3 SQL

3.1 SQL

- SQL stands for Structured Query Language.
- We use SQL as commercial relational database language.
- Initially, SQL was known as Structured English QUERY Language(SEQUEL). It was designed and implemented by IBM Research.
- SQL is a standard way to add, delete, modify and obtain the data.
- SQL is a declarative programming language. It means we need to declare what we want, but we need not focus on how to get data. We do not specify step by step procedure to obtain data.

Features of SQL languages are:

Embedded SQL:
Embedded SQL is a feature using which a host language can call an SQL code.

Dynamic SQL:
In dynamic SQL, we perform queries at run time.

Security: SQL provides mechanisms to control user's access to data objects such as tables and views.

Transaction management:
A user is allowed to explicitly control how a transaction to be executed using various commands.

Execution of SQL query:
The DBMS performs a number of steps while executing a query. The conceptual view of this process is shown below.

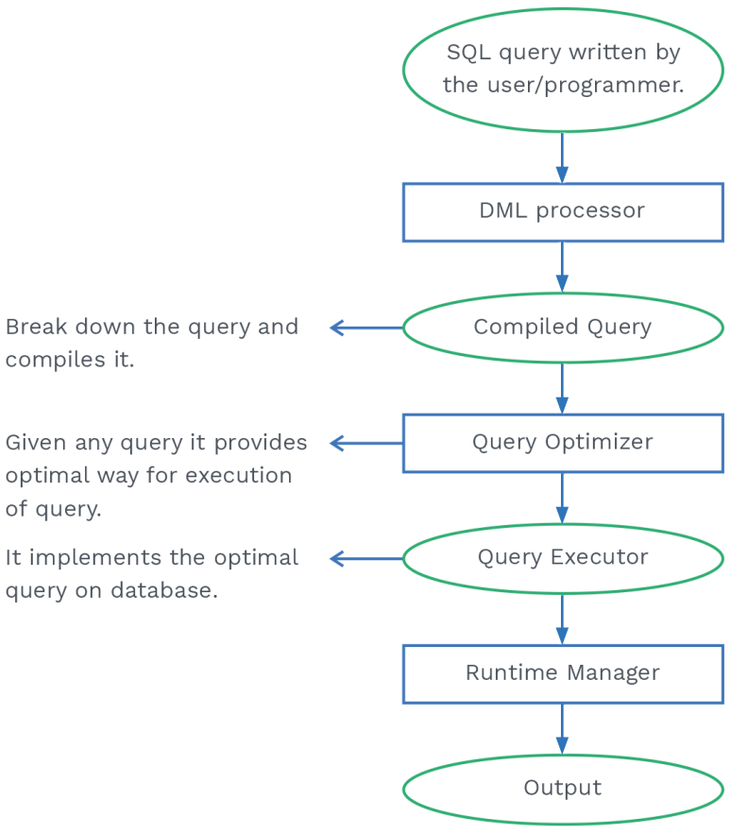


Fig. 3.1 Execution of SQL Query

- Terms like table, row and column in SQL are used for the relational model terms like relation, tuple and attribute, respectively. We can use the corresponding terms interchangeably.
- SQL uses certain commands. These SQL commands are mainly:
  1) DDL: Data Definition Language
  2) DML: Data Manipulation Language
  3) DQL: Data Query Language
  4) DCL: Data Control Language

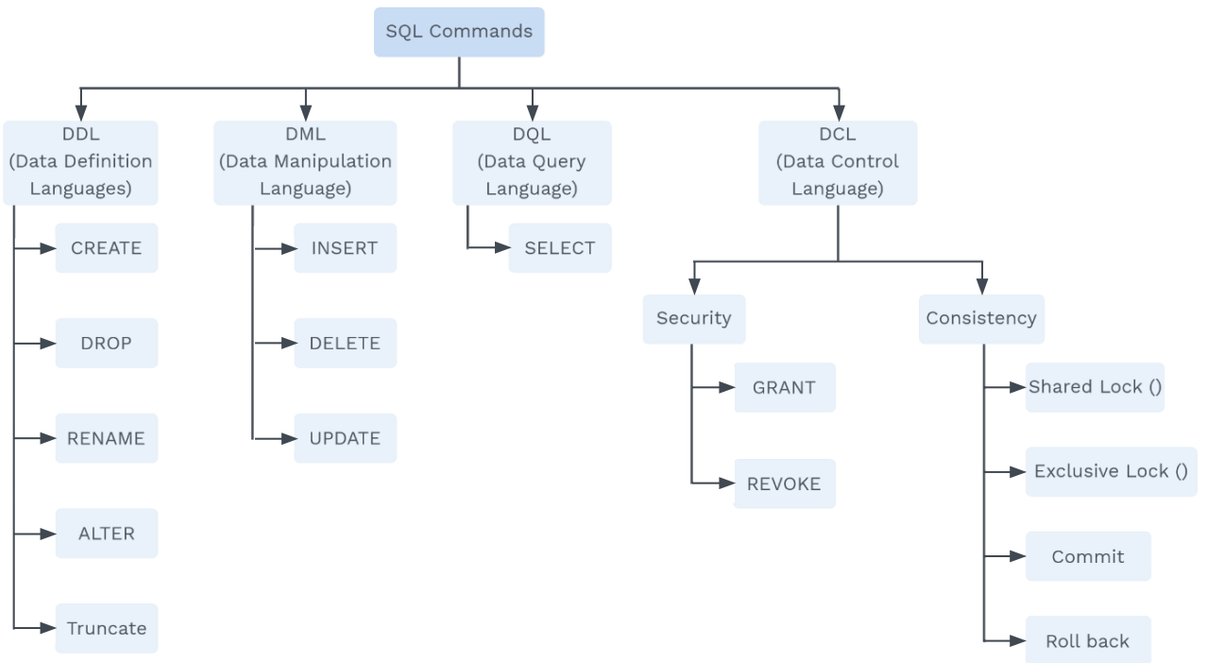


Fig. 3.2 SQL Commands

DDL (Data Definition Language):

- DDL is used for creating, deleting and modifying tables.
- Common examples of DDL statements include CREATE, ALTER and DROP.

Example: DROP TABLE Employees

DML (Data Manipulation Language):

- DML is used for inserting, deleting and modifying data in database.

Example: INSERT INTO Employees (Fname, Lname) VALUES ('Shikha', 'Sharma')

DQL (Data Query Language):

- Data query language performs queries on the data within schema objects.
- The DQL commands are used to get the relation based on whatever query we pass to it.
- To retrieve data from Database, we use SELECT Command.

DCL (Data Control Language):

- Data Control Language feature is used to control access to data stored in a database.
- GRANT and REVOKE are examples of DCL.
- GRANT: It is used to allow specified users to perform specific tasks.
- REVOKE: It is used to remove the user accessibility to database objects.

Basic structure of SQL queries:

- The basic structure of an SQL queries:

SELECT, **FROM** and **WHERE:**

- The SELECT clause projects the output desired attributes. It bears similarity with the projection operation of relational algebra.
- The FROM clause bears resemblance with the cartesian product of the tables involved.
- The WHERE clause deals with the specification of the condition to be followed while fetching a record from the resultant of FROM clause. The work which is done by WHERE clause is the same as the work done by selection operation of the relational algebra.
- A typical SQL query has the form:
  SELECT <attributes list>
  FROM <table list>
  WHERE <condition>;
  The query is equivalent to the relational-algebra expression:

  $$(|A_1| \cup |A_2|) = |A_1| + |A_2| - |A_1| \cap |A_2| = 2^{n-1} + 2^{n-1} - 2^{n-2} = 2^n - 2^{n-2}$$

  Each A_i is an attribute, and each r_i is a relation and c is a condition. The only difference is the result of a relational algebra expression do not produce duplicate rows, but the result of the SQL query might produce more than one copy of some rows.
- SQL first does the cartesian product of the tables present in the FROM clause, then it performs a relational-algebra selection using the condition specified in the WHERE clause and then project the result onto the attributes of the SELECT clause.

The SELECT clause:

Syntax: SELECT < attribute-list > FROM < table-list>;
Let us realise the above clauses through this query:
“Find the names of all employees in the Employee relation”.

Sol: SELECT Emp-name FROM Employee;

- This query will result a table that consists of a single attribute with the Empname as a heading.
- SQL permits duplicates in the results.
- Thus, the above query will give each employee name once for every row in which it appears in the Employee table.
- There are cases where we want to eliminate duplicates; for that, we use keyword “distinct” after “select”.

Example: SELECT DISTINCT Emp-name FROM Employee;

- SELECT clause might also contain arithmetic expressions that involve the operators +, –, /, and * operating on constants or attributes of tuples.

Example: SELECT Emp-name, Salary * 200 FROM Employee;
This will return a table containing columns Emp-name and Salary with the column Salary is multiplied by 200.

The WHERE clause:
Let us consider the query on relation Employee:
"Find the names of all employees who works for department CS".
In SQL: SELECT Emp-name FROM Employee WHERE dept = CS;

The FROM clause:
The FROM clause by itself defines a cartesian product of the tables in the clause.
Let us consider the query in relation to Employee and Department:
"Retrieve the employee id and name of the employees who works for the IT department".

Department

Dept_id	Dept_name
1	CSE
2	IT
3	ECE
4	ME

Employee

Emp_id	Dept_no	Emp_name
1	3	Raju
2	4	Rima
3	2	Puja
4	4	Nisha
5	1	Koyal

Table 3.1

In SQL: SELECT Emp_id, Emp_name FROM Employee, Department WHERE Dept_no = Dept_id and Dept_name = 'IT';

- The above SQL query will give the result as {3, puja}
  Consider the following table definitions:
  Shopkeepers (Sid: integer, Shname: string, Rating: integer, Age: integer)

Sid	Shname	Rating	Age
1	Ramu	7	20
2	Rupa	2	29
4	Kajal	4	22
5	Koyal	5	23
10	Mahima	8	25
9	Gopal	9	24
7	Rupa	3	29

Table 3.2

PRACTICE QUESTIONS

Q1 **Find the names and ages of all Shopkeepers.**

Sol: SELECT Shname, Age FROM Shopkeepers;

Shname	Age
Ramu	20
Rupa	29
Kajal	22
Koyal	23
Mahima	25
Gopal	24
Rupa	29

Q2 **Find the names and ages of all Shopkeepers having no duplicate names and ages.**

Sol: SELECT DISTINCT Shname, Age from Shopkeepers;

Output:

Shname	Age
Ramu	20
Rupa	29
Kajal	22
Koyal	23
Mahima	25
Gopal	24

We will all distinct <Shname, Age> if two or more Shopkeepers have the same name and age, the answer will contain only one pair.

Q3 **Find all the Shopkeepers with a rating above 5.**

Sol: SELECT * FROM Shopkeepers WHERE rating > 5;

Output:

Sid	Shname	Rating	Age
1	Ramu	7	20
10	Mahima	8	25
9	Gopal	9	24

Note:

When we want to retrieve all columns, SQL provides a convenient shorthand: SELECT *. This notation is useful for interactive querying but is poor for queries that are meant to be reused and maintained.

Note:

Hey Learners!!!
Now, we will discuss DISTINCT keyword.
By default, an SQL query contains duplicates in the result. In order to get the distinct result, we use DISTINCT Keyword.
We use DISTINCT to eliminate duplicate tuples.

The RENAME operation:

- SQL provides a mechanism for renaming both relations and attributes.
- It uses the AS clause.

Example: Consider the following given relation Employee and for each employee retrieve employee's id, name, salary and the name of his/her immediate supervisor.

Employee

Emp_id	Dept_id	Sup_id	Sal	Emp_name
1	1	7	1100	Sindhu
2	3	4	2200	Somya
3	1	7	2100	Santosh
4	4	7	3000	Megha
5	2	4	2000	Murali
6	1	7	5000	Ravi
7	1	3	1800	Rupa

Table 3.3

Sol: SELECT E. Emp_id AS "Employee_id", E. sal AS "Salary", E. Emp_name AS "Employee_name", S. Emp_name AS "Supervisor_name" FROM Employee AS E, Employee AS S WHERE E. Sup_id = S. Emp_id;

Rack Your Brain

Consider the following relation Department and Employee. Compute an SQL query that lists out employees names and department names.

Department

Dept_id	Dept_name
1	Accouting
2	Sales
3	Research
4	Aviation

Employee

Emp_id	Dept_id	Salary	Emp_name
1	1	1100	Koyal
2	3	2200	Komal
3	4	2100	Raj
4	2	3000	Sonu

String operation:

- Pattern matching is the most frequently used operation on strings that uses operator named as LIKE.
- There are two special characters that is used to describe patterns:
  1) % (percent): The % character matches any substring.
  2) _ (underscore): The underscore (_) character matches any character.
- 'A%' matches any string that starts (begins) with 'A'.
- '%ai%' matches any string containing "ai" as a substring. For example, 'Rain', 'Paid', 'Sail' etc.
- '_ _' matches any string of exactly two characters.
- '_ _ _ %' matches any string of at least three characters.
- SQL uses the LIKE comparison operator to express patterns.

Note:
Patterns are case sensitive, i.e. uppercase characters are different than lowercase characters or vice-versa.

Example: Following is the given relation Student.

Name	Marks	Rank	Email
Sindhu	78	188	abc@gmail.com
Somya	92	15	def@gmail.com
Komal	48	1500	mno@gmail.com

Table 3.4

PRACTICE QUESTIONS

Q1 **Retrieve the name of all the students whose name starts with 'S'.**

Sol: SELECT Name FROM Student WHERE Name LIKE 'S%';

Name
Sindhu
Somya

Q2 **Display the name of the students who secure 2 digit rank.**

Sol: SELECT Name FROM Student WHERE Rank LIKE '_ _';

Name
Somya

Q3 **Retrieve name, marks of all the students whose name includes substring 'ind'.**

Sol: SELECT Name, Marks FROM Student WHERE Name LIKE '%ind%';
This will result a relation containing attribute (Name, Marks) as:

Name	Marks
Sindhu	78

Note:

- SQL uses escape character to include the special characters (% and _).
- When the escape character is used just before a special pattern character, then the special pattern character will be treated as a normal character.

Example: SELECT Name FROM Student WHERE Marks LIKE '90 \ %';
gives names of the student whose marks is 90%.

Rack Your Brain

Consider the following relation Sailor.

Shopkeepers

Sid	Sname	Rating	Age
22	AMAYRA	7	20
24	ROHIT	8	24
27	ARUN	9	25
29	ASHA	5	27

Retrieve the list of ages of the shopkeepers whose name contains 'A' at start and end and is made up of three or more characters.

1) SELECT Age FROM Sailor WHERE Sname LIKE 'A _ _%';
2) SELECT Age FROM Sailor WHERE Sname LIKE 'A_%A';
3) SELECT Age FROM Sailor WHERE Sname LIKE 'A%';
4) None of these

ORDER BY:

The ORDER BY clause is used to sort/order the result of an SQL query in ascending (or) descending order.

Syntax: SELECT <column_list> FROM <table_list> ORDER BY <column_1> ASC/DESC, <column_2> ASC/DESC ...;

Note:

- By default, the ORDER BY sorts the result of an SQL query in ascending order.
- To specify the sort order, we need to specify explicitly
  1) DESC for descending order
  2) ASC for ascending order

Example: Consider the following given table R.

R

A	B	C
1	2	3
1	2	1
2	1	3
2	1	1
3	5	4
3	4	3

Table 3.5

1) SELECT A, B, C FROM R ORDER BY A, B, C;
Output:

A	B	C
1	2	1
1	2	3
2	1	1
2	1	3
3	4	3
3	5	4

Table 3.6

Here, by default A, B, C are all in ascending order.

2) SELECT A, B, C FROM R ORDER BY A DESC, B, C;
Output:

A	B	C
3	4	3
3	5	4
2	1	1
2	1	3
1	2	1
1	2	3

Table 3.7

Here, A will be ordered in descending order, and by default B, C will be ordered in ascending order.

3) SELECT A, B, C FROM R ORDER BY A DESC, B, C DESC;
Output:

A	B	C
3	4	3
3	5	4
2	1	3
2	1	1
1	2	3
1	2	1

Table 3.8

Here, A will be arranged (sorted) in descending order; by default B will be sorted in ascending order, and C will be sorted in descending order.

Set operations and Null values:

Set operations:

- The SQL operations UNION, INTERSECT and EXCEPT operate on relations.
- Similar to UNION, INTERSECTION and SET DIFFERENCE in relational algebra, the relations participating in the operations must have the same set of attributes.

Hey Learners!!!
Do you know about IN, EXISTS, ALL, ANY set operations in SQL?
Let's discuss these set operations now:

1) IN: It checks whether an element is present in a given set or not.
2) ALL, ANY: To match or correlate a specified value with the elements of a set associating these operations with a comparison operator.
3) EXISTS: To examine if a set follows an empty condition.

Note:
"EXISTS and IN can be prefixed by NOT."

Note:

- By default, the UNION, INTERSECT, and EXCEPT commands remove all the duplicates.
- If we want to retain all duplicates, we have to write UNION ALL, INTERSECT ALL, EXCEPT ALL in place of UNION, INTERSECT and EXCEPT respectively.

Example: Consider the following relation ENROLL.

ENROLL

Stud_id	Course_name	Grade
1	Maths	A
2	Physics	A
3	Chemistry	C
2	Maths	B
2	Biology	B
1	Chemistry	A
3	Physics	D
2	Chemistry	A
2	History	A

Table 3.9

PRACTICE QUESTIONS

Q1 **Retrieve the list of ids of students who got either grade 'A' or grade 'B'.**

Sol: SELECT Stud_id FROM ENROLL WHERE Grade = 'A' UNION SELECT Stud_id FROM ENROLL WHERE Grade = 'B';

Output:

Stud_id
1
2

SELECT Stud_id FROM ENROLL WHERE Grade = 'A' UNION ALL SELECT Stud_id FROM ENROLL WHERE Grade = 'B';

Output:

Stud_id
1
2
2
2
1
2
2

Q2 **Retrieve the list of the ids of students who got both grade 'A' and grades 'B'.**

Sol: SELECT Stud_id FROM ENROLL WHERE Grade = 'A' INTERSECT SELECT Stud_id FROM ENROLL WHERE Grade = 'B';

Output:

Stud_id
2

SELECT Stud_id FROM ENROLL WHERE Grade = 'A' INTERSECT ALL SELECT Stud_id FROM ENROLL WHERE Grade = 'B';
Output:

Stud_id
2
2

Q3 **Retrieve the list of studs Id's who got grade 'A' but not grade 'B'.**

Sol: SELECT Stud_id FROM ENROLL WHERE Grade = 'A' EXCEPT SELECT Stud_id FROM ENROLL WHERE Grade = 'B';
Output:

Stud_id
1

SELECT Stud_id FROM ENROLL WHERE Grade = 'A' EXCEPT ALL SELECT Stud_id FROM ENROLL WHERE Grade = 'B';
Output:

Stud_id
1
1
2

Previous Years' Question

SELECT Operation in SQL is equivalent to

1) The selection operation in relational algebra.
2) The selection operation in relational algebra, except that SELECT in SQL retains duplicates.
3) The projection operation in relational algebra.
4) The projection operation in relational algebra, except that SELECT in SQL retains duplicates.

Sol: Option 4) **(GATE-2021 Set-1)**

Arithmetic operators:

- SQL allows the use of arithmetic operators (+,-,*, /) in queries on attributes having numeric domains.

Example: Consider the relation schema Employee (Eid, Fname, Sex, Salary). Increase the salary of the employee by 10 percent and display the salary and employee's first name.

Sol: In SQL: SELECT Fname, Salary * 1.1 FROM Employee;

Concatenate operator:

- We use || (concatenate operator) to append two strings.
  Example: SELECT Fname || Lname as "FULL-NAME" FROM Employee;
- This will result in a relation having the attribute FULL-NAME, which contains the concatenation of Fname and Lname (i.e. first name and last name).

Employee

Fname	Lname
Komal	Jain

→

FULL-NAME
KomalJain

Between operator:

- It is a comparison operator on the Numeric domain.
- **Syntax:** SELECT column_name(s) FROM Table-Name WHERE column-name BETWEEN value-1 AND value-2;

Example: List all staff whose payscale is between 50,000 and 70,000. Relation Schema: Staff (Sid, Sname, Sex, Payscale)

Sol: SELECT * FROM Staff WHERE Payscale BETWEEN 50000 AND 70000;

- It is also a comparison operator on text and dates.

Example: SELECT * FROM Staff WHERE Hire-date BETWEEN '01-JAN-2010' AND '31-DEC-2010';

NULL values:

- In SQL, NULL value is used to indicate that the information about the value of an attribute is absent.
- The special keyword NULL is used in a predicate to check for null values.
- The output of an arithmetic expression that involves +,-,* or / is NULL if any of the input is NULL.
- The result is considered to be UNKNOWN (i.e. it may be true or false) if a comparison operation implies NULL.

Example: (1 < NULL) = UNKNOWN.

- As WHERE clause condition can involve Boolean operations (AND, OR and NOT) on the results of comparisons. Therefore, we can extend the definition of boolean operations to deal with the UNKNOWN value.

1) AND:

A	B	A and B
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE
TRUE	UNKNOWN	UNKNOWN
FALSE	UNKNOWN	FALSE
UNKNOWN	UNKNOWN	UNKNOWN

Table 3.10

2) OR:

A	B	A or B
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE
TRUE	UNKNOWN	TRUE
FALSE	UNKNOWN	UNKNOWN
UNKNOWN	UNKNOWN	UNKNOWN

Table 3.11

2) NOT: The result of NOT UNKNOWN is UNKONWN.
NOT (TRUE) = FALSE
NOT (FALSE) = TRUE
NOT (UNKONWN) = UNKNOWN

Note:
In SQL, there are operators that can check whether the value of an attribute is NULL or not.
The IS or IS NOT clause is used to analyse if a given value is NULL or NOT.
Reason: Each NULL value is treated distinctly in SQL terminology. So, using = or <> is not at all appropriate.

Example: Consider the following relation R(A, B)

R

A	B
a	1
b	2
c	NULL
d	NULL

Table 3.12

What is the output produced by the following query?
Example: SELECT A FROM R WHERE B is NULL;
Sol: Output:

A
c
d

Table 3.13

Example: SELECT A FROM R WHERE B IS NOT NULL;
Sol: Output:

A
a
b

Table 3.14

Example: SELECT B FROM R;
Sol: Output:

B
1
2
NULL
NULL

Table 3.15

Example: SELECT DISTINCT B FROM R;
Sol: Output:

B
1
2
NULL

Table 3.16

Example: Consider the following relation Employee (Eid, Fname, Ssn, Sex, Super-Ssn)

Eid	Fname	Ssn	Sex	Super-Ssn
1	John	12	M	25
2	Ahmad	34	M	NULL
3	Ruchi	25	F	NULL

Table 3.17

Example: write a SQL query that lists the names of all employees who is not having supervisors.
Sol: SELECT Fname FROM Employee WHERE Super-Ssn IS NULL;
Output:

Fname
Ahmad
Ruchi

Table 3.18

Aggregate functions:

Definition

"Aggregate functions take a collection (a set of multiset) of values as input and return a single value."

- SQL offers 5 built-in aggregate functions:
  1) Average: AVG **2)** Minimum: MIN
  3) Maximum: MAX **4)** Total: SUM
  5) Count: COUNT

Example: Given Relation: Shopkeepers (sid, sname, rating, age)

Sid	Sname	Rating	age
1	Rupa	9	12
2	Puja	8	14
3	Sonu	10	18
4	Dilip	9	22

Table 3.19

1) Find the average age of Shopkeepers with a rating of 9.

Sol: SELECT AVG (age) Shopkeepers WHERE rating = 9;
Output: 17
2) List the name and age of the shopkeepers who are the older ones.
Sol: SELECT sname, MAX(age) FROM Shopkeepers; // Illegal query in SQL.

Note:

- In the SELECT clause, if an aggregate operation is used, then we must use aggregate operations only unless GROUP BY is present in that query.
- Thus, we cannot use MAX(age) as well as sname in the SELECT clause.
- We can use the Nested query to compute the desired answer to this question.

Nested query:

SELECT sname, age FROM Shopkeepers WHERE age = (SELECT MAX (age) from Shopkeepers);

- We will discuss later how the nested query works.

3) Count the number of Shopkeepers.

Sol: SELECT COUNT (*) FROM Shopkeepers;

⇒ COUNT (*) will consider all columns and count the number of rows.

⇒ It includes duplicates, i.e. it will not give distinct rows.

⇒ Output = 4

4) Find the sum of rating of Shopkeepers.

Sol: SELECT SUM (rating) FROM Shopkeepers;

⇒ The above query returns a single value which is the sum of all values in attribute rating.

⇒ Output = 36

5) Find the minimum age of Shopkeepers.

Sol: SELECT MIN (age) FROM Shopkeepers;

⇒ The above query returns a single value which is the minimum age from relation Shopkeepers.

⇒ Output = 12.

6) Find the maximum age of Shopkeepers.

Sol: SELECT MAX (age) FROM Shopkeepers;

⇒ The above query returns a single value which is maximum age from relation Shopkeepers.

⇒ Output = 22.

Dealing with NULL values in aggregate functions:

- All aggregate functions except COUNT (*) ignore NULL values in their input collections.
- The count of an empty collection is zero(0).
- All other aggregate functions give NULL value when it is applied to a set which is empty.

Example: Consider a relation R(A, B):

R

A	B
1	6
2	NULL
3	2
NULL	NULL

Table 3.20

1) COUNT (*) = 4
2) COUNT (A) = 3
3) COUNT (B) = 2
4) SUM (A) = 6
5) SUM (B) = 8
6) MAX (A) = 3
7) MAX (B) = 6
8) MIN (A) = 1
9) MIN (B) = 2
10) AVG (A) = 2
11) AVG (B) = 4

Nested queries in SQL:

Definition:

"A nested query is a query that has another query embedded in it; the embedded query is called subquery."

- Mostly, a subquery presents within the WHERE clause of a query.

IN operators:

- The IN operator in SQL is used to check whether a value is in a given set of elements.

Note:
NOT IN is an operator is used to test whether a value is absent in given set of elements.

EXISTS/NOT EXISTS:

- EXISTS used with a subquery.
- It is said to be met when the subquery return at least 1 row.
- NOT EXISTS can be used to test which rows do not exist in a subquery.

ANY/SOME, ALL:

- These operators are used in the Nested subquery to compare sets.
- These operators are combined with (>, <, > =, < =, <>, =)

- "ANY returns true when any of the subquery values meet the condition."
- "ALL returns true when all of the subquery values meet the condition."

Q1 **Following relations are given below:**
Shopkeepers (shid, shname, shrating, shage)
Sales (shid, Iid, Day)
Items (Iid, Iname, Iprice);

Sol: Shopkeepers

Shid	Shname	Shrating	Shrage
1	Rupa	7	20
2	Puja	8	24
3	Sonu	9	23
4	Rani	8	29
5	Mahima	10	40
6	Sonu	7	23

Table 3.21

Sales

Shid	Iid	Day
1	101	10/10/21
2	103	10/10/21
4	101	10/8/21
1	102	18/6/21
2	102	18/5/21
3	104	9/7/21
4	103	9/7/21
5	103	8/7/21
6	101	8/7/21

Table 3.22

Items

Iid	Iname	Iprice
101	ItemA	10K
102	ItemB	20K
103	ItemC	30K
104	ItemA	20K

Table 3.23

Example: Write a SQL query that will find the name of shopkeepers who have sold item 101.
Sol: SELECT sname FROM Shopkeepers WHERE sid IN (SELECT sid FROM sales WHERE bid = 101);
⇒ The nested subquery first gives the set of Shids for shopkeepers who have sold item 101 (the set contains 1, 4 and 6 as Shid), and then the outer (toplevel) query retrieves the names of shopkeepers whose Shid is in this set.
⇒ Output {Rupa, Rani, Sonu}

Example: Write an SQL query to find the names of shopkeepers who have sold item worth 20K.
Sol: SELECT Shname FROM Shopkeepers WHERE Shid IN (SELECT Shid FROM Sales WHERE Iid IN (SELECT Iid FROM Items WHERE Iprice = '20K'));
⇒ The inner subquery finds the set of Iids of items having a price of 20K, which is 102 and 104.
⇒ The one level above subquery finds the set of Shids of shopkeeper's who have sold one of these items. The set contains 1, 2 and 3 as Shid.
⇒ Lastly, the outer top-level query gives the shopkeepers name whose shid is present in this set of shids.
⇒ Output {Rupa, Puja, Sonu}

Example: Write an SQL query to find the shopkeepers name who have not sold an item worth 20K.
Sol: SELECT Shname FROM Shopkeepers WHERE Shid NOT IN (SELECT Shid FROM Sales WHERE Iid IN (SELECT Iid FROM Items WHERE Iprice = '20K'));
⇒ The inner subquery finds the set of Iids having a price of 20K, which is 102 and 104.
⇒ The one level above subquery finds the set of Shid who have sold either of these items. The set contains Shid as {1, 2, 3}.
⇒ Finally, the top-level query will give the name of shopkeepers whose Shid is not present in this Set {1, 2, 3}.
⇒ Output {Rani, Mahima, Sonu}

Correlated nested queries:

Definition:

"Whenever a condition in the WHERE clause of a nested query references some attributes of a relation declared in the outer query, the two queries are said to be correlated."

Example: Write a query to find the names of shopkeepers who have sold item number 101.
Sol: SELECT S. Shname FROM Shopkeepers S WHERE EXISTS (SELECT * FROM Sales R WHERE R.Iid = 101 AND R.Shid = S.Shid);

⇒ Here, for each shopkeepers in row S, we test whether the set of Sales rows R such that R.Iid = 101 AND S.Shid = R.Shid is nonempty.
⇒ If this is so, shopkeeper S has sold item 101, and we retrieve the name of shopkeepers.
⇒ Output {Rupa, Rani, Sonu}
⇒ It clearly depicts the correlation(dependency) between inner and outer query.

Example: Write an SQL query to find shopkeepers whose rating is better than some shopkeepers whose name is Sonu.
Sol: SELECT S1.Shid FROM Shopkeepers S1 WHERE S1.Shrating > ANY (SELECT S2.Shrating FROM Shopkeepers S2 WHERE S2.Shname = 'Sonu');
⇒ The above query will give shid's of shopkeepers whose rating is better than rating 7 or 9.
⇒ Output of the above query gives Shids 2, 3, 4, and 5.

Note:

- The above query is a correlated subquery because for every shopkeeper in the outer query, we need to run an inner query.

Note:

ANY is similar to SOME keyword in SQL.
Suppose if there are no shopkeepers named Sonu in the above query, then S1.rating > ANY... will return false, and therefore, the query will return an empty set as an answer.
It means the inner subquery must return at least one row to make the comparison true in the case of ANY.

Example: Write an SQL query to find shopkeepers whose rating is better than every shopkeeper named Sonu.

Sol: SELECT S1.Shid FROM Shopkeepers S1 WHERE S1.Shrating > ALL (SELECT S2. Shrating FROM Shopkeepers S2 WHERE S2. Shname = 'Sonu');

⇒ The above query will give Shid of shopkeepers whose rating is better than rating 7 and 9.

⇒ Output of the above query gives Shid 5.

Note:

- If there are no shopkeepers named sonu, then the comparison S1.Shrating > ALL ... is going to be true. In this case, the above query will return the names of all shopkeepers.

Grey Matter Alert!

IN is equivalent to = ANY, and NOT IN is equivalent to <> ALL.

Q1 **Scan the below relational schema:**
Staff (Firstname, Lastname, Sex, Salary, Bno)
Branch (Bname, Bnumber)
Branch_locations (Bnumber, Blocation)

Sol:

Staff

Firstname	Lastname	Sex	Salary	Bno.
Sonu	Sharma	M	30000	5
Anjali	Singh	F	35000	5
Mahima	Seth	F	40000	4
Somya	Jain	F	25000	1
Rahul	Goyal	M	45000	4

Table 3.24

Branch_locations

Bnumber	Blocation
1	Mumbai
5	Chennai
4	Banglore
5	Mumbai

Table 3.25

Branch

Bname	Bnumber
CSIT	1
Chemical	5
Electrical	4

Table 3.26

Example: Write an SQL query to retrieve the First name of staff who works for branches 1, and 4.

Sol: SELECT Fnames FROM Employee WHERE Dno IN (1, 4);

Example: Write an SQL query to find the First name and Last name of the staff who works for the branch located in 'Bangalore'.

Sol: SELECT Firstname, Lastname FROM Staff WHERE Bno IN (SELECT Bnumber FROM Branch_locations WHERE Blocation = 'Bangalore');

Example: Write an SQL query to find out the First names of all the staff where salary is greater than the salary of all staff in branch number 5.

Sol: SELECT Firstname FROM Staff WHERE Salary > ALL (SELECT Salary FROM Staff WHERE Bno = 5);
⇒ Output of this query gives First names = {Mahima, Rahul}

Previous Years' Question

Consider the following relation:
Cinema (theatre, address, capacity)
Which of the following options will be needed at the end of the SQL query?
SELECT P1.address FROM Cinema P1
Such that it always finds the addresses of theatres with maximum capacity.

1) WHERE P1.capacity > = ALL (Select P2 × capacity from Cinema P2)
2) WHERE P1.capacity > = ANY (Select P2 × capacity from Cinema P2)
3) WHERE P1.capacity > ALL (Select max (P2 × capacity) from Cinema P2)
4) WHRE P1.capacity > ANY (Select max (P2 × capacity) from Cinema P2)

Sol: Option 1) **(GATE-2015 Set-3)**

The GROUP BY and HAVING Clauses

Group by:

- GROUP BY clause tells that a SQL SELECT statement can partition result rows into groups depending on their values in one or several columns.
- It is mandatory for all the attributes that are used along with the GROUP BY clause to appear in the SELECT clause.
- If an attribute is not present in the GROUP BY clause, then it must appear only inside the aggregate function in the SELECT clause.

Note:

- GROUP BY clause is often used with COUNT, MIN, MAX, SUM (i.e. with aggregate function)
- If the grouping column contains NULL values, then all NULL values are grouped together.

Having clause:

- SQL provides a HAVING clause, which is often used in conjunction with a GROUP BY clause, to return only those group of tuples which satisfies the provided condition.

Example: Consider the following relation schema:
Employee (Eid, Ename, Sex, Salary, Dno)

The relation given is used to store information about the employees of a company, where Eid is the key and Dno indicates the department to which the employee is assigned.

Sid	Sname	Sex	Salary	Bno
101	Maya	F	20,000	1
102	Rohan	M	30,000	2
103	Kavita	F	28,000	1
104	Rahul	M	24,000	3
105	Manoj	M	30,000	1
106	Supriya	F	10,000	2
107	Rekha	F	32,000	3

Table 3.27

Q1 **Write a SQL query to find the average salary of staff in each branch.**

Sol: SELECT Dno, AVG(Salary) as Avg-Salary FROM Employee GROUP BY Dno;

Q2 **What will be the output of the query 'To find the average salary in each branch'.**

Sol: Output:

Bno	Avg-Salary
1	26,000
2	20,000
3	28,000

Q3 **Write a SQL query to list the branch numbers with an average salary greater than 20,000.**

Sol: SELECT Dno FROM Employee GROUP BY Dno HAVING AVG(Salary) > 17,000;

⇒ This query results Dno = {1, 2} because department numbers 1 and 2 have an average salary greater than 17,000.

Q4 **Consider the following relations:**

Employee

Emp_id	Emp_name
101	Amit
102	Rohan
103	Somya

Performance

Emp_id	Project_code	Rating
101	PA	10
102	PB	9
103	PB	10
102	PA	8
103	PC	5

The SQL query is given below
SELECT E. Emp_name, SUM(P.rating) FROM Employee E, Performance P
WHERE E. Emp_id = P. Emp_id GROUP BY E. Emp_name;
The number of tuples returned by the SQL query is _____

Sol: GROUP BY E. Emp_name means all employee names that are same should be kept in one row. Here, there are 3 employee names, and all are distinct. So, no need to execute the query, and we can tell the number of tuples returned = 3.

OR

Step 1: Perform cross product between Employee E and performance P, we will get 15 rows-relation.

Step 2: Execute WHERE clause; WHERE E. Emp_id = P. Emp_id
Delete rows which does not satisfy WHERE condition.

Step 3: Execute GROUP BY clause: GROUP BY E. Emp_name and then SELECT clause.

Output:

E. Emp_name	SUM(P.rating)
Amit	10
Rohan	17
Somya	15

Rack Your Brain

Consider the relation student with Roll no. as the key

Student

Rollno	Marks
1	92
2	93
3	94
4	NULL

The following SQL query is successfully executed on the relation student.
SELECT avg(Marks) FROM student;
The output of the above query is:

1) 93 **2)** 93.5
3) 94 **4)** NULL

Having and Where clause:

Note:

Generally, HAVING is used in conjunction with GROUP BY, but it is not mandatory.

Example: SELECT Eid, Ename FROM Employee HAVING salary > 20,000;

Note:

If HAVING clause is used without GROUP BY, then it will work the same as the WHERE clause, i.e. we can write the same above query as:
SELECT Sid, Sname FROM staff WHERE salary > 20000;

Q5 **Consider the following relations:**

Shopkeepers

Shid	Shname	Shrating	Shage
22	Komal	7	20
29	Jon	8	22
31	Rahul	9	23
32	Sara	10	21
58	Vikas	5	24
64	Rohan	10	25
71	Preeti	8	22
85	Rahul	7	27
75	Seema	9	28

Output the result of the SQL query specified below:
SELECT S. Shrating, MIN(S. Shage) AS Minage FROM Shopkeepers S WHERE S. Shage > = 23 GROUP BY S. Shrating HAVING COUNT (*) >1;

Sol: The above query will result in a relation containing two attributed rating and minage.

Step 1: Condition in WHERE clause is given as S.Shage > = 23.
It will eliminate all the rows from the Shopkeeper relation which does not satisfy the given condition.

Shrating	Shage
9	23
5	24
10	25
7	27
9	28

(Eliminate all other attributes Shid and Shname as they are not required.)
Step 2: Now, we will apply the GROUP BY clause and sort the table by grouping the attribute rating.

Shrating	Shage
5	24
7	27
9	23
9	28
10	25

Step 3: Finally, we will apply the HAVING clause condition, i.e. COUNT(*)>1.
The groups having rating 5,7 and 10 will be eliminated.
Final answer:

Shrating	Minage
9	23

(As query will result a relation with two attributes rating and minage). Minage will contain 23 corresponding to rating 9 as it is the minimum value among 23 and 28.

Note:
Conditions specified in the WHERE clause are applied before forming the groups and Conditions specified in the HAVING clause are applied only after forming the groups

Order of keywords in SQL:
When we write a SQL query, then the order of keywords that we follow is as follows:

1) SELECT
2) FROM
3) WHERE
4) GROUP BY
5) HAVING
6) ORDER BY

Order of execution in SQL:

1) FROM
2) WHERE
3) GROUP BY
4) HAVING
5) SELECT
6) ORDER BY

The WITH clause:

Definition:

"Using SQL WITH clause, we can give a sub-query block a name (a process also called sub-query refactoring), and later it can be referenced in several places within the main SQL query."

- We can define a temporary view using the WITH clause.
- It is basically a drop-in replacement to the normal sub-query.
- It's not easy to read a query if a nested subquery is used to write it.
- We can write any query using WITH clause in such a way that its logic becomes clearer.
- Also, it allows us to use view definition in multiple places within a query

Example: Consider the relational schemas specified below:
Account (account_number, branch_name, balance)

Account

Account_number	Branch_name	Balance
12345678	Mumbai	25,000
45289243	Hyderabad	30,000
34219341	Bangalore	40,000
98214623	Mumbai	45,000
81243258	Hyderabad	20,000
24319824	Mumbai	20,000
34294526	Banagalore	30,000

Table 3.28

The number of tuples returned by the following SQL query is _____

```
WITH branch_total (branch_name, value) As
        SELECT branch_name, SUM (balance)
        FROM account
        GROUP BY branch_name
WITH branch_total_avg (value) As
        SELECT AVG (value)
        FROM branch-total;

SELECT branch_name FROM branch_total, branch_total_avg WHERE
branch_total.value ≥ branch_total_avg.value;
```

Sol: The first query will return:

branch_total

Branch_name	Value
Bangalore	35,000
Hyderabad	25,000
Mumbai	30,000

Table 3.29

Second query will return: branch_total_avg as average of (35,000, 25,000, 30,000) = 30,000.

Value
30,000

Now, the last query will give the name of the branch where branch-total, value ≥ 30,000

Branch_name
Bangalore

Thus, 1 tuple will be returned.

Joins in SQL:

- To integrate the record sets of two or more relations based on the equality of the common attributes shared by them, the join operation is used.
- Different types of join between relations.
  1) NATURAL JOIN
  2) INNER JOIN
  3) LEFT OUTER JOIN
  4) RIGHT OUTER JOIN
  5) FULL OUTER JOIN

Consider the following relations R and S:

R

A	B
1	a
2	b
3	c
4	d
5	e
6	f

S

A	B
1	g
2	h
3	i
7	j
8	k
9	l

Table 3.30

1) Natural join:

- When we apply natural join on two relations R and S, no need to specify any join condition explicitly.
- Each such pair of common attributes is included only once in the resulting relation.
  Example: SELECT * FROM (R NATURAL JOIN S);
- It will directly join R and S based on attribute A.
- This is the same as query:
  SELECT * FROM R NATURAL JOIN S ON R.A = S.A;

2) Inner join:

- The default type of join in a joined relation.
- Two or more relations are joined based on some join condition on attributes of two relations.

Example: SELECT * FROM R INNER JOIN S ON R.A = S.A;

A	B	A	B
1	a	1	g
2	b	2	h
3	c	3	i

Table 3.31

Consider the relations R and S

R

A	B
1	a
2	b
3	c
4	d
5	e
6	f

S

C	D
1	g
2	h
3	i
7	j
8	k
9	l

Table 3.32

1) SELECT * FROM R INNER JOIN S ON R.A = S.C;
Sol: It will give output:

A	B	C	D
1	a	1	g
2	b	2	h
3	c	3	i

Table 3.33

- INNER JOIN can be represented diagrammatically as:

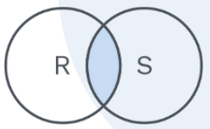


Fig. 3.3

Note:
INNER is a optional keyword. INNER JOIN is same as JOIN.

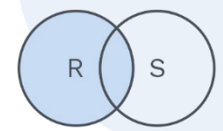
3) left outer join:

- Along with the condition satisfying tuples, all the condition failing records of the left-hand side relation must show up in the resultant relation.
- If the tuple from left-hand side relation that do not match any tuple in right-hand side relation, then NULL values need to be added for the columns of the right relation.

Example: SELECT * FROM R LEFT OUTER JOIN S ON R.A = S.C;
Output:

A	B	C	D
1	a	1	g
2	b	2	h
3	c	3	i
4	d	NULL	NULL
5	e	NULL	NULL
6	f	NULL	NULL

Table 3.34



LEFT OUTER JOIN

Fig. 3.4

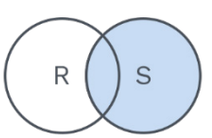
4) Right outer join:

- Every tuple in the right relation must appear in the result.
- If the tuples from the right hand side relation that do not match any tuple in the left-hand side relation, then it is padded with NULL values for the attribute of the right relation.

Example: SELECT * FROM R RIGHT OUTER JOIN S ON R.A = S. C;
Output:

A	B	C	D
1	a	1	g
2	b	2	h
3	c	3	i
NULL	NULL	7	j
NULL	NULL	8	k
NULL	NULL	9	l

Table 3.35



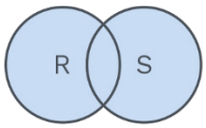
RIGHT OUTER JOIN

Fig. 3.5

5) Full outer join:

- Full outer join includes results of both left and right outer join combined.
- The rows of the left and right-hand side relation that fails to satisfy the JOIN condition are included in the resultant set with NULL values in the right and left-hand relation attributes respectively.

Example: SELECT * FROM R FULL OUTER JOIN S ON R.A = S.C;



FULL OUTER JOIN

Fig. 3.6

Output:

A	B	C	D
1	a	1	g
2	b	2	h
3	c	3	i
4	d	NULL	NULL
5	e	NULL	NULL
6	f	NULL	NULL
NULL	NULL	7	j
NULL	NULL	8	k
NULL	NULL	9	l

Table 3.36

Q6 **Consider the following given relations.**

Restaurant

Rest_id	Rest_name	Rest_city
101	Madhulikam	Bangalore
121	Lazzez	Mumbai
114	Sagar Ratna	Delhi
120	Food Plaza	Bangalore
180	Tandoor Palace	Kolkata
135	Foodie	Chennai

Food_item

Food_id	Food_name	Rest_id
22	Sweets	101
24	Dosa-Idli	114
15	Veg Roll	120
18	Biryani	180
20	Pizza	121
14	Burger	121
10	Paneer Roll	120

Consider the SQL query specified below:
SELECT * FROM Restaurant R LEFT OUTER JOIN Food_item F ON R. Rest_id= F. Rest_id;
What will be the number of tuples returned by the given query?

Sol: The above SQL query returns 8 tuples.
The resultant relation is

Rest_id	Rest_name	Rest_city	Food_id	Food_name
101	Madhulikam	Bangalore	22	Sweets
121	Lazeez	Mumbai	20	Pizza
121	Lazeez	Mumbai	14	Burger
114	Sagar Ratna	Delhi	24	Dosa-Idli
120	Food Plaza	Bangalore	10	Paneer Roll
120	Food Plaza	Bangalore	15	Veg Roll
180	Tandoor Palace	Kolkata	18	Biryani
135	Foodie	Chennai	NULL	NULL

Views in SQL:

Definition:

"known as Virtual Tables in SQL. A view in SQL is a single table that is derived from the other tables; Here, these other tables can be base tables or previously defined views."

- A view can be defined using the create view command.
- We need to declare a view name along with the query that computes view.
- Create view command is defined as:
  CREATE VIEW V AS <query expression> where <query expression>.

Example: Consider the following relation schemas:
Staff(Firstname, Lastname, Sex, Salary,Ssn)
Works_for(Cno,Essn, Hours)
Company(Cname, Cnumber,Clocation,)
CREATE VIEW WORKS_FOR1 AS
SELECT Firstname, Lastname, Cname, Hours FROM Staff, Company, Works_for WHERE Cno
= Cnumber AND Ssn = Essn;

- Here, WORKS_FOR1 will take attribute names from given relations (Staff, Works_for, Company) as we have not specified any new attribute names for it.

WORKS_FOR1

Firstname	Lastname	Cname	Hours

We can also explicitly specify the name of attributes of a view.

Schema definition in SQL:

- CREATE TABLE command is used to define a relation.

Syntax: CREATE TABLE <table name> (<column 1> datatype (width), <column 2> datatype (width) ....., <integrity-constraint$_1$>, ..., ... <integrity-constraint$_k$>);

- We have numerous datatypes in SQL
  1) Numeric: Number, Float, int, Real, Decimal
  2) Character: Char, Varchar2, Nchar, Long, Nvarchar2
  3) Date: Date, Time, Interval
  4) Boolean: Boolean

- **Constraints in SQL:**
  Sometimes while creating an SQL relation (table), we need to specify some constraints on attributes.
  These constraints are

1) NOT NULL
2) UNIQUE
3) PRIMARY KEY
4) FOREIGN KEY
5) CHECK
6) DEFAULT

Example: A SQL query having constraints where the requirement is as follows: For a relation department: dept_id should be a primary key, dept_name should be unique, and there should be an attribute location with no constraints. Create a table (relation) department specifying the above requirements.

Sol: Create table Department (dept_id NUMBER PRIMARY KEY, dept_name VARCHAR2(30), location VARCHAR2(50), UNIQUE (dept_name));

Note:
Constraints in SQL can be defined at two levels.
1) Column level **2)** Table level

Constraints

1) **NOT NULL:** NOT NULL is a constraint that is used when an attribute value is not allowed to be NULL.
2) **UNIQUE:** UNIQUE constraint is used when two rows in any relation should not be same.
   Syntax: UNIQUE (Aj_1, Aj_2, ..., Aj_m)
3) **PRIMAR KEY:** PRIMAR KEY (Aj_1, Aj_2, ..., Aj_m) specification says that attributes Aj_1, Aj_2, ..., Aj_m form the primary key for the relation. The primary key attributes have to be UNIQUE and not NULL.
   There will be only one primary key for one table.
4) **FOREIGN KEY:** FOREIGN KEY is used to show the referential integrity. By default, the primary key attribute of a referenced table is referred by a foreign key of another table.
5) **CHECK (P):** The CHECK clause specifies a condition C that needs to be satisfied by every tuple in the relation.
   Whenever a tuple is inserted or modified, the check clause checks the specified condition on them.
6) **DEFAULT:** DEFAULT constraint is used to set a default value for a column.

Grey Matter Alert!

It is always better to mention a primary key for any relation, but it is not mandatory.

Examples:

1) CREATE TABLE Customer (Customer-name char (20), Customer_street char (30), Customer_city char (30), PRIMARY KEY (Customer-name));

2) CREATE TABLE Account (account_number char (10), branch_name char (15), balance integer, PRIMARY KEY (account_number), CHECK (balance > = 0));

INSERT, DELETE and UPDATE Statements in SQL:

These three commands are used to modify the database in SQL.

INSERT Command:

- We use the INSERT command to insert(add) a new tuple to a relation.
- We need to mention the name of the relation as well as the list of attribute-values for the tuple.
- All the attributes should be used with the INSERT command in the same linear manner as in CREATE TABLE command.

Syntax: INSERT INTO <table name> VALUES (attr_value1, attr_value2, ...);

Let us create a table Staff, and then we will insert values in it.

```
CREATE TABLE Employee
(   Fname       Varchar (15)        NOT NULL,
    Lname       Varchar (15)        NOT NULL,
    Ssn         Char (9)            NOT NULL,
    Bdate       Date,
    Sex         Char,
    Salary      Decimal (10, 2),
    Super_Ssn   Char (9),
    Dno         int                 NOT NULL,
PRIMARY KEY (Ssn), FOREIGN KEY (Super_Ssn) REFERENCES Staff (Ssn),
FOREIGN KEY (Dno) REFERENCES Department ((Dnumber));
INSERT INTO Staff VALUES ('Rohan', 'Sahni', '653298653', '1996-10-10', 'M',
37000, '532467821', 4);
```

Note:

- User can also specify explicit attribute names that correspond to the values given in the INSERT command.

Example: Suppose, we want to insert a tuple in a relation Staff(Firstname,Lastname,Deptno, Ssn).

Then, we have to write:

```
INSERT INTO Staff(Firstname,Lastname,Deptno,Ssn)
```

VALUES ('Rohan', 'Sahni', 4, '653298653');

DELETE Command:

- It removes tuple from a relation.
- At once, Rows are deleted from only one table.

Note:

0,1, or more than 1 row can be deleted by a single DELETE command depending on the number of rows selected by the condition specified in the WHERE clause.

Syntax: DELETE FROM <table name> WHERE <condition>

Example: DELETE FROM employee WHERE Lname = 'Sahni'

DELETE FROM employee WHERE Dno = 5

DELETE FROM employee // Deletes all rows from Employee table

UPDATE Command:

- To modify values of attributes of one or more selected tuples, we use UPDATE Command
- In UPDATE Command, a WHERE clause is used, which selects all those tuples we need to modify.
- A SET clause is used in the UPDATE command to mention the attributes to be modified with their new values.

Syntax: UPDATE <table name> SET <column name> = <value expression>{, <column name> = <value expression>} [WHERE <selection condition>];

Example: Let us want to modify the location of Unacademy's project number 20 to 'Chandigarh' as well as controlling department number to '25' in a relation UnacademyProject.

UnacademyProject-schema (Pname, Pnumber, Plocation, Dnum);

SQL Query: UPDATE Project SET Plocation = 'Mumbai', Dnum = 5 WHERE Pnumber = 20;

Schema change statements in SQL:

- There are some commands in SQL that are used to alter a schema.

DROP Command:

- The DROP TABLE command is used to removes a relation from SQL database.
- In other words, The DROP TABLE command deletes all information regarding the relation, which we are considering to drop from the database.

Syntax: DROP TABLE <table name>;

Note:
Difference between command DROP TABLE r and DELETE FROM r:

- DROP TABLE r deletes not only all tuples of relation r, but also the schema for r.
- Once r is dropped, we need to recreate the table using CREATE TABLE Command to insert new rows.
- DELETE FROM r, retains relation r but deletes all tuple in r.

ALTER Command:

- ALTER TABLE command is used for adding attributes to an existing relation.
- **Syntax:** ALTER TABLE <table name> ADD <column name> <column type>
- We can drop attributes from a relation by the command
  ALTER TABLE <table name> DROP <column name>

Rack Your Brain

Which of the following is a data manipulation command?

1) SELECT **2)** GRANT
3) DROP **4)** INSERT

3.2 RELATIONAL ALGEBRA

Introduction:

- Relational algebra is a procedural query language.
- In a procedural language, the user provides a specific procedure to execute the operations on the database to get the desired output.
- Relational algebra employs a set of operations that inputs one or two tables and produces a resultant output relation based on some pre-defined clauses.
- Select, project, union, set difference, cartesian product, and rename are the basic operations that summarises relational algebra.
- There are several other operations such as set intersection, natural join, and division. We can define these operations in terms of the fundamental operations.
- The select, project and rename operations operate on one relation. So, they are known as unary operations.

Grey Matter Alert!

In a non-procedural language, the user doesn't provide any specific procedure to obtain the information but specifies the desired information.

Selection and projection:

- The SELECT operator chooses those tuples in the output that satisfy the specified condition.
- The select operation is denoted by
  $\sigma_{<selection\text{-}condition>}$ (Relation name)
  Consider the relation Sailors:

Eid	Ename	Rating	Age
101	Richa	7	24
105	Rohan	9	20
120	Mahesh	8	26
145	Abhishek	10	29

Table 3.37

To select those tuples of Sailors relation where rating is greater than 8. We can write:

$\sigma_{rating > 8}$ (Sailors)

The output we get the relation shown below:

Sid	Ename	Rating	Age
105	Rohan	9	20
145	Abhishek	10	29

Table 3.38

Example: Consider the relation Employee:

Employee

Eid	Ename	Salary	Dno
1	a	10K	1
2	b	12K	1
3	c	11K	2
4	d	13K	3
5	e	7K	3

Table 3.39

The expression: $\sigma_{salary > 10K \text{ AND } Dno = 3}$ (Employee) evaluates to the relation.

Eid	Ename	Salary	Dno
4	d	13K	3

- Comaprison operators that can be used in selection conditions are: <,<=,=,≠,>=,>.
- The selection operation is applied to each tuple individually.
- The degree (i.e. number of attributes) of the resultant relation after applying the SELECT operation is going to have the same degree as of relation R.
- Number of rows in final output relation £ Number of rows in given input relation(R).
  i.e. $|\sigma_c(R)| = |R|$ where c is any condition.

Note:

- The select operation is commutative i.e.

$$\sigma_{<cond\ 1>}(\sigma_{<cond\ 2>}(R)) = \sigma_{<cond\ 2>}(\sigma_{<cond\ 1>}(R))$$

- We can also write,

$$\sigma_{<cond\ 1>}(\sigma_{<cond\ 2>}(\ldots(\sigma_{<cond\ n>}(R))\ldots)) = \sigma_{<cond\ 1> \text{ AND } \ldots <codn\ 2> \ldots \text{ AND } <cond\ n>}(R)$$

Project operation:

- The project operation is used to choose certain columns from the table and trashes out the other attribute fields.
- The projection operator is denoted by π.
- One can visualise the result of the project operations as an input relation is vertically separated into two relations. One having needed columns (attributes) and the other comprising of the discarded columns.
- The general form to represent the project operation is:

$$\pi_{<attribute\ list>}(Relation)$$

  where <attribute list> is the targeted sub-list of attributes belonging to the relational attribute set.
- The resultant relation after applying the project operation will contain only those attributes that are mentioned in <attribute list> in the same order as they are oriented in the list.

Example: Consider the relation Employee:

Employee

Eid	Ename	Salary	Dno	Sex
101	Raman	30,000	1	M
102	Sneha	20,000	1	F
103	Maya	20,000	2	F
104	Ranjith	20,000	2	M
105	Mahesh	15,000	3	M

Table 3.40

The expression, $\pi_{Salary, Sex}$ (Employee) evaluates to the relation:

Salary	Sex
30,000	M
20,000	F
20,000	M
15,000	M

Table 3.41

The expression $\pi_{Ename, Salary, Sex}$ (Employee) evaluates to the relation:

Ename	Salary	Sex
Raman	30,000	M
Sneha	20,000	F
Maya	20,000	F
Ranjith	20,000	M
Mahesh	15,000	M

Table 3.42

- The project operation results in a set of a distinct tuple as the Project operation removes duplicate tuples.

Note:

- $\pi_{<\text{list 1}>}(\pi_{<\text{list 2}>}(R)) = \pi_{<\text{list 1}>}(R)$
  is true if attributes in <list 1> is also present in <list 2>
- Project operation is not commutative.

Note:

Projection operation in relational algebra is equivalent to SELECT DISTINCT in SQL.

E.g. $\pi_{\text{Sex, Salary}}$ (Employee)
is similar to
SELECT DISTINCT Sex, Salary FROM Employee;

Previous Years' Question

Which of the following query transformations (i.e. replacing the LHS expression by the RHS expression) is incorrect?
R_1 and R_2 are relation, C_1 and C_2 are selection conditions, and A_1 and A_2 are attributes of R_1.

1) $\sigma_{C_1}(\sigma_{C_2}(R_1)) \rightarrow \sigma_{C_2}(\sigma_{C_1}(R_1))$ **2)** $\sigma_{C_1}(\pi_{A_1}(R_1)) \rightarrow \pi_{A_1}(\sigma_{C_1}(R_1))$

3) $\sigma_{C_1}(R_1 \cup R_2) \rightarrow \sigma_{C_1}(R_1) \cup \sigma_{C_1}(R_2)$ **4)** $\pi_{A_1}(\sigma_{C_1}(R_1)) \rightarrow \sigma_{C_1}(\pi_{A_1}(R_1))$

Sol: Option 4) **(GATE-1998)**

Rename operation:

- Rename operation is used to rename either the relation name or the attribute names or both.
- The rename operator is denoted by rho (ρ).
- The general rename operation when applied to a relation R of degree n is denoted by any of the following three forms:
  $\rho_{s(B_1, B_2, \ldots B_n)}(R)$
  (or) $\rho_s(R)$
  (or) $\rho_{(B_1, B_2, \ldots B_n)}(R)$
  where s is the new relation name, and B_1, B_2, ... B_n are the new attribute names.
- $\rho_{s(B_1, B_2, \ldots B_n)}(R)$ renames both the relation and its attributes.
- $\rho_s(R)$ renames the relation only.

- $\rho_{(B_1, B_2, \ldots B_n)}(R)$ renames the attributes only.

  E.g. Consider the relation customer:

Cid	Cname	Sex
101	S_1	M
102	S_2	F
103	S_3	F

Table 3.43

$\rho_{(Customerid,\ Customername,\ Sex)}$ (customer)

We will get

Customer id	Customer name	Sex
101	S_1	M
102	S_2	F
103	S_3	M

Table 3.44

Set operations:

- The following standard operations on sets are available in relational algebra:
  1) Union ($\cup$)
  2) Intersection ($\cap$)
  3) Set-difference (–)
- There are binary operations and they are applied to two sets.

Grey Matter Alert!

- The two relations on which either union or intersection or set difference operations are implemented upon must necessarily have similar data types of tuples. This condition is called type compatibility.
- Two relations $R(A_1, A_2, \ldots A_n)$ and $Q(B_1, B_2, \ldots B_n)$ are referred to as type compatible if they have the same degree and domain (A_i) is equal to the domain (B_i).
- Type compatible is also called as union compatible.

The union operation:

- P ⋃ Q (P union Q): The resultant relation stores the record set comprising of tuples available in P or Q or both.
- The resultant relation has same schematic representation as that of P.
- The UNION operation does not involve duplicate records in the final relation.

The intersection operation:

- Given two relations R and S, R ⋂ S gives the resultant relation that includes all tuples that are in both relation R and S.
- We will assume that the schema of the resulting relation of R ⋂ S will be the same as schema of R.

The set-difference operation:

- Set-difference, denoted by R-S (R minus S), the result of this relation is a relation that includes all tuples that are present in relation R but not in relation S.
- Here also, we assume that the schema of the resulting relation R-S is same as schema of R.

Note:

- (i) R ⋃ S = S ⋃ R (ii) R ⋂ S = S ⋂ R. It means union and intersection both are commutative operations.
- The set difference operation is not commutative i.e. R – S ≠ S – R.

Example: Consider a relation between employee E_1 and E_2.

E_1

Eid	Ename	Age	Rating
20	Somya	24.0	7
30	Rahul	25.0	8
40	Ranjith	24.0	9
50	Yashvi	23.0	10
60	Sonam	27.0	8

Table 3.45

E_2

Eid	Ename	Age	Rating
30	Rahul	25.0	8
35	Satyam	24.0	9
50	Yashvi	23.0	10
60	Sonam	27.0	8

Table 3.46

I) Now, $E_1 \cup E_2$ will be:

Eid	Ename	Age	Rating
20	Somya	24.0	7
30	Rahul	25.0	8
35	Satyam	24.0	9
40	Ranjith	24.0	9
50	Yashvi	23.0	10
60	Sonam	27.0	8

Table 3.47

II) $E_1 \cap E_2$ will be:

Eid	Ename	Age	Rating
30	Rahul	25.0	8
50	Yashvi	23.0	10
60	Sonam	27.0	8

Table 3.48

III) $E_1 - E_2$ will be:

Eid	Ename	Age	Rating
20	Somya	24.0	7
40	Ranjith	24.0	9

Table 3.49

IV) $E_2 - E_1$ will be:

Eid	Ename	Age	Rating
35	Satyam	24.0	9

Table 3.50

Note:

- Intersection can be expressed in terms of union and set-difference as follows:

$$R \cap S = ((R \cup S) - (R - S)) - (S - R)$$

- Both union and intersection are associative operations.

$$R \cup (S \cup T) = (R \cup S) \cup T$$
$$R \cap (S \cap T) = R \cap (S \cap T)$$

Cartesian product (cross product) operation:

- It is also known as cross join, denoted by ×.
- Cross Product is a binary set operation.
- Relations on which we apply cross product need not be union compatible
- The cartesian product operation, allows us to combine information from any two relations.
- R × S returns a relation whose schema contains all the attributes of R followed by all the attribute of S, i.e. the result of $R(A_1, A_2, ..., A_n)$ × $S(B_1, B_2, .., B_m)$ will result in a relation T with degree n + m attributes. $T(A_1, A_2, ... A_n, B_1, B_2, ... B_m)$ in that order.
- In cross product of R and S, if R has m tuples denoted as |R| = m and S has n tuples denoted as |S| = n then R × S will have m × n tuples.

Note:

The cartesian product operation is mostly useful when followed by a selection.

Example: Consider the relation R and S:

R

A	B	C
1	a	b
2	c	d
3	e	f

S

D	E
1	c
2	d

Table 3.51

(R contains 3 tuples and 3 attributes. S contains 2 tuples and 2 attributes)
Then R × S will be:

A	B	C	D	E
1	a	b	1	c
1	a	b	2	d
2	c	d	1	c
2	c	d	2	d
3	e	f	1	c
3	e	f	2	d

Table 3.52

Containing 3 × 2 = 6 tuples and 3 + 2 = 5 attributes.
The expression $\sigma_{A = D}$ (R × S) will give the result:

A	B	C	D	E
1	a	b	1	c
2	c	d	2	d

Table 3.53

Division operations:

- It is represented by ÷ and is useful for expressing certain kind of queries.
- The Division operation is defined using the basic operators of the algebra.
- Implementation of Division using basic operations:

$R \div S$

Step 1: $T_1 \leftarrow \pi_{(R-S)}(R)$

Step 2: $T_2 \leftarrow \pi_{(R-S)}((S \times T_1) - R)$

Step 3: $T \leftarrow T_1 - T_2$

Example: Consider relation R and S as follows:

R

A	B
a_1	b_1
a_2	b_1
a_1	b_2
a_2	b_2
a_1	b_3

S

A
a_1
a_2

Table 3.54

Then $R \div S$ will be : $T \neg R \leftarrow S$

T

B
b_1
b_2

Step 1: $T_1 \leftarrow \pi_{(R-S)}(R) = T_1 \leftarrow \pi_B R = T_1$

B
b_1
b_2
b_3

Table 3.55

Step 2: (a) $S \times T_1 =$

A	B
a_1	b_1
a_1	b_2
a_1	b_3
a_2	b_1
a_2	b_2
a_2	b_3

Table 3.56

(b) $(S \times T_1) - R$

A	B
a_2	b_3

$T_2 \leftarrow \pi_{(R-S)}((S \times T_1) - R)$ will give: T_2

B
b_3

Step 3: $T \leftarrow T_1 - T_2$

T_2

B
b_1
b_2

Table 3.57

Rename operation:

- In relational algebra, we do not have any name for the results through which we can refer them.
- Most of the time result of a relational algebraic expression usually takes name of field from the original relation.

Previous Years' Question

Consider a database that has the relation schema CR(StudentName, CourseName). An instance of the schema CR is as given below:

StudentName	CourseName
SA	CA
SA	CB
SA	CC
SB	CB
SB	CC
SC	CA
SC	CB
SC	CC
SD	CA
SD	CB
SD	CC
SD	CD
SE	CD
SE	CA
SE	CB
SF	CA
SF	CB
SF	CC

The following query is made on the database.

- $T_1 \leftarrow \pi_{CourseName}\ (\sigma_{StudentName\ =\ SA}\ (CR))$
- $T_2 \leftarrow CR\ ,\ T_1$

The number of rows in T_2 is _____

Sol: Range 4 to 4 **(GATE Set-1-2017)**

- But, It is always good to give them names, using the rename operator denoted by ρ.
- Given a relational-algebra expression E, the expression ρ_x (E) returns the result of expression E under the name x.
- Another form of the rename operation is as follows:

$$\rho_{x(A1, A2, \ldots An)} (E)$$

returns the result of expression E under the name x and with attributes renamed to $A_1, A_2, \ldots, A_n$.

Example: Rename the attributes Empno, Empname of relation employee to Eno, Ename.

$$\rho_{(Eno, Ename)} (Employee)$$

A complete set of relational algebra operations:

- $\{\sigma, \pi, U, -, \times\}$ is a complete set of relational algebra operations.
- It means using a sequence of operations from this set, any other relational algebra operations can be expressed.

Example: Intersection operation can be expressed using union and set-difference as follows.

$$R \cap S \equiv (R \cup S) - ((R - S) \cup (S - R))$$

Division operation can be implemented using p, x and – (set-difference)
Similarly, a join operation can be implemented using x(cartesian product) and select operations.

$$R \bowtie_{<condition>} S \equiv \sigma_{<condition>} (R \times S)$$

3.3 JOINS

- The join operations is one of the most useful operations in relational algebra.
- The join operation denoted by $\bowtie$, is used to combine related tuples from two relations.
- Join operation is used more frequently than cross-product even though a join is a cross product followed by a selection.
- Reason behind it is cross-product produces a larger results compare to result produce by join.
- Thus, join is very important relational operation.

Condition joins:

- Conditional join operation takes a pair of relation instances as arguments and a join condition c and returns a new relation.
- It is defined as:

$$R \bowtie_c S = \sigma_c (R \times S)$$

Example: Consider the relation Employee E and Project P.

E

Eid	Ename	Rating	Age
20	Ravish	8	24.0
30	Radha	7	25.0
40	Shyam	9	26.0

P

Eid	Pid	Day
20	120	2/8/21
40	115	5/6/21

Table 3.58

The result of $E \bowtie_{E.Eid < P.Eid} P$ is

Eid	Ename	Rating	Age	Eid	Pid	Day
20	Ravish	8	24.0	40	115	5/6/21
30	Radha	7	25.0	40	115	5/6/21

Table 3.59

Equijoin:

- Equijoin involves join condition with equality comparisons.
- Equijoin is a join where = comparison operator is used.

Note:

In the result of an Equijoin, we always have one or more pairs of attribute that have identical values in every tuples.

Consider the relation Employee and Drives as follows:

Employee

Eno	Ename	Sex	Age
101	Ravi	M	24.0
102	Satyam	M	25.0
103	Meera	F	23.0
104	Rohan	M	27.0

Drives

EmpNo	Car
101	Volvo
103	Mercedes
104	Jaguar
104	Toyota

Table 3.60

The result of Employee $\bowtie_{\text{Employee.Eno = Drives.EmpNo}}$ Drives

Eno	Ename	Sex	Age	Emp No	Car
101	Ravi	M	24.0	101	Volvo
103	Meera	F	23.0	103	Mercedes
104	Rohan	M	27.0	104	Jaguar
104	Rohan	M	27.0	104	Toyota

Table 3.61

Natural join:

- Natural join is denoted by *.
- Natural join is used to combine two relation R and S having a common attribute(s) names.
- If we apply Natural join on two relations R and S, then it is denoted as R * S.
- We don't need to write equality conditions explicitly when two relations are joined using natural join.

Note:

The standard definition of natural join requires that the two join attributes should have the same name in both relations.
If not, then a renaming operation is applied first before applying Natural join.

Natural join returns similar attributes only once in the resulting relation. Consider the relation between Student and Performance.

Student

Rollno	Student_name
1	Amit
2	Priya
3	Rohan
4	Komal

Performance

Rollno	Subject_code	Marks
1	A	84
1	B	90
2	C	92
3	A	85

Table 3.62

The result of stu_per ← student * performance

Stu_per

Rollno	Student_name	Subject_code	Marks
1	Amit	A	84
1	Amit	B	90
2	Priya	C	92
3	Rohan	A	85

Table 3.63

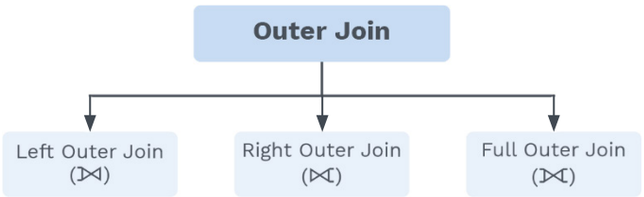
Note:

Condition join, Equijoin and Natural join are called Inner join.

Outer-join:

- It is a variation of JOIN that efficiently deals with missing data.
- Here, since the schema of the result includes all attributes from both relations (table), if a tuple from the first relation does not match tuple in the second relation, we simply put NULL values in the tuple of the resulting relation on the attributes of the second relation.

- There are three forms of outer join:



1) **Left outer join (⟕):** The left outer join takes all tuples in the left relation that do not match with any tuple in the right relation, and padded these tuples with NULL values for all the attributes of the right relation. Add them to the result of inner join.
2) **Right outer join (⟖):** It considers the records of the right-hand side relations that fails to follow the JOIN condition. It pads NULL values in the left-hand-side attributes for those records.
3) **Full outer join (⟗):** In full outer join, along with the result of the inner join of two relation, it will contain padding tuples from both left and right relation that does not match with any tuples from other (right and left relation respectively), relations.

E.g. Consider the two relations R and S as follows:

R

A	B	C
a_1	b_1	c_1
a_2	b_2	c_2
a_3	b_3	c_3

S

D	E
a_1	e_1
a_3	e_2
a_4	e_3

Table 3.64

i) The result of R $⟕_{A=D}$ S will be

A	B	C	D	E
a_1	b_1	c_1	a_1	e_1
a_2	b_2	c_2	NULL	NULL
a_3	b_3	c_3	a_3	e_2

Table 3.65

ii) The result of R $⟖_{A=D}$ S will be

A	B	C	D	E
a_1	b_1	c_1	a_1	e_1
a_3	b_3	c_3	a_3	e_2
NULL	NULL	NULL	a_4	e_3

Table 3.66

iii) The result of the full outer join $R \overset{\leftrightarrow}{\bowtie}_{A=D} S$ will be

A	B	C	D	E
a_1	b_1	c_1	a_1	e_1
a_3	b_3	c_3	a_3	e_2
a_2	b_2	c_2	NULL	NULL
NULL	NULL	NULL	a_4	e_3

Table 3.67

Note:

1) $R \bowtie S \subseteq R \overset{\leftrightarrow}{\bowtie} S$

2) $R \bowtie S \subseteq R \overset{\leftarrow}{\bowtie} S$

3) $R \overset{\leftrightarrow}{\bowtie} S = R \overset{\leftarrow}{\bowtie} S \cup R \overset{\rightarrow}{\bowtie} S$

Previous Years' Question

Let R and S be two relations with the following schema.

$$R(P, Q, R_1, R_2, R_3)$$

$$S(P, Q, S_1, S_2)$$

Where {P, Q} is the key for both schemas. Which of the following queries are equivalent?

I. $\pi_P (R \bowtie S)$

II. $\pi_P (R) \bowtie \pi_P (S)$

III. $\pi_P (\pi_{P,Q} (R) \cap \pi_{P,Q} (S))$

IV. $\pi_P (\pi_{P,Q} (R) - (\pi_{P,Q} (R) - \pi_{P,Q} (S)))$

1) Only I and II **2)** Only I and III
3) Only I, II and III **4)** Only I, III and IV

Sol: Option 4) **(GATE-2008)**

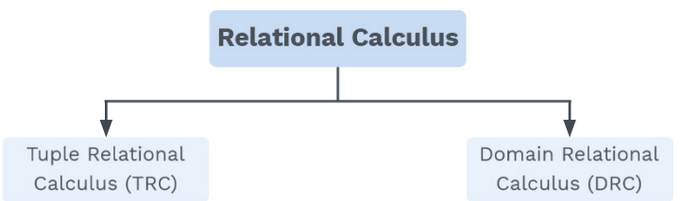
Extended relational algebra operations:

- In basic relational algebra, we don't have some features like arithmetic operations, string concatenation etc.
- Later, the basic relational algebra has been extended.
- Example – To allow arithmetic operations as a part of the projection, to allow aggregate operations etc.
- We can extend the projection operation by allowing arithmetic functions to be used in the projection list.

E.g. $\pi_{id, name, dept\_name, salary\ ,\ 12}$ (Instructor)

Relational calculus:

- Relational calculus is an alternative to relational algebra.
- Relational calculus is considered to be non-procedural language.



- Relational algebra, tuple relational calculus and domain relational calculus are equivalent in power (expressive power of the languages is identical).
- Thus, relational algebra and relational calculus are relationally complete.

Tuple relational calculus:

- In TRC, we specify the tuple variables.
- A tuple relational calculus query is {t|COND (t)}
  where t is a tuple variable and COND(t) is a conditional (Boolean) expression that describes t.
- **Example:** Shopkeeper(Firstname, Lastname, Rating)
  To find all shopkeepers whose rating is more than 8. It can be represented in TRC as:
  {t | Shopkeeper(t) AND t.rating > 8}
  It will give all the tuples from shopkeeper relation that satisfies t.rating > 8 conditions.
  When we want to list only some attributes from Shopkeeper relation ,we can write
  the following expression.
  {t.Firstname, t.Lastname | Shopkeeper(t) AND t.rating > 8 }
- A formula (Boolean condition) in TRC expression is made up of one or more atoms connected using the logical operators AND, OR and NOT.
- A formula is recursively defined as one of the following:
  I) Any atomic formula
  II) If p and q are formulas then so are
  $\neg p$, $p \wedge q$, $p \vee q$, $p \Rightarrow q$.
  III) $\exists t\ (p(t))$, where t is a tuple variable.
  IV) $\forall t\ (p(t))$, where t is a tuple variable.

Note:

$\exists$ represents existential quantifier
$\forall$ represents universal quantifier

Free and bounded variables:

- $\exists$ and $\forall$ are used to **bind** the variable.
- In a formula, if it does not contain an occurrence of a quantifier that binds, the variable then a variable is said to be free.

Example:
Consider the following relational schema:
Employee (Fname, Lname, Ssn, Sex, Salary, Bdate, Address, Super_Ssn, Dno)
Department (Dname, Dnumber, Mgr_Ssn, Mgr_start_date)
Dept_locations (Dnumber, Dlocation)
UnacademyProject (UPname, UPnumber, UPlocation, UDnum)
Works_on (Essn, Pno, Hours)
Dependent (Essn, Dependent_name, Sex, Bdate, Relationship)

Q1 **Project out the name and address information of the department number ten employees.**

Sol: $\{e \times Fname, e \times Address \mid Employee(e) \wedge e \times Dno = 10\}$

- Only the free tuple variables in a TRC expression need to appear to the left of the vertical bar(|).
- Whatever we want to display should be written left to the vertical bar.
- Here, if a tuple satisfies the conditions specified after the bar then the attributes Fnames and Address are retrieved for each such tuple.

Q2 **List out the names of the dependent(s) of the employee with first name as 'Ram'.**

Sol: $\{d.Dependent\_name \mid Dependent\ (d) \wedge (\exists e)\ (Employee\ (e) \wedge e.Ssn = d.Essn \wedge e.Fname = \text{'Ram'})\}$

- Here, d is free tuple variable, tuple variable e is bound to the existential quantifier.
- We are performing join between two relation Employee and Dependent and then selecting those combinations of tuples that satisfy the condition.
- Tuple variable d ranges over Dependent relation and tuple variable e ranges over Employee relations.

Q3 **Retrieve the first names and addreses of the 'Research' department employees.**

Sol: {e.Fname, e.Address | Employee (e) AND (∃d) (Department(d) ∧ e.Dno= d.Dnumber ∧ d.Dname = 'Research'}

- Tuple variable e ranges over the relation Employee and, tuple variable d ranges over the relation Department.

Q4 **Retrieve the project number, authority department number and respective Ssn of the manager for all such projects having the base location in 'Mumbai'.**

Sol: {p.Pnumber, p.Dnum, d.Mgr_Ssn | Project (p) AND Department (d) ∧ p.location = 'Mumbai' ∧ p.Dnum = d.Dnumber}

Q5 **Retrieve the project number, authority department number and respective last-name, DOB and address of the manager for all such projects having base location in 'Mumbai'.**

Sol: {p.Pnumber, p.Dnum, e.Lname, e.Bdate, e..Address | Project (p) ∧ Employee (e) ∧ p×location = 'Mumbai' ∧ (∃d) (Department (d) ∧ p.Dnum = d×Dnumber ∧ d×Mgr_Ssn = e×Ssn)}

- When there are two or more free tuple variables, take cartesian product of those relations. Out of all possible combinations of tuples, only the combinations that satisfy the condition are selected.

Q6 **Retrieve the project number in which people with teh last name 'Kohli' is either a worker or a manager in control of the concerned department.**

Sol: {p.Pnumber | Project (p) AND (((∃e) (∃w) (Employee (e) AND works-on (w) AND w.Pno = p.Pnumber AND e.Lname = 'Kohli' AND e.Ssn = w.Essn)) OR ((∃m) (∃d) (Employee (m) AND Department (d) AND p.Dnum = d.Dnumber AND d.Mgr_Ssn= m.Ssn AND m.Lname = 'Kohli'))))}

Q7 **List out the first names of the employees working on projects under department number 7.**

Sol: {e.Fname | Employee (e) AND ((∃(x)) (∃(w)) (project(x) AND Works_on(w) AND x.Dnum = 7 AND w.Essn = e.Ssn AND x.Pnumber = w.Pno))}

Note:

1) $P \rightarrow Q \equiv \neg PVQ$

2) $\neg(\forall x(p(x))) \equiv \exists x(\neg p(x))$

3) $(\forall x)\ (p(x)) \equiv \neg(\exists x)\ (\neg p(x))$

4) $(\exists x)\ (p(x)) \equiv \neg(\forall x)\ (\neg p(x))$

Q8 **Project out the employees first names with no dependents.**

Sol: {e.Fname | Employee (e) AND (NOT (∃d) (Dependent(d) AND e.Ssn = d.Essn))}

Rack Your Brain

Which of the following relational algebra expression is equivalent to the given tuple calculus expression.
{t | t ≡ r ∧ (t[M] = 25 ∧ t[N] = 40)}

1) $\sigma_{(M = 25)}(r) - \sigma_{(N = 40)}(r)$

2) $\sigma_{(M = 25)}(r) - \sigma_{(N = 40)}(r)$

3) $\sigma_{(M = 25)}(r) - \sigma_{(N = 40)}(r)$

4) None of these

Unsafe relational calculus expression:

- The expression is known to be a safe expression if it is guaranteed to yield a finite number of tuples as its result.
- If relational calculus is not yielding finite number of tuples as its result then expression is unsafe.

E.g. {e | ¬ employee(e)}
There are infinitely many tuples that does not belong to employee relation.
So, the above example is unsafe.

Previous Years' Question

Which of the relational calculus expression is not safe?

1) $\{t \mid \exists u \equiv R_1 (t[A] = u[A]) \wedge \neg \exists s \equiv R_2 (t[A] = s[A])\}$

2) $\{t \mid \forall u \equiv R_1 (u[A] = \forall x \forall \Rightarrow \exists s \equiv R_2 (t[A] = s[A] \wedge s[A] = u[A]))\}$

3) $\{t \mid \neg (t \equiv R_1)\}$

4) $\{t \mid \exists u \equiv R_1 (t[A] = u[A]) \wedge \exists s \equiv R_2 (t[A] = s[A])\}$

Sol: Option 3) **(GATE-2001)**

Note:

For every safe Tuple relational calculus query there is an equivalent relational algebra query.

Domain relational calculus:

- Another type of relational calculus is domain relational calculus.
- Domain relational calculus uses Domain variables.
- Variable that ranges over the values in the domain of some attributes is known as domain Variable.
- Domain relational calculus is equivalent in power to tuple relational calculus.
- A Domain relational calculus query is of the form

$$\{x_1, x_2, x_3, \ldots, x_n \mid P(x_1, x_2, \ldots, x_n)\}$$

Where $x_1, x_2, \ldots, x_n$ are domain variables and P represents a formula which is made up of atoms. An atom in the domain relational calculus is of the form:

1) An atom of the form $R(x_1, x_2, \ldots x_n)$ where R is a relation name.

Note:

To make a domain calculus expression more brief, we can write

$$\{x_1, x_2, x_3, \ldots x_n \mid R(x_1\ x_2\ x_3) \text{ AND } \ldots\}$$

Instead of $\{x_1, x_2, x_3, \ldots xn \mid R(x_1, x_2, x_3)$ AND ...$\}$
by removing commas in a list of variables.

2) An atom of the form x op y where x and y are domain variable and op is a comparison operator {<, ≤, =, >, ≥, ≠}.

3) An atom of the form x op c where x is a domain variable and op comparison operator and c is a constant.

- Domain relational calculus formulae is build up from atoms using following rules:
- Domain relational calculus formulae is build up from atoms similar to tuple calculus relation

Example: Consider the relation schemas:
Employee (Fname, Lname, Ssn, Sex, Salary, Bdate, Address, Super_Ssn, Dno)
Department (Dname, Dnumber, Mgr_Ssn, Mgr_start_date)
Project (Pname, Pnumber, Plocation, Dnum)
Works-on (Essn, Pno, Hours)
Dependent (Essn, Dependent × name, sex, Bdate, relationship)

Q1 **List the salary as well as address of the employee named 'Somya Jain'.**

Sol: $\text{Employee}\left(\overset{m}{\text{Fname}}, \overset{n}{\text{Lname}}, \overset{o}{\text{Ssn}}, \overset{p}{\text{Sex}}, \overset{q}{\text{Salary}}, \overset{r}{\text{Bdate}}, \overset{s}{\text{Address}}, \overset{t}{\text{Super\_Ssn}}, \overset{u}{\text{Dno}},\right)$

- We need 9 variables for employee relation, ranging over each of the domains of attributes of employee in order.
- {q, s | (∃m) (∃n) (∃o) (∃p) (∃r) (∃t) (∃u) (Employee (mnopqrstu) ∧ m = 'Somya' ∧ n = 'Jain')}
- q and s are free variable, as they appear to the left of the vertical bar and not bounded to any quantifier.
- For convenience purpose, we generally quantify only those
- variables that appears in the condition.

Q2 **List the first name, last name and address of all employees who work for the department 'Research'.**

Sol: {m, n, s | (∃g) (∃h) (∃u) (Employee (mnopqrstu) ∧ Department (ghij) ∧ g = 'Research' ∧ h = u)}

$\text{Employee}\left(\overset{m}{\text{Fname}}, \overset{n}{\text{Lname}}, \overset{o}{\text{Ssn}}, \overset{p}{\text{Sex}}, \overset{q}{\text{Salary}}, \overset{r}{\text{Bdate}}, \overset{s}{\text{Address}}, \overset{t}{\text{Super\_Ssn}}, \overset{u}{\text{Dno}},\right)$

$$\text{Department}\left(\underset{g}{\text{Dname}}, \underset{h}{\text{Dnumber}}, \underset{i}{\text{Mgr\_Ssn}}, \underset{j}{\text{Mgr\_start\_date}}\right)$$

- Here, we are joining two relation employee and department using h = u where h is a domain variable ranges over attribute Dnumber of Department and u is a domain variable ranges over attribute Dno of relation Employee.

Q3 **For every project located in 'Mumbai', list the project number, the controlling department number and department manager's surname and address**

Sol: {x, z, n, s | (∃y) (∃h) (∃i) (∃o) (Project (wxyz) ∧ Employee (mnopqrstu) ∧ Department (ghij) ∧ h = z ∧ i = o ∧ y = 'Mumbai')}

$$\text{Employee}\left(\underset{m}{\text{Fname}}, \underset{n}{\text{Lname}}, \underset{o}{\text{Ssn}}, \underset{p}{\text{Sex}}, \underset{q}{\text{Salary}}, \underset{r}{\text{Bdate}}, \underset{s}{\text{Address}}, \underset{t}{\text{Super\_Ssn}}, \underset{u}{\text{Dno}},\right)$$

$$\text{Department}\left(\underset{g}{\text{Dname}}, \underset{h}{\text{Dnumber}}, \underset{i}{\text{Mgr\_Ssn}}, \underset{j}{\text{Mgr\_start\_date}}\right)$$

$$\text{Project}\left(\underset{w}{\text{Pname}}, \underset{x}{\text{Pnumber}}, \underset{y}{\text{Plocation}}, \underset{z}{\text{Dnum}}\right)$$

Q4 **Find the names of employees who haven't any dependents.**

Sol: {m, n | ∃(o) (Employee (mnopqrstu) ∧ (Not (∃a) (Dependent (abcde) ∧ o = a)))}

$$\text{Employee}\left(\underset{m}{\text{Fname}}, \underset{n}{\text{Lname}}, \underset{o}{\text{Ssn}}, \underset{p}{\text{Sex}}, \underset{q}{\text{Salary}}, \underset{r}{\text{Bdate}}, \underset{s}{\text{Address}}, \underset{t}{\text{Super\_Ssn}}, \underset{u}{\text{Dno}},\right)$$

$$\text{Department}\left(\underset{a}{\text{Essn}}, \underset{b}{\text{Dependent\_name}}, \underset{c}{\text{Sex}}, \underset{d}{\text{Bdate}}, \underset{e}{\text{Relationship}}\right)$$

Previous Years' Question

What is the optimized version of the relation algebra expression $\pi_A\left(\pi_A\left(\sigma_F\left(\sigma_F\left(r\right)\right)\right)\right)$, where A_1, A_2 are sets of attributes in r with $A_1 \subset A_2$ and F_1, F_2 are Boolean expression based on the attributes in r?

1) $\pi_{A_1}\left(\sigma_{(F_1 \wedge F_2)}(r)\right)$

2) $\pi_{A_1}\left(\sigma_{(F_1 \wedge F_2)}(r)\right)$

3) $\pi_{A_2}\left(\sigma_{(F_1 \wedge F_2)}(r)\right)$

4) $\pi_{A_2}\left(\sigma_{(F_1 \vee F_2)}(r)\right)$

Sol: Option 1) **(GATE Set-3-2014)**

Chapter Summary

- **SQL :** SQL stands for Structured Query Language. It is used to add, delete, modify or obtain data.
- Types of SQL commands:
  DDL, DML, DQL, DCL
- SELECT Clause: SELECT clause is used to list all the desired attributes in the result of a query.
- A simple form of SQL query:

$$\text{SELECT } A_1, A_2, \ldots, A_n$$

$$\text{FROM } r_1, r_2, \ldots, r_m$$

$$\text{WHERE <Condition>;}$$

- **LIKE :** It is a operator that is used for pattern matching of strings.
- **Aggregate function in SQL:**
  SQL provides 5 inbuilt aggregate functions.
  1) Average : AVG **2)** Minimum : MIN
  3) Maximum : MAX **4)** Total : SUM
  5) Count : COUNT
- **Correlated nested query:** Two queries are said to be correlated when a condition in the WHERE clause of the inner query captures and utilizes some fields of the table name specified in the outer query.
- **Joins :** It is used to combine two or more relation based on common attributes between them.
- **Types of Joins:**
  1) Natural Join **2)** Inner Join
  3) Left Outer Join **4)** Right Outer Join
  5) Full Outer Join
- **Relational algebra:** It is a procedural query language.
- Fundamental operation on relational algebra:
- Selection(s)
- Projection(p)
- Union(U)
- Set-difference(-)
- Cartesian Product (x)
- Renaming(ρ)