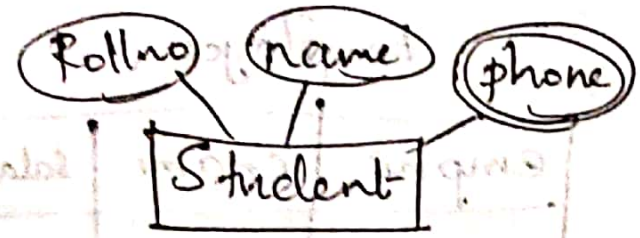
eno	ename	Basic	TA	DA	HRA

- In relational model don't represent the composite attributes

3). Entity set with multivalued attributes :-

→ If an entity contains multivalued attributes it is represented with two relations, one with all simple attributes and other with key and all

multivalued attributes.

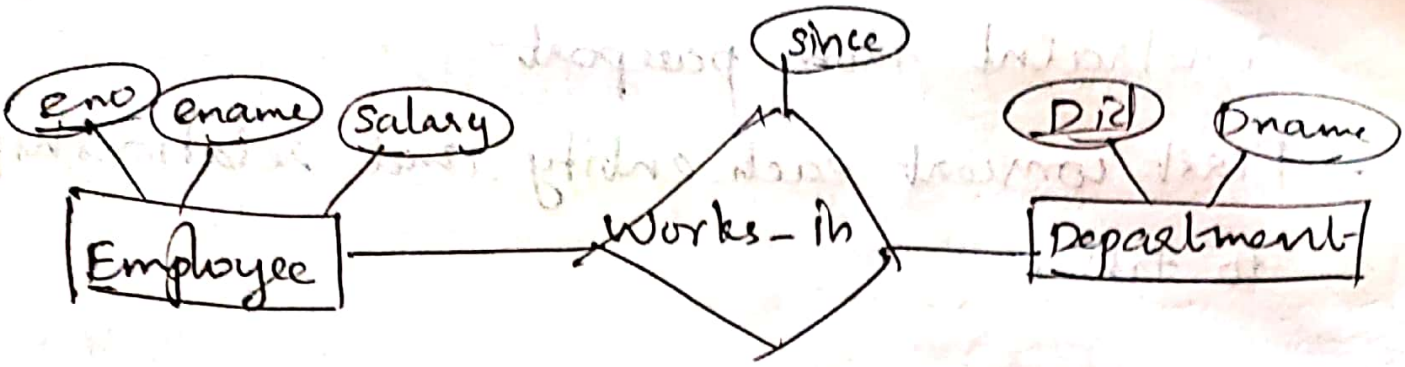


Rollno	name

Rollno	phone

* Translating relationship set into a table :-

- A relationship set is mapped into a relation, the attributes of a relation includes:
  - The key attributes of the participating relation and are declared as foreign keys to the respective relation.
  - Descriptive ~~are~~ attributes (if any)
  - Set of non descriptive attributes is ~~are~~ the primary key of the relation



Employee.

eno	ename	salary

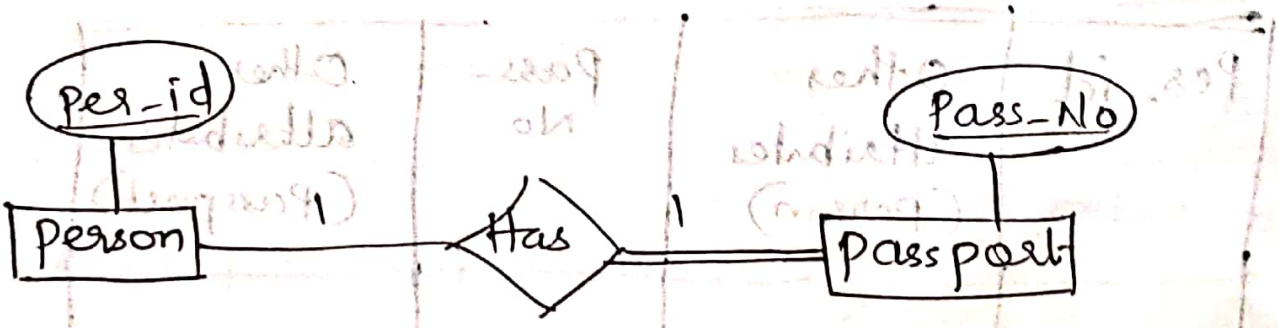
Works_in

eno	Did	Since

Department

Did	Dname

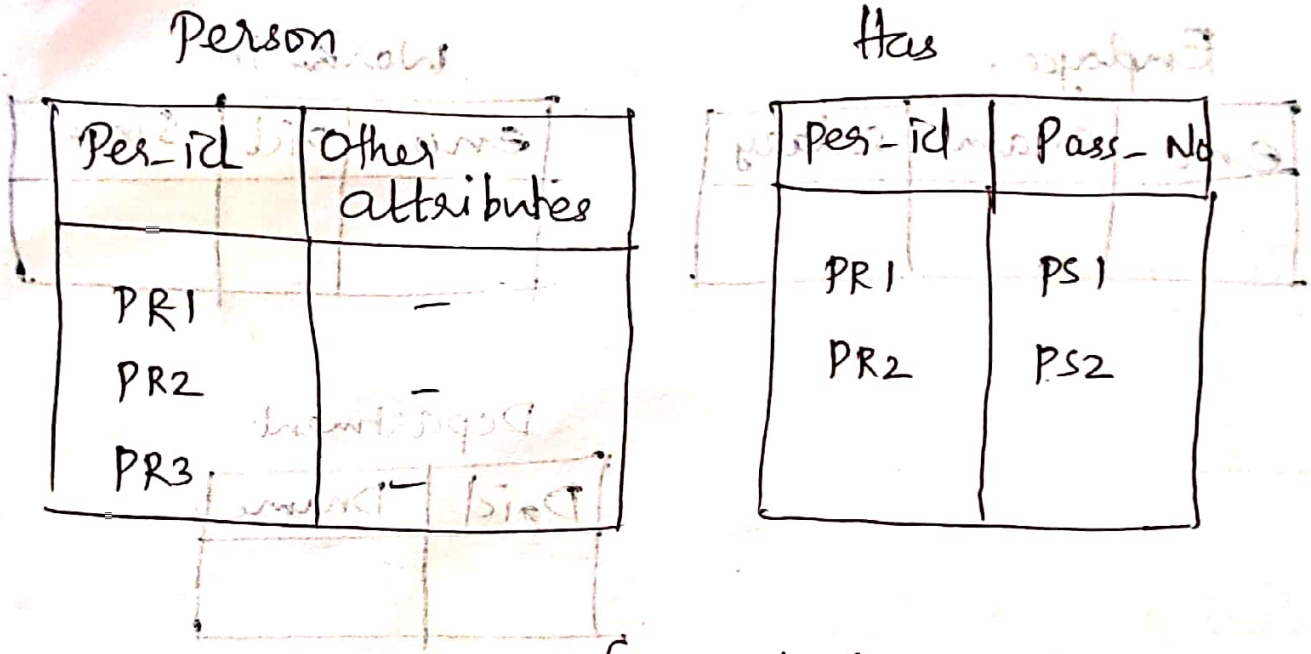
*). Bineiry relationship with 1:1 cardinality with total participation of an entity.



→ A person has 0 or 1 passport number and passport is always owned by 1 person. so it is 1:1 cardinality with full participation

Constraint from passport

→ First convert each entity and relationship to tables.



Person

Per_id	Other attributes
PR1	–
PR2	–
PR3	–

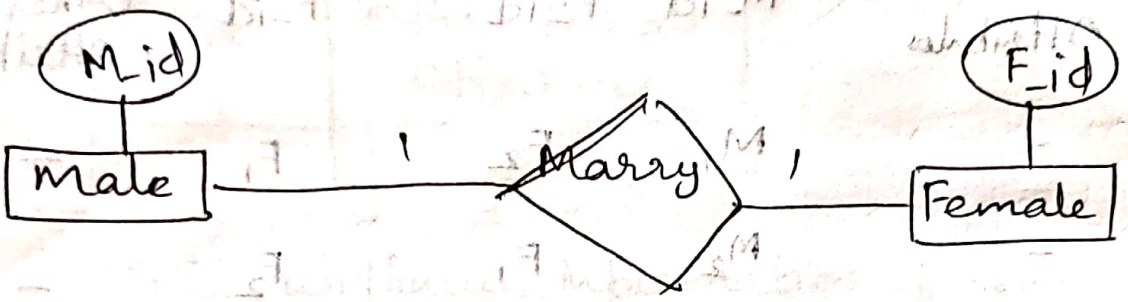
Has

Pers_id	Pass_No
PR1	PS1
PR2	PS2

As we can see from table, each person_id and Pass_No has only one entry in Has table. So we can merge all tables.

Per_id	Other attributes (person)	Pass-No	Other attributes (Passport)

*). Binary relationship with 1:1 cardinality and partial participation of both entities:-



→ A male marries 0 or 1 female and vice versa as well. So it is 1:1 cardinality with partial participation constraint from both.

First convert each entity and relationship to tables. Male table corresponds to male entity with key as M_id. Similarly Female table corresponds to Female entity with key as F_id. Marry table represents relationship between Male and Female (Which male marries which Female). So it will take attribute M_id from Male and F_id from Female.

Male

M_id	Other Attributes
M_1	–
M_2	–
M_3	–

Marry

M_id	F_id
M_1	F_2
M_2	F_1

Female

F_id	Other Attributes
F_1	–
F_2	–
F_3	–

As we can see from table some males and some females do not marry. If we merge 3 tables into 1, for some M_id, F_id will be NULL. So there is no attribute which is always not null. so we can't merge all three tables into 1. We can convert into 2 tables.

M_id	Other Male Attributes	F_id

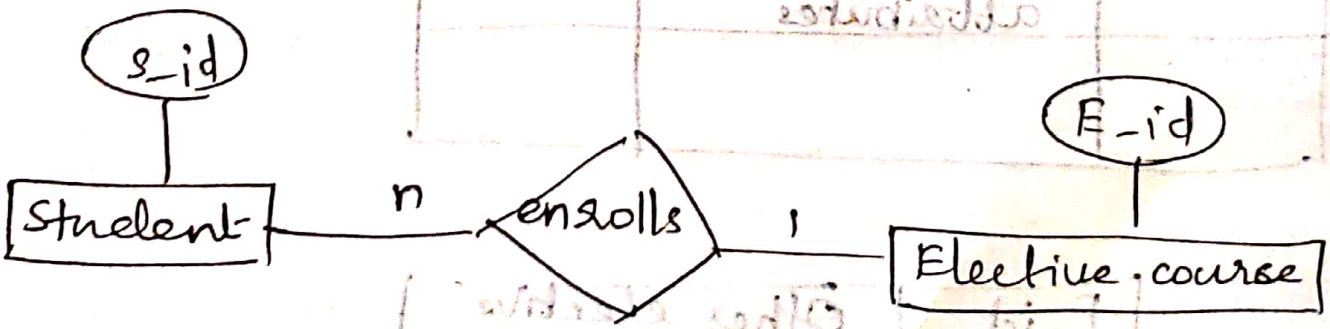
M_id who are married will have F_id associated. For others it will be NULL.

F_id	Other Female Attributes.

Table will have information of all females.

Note:- Binary relationship with 1:1 cardinality will have 2 table if partial participation of both entities in the relationship. If atleast ~~two~~ one entity has total participation, number of tables required will be 1.

* Binary relationship with n:1 cardinality



In this scenario, every student can enroll only in one elective course, but for an elective course there can be more than one student

Student

S_id	Other student attributes
S_1	–
S_2	–
S_3	–
S_4	–

Enrolls

S_id	E_id
S_1	E_1
S_2	E_2
S_3	E_1
S_4	E_1

Elective course

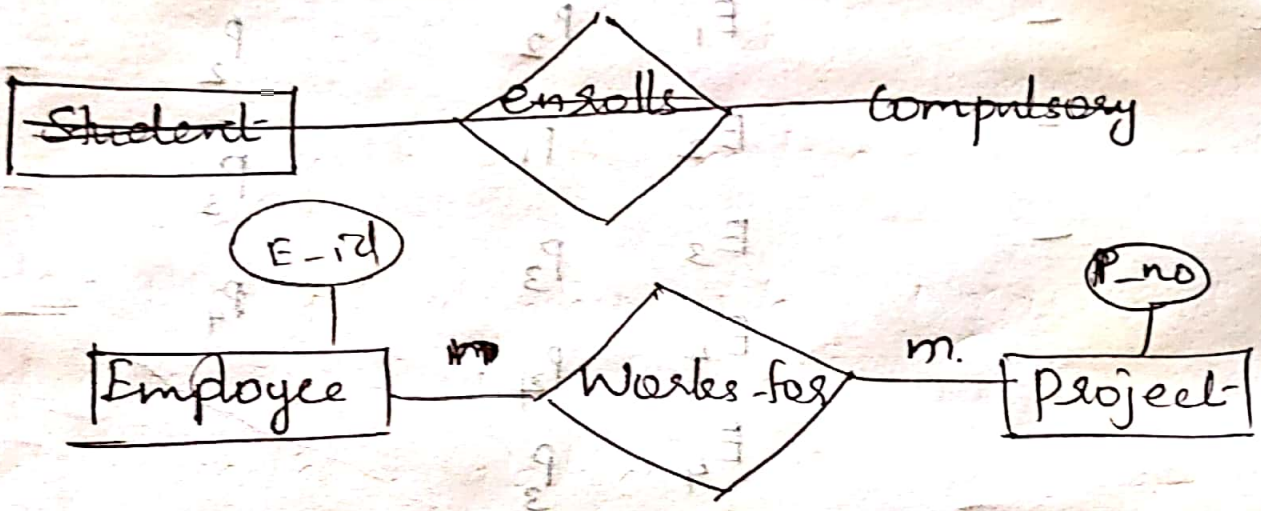
E_id	Other attributes
E_1	–
E_2	–
E_3	–

As we can see from table, S_id is not repeating in enrolls table. So we can merge Student and enrolls table.

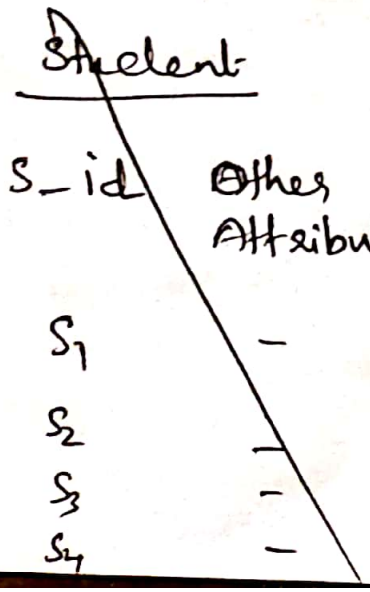
S_id	Other student attributes	E_id

E_id	Other elective Course attributes

(*). Binary relationship with m:n Cardinality



→ In this scenario, every employee can works-for more than 1 project and for a project there can be more than one employee.



Employee		Works_for		Project	
E_id	other	E_id	P_no	P_id	other
E_1	–	E_1	P_1	P_1	–
E_2	–	E_1	P_2	P_2	–
E_3	–	E_2	P_1	P_3	–
E_4	–	E_3	P_3	P_4	–
		E_4	P_4		
		E_4	P_3		

As we can see, E_id and P_no both are repeating in Works_for table. so these can't be merged.