# Database Management Systems

# UNIT-II

**Data Models**

## E-R diagram

ER diagram or Entity Relationship diagram is a conceptual model that gives the graphical representation of the logical structure of the database. It shows all the constraints and relationships that exist among the different components.

Components of E-R diagram

An ER diagram is mainly composed of following three components:

* + 1. Entity Sets
    2. Attributes
    3. Relationship Set

Consider the following Student table. This complete table is referred to as “Student Entity Set” and each row represents an “entity”.

|  |  |  |
| --- | --- | --- |
| **Roll no** | **Name** | **Age** |
| 1 | Akshay | 20 |
| 2 | Rahul | 19 |
| 3 | Pooja | 20 |
| 4 | Aarti | 19 |

Table 2.1: Student table.

Representation as ER Diagram

The above table may be represented as ER diagram as shown in Figure. [2.1](#_bookmark2). Here, Roll no is a primary key that can identify each entity uniquely. Thus, by using student’s roll number, a student can be identified uniquely.

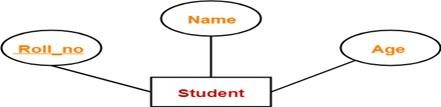


Figure 2.1: Student E-R diagram.

* + 1. **ER Diagram Symbols**

An ER diagram is composed of several components and each component in ER diagram is represented using a specific symbol. ER diagram symbols are discussed below.

##### Entity sets

* + - 1. Entity Sets

An entity set is a set of same type of entities. An entity refers to any object having

-

* + - * + Either a physical existence such as a particular person, office, house or car.
        + Or a conceptual/logical existence such as a school, a university, a company or a job.
        + Attributes are associated with an entity set.
        + Attributes describe the properties of entities in the entity set. Based on the values of certain attributes, an entity can be identified uniquely.

An entity set may be of the following two types.

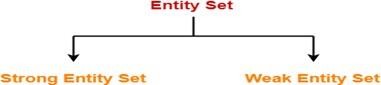


Figure 2.2: Entity sets.

**A strong entity** set is an entity set that contains sufficient attributes to uniquely identify all its entities. In other words, a primary key exists for a strong entity set. Primary key of a strong entity set is represented by underlining it. It is represented using a single rectangle.



Figure 2.3: Strong and weak entity sets.

**A weak entity** set is an entity set that does not contain sufficient attributes to uniquely identify its entities. In other words, a primary key does not exist for a weak entity set. However, it contains a partial key called as a discriminator. Discriminator can identify a group of entities from the entity set. Discriminator is represented by underlining with a dashed line. It is represented using a double rectangle.

The combination of discriminator and primary key of the strong entity set makes it possible to uniquely identify all entities of the weak entity set. Thus, this combina - tion serves as a primary key for the weak entity set as shown in Figure. [2.4.](#_bookmark3)



Figure 2.4: Primary key of a weak entity set.

A double rectangle is used for representing a weak entity set. 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 set with the relation - ship set. Total participation always exists in the identifying relationship.

Consider the following ER diagram shown in Figure.[2.5 .](#_bookmark4) In this ER diagram

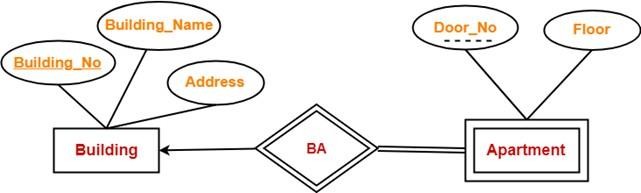


Figure 2.5: Example ER diagram.

* One strong entity set “Building” and one weak entity set “Apartment” are related to each other.
* Strong entity set “Building” has building number as its primary key.
* Door number is the discriminator of the weak entity set “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.
* Double line between Apartment and relationship set signifies total participation.
* It suggests that each apartment must be present in at least one building.
* Single line between Building and relationship set signifies partial participation.
* It suggests that there might exist some buildings which has no apartment.

##### Attributes

Attributes describe the properties of entities in an Entity Set. There exists a specific do- main or set of values for each attribute from where the attribute can take its values.

The following are the types of attributes and the symbols are shown in Figure. [2.6.](#_bookmark5)

* + - 1. Simple attributes
      2. Composite attributes
      3. Single valued attributes
      4. Multi valued attributes
      5. Derived attributes
      6. Key attributes

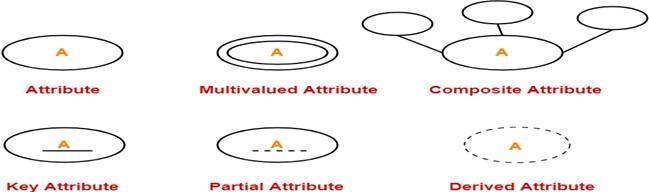


Figure 2.6: Attribute symbols used in E-R diagrams.

Simple Attributes

Simple attributes are those attributes which cannot be divided further. Here, all the attributes are simple attributes as they cannot be divided further. They are represented with an oval/ellipse.

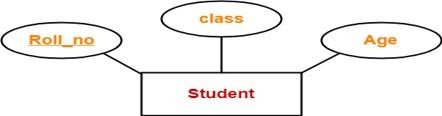


Figure 2.7: Simple attribute.

Composite Attributes

Composite attributes are those attributes which are composed of many other simple at - tributes.Here, the attributes “Name” and “Address” are composite attributes as they are composed of many other simple attributes.

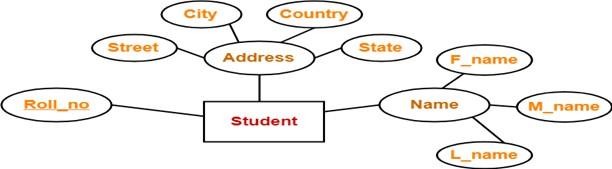


Figure 2.8: Composite attribute.

Single Valued Attributes

Single valued attributes are those attributes which can take only one value for a given en - tity from an entity set. Here, all the attributes are single valued attributes as they can take only one specific value for each entity. They are represented with an oval/ellipse.

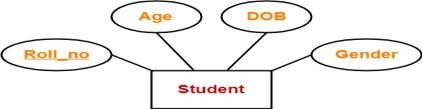


Figure 2.9: Single valued attribute.

Multi Valued Attributes

Multi valued attributes are those attributes which can take more than one value for a given entity from an entity set. They are represented with double oval/ellipse. Here, the attributes “Mob no” and “Email id” are multi valued attributes as they can take more than one values for a given entity.

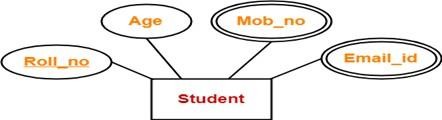


Figure 2.10: Multi valued attribute.

Derived Attributes

Derived attributes are those attributes which can be derived from other attribute(s). Here, the attribute “Age” is a derived attribute as it can be derived from the attribute “DOB”. Derived attributes are represented with a dashed oval/ellipse.



Figure 2.11: Derived attribute.

Key Attributes

Key attributes are those attributes which can identify an entity uniquely in an entity set. Here, the attribute “Roll no” is a key attribute as it can identify any student uniquely. It is represented with an underline in the oval/ellipse.

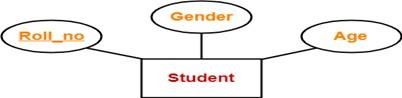


Figure 2.12: Key attribute.

##### Generalization, Specialization, and Aggregation

**Generalization** is like a bottom-up approach in which two or more entities of lower level combine to form a higher level entity if they have some attributes in common.

* In generalization, an entity of a higher level can also combine with the entities of the lower level to form a further higher level entity.
* Generalization is more like subclass and superclass system, but the only difference is the approach. Generalization uses the bottom-up approach.
* In generalization, entities are combined to form a more generalized entity, i.e., sub- classes are combined to make a superclass.

For example, Faculty and Student entities can be generalized and create a higher level entity Person.

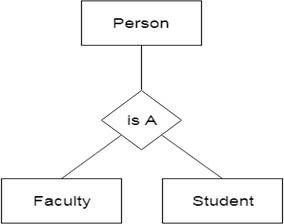


Figure 2.13: Generalization.

**Specialization** is a top-down approach, and it is opposite to Generalization.

* In specialization, one higher level entity can be broken down into two lower level entities.
* Specialization is used to identify the subset of an entity set that shares some distin- guishing characteristics.
* Normally, the superclass is defined first, the subclass and its related attributes are defined next, and relationship set are then added.

For example: In an Employee management system, EMPLOYEE entity can be spe- cialized as TESTER or DEVELOPER based on what role they play in the company.

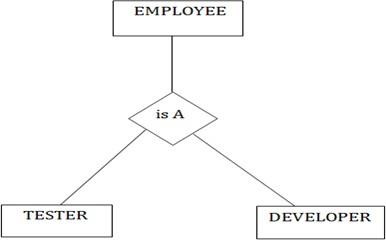


Figure 2.14: Specialization.

Specialization or generalization are represented in an E-R diagram as shown in Figure. [2.15](#_bookmark6).



Figure 2.15: Specialization or generalization symbol.

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

For example: Center entity offers the Course entity act as a single entity in the relationship which is in a relationship with another entity visitor. In the real world, if a visitor visits a coaching center then he will never enquiry about the Course only or just about the Center instead he will ask the enquiry about both.

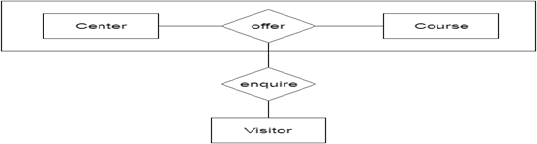


Figure 2.16: Aggregation.

##### Relationship Sets

Relationship defines an association among several entities. A relationship set is a set of same type of relationships. A relationship set may be of the following two types.

* + - 1. Strong relationship
         * A strong relationship exists between two strong entity sets.
         * It is represented using a diamond symbol.
      2. Weak relationship
         * A weak or identifying relationship exists between the strong and weak entity set.
         * It is represented using a double diamond symbol.



Figure 2.17: Relationship set symbols in E-R diagram.

##### Participation Constraints

Participation constraint defines the least number of relationship instances in which an entity has to necessarily participate. There are two types of participation constraints and the symbols are as shown in Figure. **??**.

* + - 1. Partial participation

Partial participation is represented using a single line between the entity set and relationship set.

* + - 1. Total participation

Total participation is represented using a double line between the entity set and relationship set.

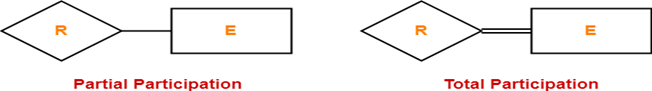


Figure 2.18: Participation symbols in E-R diagram.

##### Cardinality Constraints / Ratios

Cardinality constraint defines the maximum number of relationship instances in which an entity can participate. There are 4 types of cardinality ratios:

* + - 1. Many-to-many cardinality (*m* : *n*)
      2. Many-to-one cardinality (*m* : 1)
      3. One-to-many cardinality (1 : *n*)
      4. One-to-one cardinality (1 : 1)

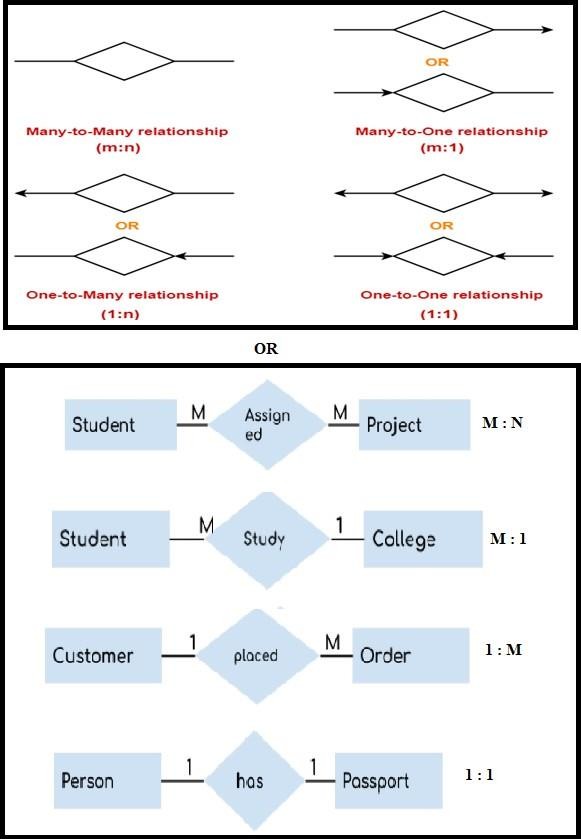


Figure 2.19: Relations in E-R diagram.

##### Relationship

Relationships are represented by diamond-shaped box. Name of the relationship is written inside the diamond-box. All the entities (rectangles) participating in a relationship, are

connected to it by a line. A relationship is defined as an association among several entities. Example: ‘Enrolled in’ is a relationship that exists between entities Student and Course.



Figure 2.20: Relations in E-R diagram.

Relationship Set

A relationship set is a set of relationships of same type. Set representation of ER diagram shown in Figure. [2.20](#_bookmark7) is represented as:

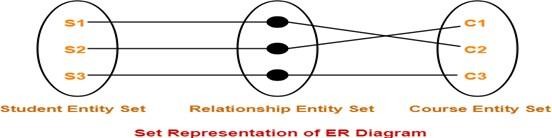


Figure 2.21: Relationship set in E-R diagram.

Degree of a Relationship Set

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

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

Types of Relationship Sets

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

* + - 1. Unary relationship set

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

* + - 1. Binary relationship set

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

* + - 1. Ternary relationship set

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

* + - 1. N-ary relationship set

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

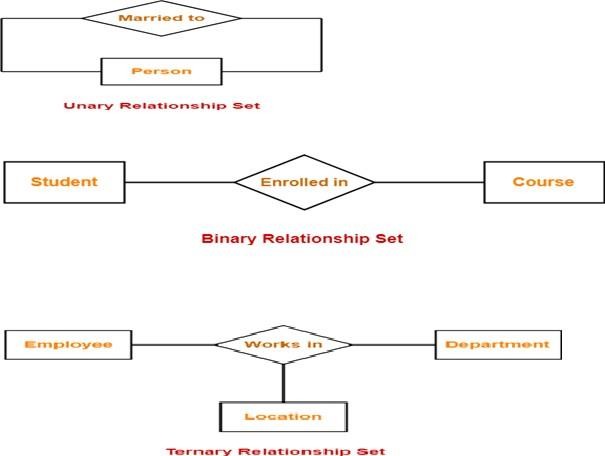


Figure 2.22: Relationship types in E-R diagram.

##### Participation Constraints

Participation constraints define the least number of relationship instances in which an en - tity must compulsorily participate. There are two types of participation constraints.

* + - 1. Total Participation

It specifies that each entity in the entity set must compulsorily participate in at least one relationship instance in that relationship set. That is why, it is also called as mandatory participation. Total participation is represented using a double line between the entity set and relationship set.

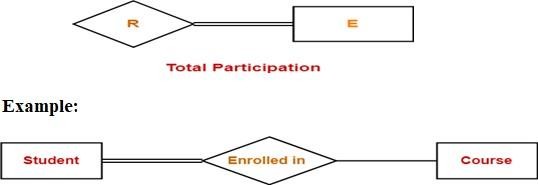


Figure 2.23: Total participation in E-R diagram.

Here, double line between the entity set “Student” and relationship set “Enrolled in” signifies total participation. It specifies that each student must be enrolled in at

least one course.

* + - 1. Partial Participation

It specifies that each entity in the entity set may or may not participate in the relationship instance in that relationship set. That is why, it is also called as optional participation. Partial participation is represented using a single line between the entity set and relationship set.

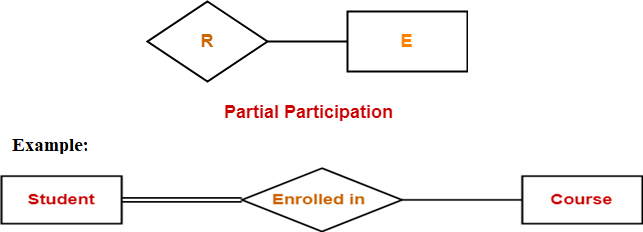


Figure 2.24: Partial participation in E-R diagram.

Here, Single line between the entity set “Course” and relationship set “Enrolled in” signifies partial participation. It specifies that there might exist some courses for which no enrollments are made.

Relationship between Cardinality and Participation Constraints

Minimum cardinality tells whether the participation is partial or total.

* If minimum cardinality = 0, then it signifies partial participation.
* If minimum cardinality = 1, then it signifies total participation.

Maximum cardinality tells the maximum number of entities that participates in a rela- tionship set.

## Relational Model

A database is a collection of 1 or more ‘relations’, where each relation is a table with rows and columns. The major advantages of the relational model over the older data models are:

* + 1. It is simple and elegant.
    2. Simple data representation.
    3. The ease with which even complex queries can be expressed.

The main construct for representing data in the relational model is a ‘relation’. A relation consists of following fields:

##### Relation Schema

The relation schema describes the column heads for the table. The schema specifies the relation’s name, the name of each field (column, attribute) and the ‘domain’ of each field. A domain is referred to in a relation schema by the domain name and has a set of associated values.

Example: Student information in a university database to illustrate the parts of a relation schema. Students (Sid: string, name: string, login: string, age: integer, gross: real) this says that the field named ‘sid’ has a domain named ‘string’. The set of values associated with domain ‘string’ is the set of all character strings.

##### Relation Instance

This is a table specifying the information. An instance of a relation is a set of ‘tuples’, also called ‘records’, in which each tuple has the same number of fields as the relation schemas. A relation instance can be thought of as a table in which each tuple is a row and all rows have the same number of fields. The relation instance is also calle d as ‘relation’. Each relation is defined to be a set of unique tuples or rows.

Example: Fields (Attributes, Columns) Field names Tuples (Records, Rows) This example is an instance of the students relation, which consists 4 tuples and 5 fields. No two rows are identical.

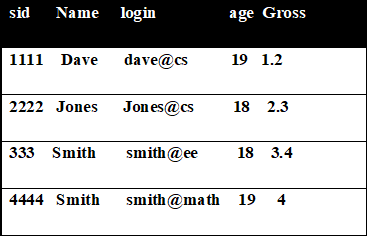


Figure 2.25: Relational database example.

**Degree:** The number of fields is called as ‘degree’. This is also called as ‘arity’. **Cardi- nality:** The cardinality of a relation instance is the number of tuples in it.

Example: In the above example, the degree of the relation is 5 and the cardinality is 4.

**Relational database:** It is a collection of relations with distinct relation names.

**Relational database schema:** It is the collection of schemas for the relations in the database. Instance: An instance of a relational database is a collection of relation in- stances, one per relation schema in the database schema. Each relation instance must satisfy the domain constraints in its schema.

##### Integrity constraints over relations

An integrity constraint (IC) is a condition that is specified on a database schema and restricts the data can be stored in an instance of the database. Various restrictions on data can be specified on a relational database schema in the form of ‘constraints’. A DBMS enforces integrity constraints, in that it permits only legal instances to be stored in the database. Integrity constraints are specified and enforced at different times as below.

* + - 1. When the DBA or end user defines a database schema, he or she specifies the ICs that must hold on any instance of this database.
      2. When a data base application is run, the DBMS checks for violations and disallows changes to the data that violate the specified ICs. Legal Instance: If the database instance satisfies all the integrity constraints specified on the database schema.

The constraints can be classified into 4 types as below.

1. Domain Constraints

Domain constraints are the most elementary form of integrity constraints. They are tested easily by the system whenever a new data item is entered into the database. Domain constraints specify the set of possible values that may be associated with an attribute. Such constraints may also prohibit the use of null values for particular attributes. The data types associated with domains typically include standard nu- meric data types for integers A relation schema specifies the domain of each field or column in the relation instance. These domain constraints in the schema specify an important condition that each instance of the relation to satisfy: The values that appear in a column must be drawn from the domain associated with that column. Thus the domain of a field is essentially the type of that field.

1. Key Constraints

A key constraint is a statement that a certain minimal subset of the fields of a relation is a unique identifier for a tuple.

Example: The ‘students’ relation and the constraint that no 2 students have the same student id (sid).

Different keys:

* + Candidate Key or Key
  + Super Key
  + Primary Key

1. Entity Integrity Constraints

This states that no primary key value can be null. The primary key value is used to identify individual tuples in a relation. Having null values for the primary key

implies that we cannot identify some tuples. NOTE: Key Constraints, Entity In- tegrity Constraints are specified on individual relations. PRIMARY KEYS comes under this.

1. Referential Integrity Constraints

The Referential Integrity Constraint is specified between 2 relations and is used to maintain the consistency among tuples of the 2 relations. Informally, the referential integrity constraint states that ‘a tuple in 1 relation that refers to another relation must refer to an existing tuple in that relation. We can diagrammatically display the referential integrity constraints by drawing a directed arc from each foreign key to the relation it references.

##### Keys in DBMS

Tables store a lot of data in them. Tables generally extend to thousands of records stored in them, unsorted and unorganized.Now to fetch any particular record from such dataset, you will have to apply some conditions, but what if there is duplicate data present and every time you try to fetch some data by applying certain condition, you get the wrong data. How many trials before you get the right data? To avoid all this, Keys are defined to easily identify any row of data in a table. Let’s try to understand about all the keys using a simple example.

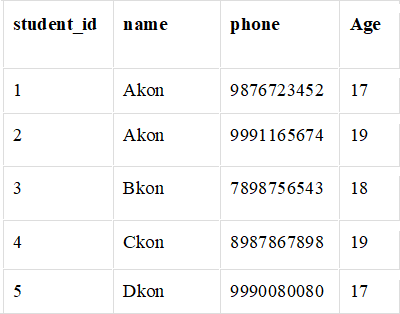


Figure 2.26: Keys in relational database example.

Super Key

Super Key is defined as a set of attributes within a table that can uniquely identify each record within a table. Super Key is a super set of Candidate key. In the table defined above super key would include studentid, (student id, name), phone etc.

The first one is pretty simple as student id is unique for every row of data, hence it can

be used to identity each row uniquely. Next comes, (student id, name), now name of two students can be same, but their student id can’t be same hence this combinati on can also be a key. Similarly, phone number for every student will be unique, hence again, phone can also be a key. So they all are super keys.

Candidate Key

Candidate keys are defined as the minimal set of fields which can uniquely identify each record in a table. It is an attribute or a set of attributes that can act as a Primary Key for a table to uniquely identify each record in that table. There can be more than one candidate key. In our example, student id and phone both are candidate keys for table Student.

* A candidate key can never be NULL or empty. And its value should be unique.
* There can be more than one candidate keys for a table.
* A candidate key can be a combination of more than one columns(attributes).

Primary Key

Primary key is a candidate key that is most appropriate to become the main key for any table. It is a key that can uniquely identify each record in a table.

For the table Student we can make the student 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. But the attributes which together form the Composite key are not a key independently or individually.

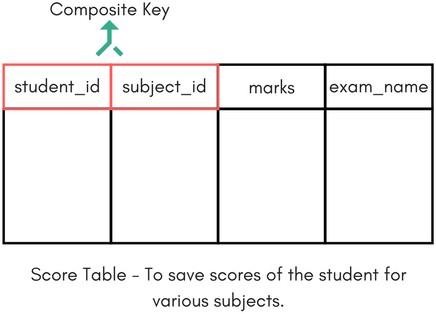


Figure 2.27: Composite keys in relational database example.

In the Figure. [2.27](#_bookmark8) we have a Score table which stores the marks scored by a student in a particular subject. In this table student id and subject id together will form the primary key, hence it is a composite key.

Secondary or Alternative key

The candidate key which are not selected as primary key are known as secondary keys or alternative keys.

## E-R Diagram to Table Conversion

E-R diagram can be converted to a table by following the rules given below:

1. Strong Entity Set With Only Simple Attributes
   * A strong entity set with only simple attributes will require only one table in relational model.
   * Attributes of the table will be the attributes of the entity set.
   * The primary key of the table will be the key attribute of the entity set.

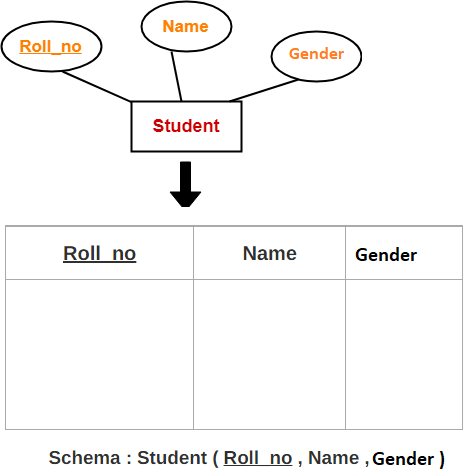


Figure 2.28: E-R diagram to Table - Rule 1.

1. Strong Entity Set With Composite Attributes
   * A strong entity set with any number of composite attributes will require only one table in relational model.
   * While conversion, simple attributes of the composite attributes are taken into account and not the composite attribute itself.

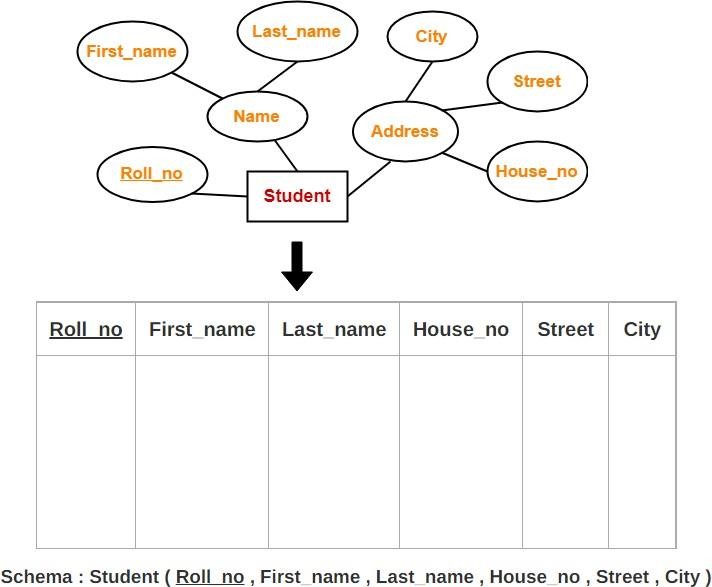


Figure 2.29: E-R diagram to Table - Rule 2.

1. Strong Entity Set With Multi Valued Attributes
   * A strong entity set with any number of multi valued attributes will require two tables in relational model.
   * One table will contain all the simple attributes with the primary key.
   * Other table will contain the primary key and all the multi valued attributes.

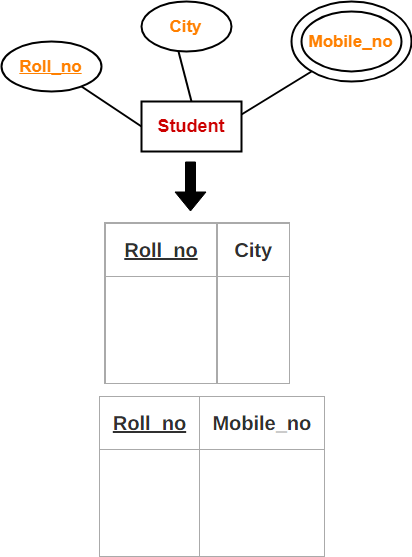


Figure 2.30: E-R diagram to Table - Rule 3.

1. Translating Relationship Set into a Table
   * A relationship set will require one table in the relational model.
   * Attributes of the table are:
2. Primary key attributes of the participating entity sets
3. Its own descriptive attributes if any.
4. Set of non-descriptive attributes will be the primary key.

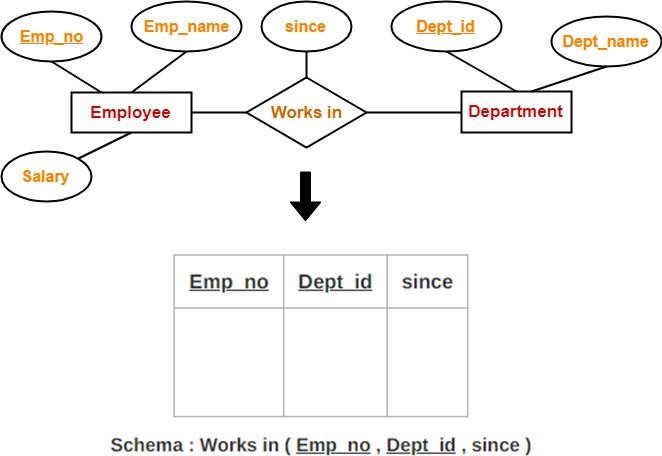


Figure 2.31: E-R diagram to Table - Rule 4.

1. Binary Relationships With Cardinality Ratios
   * **Case-01:** Binary relationship with cardinality ratio m:n
     + E1, R, E2 – 3 tables

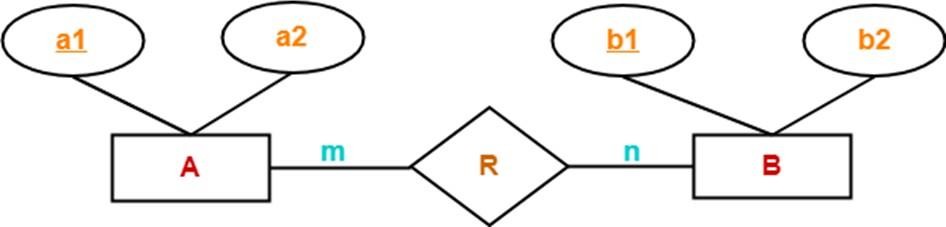


Figure 2.32: E-R diagram to Table - Rule 5(a).

This E-R diagram is translated into three tables

1. A ( a1 , a2 )
2. R ( a1 , b1 )
3. B ( b1 , b2 )
   * **Case-02:** Binary relationship with cardinality ratio 1:n
     + E1, E2R - 2 tables

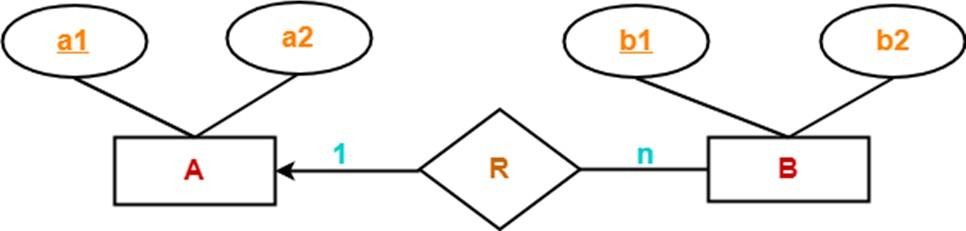


Figure 2.33: E-R diagram to Table - Rule 5(b).

This E-R diagram is translated into two tables

1. A ( a1 , a2 )
2. BR ( a1 , b1 , b2 )
   * **Case-03:** Binary relationship with cardinality ratio m:1
     + E1R, E2 - 2 tables

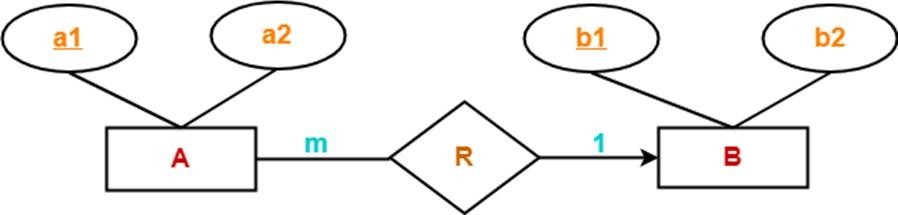


Figure 2.34: E-R diagram to Table - Rule 5(c).

This E-R diagram translates to two tables

1. AR ( a1 , a2 , b1 )
2. B ( b1 , b2 )
   * **Case-04:** Binary relationship with cardinality ratio 1:1
     + E1R, E2 orE1, E2R - 2 tables

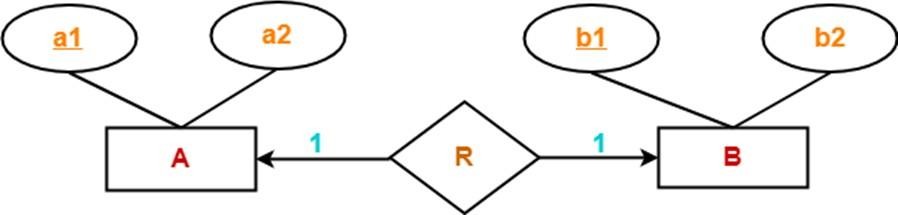


Figure 2.35: E-R diagram to Table - Rule 5(d).

This E-R diagram translates to two tables

1. AR ( a1 , a2 , b1 )
2. B ( b1 , b2 )

(OR)

1. A ( a1 , a2 )
2. BR ( a1 , b1 , b2 )
3. Binary Relationship With Both Cardinality Constraints and Participa- tion Constraints
   * Cardinality constraints will be implemented as discussed in Rule-05.
   * Total participation constraint ⇒ foreign key acquires NOT NULL constraint

i.e. now foreign key can not be null.

* + **Case-01:** Binary Relationship With Cardinality Constraint and Total Partic- ipation Constraint From One Side

E1, E2R – with a foreign key constraint

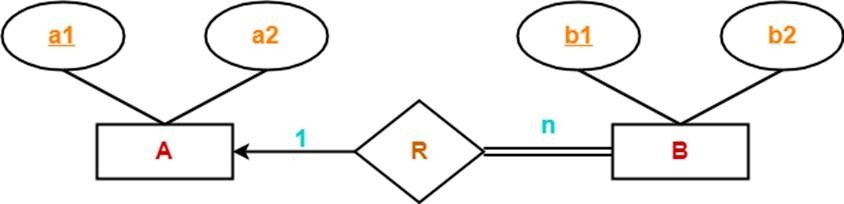


Figure 2.36: E-R diagram to Table - Rule 6(a).

This E-R diagram translates to Two tables.

1. A ( a1 , a2 )
2. BR ( a1 , b1 , b2 )
   * **Case-02:** Binary Relationship With Cardinality Constraint and Total Partic- ipation Constraint From Both Sides

E1R E2 – One table

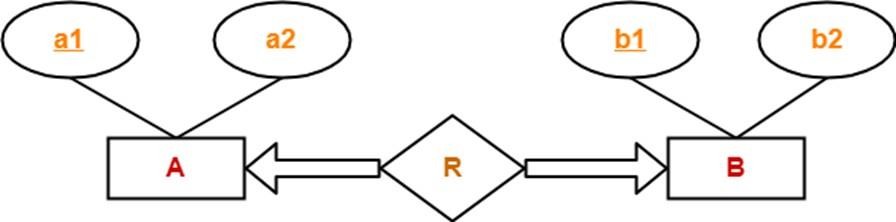


Figure 2.37: E-R diagram to Table - Rule 6(b).

This E-R diagram translates to One tables.

(a) ARB ( a1 , a2 , b1 , b2 )

1. **Binary Relationship With Weak Entity Set** Weak entity set always appears in association with identifying relationship with total participation constraint.
   * E1, E2R – Two tables

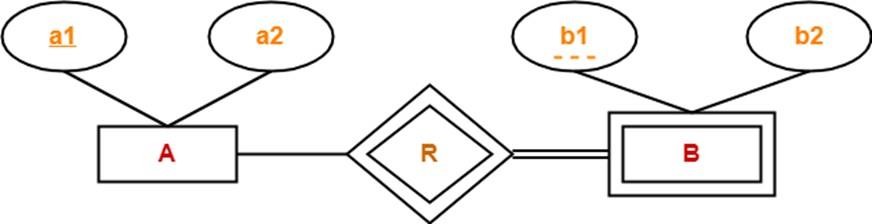


Figure 2.38: E-R diagram to Table - Rule 7.

This E-R diagram translates to Two tables.

1. A ( a1 , a2 )
2. BR ( a1 , b1 , b2 )

Example: E-R diagram to Table Conversion

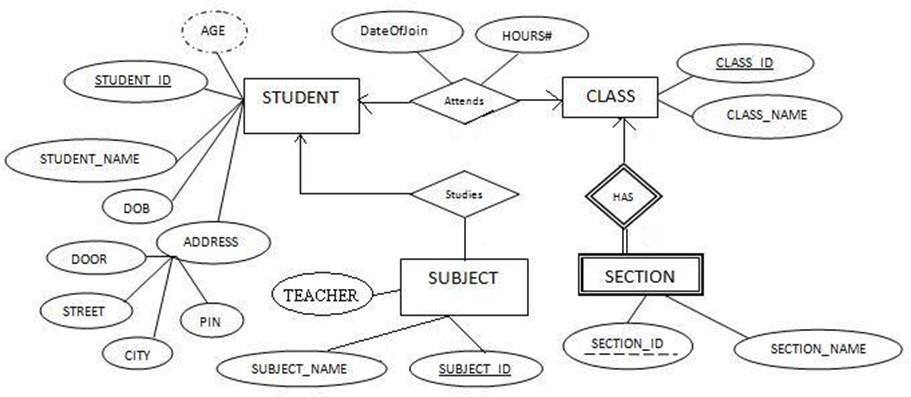


Figure 2.39: E-R diagram to Table - Example problem.

The final database contains the following tables:

1. STUDENT (StudentID, StudentName, DOB, Door, Street, City, Pin)
2. CLASS (ClassID, ClassName, StudentID, DateOfJoin, Hours)
   * StudentID is the foreign key refers STUDENT table
3. SUBJECT (SubjectID, SubjectName, Teacher, StudentID)
   * StudentID is the foreign key refers STUDENT table
4. SECTION (SectionID, ClassID, SectionName)
   * ClassID is the foreign key refers CLASS table

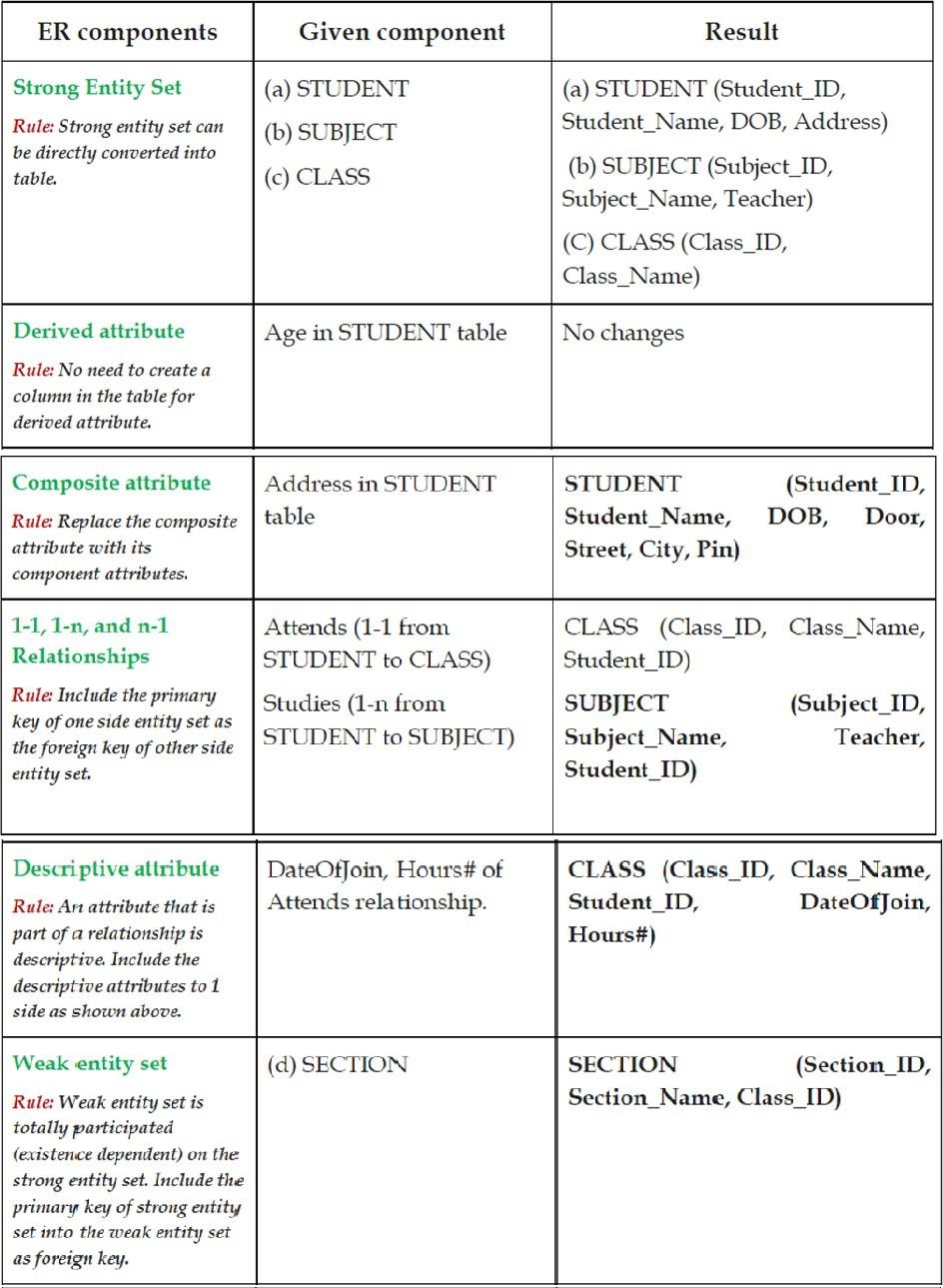


Figure 2.40: E-R diagram to Table - Example problem Solution.

* + 1. **Views (Virtual Table)**

A view is a virtual table based on the result-set of an SQL statement. A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database. We can create a view by selecting fields from one or more tables present in the database. A View can either have all the rows of a table or specific rows based on certain condition. Views are a logical virtual table created by “select query” but the result is not stored anywhere in the disk and every time we need to fire the query when we need data, so always we get updated or latest data from original tables.

* + - 1. A view is a predefined query on one or more tables.
      2. Retrieving information from a view is done in the same manner as retrieving from a table.
      3. With some views you can also perform DML operations (delete, insert, update) on the base tables.
      4. Views don’t store data, they only access rows in the base tables.
      5. user tables, user sequences, and user indexes are all views.
      6. view can hide the underlying base tables.
      7. By writing complex queries as a view, we can hide complexity from an end user.
      8. View only allows a user to access certain rows in the base tables.

Creating Views

A View can be created from a single table or multiple tables.

CREATE VIEW view \_ name

AS SELECT column 1 , column 2 ,.. columnn FROM table \_ name WHERE condition ;

Listing 2.1: Creating views

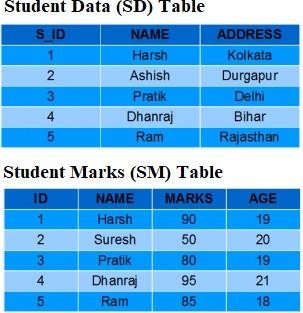


Figure 2.41: Creating views.

CREATE VIEW Details AS SELECT NAME , ADDRESS FROM SD

WHERE S\_ID < 5;

Listing 2.2: Creating view from a single table

To see the data in the View, we can query the view in the same manner as we query a table. The output is shown in Figure. [2.42.](#_bookmark9)

SELECT \* FROM Details ;

Listing 2.3: See the data in views

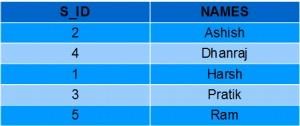


Figure 2.42: Output of one table view.

Creating View from multiple tables

CREATE VIEW Marks AS

SELECT SD. NAME , SD. ADDRESS , SM. MARKS FROM SD , SM WHERE SD. NAME = SM. NAME ;

Listing 2.4: Creating view from a multiple tables

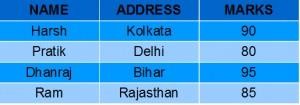


Figure 2.43: Output of multi table view.

Updating Views

A view can be updated under certain conditions which are given below:

* The SELECT clause may not contain the keyword DISTINCT.
* The SELECT clause may not contain summary functions.
* The SELECT clause may not contain set functions.
* The SELECT clause may not contain set operators.
* The SELECT clause may not contain an ORDER BY clause
* The query may not contain GROUP BY or HAVING.
* Calculated columns may not be updated.

We can use the CREATE OR REPLACE VIEW statement to add or remove fields from a view.

Create OR Replace View

view \_ name AS SELECT column 1 , coulmn 2 ,.. FROM table \_ name

WHERE condition ;

Listing 2.5: Create or replace view

For example, if we want to update the view Marks and add the field AGE to this View from SM Table, we can do this as:

CREATE OR REPLACE VIEW Marks

AS SELECT SD. NAME , SD. ADDRESS , SM. MARKS , SM. AGE FROM SD , SM WHERE SD. NAME = SM. NAME ;

Listing 2.6: Create or replace view

INSERTING a row

INSERT INTO View name ( C 1 , C 2... Cn) VALUES ( V 1 , V 2 . Vn);

INSERT INTO Details ( NAME , ADDRESS ) VALUES (" Suresh "," Gurgaon ");

Listing 2.7: Inserting a row

DELETING Rows

Deleting rows from a view is also as simple as deleting rows from a table. We can use the DELETE statement of SQL to delete rows from a view. Also deleting a row from a view first delete the row from the actual table and the change is then reflected in the view.

DELETE FROM view \_ name WHERE condition ; DELETE FROM Details WHERE NAME =" Suresh ";

Listing 2.8: Deleting a row

DELETING Views

We can delete or drop a View using the DROP statement.

DROP VIEW view \_ name ;

Listing 2.9: Deleting a row

Materialized View

Materialized views are also the logical view of our data-driven by the select query but the result of the query will get stored in the table or disk.

## Relational Algebra

Relational Algebra is procedural query language, which takes Relation as input and gen - erate relation as output. Relational algebra mainly provides theoretical foundation for relational databases and SQL. Each relational query describes a step-by-step procedure for computing the desired answer, based on the order in which operators are applied in the query.

##### Importance of Relational Algebra

First, it provides a formal foundation for relational model operations.

Second, and perhaps more important, it is used as a basis for implementing and optimiz - ing queries in the query processing and optimization modules that are integral parts of relational database management systems (RDBMS).

Third, some of its concepts are incorporated into the SQL standard query language for RDBMS.

The fundamental operation included in relational algebra are Select (*σ*), Project (*π*), Union (∪ ), Set Difference (−), Cartesian product (×) and Rename (*ρ*)

##### Relational Calculus

Unlike Relational Algebra, Relational Calculus is a higher level Declarative language. In converse to the relational algebra, relational calculus defines what result is to be obtained. Like Relational Algebra, Relational Calculus does not specify the sequence of operations in which query will be evaluated.

The sequence of relational calculus operations is called relational calculus expression that also produces a new relation as a result. The Relational Calculus has two variations namely

* + - 1. Tuple Relational Calculus and
      2. Domain Relational Calculus.

The Tuple Relational Calculus list the tuples to selected from a relation, based on a certain condition provided. It is formally denoted as:

{*t*|*Pt*} (2.1)

Where *t* is the set of tuples for which the condition *P* is true.

The next variation is Domain Relational Calculus, which in contrast to Tuple Relational Calculus list the attributes to be selected from a relation, based on certain condition. The formal definition of Domain Relational Calculus is as follows:

{*< X*1*, X*2*, X*3*, ...Xn >* |*PX*1*, X*2*, X*3*, ...Xn*} (2.2)

Where *X*1*, X*2*, X*3*, ...Xn* are the attributes and *P* is the certain condition.

Key Differences Between Relational Algebra and Relational Calculus

* The basic difference between Relational Algebra and Relational Calculus is that Relational Algebra is a Procedural language whereas, the Relational Calculus is a Non-Procedural, instead it is a Declarative language.
* The Relational Algebra defines how to obtain the result whereas, the Relational Calculus define what information the result must contain.
* Relational Algebra specifies the sequence in which operations have to be performed in the query. On the other hands, Relational calculus does not specify the sequence of operations to performed in the query.
* The Relational Algebra query language is closely related to programming language whereas; the Relational Calculus is closely related to the Natural Language.

Relational Algebra and Relational Calculus both have equivalent expressive power. The main difference between them is just that Relational Algebra specify how to retrieve data and Relational Calculus defines what data is to be retrieved

Relational algebra is a procedural query language, which takes instances of relations as input and yields instances of relations as output. It uses operators to perform queries. An operator can be either unary or binary. They accept relations as their input and yield relations as their output. Relational algebra is performed recursively on a relation and intermediate results are also considered relations. The fundamental operations of relational algebra are as follows:

* Select
* Project
* Union
* Set different
* Cartesian product
* Rename

In addition to the fundamental operations, there are several other operations—namely, set intersection, natural join, division, and assignment.We will define these operations in terms of the fundamental operations.

##### Select Operation (*σ*)

It selects tuples that satisfy the given predicate from a relation.

Notation:

*σp*(*r*) (2.3)

Where *σ* stands for selection predicate and *r* stands for relation. *p* is prepositional logic formula which may use connectors like and, or, and not. These terms may use relational

operators like =*,* =/ *,* ≥*, <, >,* ≤.

To select those tuples of the loan relation where the branch is “Perryridge,” we write

*σbranch−−name* = *Perryridge*(*loan*)

We can find all tuples in which the amount lent is more than $ 1200 by writing

*σamount >* 1200(*loan*)

To find those tuples pertaining to loans of more than $ 1200 made by the Perryridge branch, we write

*σbranch−name* = *Perryridge* ∧ *amount >* 1200(*loan*)

##### Project Operation (Π)

It projects column(s) that satisfy a given predicate.

Notation:

Π*A*1*, A*2*, An*(*r*) (2.4)

Where *A*1, *A*2 , *An* are attribute names of relation *r*. Duplicate rows are automatically eliminated, as relation is a set.

To list all loan numbers and the amount of the loan:

Π*loan−number,amount*(*loan*) (2.5)

Composition of Relational Operations

To “Find those customers who live in Harrison.” We write:

Π*customer−name*(*σcustomer−city* = *Harrison*(*customer*)) (2.6)

We can define the three operations UNION, INTERSECTION, and SET DIFFERENCE on two union-compatible relations *R* and *S* as follows:

* UNION:

The result of this operation, denoted by

*R* ∪ *S*

, is a relation that includes all tuples that are either in *R* or in *S* or in both *R* and

*S*. Duplicate tuples are eliminated.

* INTERSECTION:

The result of this operation, denoted by

*R* ∩ *S*

, is a relation that includes all tuples that are in both *R* and *S*.

* SET DIFFERENCE (or MINUS):

The result of this operation, denoted by

*R*˘*S*

, is a relation that includes all tuples that are in *R* but not in *S*.

##### The Union Operation

Consider a query to find the names of all bank customers who have either an account or a loan or both.

Π*customer−name*(*borrower*) ∪ Π*customer−name*(*depositor*) (2.7) For a union operation *R* ∪ *S* to be valid, we require that two conditions hold:

* + - 1. The relations r and s must be of the same arity. That is, they must have the same number of attributes.
      2. The domains of the *ith* attribute of *r* and the *ith* attribute of *s* must be the same, for all *i*.

##### The Set Difference Operation

Notation :

*r* − *s*

The set-difference operation, denoted by , allows us to find tuples that are in one relation but are not in another. The expression *r*−*s* produces a relation containing those tuples

in *r* but not in *s*.

To find all customers of the bank who have an account but not a loan

Π*customer−name*(*depositor*)Π*customer−name*(*borrower*) (2.8)

As with the union operation, we must ensure that set differences are taken between compatible relations.

* Tuple relational calculus which was originally proposed by Codd in the year 1972 and
* Domain relational calculus which was proposed by Lacroix and Pirotte in the year 1977

In first-order logic or predicate calculus, a predicate is a truth-valued function with arguments. When we replace with values for the arguments, the function yields an expression, called a proposition, which will be either true or false.

Example:

For example, steps involved in listing all the employees who attend the ’Networking’ Course would be:

SELECT the tuples from COURSE

relation with COURSENAME = ' NETWORKING ' PROJECT the COURSE \_ID from above result

Listing 2.10: Example

SELECT the tuples from EMP relation with COURSE*IDresultedabove.*

Tuple Relational Calculus

In the tuple relational calculus, you will have to find tuples for which a predicate is true. The calculus is dependent on the use of tuple variables. A tuple variable is a variable that ’ranges over’ a named relation: i.e., a variable whose only permitted values are tuples of the relation. **Domain Relational calculus:**

In the tuple relational calculus, you have use variables that have a series of tuples in a relation. In the domain relational calculus, you will also use variables, but in this case, the variables take their values from domains of attributes rather than tuples of relations. A domain relational calculus expression has the following general format:

*d*1*, d*2*, ..., dn*|*F* (*d*1*, d*2*, ..., dm*)*m* ≥ *n* (2.9)

where *d*1*, d*2*, ..., dn, ..., dm* stand for domain variables and *F* (*d*1*, d*2*, ..., dm*) stands for a formula composed of atoms.