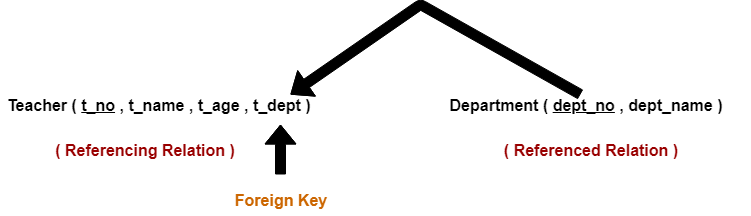
**Foreign Key-**

* An attribute ‘X’ is called as a foreign key to some other attribute ‘Y’ when its values are dependent on the values of attribute ‘Y’.
* The attribute ‘X’ can assume only those values which are assumed by the attribute ‘Y’.
* Here, the relation in which attribute ‘Y’ is present is called as the **referenced relation**.
* The relation in which attribute ‘X’ is present is called as the **referencing relation**.
* The attribute ‘Y’ might be present in the same table or in some other table.

**Example-**

Consider the following two schemas-



Here, t\_dept can take only those values which are present in dept\_no in Department table since only those departments actually exist.

**NOTES-**

* Foreign key references the primary key of the table.
* Foreign key can take only those values which are present in the primary key of the referenced relation.
* Foreign key may have a name other than that of a primary key.
* Foreign key can take the NULL value.
* There is no restriction on a foreign key to be unique.
* In fact, foreign key is not unique most of the time.
* Referenced relation may also be called as the master table or primary table.
* Referencing relation may also be called as the foreign table.

**Converting ER Diagrams to Tables-**

After designing an [**ER Diagram**](https://www.gatevidyalay.com/er-diagrams/),

* ER diagram is converted into the tables in relational model.
* This is because relational models can be easily implemented by RDBMS like MySQL , Oracle etc.

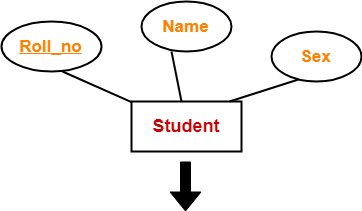
Following rules are used for converting an ER diagram into the tables-

**Rule-01: For 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.

**Example-**



|  |  |  |
| --- | --- | --- |
| **Roll\_no** | **Name** | **Sex** |
|  |  |  |

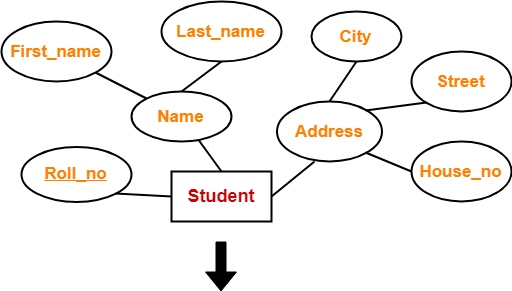
**Schema : Student ( Roll\_no , Name , Sex )**

**Also Read-** [**Entity Sets in DBMS**](https://www.gatevidyalay.com/entity-sets-in-dbms/)

**Rule-02: For 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.

**Example-**



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Roll\_no** | **First\_name** | **Last\_name** | **House\_no** | **Street** | **City** |
|  |  |  |  |  |  |

**Schema : Student ( Roll\_no , First\_name , Last\_name , House\_no , Street , City )**

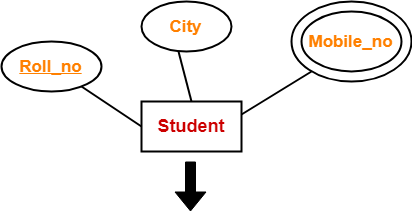
**Also Read-** [**Types of Attributes in DBMS**](https://www.gatevidyalay.com/types-of-attributes/)

**Rule-03: For 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.

**Example-**



|  |  |
| --- | --- |
| **Roll\_no** | **City** |
|  |  |

|  |  |
| --- | --- |
| **Roll\_no** | **Mobile\_no** |
|  |  |

**Rule-04: Translating Relationship Set into a Table-**

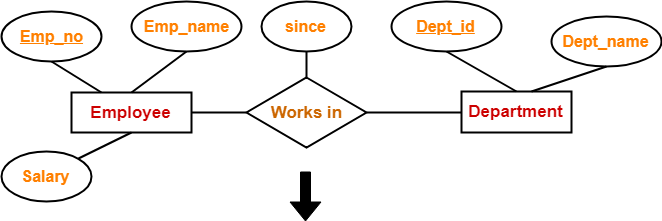
A relationship set will require one table in the relational model.

Attributes of the table are-

* Primary key attributes of the participating entity sets
* Its own descriptive attributes if any.

Set of non-descriptive attributes will be the primary key.

**Example-**



|  |  |  |
| --- | --- | --- |
| **Emp\_no** | **Dept\_id** | **since** |
|  |  |  |

**Schema : Works in ( Emp\_no , Dept\_id , since )**

**NOTE-**

If we consider the overall ER diagram, three tables will be required in relational model-

* One table for the entity set “Employee”
* One table for the entity set “Department”
* One table for the relationship set “Works in”

**Rule-05: For Binary Relationships With Cardinality Ratios-**

The following four cases are possible-

**Case-01:** Binary relationship with cardinality ratio m:n

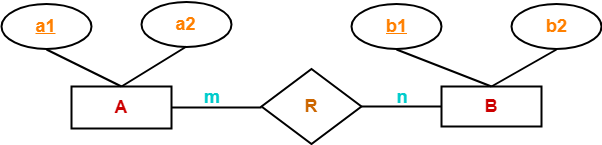
**Case-02:** Binary relationship with cardinality ratio 1:n

**Case-03:** Binary relationship with cardinality ratio m:1

**Case-04:** Binary relationship with cardinality ratio 1:1

**Also read-** [**Cardinality Ratios in DBMS**](https://www.gatevidyalay.com/cardinality-in-er-diagram/)

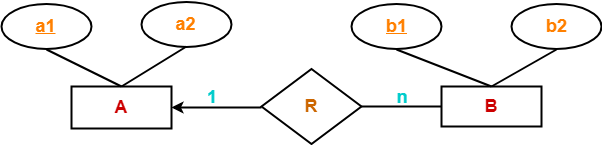
**Case-01: For Binary Relationship With Cardinality Ratio m:n**



Here, three tables will be required-

1. A ( a1 , a2 )
2. R ( a1 , b1 )
3. B ( b1 , b2 )

**Case-02: For Binary Relationship With Cardinality Ratio 1:n**

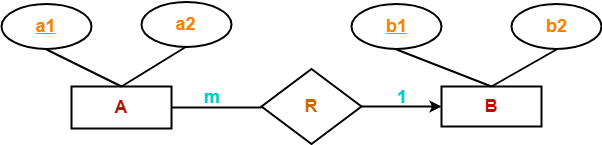


Here, two tables will be required-

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

**NOTE-** Here, combined table will be drawn for the entity set B and relationship set R.

**Case-03: For Binary Relationship With Cardinality Ratio m:1**

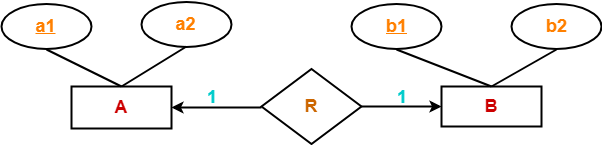


Here, two tables will be required-

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

**NOTE-** Here, combined table will be drawn for the entity set A and relationship set R.

**Case-04: For Binary Relationship With Cardinality Ratio 1:1**



Here, two tables will be required. Either combine ‘R’ with ‘A’ or ‘B’

**Way-01:**

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

**Way-02:**

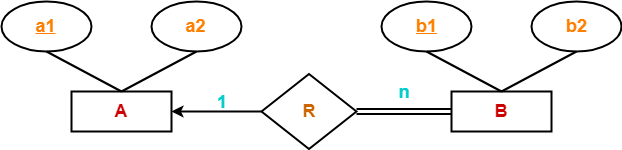
1. A ( a1 , a2 )
2. BR ( a1 , b1 , b2 )

|  |
| --- |
| **Thumb Rules to Remember**    While determining the minimum number of tables required for binary relationships with given cardinality ratios, following thumb rules must be kept in mind-   * For binary relationship with cardinality ration m : n , separate and individual tables will be drawn for each entity set and relationship. * For binary relationship with cardinality ratio either m : 1 or 1 : n , always remember “many side will consume the relationship” i.e. a combined table will be drawn for many side entity set and relationship set. * For binary relationship with cardinality ratio 1 : 1 , two tables will be required. You can combine the relationship set with any one of the entity sets. |

**Rule-06: For Binary Relationship With Both Cardinality Constraints and Participation Constraints-**

* Cardinality constraints will be implemented as discussed in Rule-05.
* Because of the total participation constraint, foreign key acquires **NOT NULL** constraint i.e. now foreign key can not be null.

**Case-01: For Binary Relationship With Cardinality Constraint and Total Participation Constraint From One Side-**



Because cardinality ratio = 1 : n , so we will combine the entity set B and relationship set R.

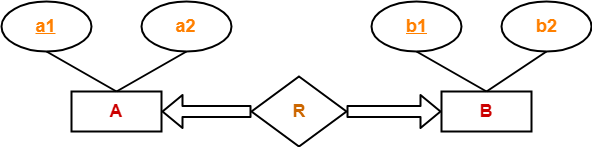
Then, two tables will be required-

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

Because of total participation, foreign key a1 has acquired NOT NULL constraint, so it can’t be null now.

**Case-02: For Binary Relationship With Cardinality Constraint and Total Participation Constraint From Both Sides-**

If there is a key constraint from both the sides of an entity set with total participation, then that binary relationship is represented using only single table.

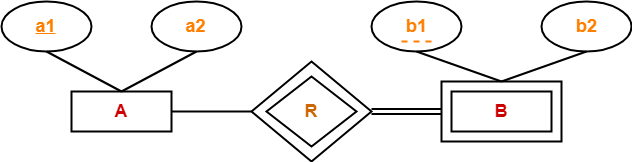


Here, Only one table is required.

* ARB ( a1 , a2 , b1 , b2 )

**Rule-07: For Binary Relationship With Weak Entity Set-**

Weak entity set always appears in association with identifying relationship with total participation constraint.



Here, two tables will be required-

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

**ER Diagrams to Tables-**

Before you go through this article, make sure that you have gone through the previous article on [**ER Diagrams to Tables**](https://www.gatevidyalay.com/er-diagrams-to-tables/).

After designing an [**ER Diagram**](https://www.gatevidyalay.com/er-diagrams/),

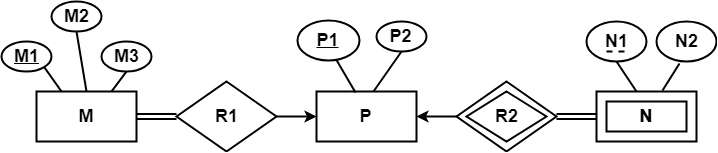
* ER diagram is converted into the tables in relational model.
* This is because relational models can be easily implemented by RDBMS like MySQL , Oracle etc.
* The rules used for converting an ER diagram into the tables are already discussed.

In this article, we will discuss practice problems based on converting ER Diagrams to Tables.

**PRACTICE PROBLEMS BASED ON CONVERTING ER DIAGRAM TO TABLES-**

**Problem-01:**

Find the minimum number of tables required for the following ER diagram in relational model-



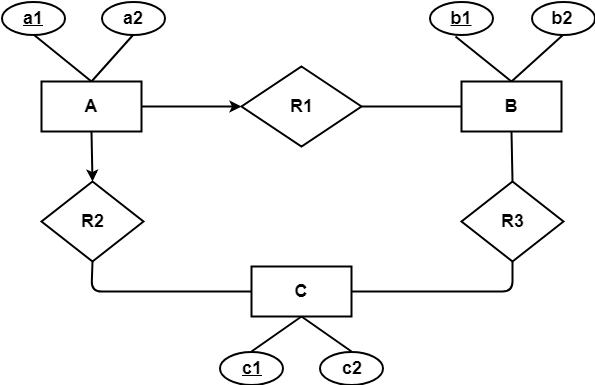
**Solution-**

Applying the rules, minimum 3 tables will be required-

* MR1 (M1 , M2 , M3 , P1)
* P (P1 , P2)
* NR2 (P1 , N1 , N2)

**Problem-02:**

Find the minimum number of tables required to represent the given ER diagram in relational model-



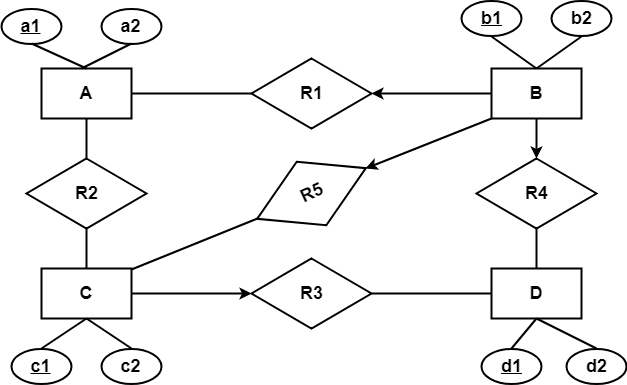
**Solution-**

Applying the rules, minimum 4 tables will be required-

* AR1R2 (a1 , a2 , b1 , c1)
* B (b1 , b2)
* C (c1 , c2)
* R3 (b1 , c1)

**Problem-03:**

Find the minimum number of tables required to represent the given ER diagram in relational model-



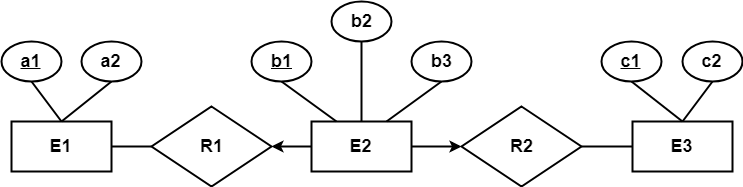
**Solution-**

Applying the rules, minimum 5 tables will be required-

* BR1R4R5 (b1 , b2 , a1 , c1 , d1)
* A (a1 , a2)
* R2 (a1 , c1)
* CR3 (c1 , c2 , d1)
* D (d1 , d2)

**Problem-04:**

Find the minimum number of tables required to represent the given ER diagram in relational model-



**Solution-**

Applying the rules, minimum 3 tables will be required-

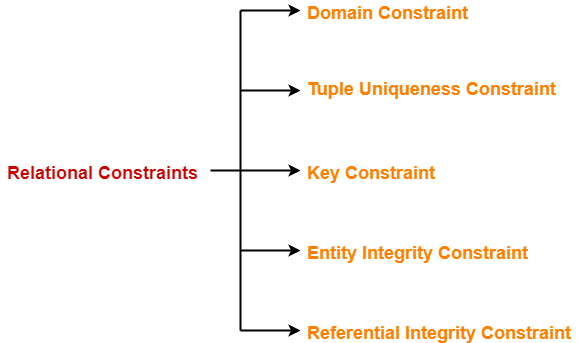
* E1 (a1 , a2)
* E2R1R2 (b1 , b2 , a1 , c1 , b3)
* E3 (c1 , c2)

**Constraints in DBMS-**

* Relational constraints are the restrictions imposed on the database contents and operations.
* They ensure the correctness of data in the database.

**Types of Constraints in DBMS-**

In DBMS, there are following 5 different types of relational constraints-



1. Domain constraint
2. Tuple Uniqueness constraint
3. Key constraint
4. Entity Integrity constraint
5. Referential Integrity constraint

**1. Domain Constraint-**

* Domain constraint defines the domain or set of values for an attribute.
* It specifies that the value taken by the attribute must be the atomic value from its domain.

**Example-**

Consider the following Student table-

|  |  |  |
| --- | --- | --- |
| **STU\_ID** | **Name** | **Age** |
| S001 | Akshay | 20 |
| S002 | Abhishek | 21 |
| S003 | Shashank | 20 |
| S004 | Rahul | **A** |

Here, value**‘A’** is not allowed since only integer values can be taken by the age attribute.

**2. Tuple Uniqueness Constraint-**

Tuple Uniqueness constraint specifies that all the tuples must be necessarily unique in any relation.

**Example-01:**

Consider the following Student table-

|  |  |  |
| --- | --- | --- |
| **STU\_ID** | **Name** | **Age** |
| S001 | Akshay | 20 |
| S002 | Abhishek | 21 |
| S003 | Shashank | 20 |
| S004 | Rahul | 20 |

This relation satisfies the tuple uniqueness constraint since here all the tuples are unique.

**Example-02:**

Consider the following Student table-

|  |  |  |
| --- | --- | --- |
| **STU\_ID** | **Name** | **Age** |
| **S001** | **Akshay** | **20** |
| **S001** | **Akshay** | **20** |
| S003 | Shashank | 20 |
| S004 | Rahul | 20 |

This relation does not satisfy the tuple uniqueness constraint since here all the tuples are not unique.

**3. Key Constraint-**

Key constraint specifies that in any relation-

* All the values of primary key must be unique.
* The value of primary key must not be null.

**Example-**

Consider the following Student table-

|  |  |  |
| --- | --- | --- |
| **STU\_ID** | **Name** | **Age** |
| **S001** | Akshay | 20 |
| **S001** | Abhishek | 21 |
| S003 | Shashank | 20 |
| S004 | Rahul | 20 |

This relation does not satisfy the key constraint as here all the values of primary key are not unique.

**4. Entity Integrity Constraint-**

* Entity integrity constraint specifies that no attribute of primary key must contain a null value in any relation.
* This is because the presence of null value in the primary key violates the uniqueness property.

**Example-**

Consider the following Student table-

|  |  |  |
| --- | --- | --- |
| **STU\_ID** | **Name** | **Age** |
| S001 | Akshay | 20 |
| S002 | Abhishek | 21 |
| S003 | Shashank | 20 |
|  | Rahul | 20 |

This relation does not satisfy the entity integrity constraint as here the primary key contains a NULL value.

**5. Referential Integrity Constraint-**

* This constraint is enforced when a foreign key references the primary key of a relation.
* It specifies that all the values taken by the foreign key must either be available in the relation of the primary key or be null.

**Read more-** [**Foreign Key in DBMS**](https://www.gatevidyalay.com/keys-in-dbms/)

**Important Results-**

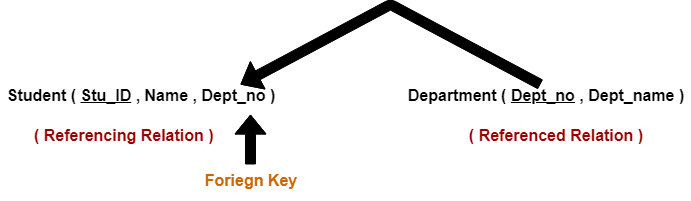
The following two important results emerges out due to referential integrity constraint-

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

**Example-**

Consider the following two relations- ‘Student’ and ‘Department’.

Here, relation ‘Student’ references the relation ‘Department’.



**Student**

|  |  |  |
| --- | --- | --- |
| **STU\_ID** | **Name** | **Dept\_no** |
| S001 | Akshay | D10 |
| S002 | Abhishek | D10 |
| S003 | Shashank | D11 |
| S004 | Rahul | **D14** |

**Department**

|  |  |
| --- | --- |
| **Dept\_no** | **Dept\_name** |
| D10 | ASET |
| D11 | ALS |
| D12 | ASFL |
| D13 | ASHS |

Here,

* The relation ‘Student’ does not satisfy the referential integrity constraint.
* This is because in relation ‘Department’, no value of primary key specifies department no. 14.
* Thus, referential integrity constraint is violated.

**Handling Violation of Referential Integrity Constraint-**

To ensure the correctness of the database, it is important to handle the violation of referential integrity constraint properly.

* Referential Integrity constraint is enforced when a foreign key references the primary key of a relation.
* It specifies that all the values taken by the foreign key must either be available in the relation of the primary key or be null.

**Also read-** [**Foreign Key in DBMS**](https://www.gatevidyalay.com/keys-in-dbms/)

**Referential Integrity Constraint Violation-**

There are following three possible causes of violation of referential integrity constraint-

**Cause-01:** Insertion in a referencing relation

**Cause-02:** Deletion from a referenced relation

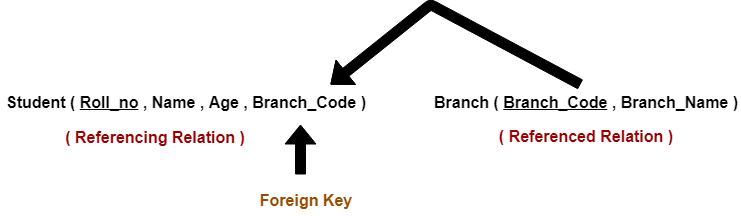
**Cause-03:** Updation in a referenced relation

**Cause-01: Insertion in a Referencing Relation-**

* It is allowed to insert only those values in the referencing attribute which are already present in the value of the referenced attribute.
* Inserting a value in the referencing attribute which is not present in the value of the referenced attribute violates the referential integrity constraint.

**Example-**

Consider the following two schemas-



Here, relation “Student” references the relation “Branch”.

**Student**

|  |  |  |  |
| --- | --- | --- | --- |
| **Roll\_no** | **Name** | **Age** | **Branch\_Code** |
| 1 | Rahul | 22 | CS |
| 2 | Anjali | 21 | CS |
| 3 | Teena | 20 | IT |

**Branch**

|  |  |
| --- | --- |
| **Branch\_Code** | **Branch\_Name** |
| CS | Computer Science |
| EE | Electronics Engineering |
| IT | Information Technology |
| CE | Civil Engineering |

Here,

* In relation “Student”, we can not insert any student having branch code ME (Mechanical Engineering).
* This is because branch code ME is not present in the relation “Branch”.

**Cause-02: Deletion from a Referenced Relation-**

* It is not allowed to delete a row from the referenced relation if the referencing attribute uses the value of the referenced attribute of that row.
* Such a deletion violates the referential integrity constraint.

**Example-**

Consider the above two relations,

* We can not delete a tuple from the relation “Branch” having branch code ‘CS’.
* This is because the referencing attribute “Branch\_Code” of the referencing relation “Student” references the value ‘CS’.
* However, we can safely delete a tuple from the relation “Branch” having branch code ‘CE’.
* This is because the referencing attribute “Branch\_Code” of the referencing relation “Student” does not uses this value.

**Handling the Violation-**

The violation caused due to a deletion from the referenced relation can be handled in the following three ways-

**Method-01:**

* This method involves simultaneously deleting those tuples from the referencing relation where the referencing attribute uses the value of referenced attribute being deleted.
* This method of handling the violation is called as **On Delete Cascade**.

**Method-02:**

* This method involves aborting or deleting the request for a deletion from the referenced relation if the value is used by the referencing relation.

**Method-03:**

* This method involves setting the value being deleted from the referenced relation to NULL or some other value in the referencing relation if the referencing attribute uses that value.

**Cause-03: Updation in a Referenced Relation-**

* It is not allowed to update a row of the referenced relation if the referencing attribute uses the value of the referenced attribute of that row.
* Such an updation violates the referential integrity constraint.

**Example-**

Consider the above relation.

* We can not update a tuple in the relation “Branch” having branch code ‘CS’ to the branch code ‘CSE’.
* This is because referencing attribute “Branch\_Code” of the referencing relation “Student” references the value ‘CS’.

**Handling the Violation-**

The violation caused due to an updation in the referenced relation can be handled in the following three ways-

**Method-01:**

* This method involves simultaneously updating those tuples of the referencing relation where the referencing attribute uses the referenced attribute value being updated.
* This method of handling the violation is called as **On Update Cascade**.

**Method-02:**

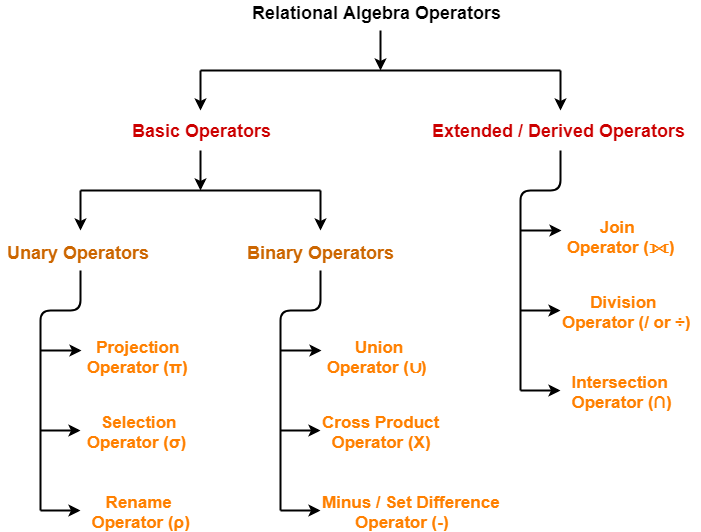
* This method involves aborting or deleting the request for an updation of the referenced relation if the value is used by the referencing relation.

**Method-03:**

* This method involves setting the value being updated in the referenced relation to NULL or some other value in the referencing relation if the referencing attribute uses that value.

## ****Relational Algebra-****

|  |
| --- |
| Relational Algebra is a procedural query language which takes a relation as an input and generates a relation as an output. |



## ****Characteristics-****

Following are the important characteristics of relational operators-

* Relational Operators always work on one or more relational tables.
* Relational Operators always produce another relational table.
* The table produced by a relational operator has all the properties of a relational model.

**Selection Operator-**

* Selection Operator (σ) is a unary operator in relational algebra that performs a selection operation.
* It selects those rows or tuples from the relation that satisfies the selection condition.

**Syntax-**

|  |
| --- |
| **σ<selection\_condition>(R)** |

**Examples-**

* Select tuples from a relation “Books” where subject is “database”

σsubject = “database” (Books)

* Select tuples from a relation “Books” where subject is “database” and price is “450”

σsubject = “database” ∧ price = “450” (Books)

* Select tuples from a relation “Books” where subject is “database” and price is “450” or have a publication year after 2010

σsubject = “database” ∧ price = “450” ∨ year >”2010″ (Books)

**Important Points-**

**Point-01:**

* We may use logical operators like ∧ , ∨ , ! and relational operators like  = , ≠ , > , < , <= , >= with the selection condition.

**Point-02:**

* Selection operator only selects the required tuples according to the selection condition.
* It does not display the selected tuples.
* To display the selected tuples, projection operator is used.

**Point-03:**

* Selection operator always selects the entire tuple. It can not select a section or part of a tuple.

**Point-04:**

* Selection operator is commutative in nature i.e.

σ A ∧ B (R) = σ B ∧ A (R)

**Point-05:**

* Degree of the relation from a selection operation is same as degree of the input relation.

**Point-06:**

* The number of rows returned by a selection operation is obviously less than or equal to the number of rows in the original table.

Thus,

* Minimum Cardinality = 0
* Maximum Cardinality = |R|

## ****Projection Operator-****

* Projection Operator (π) is a unary operator in relational algebra that performs a projection operation.
* It displays the columns of a relation or table based on the specified attributes.

## ****Syntax-****

|  |
| --- |
| **π<attribute list>(R)** |

## ****Example-****

Consider the following Student relation-

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Name** | **Subject** | **Age** |
| 100 | Ashish | Maths | 19 |
| 200 | Rahul | Science | 20 |
| 300 | Naina | Physics | 20 |
| 400 | Sameer | Chemistry | 21 |

#### ****Student****

Then, we have-

**Result for Query πName, Age(Student)-**

|  |  |
| --- | --- |
| **Name** | **Age** |
| Ashish | 19 |
| Rahul | 20 |
| Naina | 20 |
| Sameer | 21 |

**Result for Query πID , Name(Student)-**

|  |  |
| --- | --- |
| **ID** | **Name** |
| 100 | Ashish |
| 200 | Rahul |
| 300 | Naina |
| 400 | Sameer |

**Important Points-**

**Point-01:**

* The degree of output relation (number of columns present) is equal to the number of attributes mentioned in the attribute list.

**Point-02:**

* Projection operator automatically removes all the duplicates while projecting the output relation.
* So, cardinality of the original relation and output relation may or may not be same.
* If there are no duplicates in the original relation, then the cardinality will remain same otherwise it will surely reduce.

**Point-03:**

* If attribute list is a super key on relation R, then we will always get the same number of tuples in the output relation.
* This is because then there will be no duplicates to filter.

**Point-04:**

* Projection operator does not obey commutative property i.e.

π <list2> (π <list1>(R)) ≠ π <list1> (π <list2>(R))

**Point-05:**

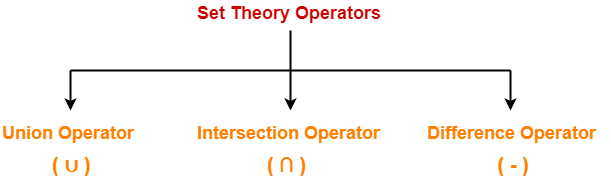
* [**Selection Operator**](https://www.gatevidyalay.com/selection-operator-relational-algebra-dbms/) performs horizontal partitioning of the relation.
* Projection operator performs vertical partitioning of the relation.

**Point-06:**

* There is only one difference between projection operator of relational algebra and SELECT operation of SQL.
* Projection operator does not allow duplicates while SELECT operation allows duplicates.
* To avoid duplicates in SQL, we use “distinct” keyword and write SELECT distinct.
* Thus, projection operator of relational algebra is equivalent to SELECT operation of SQL.

## ****Set Theory Operators-****

Following operators are called as set theory operators-



1. Union Operator (∪)
2. Intersection Operator (∩)
3. Difference Operator (-)

|  |
| --- |
| ****Condition For Using Set Theory Operators****   To use set theory operators on two relations,  The two relations must be union compatible.  Union compatible property means-   * Both the relations must have same number of attributes. * The attribute domains (types of values accepted by attributes) of both the relations must be compatible. |

**Also read-** [**Selection Operator**](https://www.gatevidyalay.com/selection-operator-relational-algebra-dbms/) and [**Projection Operator**](https://www.gatevidyalay.com/projection-operator-relational-algebra-dbms/)

## ****1. Union Operator (∪)-****

Let R and S be two relations.

Then-

* R ∪ S is the set of all tuples belonging to either R or S or both.
* In R ∪ S, duplicates are automatically removed.
* Union operation is both commutative and associative.

## ****Example-****

Consider the following two relations R and S-

|  |  |  |
| --- | --- | --- |
| **ID** | **Name** | **Subject** |
| 100 | Ankit | English |
| 200 | Pooja | Maths |
| 300 | Komal | Science |

#### ****Relation R****

|  |  |  |
| --- | --- | --- |
| **ID** | **Name** | **Subject** |
| 100 | Ankit | English |
| 400 | Kajol | French |

#### ****Relation S****

Then, R ∪ S is-

|  |  |  |
| --- | --- | --- |
| **ID** | **Name** | **Subject** |
| 100 | Ankit | English |
| 200 | Pooja | Maths |
| 300 | Komal | Science |
| 400 | Kajol | French |

#### ****Relation R ∪ S****

## ****2. Intersection Operator (∩)-****

Let R and S be two relations.

Then-

* R ∩ S is the set of all tuples belonging to both R and S.
* In R ∩ S, duplicates are automatically removed.
* Intersection operation is both commutative and associative.

## ****Example-****

Consider the following two relations R and S-

|  |  |  |
| --- | --- | --- |
| **ID** | **Name** | **Subject** |
| 100 | Ankit | English |
| 200 | Pooja | Maths |
| 300 | Komal | Science |

#### ****Relation R****

|  |  |  |
| --- | --- | --- |
| **ID** | **Name** | **Subject** |
| 100 | Ankit | English |
| 400 | Kajol | French |

#### ****Relation S****

Then, R ∩ S is-

|  |  |  |
| --- | --- | --- |
| **ID** | **Name** | **Subject** |
| 100 | Ankit | English |

#### ****Relation R ∩ S****

## ****3. Difference Operator (-)-****

Let R and S be two relations.

Then-

* R – S is the set of all tuples belonging to R and not to S.
* In R – S, duplicates are automatically removed.
* Difference operation is associative but not commutative.

### ****Example-****

Consider the following two relations R and S-

|  |  |  |
| --- | --- | --- |
| **ID** | **Name** | **Subject** |
| 100 | Ankit | English |
| 200 | Pooja | Maths |
| 300 | Komal | Science |

#### ****Relation R****

|  |  |  |
| --- | --- | --- |
| **ID** | **Name** | **Subject** |
| 100 | Ankit | English |
| 400 | Kajol | French |

#### ****Relation S****

Then, R – S is-

|  |  |  |
| --- | --- | --- |
| **ID** | **Name** | **Subject** |
| 200 | Pooja | Maths |
| 300 | Komal | Science |

#### ****Relation R – S****