

UNIT -2

SYLLABUS

UNIT-II:

- The E/R Models
- The Relational Model
- Relational Calculus
- Introduction to Database Design
- Database Design and E/R Diagrams
- Entities Attributes and Entity Sets-Relationship and Relationship Sets
- Conceptual Design with the E/R Models
- The Relational Model Integrity
- Constraints over Relations
- Key Constraints
- Foreign Key Constraints
- General Constraints
- Relational Algebra and Calculus
- Relational Algebra
- Selection and Projection
- Set Operation
- Renaming
- Joins
- Division
- More Examples of Queries
- Relational Calculus
- Tuple Relational Calculus
- Domain Relational Calculus

ER Model - Basic Concepts:-

The ER model defines the conceptual view of a database. It works around real-world entities and the associations among them. At view level, the ER model is considered a good option for designing databases.

Entity

An entity can be a real-world object, either animate or inanimate, that can be easily identifiable. For example, in a school database, students, teachers, classes, and courses offered can be considered as entities. All these entities have some attributes or properties that give them their identity.

An entity set is a collection of similar types of entities. An entity set may contain entities with attribute sharing similar values. For example, a Students set may contain all the students of a school; likewise a Teachers set may contain all the teachers of a school from all faculties. Entity sets need not be disjoint.

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.

There exists a domain or range of values that can be assigned to attributes. For example, a student's name cannot be a numeric value. It has to be alphabetic. A student's age cannot be negative, etc.

Types of Attributes

- **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.
- **Composite attribute** – Composite attributes are made of more than one simple attribute. For example, a student's complete name may have first_name and last_name.
- **Derived attribute** – Derived attributes are the attributes that do not exist in the physical database, but their values are derived from other attributes present in the database. For example, average_salary in a department should not be saved directly in the database, instead it can be derived. For another example, age can be derived from data_of_birth.

- **Single-value attribute** – Single-value attributes contain single value. For example – Social_Security_Number.
- **Multi-value attribute** – Multi-value attributes may contain more than one values. For example, a person can have more than one phone number, email_address, etc.

These attribute types can come together in a way like –

- simple single-valued attributes
- simple multi-valued attributes
- composite single-valued attributes
- composite multi-valued attributes

Entity-Set and Keys

Key is an attribute or collection of attributes that uniquely identifies an entity among entity set.

For example, the roll_number of a student makes him/her identifiable among students.

- **Super Key** – A set of attributes (one or more) that collectively identifies an entity in an entity set.
- **Candidate Key** – minimal super key is called a candidate key. An entity set may have more than one candidate key.
- **Primary Key** – A primary key is one of the candidate keys chosen by the database designer to uniquely identify the entity set.

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 relationships.

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 attributes**.

Degree of Relationship

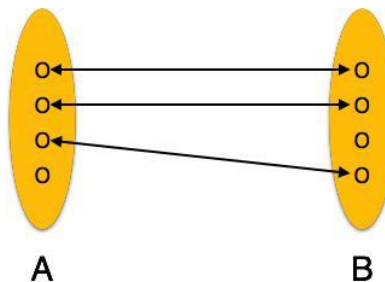
The number of participating entities in a relationship defines the degree of the relationship.

- Binary = degree 2
- Ternary = degree 3
- n-ary = degree

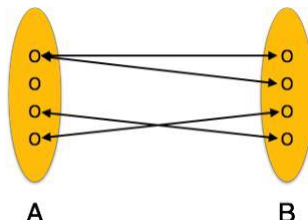
Mapping Cardinalities

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.

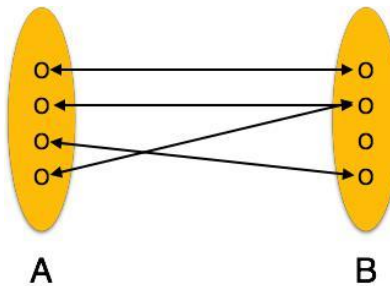
- **One-to-one** – One entity from entity set A can be associated with at most one entity of entity set B and vice versa.



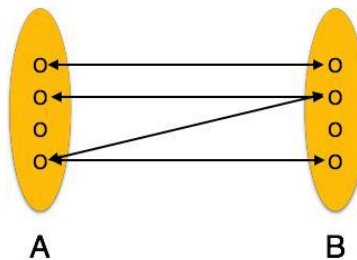
- **One-to-many** – One entity from entity set A can be associated with more than one entities of entity set B however an entity from entity set B, can be associated with at most one entity.



- **Many-to-one** – More than one entities from entity set A can be associated with at most one entity of entity set B, however an entity from entity set B can be associated with more than one entity from entity set A.



- **Many-to-many** – One entity from A can be associated with more than one entity from B and vice versa.



Relational Model:-

The most popular data model in DBMS is the Relational Model. It is more scientific a model than others. This model is based on first-order predicate logic and defines a table as an **n-ary relation**.

attributes

column

SID	SName	SAge	SClass	SSection
1101	Alex	14	9	A
1102	Maria	15	9	A
1103	Maya	14	10	B
1104	Bob	14	9	A
1105	Newton	15	10	B

tuple

table (relation)

The main highlights of this model are –

- Data is stored in tables called **relations**.
- Relations can be normalized.
- In normalized relations, values saved are atomic values.
- Each row in a relation contains a unique value.
- Each column in a relation contains values from a same domain.

Relational Calculus

In contrast to Relational Algebra, Relational Calculus is a non-procedural query language, that is, it tells what to do but never explains how to do it.

Relational calculus exists in two forms –

Tuple Relational Calculus (TRC)

Filtering variable ranges over tuples

Notation – $\{T \mid \text{Condition}\}$

Returns all tuples T that satisfies a condition.

For example –

```
{ T.name | Author(T) AND T.article = 'database' }
```

Output – Returns tuples with 'name' from Author who has written article on 'database'.

TRC can be quantified. We can use Existential ($\exists$) and Universal Quantifiers ($\forall$).

For example –

```
{ R |  $\exists T \in \text{Authors}(T.\text{article} = \text{'database' AND } R.\text{name} = T.\text{name})$  }
```

Output – The above query will yield the same result as the previous one.

Domain Relational Calculus (DRC)

In DRC, the filtering variable uses the domain of attributes instead of entire tuple values (as done in TRC, mentioned above).

Notation –

$\{ a_1, a_2, a_3, \dots, a_n \mid P(a_1, a_2, a_3, \dots, a_n) \}$

Where a_1, a_2 are attributes and P stands for formulae built by inner attributes.

For example –

```
{< article, page, subject > |  $\_ \in \text{TutorialsPoint} \wedge \text{subject} = \text{'database'}$ }
```

Output – Yields Article, Page, and Subject from the relation TutorialsPoint, where subject is database.

Just like TRC, DRC can also be written using existential and universal quantifiers. DRC also involves relational operators.

The expression power of Tuple Relation Calculus and Domain Relation Calculus is equivalent to Relational Algebra.

Database Design:

Introduction to Database Design:-

Database design is the design of the database structure that will be used to store and manage data rather than the design of the DBMS software. Once the database design is completed, the DBMS handles all the complicated activities required to translate the designer's view of the structures into structures that are usable to the computer.

A poorly designed database tends to generate errors that are likely to lead to bad decisions. A bad database design eventually can be self correcting: organizations using poorly designed databases often fail because their managers do not have access to timely (or even correct) information, thereby dominating the bad database design.

The availability of a DBMS makes it possible to tackle far more sophisticated uses of the data resources, if the database is designed to make use of that available power. The kinds of data structures created within the database and the extent of the relationships among them play a powerful role in determining how effective the DBMS is. Therefore, database design become a crucial activity in the database environment.

Database design is made much simpler when we use **models**. A Database model is a collection of a logical constructs used to represent the data structure and the data relationships found within the database i.e. simplified abstractions of real-world events or conditions. If the models are not logically sound, the database designs derived from them will not deliver the database system's promise to effective information drawn from an efficient database. *"Good models yield good database design that are the basis for good applications"*.

Goals of Database Design :

Database Design normally involves defining the logical attributes of the database designing the layout of the database file structure. The main objectives of database design is

1. To satisfy the information content requirement of the specified user and application.
2. To provide a natural and easy way to understand structuring of the information.
3. To support processing requirements and any performance objectives such as

- i. Response time
- ii. Processing time
- iii. Storage space

The main objective of the database design is to ensure that the database meets the reporting and information requirements of the users efficiently. The database should be designed in such a way that :

- i. It eliminates or minimizes data redundancy.
- ii. Maintains the integrity and independence of the data.

Phases in Database Design :

First phase: The overall purpose of the database initial study is to

- a. analyze the organization/system situation
- b. define problem and constraints
- c. define objectives
- d. define scope and boundaries.

Second phase : The second phase focuses on the design of the database model that will support organization operations and objectives.

In this phase, we can identify six main phases of the database design :

I. Requirements collection and analysis

. Conceptual database design

. Choice of DBMS

IV. Data model mapping

V. Physical database design

VI. Database system implement

E-R Diagrams:-

ER Diagram Representation

Let us now learn how the ER Model is represented by means of an ER diagram. Any object, for example, entities, attributes of an entity, relationship sets, and attributes of relationship sets, can be represented with the help of an ER diagram.

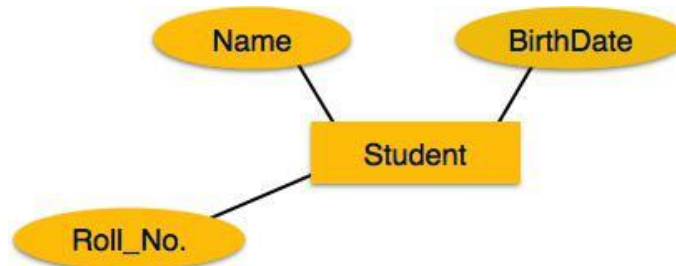
Entity

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

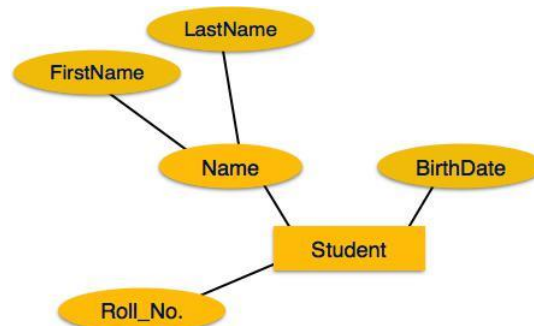


Attributes

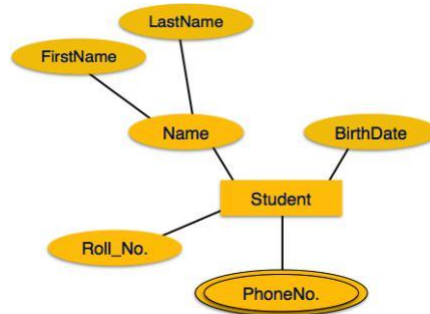
Attributes are the properties of entities. Attributes are represented by means of ellipses. Every ellipse represents one attribute and is directly connected to its entity (rectangle).



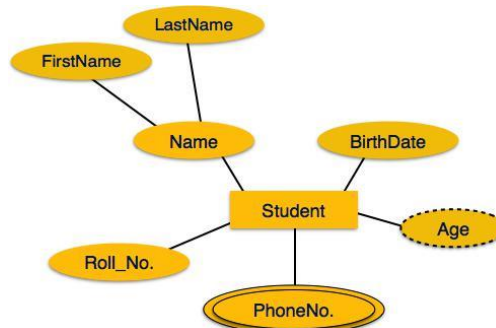
If the attributes are **composite**, they are further divided in a tree like structure. Every node is then connected to its attribute. That is, composite attributes are represented by ellipses that are connected with an ellipse.



Multivalued attributes are depicted by double ellipse.



Derived attributes are depicted by dashed ellipse.



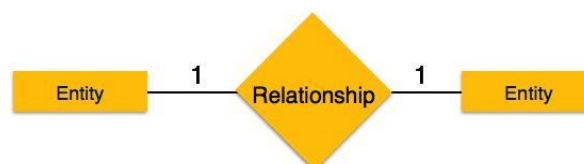
Relationship

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.

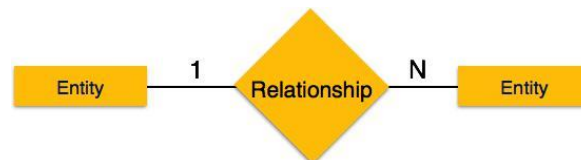
Binary Relationship and Cardinality

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

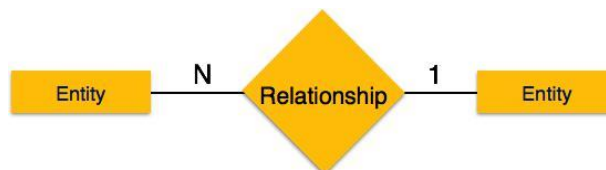
- **One-to-one** – When only one instance of an entity is associated with the relationship, it is marked as '1:1'. The following image reflects that only one instance of each entity should be associated with the relationship. It depicts one-to-one relationship.



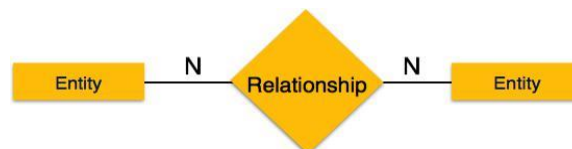
- **One-to-many** – When more than one instance of an entity is associated with a relationship, it is marked as '1:N'. The following image reflects that only one instance of entity on the left and more than one instance of an entity on the right can be associated with the relationship. It depicts one-to-many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



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

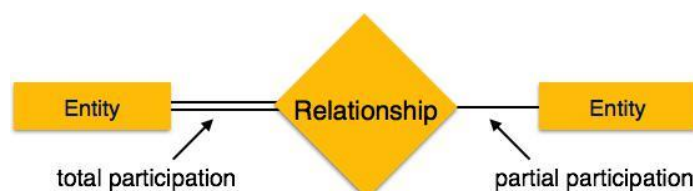


- **Many-to-many** – The following image reflects that more than one instance of an entity on the left and more than one instance of an entity on the right can be associated with the relationship. It depicts many-to-many relationship.



Participation Constraints

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

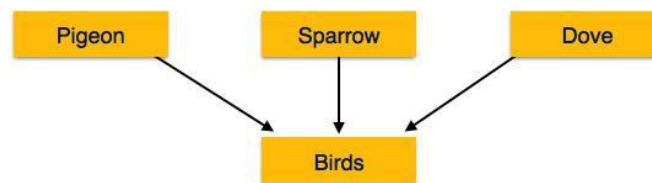


The ER Model has the power of expressing database entities in a conceptual hierarchical manner. As the hierarchy goes up, it generalizes the view of entities, and as we go deep in the hierarchy, it gives us the detail of every entity included.

Going up in this structure is called **generalization**, where entities are clubbed together to represent a more generalized view. For example, a particular student named Mira can be generalized along with all the students. The entity shall be a student, and further, the student is a person. The reverse is called **specialization** where a person is a student, and that student is Mira.

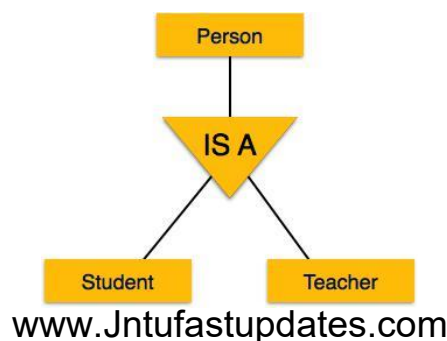
Generalization

As mentioned above, the process of generalizing entities, where the generalized entities contain the properties of all the generalized entities, is called generalization. In generalization, a number of entities are brought together into one generalized entity based on their similar characteristics. For example, pigeon, house sparrow, crow and dove can all be generalized as Birds.



Specialization

Specialization is the opposite of generalization. In specialization, a group of entities is divided into sub-groups based on their characteristics. Take a group 'Person' for example. A person has name, date of birth, gender, etc. These properties are common in all persons, human beings. But in a company, persons can be identified as employee, employer, customer, or vendor, based on what role they play in the company.



Similarly, in a school database, persons can be specialized as teacher, student, or a staff, based on what role they play in school as entities.

Conceptual Design with E-R Model:-

- v Requirement analysis
 - Data to be stored
 - Applications to be built
 - Operations (most frequent) subject to performance requirement
- v Conceptual db design
 - Description of the data (including constraints)
 - By high level model such as ER
- v Logical db design
 - Choose DBMS to implement
 - Convert conceptual db design into database schema

Issues to consider: (*ER Model is used at this stage.*)

- What are the *entities* and *relationships* in the enterprise?
- What information about these entities and relationships should we store in the database (i.e., attributes)?
- What are the *integrity constraints* or *business rules* that hold?

Solution:

- A database 'schema' in the ER Model can be represented pictorially (*ER diagrams*).
- Can map an ER diagram into a relational schema.

The Relational Model Integrity: Constraints Over Relations

Constraints:-

Every relation has some conditions that must hold for it to be a valid relation. These conditions are called **Relational Integrity Constraints**. There are three main integrity constraints –

- Key constraints
- Domain constraints
- Referential integrity constraints

Key Constraints:-

There must be at least one minimal subset of attributes in the relation, which can identify a tuple uniquely. This minimal subset of attributes is called **key** for that relation. If there are more than one such minimal subsets, these are called **candidate keys**.

Key constraints force that –

- in a relation with a key attribute, no two tuples can have identical values for key attributes.
- a key attribute can not have NULL values.

Key constraints are also referred to as Entity Constraints.

Domain Constraints: -

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.

Referential integrity Constraints:-

Referential integrity constraints work on the concept of Foreign Keys. A foreign key is a key attribute of a relation that can be referred in other relation.

Referential integrity constraint states that if a relation refers to a key attribute of a different or same relation, then that key element must exist.

FOREIGN KEY Constraint:-

A FOREIGN KEY is a key used to link two tables together.

A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.

The table containing the foreign key is called the child table, and the table containing the candidate key is called the referenced or parent table.

Look at the following two tables:

Look at the following two tables:

"Persons" table:

PersonID	LastName	FirstName	Age
1	Hansen	Ola	30
2	Svendson	Tove	23
3	Pettersen	Kari	20

"Orders" table:

OrderID	OrderNumber	PersonID
1	77895	3
2	44678	3

3	22456	2
4	24562	1

Notice that the "PersonID" column in the "Orders" table points to the "PersonID" column in the "Persons" table.

The "PersonID" column in the "Persons" table is the PRIMARY KEY in the "Persons" table.

The "PersonID" column in the "Orders" table is a FOREIGN KEY in the "Orders" table.

The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

The FOREIGN KEY constraint also prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the table it points to.

FOREIGN KEY on CREATE TABLE:-

The following SQL creates a FOREIGN KEY on the "PersonID" column when the "Orders" table is created:

MySQL:

```
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```

Relational Algebra and Calculus:-

Introduction:-

Relational database systems are expected to be equipped with a query language that can assist its users to query the database instances. There are two kinds of query languages – relational algebra and relational calculus.

Relational Algebra

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

We will discuss all these operations in the following sections.

Select Operation (σ)

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

Notation – $\sigma_p(r)$

Where σ stands for selection predicate and r stands for relation. p is propositional logic formula which may use connectors like **and**, **or**, and **not**. These terms may use relational operators like $=$, $\neq$, $\geq$, $<$, $>$, $\leq$.

For example –

```
 $\sigma_{\text{subject} = \text{"database"}}(\text{Books})$ 
```

Output – Selects tuples from books where subject is 'database'.

```
 $\sigma_{\text{subject} = \text{"database"} \text{ and } \text{price} = \text{"450"}}(\text{Books})$ 
```

Output – Selects tuples from books where subject is 'database' and 'price' is 450.

$\sigma_{\text{subject} = \text{"database"} \text{ and price} = \text{"450"} \text{ or year} > \text{"2010"} (\text{Books})$

Output – Selects tuples from books where subject is 'database' and 'price' is 450 or those books published after 2010.

Project Operation (Π)

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

Notation – $\Pi_{A_1, A_2, A_n} (r)$

Where A_1, A_2, A_n are attribute names of relation **r**.

Duplicate rows are automatically eliminated, as relation is a set.

For example –

$\Pi_{\text{subject, author}} (\text{Books})$

Selects and projects columns named as subject and author from the relation Books.

Union Operation ($\cup$)

It performs binary union between two given relations and is defined as –

$r \cup s = \{ t \mid t \in r \text{ or } t \in s \}$

Notation – $r \cup s$

Where **r** and **s** are either database relations or relation result set (temporary relation).

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

- **r**, and **s** must have the same number of attributes.
- Attribute domains must be compatible.

- Duplicate tuples are automatically eliminated.

$\Pi_{\text{author}} (\text{Books}) \cup \Pi_{\text{author}} (\text{Articles})$

Output – Projects the names of the authors who have either written a book or an article or both.

Set Difference (–)

The result of set difference operation is tuples, which are present in one relation but are not in the second relation.

Notation – $r - s$

Finds all the tuples that are present in **r** but not in **s**.

```
 $\Pi_{\text{author}}(\text{Books}) - \Pi_{\text{author}}(\text{Articles})$ 
```

Output – Provides the name of authors who have written books but not articles.

Cartesian Product (X)

Combines information of two different relations into one.

Notation – $r \times s$

Where **r** and **s** are relations and their output will be defined as –
 $r \times s = \{ q \mid q \in r \text{ and } q \in s \}$

```
 $\sigma_{\text{author} = \text{'tutorialspoint'}}(\text{Books} \times \text{Articles})$ 
```

Output – Yields a relation, which shows all the books and articles written by tutorialspoint.

Rename Operation (ρ)

The results of relational algebra are also relations but without any name. The rename operation allows us to rename the output relation. 'rename' operation is denoted with small Greek letter **ρ** .

Notation – $\rho_x(E)$

Where the result of expression **E** is saved with name of **x**.

Additional operations are –

- Set intersection
- Assignment
- Natural join

Joins:-

Join is a combination of a Cartesian product followed by a selection process. A Join operation pairs two tuples from different relations, if and only if a given join condition is satisfied.

Theta (θ) Join

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 $(A_1, A_2, \dots, A_n)$ and $(B_1, B_2, \dots, B_n)$ such that the attributes don't have anything in common, that is $R_1 \cap R_2 = \Phi$.

Theta join can use all kinds of comparison operators.

		Student	
SID	Name		Std
101	Alex		10
102	Maria		11
		Subjects	
Class		Subject	
10		Math	
10		English	
11		Music	

11	Sports
----	--------

Student_Detail –

STUDENT ⋈ Student.Std = Subject.Class SUBJECT

Student detail				
SID	Name	Std	Class	Subject
101	Alex	10	10	Math
101	Alex	10	10	English
102	Maria	11	11	Music
102	Maria	11	11	Sports

Equijoin

When Theta join uses only **equality** comparison operator, it is said to be equijoin. The above example corresponds to equijoin.

Natural Join ($\bowtie$)

Natural join does not use any comparison operator. It does not concatenate the way a Cartesian product does. We can perform a Natural Join only if there is at least one common attribute that exists between two relations. In addition, the attributes must have the same name and domain.

Natural join acts on those matching attributes where the values of attributes in both the relations are same.

Courses		
CID	Course	Dept

CS01		Database	CS
ME01		Mechanics	ME
EE01		Electronics	EE
HoD			
Dept		Head	
CS		Alex	
ME		Maya	
EE		Mira	
Courses m HoD			
Dept	CID	Course	Head
CS	CS01	Database	Alex
ME	ME01	Mechanics	Maya
EE	EE01	Electronics	Mira

Outer Joins

Theta Join, Equijoin, 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, and full outer join.

Left Outer Join($R \bowtie S$)

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

Left			
A		B	
100		Database	
101		Mechanics	
102		Electronics	
Right			
A		B	
100		Alex	
102		Maya	
104		Mira	
Courses ⌋ HoD			
A	B	C	D
100	Database	100	Alex
101	Mechanics	---	---

102	Electronics	102	Maya
-----	-------------	-----	------

Right Outer Join: (R $\bowtie$ 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.

Courses $\bowtie$ HoD			
A	B	C	D
100	Database	100	Alex
102	Electronics	102	Maya
---	---	104	Mira

Full Outer Join: (R $\bowtie$ S)

All the tuples from both participating relations are included in the resulting relation. If there are no matching tuples for both relations, their respective unmatched attributes are made NULL.

Courses $\bowtie$ HoD			
A	B	C	D
100	Database	100	Alex
101	Mechanics	---	---
102	Electronics	102	Maya
---	---	104	Mira

DIVISION:-

Division is typically required when you want to find out entities that are interacting with **all entities** of a set of different type entities.

Some instances where division operator is used are:

- Which person has account in all the banks of a particular city?
- Which students have taken all the courses required to graduate?
-

In all these queries, the description after the keyword 'all' defines a set which contains some elements and the final result contains those units who satisfy these requirements.

Important: Division is not supported by SQL implementations. However, it can be represented using other operations.(like cross join, Except, In)SQL Implementation of Division.

Given two relations(tables): $R(x,y)$, $S(y)$.

R and S : tables

x and y : column of R

y : column of S

$R(x,y) \text{ div } S(y)$ means gives all distinct values of x from R that are associated with all values of y in S.

Computation of Division: $R(x,y) \text{ div } S(y)$

Steps:

- Find out all possible combinations of $S(y)$ with $R(x)$ by computing $R(x) \times S(y)$, say $r1$
- Subtract actual $R(x,y)$ from $r1$, say $r2$
- x in $r2$ are those that are not associated with every value in $S(y)$; therefore $R(x)-r2(x)$ gives us x that are associated with all values in

Relational Calculus

In contrast to Relational Algebra, Relational Calculus is a non-procedural query language, that is, it tells what to do but never explains how to do it.

Relational calculus exists in two forms –

Tuple Relational Calculus (TRC)

Filtering variable ranges over tuples

Notation – $\{T \mid \text{Condition}\}$

Returns all tuples T that satisfies a condition.

For example –

```
{ T.name | Author(T) AND T.article = 'database' }
```

Output – Returns tuples with 'name' from Author who has written article on 'database'.

TRC can be quantified. We can use Existential ($\exists$) and Universal Quantifiers ($\forall$).

For example –

```
{ R |  $\exists T \in \text{Authors}(T.\text{article} = \text{'database' AND } R.\text{name} = T.\text{name})$ }
```

Output – The above query will yield the same result as the previous one.

Domain Relational Calculus (DRC)

In DRC, the filtering variable uses the domain of attributes instead of entire tuple values (as done in TRC, mentioned above).

Notation –

$$\{ a_1, a_2, a_3, \dots, a_n \mid P(a_1, a_2, a_3, \dots, a_n) \}$$

Where a_1, a_2 are attributes and **P** stands for formulae built by inner attributes.

For example –

```
{< article, page, subject > |  $\_ \in \text{TutorialsPoint} \wedge \text{subject} = \text{'database'}$ }
```

Output – Yields Article, Page, and Subject from the relation TutorialsPoint, where subject is database.

Just like TRC, DRC can also be written using existential and universal quantifiers. DRC also involves relational operators.

The expression power of Tuple Relation Calculus and Domain Relation Calculus is equivalent to Relational Algebra.