RGPVNOTES.IN

Program : **B.Tech**

Subject Name: **Database Management System**

Subject Code: **IT-405**

Semester: **4th**

UNIT III

Introduction to SQL Structure Query Language (SQL) is a programming language used for storing and managing data in RDBMS. SQL was the first commercial language introduced for E.F Codd's Relational model. Today almost all RDBMS (MySQL, Oracle, Infomax, Sybase, MS Access) uses SQL as the standard database language. SQL is used to perform all type of data operations in RDBMS.

SQL Command

DDL: Data Definition Language

All DDL commands are auto-committed. That means it saves all the changes permanently in the database.

Command	Description
create	to create a new table or database
alter	for alteration
truncate	delete data from a table
drop	to drop a table
rename	to rename a table

DML: Data Manipulation Language

DML commands are not auto-committed. It means changes are not permanent to the database; they can be rolled back.

Command	Description
insert	to insert a new row
update	to update an existing row
delete	to delete a row
merge	merging two rows or two tables

TCL: Transaction Control Language

These commands are to keep a check on other commands and their effect on the database. These commands can annul changes made by other commands by rolling back to the original state. It can also make changes permanent.

Command	Description
commit	to permanently save
rollback	to undo the change
save point	to save temporarily

DCL: Data Control Language

Data control language provides a command to grant and take back authority.

Command	Description
grant	grant permission of the right

revoke	take back permission.

DQL: Data Query Language

Command	Description
select	retrieve records from one or more table

Create command
create is a DDL command used to create a table or a database.
Creating a Database
To create a database in RDBMS, *create* command is used. Following is the Syntax,
create database *database-name*;
Example for Creating Database
create database Test;
The above command will create a database named **Test**.
Creating a Table
Create command also used to create a table. We can specify names and datatypes of various columns along. Following is the Syntax,
create table *table-name*
{
column-name1 datatype1,
column-name2 datatype2,
column-name3 datatype3,
column-name4 datatype4
};
create table command will tell the database system to create a new table with the given table name and column information.
Example for creating Table
create table Student (id int, name varchar, age int);

alter command
Alter command used for alteration of table structures. There are various uses of *altering* command, such as,

- to add a column to the existing table
- to rename any existing column
- to change the datatype of any column or to modify its size.
- *Alter* is also used to drop a column.

Using alter command we can add a column to an existing table. Following is the Syntax,

alter table table-name add (column-name datatype);
Here is an Example for this, **alter table Student add (address char);**

To Add a column with Default Value
alter command can add a new column to an existing table with default values. Following is the Syntax,
alter table table-name add (column-name1 datatype1 default data);
Example **alter table Student add (dob date default '1-Jan-99');**

To Modify an Existing Column
Alter command used to modify data type of an existing column. Following is the Syntax,
alter table table-name modify (column-name datatype);
Here is an Example for this, **alter table Student modify (address varchar (30));**

The above command will modify the address column of the Student table

To Rename a column

Using alter command you can rename an existing column.

alter table table-name rename old-column-name to column-name;

Here is an Example for this, **alter table Student rename address to Location;**

The above command will rename address column to Location.

To Drop a Column

alter command also used to drop columns also.

alter table table-name drop(column-name);

Here is an Example for this, **alter table Student drop(address);**

The above command will drop the address column from the Student table.

truncate command

The truncate command removes all records from a table. However, this command will not destroy the table's structure. When we apply to truncate command on a table its Primary key is initialised. Following is its Syntax,

truncate table table-name

Example **truncate table Student;**

drop command

Drop query completely removes a table from the database. This command will also destroy the table structure. Following is its Syntax,

drop table table-name

Here is an Example explaining it. **Drop table Student;**

rename query

Rename command is used to rename a table. Following is its Syntax,

rename table old-table-name to new-table-name

Here is an Example explaining it. **Rename table Student to Student-record;**

DML Commands

1) INSERT command

Insert command is used to insert data into a table. Following is its general syntax,

INSERT into table-name values (data1, data2,)

example,

Consider a table Student with following fields.

S_id S_Name age

INSERT into Student values(101,'Adam',15);

The above command will insert a record into Student table.

S_id	S_Name	age
101	Adam	15

Example to Insert NULL value to a column

Both the statements below will insert a NULL value into the age column of the Student table.

INSERT into Student (id, name) values(102,'Alex');

Alternatively,

INSERT into Student values (102,'Alex', null);

The above command will insert only two column value another column is set to null.

S_id	S_Name	age
101	Adam	15
102	Alex	

Example to Insert Default value to a column
INSERT into Student values (103,'Chris', default)

S_id	S_Name	age
101	Adam	15
102	Alex	
103	Chris	14

Suppose the age column of student table has a default value of 14.

2) UPDATE command

Update command is used to update a row of a table. Following is its general syntax,
UPDATE table-name set column-name = value where condition;
example,
update Student set age=18 where s_id=102;

S_id	S_Name	age
101	Adam	15
102	Alex	18
103	chris	14

Example
UPDATE Student set s_name='Abhi', age=17 where s_id=103;
The above command will update two columns of a record.

S_id	S_Name	age
101	Adam	15
102	Alex	18
103	Abhi	17

3) Delete command

Delete command is used to delete data from a table. Delete command can also be used with the condition to delete a row. Following is its general syntax,

DELETE from table-name;
Example
DELETE from Student;
The above command will delete all the records from the Student table.
Example to Delete a Record from a Table
Consider the following Student table

S_id	S_Name	age
101	Adam	15
102	Alex	18
103	Abhi	17

DELETE from Student where s_id=103;
The above command will delete the record where s_id is 103 from the Student table.

S_id	S_Name	age
101	Adam	15
102	Alex	18

TCL command

Transaction Control Language (TCL) commands are used to manage transactions in the database. These are used to manage the changes made by DML statements. It also allows statements to be grouped into logical transactions.

Commit command

Commit command is used to save any transaction into the database permanently.

Following is Commit command's syntax,
commit;

Rollback command

This command restores the database to the last committed state. It is also used with savepoint command to jump to a save point in a transaction.

Following is Rollback command's syntax,
rollback to savepoint-name;
Save point command
savepoint command used to temporarily save a transaction so that you can roll back to that point whenever necessary.
Following is save point command's syntax,
savepoint savepoint-name;

DCL command

System: creating a session, table etc. are all types of system privilege.
Object: any command or query to work on tables comes under object privilege.
DCL defines two commands,
Grant: Gives user access privileges to the database.
Revoke: Take back permissions from the user.
Example **grant create session to username;**

S_id	s_Name	age	address
101	Adam	15	Noida
102	Alex	18	Delhi
103	Abhi	17	Rohtak
104	Ankit	22	Panipat

WHERE clause

Where clause is used to specify condition while retrieving data from the table. Where clause is used mostly with Select, Update and Delete query. If the condition specified by where clause is true, then only the result from the table is returned.

The syntax for WHERE clause

SELECT column-name1,
column-name2,
column-name3,
column-name N
from table-name WHERE [condition];
Example SELECT s_id, s_name, age, address
from Student WHERE s_id=101;

SELECT Query

The Select query is used to retrieve data from a table. It is the most used SQL query. We can retrieve complete tables, or partial by mentioning conditions using WHERE clause.

Syntax of SELECT Query
SELECT column-name1, column-name2, column-name3, column-nameN from table-name;
Example **SELECT s_id, s_name, age from Student.**

Like clause
Like clause is used as a condition in SQL query. Like clause compares data with an expression using wildcard operators. It is used to find similar data from the table.

Wildcard operators
Two wildcard operators are used in like clause.
Per cent sign % represents zero, one or more than one character.
Underscore sign _: represents only one character.
Example: - SELECT * from Student where s_name like 'A%';

Order by Clause
Order by clause is used with a Select statement for arranging retrieved data in sorted order. The Order by clause by default sort data in ascending order. To sort data in descending order DESC keyword is used with Order by clause.

The syntax of Order By
SELECT column-list|* from table-name order by asc|desc;
Example SELECT * from Emp order by salary;

Group by Clause
Group by clause is used to group the results of a SELECT query based on one or more columns. It is also used with SQL functions to group the result from one or more tables.
The syntax for using Group by in a statement.

SELECT column_name, function(column_name)
FROM table_name
WHERE condition
GROUP BY column_name
Example select name, salary
from Emp
where age > 25
group by salary

HAVING Clause
Having clause is used with SQL Queries to give a more precise condition for a statement. It is used to mention condition in Group based SQL functions, just like WHERE clause.
Syntax for having will be,
select column_name, function(column_name)
FROM table_name
WHERE column_name condition
GROUP BY column_name
HAVING function(column_name) condition
Example SELECT *
from sale group customer
having sum(previous_balance) > 3000

Distinct keyword

The distinct keyword is used with a Select statement to retrieve unique values from the table. Distinct removes all the duplicate records while retrieving from the database.
The syntax for DISTINCT Keyword
SELECT distinct column-name from table-name;
select distinct salary from Emp;

Moreover, & OR operator
AND and OR operators are used with Where clause to make more precise conditions for fetching data from database by combining more than one condition together.
Example
SELECT * from Emp WHERE salary < 10000 AND age > 25
SELECT * from Emp WHERE salary > 10000 OR age > 25

SQL Constraints
SQL Constraints are rules used to limit the type of data that can go into a table, to maintain the accuracy and integrity of the data inside the table.

Constraints can be divided into the following two types,
Column level constraints: limits only column data
Table-level constraints: limits whole table data

Constraints are used to make sure that the integrity of data is maintained in the database. Following are the most used constraints that can be applied to a table.

NOT NULL Constraint
NOT NULL constraint restricts a column from having a NULL value. Once NOT NULL constraint is applied to a column, you cannot pass a null value to that column.
Ex. CREATE table Student (s_id int NOT NULL, Name varchar (60), Age int);

UNIQUE Constraint
UNIQUE constraint ensures that a field or column will only have unique values. A UNIQUE constraint field will not have duplicate data.
Ex. CREATE table Student (s_id int NOT NULL UNIQUE, Name varchar (60), Age int);

Primary Key Constraint
Primary key constraint uniquely identifies each record in a database. A Primary Key must contain unique value and it must not contain a null value.
Ex. CREATE table Student (s_id int PRIMARY KEY, Name varchar (60) NOT NULL, Age int);

Foreign Key Constraint
FOREIGN KEY is used to relate two tables. The foreign KEY constraint is also used to restrict actions that would destroy links between tables.
Ex. CREATE table Order_Detail (order_id int PRIMARY KEY,
order_name varchar (60) NOT NULL,
c_id int FOREIGN KEY REFERENCES Customer_Detail(c_id));
On Delete Cascade: This will remove the record from the child table if that value of the foreign key is deleted from the main table.
On Delete Null: This will set all the values in that record of child table as NULL, for which the value of the foreign key is deleted from the main table.

CHECK Constraint

A check constraint is used to restrict the value of a column between a range. It performs check on the values, before storing them into the database. It's like condition checking before saving data into a column.
create table Student (s_id int NOT NULL CHECK (s_id > 0),
Name varchar (60) NOT NULL, Age int);

SQL Functions
SQL provides many built-in functions to perform operations on data. These functions are useful while performing mathematical calculations, string concatenations, sub-strings etc. SQL functions are divided into two categories,

Aggregate Functions: -These functions return a single value after calculating from a group of values. Following are some frequently used aggregate functions.
AVG (), COUNT ()

Scalar Functions: -
Scalar functions return a single value from an input value. Following are soe frequently used Scalar Functions.
UCASE ()
UCASE function is used to convert the value of string column to the Uppercase character.

Join in SQL
SQL Join is used to fetch data from two or more tables, which is joined to appear as a single set of data. SQL Join is used for combining column from two or more tables by using values common to both tables. Join Keyword is used in SQL queries for joining two or more tables. Minimum required condition for joining table is (n-1) where n, is a number of tables. A table can also join to itself known as, Self-Join.

The SQL **Joins** clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each.
Consider the following two tables –
Table 1 – CUSTOMERS Table

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
| 1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
| 2 | Khilan   |  25 | Delhi     |  1500.00 |
| 3 | kaushik  |  23 | Kota      |  2000.00 |
| 4 | Chaitali |  25 | Mumbai    |  6500.00 |
| 5 | Hardik   |  27 | Bhopal    |  8500.00 |
| 6 | Komal    |  22 | MP        |  4500.00 |
| 7 | Muffy    |  24 | Indore    | 10000.00 |
```

Table 2 – ORDERS Table

```
-----+---------------------+-------------+--------+
|OID | DATE                | CUSTOMER_ID | AMOUNT |
+-----+---------------------+-------------+--------+
| 102 | 2009-10-08 00:00:00 |           3 |   3000 |
| 100 | 2009-10-08 00:00:00 |           3 |   1500 |
| 101 | 2009-11-20 00:00:00 |           2 |   1560 |
| 103 | 2008-05-20 00:00:00 |           4 |   2060 |
+-----+---------------------+-------------+--------+
```

Now, let us join these two tables in our SELECT statement as shown below.

```
SQL> SELECT ID, NAME, AGE, AMOUNT
  FROM CUSTOMERS, ORDERS
  WHERE CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

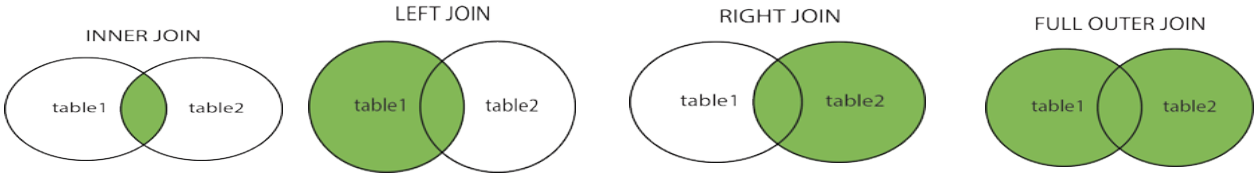
This would produce the following result.

```
+----+----------+-----+--------+
| ID | NAME     | AGE | AMOUNT |
+----+----------+-----+--------+
| 3 | kaushik  |  23 |   3000 |
| 3 | kaushik  |  23 |   1500 |
| 2 | Khilan   |  25 |   1560 |
| 4 | Chaitali |  25 |   2060 |
+----+----------+-----+--------+
```

Here, it is noticeable that the join is performed in the WHERE clause. Several operators can be used to join tables, such as =, <, >, <>, <=, >=,!=, BETWEEN, LIKE, and NOT; they can all be used to join tables. However, the most common operator is the equal to symbol.

There are several types of joins available in SQL –

- INNER JOIN − returns rows when there is a match in both tables.
- LEFT JOIN − returns all rows from the left table, even if there are no matches in the right table.
- RIGHT JOIN − returns all rows from the right table, even if there are no matches in the left table.
- FULL JOIN − returns rows when there is a match in one of the tables.
- SELF JOIN − is used to join a table to itself as if the table were two tables, temporarily renaming at least one table in the SQL statement.
- CARTESIAN JOIN − returns the Cartesian product of the sets of records from the two or more joined tables.



Example: -

Orders

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID

Customers

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country

Inner Join: -

```
1. SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders
INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;
2. SELECT Orders.OrderID, Customers.CustomerName, Shippers.ShipperName
FROM ((Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID)
INNER JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID);
```

Left Join: -

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
```

```
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerName;
```

Right Join: -

```
SELECT Orders.OrderID, Employees.LastName, Employees.FirstName
FROM Orders
RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
ORDER BY Orders.OrderID;
```

Full Outer Join: -

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;
```

Self-Join: -

```
SELECT A. CustomerName AS CustomerName1, B.CustomerName AS CustomerName2, A.City
FROM Customers A, Customers B
WHERE A.CustomerID <> B.CustomerID
AND A.City = B.City
ORDER BY A.City;
```

Query Processing and Optimization

Query Processing

Query Processing is a procedure of transforming a high-level query (such as SQL) into a correct and efficient execution plan expressed in low-level language. A query processing selects a most appropriate plan that is used in responding to a database request. When a database system receives a query for update or retrieval of information, it goes through a series of compilation steps, called execution plan.

Phases are: -

- In the first phase called syntax checking phase, the system parses the query and checks that it follows the syntax rules or not.
- It then matches the objects in the query syntax with the view tables and columns listed in the system table.
- Finally, it performs the appropriate query modification. During this phase, the system validates the user privileges and that the query does not disobey any integrity rules.
- The execution plan is finally executing to generate a response.

So, query processing is a stepwise process.

- The user gives a query request, which may be in QBE or another form. This is first transformed into a standard high-level query language, such as SQL.
- This SQL query is read by syntax analyser so that it can be check for correctness.
- At this step, the syntax analyser uses the grammar of SQL as input and the parser portion of the query processor check the syntax and verify whether the relation and attributes of the requested query are defined in the database.
- At this stage, the SQL query is translated into an algebraic expression using various rules.
- So that the process of transforming a high-level SQL query into a relational algebraic form is called Query Decomposition.
- The relational algebraic expression now passes to the query optimiser. Here optimisation is performed by substituting equivalent expression depends on the factors such that the existence of specific database structures, whether a given file is stored, the presence of different indexes & so on.

- Query optimisation module work in tandem with the join manager module to improve the order in which joins are performed. At this stage, the cost model and several other estimation formulas are used to rewrite the query.
- The modified query is written to utilise system resources to bring the optimal performance.
- The query optimiser then generates an action plan also called an execution plan. These action plans are converted into a query code that are finally executed by a runtime database processor.
- The runtime database processor estimated the cost of each action plan and chose the optimal one for the execution.

Query Analyzer

- The syntax analyser takes the query from the users, parses it into tokens and analyses the tokens and their order to make sure they follow the rules of the language grammar.
- Is an error is found in the query submitted by the user, it is rejected, and an error code together with an explanation of why the query was rejected is a return to the user.

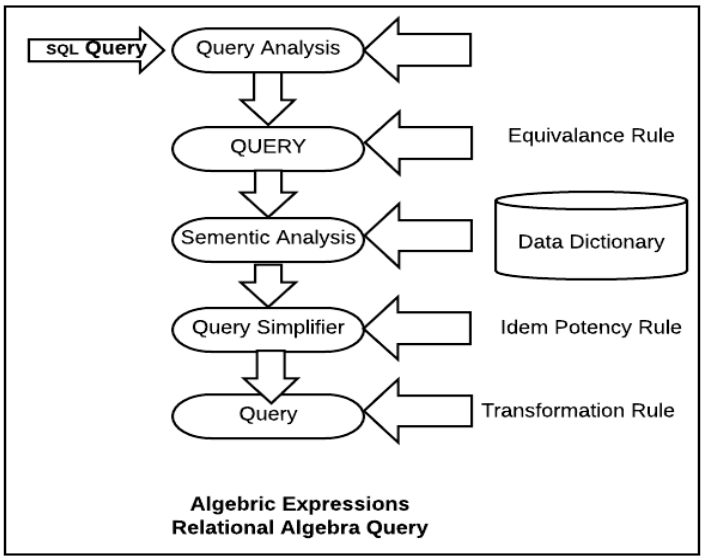
A simple form of the language grammar that could use to implement SQL statement is given bellow:

- QUERY = SELECT + FROM + WHERE
- SELECT = 'SELECT' + <CLOUMN LIST>
- FROM = 'FROM' + <TABLE LIST>
- WHERE = 'WHERE' + VALUE1 OP VALUE2
- VALUE1 = VALUE / COLUMN NAME
- VALUE2 = VALUE / COLUMN NAME
- OP = >, <, >=, <=, =, <>

Query Decomposition

The query decomposition is the first phase of the query processing whose aims are to transfer the high-level query into a relational algebra query and to check whether that query is syntactically and semantically correct.

- Thus, the query decomposition starts with a high-level query and transform into a query graph of low-level operations, which satisfy the query.
- The SQL query is decomposed into query blocks (low-level operations), which form the basic unit.
- Hence nested queries within a query are identified as separate query blocks.
- The query decomposer goes through five stages of processing for decomposition into low-level operation and translation into algebraic expressions.



Phases of Query Processing

1) Query Analysis: -

- During the query analysis phase, the query is syntactically analysed using the programming language compiler (parser).
- A syntactically legal query is then validated, using the system catalogue, to ensure that all data objects (relations and attributes) referred to by the query are defined in the database.
- The type specification of the query qualifiers and result is also checked at this stage.
  Let us consider the following query :
  SELECT emp_nm FROM EMPLOYEE WHERE emp_desg>100
  This query will be rejected because the comparison ">100" is incompatible with the data type of emp_desg which is a variable character string.

QUERY TREE NOTATIONS: -

At the end of the query analysis phase, the high-level query (SQL) is transformed into some internal representation that is more suitable for processing. This internal representation is typically a kind of query tree.

A Query Tree is a tree data structure that corresponds expression.

A Query Tree is also called a relational algebra tree.

- The leaf node of the tree, representing the base input relations of the query.
- Internal nodes of the tree representing an intermediate relation, which is the result of applying operation in the algebra.
- The root of the tree representing a result of the query.
- The sequence of operations is directed from ***leaf to root***.

The query tree is executed by executing a

- The resulting relation then replaces the internal.
- The execution is terminated when the root relation for the query.

Example: -

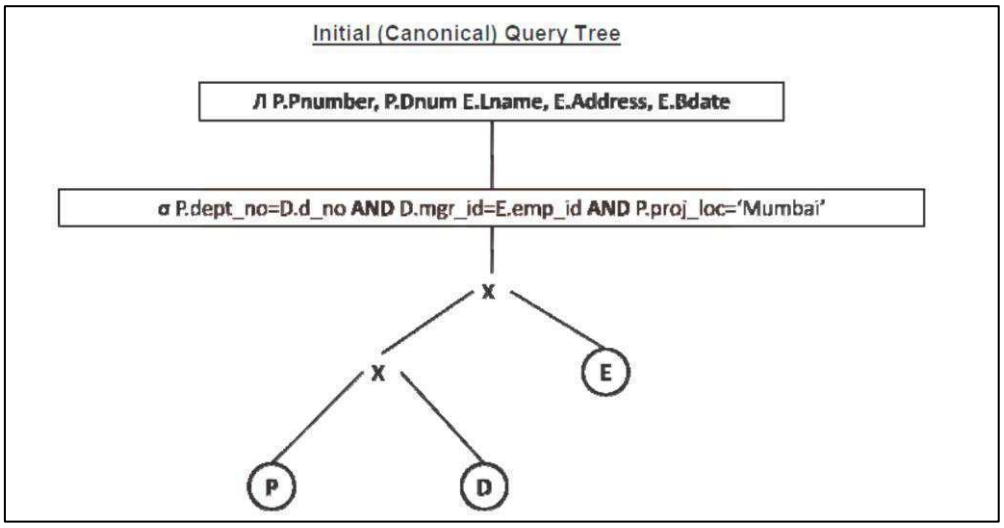
SELECT (P. proj_no, P.dept_no, E.name, E.add, E.dob)
FROM PROJECT P, DEPARTMENT D, EMPLOYEE
WHERE P. dept_no = D.d_no AND D.mgr_id = E.emp_id
Moreover, P. proj_loc = 'Mumbai' ;

Mumbai-PROJ ← σ proj_loc = 'Mumbai' (PROJECT)
CONT-DEPT ← (Mumbai-PROJ ⋈ dept_no = d_no DEPARTMENT)

PROJ-DEPT-MGR← (CONT-DEPT ⋈ mgr_id=emp_id EMPLOYEE)
RESULT← Π proj_no, dept_no, name, add, dob (PROJ-DEPT-MGR)

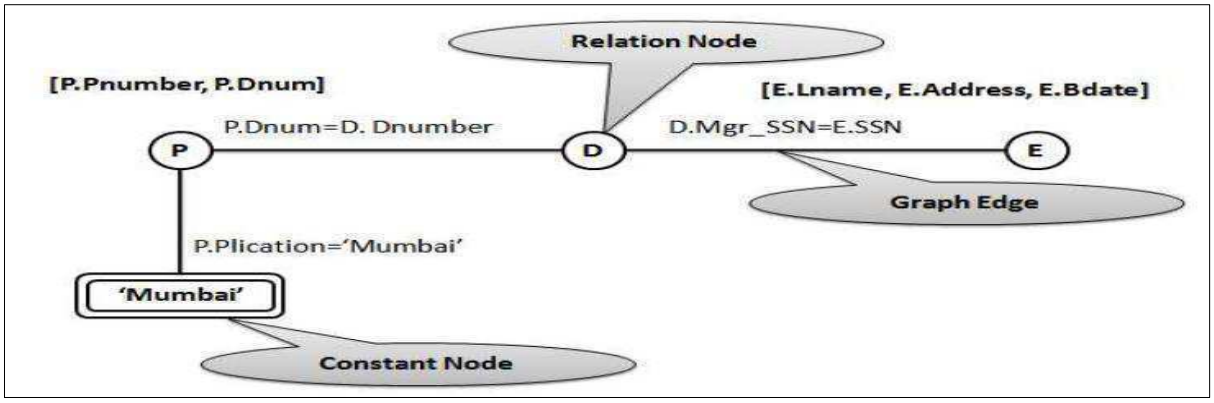
The three relations PROJECT, DEPARTMENT, EMPLOYEE is representing as a leaf nodes P, D and E, while the relational algebra operations of the represented by internal tree nodes.

- The same SQL query can have man different relational algebra expressions and hence many different query trees.
- The query parser typically generates a standard initial (canonical) query tree.



QUERY GRAPH NOTATIONS

- Query graph is sometimes also used for the representation of a query. In query graph representation, the relations in the query are represented **relation nodes** are displayed as a SINGLE CIRCLE.
- The constant values from the query selection (proj_loc = 'Mumbai') are represented by a **constant node**, displayed as a DOUBLE CIRCLE.
- The **selection and join conditions are represented as a Graph Edge** (e.g. P. dept_no = p.dept_num).
- Finally, the attributes retrieve from the relation are displayed in square brackets (P. proj_num, P.dept_no).



2) Query Normalization: -

- The primary phase of the normalisation is to avoid redundancy. The normalisation phase converts the query into a normalised form that can be more easily manipulated.
- In the normalisation phase, a set of equivalence rules are applied so that the projection and selection operations included on the query are simplified to avoid redundancy.
- The projection operation corresponds to the SELECT clause of the SQL query and the selection operation

correspond to the predicate found in WHERE clause.

- The equivalency transformation rules that are applied to SQL query is shown in the following table, in which UNARYOP means UNARY operation, BINOP means BINARY operation and REL1, REL2, REL3 are the relations.

	Rule Name	Rule Description
1.	Commutative of a UNARY operation	UNARYOP1 UNARYOP2 REL ↔ UNARYOP2 UNARYOP1 REL
2.	Commutative of BINARY operation	REL1 BINOP (REL2 BINOP REL3) ↔ (REL1 BINOP REL2) BINOP REL3
3.	Idempotency of UNARY operations UNARYOP1 UNARYOP2 REL	UNARYOP REL
4.	Distributivity of UNARY operations	UNARYOP1 (REL1 BINOP REL2) ↔ UNARYOP (REL1) BINOP UNARYOP (REL2)
5.	Factorisation ofUNARY operations	UNARIOP (REL1) BINOP UNARYOP (REL2) ↔ UNARYOP (REL1 BINOP REL2)

3) Semantic Analyzer: -

- The objective of this phase of query processing is to reduce the number of predicates.
- The semantic analyser rejects the normalised queries that are incorrectly formulated.
- A query is incorrectly formulated if components do not contribute to the generation of the result. This happens in the case of missing join specification.
- A query is contradictory if its predicate cannot satisfy by any tuple in the relation. The semantic analyser examines the relational calculus query (SQL) to make sure it contains only data objects that are a table, columns, views, indexes that are defined in the database catalogue.
- It makes sure that each object in the query is referenced correctly according to its data type.
- In the case of missing join specifications, the components do not contribute to the generation of the results, and thus, a query may be incorrectly formulated.
- A query is contradictory if its predicates cannot be satisfied by any of the tuples.
  For example: -
  (emp_des = 'Programmer' ∧ emp_des = 'Analyst')
  As an employee cannot be both 'Programmer' and 'Analyst' simultaneously, the above predicate on the EMPLOYEE relation is contradictory.

Example of Correctness and Contradictory: -

Incorrect Formulated: - (Missing Join Specification)

```
SELECT p. projno, p.proj_location
FROM project p, viewing v, department d
WHERE d. dept_id = v.dept_id AND d.max_budj >= 8000
AND d.mgr = 'Methew'
```

Contradictory: - **(Predicate cannot satisfy)**

```
SELECT p. proj_no, p.proj_location
FROM project p, cost_proj c, department d
WHERE d.max_budj >80000 AND d.max_budj < 50000 AND
d.dept_id = v. dept_id AND v.proj_no = p.proj_no
```

4.Query Simplifier: -

The objectives of this phase are to detect redundant qualification, eliminate common sub-expressions and transform sub-graph to semantically equivalent but easier and efficiently computed form.

Why simplify? :-
Commonly integrity constraints view definitions, and access restrictions are introduced into the graph at this stage of analysis so that the query must be simplified as much as possible.
Integrity constraints define constants which must hold for all state of the database, so any query that contradicts integrity constraints must be avoided and can be rejected without accessing the database.

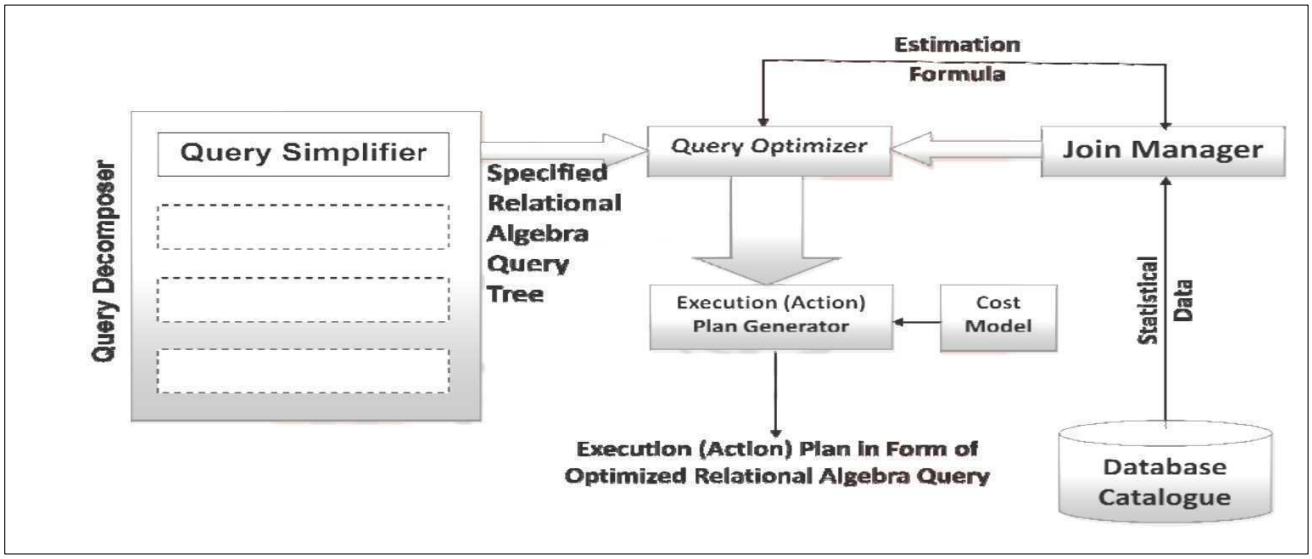
The final form of simplification is obtaining by applying **idempotency rules** of Boolean algebra.

	Description	Rule Format
1.	**PRED** AND **PRED** = **PRED**	P ∧ (P) = P
2.	**PRED** AND TRUE = **PRED**	P ∨ TRUE = P
3.	**PRED** AND FALSE = FALSE	P ∧ FALSE = FALSE
4.	**PRED** AND NOT**(PRED)** = FALSE	P ∧ (~P) = FALSE
5.	**PRED1** AND **(PRED1** OR **PRED2) = PRED1**	P1 ∧ (P1 ∨ P2) = P1
6.	**PRED** OR **PRED** = **PRED**	P ∨ (P) = P
7.	**PRED** OR TRUE = TRUE	P ∨ TRUE = TRUE
8.	**PRED** OR FALSE = **PRED**	P ∨ FALSE = P
9.	**PRED** OR NOT**(PRED**) = TRUE	P ∨ (~P) = TRUE
10.	**PRED1** OR (**PRED1** AND **PRED2**) = **PRED1**	P1 ∨ (P1 ∧ P2) = P1

5) Query Restructuring: -

- In the final stage of the query decomposition, the query can be restructured to give a more efficient implementation.
- Transformation rules are used to convert one relational algebra expression into an equivalent form that is more efficient.
- The query can now be regarded as a relational algebra program, consisting of a series of operations on the relation.

Query Optimization



Query Optimization Phases

The primary goal of query optimisation is of choosing an efficient execution strategy for processing a query.

- The query optimiser attempts to minimise the use of specific resources (mainly the number of I/O and CPU time) by selecting the best execution plan (access plan).
- A query optimisation starts during the validation phase by the system to validate the user has

appropriate privileges.

- Now an action plan is generating to perform the query.

The Relational algebra query tree is generated by the query simplifier module of query decomposer.

- Estimation formulas used to determine the cardinality of the intermediate result table.
- A cost Model
- Statistical data from the database catalogue.

The output of the query optimiser is the execution plan in the form of optimised relational algebra query.

- A query typically has many possible execution strategies, and the process of choosing a suitable one for processing a query is known as Query Optimization.
- The fundamental issues in Query Optimization are:
  - How to use available indexes.
  - How to use memory to accumulate information and perform immediate steps such as sorting.
  - How to determine the order in which joins should be performed.
- The term query optimisation does not mean always giving an optimal (best) strategy as the execution plan.
- It is just a responsibly efficient strategy for execution of the query.
- The decomposed query block of SQL is translating into an equivalent extended relational algebra expression and then optimised.

There are two main techniques for implementing Query Optimization:

The **first** technique is based on ***Heuristic Rules*** for ordering the operations in a query execution strategy. The **second** technique involves the ***systematic estimation*** of the cost of the different execution strategies and choosing the execution plan with the *lowest cost*.

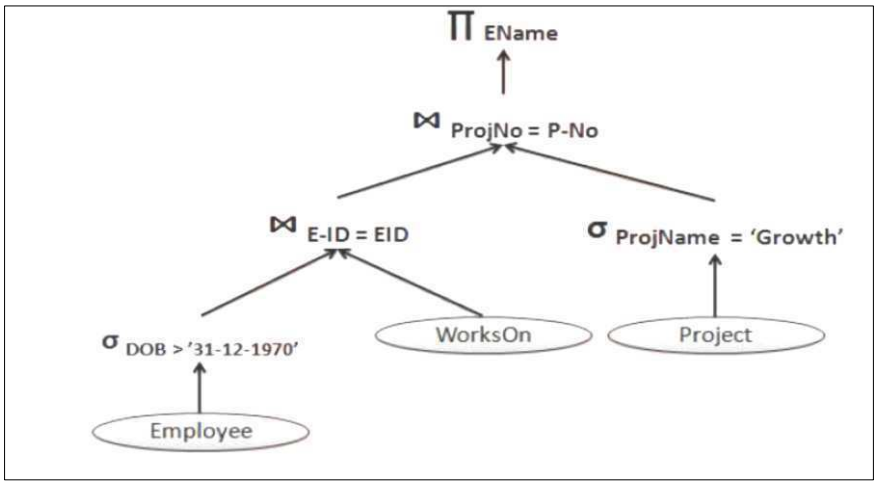
- Semantic query optimisation is used with the combination with the heuristic query transformation rules.
- It uses constraints specified on the database schema such as unique attributes and other more complex constraints, to modify one query into another query that is more efficient to execute.

1. Heuristic Rules: -

- The heuristic rules are used as an optimisation technique to modify the internal representation of the query. Usually, heuristic rules are used in the form of query tree of query graph data structure, to improve its performance.
- One of the main heuristic rules is to apply SELECT operation before applying the JOIN or other BINARY operations.
- This is because the size of the file resulting from a binary operation such as JOIN is usually a multi-value function of the sizes of the input files.
- The SELECT and PROJECT reduced the size of the file and hence, should be applied before the JOIN or other binary operation.
- Heuristic query optimiser transforms the initial (canonical) query tree into final query tree using equivalence transformation rules. This final query tree is efficient to execute.

For example, consider the following relations:
Employee (EName, EID, DOB, EAdd, Sex, ESalary, EDeptNo)
Department (DeptNo, DeptName, DeptMgrID, Mgr_S_date)
DeptLoc (DeptNo, Dept_Loc)
Project (ProjName, ProjNo, ProjLoc, ProjDeptNo)
WorksOn (E-ID, P-No, Hours)
Dependent (E-ID, DependName, Sex, DDOB, Relation)
Now let us consider the query in the above database to find the name of employees born after 1970 who work on a project named 'Growth'.
SELECT EName

FROM Employee, WorksOn, Project
WHERE ProjName = 'Growth' AND ProjNo = P-No
AND EID = E-ID AND DOB > '31-12-1970';



General Transformation Rules: -

Transformation rules are used by the query optimiser to transform one relational algebra expression into an equivalent expression that is more efficient to execute.

- A relation is considering as equivalent of another relation if two relations have the same set of attributes in a different order but representing the same information.
- These transformation rules are used to restructure the initial (canonical) relational algebra query tree attributes during query decomposition.

1. Cascade of σ :-

σ c1 AND c2 AND …AND cn (R) = σ c1 (σ c2 (… (σ cn (R))…))

2. Commutativity of σ :-

σ C1 (σ C2 (R)) = σ C2 (σ C1 (R))

3. Cascade of Л :-

Л List1 (Л List2 (… (Л List n (R))…)) = Л List1 (R)

4. Commuting σ with Л :-

Л A1, A2,A3…An (σ C (R)) = σ C (Л A1,A2,A3…An (R))

5. Commutativity of ⋈ AND x :-

R ⋈ c S = S ⋈ c R

R x S = S x R

6. Commuting σ with ⋈ or x :-

If all attributes in selection condition c involved only attributes of one of the relation schemas (R).

σ c (R ⋈ S) = (σ c (R)) ⋈ S

Alternatively, selection condition c can be written as (c1 AND c2) where condition c1 involves only attributes of R and condition c2 involves only attributes of S then:

σ c (R ⋈ S) = (σ c1 (R)) ⋈ (σ c2 (S))

7. Commuting Л with ⋈ or x:-

- The projection list L = {A1, A2,..An,B1,B2,…Bm}.
- A1…An attribute of R and B1…Bm attributes of S.
- Join condition C involves only attributes in L then:
- ЛL (R ⋈ c S) = (ЛA1,…An (R)) ⋈c (ЛB1,…Bm(S))

8. Commutativity of SET Operation: -

- R ⋃ S = S ⋃ R
- R ⋂ S = S ⋂ R

Minus (R-S) is not commutative.

9. Associatively of ⋈, x, ⋂, and ⋃:-

- If Ø stands for any one of this operation throughout the expression then:

(R Ø S) Ø T = R Ø (S Ø T)

10. Commutativity of σ with SET Operation: -

- If Ø stands for any one of three operations (⋃, ⋂,and-) then :

σ c (R Ø S) = (σ c (R)) ⋃ (σ c (S))

Л c (R Ø S) = (Л c (R)) ⋃ (Лc (S))

11. The Л operation commute with ⋃:-

Л L (R ⋃ S) = (Л L(R)) ⋃ (Л L(S))

12. Converting a (σ, x) sequence with ⋃

(σ c (R x S)) = (R ⋈ c S)

Heuristic Optimization Algorithm: -

The Database Management System use Heuristic Optimization Algorithm that utilises some of the transformation rules to transform an initial query tree into an optimised and efficient executable query tree.

- The steps of the heuristic optimisation algorithm that could be applied during query processing and optimization are:

  Step-1: -
- Perform SELECT operation to reduce the subsequent processing of the relation:
- Use the transformation rule 1 to break up any SELECT operation with conjunctive condition into a cascade of SELECT operation.

Step-2: -

- Perform commutativity of SELECT operation with other operation at the earliest to move each SELECT operation down to query tree.
- Use transformation rules 2, 4, 6 and 10 concerning the commutativity of SELECT with other operation such as unary and binary operations and move each select operation as far down the tree as is permitted by the attributes involved in the SELECT condition. Keep selection predicates on the same relation together.

Step-3: -

- Combine the Cartesian product with a subsequent SELECT operation whose predicates represents the join condition into a JOIN operation.
- Use the transformation rule 12 to combine the Cartesian product operation with the subsequent SELECT operation.

Step-4: -

- Use the commutativity and associativity of the binary operations.
- Use transformation rules 5 and 9 concerning commutativity and associativity to rearrange the leaf nodes of the tree so that the leaf node with the most restrictive selection operation is executed first in the query tree representation. The most restrictive SELECT operation means:
- Either the one that produces a relation with the fewest tuples or with the smallest size.
- The one with the smallest selectivity.

Step-5: -

- Perform the projection operation as early as possible to reduce the cardinality of the relationship and the subsequent processing of the relation and move the Projection operations as far down the query tree as possible.
- Use transformation rules 3, 4, 7 and 10 concerning the cascading and commuting of projection operations with other binary operation. Break down and move the projection attributes down the tree as far as needed. Keep the projection attributes in the same relation together.

Step-6: -

- Compute common expression once.

- Identify sub-tree that represent a group of operations that can be executed by a single algorithm.

2. Cost Estimation in Query Optimization: -

- The main aim of query optimisation is to choose the most efficient way of implementing the relational algebra operations at the lowest possible cost.
- Therefore, the query optimiser should not depend solely on heuristic rules, but it should also estimate the cost of executing the different strategies and find out the strategy with the minimum cost estimate.
- The method of optimising the query by choosing a strategy those result in minimum cost is called cost-based query optimisation.
- The cost-based query optimisation uses the formula that estimates the cost for some options and selects the one with the lowest cost and the most efficient to execute.
- The cost functions used in query optimisation are estimates and not exact cost functions.
- The cost of an operation is heavily dependent on its selectivity, that is, the proportion of select operation(s) that forms the output.
- In general, the different algorithms are suitable for low or high selectivity queries. For the query optimiser to choose the suitable algorithm for an operation an estimate of the cost of executing that algorithm must be provided.
- The cost of an algorithm depends on cardinality of its input.
- To estimate the cost of different query execution strategies, the query tree is viewed as containing a series of basic operations which are linked to perform the query.
- It is also essential to know the expected cardinality of an operation's output because of this form the input to the next operation.

Cost Components of Query Execution: -

- The success of estimating the size and cost of standard relational algebra operations depends on the amount the accuracy of statistical data information stored with DBMS.

The cost of executing the query includes the following components: -

- Access cost to secondary storage.
- Storage cost.
- Computation cost.
- Memory uses cost.
- Communication cost.

1. **Access cost to secondary storage: -**

Access cost is the cost of searching for reading and writing data blocks (containing the number of tuples) that reside on secondary storage, mainly on disk of the DBMS.

The cost of searching for tuples in the database relations depends on the type of access structures on that relation, such as ordering, hashing and primary or secondary indexes.

2. **Storage cost: -**

The storage cost is of storing any intermediate relations that are generated by the executing strategy for the query.

3. **Computation cost: -**

_ Computation cost is the cost of performing in-memory operations on the data buffers during query execution.

_ Such operations contain searching for and sorting records, merging records for a join and performing computation on a field value.

4. **Memory uses cost: -**

_ Memory uses cost a cost about the number of memory buffers needed during query execution.

5. **Communication cost: -**

_ It is the cost of transferring query and its results from the database site to the site of the terminal of query organisation.

_ Out of the above five cost components; the most important is the secondary storage access cost.

_ The emphasis of the cost minimisation depends on the **size** and **type** of database applications.
_ For example, in the smaller database the emphasis is on the minimising computing cost as because most of the data in the files involved in the query can be completely stored in the main memory. For a large databaprimarythe main emphasis is on minimising the access cost to the secondary device.

- For the distributed database, the communication cost is minimised as because many sites are involved in the data transfer.
- To estimate the cost of various execution strategies, we must keep track of any information that is needed for the cost function. This information may be stored in the database catalogue, where the query optimizer accesses it.
- Typically, the DBMS is expected to hold the following types of information in its system catalogue.

· The number of tuples in relation to R [nTuples(R)].
· The average record size in relation R.
· The number of blocks required to store relation R as [nBlocks(R)].
· The blocking factors in relation R (that is the number of tuples of R that fit into one block) as [bFactor(R)].
· Primary access method for each file.
· Primary access attributes for each file.
· The number of levels of each multilevel index I (primary, secondary or clustering) as [nLevelsA(I)].
· The number of first level index blocks as [nBlocksA (I)].
· The number of distinct values that appear for attribute A in relation R as [nDistinctA(R)].
· The minimum and maximum possible values for attribute A in relation R as [minA(R), maxA(R)].
· The selectivity of an attribute, which is the fraction of records satisfying an equality condition on the attribute.
· The selection cardinality of given attribute A in relation R as [SCA(R)]. The selection cardinality is the average number of tuples that satisfied an equality condition on attribute A.

- **Cost functions for SELECT Operation: -**

➢ Linear Search: -
  · [nBlocks(R)/2], if the record is found.
  · [nBlocks(R)], if no record satisfied the condition.
➢ Binary Search: -
  · [log2(nBlocks(R))], if equality condition is on key attribute, because SCA(R) = 1 this case.
  · [log2(nBlocks(R))] + [SCA(R)/bFactor(R)] – 1, otherwise.
➢ Using primary index or hash key to retrieve a single record: -
  · 1, assuming no overflow
➢ Equity condition on Primary key: -
  · [nLevelA(I) + 1]
➢ Equity condition on Non-Primary key: -
  · [nLevelA(I) + 1] + [nBlocks(R)/2]
➢ Using inequality condition on a secondary index (B+ Tree): -
  · [nLevelA(I) + 1] + [nLfBlocksA(I)/2] + [nTuples(R)/2]
➢ Equity condition on clustering index: -
  · [nLevelA(I) + 1] + [SCA(R)/bFactor(R)]
➢ Equity condition on non-clustering index: -
  · [nLevelA(I) + 1] + [SCA(R)]

Example of Cost Estimation for SELECT Operation: -
· Let us consider the relation EMPLOYEE having following attributes: -
EMPLOYEE (EMP-ID, DEPT-ID, POSITION, SALARY)
· Let us consider the following assumptions: -
o There is a hash index with no overflow on primary key attribute EMP-ID.
o There is a clustering index on foreign key attribute DEPT-ID.
o There is a B+-Tree index on the SALARY attribute.
· Let us also assume that the EMPLOYEE relation has the following statistics in the system catalog:

nTuples(EMPLOYEE) = 6

nTuples(EMPLOYEE)	=	6,000
bFactor(EMPLOYEE)	=	60
nBlocks(EMPLOYEE)	= =	nTuples(EMPLOYEE) / bFactor(EMPLOYEE) 6,000 / 60 = 100
$nDistinct_{DEPT-ID}$(EMPLOYEE)	=	1,000
$SC_{DEPT-ID}$(EMPLOYEE)	= =	nTuples(EMPLOYEE) / $nDistinct_{DEPT-ID}$(EMPLOYEE) 6,000 / 1,000 = 6
$nDistinct_{POSITION}$(EMPLOYEE)	=	20
$SC_{POSITION}$(EMPLOYEE)	= =	nTuples(EMPLOYEE) / $nDistinct_{POSITION}$(EMPLOYEE) 6,000 / 20 = 300
$nDistinct_{SALARY}$(EMPLOYEE)	=	1,000
SC_{SALARY}(EMPLOYEE)	= =	nTuples(EMPLOYEE) / $nDistinct_{SALARY}$(EMPLOYEE) 6,000 / 1,000 = 6
min_{SALARY}(EMPLOYEE)	=	20,000
max_{SALARY}(EMPLOYEE)	=	80,000
$nLevels_{DEPT-ID}$(I)	=	2
nLevelsSALARY(I)	=	2
nLfBlocksSALARY(I)	=	50

· Selection 1: - б EMP-ID = '106519' (EMPLOYEE)
· Selection 2: - б POSITION = 'MANAGER' (EMPLOYEE)
· Selection 3: - б DEPT-ID = 'SAP-04' (EMPLOYEE)
· Selection 4: - б SALARY = 30,000 (EMPLOYEE)
· Selection 5: - б POSITION = 'MANAGER' Λ DEPT-ID = 'SPA-04' (EMPLOYEE)
Now we will choose the query execution strategies by comparing the cost as follows:

Selection -1	The selection operation contains an equality condition on the primary key EMP-ID of the relation EMPLOYEE. Therefore, as the attribute EMP-ID is hashed we can use the strategy 3 to estimate the cost as one block. The estimated cardinality of the result relation is **SC_{EMP-ID} (EMPLOYEE) = 1**.
Selection -2	The attribute in the predicate is the non-key, non-indexed attribute. Therefore, we can improve on the linear search method, giving an estimated cost of 100 blocks. The estimated cardinality of the result relation is **$SC_{POSITION}$ (EMPLOYEE) = 300**.
Selection -3	The attribute in the predicate is a foreign key with a clustering index. Therefore, we can use strategy 7 to estimate the cost as **(2 + (6/30)) = 3 blocks**. The estimated cardinality of result relation is **$SC_{DEPT-ID}$ (EMPLOYEE) = 6**.
Selection -4	The predicate here involves a range search on the SALARY attribute, which has the B^+-Tree index. Therefore we can use the strategy 6 to estimate the cost as (2 + (50/2) + (6,000/2)) = 3027 blocks. Thus, the linear search strategy is used in this case, the estimated cardinality of the result relation is SC_{SALARY}(EMPLOYEE) = [6000*(8000-2000*2)/ (8000-2000)] = 4000.

Selection -5	While we are retrieving, each tuple using the clustering index, we can check whether they satisfied the first condition (POSITION = 'MANAGER'). We know that estimated cardinality of the second condition **SC DEPT-ID (EMPLOYEE) = 6**. Let us assume that this intermediate condition is S. then the number of distinct values of POSITION in S can be estimated as **[(6 + 20)/2] = 9**. Let us apply now the second condition using the clustering index on DEPT-ID, which has an estimated cost of 3 blocks. Thus, the estimated cardinality of the result relation will be SC POSITION (S) = 6/9 = 1, which would be correct if there is one manager for each branch.

Join operation is the most time-consuming operation to process. An estimate for the size (number of tuples) of the file that results after the JOIN operation is required to develop reasonably accurate cost functions for JOIN operations.

The JOIN operations define the relation containing tuples that satisfy a specific predicate F from the Cartesian product of two relations R and S.

Following table shows the different strategies for JOIN operations.

Strategies	**Cost Estimation**
Block nested-loop JOIN	**a) nBlocks(R) + (nBlocks(R) * nBlocks(S)) If the buffer has only one block. b) nBlocks(R) + [nBlocks(S) * (nBlocks(R)/(nBuffer-2))] If (nBuffer-2) blocks is there for R c) nBlocks(R) + nBlocks(S) If all blocks of R can be read into the database buffer**
Indexed nested-loop JOIN	**a) nBlocks(R) + nTuples(R) * (nLevelA(I) + 1) If join attribute A in S is a primary key b) nBlocks(R) + nTuples(R) * (nLevelA(I) + [SCA(R) / bFactor(R)]) If clustering index I is on attribute A.**
Sort-merge JOIN	**a) nBlocks(R) * [log2nBlocks(R)] + nBlocks(S) * [log2nBlocks(R)] For Sort b) nBlocks(R) + nBlocks(S) For Merge**
Hash JOIN	**a) 3 (nBlocks(R) + nBlocks(S)) If Hash index is in memory b) 2 (nBlocks(R) + nBlocks(S)) * [log (nBlocks(S)) - 1] + nBlocks(R) + nBlocks(S)**
Sort-merge JOIN	**a) nBlocks(R) * [log2nBlocks(R)] + nBlocks(S) * [log2nBlocks(R)] For Sort b) nBlocks(R) + nBlocks(S) For Merge**
Hash JOIN	**a) 3 (nBlocks(R) + nBlocks(S)) If Hash index is in memory b) 2 (nBlocks(R) + nBlocks(S)) * [log (nBlocks(S)) - 1] + nBlocks(R) + nBlocks(S)**

RGPVNOTES.IN

We hope you find these notes useful.

You can get previous year question papers at https://qp.rgpvnotes.in .

If you have any queries or you want to submit your study notes please write us at rgpvnotes.in@gmail.com

LIKE & FOLLOW US ON FACEBOOK
facebook.com/rgpvnotes.in