**Domains :**

<https://www.geeksforgeeks.org/domain-constraints-in-dbms/>

**Tuples :**

<https://www.geeksforgeeks.org/tuple-in-dbms/>

**Attributes :**

<https://www.geeksforgeeks.org/attributes-in-dbms/>

**Relations :**

<https://www.geeksforgeeks.org/what-is-relationship-in-dbms/>

**What are the characteristics of a relation**

**Relation**: In the context of relational databases, a relation refers to a table that stores data in rows and columns. It's essentially a mathematical concept that represents a set of tuples, where each tuple is a collection of related data values.

**Domain**: A domain is the set of all possible values that an attribute (or column) in a relation can have. It defines the range of valid values for that attribute. For example, a domain for an attribute "Age" might be all non-negative integers.

**Tuple**: A tuple, also known as a row or record, is a single entry in a relation, representing a complete set of data values corresponding to one entity or record. Each tuple in a relation is unique.

**Degree**: The degree of a relation is the number of attributes (or columns) it contains. It represents the complexity or structure of the relation. For example, if a relation has attributes for "Name", "Age", and "Gender", its degree would be 3.

**Attribute**: An attribute is a named column of a relation, representing a particular property or characteristic of the entities described by the relation. Each attribute has a domain that defines the type of values it can hold. For example, in a relation representing employees, attributes might include "EmployeeID", "Name", "Salary", etc.

**Cardinality**: Cardinality refers to the number of tuples (rows) in a relation. It represents the size or count of the relation. Cardinality can also refer to the relationship between two entities in a database, indicating how many instances of one entity are associated with each instance of another entity. For example, in a one-to-many relationship between "Department" and "Employee", the cardinality might indicate how many employees are associated with each department.

Top of Form

**Keys :**

<https://www.geeksforgeeks.org/types-of-keys-in-relational-model-candidate-super-primary-alternate-and-foreign/>

**Relational database :**

<geeksforgeeks.org/rdbms-full-form/>

**Schemas :**

<https://www.geeksforgeeks.org/database-schemas/>

**Integrity constraints :**

<https://www.geeksforgeeks.org/dbms-integrity-constraints/>

**Relational Query languages::**

<https://www.geeksforgeeks.org/relational-query-language-in-dbms/>

**various joins :**

<https://www.geeksforgeeks.org/sql-join-set-1-inner-left-right-and-full-joins/>

**indexing :**

<https://www.geeksforgeeks.org/indexing-in-databases-set-1/>

**Relational algebra :**

<https://www.geeksforgeeks.org/introduction-of-relational-algebra-in-dbms/>

**relational calculus :**

Relational calculus are two types -

<https://www.geeksforgeeks.org/tuple-relational-calc-ulus-trc-in-dbms/>

<https://www.geeksforgeeks.org/domain-relational-calculus-in-dbms/>

**Explain select, project, Cartesian product, join, division, union, intersection etc. with suitable examples.**

**Select (σ)**: The select operation retrieves tuples from a relation that satisfy a given condition.

Example: Consider a relation "Students" with attributes (ID, Name, Age). A select operation σ\_(Age>20) would retrieve all tuples from "Students" where the age is greater than 20.

**Project (π)**: The project operation extracts specific attributes from a relation, discarding the others.

Example: Continuing with the "Students" relation, a project operation π\_(Name, Age) would extract only the "Name" and "Age" attributes, discarding the "ID".

**Cartesian Product (×)**: The Cartesian product operation combines every tuple from one relation with every tuple from another relation.

Example: Consider two relations "Students" (ID, Name) and "Courses" (CourseID, CourseName). The Cartesian product Students × Courses would produce a new relation where each student is paired with every course.

**Join (⋈)**: The join operation combines tuples from two relations based on a common attribute.

Example: Using the same "Students" and "Courses" relations, if both have a common attribute "ID", a natural join Students ⋈ Courses would combine tuples where the student ID matches the course ID.

**Division (÷)**: The division operation is used to find tuples that are related to all tuples in another relation.

Example: Consider two relations "Books" (BookID, Title) and "Authors" (AuthorID, Name). If we want to find authors who have written all books, we can perform the division Authors ÷ Books.

**Union (∪)**: The union operation combines tuples from two relations, removing duplicates.

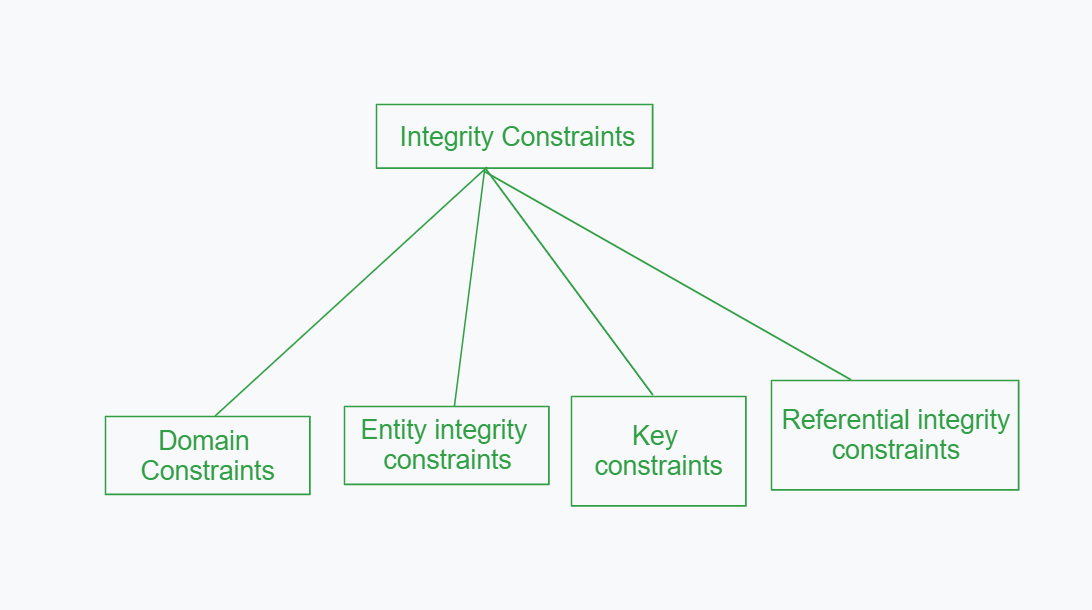
Example: Suppose we have two relations "MaleStudents" and "FemaleStudents". The union operation MaleStudents ∪ FemaleStudents would merge both relations, removing duplicate tuples.

**Intersection (∩)**: The intersection operation retrieves tuples that appear in both relations.

Example: Continuing with the "MaleStudents" and "FemaleStudents" relations, the intersection operation MaleStudents ∩ FemaleStudents would retrieve tuples that represent students who are both male and female (if such a case is applicable).

**Explain integrity constrains with example.**

Integrity constraints are the set of predefined rules that are used to maintain the quality of information. Integrity constraints ensure that the data insertion, data updating, data deleting and other processes have to be performed in such a way that the data integrity is not affected.



**Domain Constraints**

These are defined as the defination of valid set of values for an atribute. The data type of domain include string, char, time, integer, date, currency etc. The value of the attribute must be avialable in comparable domains.

**Example:**

| Student\_Id | Name | Semester | Age |
| --- | --- | --- | --- |
| 21CSE100 | Ramesh | 5th | 20 |

**Entity Integrity Constraints**

Entity integrity constraints state that primary key can never contain null value because primary key is used to determine individual rows in a relation uniquely, if primary key contains null value then we cannot identify those rows. A table can contain null value in it except primary key field.

**Example:**

It is not allowed because it is containing primary key as NULL value.

| Student\_id | Name | Semester | Age |
| --- | --- | --- | --- |
| 21CSE101 | Ramesh | 5th | 20 |
| 21CSE102 | Kamlesh | 5th | 21 |

**Key Constraints**

**Keys are the entity set that are used to identify an entity within its**[**entity set**](https://www.geeksforgeeks.org/difference-between-entity-entity-set-and-entity-type/)**uniquely. An entity set can contain multiple keys, bit out of them one key will be primary key. A primary key is always unique, it does not contain any null value in table.**

**Example:**

| Student\_id | Name | | Semester | | Age | |
| --- | --- | --- | --- | --- | --- | --- |
| 21CSE101 | | Ramesh | | 5th | | 20 | |
| 21CSE102 | | Kamlesh | | 5th | | 21 | |

**Referential integrity constraints**

It can be specified between two tables. In case of referential integrity constraints, if a [Foreign key](https://www.geeksforgeeks.org/foreign-key-constraint-in-sql/) in Table 1 refers to Primary key of Table 2 then every value of the Foreign key in Table 1 must be null or avialable in Table 2.

Example:

Here, in below example Block\_No 22 entry is not allowed because it is not present in 2nd table.

**Write the commands of DDL, DML, DCL and TCL.**

**DDL (Data Definition Language)**

[**CREATE**](https://www.geeksforgeeks.org/sql-create/)**:** This command is used to create the database or its objects (like table, index, function, views, store procedure, and triggers).

[**DROP**](https://www.geeksforgeeks.org/sql-drop-truncate/): This command is used to delete objects from the database.

[**ALTER**](https://www.geeksforgeeks.org/sql-alter-add-drop-modify/)**:**This is used to alter the structure of the database.

[**TRUNCATE**](https://www.geeksforgeeks.org/sql-drop-truncate/)**:**This is used to remove all records from a table, including all spaces allocated for the records are removed.

[**COMMENT**](https://www.geeksforgeeks.org/sql-comments/): This is used to add comments to the data dictionary.

[**RENAME**](https://www.geeksforgeeks.org/sql-alter-rename/)**:**This is used to rename an object existing in the database.

**DQL (Data Query Language)**

[**SELECT**](https://www.geeksforgeeks.org/sql-select-clause/)**:**It is used to retrieve data from the database.

**DML(Data Manipulation Language)**

[**INSERT**](https://www.geeksforgeeks.org/sql-insert-statement/): It is used to insert data into a table.

[**UPDATE**](https://www.geeksforgeeks.org/sql-update-statement/): It is used to update existing data within a table.

[**DELETE**](https://www.geeksforgeeks.org/sql-delete-statement/)**:** It is used to delete records from a database table.

[**LOCK:**](https://www.geeksforgeeks.org/sql-lock-table/)Table control concurrency.

**CALL:**Call a PL/SQL or JAVA subprogram.

**EXPLAIN PLAN:** It describes the access path to data.

**What are the aggregate functions of SQL?**

An aggregate function is a function that performs a calculation on a set of values, and returns a single value.

Aggregate functions are often used with the GROUP BY clause of the SELECT statement. The GROUP BY clause splits the result-set into groups of values and the aggregate function can be used to return a single value for each group.

The most commonly used SQL aggregate functions are:

MIN() - returns the smallest value within the selected column

MAX() - returns the largest value within the selected column

COUNT() - returns the number of rows in a set

SUM() - returns the total sum of a numerical column

AVG() - returns the average value of a numerical column

**Write short note on Primary Index and Secondary Index.**

**Primary Index:**

A primary index is an index created on the primary key column(s) of a table.

The primary key uniquely identifies each row in the table, so the primary index provides fast access to individual rows.

Typically, the primary index is implemented as a clustered index, where the physical order of rows in the table corresponds to the order of the primary key values.

In a clustered index, the data is physically stored in the order of the index key, which reduces the need for additional I/O operations when retrieving data.

Each table can have only one primary index.

**Secondary Index:**

A secondary index is an index created on columns other than the primary key column(s) of a table.

Unlike the primary index, a secondary index does not define the physical order of rows in the table.

Secondary indexes provide fast access to rows based on the values of columns other than the primary key.

They are useful for accelerating queries that filter, sort, or join data based on non-primary key columns.

While a table can have multiple secondary indexes, each index adds overhead to data modification operations (such as inserts, updates, and deletes) because the index must be updated to reflect changes in the underlying data.

**Q8. Consider the employee data. Give an expression in SQL and Relational algebra for the following query:**

**Employee (employee-name, street, city)**

**Works (employee-name, company-name, salary)**

**Company (company-name, city)**

**Manages (employee-name, manager-name)**

1. **Find the name of all employees who work for State Bank.**

SELECT employee\_name

FROM Employee e, Works w

WHERE e.employee\_name = w.employee\_name

AND w.company\_name = 'State Bank';

*π*employee-name​(*σ*company-name=′*StateBank*′​(*Works*⋈*Employee*))

1. **Find the names and cities of residence of all employees who work for HDFC Bank.**

SELECT e.employee\_name, e.city

FROM Employee e, Works w, Company c

WHERE e.employee\_name = w.employee\_name

AND w.company\_name = c.company\_name

AND w.company\_name = 'HDFC Bank';

*π*employee-name, city​(*σ*company-name=′*HDFCBank*′​(*Works*⋈*Employee*⋈*Company*))

1. **Find the all employees who do not work for State Bank and ICICI Bank.**

SELECT employee\_name

FROM Employee

WHERE employee\_name NOT IN (

SELECT employee\_name

FROM Works

WHERE company\_name IN ('State Bank', 'ICICI Bank')

);

*π*employee-name​(*Employee*)−*π*employee-name​(*σ*company-name=′*StateBank*′∨company-name=′*ICICIBank*′​(*Works*))

1. **Find the all employees who belongs same city where they are working.**

**SELECT e.employee\_name**

**FROM Employee e, Works w**

**WHERE e.employee\_name = w.employee\_name**

**AND e.city = (**

**SELECT city**

**FROM Company c**

**WHERE c.company\_name = w.company\_name**

**);**

*π*employee-name​(*σ*Employee.city = Company.city​(*Employee*×*Works*×*Company*))

**v) Find the employee name and his salary whose manager is Ravi.**

SELECT w.employee\_name, w.salary

FROM Works w, Manages m

WHERE w.employee\_name = m.employee\_name

AND m.manager\_name = 'Ravi';

*π*employee-name, salary​(*σ*Manages.manager-name=’Ravi’​(*Works*⋈*Manages*)

**Q9. We have following relations:**

**Emp (empno, ename, jobtitle, managerno, hiredte, sal, comm, deptno)**

**Dept(deptno, dname, loc)**

**Answer the following:**

**i) Write SQL and relational algebra query to find the employees working in the department 10,20,30.**

SELECT \*

FROM Emp

WHERE deptno IN (10, 20, 30)

*π*empno, ename, jobtitle, managerno, hiredte, sal, comm, deptno​(*σ*deptno=10∨deptno=20∨deptno=30​(*Emp*))

**ii) Write SQL query to find employees whose names starts with letter A or a.**

SELECT \*

FROM Emp

WHERE ename LIKE 'A%' OR ename LIKE 'a%';

**iii) Write SQL query find the employee and his department name.**

SELECT e.\*, d.dname

FROM Emp e

JOIN Dept d ON e.deptno = d.deptno;

*π*Emp.\*, Dept.dname​(Emp⋈Dept)

1. **Find the employees who are working in Smith’s department.**

SELECT e.\*

FROM Emp e

JOIN Emp m ON e.managerno = m.empno

WHERE m.ename = 'KING';

*π*Emp​(*σ*Emp.deptno=(*π*Dept.deptno​(*σ*dname=′*Smith*′​(Dept)))​(Emp))

1. **Find the employees who get more than Allen’s salary.**

SELECT \*

FROM Emp

WHERE sal > (

SELECT sal

FROM Emp

WHERE ename = 'Allen'

);

*π*Emp​(*σ*sal>(*π*sal​(*σ*ename=′*Allen*′​(Emp)))​(Emp))

1. **Find the employees whose manager is KING.**

SELECT e.\*

FROM Emp e

JOIN Emp m ON e.managerno = m.empno

WHERE m.ename = 'KING';

*π*Emp​(*σ*Emp.managerno = KING.empno​(Emp⋈*ρ*KING​(Emp)))

1. **Display the employees who are getting maximum salary in each department.**

SELECT e.\*

FROM Emp e

JOIN SELECT deptno, MAX(sal) AS max\_sal

FROM Emp

GROUP BY deptno

) max\_salaries ON e.deptno = max\_salaries.deptno AND e.sal = max\_salaries.max\_sal;

**Q10. Consider the following relations (primary keys are underlined)?**

1. **Account (acc-no, balance, branch-name)**
2. **Depositor (acc-no, cust-no)**
3. **Customer (cust-no, name, city)**
4. **Loan (loan-no, amt, branch-name)**

**v. Borrower (cust-no, loan-no)**

**Solve the following queries using SQL.**

1. **Find all customer-no and loan-no who have a loan at the ‘Perryridge’ branch.**

SELECT d.cust\_no, b.loan\_no

FROM Borrower b

JOIN Loan l ON b.loan\_no = l.loan\_no

WHERE l.branch\_name = 'Perryridge';

*π*Depositor.cust-no, Borrower.loan-no​(*σ*Loan.branch-name = ’Perryridge’​(Depositor⋈Borrower⋈Loan))

1. **Find all customer who have an account but no loan at the bank.**

SELECT c.cust\_no, c.name

FROM Customer c

LEFT JOIN Depositor d ON c.cust\_no = d.cust\_no

WHERE d.cust\_no IS NULL;

*π*Customer.cust-no​(Customer−*π*Depositor.cust-no​(Depositor))

1. **Find branch-name and average account balance where average account balance is greater than 1000.**

SELECT a.branch\_name, AVG(a.balance) AS avg\_balance

FROM Account a

GROUP BY a.branch\_name

HAVING AVG(a.balance) > 1000;

*π*Account.branch-name, AVG(Account.balance)​(*γ*branch-name,AVG(balance)​(Account))

**Q11. Consider the employee data. Give an expression in SQL and Relational algebra for the following query:**

**Employee (employee-name, street, city)**

**Works (employee-name, company-name, salary)**

**Company (company-name, city)**

**Manages (employee-name, manager-name)**

1. **Find the names employees who work for first bank cooperation.**

SELECT employee\_name

FROM Employee e, Works w

WHERE e.employee\_name = w.employee\_name

AND w.company\_name = 'First Bank Corporation';

*π*employee-name​(*σ*company-name=′*FirstBankCorporation*′​(Employee⋈Works))

1. **Find the name , street addresssse and cities of residence of all employees who work for First Bank Corporation and earn more than 200000 per annum.**

SELECT e.employee\_name, e.street, e.city

FROM Employee e, Works w

WHERE e.employee\_name = w.employee\_name

AND w.company\_name = 'First Bank Corporation'

AND w.salary > 200000**;**

*π*employee-name, street, city​(*σ*company-name=′*FirstBankCorporation*′∧salary>200000​(Employee⋈Works))