AIM: 1.1 Familiarization with installation of any DBMS.

Description:

Introduction to Oracle 10g Express Edition and Installation

a. Overview of Oracle 10g Express Edition:

Oracle Database 10g Express Edition (Oracle Database XE) is a free, downloadable version of the world's most capable relational database. Oracle Database XE is easy to install and easy to manage.

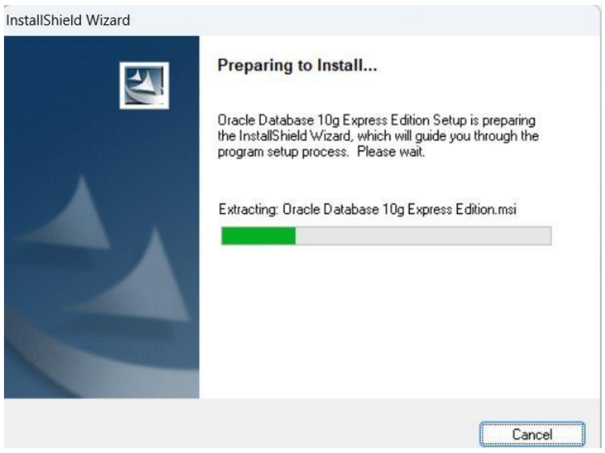
Interfaces

Database Home Page: Oracle XE included a web-based interface accessible through a browser, typically at http://localhost:8080/apex.

SQL Command Line: A simple command-line interface called "SQL Command Line" was included for direct SQL execution and basic administration tasks.

b. Installation Steps

Click on the first file named OracleXE.exe to begin your downloading



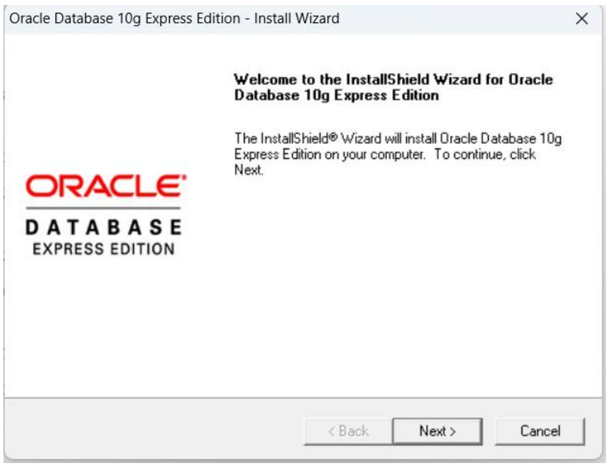
InstallShield Wizard

Preparing to Install...

Oracle Database 10g Express Edition Setup is preparing the InstallShield Wizard, which will guide you through the program setup process. Please wait.

Extracting: Oracle Database 10g Express Edition.msi

Click the Next button to begin to install the Oracle Database 10g Express Edition.

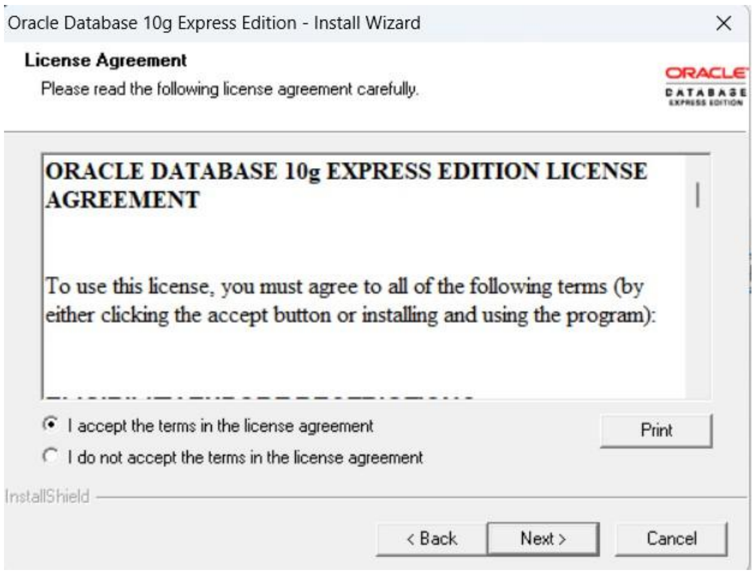


Oracle Database 10g Express Edition - Install Wizard

Welcome to the InstallShield Wizard for Oracle Database 10g Express Edition

The InstallShield® Wizard will install Oracle Database 10g Express Edition on your computer. To continue, click Next.

Select and click the Accept radio button followed by clicking the Next button to go to the next page.


Oracle Database 10g Express Edition - Install Wizard

License Agreement

Please read the following license agreement carefully.

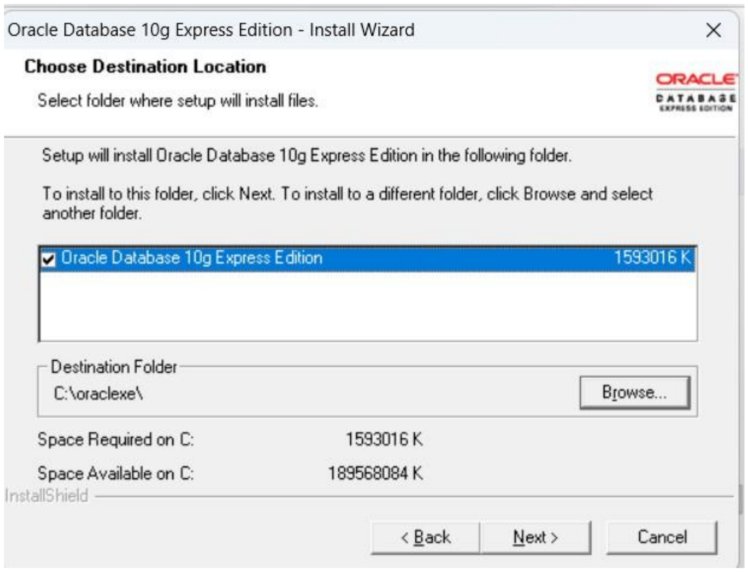
ORACLE DATABASE 10g EXPRESS EDITION LICENSE AGREEMENT

To use this license, you must agree to all of the following terms (by either clicking the accept button or installing and using the program):

[✓ I accept the terms in the license agreement]

I do not accept the terms in the license agreement

Select folder where setup will install files.


Oracle Database 10g Express Edition - Install Wizard

Choose Destination Location

Select folder where setup will install files.

Setup will install Oracle Database 10g Express Edition in the following folder.

To install to this folder, click Next. To install to a different folder, click Browse and select another folder.

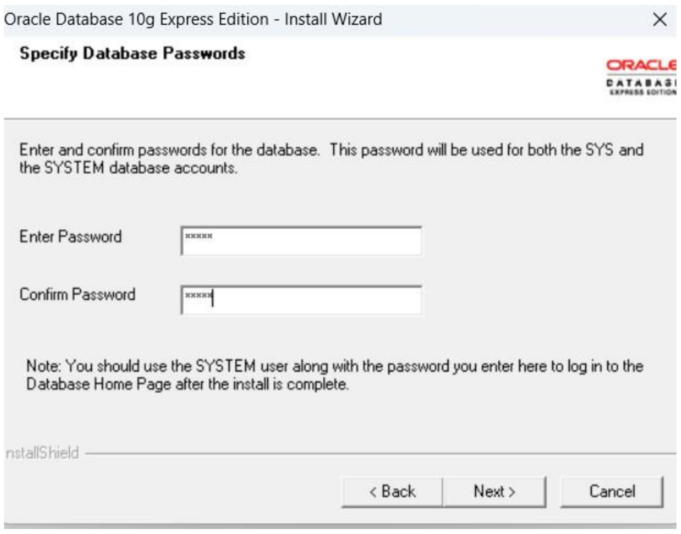
[✓] Oracle Database 10g Express Edition 1593016 K

Destination Folder: C:\oraclexe\

Space Required on C: 1593016 K

Space Available on C: 189568084 K

In the next dialog, you need to enter your password. You must remember this password since you need to use it to access your database installed in the Oracle server from your client computer later. The username is SYSTEM by default. This means that you need to use the SYSTEM as your username and the password you selected to open and access the database you created and installed in the Oracle server later.


Oracle Database 10g Express Edition - Install Wizard

Specify Database Passwords

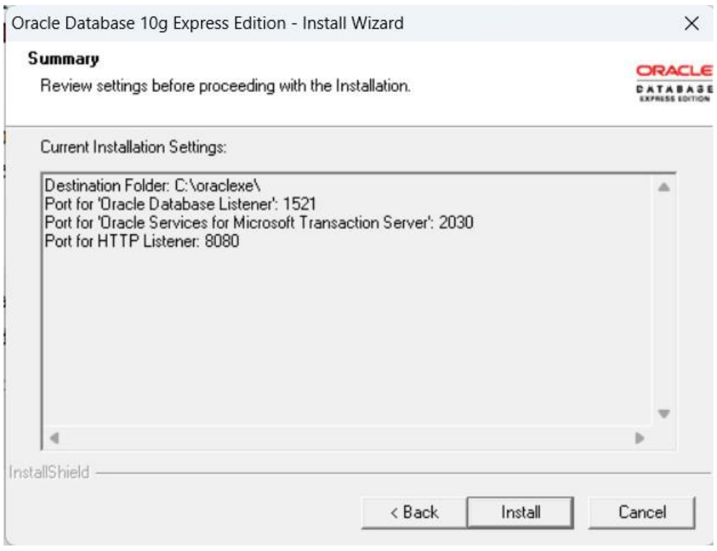
Enter and confirm passwords for the database. This password will be used for both the SYS and the SYSTEM database accounts.

Enter Password: *****

Confirm Password: *****

Note: You should use the SYSTEM user along with the password you enter here to log in to the Database Home Page after the install is complete.

Click the Next button after you entered your desired password, and then click the Install button to begin this installation.

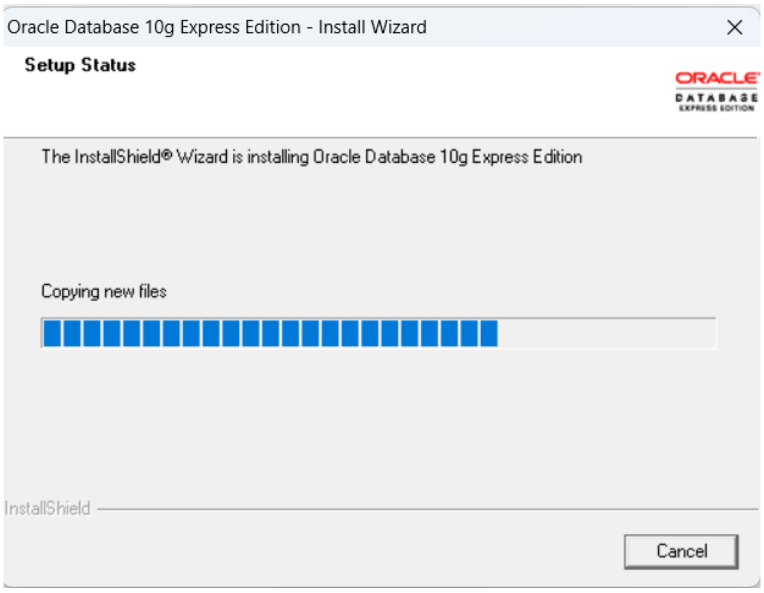

Oracle Database 10g Express Edition - Install Wizard

Summary

Review settings before proceeding with the Installation.

Current Installation Settings:

```
Destination Folder: C:\oraclexe\
Port for 'Oracle Database Listener': 1521
Port for 'Oracle Services for Microsoft Transaction Server': 2030
Port for HTTP Listener: 8080
```

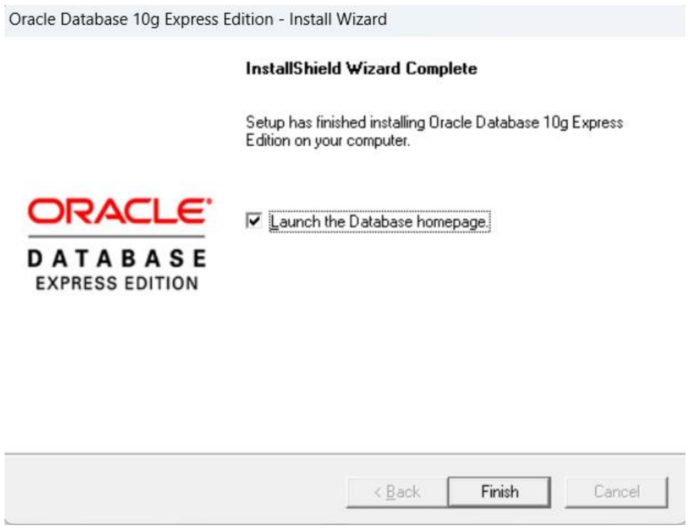

Oracle Database 10g Express Edition - Install Wizard

Setup Status

The InstallShield® Wizard is installing Oracle Database 10g Express Edition

Copying new files

When this installation is complete, a finish dialog is displayed



Oracle Database 10g Express Edition - Install Wizard

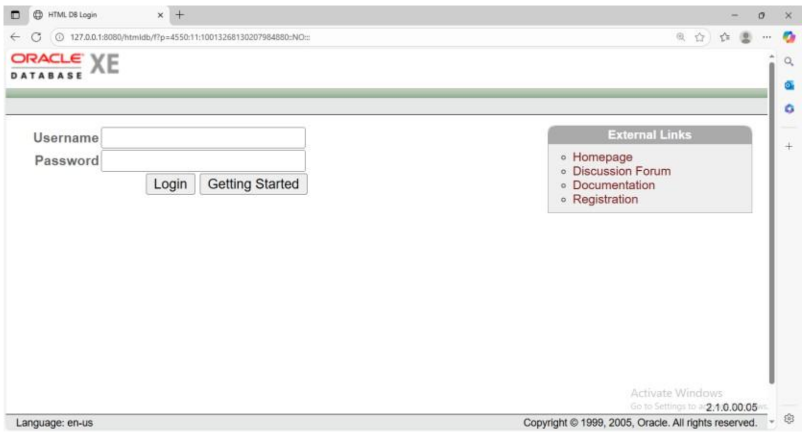
InstallShield Wizard Complete

Setup has finished installing Oracle Database 10g Express Edition on your computer.

[✓] Launch the Database homepage.

Click the Finish button to complete this installation.

The Database Login page is opened to allow you to enter your username and password to access the Oracle server to create, access and manipulate the database you created in this server.

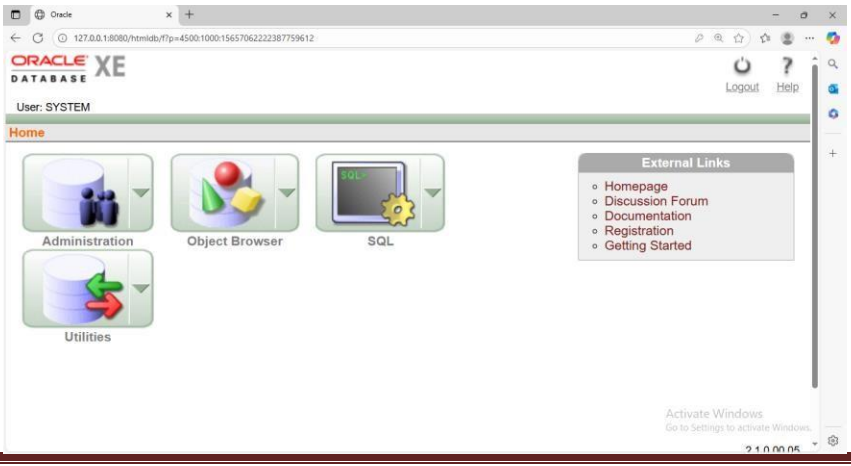


127.0.0.1:8080/htmldb/f?p=4550:11:1001326813020798488O::NO:::

Username	
Password	

Enter the SYSTEM as your username and the password you selected when you installed the server, and then click the Login button to open the home page.

There are four icons on the opened home page



127.0.0.1:8080/htmldb/f?p=4500:1000:1565706222387759612

User: SYSTEM

At this point, you have successfully finished the download and installation of the Oracle Database 10g Express Edition, which include the Oracle Database 10g Express server and the client. A Database Started Guide icon has been automatically added to your desktop on your computer. You can open this icon to follow the guide to access your database if you like. To connect your client to the server and start creating, accessing and manipulating your database, you need to open the Database home page by clicking the Start|All Program Files|Oracle Database 10g Express Edition|Go To Database Home Page. Then you can create and manipulate your database staring from the Object Browser icon.

AIM: 1.2 Implementing a University Database System.

Description:

a. Schema

It is overall design of the database. It tells how many tables are there, what are their attributes, and how they are related. It includes table names, column names(attributes), datatypes, and keys.

Syntax of Schema:

Table Name (column1 : datatype, column2: datatype, column3: datatype, ……., columnN: datype)

b. Attributes per Table

- **Students Table**
  Attributes: StudentID, StudentName, Major
- **Courses Table**
  Attributes: CourseID, CourseName, Credits
- **Enrollments Table**
  Attributes: StudentID, CourseID, EnrollmentDate
- **Instructors Table**
  Attributes: InstructorID, InstructorName, Phone
- **Course_Instructors Table**
  Attributes: CourseID, InstructorID

c. Schema for University Database:

Students (StudentID:string, StudentName:string, Major:string)

Courses (CourseID:string, CourseName:string, Credits:integer)

Enrollments (StudentID:string, CourseID:string, EnrollmentDate:date)

Instructors (InstructorID:integer, InstructorName:string, Phone:integer)

Course_Instructors (CourseID:string, InstructorID:integer)

AIM:2.1 Querying and modifying the database using Data Manipulation Language commands -select, insert, update, delete

Description:

DML COMMANDS are INSERT, UPDATE, DELETE and SELECT.

INSERT COMMAND:

This command is used to create data into the table which is already defined through DDL commands. The data can be entered in the form of rows and columns.

Syntax: INSERT INTO <Table name>
```
    (column1, [column2,............,columnN])
    values (column1value,column2value,...... ,columnNvalue);
                                 OR
    INSERT INTO <Table name>
    Values (column1value,column2value,…..,columnNvalue);
```

UPDATE COMMAND:

This command is used to modify or change or replace the existingdata of a table.

Syntax:
```
UPDATE <Table_name>
 Set <column1>=<column1value>
 [,<column2>=<column2value>,………
 ,<columnN>=<columnNvalue>]
 [where<condition>];
```

DELETE COMMAND:

This command is used to remove a single row or multiple rows of a table.

Syntax:
```
DELETE FROM<Table_name>
[where<condition>];
```

SELECT COMMAND:

This command is used to view a single row or multiple rows or single column or multiple columns of a table.

Syntax:
```
Select [distinct] column1,column2 as newname from table1,table2
Where condition
Group by columnname
Having condition
Order by columname asc|desc
```

Creating Students Table:
```
create table students
(
  rollno varchar2(30),
  name varchar2(30)
);
Table Created
```

Inserting Data into the table
```
insert into students values('24B11CS234','Bala');
1 row(s) inserted.

insert into students values('24B11CS381','Kiran');
1 row(s) inserted.
```

Displaying Data from the table

select * from students;

ROLLNO	NAME
24B11CS234	Bala
24B11CS381	Kiran

select name from students;

NAME
Bala
Kiran

select * from students9 where rollno='24B11CS234'

ROLLNO	NAME
24B11CS234	Bala

Deleting a row from the table

```
  delete from students where rollno='24B11CS381';
  1 row(s) deleted.
```

Updating a row in the table

```
  update students  set name='Bala Raju'  where rollno='24B11CS381';
  1 row(s) updated.
```

AIM: 2.2 Implementation of Aggregate Functions – sum, avg, min, max, count. Use group-by and having clause.

Description:

Aggregate Functions

AVG function: It can be used on numeric data or character data that contains only numeric's.

MAX function: It is used to find the maximum value of x. It can be used on any type of data.

MIN function: It is used to find the minimum value of x

SUM function: It sums the values and can be used on numeric data also.

COUNT(*): It counts the number of rows in the table or the number of row in the group including NULL.

Group by: The attributes given in the clauses are used to form groups. Tuples with the same value on all attributes in the group by clause are placed in one group.

Having: SQL applies predicates (conditions) in the having clause after groups have been formed, so aggregate function be used.

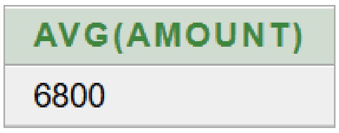
Source Table

```
select * from company;
companyn        amount
wipro             5000
ibm               8000
dell              9000
wipro             2000
dell              10000
```

Queries

Find the average salary of company

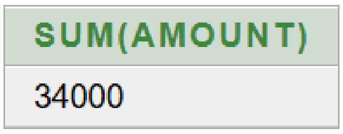
Select AVG(amount) from company;



AVG(AMOUNT)
6800

Find the Sum of salaries of company

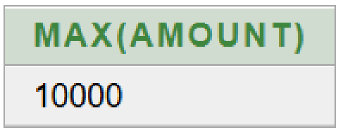
Select SUM(amount) from company;



SUM(AMOUNT)
34000

Find the Maximum amount of company

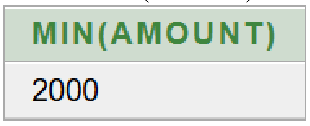
Select Max(amount) from company;



MAX(AMOUNT)
10000

Find the Minimum amount of company

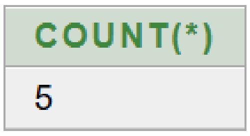
Select Min(amount) from company;



MIN(AMOUNT)
2000

Find the number of rows in a company

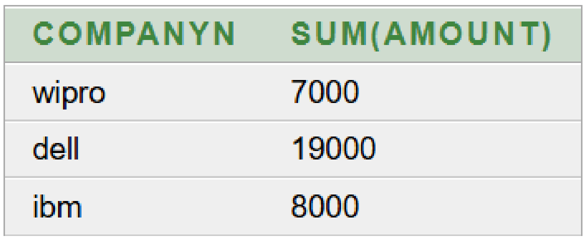
Select Count(*) from company;



COUNT(*)
5

Find the sum of amount of each company.

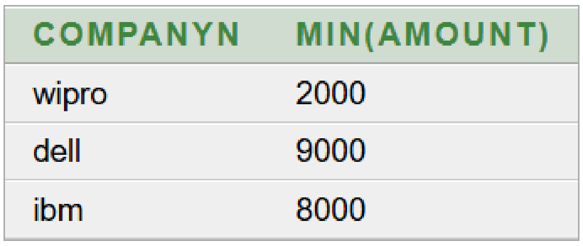
select companyn,sum(amount) from company group by companyn;



COMPANYN	SUM(AMOUNT)
wipro	7000
dell	19000
ibm	8000

Find the minimum amount of each company.

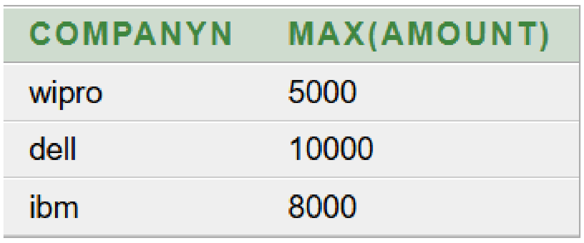
select companyn,min(amount) from company group by companyn;



COMPANYN	MIN(AMOUNT)
wipro	2000
dell	9000
ibm	8000

Find the maximum amount of each company.

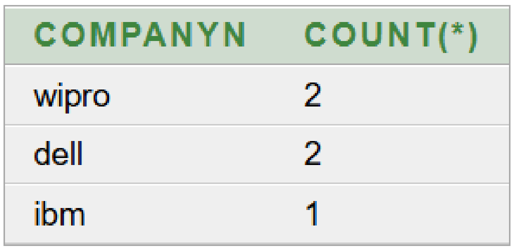
select companyn,max(amount) from company group by companyn;



COMPANYN	MAX(AMOUNT)
wipro	5000
dell	10000
ibm	8000

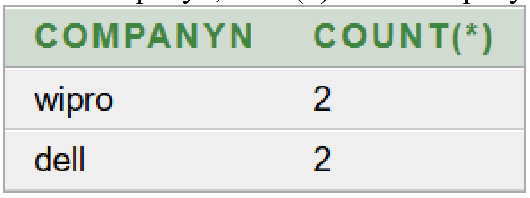
Find the count of all the rows grouped by each company name.

select companyn,count(*) from company group by companyn;



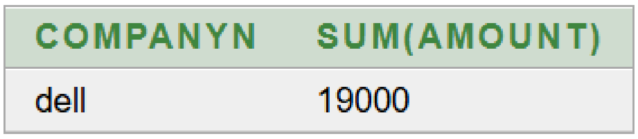
COMPANYN	COUNT(*)
wipro	2
dell	2
ibm	1

Find the count of all the rows grouped by each company name & having count greater than 1.
select companyn,count(*) from company group by companyn having count(*)>1;



COMPANYN	COUNT(*)
wipro	2
dell	2

Find the sum of amount of each company and having sum of amount greater than 10000.
select companyn,sum(amount) from company group by companyn having sum(amount)>10000;



COMPANYN	SUM(AMOUNT)
dell	19000

AIM: 3.1 Perform Join Operations-Natural Join, Equi-Join, Outer Join, Left Outer Join, Right Outer Join, Inner Join and assess the impact of query plans on the performance of join heavy queries.

Description:

JOIN Keyword is used in SQL queries for joining two or more tables.

Types of Joins

Inner Join

The INNER JOIN keyword selects records that have matching values in both tables.

Syntax: select * from tablename1 inner join tablename2 on condition

Outer Join

Outer Join is based on both matched and unmatched data.
It is divided into

1) Left outer join

   The left outer join returns a resultset table with the matched data from the two tables and then the remaining rows of the left table and null from the right table's columns.
   Syntax:
   SELECT column-name-list FROM
   table-name1 LEFT OUTER JOIN table-name2
   ON table-name1.column-name = table-name2.column-name;

2) Right outer join

   The right outer join returns a resultset table with the matched data from the two tables being joined, then the remaining rows of the right table and null for the remaining left table's columns.
   Syntax:

   SELECT column-name-list FROM
   table-name1 RIGHT OUTER JOIN table-name2
   ON table-name1.column-name = table-name2.column-name;

Natural join

It is based on column having same name and same datatype present in both the tables to be joined.
Syntax:
SELECT * FROM table-name1 NATURAL JOIN table-name2;

Cross join

It will return a table which consists of records which combines each row from the first table -with each row of the second table.
Syntax:
select * from tablename1,tablename2

Self Join

A **self join** is a **join** in which a table is joined with itself.

EQUI Join

An Equi Join in SQL is a type of join that combines rows from two or more tables based on a common column or set of columns, using only the equality operator (=) to compare the values in those columns.
Syntax
SELECT column-name-list FROM table1, table2....
WHERE table1.column_name =table2.column_name;

Source Tables

select *from tb1;

RNO	NAME	MARKS
503	Suma	40
504	Raju	70
505	Ramu	45
501	Abhi	50
502	Ravi	60

select * from tb2;

RNO	FEE
501	15000
502	5000
503	10000
504	25000

Inner Join

select *from tb1 inner join tb2 on tb1.rno=tb2.rno;

Or use the below query also

select *from tb1 join tb2 on tb1.rno=tb2.rno;

RNO	NAME	MARKS	RNO	FEE
503	Suma	40	503	10000
504	Raju	70	504	25000
501	Abhi	50	501	15000
502	Ravi	60	502	5000

Left Outer Join

select * from tb1 left outer join tb2 on tb1.rno=tb2.rno;

RNO	NAME	MARKS	RNO	FEE
501	Abhi	50	501	15000
502	Ravi	60	502	5000
503	Suma	40	503	10000
504	Raju	70	504	25000
505	Ramu	45	-	-

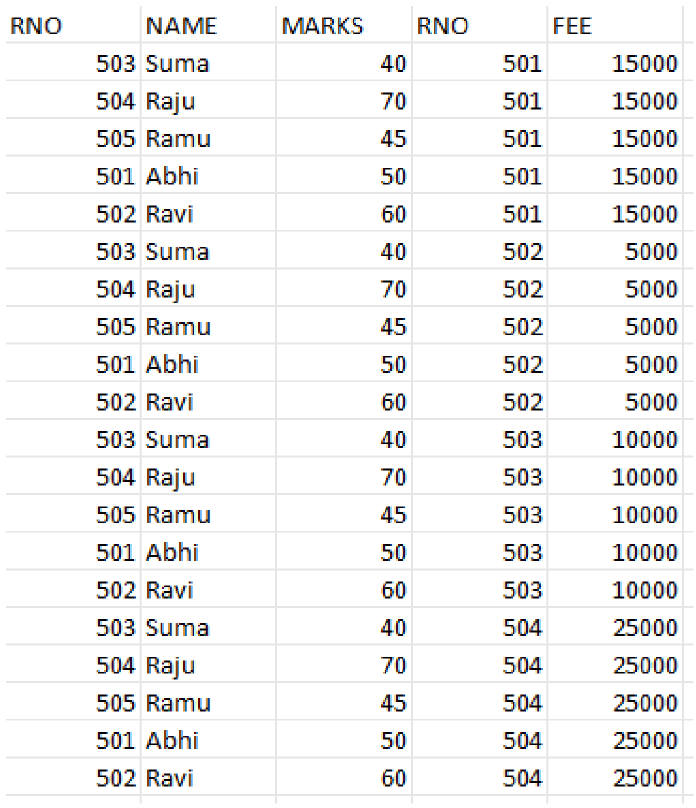
Right outer join
select * from tb1 right outer join tb2 on tb1.rno=tb2.rno;

RNO	NAME	MARKS	RNO	FEE
503	Suma	40	503	10000
504	Raju	70	504	25000
501	Abhi	50	501	15000
502	Ravi	60	502	5000

Natural join
select *from tb1 natural join tb2;

RNO	NAME	MARKS	FEE
503	Suma	40	10000
504	Raju	70	25000
501	Abhi	50	15000
502	Ravi	60	5000

Cross join
select *from tb1 cross join tb2;
or use the below query
select * from tb1,tb2;



RNO	NAME	MARKS	RNO	FEE
503	Suma	40	501	15000
504	Raju	70	501	15000
505	Ramu	45	501	15000
501	Abhi	50	501	15000
502	Ravi	60	501	15000
503	Suma	40	502	5000
504	Raju	70	502	5000
505	Ramu	45	502	5000
501	Abhi	50	502	5000
502	Ravi	60	502	5000
503	Suma	40	503	10000
504	Raju	70	503	10000
505	Ramu	45	503	10000
501	Abhi	50	503	10000
502	Ravi	60	503	10000
503	Suma	40	504	25000
504	Raju	70	504	25000
505	Ramu	45	504	25000
501	Abhi	50	504	25000
502	Ravi	60	504	25000

Self Join:

select t1.rno, t2.name from tb1 t1,tb1 t2 where t1.rno=t2.rno;

RNO	NAME
503	Suma
504	Raju
505	Ramu
501	Abhi
502	Ravi

EQUI Join:

select * from tb1,tb2 where tb1.rno=tb2.rno;

RNO	NAME	MARKS	RNO	FEE
503	Suma	40	503	10000
504	Raju	70	504	25000
501	Abhi	50	501	15000
502	Ravi	60	502	5000

AIM: 3.2 Perform Set Operations-Union, Intersection, Set Difference

Description:

Set Operators

1) UNION is used to combine the results of two or more `SELECT` statements. It will -eliminate duplicate rows from its resultset.

   Syntax:-

   ```
   SELECT column_name FROM table1
   UNION
   SELECT column_name FROM table2;
   ```

2) UNION ALL This is similar to Union. But it also shows the duplicate rows.

   Syntax:-

   ```
   SELECT column_name FROM table1
   UNION ALL
   SELECT column_name FROM table2;
   ```

3) Intersect operation is used to combine two `SELECT` statements, but it only retuns the records which are common from both `SELECT` statements.

   Syntax:-

   ```
   SELECT column_name FROM table1
   INTERSECT
   SELECT column_name FROM table2;
   ```

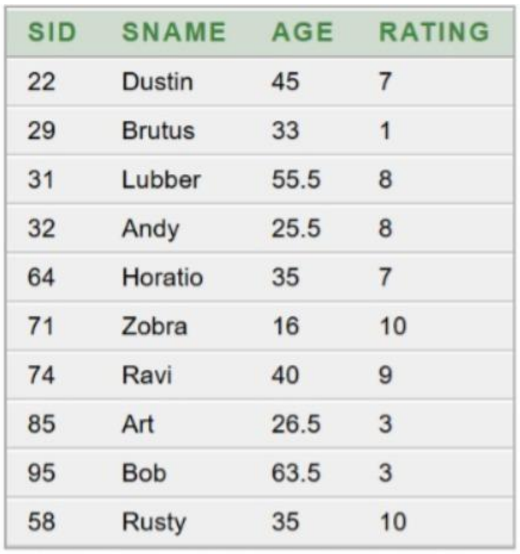
4) The Minus operation combines results of two `SELECT` statements and return only those in the final result, which belongs to the first set of the result.

   Syntax:-

   ```
   SELECT column_name FROM table1
   MINUS
   SELECT column_name FROM table2;
   ```

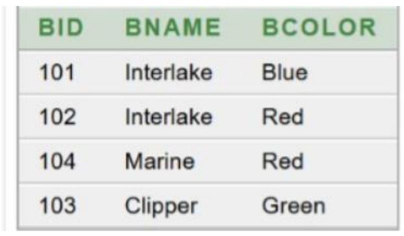
Source Tables

Sailors Table



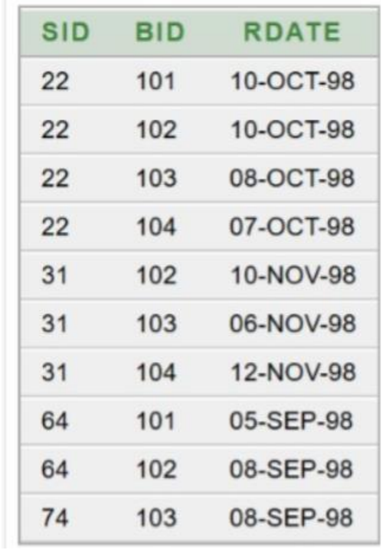
SID	SNAME	AGE	RATING
22	Dustin	45	7
29	Brutus	33	1
31	Lubber	55.5	8
32	Andy	25.5	8
64	Horatio	35	7
71	Zobra	16	10
74	Ravi	40	9
85	Art	26.5	3
95	Bob	63.5	3
58	Rusty	35	10

Boats Table



BID	BNAME	BCOLOR
101	Interlake	Blue
102	Interlake	Red
104	Marine	Red
103	Clipper	Green

Reserves Table



SID	BID	RDATE
22	101	10-OCT-98
22	102	10-OCT-98
22	103	08-OCT-98
22	104	07-OCT-98
31	102	10-NOV-98
31	103	06-NOV-98
31	104	12-NOV-98
64	101	05-SEP-98
64	102	08-SEP-98
74	103	08-SEP-98

Find the names of sailors who have reserved a red or a green boat

```
select s.sname from sailors s, reserves r,boats b
where s.sid=r.sid and b.bid=r.bid and b.bcolor='Red'
UNION
select s1.sname from sailors s1,reserves r1,boats b1
where s1.sid=r1.sid and r1.bid=b1.bid and b1.bcolor='Green';
```

SNAME
Dustin
Horatio
Lubber
Ravi

Find the names of sailors who have reserved a red or a green boat

```
select s.sname from sailors s, reserves r,boats b
where s.sid=r.sid and b.bid=r.bid and b.bcolor='Red'
UNION all
select s1.sname from sailors s1,reserves r1,boats b1
where s1.sid=r1.sid and r1.bid=b1.bid and b1.bcolor='Green';
```

SNAME
Dustin
Lubber
Horatio
Dustin
Lubber
Dustin
Lubber
Ravi

Find the names of sailors who have reserved both a red and green boat.

```
select s.sname from sailors s, reserves r,boats b
where s.sid=r.sid and b.bid=r.bid and b.bcolor='Red'
INTERSECT
select s1.sname from sailors s1,reserves r1,boats b1
where s1.sid=r1.sid and r1.bid=b1.bid and b1.bcolor='Green';
```

SNAME
Dustin
Lubber

Find the names of sailors who have reserved a red boat but not a green boat

```
select s.sname from sailors s, reserves r,boats b
where s.sid=r.sid and b.bid=r.bid and b.bcolor='Red'
MINUS
select s1.sname from sailors s1,reserves r1,boats b1
where s1.sid=r1.sid and r1.bid=b1.bid and b1.bcolor='Green';
```

SNAME
Horatio

AIM:3.3 Implementation of Correlated sub-queries and Nested queries

Description:

Nested Queries

A query within another SQL query and embedded within the WHERE clause.In Nested Query, Inner query runs first, and only once. Outer query is executed with result from Inner query.

1) IN and NOT IN

It tests whether a value is in a given set of elements

Syntax:-

```
Select column_names from table_name
Where column_name IN/NOT IN (Select column_name from table_name
                                Where condition);
```

2) ALL and ANY

It is used to compare a value to a list. It is preceded by comparison operator and followed by a list.

Syntax:-

```
Select column_name from table_name
Where column_name comparison operator ALL/ANY(subquery);
```

Correlated Sub query

In Correlated sub query**,** a query is nested inside another query and an inner query uses values from the outer query

Syntax:-

SELECT *column_names* FROM *table_name*
WHERE EXISTS/NOT EXISTS
(SELECT *column_name* FROM *table_name* WHERE *condition*);

Find the names of sailors who have reserved boat no 103.

```
select s.sname from sailors s
where s.sid IN( select r.sid from reserves r
                        where r.bid=103);
```

SNAME
Dustin
Lubber
Ravi

Find the names of sailors who have not reserved boat no 103.

```
select s.sname from sailors s
where s.sid NOT IN( select r.sid from reserves r
                              where r.bid=103);
```

SNAME
Brutus
Andy
Horatio
Zobra
Art
Bob
Rusty

Find the sailor id with the highest rating

```
select s.sid from sailors s where s.rating>=all(select s1.rating from sailors s1);
```

SID
71
58

Find the sailor id whose rating is better than some sailor called andy

```
select s.sid from sailors s
where s.rating>ANY(select s1.rating from sailors s1 where s1.sname='Andy');
```

SID
71
58
74

Correlated Sub query

Find the names of sailors who have reserved boat no 103

select s.sname from sailors s where EXISTS(select * from reserves r where s.sid=r.sid and r.bid=103);

SNAME
Dustin
Lubber
Ravi

Find the names of sailors who have not reserved boat no 103

select s.sname from sailors s where NOT EXISTS(select * from reserves r where s.sid=r.sid and r.bid=103);

SNAME
Brutus
Andy
Horatio
Zobra
Art
Bob
Rusty

AIM: 3.4 Creating and Querying views and Materialized views.

Description:

View

A view is a logical table based on the result set of an SQL Statement. A view contains rows and columns, just like a table. The fields in a view are fields from one or more base tables in the database. We can apply all DDL and DML statements on views.

Syntax:

Create or replace force/noforce view viewname as

Select column_list from table_list

Where condition with read only|check option

Dropping a view syntax:

Drop view viewname;

Source Table

ROLLNO	NAME	MARKS
501	jyothi	90
502	sai	95
504	yamuna	70
505	padma	60
503	ravi	80

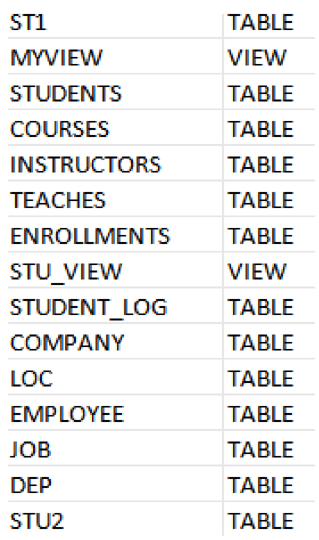
Creating View

create view myview as select rollno,name from st1;

view created

Display Views and Tables in your login

select * from tab;



```
ST1          TABLE
MYVIEW       VIEW
STUDENTS     TABLE
COURSES      TABLE
INSTRUCTORS  TABLE
TEACHES      TABLE
ENROLLMENTS  TABLE
STU_VIEW     VIEW
STUDENT_LOG  TABLE
COMPANY      TABLE
LOC          TABLE
EMPLOYEE     TABLE
JOB          TABLE
DEP          TABLE
STU2         TABLE
```

Inserting a row in view

insert into myview values(506,'prathisha');

1 row(s) inserted

Display view

select * from myview;

ROLLNO	NAME
501	jyothi
502	sai
506	prathisha
504	yamuna
505	padma
503	ravi

Displaying Table

select * from st1;

ROLLNO	NAME	MARKS
501	jyothi	90
502	sai	95
506	prathisha	-
504	yamuna	70
505	padma	60
503	ravi	80

Deleting a row in a view

delete from myview where rollno=506;

1 row(s) deleted

Display view

select * from myview;

ROLLNO	NAME
501	jyothi
502	sai
504	yamuna
505	padma
503	ravi

Displaying Table

select * from st1;

ROLLNO	NAME	MARKS
501	jyothi	90
502	sai	95
504	yamuna	70
505	padma	60
503	ravi	80

Change the Structure of the View

create or replace view myview as select * from st1;

view created.

Creating View when base table doesn't exist

create or replace force view abc as select * from dummy_table;

Warning: View created with compilation errors.

Creating Read only view

create view myview1 as select * from st1 with read only;

view created

Inserting Data in Read only view

insert into myview1 values(503,'prathisha',80);

ORA-01733: virtual column not allowed here

update myview1 set name='suma' where rollno=505;

ORA-01733: virtual column not allowed here

Displaying view

select * from myview1;

ROLLNO	NAME	MARKS
501	jyothi	90
502	sai	95
504	yamuna	70
505	padma	60
503	ravi	80

Creating View with check option

create view myview2 as select * from st1 where marks<101 with check option;

view created.

Inserting a row into view

```
insert into myview2 values(504,'siri',101);
```

```
ORA-01402: view WITH CHECK OPTION where-clause violation
```

Dropping view

```
Drop View myview1;
```

```
View dropped
```

AIM: 4.1 Implement SQL queries on a normalized database schema based on the provided schema

Description:

Normalization is a process of organizing data in a database to reduce redundancy and improve data integrity.

In this experiment, we will use the **University Database Schema** consisting of the following tables:

- **Students (StudentID, StudentName, Major)**
- **Courses (CourseID, CourseName, Credits)**
- **Enrollments (StudentID, CourseID, EnrollmentDate)**
- **Instructors (InstructorID, InstructorName, Phone)**
- **Course_Instructors (CourseID, InstructorID)**

Creating Tables:

```
CREATE TABLE Students (
   StudentID VARCHAR2(10),
   StudentName VARCHAR2(30),
   Major VARCHAR2(30)
);

CREATE TABLE Courses (
   CourseID VARCHAR2(10),
   CourseName VARCHAR2(30),
   Credits NUMBER
);

CREATE TABLE Enrollments (
   StudentID VARCHAR2(10),
   CourseID VARCHAR2(10),
   EnrollmentDate DATE
);

CREATE TABLE Instructors (
   InstructorID VARCHAR2(10),
   InstructorName VARCHAR2(30),
   Phone NUMBER
);

CREATE TABLE Course_Instructors (
   CourseID VARCHAR2(10),
   InstructorID VARCHAR2(10)
);
```

Inserting Sample Data:

```
INSERT INTO Students VALUES ('S01', 'Kiran', 'CSE');
INSERT INTO Students VALUES ('S02', 'Bala', 'ECE');
INSERT INTO Students VALUES ('S03', 'Ravi', 'EEE');

INSERT INTO Courses VALUES ('C01', 'DBMS', 4);
INSERT INTO Courses VALUES ('C02', 'OS', 3);
INSERT INTO Courses VALUES ('C03', 'Networks', 3);

INSERT INTO Enrollments VALUES ('S01', 'C01', '12-SEP-2024');
INSERT INTO Enrollments VALUES ('S02', 'C02', '15-SEP-2024');
INSERT INTO Enrollments VALUES ('S03', 'C03', '20-SEP-2024');

INSERT INTO Instructors VALUES ('I01', 'Suma', 9876543210);
INSERT INTO Instructors VALUES ('I02', 'Raju', 9876501234);

INSERT INTO Course_Instructors VALUES ('C01', 'I01');
INSERT INTO Course_Instructors VALUES ('C02', 'I02');
```

Example Queries:

Display all students and their majors

```
SELECT * FROM Students;
```

STUDENTID	STUDENT NAME	MAJOR
S01	Kiran	CSE
S02	Bala	ECE
S03	Ravi	EEE

Display all courses with credits

```
SELECT CourseName, Credits FROM Courses;
```

ROLLNO	NAME
DBMS	4
OS	3
Networks	3

Display students who enrolled in DBMS

```
SELECT s.StudentName, c.CourseName
FROM Students s, Enrollments e, Courses c
WHERE s.StudentID = e.StudentID AND e.CourseID = c.CourseID AND c.CourseName = 'DBMS';
```

STUDENT NAME	COURS ENAME
Kiran	DBMS

Display instructor names along with the courses they teach

```
SELECT i.InstructorName, c.CourseName
FROM Instructors i, Courses c, Course_Instructors ci
WHERE i.InstructorID = ci.InstructorID AND ci.CourseID = c.CourseID;
```

INSTRUCTORNAME	COURSENAME
Suma	DBMS
Raju	OS

Count number of students enrolled in each course

```
SELECT c.CourseName, COUNT(e.StudentID) AS Total_Students
FROM Courses c, Enrollments e
WHERE c.CourseID = e.CourseID
GROUP BY c.CourseName;
```

COURSENAME	TOTAL_STUDENTS
DBMS	1
OS	1
Networks	1

AIM: 4.2 (A) Implementation of Data Control Language commands — GRANT and REVOKE

Description:

DCL commands control access to data in the database.

- **GRANT:** Allows users to access and manipulate database objects.
- **REVOKE:** Removes previously granted privileges.

Syntax:

```
GRANT privilege_name ON object_name TO user_name;
REVOKE privilege_name ON object_name FROM user_name;
```

Example:

```
CREATE TABLE student_login (
   userid VARCHAR2(10),
   password VARCHAR2(20)
);

GRANT SELECT, INSERT ON student_login TO user1;
REVOKE INSERT ON student_login FROM user1;
```

Explanation:

- The GRANT statement gives user1 permission to **select** and **insert** records into the table.
- The REVOKE statement removes the **insert** permission from user1.

AIM: 4.2 (B) Implementation of Transaction Control Language commands — COMMIT, SAVEPOINT, and ROLLBACK

Description:

TCL commands are used to manage transactions in the database.

- **COMMIT:** Saves all the changes made in the current transaction.
- **ROLLBACK:** Undoes the changes of the current transaction.
- **SAVEPOINT:** Creates a temporary point in a transaction for partial rollback.

Syntax:

```
COMMIT;
ROLLBACK;
SAVEPOINT savepoint_name;
ROLLBACK TO savepoint_name;
```

Example:

```
CREATE TABLE accounts (
   accno NUMBER,
   name VARCHAR2(20),
   balance NUMBER
);

INSERT INTO accounts VALUES (101, 'Ravi', 2000);
SAVEPOINT A;

INSERT INTO accounts VALUES (102, 'Suma', 3000);
SAVEPOINT B;

UPDATE accounts SET balance = balance + 500 WHERE accno = 101;

ROLLBACK TO B;
COMMIT;
```

ACCNO	NAME	BALANCE
101	Ravi	2000
102	Suma	3000

Explanation:

1. **SAVEPOINT A** and **SAVEPOINT B** mark transaction stages.
2. **ROLLBACK TO B** cancels the update done after B but retains all changes before it.
3. **COMMIT** makes the remaining changes permanent.

AIM: 5.1 Create a Primary and Secondary Index on a Column

Description:

An **index** improves the speed of data retrieval operations in a database.
There are two main types:

- **Primary Index** – created automatically when a primary key is defined.
- **Secondary Index** – created manually on non-primary key columns to improve search performance.

Indexes work like book indexes — instead of scanning all pages (rows), the database can jump directly to the desired data.

Syntax:

```
-- Primary Index (Automatically created using PRIMARY KEY)
CREATE TABLE table_name (
  column1 datatype PRIMARY KEY,
  column2 datatype,
  ...
);

-- Secondary Index (Manually created)
CREATE INDEX index_name
ON table_name (column_name);
```

Example:

```
CREATE TABLE Students (
  StudentID VARCHAR2(10) PRIMARY KEY,
  StudentName VARCHAR2(30),
  Major VARCHAR2(20)
);

-- Creating Secondary Index on StudentName
CREATE INDEX idx_studentname
ON Students (StudentName);
```

Output:

```
Table created.
Index created.
```

AIM: 5.2 Retrieve Data Using an Index

Description:

When a query uses an indexed column in the WHERE clause, the database engine uses the index to quickly locate the matching rows, improving performance.

Example:

```
SELECT * FROM Students WHERE StudentName = 'Kiran';
```

Since the StudentName column has a secondary index (idx_studentname), this query will retrieve the record faster than a full table scan.

StudentID	StudentName	Major
S01	Kiran	CSE

AIM: 5.3 Insert Data and Update Indexes

Description:

When new records are inserted, the indexes are automatically updated by the database. This ensures that all future retrievals remain efficient.

Example:

```
INSERT INTO Students VALUES ('S02', 'Ravi', 'ECE');
INSERT INTO Students VALUES ('S03', 'Bala', 'EEE');
```

The database automatically updates:

- The **Primary Index** for StudentID
- The **Secondary Index** for StudentName

To Verify:

```
SELECT * FROM Students;
```

STUDENTID	STUDENT NAME	MAJOR
S01	Kiran	CSE
S02	Ravi	ECE
S03	Bala	EEE

AIM: 5.4 Delete Data and Observe Impact on Indexes

Description:

When a row is deleted, the corresponding entries in all indexes are also automatically removed by the database.
This maintains data consistency and prevents invalid index references.

Example:

```
DELETE FROM Students WHERE StudentID = 'S02';
```

The index entries for 'S02' and 'Ravi' are automatically deleted from the primary and secondary indexes.

To Verify:

```
SELECT * FROM Students;
```

STUDENTID	STUDENT NAME	MAJOR
S01	Kiran	CSE
S03	Bala	ECE

Conclusion:

- Primary indexes are automatically created and maintained by the DBMS.
- Secondary indexes are created manually for faster access on non-key columns.
- Both indexes automatically update when data is inserted, deleted, or modified.