

Relational Data Model in DBMS

What is Relational Model?

Relational Model (RM) represents the database as a collection of relations. A relation is nothing but a table of values. Every row in the table represents a collection of related data values. These rows in the table denote a real-world entity or relationship.

The table name and column names are helpful to interpret the meaning of values in each row. The data are represented as a set of relations. In the relational model, data are stored as tables. However, the physical storage of the data is independent of the way the data are logically organized.

Some popular Relational Database management systems are:

- DB2 and Informix Dynamic Server – IBM
- Oracle and RDB – Oracle
- SQL Server and Access – Microsoft

Relational Model Concepts in DBMS

1. **Attribute:** Each column in a Table. Attributes are the properties which define a relation. e.g., Student_Rollno, NAME, etc.
2. **Tables** – In the Relational model the, relations are saved in the table format. It is stored along with its entities. A table has two properties rows and columns. Rows represent records and columns represent attributes.
3. **Tuple** – It is nothing but a single row of a table, which contains a single record.
4. **Relation Schema:** A relation schema represents the name of the relation with its attributes.
5. **Degree:** The total number of attributes which in the relation is called the degree of the relation.

6. **Cardinality:** Total number of rows present in the Table.
7. **Column:** The column represents the set of values for a specific attribute.
8. **Relation instance** – Relation instance is a finite set of tuples in the RDBMS system. Relation instances never have duplicate tuples.
9. **Relation key** – Every row has one, two or multiple attributes, which is called relation key.
10. **Attribute domain** – Every attribute has some pre-defined value and scope which is known as attribute domain

Table also called Relation

The diagram shows a table with three columns: CustomerID, CustomerName, and Status. The first column is highlighted in yellow and labeled 'Primary Key'. The second column is labeled 'Domain' with the example 'Ex: NOT NULL'. The table contains three rows of data. Annotations point to the columns and rows, explaining that columns are also called attributes and that rows are also called tuples. The total number of rows is identified as the cardinality, and the total number of columns is identified as the degree.

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive

© guru99.com

Primary Key

Domain
Ex: NOT NULL

Tuple OR Row
Total # of rows is **Cardinality**

Column OR Attributes
Total # of column is **Degree**

Relational Integrity Constraints

Relational Integrity constraints in DBMS are referred to conditions which must be present for a valid relation. These Relational constraints in DBMS are derived from the rules in the mini-world that the database represents.

There are many types of Integrity Constraints in DBMS. Constraints on the Relational database management system is mostly divided into three main categories are:

1. Domain Constraints
2. Key Constraints
3. Referential Integrity Constraints

Domain Constraints

Domain constraints can be violated if an attribute value is not appearing in the corresponding domain or it is not of the appropriate data type.

Domain constraints specify that within each tuple, and the value of each attribute must be unique. This is specified as data types which include standard data types integers, real numbers, characters, Booleans, variable length strings, etc.

Example:

```
Create DOMAIN CustomerName  
CHECK (value not NULL)
```

The example shown demonstrates creating a domain constraint such that CustomerName is not NULL

Key Constraints

An attribute that can uniquely identify a tuple in a relation is called the key of the table. The value of the attribute for different tuples in the relation has to be unique.

Example:

In the given table, CustomerID is a key attribute of Customer Table. It is most likely to have a single key for one customer, CustomerID =1 is only for the CustomerName =" Google".

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active

Referential Integrity Constraints

Referential Integrity constraints in DBMS are based on the concept of Foreign Keys. A foreign key is an important attribute of relation that should be referred to in other relationships. A referential integrity constraint state happens where relation refers to a key attribute of different or same relation. However, that key element must exist in the table.

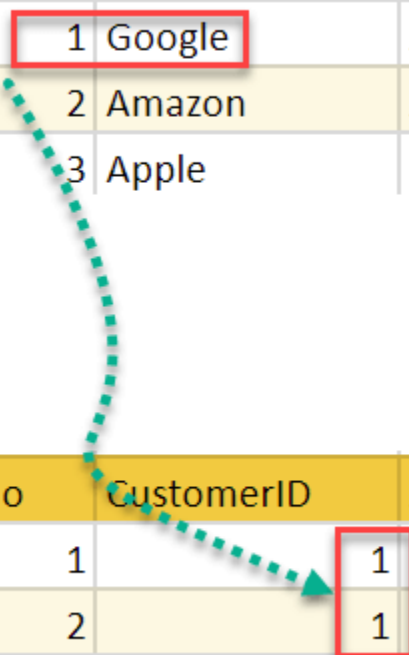
Example:

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive

Customer

InvoiceNo	CustomerID	Amount
1	1	\$100
2	1	\$200
3	2	\$150

Billing



In the above example, we have 2 relations, Customer and Billing. Tuple for CustomerID =1 is referenced twice in the relation Billing. So we know CustomerName=Google has billing amount \$300

Operations in Relational Model

Four basic update operations performed on relational database model are Insert, update, delete and select.


- Insert is used to insert data into the relation
- Delete is used to delete tuples from the table.
- Modify allows you to change the values of some attributes in existing tuples.
- Select allows you to choose a specific range of data.

Whenever one of these operations are applied, integrity constraints specified on the relational database schema must never be violated.

Insert Operation

The insert operation gives values of the attribute for a new tuple which should be inserted into a relation.

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive



CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive
4	Alibaba	Active

Update Operation

You can see that in the below-given relation table CustomerName= 'Apple' is updated from Inactive to Active.

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive
4	Alibaba	Active



CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Active
4	Alibaba	Active

Delete Operation

To specify deletion, a condition on the attributes of the relation selects the tuple to be deleted.

CustomerID	CustomerName	Status		CustomerID	CustomerName	Status
1	Google	Active		1	Google	Active
2	Amazon	Active		2	Amazon	Active
3	Apple	Active		4	Alibaba	Active
4	Alibaba	Active				

In the above-given example, CustomerName= “Apple” is deleted from the table.

The Delete operation could violate referential integrity if the tuple which is deleted is referenced by foreign keys from other tuples in the same [database](#).

Select Operation

CustomerID	CustomerName	Status		CustomerID	CustomerName	Status
1	Google	Active		2	Amazon	Active
2	Amazon	Active				
4	Alibaba	Active				

In the above-given example, CustomerName=“Amazon” is selected

Best Practices for creating a Relational Model

- Data need to be represented as a collection of relations
- Each relation should be depicted clearly in the table
- Rows should contain data about instances of an entity
- Columns must contain data about attributes of the entity
- Cells of the table should hold a single value
- Each column should be given a unique name
- No two rows can be identical
- The values of an attribute should be from the same domain

Advantages of Relational Database Model

- **Simplicity:** A Relational data model in DBMS is simpler than the hierarchical and network model.
- **Structural Independence:** The relational database is only concerned with data and not with a structure. This can improve the performance of the model.

- **Easy to use:** The Relational model in DBMS is easy as tables consisting of rows and columns are quite natural and simple to understand
- **Query capability:** It makes possible for a high-level query language like [SQL](#) to avoid complex database navigation.
- **Data independence:** The Structure of Relational database can be changed without having to change any application.
- **Scalable:** Regarding a number of records, or rows, and the number of fields, a database should be enlarged to enhance its usability.

Disadvantages of Relational Model

- Few relational databases have limits on field lengths which can't be exceeded.
- Relational databases can sometimes become complex as the amount of data grows, and the relations between pieces of data become more complicated.
- Complex relational database systems may lead to isolated databases where the information cannot be shared from one system to another.

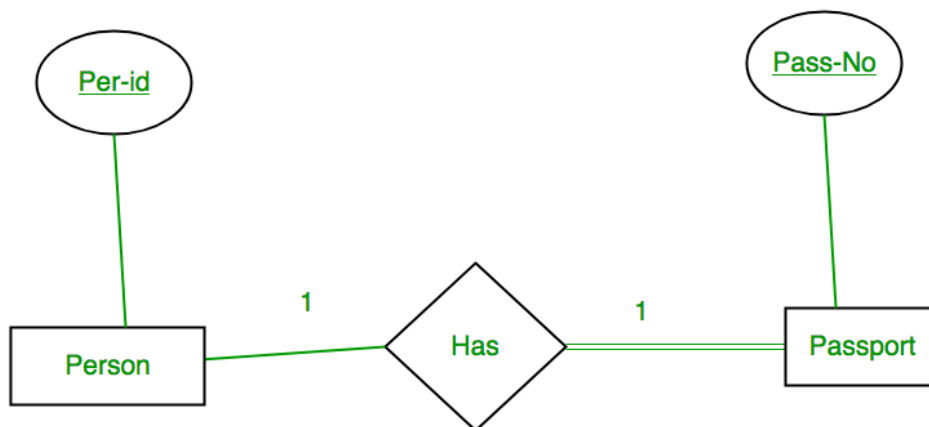
Summary

- The Relational database modelling represents the database as a collection of relations (tables)
- Attribute, Tables, Tuple, Relation Schema, Degree, Cardinality, Column, Relation instance, are some important components of Relational Model
- Relational Integrity constraints are referred to conditions which must be present for a valid Relation approach in DBMS
- Domain constraints can be violated if an attribute value is not appearing in the corresponding domain or it is not of the appropriate data type
- Insert, Select, Modify and Delete are the operations performed in Relational Model constraints
- The relational database is only concerned with data and not with a structure which can improve the performance of the model

- Advantages of Relational model in DBMS are simplicity, structural independence, ease of use, query capability, data independence, scalability, etc.
- Few relational databases have limits on field lengths which can't be exceeded.

Mapping from ER Model to Relational Model

Case 1: Binary Relationship with 1:1 cardinality with total participation of an entity



A person has 0 or 1 passport number and Passport is always owned by 1 person. So it is 1:1 cardinality with full participation constraint from Passport.

First Convert each entity and relationship to tables. Person table corresponds to Person Entity with key as Per-Id. Similarly Passport table corresponds to Passport Entity with key as Pass-No. Has Table represents relationship between Person and Passport (Which person has which passport). So it will take attribute Per-Id from Person and Pass-No from Passport.

Person		Has		Passport	
<u>Per-Id</u>	Other Person Attribute	<u>Per-Id</u>	Pass-No	<u>Pass-No</u>	Other PassportAttribute
PR1	-	PR1	PS1	PS1	-
PR2	-	PR2	PS2	PS2	-
PR3	-				

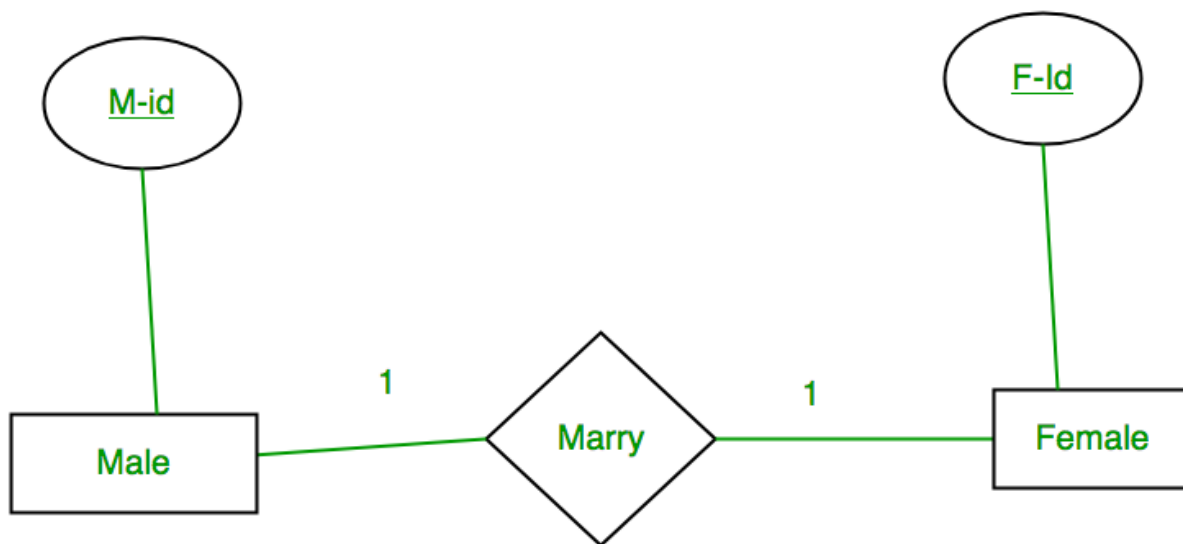
Table 1

As we can see from Table 1, each Per-Id and Pass-No has only one entry in Has Table. So we can merge all three tables into 1 with attributes shown in Table 2. Each Per-Id will be unique and not null. So it will be the key. Pass-No can't be key because for some person, it can be NULL.

Per-Id Other Person Attribute Pass-No Other PassportAttribute

Table 2

Case 2: Binary Relationship with 1:1 cardinality and partial participation of both entities



A male marries 0 or 1 female and vice versa as well. So it is 1:1 cardinality with partial participation constraint from both. First Convert each entity and relationship to tables. Male table corresponds to Male Entity with key as M-Id. Similarly Female table corresponds to Female Entity with key as F-Id. Marry Table represents relationship between Male and Female (Which Male marries which female). So it will take attribute M-Id from Male and F-Id from Female

Male		Marry		Female	
<u>M-Id</u>	Other Male Attribute	<u>M-Id</u>	F-Id	<u>F-Id</u>	Other FemaleAttribute
M1	-	M1	F2	F1	-
M2	-	M2	F1	F2	-
M3	-			F3	-

Table 3

As we can see from Table 3, some males and some females do not marry. If we merge 3 tables into 1, for some M-Id, F-Id will be NULL. So there is no attribute which is always not NULL. So we can't merge all three tables into 1. We can convert into 2 tables. In table 4, M-Id who are married will have F-Id associated. For others, it will be NULL. Table 5 will have information of all females. Primary Keys have been underlined.

M-Id Other Male Attribute F-Id

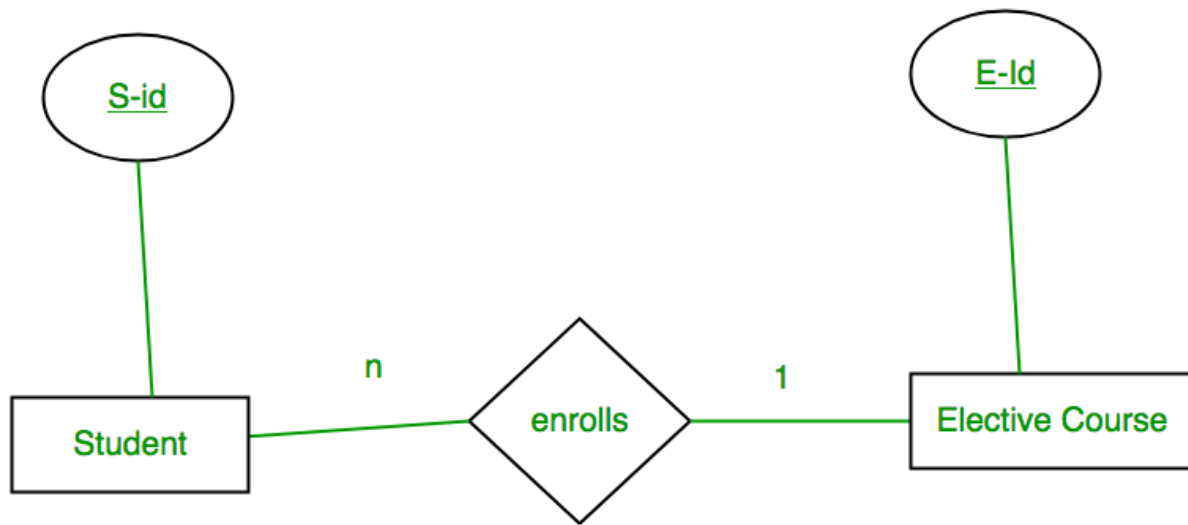
Table 4

F-Id Other FemaleAttribute

Table 5

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

Case 3: Binary Relationship with n: 1 cardinality



In this scenario, every student can enroll only in one elective course but for an elective course there can be more than one student. First Convert each entity and relationship to tables. Student table corresponds to Student Entity with key as S-Id. Similarly Elective_Course table corresponds to Elective_Course Entity with key as E-Id. Enrolls Table represents relationship between Student and Elective_Course (Which student enrolls in which course). So it will take attribute S-Id from Student and E-Id from Elective_Course.

Student		Enrolls		Elective_Course	
<u>S-</u> <u>Id</u>	Other Student Attribute	<u>S-</u> <u>Id</u>	E- Id	<u>E-</u> <u>Id</u>	Other Elective CourseAttribute
S1	-	S1	E1	E1	-
S2	-	S2	E2	E2	-
S3	-	S3	E1	E3	-
S4	-	S4	E1		

Table 6

As we can see from Table 6, S-Id is not repeating in Enrolls Table. So it can be considered as a key of Enrolls table. Both Student and Enrolls Table's key is same; we can merge it as a single table. The resultant tables are shown in Table 7 and Table 8. Primary Keys have been underlined.

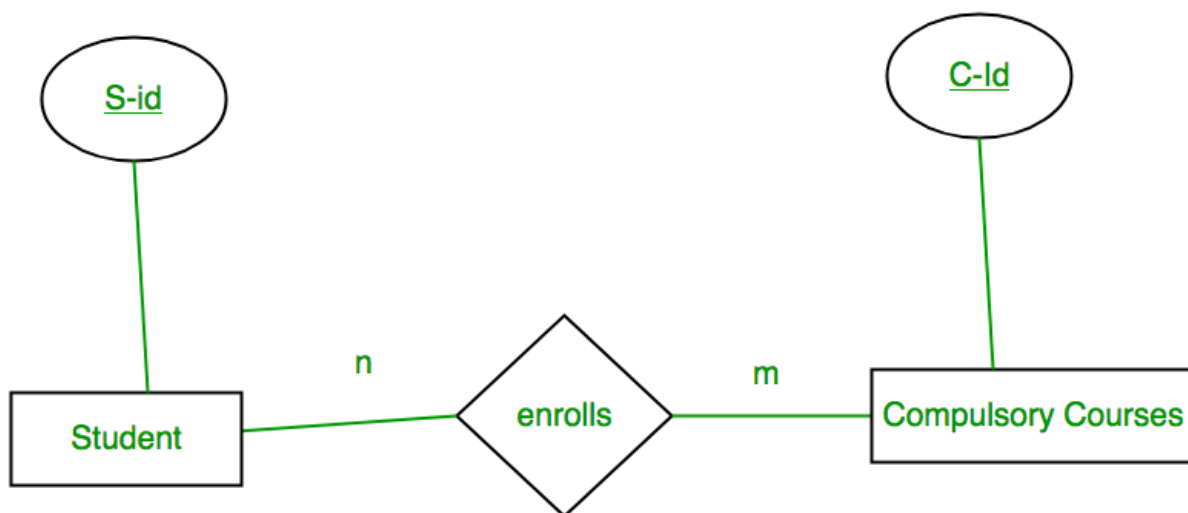
S-Id Other Student Attribute E-Id

Table 7

E-Id Other Elective CourseAttribute

Table 8

Case 4: Binary Relationship with m: n cardinality



In this scenario, every student can enroll in more than 1 compulsory course and for a compulsory course there can be more than 1 student. First Convert each entity and relationship to tables. Student table corresponds to Student Entity with

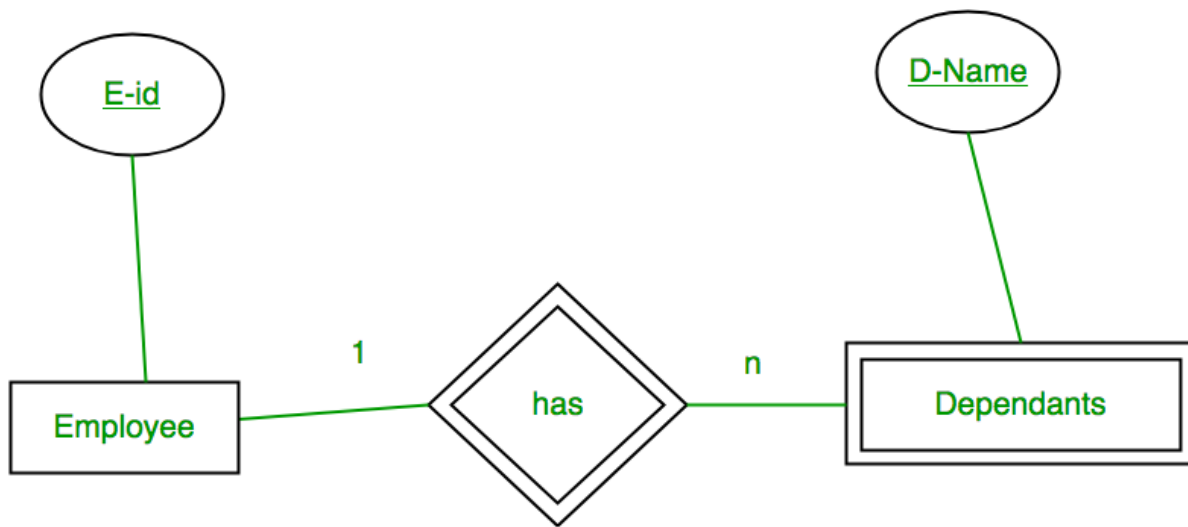
key as S-Id. Similarly Compulsory_Courses table corresponds to Compulsory Courses Entity with key as C-Id. Enrolls Table represents relationship between Student and Compulsory_Courses (Which student enrolls in which course). So it will take attribute S-Id from Person and C-Id from Compulsory_Courses.

Student		Enrolls		Compulsory_Courses	
<u>S-Id</u>	Other Student Attribute	<u>S-Id</u>	<u>C-Id</u>	<u>C-Id</u>	Other Compulsory CourseAttribute
S1	-	S1	C1	C1	-
S2	-	S1	C2	C2	-
S3	-	S3	C1	C3	-
S4	-	S4	C3	C4	-
		S4	C2		
		S3	C3		

Table 9

As we can see from Table 9, S-Id and C-Id both are repeating in Enrolls Table. But its combination is unique; so it can be considered as a key of Enrolls table. All tables' keys are different, these can't be merged. Primary Keys of all tables have been underlined.

Case 5: Binary Relationship with weak entity



In this scenario, an employee can have many dependents and one dependent can depend on one employee. A dependent does not have any existence without an employee (e.g; you as a child can be dependent of your father in his company). So it will be a weak entity and its participation will always be total. Weak Entity does not have key of its own. So its key will be combination of key of its identifying entity (E-Id of Employee in this case) and its partial key (D-Name).

First Convert each entity and relationship to tables. Employee table corresponds to Employee Entity with key as E-Id. Similarly Dependents table corresponds to Dependent Entity with key as D-Name and E-Id. Has Table represents relationship between Employee and Dependents (Which employee has which dependents). So it will take attribute E-Id from Employee and D-Name from Dependents.

Employee		Has		Dependents		
<u>E-Id</u>	Other Employee Attribute	<u>E-Id</u>	<u>D-Name</u>	<u>D-Name</u>	<u>E-Id</u>	Other DependentsAttribute
E1	-	E1	RAM	RAM	E1	-
E2	-	E1	SRINI	SRINI	E1	-
E3	-	E2	RAM	RAM	E2	-
		E3	ASHISH	ASHISH	E3	-

Table 10

As we can see from Table 10, E-Id, D-Name is key for **Has** as well as Dependents Table. So we can merge these two into 1. So the resultant tables are shown in Tables 11 and 12. Primary Keys of all tables have been underlined.

E-Id Other Employee Attribute

Table 11

D-Name E-Id Other DependentsAttribute

Table 12

Introduction of Relational Algebra in DBMS

Relational Algebra is procedural query language, which takes Relation as input and generate relation as output. Relational algebra mainly provides theoretical foundation for relational databases and SQL.

Relational Algebra

RELATIONAL ALGEBRA is a widely used procedural query language. It collects instances of relations as input and gives occurrences of relations as output. It uses various operations to perform this action. SQL Relational algebra query operations are performed recursively on a relation. The output of these operations is a new relation, which might be formed from one or more input relations.

In this tutorial, you will learn:

- Relational Algebra
- SELECT(σ)
- Projection(π)
- Rename (ρ)
- Union operation (\cup)
- Set Difference ($-$)
- Intersection
- Cartesian product(\times)
- Join Operations
- Inner Join:
- Theta Join:
- EQUI join:
- NATURAL JOIN (\bowtie)
- OUTER JOIN
- Left Outer Join($A \ltimes B$)
- Right Outer Join: ($A \rtimes B$)
- Full Outer Join: ($A \Join B$)

Basic SQL Relational Algebra Operations

Relational Algebra divided in various groups

Unary Relational Operations

- SELECT (symbol: σ)
- PROJECT (symbol: π)
- RENAME (symbol: ρ)

Relational Algebra Operations From Set Theory

- UNION (\cup)
- INTERSECTION (\cap),
- DIFFERENCE ($-$)
- CARTESIAN PRODUCT (\times)

Binary Relational Operations

- JOIN

- DIVISION

Let's study them in detail with solutions:

SELECT (σ)

The SELECT operation is used for selecting a subset of the tuples according to a given selection condition. σ Symbol denotes it. It is used as an expression to choose tuples which meet the selection condition. Select operator selects tuples that satisfy a given predicate.

$\sigma_p(r)$

σ is the predicate

r stands for relation which is the name of the table

p is propositional logic

Example 1

```
 $\sigma$  topic = "Database" (Tutorials)
```

Output – Selects tuples from Tutorials where topic = 'Database'.

Example 2

```
 $\sigma$  topic = "Database" and author = "guru99" ( Tutorials)
```

Output – Selects tuples from Tutorials where the topic is 'Database' and 'author' is guru99.

Example 3

```
 $\sigma$  sales > 50000 (Customers)
```

Output – Selects tuples from Customers where sales is greater than 50000

Projection(π)

The projection eliminates all attributes of the input relation but those mentioned in the projection list. The projection method defines a relation that contains a vertical subset of Relation.

This helps to extract the values of specified attributes to eliminates duplicate values. (pi) symbol is used to choose attributes from a relation. This operator helps you to keep specific columns from a relation and discards the other columns.

Example of Projection:

Consider the following table

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive
4	Alibaba	Active

Here, the projection of CustomerName and status will give

$\Pi_{\text{CustomerName, Status}}(\text{Customers})$

CustomerName	Status
Google	Active
Amazon	Active
Apple	Inactive
Alibaba	Active

Rename (ρ)

Rename is a unary operation used for renaming attributes of a relation.
 $\rho(a/b)R$ will rename the attribute 'b' of relation by 'a'.

Union operation (\cup)

UNION is symbolized by \cup symbol. It includes all tuples that are in tables A or in B. It also eliminates duplicate tuples. So, set A UNION set B would be expressed as:

The result $\leftarrow A \cup B$

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

- R and S must be the same number of attributes.
- Attribute domains need to be compatible.
- Duplicate tuples should be automatically removed.

Example

Consider the following tables.

Table A		Table B	
column 1	column 2	column 1	column 2
1	1	1	1
1	2	1	3

$A \cup B$ gives

Table $A \cup B$	
column 1	column 2
1	1
1	2

1	3
---	---

Set Difference (-)

– Symbol denotes it. The result of $A - B$, is a relation which includes all tuples that are in A but not in B.

- The attribute name of A has to match with the attribute name in B.
- The two-operand relations A and B should be either compatible or Union compatible.
- It should be defined relation consisting of the tuples that are in relation A, but not in B.

Example

$A - B$

Table A – B	
column 1	column 2
1	2

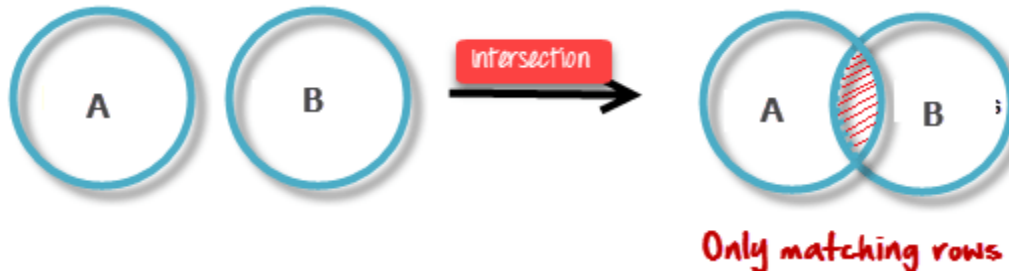
Intersection

An intersection is defined by the symbol \cap

$A \cap B$

Defines a relation consisting of a set of all tuple that are in both A and B.

However, A and B must be union-compatible.



Visual Definition of Intersection

Example:

$A \cap B$

Table $A \cap B$	
column 1	column 2
1	1

Cartesian Product(X) in DBMS

Cartesian Product in DBMS is an operation used to merge columns from two relations. Generally, a cartesian product is never a meaningful operation when it performs alone. However, it becomes meaningful when it is followed by other operations. It is also called Cross Product or Cross Join.

Example – Cartesian product

$\sigma_{\text{column 2} = '1'} (A \times B)$

Output – The above example shows all rows from relation A and B whose column 2 has value 1

$\sigma_{\text{column 2} = '1'} (A \times B)$	
column 1	column 2

1	1
1	1

Join Operations

Join operation is essentially a cartesian product followed by a selection criterion.

Join operation denoted by \bowtie .

JOIN operation also allows joining variously related tuples from different relations.

Types of JOIN:

Various forms of join operation are:

Inner Joins:

- Theta join
- EQUI join
- Natural join

Outer join:

- Left Outer Join
- Right Outer Join
- Full Outer Join

Inner Join:

In an inner join, only those tuples that satisfy the matching criteria are included, while the rest are excluded. Let's study various types of Inner Joins:

Theta Join:

The general case of JOIN operation is called a Theta join. It is denoted by symbol θ

Example

$A \bowtie_{\theta} B$

Theta join can use any conditions in the selection criteria.

For example:

$A \bowtie_{A.column\ 2 > B.column\ 2} (B)$

$A \bowtie_{A.column\ 2 > B.column\ 2} (B)$	
column 1	column 2
1	2

EQUI join:

When a theta join uses only equivalence condition, it becomes a equi join.

For example:

$A \bowtie_{A.column\ 2 = B.column\ 2} (B)$

$A \bowtie_{A.column\ 2 = B.column\ 2} (B)$	
column 1	column 2
1	1

EQUI join is the most difficult operations to implement efficiently using SQL in an RDBMS and one reason why RDBMS have essential performance problems.

NATURAL JOIN (\bowtie)

Natural join can only be performed if there is a common attribute (column) between the relations. The name and type of the attribute must be same.

Example

Consider the following two tables

C	
Num	Square
2	4
3	9

D	
Num	Cube
2	8
3	27

C ⋈ D

C ⋈ D		
Num	Square	Cube
2	4	8
3	9	27

OUTER JOIN

In an outer join, along with tuples that satisfy the matching criteria, we also include some or all tuples that do not match the criteria.

Left Outer Join($A \bowtie B$)

In the left outer join, operation allows keeping all tuple in the left relation. However, if there is no matching tuple is found in right relation, then the attributes of right relation in the join result are filled with null values.



Consider the following 2 Tables

A	
Num	Square
2	4
3	9
4	16

B	
Num	Cube

2	8
3	18
5	75

A \bowtie B

A \bowtie B		
Num	Square	Cube
2	4	8
3	9	18
4	16	—

Right Outer Join: (A \Joinr B)

In the right outer join, operation allows keeping all tuple in the right relation. However, if there is no matching tuple is found in the left relation, then the attributes of the left relation in the join result are filled with null values.



A \Joinr B

A ⋈ B		
Num	Cube	Square
2	8	4
3	18	9
5	75	—

Full Outer Join: (A ⋈ B)

In a full outer join, all tuples from both relations are included in the result, irrespective of the matching condition.

A ⋈ B

A ⋈ B		
Num	Cube	Square
2	4	8
3	9	18
4	16	—
5	—	75

Summary

Operation(Symbols)	Purpose
Select(σ)	The SELECT operation is used for selecting a subset of the tuples according to a given selection condition
Projection(π)	The projection eliminates all attributes of the input relation but those mentioned in the projection list.
Union Operation(\cup)	UNION is symbolized by symbol. It includes all tuples that are in tables A or in B.
Set Difference($-$)	– Symbol denotes it. The result of $A - B$, is a relation which includes all tuples that are in A but not in B.
Intersection(\cap)	Intersection defines a relation consisting of a set of all tuple that are in both A and B.
Cartesian Product(\times)	Cartesian operation is helpful to merge columns from two relations.
Inner Join	Inner join, includes only those tuples that satisfy the matching criteria.
Theta Join(θ)	The general case of JOIN operation is called a Theta join. It is denoted by symbol θ .

EQUI Join	When a theta join uses only equivalence condition, it becomes a equi join.
Natural Join(\bowtie)	Natural join can only be performed if there is a common attribute (column) between the relations.
Outer Join	In an outer join, along with tuples that satisfy the matching criteria.
Left Outer Join(\Join)	In the left outer join, operation allows keeping all tuple in the left relation.
Right Outer join(\Join)	In the right outer join, operation allows keeping all tuple in the right relation.
Full Outer Join(\Join)	In a full outer join, all tuples from both relations are included in the result irrespective of the matching condition.

