**DBMS – ER Design Issues**

We have already covered ER diagram in our previous article [**DBMS ER Model Concept**](https://beginnersbook.com/2015/04/e-r-model-in-dbms/). In this post, we will discuss the various issues that can arise while designing an ER diagram..

Here are some of the issues that can occur while ER diagram design process:

**1. Choosing Entity Set vs Attributes**

Here we will discuss how choosing an entity set vs an attribute can change the whole ER design semantics. To understand this lets take an example, let’s say we have an entity set Student with attributes such as student-name and student-id. Now we can say that the student-id itself can be an entity with the attributes like student-class and student-section.

Now if we compare the two cases we discussed above, in the first case we can say that the student can have only one student id, however in the second case when we chose student id as an entity it implied that a student can have more than one student id.

**2. Choosing Entity Set vs. Relationship Sets**

It is hard to decide that an object can be best represented by an entity set or relationship set. To comprehend and decide the perfect choice between these two (entity vs relationship), the user needs to understand whether the entity would need a new relationship if a requirement arise in future, if this is the case then it is better to choose entity set rather than relationship set.

Let’s take an example to understand it better: A person takes a loan from a bank, here we have two entities person and bank and their relationship is loan. This is fine until there is a need to disburse a joint loan, in such case a new relationship needs to be created to define the relationship between the two individuals who have taken joint loan. In this scenario, it is better to choose loan as an entity set rather than a relationship set.

**3. Choosing Binary vs n-ary Relationship Sets**

In most cases, the relationships described in an **ER diagrams** are binary. The **n-ary** relationships are those where entity sets are more than two, if the entity sets are only two, their relationship can be termed as binary relationship.

The n-ary relationships can make ER design complex, however the good news is that we can convert and represent any n-ary relationship using multiple binary relationships.

This may sound confusing so lets take an example to understand how we can convert an n-ary relationship to multiple binary relationships. Now lets say we have to describe a relationship between four family members: father, mother, son and daughter. This can easily be represented in forms of multiple binary relationships, father-mother relationship as “spouse”, son and daughter relationship as “siblings” and father and mother relationship with their child as “child”.

**4. Placing Relationship Attributes**

The [**cardinality ratio**](https://beginnersbook.com/2015/04/cardinality-in-dbms/) in DBMS can help us determine in which scenarios we need to place relationship attributes. It is recommended to represent the attributes of **one to one** or **one to many** relationship sets with any participating entity sets rather than a relationship set.

**For example**, if an entity cannot be determined as a separate entity rather it is represented by the combination of participating entity sets. In such case it is better to associate these entities to many-to-many relationship sets.

**DBMS – ER Diagram to Table Conversion**

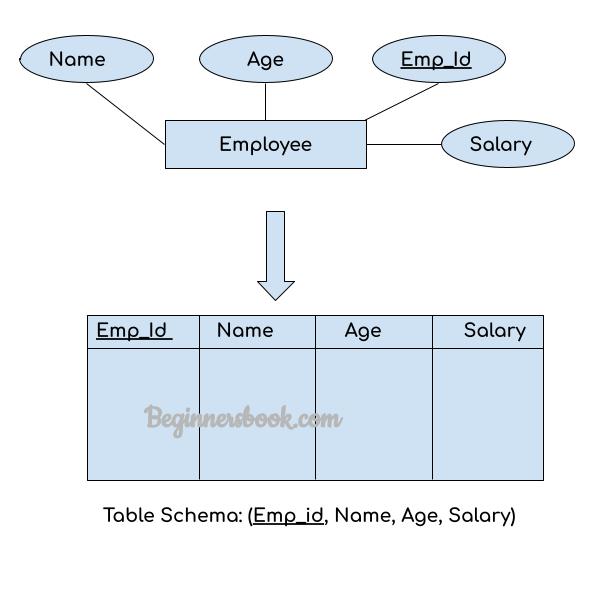
We have learned [**ER Diagram**](https://beginnersbook.com/2015/04/e-r-model-in-dbms/) and [**ER design issues**](https://beginnersbook.com/2021/07/dbms-er-design-issues/) in previous articles. In this post, we will cover how to convert ER diagram into database tables.

First we will convert simple ER diagrams to tables. In the end, we will take a complex ER diagram and then we will convert it into set of tables.

**1. Strong Entity set with Simple attributes**

The Strong Entity set becomes the table and the attributes of the Entity set becomes the table attributes. The key attribute of the entity set becomes the [**primary key**](https://beginnersbook.com/2015/04/primary-key-in-dbms/) of the table.

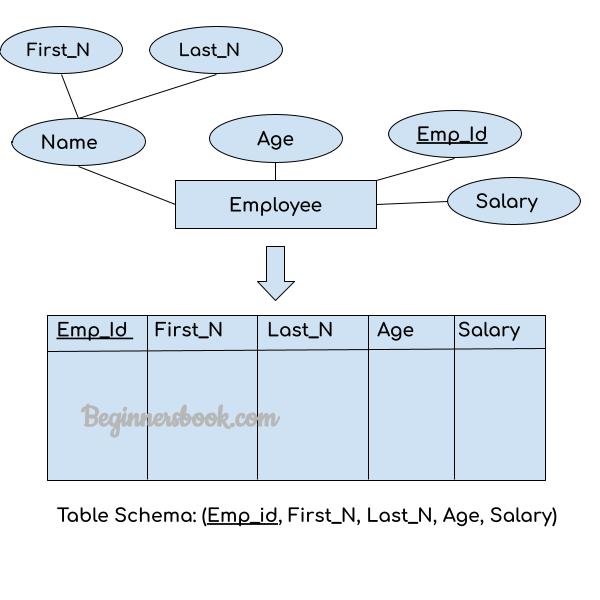
Let’s take an example: Here we have an entity set Employee with the attributes Name, Age, Emp\_Id and Salary. When we convert this ER diagram to table, the entity set becomes table so we have a table named “Employee” as shown in the following diagram. The attributes of the entity set becomes the attributes of the table.



**2. Strong Entity Set With Composite Attributes**

Now we will see how to convert Strong entity set with [**composite**](https://beginnersbook.com/2015/04/composite-key-in-dbms/) attributes ER to table. The conversion is fairly simple in this case as well. The entity set will be the table and the simple attributes of the composite attributes will become the attributes of the table while the composite attribute itself will be ignored during conversion.

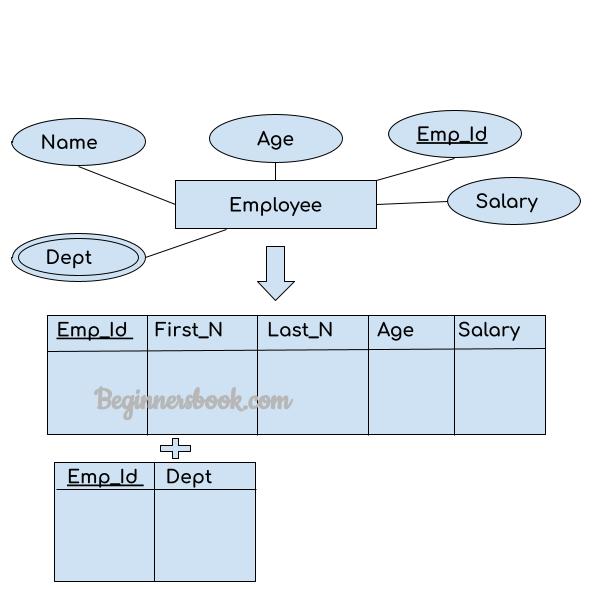
Let’s take an example. As you can see we have a composite attribute Name and this composite attribute has two simple attributes First\_N and Last\_N. While converting this ER to table we have not used the composite attribute itself in the table instead we have used the simple attributes of this composite attribute as table’s attributes.



**3. Strong Entity Set With Multi Valued Attributes**

Entity set with [**multi-valued**](https://beginnersbook.com/2015/04/multivalued-dependency-in-dbms/) attributes will require two tables in the relational model.

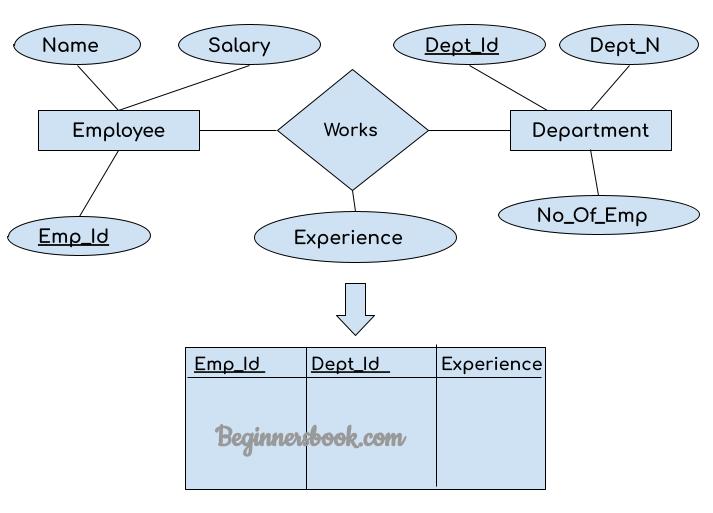
We will understand this conversion with the help of a diagram. Let’s take the same example that we have seen above, here we have added a new multi-valued attribute Dept. An employee can work in multiple department so we have this Dept attribute marked as multi-valued. Whenever we have a multi-valued attribute, there needs to be **more than one table** to represent the ER diagram. As you can see we have created two tables to represent this ER.



**4. Relationship Set to Table conversion**

While converting the relationship set to a table, the primary attributes of the two entity sets becomes the table attributes and if the relationship set has any attribute that also becomes the attribute of the table.

In the following example, we have two entity sets Employee and Department. These entity sets are associated to each other using the Works relationship set. To convert this relationship set Works to the table, we take the primary attributes of each entity set, these are Emp\_Id and Dept\_Id and all the attributes of the relationship set and form a table.

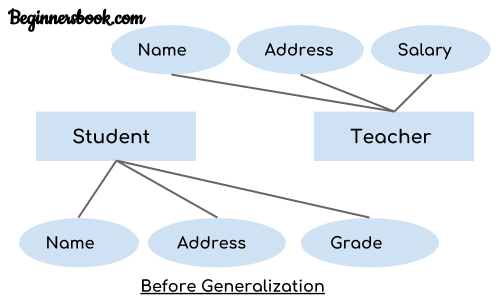


# DBMS Generalization

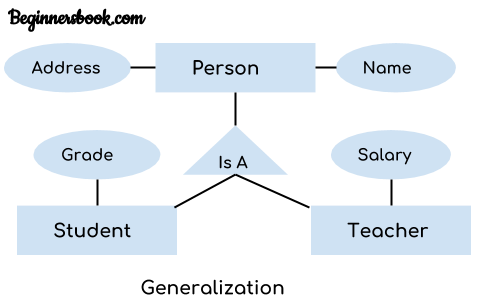
**Generalization** is a process in which the common attributes of more than one entities form a new entity. This newly formed entity is called generalized entity.

## Generalization Example

Lets say we have two entities Student and Teacher.  
Attributes of Entity Student are: Name, Address & Grade  
Attributes of Entity Teacher are: Name, Address & Salary

**The ER diagram before generalization looks like this:**  


These two entities have two common attributes: Name and Address, we can make a generalized entity with these common attributes. Lets have a look at the ER model after generalization.

**The ER diagram after generalization:**  
We have created a new generalized entity Person and this entity has the common attributes of both the entities. As you can see in the following [**ER diagram**](https://beginnersbook.com/2015/04/e-r-model-in-dbms/) that after the generalization process the entities Student and Teacher only has the specialized attributes Grade and Salary respectively and their common attributes (Name & Address) are now associated with a new entity Person which is in the relationship with both the entities (Student & Teacher).  


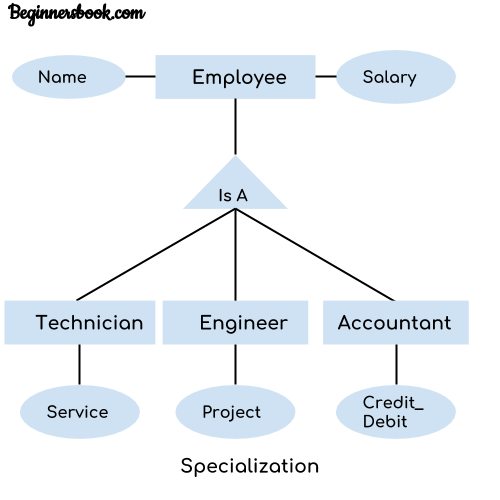
**Note:**  
1. Generalization uses bottom-up approach where two or more lower level entities combine together to form a higher level new entity.  
2. The new generalized entity can further combine together with lower level entity to create a further higher level generalized entity.

# DBMS Specialization

**Specialization** is a process in which an entity is divided into sub-entities. You can think of it as a reverse process of [**generalization**](https://beginnersbook.com/2018/11/dbms-generalization/), in generalization two entities combine together to form a new higher level entity. Specialization is a top-down process.

The idea behind Specialization is to find the subsets of entities that have few distinguish attributes. For example – Consider an entity employee which can be further classified as sub-entities Technician, Engineer & Accountant because these sub entities have some distinguish attributes.

## Specialization Example

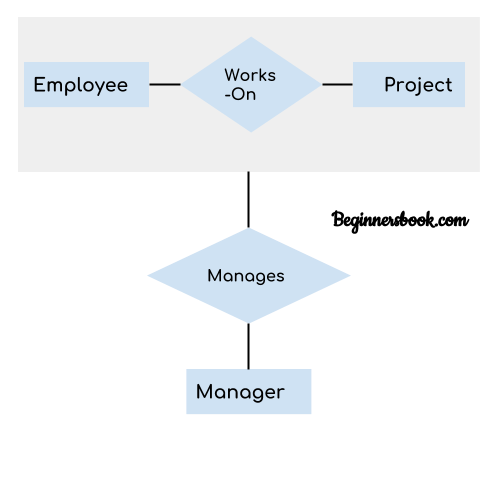


In the above diagram, we can see that we have a higher level entity “Employee” which we have divided in sub entities “Technician”, “Engineer” & “Accountant”. All of these are just an employee of a company, however their role is completely different and they have few different attributes. Just for the example, I have shown that Technician handles service requests, Engineer works on a project and Accountant handles the credit & debit details. All of these three employee types have few attributes common such as name & salary which we had left associated with the parent entity “Employee” as shown in the above diagram.

# DBMS Aggregration

**Aggregation** is a process in which a single entity alone is not able to make sense in a relationship so the relationship of two entities acts as one entity. I know it sounds confusing but don’t worry the example we will take, will clear all the doubts.

## Aggregration Example



In real world, we know that a manager not only manages the employee working under them but he has to manage the project as well. In such scenario if entity “Manager” makes a “manages” relationship with either “Employee” or “Project” entity alone then it will not make any sense because he has to manage both. In these cases the relationship of two entities acts as one entity. In our example, the relationship “Works-On” between “Employee” & “Project” acts as one entity that has a relationship “Manages” with the entity “Manager”.

**Relational model in DBMS**

In relational model, the data and relationships are represented by collection of inter-related tables. Each table is a group of column and rows, where column represents attribute of an entity and rows represents records.

**Sample relationship Model**: Student table with 3 columns and four records.

**Table: Student**

|  |  |  |
| --- | --- | --- |
| Stu\_Id | Stu\_Name | Stu\_Age |
| 111 | Ashish | 23 |
| 123 | Saurav | 22 |
| 169 | Lester | 24 |
| 234 | Lou | 26 |

**Table: Course**

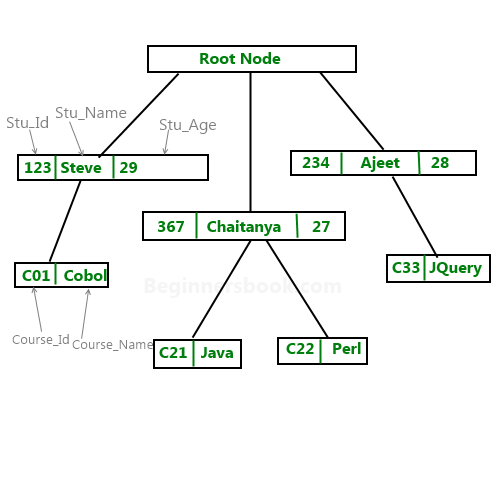
|  |  |  |
| --- | --- | --- |
| Stu\_Id | Course\_Id | Course\_Name |
| 111 | C01 | Science |
| 111 | C02 | DBMS |
| 169 | C22 | Java |
| 169 | C39 | Computer Networks |

Here Stu\_Id, Stu\_Name & Stu\_Age are attributes of table Student and Stu\_Id, Course\_Id & Course\_Name are attributes of table Course. The rows with values are the records (commonly known as tuples).

**Hierarchical model in DBMS**

In **hierarchical model**, data is organized into a tree like structure with each record is having one parent record and many children. The main drawback of this model is that, it can have only one to many relationships between nodes.

**Note: Hierarchical models are rarely used now.**

**Sample Hierarchical Model Diagram**:  
Lets say we have few students and few courses and a course can be assigned to a single student only, however a student take any number of courses so this relationship becomes one to many.  


**Example of hierarchical data represented as relational tables:** The above hierarchical model can be represented as relational tables like this:

|  |  |  |
| --- | --- | --- |
| Stu\_Id | Stu\_Name | Stu\_Age |
| 123 | Steve | 29 |
| 367 | Chaitanya | 27 |
| 234 | Ajeet | 28 |

Course Table:

|  |  |  |
| --- | --- | --- |
| Course\_Id | Course\_Name | Stu\_Id |
| C01 | Cobol | 123 |
| C21 | Java | 367 |
| C22 | Perl | 367 |
| C33 | JQuery | 234 |

# Constraints in DBMS

Constraints enforce limits to the data or type of data that can be inserted/updated/deleted from a table. The whole purpose of constraints is to maintain the **data integrity**during an update/delete/insert into a table. In this tutorial we will learn several types of constraints that can be created in RDBMS.

## Types of constraints

* NOT NULL
* UNIQUE
* DEFAULT
* CHECK
* Key Constraints – PRIMARY KEY, FOREIGN KEY
* Domain constraints
* Mapping constraints

#### NOT NULL:

NOT NULL constraint makes sure that a column does not hold NULL value. When we don’t provide value for a particular column while inserting a record into a table, it takes NULL value by default. By specifying NULL constraint, we can be sure that a particular column(s) cannot have NULL values.

Example:

CREATE TABLE STUDENT(

ROLL\_NO INT **NOT NULL**,

STU\_NAME VARCHAR (35) **NOT NULL**,

STU\_AGE INT **NOT NULL**,

STU\_ADDRESS VARCHAR (235),

PRIMARY KEY (ROLL\_NO)

);

Read more about [**this constraint here**](https://beginnersbook.com/2014/05/not-null-constraint-in-sql/).

#### UNIQUE:

UNIQUE Constraint enforces a column or set of columns to have unique values. If a column has a unique constraint, it means that particular column cannot have duplicate values in a table.

CREATE TABLE STUDENT(

ROLL\_NO INT NOT NULL,

STU\_NAME VARCHAR (35) NOT NULL **UNIQUE**,

STU\_AGE INT NOT NULL,

STU\_ADDRESS VARCHAR (35) **UNIQUE**,

PRIMARY KEY (ROLL\_NO)

);

Read more about it [**here**](https://beginnersbook.com/2014/05/unique-constraint-in-sql/).

#### DEFAULT:

The DEFAULT constraint provides a default value to a column when there is no value provided while inserting a record into a table.

CREATE TABLE STUDENT(

ROLL\_NO   INT  NOT NULL,

STU\_NAME VARCHAR (35) NOT NULL,

STU\_AGE INT NOT NULL,

EXAM\_FEE INT  **DEFAULT** 10000,

STU\_ADDRESS VARCHAR (35) ,

PRIMARY KEY (ROLL\_NO)

);

Read more: [**Default constraint**](https://beginnersbook.com/2014/05/default-constraint-in-sql/)

#### CHECK:

This constraint is used for specifying range of values for a particular column of a table. When this constraint is being set on a column, it ensures that the specified column must have the value falling in the specified range.

CREATE TABLE STUDENT(

ROLL\_NO   INT  NOT NULL CHECK(ROLL\_NO >1000) ,

STU\_NAME VARCHAR (35)  NOT NULL,

STU\_AGE INT  NOT NULL,

EXAM\_FEE INT DEFAULT 10000,

STU\_ADDRESS VARCHAR (35) ,

PRIMARY KEY (ROLL\_NO)

);

In the above example we have set the check constraint on ROLL\_NO column of STUDENT table. Now, the ROLL\_NO field must have the value greater than 1000.

## Key constraints:

#### PRIMARY KEY:

[**Primary key**](https://beginnersbook.com/2015/04/primary-key-in-dbms/) uniquely identifies each record in a table. It must have unique values and cannot contain nulls. In the below example the ROLL\_NO field is marked as primary key, that means the ROLL\_NO field cannot have duplicate and null values.

CREATE TABLE STUDENT(

ROLL\_NO   INT  NOT NULL,

STU\_NAME VARCHAR (35)  NOT NULL UNIQUE,

STU\_AGE INT NOT NULL,

STU\_ADDRESS VARCHAR (35) UNIQUE,

**PRIMARY KEY** (ROLL\_NO)

);

#### FOREIGN KEY:

Foreign keys are the columns of a table that points to the primary key of another table. They act as a cross-reference between tables.  
Read more about it [**here**](https://beginnersbook.com/2015/04/foreign-key-in-dbms/).

#### Domain constraints:

Each table has certain set of columns and each column allows a same type of data, based on its data type. The column does not accept values of any other data type.  
[**Domain constraints**](https://beginnersbook.com/2015/04/domain-constraints-in-dbms/) are **user defined data type** and we can define them like this:

Domain Constraint = data type + Constraints (NOT NULL / UNIQUE / PRIMARY KEY / FOREIGN KEY / CHECK / DEFAULT)

# Cardinality in DBMS

In DBMS you may hear cardinality term at two different places and it has two different meanings as well.

**In Context of Data Models**:  
In terms of [**data models**](https://beginnersbook.com/2015/04/data-models-in-dbms/), cardinality refers to the relationship between two tables. Relationship can be of four types as we have already seen in [**Entity relationship guide**](https://beginnersbook.com/2015/04/e-r-model-in-dbms/):

One to One – A single row of first table associates with single row of second table. For example, a relationship between person and passport table is one to one because a person can have only one passport and a passport can be assigned to only one person.

One to Many – A single row of first table associates with more than one rows of second table. For example, relationship between customer and order table is one to many because a customer can place many orders but a order can be placed by a single customer alone.

Many to One – Many rows of first table associate with a single row of second table. For example, relationship between student and university is many to one because a university can have many students but a student can only study only in single university at a time.

Many to Many – Many rows of first table associate with many rows of second table. For example, relationship between student and course table is many to many because a student can take many courses at a time and a course can be assigned to many students.

**In Context of Query Optimization**:  
In terms of query, the cardinality refers to the uniqueness of a column in a table. The column with all unique values would be having the high cardinality and the column with all duplicate values would be having the low cardinality. These cardinality scores helps in query optimization.

# RDBMS Concepts

**RDBMS** stands for relational database management system. A relational model can be represented as a table of rows and columns. A relational database has following major components:  
1. Table  
2. Record or Tuple  
3. Field or Column name or Attribute  
4. Domain  
5. Instance  
6. Schema  
7. Keys

## 1. Table

A table is a collection of data represented in rows and columns. Each table has a name in database. For example, the following table “STUDENT” stores the information of students in database.

**Table: STUDENT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Student\_Id** | **Student\_Name** | **Student\_Addr** | **Student\_Age** |
| 101 | Chaitanya | Dayal Bagh, Agra | 27 |
| 102 | Ajeet | Delhi | 26 |
| 103 | Rahul | Gurgaon | 24 |
| 104 | Shubham | Chennai | 25 |

## 2. Record or Tuple

Each row of a table is known as record. It is also known as tuple. For example, the following row is a record that we have taken from the above table.

|  |  |  |  |
| --- | --- | --- | --- |
| 102 | Ajeet | Delhi | 26 |

## 3. Field or Column name or Attribute

The above table “STUDENT” has four fields (or attributes): Student\_Id, Student\_Name, Student\_Addr & Student\_Age.

## 4. Domain

A domain is a set of permitted values for an attribute in table. For example, a domain of month-of-year can accept January, February,…December as values, a domain of dates can accept all possible valid dates etc. We specify domain of attribute while creating a table.

An attribute cannot accept values that are outside of their domains. For example, In the above table “STUDENT”, the Student\_Id field has integer domain so that field cannot accept values that are not integers for example, Student\_Id cannot has values like, “First”, 10.11 etc.

## 5. Instance and Schema

I have already covered instance and schema in a separate guide, you can refer the [**guide here**](https://beginnersbook.com/2015/04/instance-and-schema-in-dbms/).

## 6. Keys

This is our next topic, I have covered the keys in detail in separate tutorials. You can refer the [**keys index here**](https://beginnersbook.com/2015/04/keys-in-dbms/).

# Introduction to Relational algebra & Relational calculus

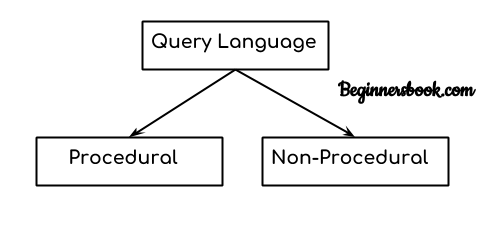
In this guide, we will discuss what is **Relational algebra and relational calculus** and why we use these concepts. In the previous tutorials, we discussed the designing of database using [**Relational model**](https://beginnersbook.com/2015/04/relational-model-in-dbms/), [**E-R diagram**](https://beginnersbook.com/2015/04/e-r-model-in-dbms/) and [**normalization**](https://beginnersbook.com/2015/05/normalization-in-dbms/). Now that we have designed the database, we need to store and retrieve data from the database, for this purpose we need to understand the concept of Relational algebra and relational calculus.

Let’s start with the basics.

## Query Language

In simple words, a Language which is used to store and retrieve data from database is known as query language. For example – **SQL**

There are two types of query language:  
1.Procedural Query language  
2.Non-procedural query language



### 1. Procedural Query language:

In procedural query language, user instructs the system to perform a series of operations to produce the desired results. Here users tells what data to be retrieved from database and how to retrieve it.

**For example –** Let’s take a real world example to understand the procedural language, you are asking your younger brother to make a cup of tea, if you are just telling him to make a tea and not telling the process then it is a non-procedural language, however if you are telling the step by step process like switch on the stove, boil the water, add milk etc. then it is a procedural language.

### 2. Non-procedural query language:

In Non-procedural query language, user instructs the system to produce the desired result without telling the step by step process. Here users tells what data to be retrieved from database but doesn’t tell how to retrieve it.

Now let’s back to our main topic of relational algebra and relational calculus.

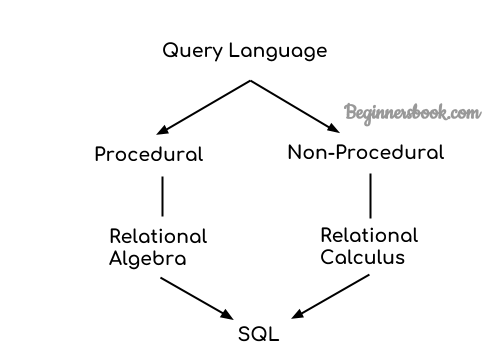
## Relational Algebra:

Relational algebra is a conceptual procedural query language used on relational model.

## Relational Calculus:

Relational calculus is a conceptual non-procedural query language used on relational model.

**Note:**  
I have used word conceptual while describing relational algebra and relational calculus, because they are theoretical mathematical system or query language, they are not the practical implementation, SQL is a practical implementation of relational algebra and relational calculus.



## Relational Algebra, Calculus, RDBMS & SQL:

Relational algebra and calculus are the theoretical concepts used on relational model.

RDBMS is a practical implementation of relational model.

SQL is a practical implementation of relational algebra and calculus.

In the next tutorials we will cover the relational algebra and calculus in detail.

# DBMS Relational Algebra

In this tutorial, we will discuss **Relational Algebra**. In the previous tutorial, we had a brief discussion on the [**basics of relational algebra and calculus**](https://beginnersbook.com/2019/02/introduction-to-relational-algebra-calculus/) where we learned the need to use these theoretical mathematical systems.

## What is Relational Algebra in DBMS?

Relational algebra is a **procedural** query language that works on relational model. The purpose of a query language is to retrieve data from database or perform various operations such as insert, update, delete on the data. When I say that relational algebra is a procedural query language, it means that it tells what data to be retrieved and how to be retrieved.

On the other hand relational calculus is a non-procedural query language, which means it tells what data to be retrieved but doesn’t tell how to retrieve it. We will discuss relational calculus in a separate tutorial.

## Types of operations in relational algebra

We have divided these operations in two categories:  
1. Basic Operations  
2. Derived Operations

### Basic/Fundamental Operations:

1. Select (σ)  
2. Project (∏)  
3. Union (∪)  
4. Set Difference (-)  
5. Cartesian product (X)  
6. Rename (ρ)

### Derived Operations:

1. Natural Join (⋈)  
2. Left, Right, Full outer join (⟕, ⟖, ⟗)  
3. Intersection (∩)  
4. Division (÷)

Lets discuss these operations one by one with the help of examples.

## Select Operator (σ)

Select Operator is denoted by sigma (σ) and it is used to find the tuples (or rows) in a relation (or table) which satisfy the given condition.

If you understand little bit of SQL then you can think of it as a [**where clause in SQL**](https://beginnersbook.com/2014/05/where-clause-in-sql/), which is used for the same purpose.

**Syntax of Select Operator (σ)**

σ Condition/Predicate(Relation/Table name)

### Select Operator (σ) Example

Table: CUSTOMER

---------------

Customer\_Id Customer\_Name Customer\_City

----------- ------------- -------------

C10100 Steve Agra

C10111 Raghu Agra

C10115 Chaitanya Noida

C10117 Ajeet Delhi

C10118 Carl Delhi

**Query:**

σ Customer\_City="Agra" (CUSTOMER)

**Output:**

Customer\_Id Customer\_Name Customer\_City

----------- ------------- -------------

C10100 Steve Agra

C10111 Raghu Agra

## Project Operator (∏)

Project operator is denoted by ∏ symbol and it is used to select desired columns (or attributes) from a table (or relation).

Project operator in relational algebra is similar to the [**Select statement in SQL**](https://beginnersbook.com/2018/11/sql-select/).

**Syntax of Project Operator (∏)**

∏ column\_name1, column\_name2, ...., column\_nameN(table\_name)

### Project Operator (∏) Example

In this example, we have a table CUSTOMER with three columns, we want to fetch only two columns of the table, which we can do with the help of Project Operator ∏.

Table: CUSTOMER

Customer\_Id Customer\_Name Customer\_City

----------- ------------- -------------

C10100 Steve Agra

C10111 Raghu Agra

C10115 Chaitanya Noida

C10117 Ajeet Delhi

C10118 Carl Delhi

**Query:**

∏ Customer\_Name, Customer\_City (CUSTOMER)

**Output:**

Customer\_Name Customer\_City

------------- -------------

Steve Agra

Raghu Agra

Chaitanya Noida

Ajeet Delhi

Carl Delhi

## Union Operator (∪)

Union operator is denoted by ∪ symbol and it is used to select all the rows (tuples) from two tables (relations).

Lets discuss union operator a bit more. Lets say we have two relations R1 and R2 both have same columns and we want to select all the tuples(rows) from these relations then we can apply the union operator on these relations.

**Note:** The rows (tuples) that are present in both the tables will only appear once in the union set. In short you can say that there are no duplicates present after the union operation.

**Syntax of Union Operator (∪)**

table\_name1 ∪ table\_name2

### Union Operator (∪) Example

Table 1: COURSE

Course\_Id Student\_Name Student\_Id

--------- ------------ ----------

C101 Aditya S901

C104 Aditya S901

C106 Steve S911

C109 Paul S921

C115 Lucy S931

Table 2: STUDENT

Student\_Id Student\_Name Student\_Age

------------ ---------- -----------

S901 Aditya 19

S911 Steve 18

S921 Paul 19

S931 Lucy 17

S941 Carl 16

S951 Rick 18

**Query:**

∏ Student\_Name (COURSE) ∪ ∏ Student\_Name (STUDENT)

**Output:**

Student\_Name

------------

Aditya

Carl

Paul

Lucy

Rick

Steve

**Note:** As you can see there are no duplicate names present in the output even though we had few common names in both the tables, also in the COURSE table we had the duplicate name itself.

## Intersection Operator (∩)

Intersection operator is denoted by ∩ symbol and it is used to select common rows (tuples) from two tables (relations).

Lets say we have two relations R1 and R2 both have same columns and we want to select all those tuples(rows) that are present in both the relations, then in that case we can apply intersection operation on these two relations R1 ∩ R2.

**Note:** Only those rows that are present in both the tables will appear in the result set.

**Syntax of Intersection Operator (∩)**

table\_name1 ∩ table\_name2

### Intersection Operator (∩) Example

Lets take the same example that we have taken above.  
Table 1: COURSE

Course\_Id Student\_Name Student\_Id

--------- ------------ ----------

C101 Aditya S901

C104 Aditya S901

C106 Steve S911

C109 Paul S921

C115 Lucy S931

Table 2: STUDENT

Student\_Id Student\_Name Student\_Age

------------ ---------- -----------

S901 Aditya 19

S911 Steve 18

S921 Paul 19

S931 Lucy 17

S941 Carl 16

S951 Rick 18

**Query:**

∏ Student\_Name (COURSE) ∩ ∏ Student\_Name (STUDENT)

**Output:**

Student\_Name

------------

Aditya

Steve

Paul

Lucy

## Set Difference (-)

Set Difference is denoted by – symbol. Lets say we have two relations R1 and R2 and we want to select all those tuples(rows) that are present in Relation R1 but **not** present in Relation R2, this can be done using Set difference R1 – R2.

**Syntax of Set Difference (-)**

table\_name1 - table\_name2

### Set Difference (-) Example

Lets take the same tables COURSE and STUDENT that we have seen above.

**Query:**  
Lets write a query to select those student names that are present in STUDENT table but not present in COURSE table.

∏ Student\_Name (STUDENT) - ∏ Student\_Name (COURSE)

**Output:**

Student\_Name

------------

Carl

Rick

## Cartesian product (X)

Cartesian Product is denoted by X symbol. Lets say we have two relations R1 and R2 then the cartesian product of these two relations (R1 X R2) would combine each tuple of first relation R1 with the each tuple of second relation R2. I know it sounds confusing but once we take an example of this, you will be able to understand this.

**Syntax of Cartesian product (X)**

R1 X R2

### Cartesian product (X) Example

Table 1: R

Col\_A Col\_B

----- ------

AA 100

BB 200

CC 300

Table 2: S

Col\_X Col\_Y

----- -----

XX 99

YY 11

ZZ 101

**Query:**  
Lets find the cartesian product of table R and S.

R X S

**Output:**

Col\_A Col\_B Col\_X Col\_Y

----- ------ ------ ------

AA 100 XX 99

AA 100 YY 11

AA 100 ZZ 101

BB 200 XX 99

BB 200 YY 11

BB 200 ZZ 101

CC 300 XX 99

CC 300 YY 11

CC 300 ZZ 101

**Note:** The number of rows in the output will always be the cross product of number of rows in each table. In our example table 1 has 3 rows and table 2 has 3 rows so the output has 3×3 = 9 rows.

## Rename (ρ)

Rename (ρ) operation can be used to rename a relation or an attribute of a relation.  
**Rename (ρ) Syntax:**  
ρ(new\_relation\_name, old\_relation\_name)

### Rename (ρ) Example

Lets say we have a table customer, we are fetching customer names and we are renaming the resulted relation to CUST\_NAMES.

Table: CUSTOMER

Customer\_Id Customer\_Name Customer\_City

----------- ------------- -------------

C10100 Steve Agra

C10111 Raghu Agra

C10115 Chaitanya Noida

C10117 Ajeet Delhi

C10118 Carl Delhi

**Query:**

ρ(CUST\_NAMES, ∏(Customer\_Name)(CUSTOMER))

**Output:**

CUST\_NAMES

----------

Steve

Raghu

Chaitanya

Ajeet

Carl