Introduction to relational databases

 A relational database is based on the relational model and uses a collection of tables to represent both data and the relationship among those data.

 It also includes DDL and DML commands.
 Relational database was originally defined by Edger Codd at IBM Research centre in 1970.

 In relational database, user only needs to understand logical structure of data, not how it is physically stored.
 Data is represented using tables → consists of rows and columns.
 A relational database simply a collection of tables.

Relational database basic concepts

i) Relations or table

 A relation is defined as set of tuples that have the same attribute
 A relation is usually described as table, which is organized into rows and columns.

ii) Base and derived relation

 In relational database, all data are stored and accessed using relation.
 Relation / table which store data are called base relations.

 Relations which do not store data, but are computed by applying relational operator are called Derived relation.

iii) Tuple / Row / Record

 It holds all information about one item

 Example: all information like roll, name, age, address, age, mark etc., of a particular student.

iv) Field / Column

 A field holds one piece of information about an item.

 Field is column in database table.

 Example: age of all the student.

v) Constraints

 Condition specified for a particular data.

 Constraints restrict data that can be stored in relations.

 Example: User can set constraints that a given integer attribute should be between 1 & 10.

vi) Data type

 Every field in a database table is assigned a data types, which describe the kind of data that can be stored in the field.

vii) Normalization

 Normalization is used to eliminate the duplication of data.

 It is an integral part of the relational model.

 It prevents data manipulation anomalies and loss of data integrity.

INTEGRITY CONSTRAINTS

Data integrity means accuracy, reliability and consistency of data which is stored in the database. Integrity constraints offer a means for ensuring that modifications made to the database by authorized users do not result in the loss of data consistency.

Need of Integrity Constraints

Many users enter data item in database which may results in inconsistencies. Therefore, data insertion, updating and other processes have to be performed in such a way that data integrity is not affected. For this integrity checks are made at data level by checking the value to certain specify rule.

Types of Data Integrity

There are different types of data integrity which are given below.

1. **Domain Integrity Constraint**

   Domain constraint defines that the value taken by the attribute must be the atomic value from its domain. It defines the domain or set of values for an attribute. In domain integrity, you define data type, length or size, here null values are allowed, is the value unique or not for an attribute.

   Consider the following employee table

EMP_ID	NAME	AGE
E101	Amit	30
E102	Aman	32
E103	Akshay	29
E104	Rahul	Q

   Here, the domain constraint is that in EMP_ID E104, in AGE ‘Q” is not allowed because in age attribute, only integer values can be taken.

2. **Entity Integrity Constraint**

   Entity integrity constraint states that primary key can’t be null. It ensures that a specific row in a table can be identified. There can be null values other than primary key fields. Null value is different from zero value or space. The two main properties are:

   - The primary key is not null, no component of the primary key may be set to null.
   - The primary key for a row is unique; it does not match the primary key of any other row in the table.

   The first property ensures that the primary key has meaning, has a value; no component of the key is missing. The uniqueness property ensures that the primary key of each row uniquely identifies it; there are no duplicates. Any operation that creates a duplicate primary key or one containing nulls is rejected. The system enforces entity integrity by not allowing operations (INSERT, UPDATE) to produce an invalid primary key.

For example, consider there is a relation "EMP" Where "Emp_id" is a primary key. These must not contain any null value whereas other attributes may contain null value e.g "Age" in the following relation contains one null value.

EMP_ID	NAME	AGE
E101	Amit	30
E102	Aman	32
E103	Akshay	29
E104	Rahul	NULL

In the above table, in EMP_ID E104, in age attribute, NULL values are not allowed.

3. Referential Integrity Constraint

The referential integrity constraint is specified between two tables and it is used to maintain consistency among rows between two tables. The properties are:

- You can't delete a record from a primary table if matching record exist in a related table.
- You can't enter value in foreign key field of the related table that doesn't exist in the primary key of the primary table.
- You can't change a primary key value in the primary table if that record related record.
- You can enter a value in foreign key, specify that the record is unrelated.
- You cannot update or delete a record of the referenced relation if the corresponding record exists in the referencing relation.
- You cannot insert a record into a referencing relation if the corresponding record does not exist in the referenced relation. For example, consider there are two relations 'EMP' and 'DEPT' .Here, relation 'EMP' references the relation 'Department'.

DEPT Table

DEPT_NO	DEPT_NAME
D10	IT
D10	MGMT
D11	ADMISSION
D11	MARKETING

EMP Table

EMP_ID	NAME	DEPT_NO
E101	Amit	D10
E102	Aman	D11
E103	Akshay	D11
E104	Rahul	D12

Here in the above table, the relation 'EMP' does not satisfy the referential integrity constraint because in relation 'DEPT', no value of primary key specifies department no. 12. So here, referential integrity constraint is violated.

4.Key Integrity Constraint

- In the relational table, a primary key contains a null and unique value in the relational table.
- In the entity set, there can be multiple keys and one key must be primary key.
- Key integrity constraint are the entity set which are used to detect an entity within its entity set (uniquely).

For example, given below is the employee table, which contains attributes like EMP_ID, NAME and DEPT_NO.

EMP Table

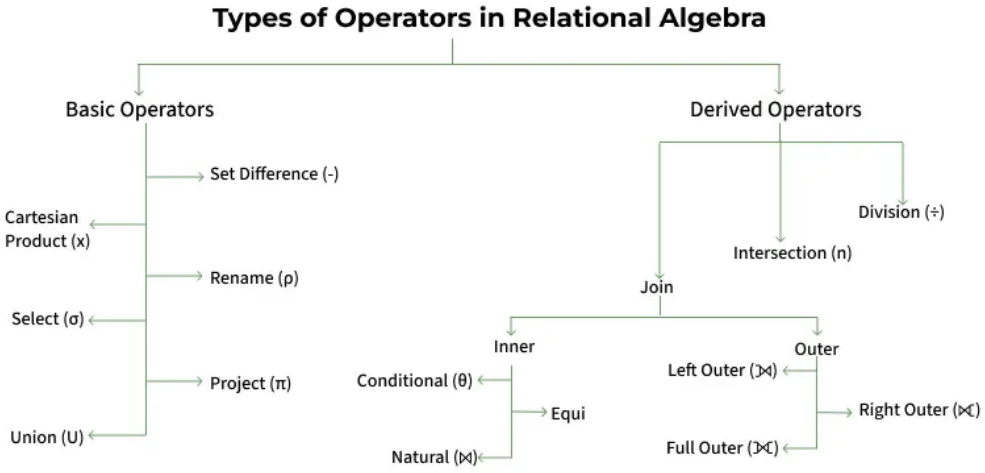
EMP_ID	NAME	DEPT_NO
E101	Amit	D10
E102	Aman	D11
E103	Akshay	D11
E104	Rahul	D12
E101	Ananya	D13
E105	Priya	D14

From the above table, it is clear that EMP_ID is a primary key. It means each entry in this domain should be unique. But it violates the rule.

Relational Algebra

Relational Algebra received attention after relational model of database was published in 1970, by Codd, who proposed this algebra as a foundation for database query languages.

- Relational Algebra is a procedural query language used to query the database tables to access data in different ways.
- In relational algebra, input is a relation(table from which data has to be accessed) and output is also a relation(a temporary table holding the data asked for by the user).



The relational algebra consists of a set of operations that take one or two relations as input and produce a new relation as their result. Some of these operations, such as the select, project, and rename operations, are called unary operations because they operate on one relation. The other operations, such as union, Cartesian product, and set difference, operate on pairs of relations and are, therefore, called binary operations.

Basic Operations There are few operators in any algebraic systems which are basic or primitive, while others are defined in terms of these basic ones. Codd made a similar arbitrary choice for his algebra. There are six primitive operators of relational algebra, proposed by Codd. These are:

1. Selection (Unary)
2. Projection (Unary)
3. Cartesian Product (also called the cross product or cross join) (Binary)
4. Set Union (Binary)
5. Set Difference (Binary)
6. Rename (Unary)

1) The Projection Operation

projection operation of relational algebra works on columns. It basically allows you to pick specific columns from a given relational table based on the given condition and ignoring all the other remaining columns. It is a unary operation.

A projection is mathematically written as

• Notation: πa1…..,an (R),

where a1 ,...,an is a set of attribute names. and R is a relation (name).

Example (book)

ID	TITLE	PRICE	YEAR
1	DBMS	250	2000
2	CP	350	2015
3	PYTHON	450	2009
4	DS	500	2010

1. Display all titles of book available in relation "book"

$$\prod \text{title} (\text{book})$$

TITLE
DBMS
CP
PYTHON
DS

2. Display all titles of book with its price

$$\prod \text{title,price} (\text{book})$$

TITLE	PRICE
DBMS	250
CP	350
PYTHON	450
DS	500

Table 5.3 *Ac_Number and Amount Fields in Emp_eTrans Table*

Ac_Number	Amount
1212	23000
3434	26000
5656	66000
7878	33000
5151	20000

The SQL statements and their corresponding Relational Algebra statements will be written in the Table 5.4 and the result is produced as data provided in 'Emp_eTrans' Table.

Table 5.4 *SQL, Corresponding Relational Algebra Statements and Result*

SQL	Result	Relational Algebra
`select Amount from Emp_eTrans`	Amount 23000 26000 66000 33000 20000	$\mathbf{PROJECT}_{Amount}(Emp\_eTrans)$
`select Ac_Number, Amount from Emp_eTrans`	Ac_Number | Amount 1212 | 23000 3434 | 26000 5656 | 66000 7878 | 33000 5151 | 20000	$\mathbf{PROJECT}_{Ac\_Number, Amount}(Emp\_eTrans)$

Note: There are no duplicate rows in the result.

2)The SELECT Operator

The SELECT operator is used to choose a subset of the tuples(rows) from a relation that satisfies a selection condition, acting as a filter to retain only tuples that fulfils a qualifying requirement.

• The SELECT operator is relational algebra is denoted by the symbol σ (sigma).

• The syntax for the SELECT statement is then as follows:

σ<Selection condition>(R)

• The σ would represent the SELECT command

• The <selection condition> would represent the condition for selection.

• The (R) would represent the Relation or the Table from which we are making a selection of the tuples.

The argument relation is in parentheses after the σ. Predicate is prepositional logic formula which may use connectors like and, or, and not. These terms may use relational operators like:

=, ≠, ≥, <, >, ≤.

EXAMPLE-1

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

The *instructor* relation.

Query-1

To select those tuples of the *instructor* relation where the instructor is in the "Physics" department, we write:

$$\sigma_{dept\_name = "Physics"}(instructor)$$

Output would be as follows:

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
33456	Gold	Physics	87000

Query-2

To find the instructors in Physics with a salary greater than ₹90,000, we write:

$$\sigma_{dept\_name = "Physics" \wedge salary > 90000}(instructor)$$

EXAMPLE-2

Example (book)

ID	TITLE	PRICE	YEAR
1	DBMS	250	2000
2	CP	350	2015
3	PYTHON	450	2009
4	DS	500	2010

1. Display book having price 500

$$\sigma_{price=500}(book)$$

ID	TITLE	PRICE	YEAR
4	DS	500	2010

2. Display all books having price greater than 300

$$\sigma_{price>300}(book)$$

ID	TITLE	PRICE	YEAR
2	CP	350	2015
3	PYTHON	450	2009
4	DS	500	2010

3. Display all books having price greater than 300 and year before 2010

$$\sigma_{price>300 \text{ and } year>2010}(book)$$

ID	TITLE	PRICE	YEAR
3	PYTHON	450	2009

EXAMPLE-3

To implement the SELECT statement in SQL, we take a look at an example in which we would like to select the EMPLOYEE tuples whose employee number is 7, or those whose date of birth is before 1980…

σempno=7(EMPLOYEE)

σdob<'01-Jan-1980′(EMPLOYEE)

The **SQL** implementation would translate into:

SELECT empno FROM EMPLOYEE WHERE empno=7

SELECT dob FROM EMPLOYEE WHERE DOB < '01-Jan-1980′

Composition of Relational Operations

The fact that the result of a relational operation is itself a relation is important. Consider the more complicated query "Find the names of all instructors in the Physics department." We write:

$$\Pi_{name}\ (\sigma_{dept\ name\ =\ \text{“Physics”}}\ (instructor))$$

In general, since the result of a relational-algebra operation is of the same type(relation) as its inputs, relational-algebra operations can be composed together into a relational-algebra expression.

3)The UNION Operator

The union operation : The union operation is a binary operation that is used to find union of relations. Here relations are considered as sets. So, duplicate values are eliminated. It is denoted by (∪).

Conditions for union operation : There are two necessary conditions for union operation.

(i) Both the relations have same number of attributes.

(ii) Data types of their corresponding attributes must be same.

Two relations are said to be union compatible if they follow the above two conditions.

SQL UNION Syntax

SELECT column_name(s) FROM table1

UNION

SELECT column_name(s) FROM table2;

Note: The UNION operator selects only distinct values by default. To allow duplicate values,

use the ALL keyword with UNION.

SQL UNION ALL Syntax

SELECT column_name(s) FROM table1

UNION ALL

SELECT column_name(s) FROM table2;

Consider the following relations

Employee

EID	Name	Salary
1E	John	10,000
2E	Ramesh	5,000
3E	Smith	8,000
4E	Jack	6,000
5E	Nile	15,000

Student

SID	Name	Fees
1S	Smith	1,000
2S	Vijay	950
3S	Gaurav	2,000
4S	Nile	1,500
5S	John	950

Example1- If you want to find the names of all employees and names of all students together then the query is $\boldsymbol{Name\ (Employee) \cup \pi Name\ (Student)}$

Name
John
Ramesh
Smith
Jack
Nile
Vijay
Gaurav

Set intersection operation : Set intersection is used to find common tuples between two relations. It is denoted by (∩). Rules of set union operations are also applicable here.

If you want to find all the employees from Relation Employee those are also students. Then the query, is

πName (Employee) ∩ πName (Student)

Name
John
Smith
Nile

Set-difference operation **:** Set-difference operation is a binary operation which is used to find tuples that are present in one relation but not in other relation. It is denoted by (—). It removes the common tuples of two relations and produce a new relation having rest of the tupels of first relation.

Ex. If you want the names of those employees that are not students, then the query, is π Name (Employee) — π Name (Student)

$$\pi_{Name}\,(Employee) - \pi_{Name}\,(Student)$$

Output of the query is

Name
Ramesh
Jack

5) Rename

In relational algebra user can rename a relation/attributes or both.

$\rho_{s(new\_attribute\_name)}\,(R)$
$\rho_{s}\,(R)$
$\rho_{(new\_attribute\_name)}\,(R)$

- ρ → Symbol of rename operation
- $_S$ → name of the new relation to be renamed
- < *new_attribute_name* > → new attributes name
- R → Relation / Table

›le

Example Consider a relation "book" with attribute (ID, title, author, price)

1. Rename both relation and attribute name

$$\rho_{temp(id1,title1,author1,price1)}(book)$$

Temp

id1	title1	author1	price1

2. Rename table name

$$\rho_{table}(book)$$

table

id	title	author	price

3. Rename only attributes

$$\rho_{(token,title,name,amount)}(book)$$

Book

token	Title	name	amount

6) **Cartesian product**

In relational algebra, the **Cartesian product** (also known as the **cross product**) is a binary operation that combines every row of one relation with every row of another relation. The result of a Cartesian product between two relations is a new relation that contains all possible combinations of tuples (rows) from the two relations.

Notation:

The Cartesian product between two relations **R** and **S** is denoted as:

$$R \times S$$

Result:

- If **R** has m tuples and n attributes, and **S** has p tuples and q attributes, the resulting relation from R×S will have m×p tuples and n+q attributes.
- The attributes in the resulting relation will be the union of the attribute sets of **R** and **S**.

Example:

Let's consider two relations:

- **R** (ID, Name) with data:

ID	Name
1	Alice
2	Bob

S (Age, City) with data:

Age	City
25	London
30	Paris

The **Cartesian product** R×S would be:

ID	Name	Age	City
1	Alice	25	London
1	Alice	30	Paris
2	Bob	25	London
2	Bob	30	Paris

As you can see, each row in **R** is paired with every row in **S**, resulting in a total of 2×2=4 tuples in the Cartesian product.