Concept of relation

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:

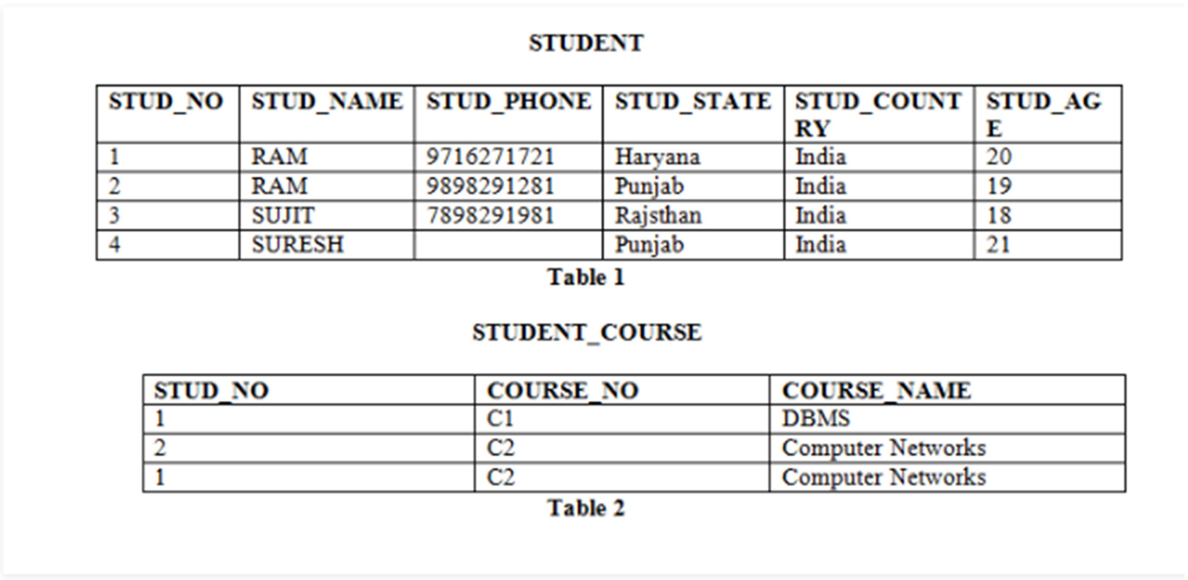
- DB2 and Informix Dynamic Server - IBM
- Oracle and RDB – Oracle
- SQL Server and Access - Microsoft

Relational Model Concepts

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

Notion of Primary and Secondary Keys

Different types of keys in Relational Model



STUDENT

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY	STUD_AGE
1	RAM	9716271721	Haryana	India	20
2	RAM	9898291281	Punjab	India	19
3	SUJIT	7898291981	Rajsthan	India	18
4	SURESH		Punjab	India	21

Table 1

STUDENT_COURSE

STUD_NO	COURSE_NO	COURSE_NAME
1	C1	DBMS
2	C2	Computer Networks
1	C2	Computer Networks

Table 2

Candidate Key:

The minimal set of attribute which can uniquely identify a tuple is known as candidate key. For Example, STUD_NO in STUDENT relation.

- The value of Candidate Key is unique and non-null for every tuple.
- There can be more than one candidate key in a relation. For Example, STUD_NO is candidate key for relation STUDENT.
- The candidate key can be simple (having only one attribute) or composite as well. For Example, {STUD_NO, COURSE_NO} is a composite candidate key for relation STUDENT_COURSE.
-

Super Key:

The set of attributes which can uniquely identify a tuple is known as Super Key. For Example, STUD_NO, (STUD_NO, STUD_NAME) etc.

- Adding zero or more attributes to candidate key generates super key.
- A candidate key is a super key but vice versa is not true.

Primary Key:

There can be more than one candidate key in relation out of which one can be chosen as the primary key. For Example, STUD_NO, as well as STUD_PHONE both, are candidate keys for relation STUDENT but STUD_NO can be chosen as the primary key (only one out of many candidate keys).

Foreign Keys

If an attribute can only take the values which are present as values of some other attribute, it will be a foreign key to the attribute to which it refers. The relation which is being referenced is called referenced relation and the corresponding attribute is called referenced attribute and the relation which refers to the referenced relation is called referencing relation and the corresponding attribute is called referencing attribute. The referenced attribute of the referenced relation should be the primary key for it.

For Example,

STUD_NO in STUDENT_COURSE is a foreign key to STUD_NO in STUDENT relation.

It may be worth noting that unlike, Primary Key of any given relation, Foreign Key can be NULL as well as may contain duplicate tuples i.e. it need not follow uniqueness constraint.

For Example,

STUD_NO in STUDENT_COURSE relation is not unique. It has been repeated for the first and third tuple. However, the STUD_NO in STUDENT relation is a primary key and it needs to be always unique and it cannot be null.

Difference between Primary and Foreign Key

S.NO.	PRIMARY KEY	FOREIGN KEY
1	A primary key is used to ensure data in the specific column is unique.	A foreign key is a column or group of columns in a relational database table that provides a link between data in two tables.
2	It uniquely identifies a record in the relational database table.	It refers to the field in a table which is the primary key of another table.
3	Only one primary key is allowed in a table.	Whereas more than one foreign key are allowed in a table.
4	It is a combination of UNIQUE and Not Null constraints.	It can contain duplicate values and a table in a relational database.
5	It does not allow NULL values.	It can also contain NULL values.
6	Its value cannot be deleted from the parent table.	Its value can be deleted from the child table.
7	It constraint can be implicitly defined on the temporary tables.	It constraint cannot be defined on the local or global temporary tables.

Difference between Primary and Candidate Key:

S.NO	PRIMARY KEY	CANDIDATE KEY
1.	Primary key is a minimal super key. So there is one and only one primary key in a relation.	While in a relation there can be more than one candidate key.
2.	Any attribute of Primary key can not contain NULL value.	While in Candidate key any attribute can contain NULL value.
3.	Primary key can be optional to specify any relation.	But without candidate key there can't be specified any relation.
4.	Primary key specifies the important attribute for the relation.	Candidate specifies the key which can qualify for primary key.
5.	Its confirmed that a primary key is a candidate key.	But Its confirmed that a candidate key can be a primary key.

Difference between Super Key and Candidate Key:

S.NO	SUPER KEY	CANDIDATE KEY
1.	Super Key is an attribute (or set of attributes) that is used to uniquely identifies all attributes in a relation.	Candidate Key is a proper subset of a super key.
2.	All super keys can’t be candidate keys.	But all candidate keys are super keys.
3.	Various super keys together makes the criteria to select the candidate keys.	Various candidate keys together makes the criteria to select the primary keys.
4.	In a relation, number of super keys are more than number of candidate keys.	While in a relation, number of candidate keys are less than number of super keys.
5.	Super key’s attributes can contain NULL values.	Candidate key’s attributes can also contain NULL values.

Structure of relational database

• A relational database is a collection of tables.

– Each table has a unique name.

– Each table consists of multiple rows.

– Each row is a set of values that by definition are related to each other in some way; these values conform to the attributes or columns of the table

– Each attribute of a table defines a set of permitted values for that attribute; this set of permitted set is the domain of that attribute.

• This definition of a database table originates from the pure mathematical concept of a relation, from which the term "relational data model" originates.

– Formally, for a table r with n attributes a1 . . . an, each attribute ak has a domain Dk, and any given row of r is an n-tuple (v1, . . . , vn) such that vk ∈ Dk.

– Thus, any instance of table r is a subset of the Cartesian product D1 × · · · × Dn. – We require that a domain Dk be atomic — that is, we do not consider the elements of Dk to be breakable into subcomponents.

– A possible member of any domain is null — that is, an unknown or non-existent value; in practice, we try to avoid the inclusion of null in our databases because they can cause a number of practical issues.

• This definition is virtually identical to the pure mathematical definition of a relation — the main difference is that, for a database, we assign names to the table's attributes, where mathematical relations only label their attributes by their integer indices 1 . . . n.

• To avoid jargon proliferation, we will stick with the more formal terms "relation" for table and "tuple" for row — which is why we used r to represent a table above instead of t.

• More notations:

– Given a tuple t that belongs to a relation r, we can say that t ∈ R since after all r is a set of tuples.

∗ by "set of tuples" we do mean the mathematical concept of a set; thus order doesn't matter.

∗ Two relations r and s are the same as long as they have the same attributes a1 . . . an with domains D1 . . . Dn and contain as elements the same tuples with the same values, regardless of the order in which these tuples appear.

– To talk about a specific attribute ak of t, we write t[ak].

– This notation also applies to a set of attributes A — the notation t[A] refers to the "sub-tuple" of t consisting only of the attributes in A.

– Alternatively, t[k] can refer to that same attribute of t, as long as we are consistent about how attributes are ordered in the relation.

Relations Algebra and extended relational algebra operations

Basic SQL Relational Algebra Operations

Relational Algebra devided in various groups

Unary Relational Operations

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

Relational Algebra Operations From Set Theory

- UNION (υ)
- INTERSECTION ( ),
- DIFFERENCE (-)
- CARTESIAN PRODUCT (x)

Binary Relational Operations

- JOIN
- DIVISION

SELECT (σ)

The SELECT operation is used for selecting a subset of the tuples according to a given selection condition. Sigma(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 prepositional logic

Example 1

$\sigma_{\text{topic = "Database"}}$ (Student)
Output - Selects tuples from Student where topic = 'Database'.

Example 2

$\sigma_{\text{topic = "Database" and author = "ankit"}}$(Student)
Output - Selects tuples from Student where the topic is 'Database' and 'author' is ankit.

Example 3

$\sigma_{\text{sales} > 50000}$ (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 eliminate 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

$\Pi_{\text{CustomerName, Status}}$ (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.

ρ (a/b)R will rename the attribute 'b' of relation by 'a'.

Union operation (υ)

UNION is symbolized by ∪ 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 <- A ∪ 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 ∪ B gives

Table A ∪ 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

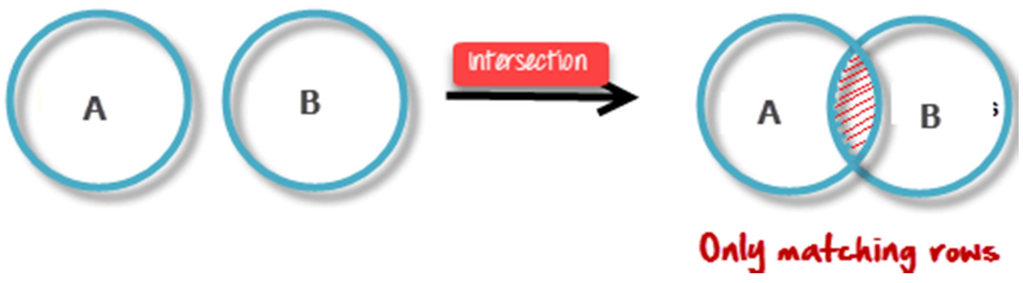
Table A - B	
column 1	**column 2**
1	2

Intersection

An intersection is defined by the symbol ∩

A ∩ 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 ∩ B

Table A ∩ 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

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

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

σ column 2 = '1' (A X 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 ⋈.

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 ⋈ 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 ⋈ 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 (⋈)

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

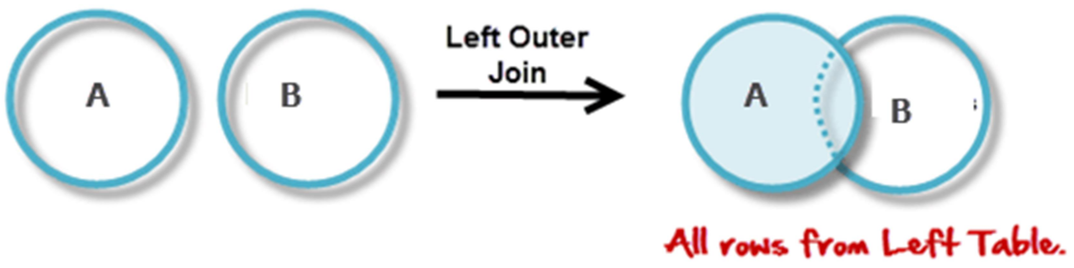
C ⋈ D		
Num	**Square**	**Cube**
2	4	4
3	9	27

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 ⟕ 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

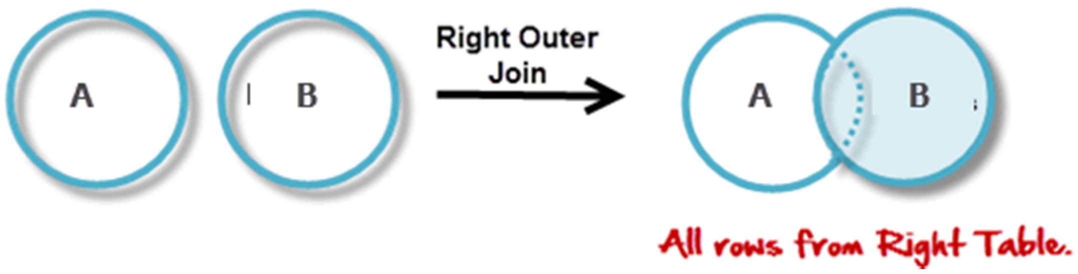
B	
Num	**Cube**
2	8
3	18
5	75

A ⟕ B

A ⋈ B		
Num	**Square**	**Cube**
2	4	4
3	9	9
4	16	-

Right Outer Join: (A ⟖ 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 ⟖ 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

Formation of queries

SELECT

SELECT chooses the fields that you want displayed in your chart. This is the specific piece of information that you want to pull from your database. In the example above, we want to find the *people* who fit the rest of the criteria.
Here is our SQL query:

```
SELECT
   first_name,
   last_name
```

FROM

FROM pinpoints the table that you want to pull the data from. In the earlier section, we found that there were six tables for each of the six states in New England: people_connecticut, people_maine, people_massachusetts, people_newhampshire, people_rhodeisland, and people_vermont. Because we're looking for people in Massachusetts specifically, we'll pull data from that specific table.

Here is our SQL query:

```
SELECT
   first_name,
   last_name
FROM
   people_massachusetts
```

WHERE

WHERE allows you to filter your query to be more specific. In our example, we want to filter our query to include only people with red hair who were born in 2003. Let's start with the red hair filter.

Here is our SQL query:

```
SELECT
   first_name,
   last_name
FROM
   people_massachusetts
WHERE
   hair_color = "red"
```

hair_color could have been part of your initial SELECT statement if you'd wanted to look at all of the people in Massachusetts along with their specific hair color. But if you want to filter to see *only* people with red hair, you can do so in the WHERE statement.

AND

AND allows you to add additional criteria to your WHERE statement. Remember, we want to filter by people who had red hair in addition to people who were born in 2003. Since our WHERE statement is taken up by the red hair criteria, how can we filter by a specific year of birth as well?

That's where the AND statement comes in. In this case, the AND statement is a date property -- but it doesn't necessary have to be. (Note: Be to check the format of your dates with your product team to make sure it is in the correct format.)

Here is our SQL query:

```
SELECT
    first_name,
    last_name
FROM
    people_massachusetts
WHERE
    hair_color = "red"
AND
    birth_date BETWEEN '2003-01-01' AND '2003-12-31'
```

ORDER BY

When you create SQL queries, you shouldn't have to export the data to Excel. The calculation and organization should be done within the query. That's where the "ORDER BY" and "GROUP BY" functions come in. First, we'll look at our SQL queries with the ORDER BY and then GROUP BY functions, respectively. Then, we'll take a brief look at the difference between the two.

Your ORDER BY clause will allow you to sort by any of the fields that you have specified in the SELECT statement. In this case, let's order by last name.

Here is our SQL query:

```
SELECT
    first_name,
    last_name
FROM
    people_massachusetts
WHERE
    hair_color = "red"
AND
    birth_date BETWEEN '2003-01-01' AND '2003-12-31'
ORDER BY
    last_name
;
```

GROUP BY

"GROUP BY" is similar to "ORDER BY," but it will aggregate data that has similarities. For example, if you have any duplicates in your data, iyou can use "GROUP BY" to count the number of duplicates in your fields.

Here is your SQL query:

```
SELECT
    first_name,
    last_name
FROM
    people_massachusetts
WHERE
    hair_color = "red"
AND
    birth_date BETWEEN '2003-01-01' AND '2003-12-31'
GROUP BY
    last_name
;
```

ORDER BY VS. GROUP BY

To clearly show you the difference between an "ORDER BY" statement and a "GROUP BY" statement, let's step outside our Massachusetts example briefly to look at a very simple dataset. Below is a list of four employees' ID numbers and names.

ID	Name
1	Peter
2	John
3	Greg
4	Peter

If we were to use an ORDER BY statement on this list, the names of the employees would get sorted in alphabetical order. The results would look like this:

ID	Name
3	Greg
2	John
1	Peter
4	Peter

If we were to use a GROUP BY statement, the employees would be counted based on the number of times they appeared in the initial table. Note that Peter appeared twice in the initial table. The results would look like this:

#	Name
1	Greg
1	John
2	Peter

With me so far? Okay. Let's return to the SQL query we've been creating about red-haired people in Massachusetts who were born in 2003.

LIMIT

Depending on the amount of data you have in your database, it may take a long time to run the queries. It can be frustrating if you find yourself waiting a long time to run a query that you didn't really want to begin with. If you want to test our query, the LIMIT function is a great one to use because it allows you to limit the number of results you get.

For example, if we suspect there are millions of people who have red hair in Massachusetts, we may want to test out our query using LIMIT before we run it in full to make sure we're getting the information we want. Let's say, for instance, we only want to see the first 100 people.

Here is our SQL query:

```
SELECT
    first_name,
    last_name
FROM
    people_massachusetts
WHERE
    hair_color = "red"
AND
    birth_date BETWEEN '2003-01-01' AND '2003-12-31'
ORDER BY
    last_name
LIMIT
    100
;
```

That’s all we will cover in basics, moving into next lectures we will try to cover more complex queries which would help you understand and work in industry.

Modification of data views

Views

In some cases, it is not desirable for all users to see the entire logical model (i.e., all the actual relations stored in the database.)

• Consider a person who needs to know a customer's loan number but has no need to see the loan amount. This person should see a relation described, in the relational algebra, by Πcustomer-name, loan-number (borrower 1 loan)

• Any relation that is not part of the conceptual model but is made visible to a user as a "virtual relation" is called a view.

View Definition

A view is defined using the create view statement which has the form create view v as where is any legal relational algebra query expression. The view name is represented by v.

• Once a view is defined, the view name can be used to refer to the virtual relation that the view generates.

• View definition is not the same as creating a new relation by evaluating the query epression. Rather, a view definition causes the saving of an expression to be substituted into queries using the view.