Sri SAI RAM ENGINEERING COLLEGE

An Autonomous Institution | Affiliated to Anna University & Approved by AICTE, New Delhi

Accredited by NBA and NAAC "A+" | BIS/EOMS ISO 21001 : 2018 and BVQI 9001 : 2015 Certified and NIRF ranked institution

Sai Leo Nagar, West Tambaram, Chennai - 600 044. www.sairam.edu.in

PROSPERITY THROUGH TECHNOLOGY

LAB MANUAL

24AMPT301 – DATABASE MANAGEMENT SYSTEMS LABORATORY WITH THEORY

III Semester CSE (AI&ML)

Academic year: 2025 - 2026

DEPARTMENT OF CSE (ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

PREFACE

“PRACTICE LEADS TO PERFECTION”

Practical work encourages and familiarizes students with the latest tools that will be required to become experts. Taking this into consideration, this manual is compiled as a preparatory note for the Database Management Systems Laboratory programs. Sufficient details have been included to impart self- learning.

This manual is intended for the III semester CSE (AI&ML) students under Autonomous Regulations 2020. Introductory information and procedure to perform is detailed in this manual.

It is expected that this will help the students develop a broad understanding and learn quick adaptations to real-time problems.

INSTITUTE VISION

To emerge as a “Centre of excellence” offering Technical Education and Research opportunities of very high standards to students, develop the total personality of the individual and instill high levels of disciple and strive to set global standards, making our students technologically superior and ethically stronger, who in turn shall contribute to the advancement of society and humankind.

INSTITUTION MISSION

We dedicate and commit ourselves to achieve, sustain and foster unmatched excellence in Technical Education. To this end, we will pursue continuous development of infra-structure and enhance state-of-art equipment to provide our students a technologically up-to date and intellectually inspiring environment of learning, research, creativity, innovation and professional activity and inculcate in them ethical and moral values.

INSTITUTE POLICY

We at Sri Sai Ram Engineering College are committed to build a better Nation through Quality Education with team spirit. Our students are enabled to excel in all values of Life and become Good Citizens. We continually improve the System, Infrastructure and Service to satisfy the Students, Parents, Industry and Society.

DEPARTMENT VISION

To emerge as a "Centre of Excellence" in the field of Artificial Intelligence and Machine Learning by providing required skill sets, domain expertise and interactive industry interface for students and shape them to be a socially conscious and responsible citizen.

DEPARTMENT MISSION

Computer Science and Engineering (Artificial Intelligence and Machine Learning), Sri Sairam Engineering College is committed to

M1: Nurture students with a sound understanding of fundamentals, theory and practice of Artificial Intelligence and Machine Learning.

M2: Develop students with the required skill sets and enable them to take up assignments in the field of AI & ML

M3: Facilitate Industry Academia interface to update the recent trends in AI &ML

M4: Create an appropriate environment to bring out the latent talents, creativity and innovation among students to contribute to the society

Program Educational Objectives (PEOs)

To prepare the graduates to:

1. Graduates imbibe fundamental knowledge in Artificial Intelligence, Programming, Mathematical modelling and Machine Learning.
2. Graduates will be trained to gain domain expertise by applying the theory basics into practical situation through simulation and modelling techniques.
3. Graduates will enhance the capability through skill development and make them industry ready by inculcating leadership and multitasking abilities

4. Graduates will apply the gained knowledge of AI & ML in Research & Development, Innovation and contribute to the society in making things simpler.

Program Outcomes (PO's)

- PO1: Engineering Knowledge: Apply knowledge of mathematics, natural science, computing, engineering fundamentals and an engineering specialization as specified in WK1 to WK4 respectively to develop to the solution of complex engineering problems.
- PO2: Problem Analysis: Identify, formulate, review research literature and analyze complex engineering problems reaching substantiated conclusions with consideration for sustainable development. (WK1 to WK4)
- PO3: Design/Development of Solutions: Design creative solutions for complex engineering problems and design/develop systems/components/processes to meet identified needs with consideration for the public health and safety, whole-life cost, net zero carbon, culture, society and environment as required. (WK5)
- PO4: Conduct Investigations of Complex Problems: Conduct investigations of complex engineering problems using research-based knowledge including design of experiments, modelling, analysis & interpretation of data to provide valid conclusions. (WK8).
- PO5: Engineering Tool Usage: Create, select and apply appropriate techniques, resources and modern engineering & IT tools, including prediction and modelling recognizing their limitations to solve complex engineering problems. (WK2 and WK6)
- PO6: The Engineer and The World: Analyze and evaluate societal and environmental aspects while solving complex engineering problems for its impact on sustainability with reference to economy, health, safety, legal framework, culture and environment. (WK1, WK5, and WK7).

- PO7: Ethics: Apply ethical principles and commit to professional ethics, human values, diversity and inclusion; adhere to national & international laws. (WK9)
- PO8: Individual and Collaborative Team work: Function effectively as an individual, and as a member or leader in diverse/multi-disciplinary teams.
- PO9: Communication: Communicate effectively and inclusively within the engineering community and society at large, such as being able to comprehend and write effective reports and design documentation, make effective presentations considering cultural, language, and learning differences
- PO10: Project Management and Finance: Apply knowledge and understanding of engineering management principles and economic decision-making and apply these to one's own work, as a member and leader in a team, and to manage projects and in multidisciplinary environments.
- PO11: Life-Long Learning: Recognize the need for, and have the preparation and ability for i) independent and life-long learning ii) adaptability to new and emerging technologies and iii) critical thinking in the broadest context of technological change. (WK8)

Program Specific Outcomes (PSOs)

Computer Science and Engineering (Artificial Intelligence and Machine Learning), graduates will be able to:

- The graduates will be in a position to design, develop, test and deploy appropriate mathematical and programming algorithms required for practical applications

- The graduates will have the required skills and domain expertise to provide solutions in the field of Artificial Intelligence and Machine Learning for the Industry and society at large.

COURSE OUTCOMES

Upon completion of the course, the students will be able to

CO1	Discuss the concepts of database to apply the Relational, ER model for design and SQL for implementation of the database. (K3)
CO2	Recognize and identify the use of normalization and functional dependencies to refine the database system. (K3)
CO3	Demonstrate various SQL queries for Transaction Processing & Locking using the concept of Concurrency Control. (K3)
CO4	Apply the query processing techniques for the optimization of SQL queries.(K3)
CO5	Implement indexing and hashing techniques for the organization of database records. (K3)
CO6	Illustrate how the advanced databases differ from the traditional databases. (K4)

24AMPT301 – DATABASE MANAGEMENT SYSTEMS LABORATORY WITH THEORY

Syllabus

OBJECTIVES:

- To design a database using ER diagrams, convert them to Relational Databases and to write SQL Queries.
- To learn SQL and normalization techniques for efficient database design.
- To understand the fundamental concepts of Transaction Processing, Concurrency Control techniques, and Recovery procedures.
- To understand file organization, indexing, and hashing techniques for efficient data storage and retrieval
- To explore distributed and object-oriented databases, focusing on architecture, data storage, and advanced query languages
- To study query processing, optimization techniques, and explore XML databases, including XML schema and XQuery.

LIST OF EXPERIMENTS:

- Data Definition Language - Create, Describe, Alter, Truncate, Drop, Data Manipulation Language - Insert, Select, Update, Delete, Transaction Control Language - Commit, Rollback, and Savepoint
- Write simple SQL queries to perform the following Nested Queries and Joins (Inner Join, Left Join, Right Join, Full Join, Cross Join)
- Write a SQL program for adding two numbers and displaying the sum.
- Write a SQLprogram to print a series of "n" numbers using for loop.
- Write a SQL program to print a series of "n" numbers using while loop.
- Write a SQL program for debiting an amount from an entry in a table
- Write a SQL program to determine the sum of a series of n numbers.
- Write a SQL program to determine the factorial of the given number.
- Write a SQL program for reversing a string using the for loop.
- Write a SQL program to implement Fibonacci series.
- Write a SQL program for displaying a particular attribute from a table.
- Mini Project (Banking System) Connect Oracle Database from Visual Basic for Banking System.

INDEX

Ex.No:1a **DATA DEFINITION LANGUAGE**

Date:

AIM:

To implement all Data Definition Language Commands.

DDL commands are:

(1) Create
(2) Describe
(3)Alter
(4)Add
(5)Modify
(6)Truncate
(7)Drop

(1) **Create**: **Syntax:**

This DDL command is used to create a table.create table <table name>(fieldname 1 datatype(size),fieldname 2datatype(size)….fieldname n datatype(size));

(2) **Describe:**

This DDL command is used to describe the table structure (displays the fields and their types).

Syntax:

desc <table name>;

(3) **Alter**:

This DDL command is used to modify the already existing table, but we cannot change or rename the table.

(i) **Add:**

This DDL command is used to add the column to an existing table.

Syntax:

alter table <table name> add (fieldname 1 datatype(size),fieldname2 datatype(size)…);

(ii) **Modify:**

It is used to modifycolumn in already existing table.

Syntax:

alter table <table name> modify (column datatype(size)…);

(4) **Truncate**:

This DDL command is used to delete data in the table.

Syntax:

truncate table <tablename>**;**

(5) **Drop**:

This DDL command is used to drop a table.

Syntax:

drop table <tablename>;

OUTPUT:-

i) Create the employee table with the following columns (EMP ID, EMP NAME, EMP AGE)

<u>*CREATE</u>

```
SQL> create table emp1(ename varchar(7),eno number(5),addr varchar(15),pho numb er(7),dept number(5));
Table created.
```

ii)Use a command to describe the structure of the employee's table.

<u>*DESCRIBE</u>

```
SQL> desc empl1;
 Name                          Type
                ______________________ __________
 ENAME                        VARCHAR2(7)
 ENO                          NUMBER(5)
 ADDR                         VARCHAR2(15)
 PHO                          NUMBER(7)
 DEPT                         NUMBER(5)
```

iii)Write an SQL statement to alter the employees table to add a new column salary.

<u>*ALTER (add entry)</u>

```
SQL> alter table emp1 add(salary number(8,2)); Table altered.

SQL> desc emp1;
 Name                          Type
                ______________________ ___________________
 ENAME                          VARCHAR2(7)
 ENO                            NUMBER(5)
 ADDR                           VARCHAR2(15)
 PHO                            NUMBER(7)
 DEPT                           NUMBER(5)
 SALARY                         NUMBER(8,2)
```

<u>*ALTER (modify entry)</u>

```
SQL> alter table emp1 modify(ename varchar(8)); Table altered.

SQL> desc emp1;
 Name                          Type
                ______________________ ___________________
 ENAME                          VARCHAR2(8)
 ENO                            NUMBER(5)
 ADDR                           VARCHAR2(15)
 PHO                            NUMBER(7)
 DEPT                           NUMBER(5)
 SALARY                         NUMBER(8,2)
```

iv)Use a command to truncate the employees table, removing all rows but keeping the structure intact.

<u>*TRUNCATE</u>

```
SQL> truncate table emp1; Table truncated.
```

v)write a command to drop the employees table completely from the database.

*DROP

```
SQL> drop table emp1; Table dropped. SQL> desc emp1;
ERROR:
ORA-04043: object emp1 does not exist
```

RESULT:

Thus the Data Definition Language Commands operation has been executed successfully.

Ex.No:1b **DATA MANIPULATION LANGUAGE**
Date:

AIM:

To implement all DML commands (1)Insert (2) Select (3)Update

(4)Delete

(1) **Insert**:

This DML command is used to insert the details into the table.

Syntax:

insert into tablename values („&field1", „&field2",…);

(2) **Select**:

This command is used to show the details present in a table.

Syntax:

select * from <table name >;

(3) **Update**:

This command is used to change a value of field in a row.

Syntax:

update <tablename> set < field="new value"> where <field="oldvalue">;

(4) **Delete**:

This command is used to delete a row in table.

Syntax:

delete from <tablename> where< fieldname="value">;

OUTPUT:-

i)Write an SQL command to insert a new student with (ID, NAME and GRADE)

```
SQL> create table emp2(empno number(5),empname varchar(7),age number(3),job varchar(20) ,deptno
number(5),salary number(8,2));

Table created.

SQL> desc emp2;
Name                          Type
     ____________________     ____________________
EMPNO                         NUMBER(5)
EMPNAME                       VARCHAR2(7)
AGE                           NUMBER(3)
JOB                           VARCHAR2(8)
DEPTNO                        NUMBER(5)
SALARY                        NUMBER(8,2)
```

*INSERT (single entry)

```
SQL> insert into emp2 values(100,'John',36,'Manager',455,17500); 1 row created.

SQL> select * from emp2;
EMPNO EMPNAME     AGE JOB DEPTNO  SALARY
      _______ ___________ _____ ______ ________ _______
   100      John            36     Manager      455       17500
```

*INSERT (multipleentries)

```
SQL> insert into emp2 values(&empno,&empname,&age,&job,&deptno,&salary); Enter value for empno:
101
Enter value for empname: 'Smith' Enter value for age: 29
Enter value for job: 'Clerk' Enter value for deptno:
43 Enter value for salary: 1200 1 row created.

SQL> /
Enter value for empno: 102 Enter value for
empname: 'A' Enter value for age: 32
Enter value for job: 'Assistant Manager' Enter value for deptno: 43
Enter value for salary: 13500 1 row created.

SQL> /
Enter value for empno: 103 Enter value for
empname: 'B' Enter value for age: 37
Enter value for job: 'General Manager' Enter value for deptno: 355
Enter value for salary: 19000 1 row created.

SQL> /
Enter value for empno: 104 Enter value for
empname: 'E' Enter value for age: 28
Enter value for job: 'Clerk' Enter value for deptno:
35 Enter value for salary: 1500 1 row created.

SQL> /
Enter value for empno: 105 Enter value for
empname: 'F' Enter value for age: 38
Enter value for job: 'General Manager' Enter value for deptno: 40
Enter value for salary: 19000 1 row created.



SQL> select * from emp2;
  EMPNO  EMPNAME      AGE       JOB                 DEPTNO  SALARY
 --------------------------------------------------------------------------
      100    John                 36      Manager                   455      17500
      101   Smith                 29      Clerk                      43      1200
      102   A                     32      Assistant Manager          43      13500
      103    B                    37      General Manager           355      19000
      104    E                    28      Clerk                       35      1500
      105    F                    38      General Manager             40     19000
6 rows selected.
```

ii)Write a select query to retrieve all records from the student's table.

*SELECT (specific attributes)

```
SQL> select empno,salary from emp2;
```

```
 EMPNO    SALARY
-----------------------
       100     17500
       101      1200
       102     13500
       103     19000
       104      1500
       105     19000
```
6 rows selected.

*SELECT (specific attributes on condition)

```
SQL> select empno,salary from emp2 where salary>15000;
EMPNO     SALARY
---------  ------------
       100     17500
       103     19000
       105     19000
```
6 rows selected.

<u>*SELECT (all attributes)</u> SQL> select * from emp2;

```
    EMPNO  EMPNAME      AGE       JOB                  DEPTNO  SALARY
    ----------------------------------------------------------------
      100    John               36     Manager                455     17500
      101    Smith              29     Clerk                   43     1200
      102    A                  32     Assistant Manager       43     13500
      103    B                  37     General Manager        355     19000
      104    E                  28     Clerk                   35      1500
      105    F                  38     General Manager         40     19000
```
6 rows selected.

iii)Write an SQL command to update the grade of the student with ID.

<u>*UPDATE (all entries)</u>
SQL> update emp2 set salary=salary+500; 6 rows updated.

```
SQL> select * from emp2;
EMPNO  EMPNAME    AGE      JOB              DEPTNO  SALARY
------------------------------------------------------------
      100         John         36    Manager                455       18000
      101         Smith        29    Clerk                  43         1700
      102         A            32    Assistant Manager      43        14000
      103         B            37    General Manager       355        19500
      104         E            28    Clerk                  35        2000
      105         F            38    General Manager       40       19500
```
6 rows selected.

<u>*UPDATE (specific entries)</u>
SQL> update emp2 set salary=salary+500 where age>35; 3 rows updated.

```
SQL> select * from emp2;
EMPNO  EMPNAME    AGE      JOB              DEPTNO  SALARY
```

```
-----------------------------------------------------------------------------------
       100         John        36      Manager                455      18500
       101         Smith       29      Clerk                  43        1700
       102         A           32      Assistant Manager      43       14000
       103         B           37      General Manager        355      20000
       104         E           28      Clerk                  35        2000
       105         F           38      General Manager        40       20000
```

6 rows selected.

iv)Write an SQL command to delete the student record with their NAME.

*DELETE (specific entries)

```
SQL> delete from emp2 where deptno=43; 2 rows deleted.

SQL> select * from emp2;
EMPNO  EMPNAME    AGE      JOB               DEPTNO  SALARY
-------------------------------------------------------------
   100   John          36      Manager              455    18500
   103    B            37      General Manager      355     20000
   104    E            28      Clerk                 35      2000
   105    F            38      General Manager       40     20000
```

4 rows selected.

*DELETE (specific entries) SQL> delete from emp2;
4 rows deleted.

SQL> select * from emp2; no rows selected

RESULT:

Thus the DML Commands operation has been executed successfully.

Ex: No: 1c **TRANSACTION CONTROL LANGUAGE (TCL) COMMANDS**
Date:

COMMIT command

COMMIT command is used to permanently save anytransaction into the database. Following is

commit command's syntax,

COMMIT;

ROLLBACK command

This command restores the database to last commited state. It is also used with SAVEPOINT command to jump to a savepoint in an ongoing transaction.

Following is rollback command's syntax, ROLLBACK TO

savepoint_name;

SAVEPOINT command

SAVEPOINT command is used to temporarily save a transaction so that you can rollback to that point whenever required.

Following is savepoint command's syntax, SAVEPOINT

savepoint_name;

Example:
i)COMMIT the transaction to make the first insertion permanent in the database.

INSERT INTO class VALUES(5, 'Rahul'); COMMIT;

ii)Insert a new ID and name and Set a SAVEPOINT after the insertion.

UPDATE class SET name = 'Abhijit' WHERE id = '5'; SAVEPOINT A;

INSERT INTO class VALUES(6, 'Chris');

SAVEPOINT B;

INSERT INTO class VALUES(7, 'Bravo');

SAVEPOINT C;

SELECT * FROM class;
The resultant table will look like,

id	name
1	Abhi
2	Adam
4	Alex
5	Abhijit
6	Chris
7	Bravo

iii)Insert another ID and name. Note that the second insertion was a mistake, so ROLLBACK to the savepoint.
ROLLBACK TO B;

SELECT * FROM class;

id	name
1	Abhi
2	Adam
4	Alex
5	Abhijit
6	Chris

ROLLBACK TO A;

SELECT * FROM class;

id	name
1	Abhi
2	Adam
4	Alex
5	Abhijit

RESULT:

Thus the TCL Commands operation has been executed successfully.

Ex No:2 SQL QUERIES TO PERFORM THE FOLLOWING NESTED QUERIES AND JOINS (INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN, CROSS JOIN)

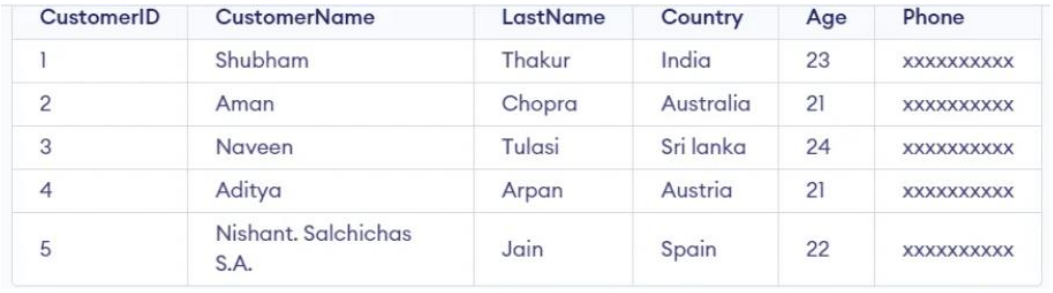
Select Command:

- The SQL SELECT query is one of the most frequently used commands to retrieve data from a database. It allows users to access and extract specific records based on defined conditions,making it an essential tool for data management and analysis.
- The SELECT clause is the first and one of the last components evaluated in the SQL query process. Before determining the final result set, the system needs to know all possible columns that might be included.

Syntax:

SELECT column1, column2 FROM table_name;

DEMO TABLE



CustomerID	CustomerName	LastName	Country	Age	Phone
1	Shubham	Thakur	India	23	xxxxxxxxxx
2	Aman	Chopra	Australia	21	xxxxxxxxxx
3	Naveen	Tulasi	Sri lanka	24	xxxxxxxxxx
4	Aditya	Arpan	Austria	21	xxxxxxxxxx
5	Nishant. Salchichas S.A.	Jain	Spain	22	xxxxxxxxxx

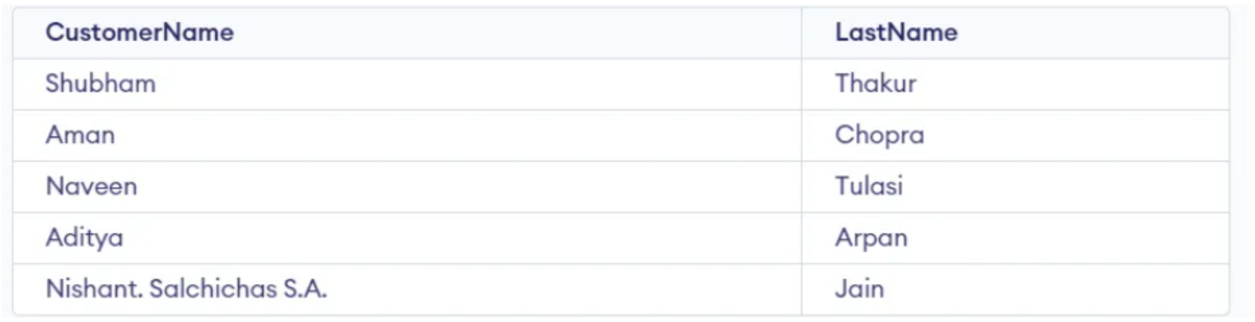
Select Specific Columns

In this example, we will demonstrate how to retrieve specific columns from the Customer table. Here we will fetch only CustomerName and LastName for each record.

Query:

SELECT CustomerName, LastName FROM Customer;

Output



CustomerName	LastName
Shubham	Thakur
Aman	Chopra
Naveen	Tulasi
Aditya	Arpan
Nishant. Salchichas S.A.	Jain

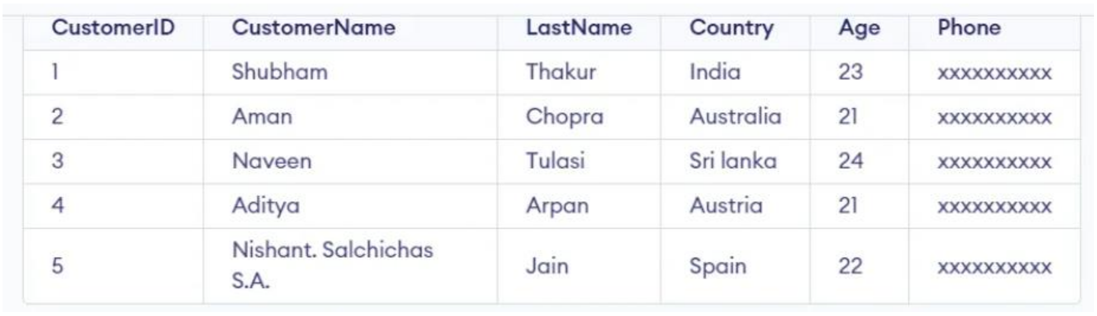
Select All Columns

In this example, we will fetch all the fields from the table Customer:

Query:

SELECT * FROM Customer;

Output



CustomerID	CustomerName	LastName	Country	Age	Phone
1	Shubham	Thakur	India	23	xxxxxxxxxx
2	Aman	Chopra	Australia	21	xxxxxxxxxx
3	Naveen	Tulasi	Sri lanka	24	xxxxxxxxxx
4	Aditya	Arpan	Austria	21	xxxxxxxxxx
5	Nishant. Salchichas S.A.	Jain	Spain	22	xxxxxxxxxx

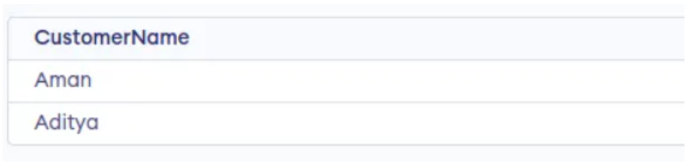
SELECT Statement with WHERE Clause

Suppose we want to see table values with specific conditions then WHERE Clause is used with select statement. In this example, filter customers who are 21 years old.

Query:

SELECT CustomerName FROM Customer where Age = '21';

Output:



CustomerName
Aman
Aditya

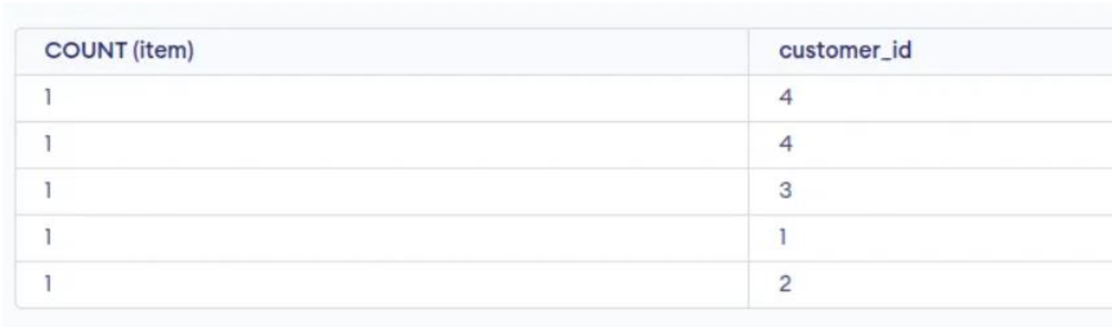
SELECT with GROUP BY Clause

In this example, we will use SELECT statement with GROUP BY Clause to Group rows and perform aggregation. Here, Count orders per customer.

Query:

SELECT Customer_id, COUNT(*) AS order_count FROM Orders GROUP BY Customer_id;

Output:



COUNT (item)	customer_id
1	4
1	4
1	3
1	1
1	2

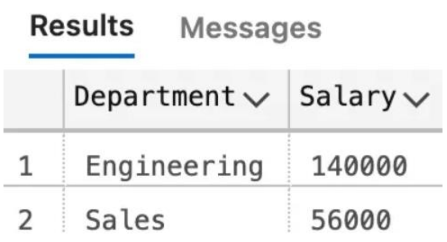
SELECT Statement with HAVING Clause

Use HAVING to filter results after grouping. Consider the following database for fetching departments with total salary above 50,000. Use WHERE for row-level filtering, HAVING for group-level filtering.

Query:

SELECT Department, sum(Salary) as Salary FROM employee
GROUP BY department
HAVING SUM(Salary) >= 50000;

Output:



Results Messages

	Department	Salary
1	Engineering	140000
2	Sales	56000

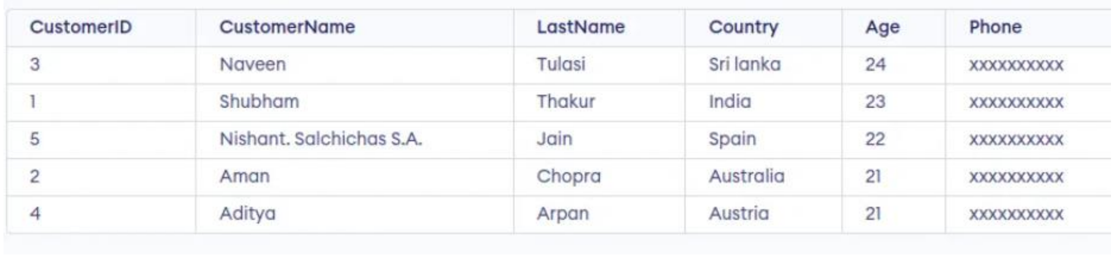
SELECT Statement with ORDER BY clause in SQL

In this example, we will use SELECT Statement with ORDER BY clause. Here, Sort results by Age in descending order.

Query:

SELECT * FROM Customer ORDER BY Age DESC;

Output:



CustomerID	CustomerName	LastName	Country	Age	Phone
3	Naveen	Tulasi	Sri lanka	24	xxxxxxxxxx
1	Shubham	Thakur	India	23	xxxxxxxxxx
5	Nishant. Salchichas S.A.	Jain	Spain	22	xxxxxxxxxx
2	Aman	Chopra	Australia	21	xxxxxxxxxx
4	Aditya	Arpan	Austria	21	xxxxxxxxxx

Limit Clause

The LIMIT clause is used to specify the number of records to return.

The LIMIT clause is useful on large tables with thousands of records. Returning a large number of records can impact performance.

LIMIT Syntax

SELECT column_name(s) FROM table_name
WHERE condition
LIMIT number;

IN Operator

The IN operator is used to check whether a column value matches any value in a list of values returned by a subquery. This operator simplifies queries by avoiding the need for multiple OR conditions.

```
SELECT S_NAME FROM STUDENT
WHERE S_ID IN (
  SELECT S_ID FROM STUDENT_COURSE
  WHERE C_ID IN (
   SELECT C_ID FROM COURSE WHERE C_NAME IN ('DSA', 'DBMS')
 )
);
```

NOT IN Operator

The NOT IN operator excludes rows based on a set of values from a subquery. It is particularly useful for filtering out unwanted results. This operator helps identify records that do not match the conditions defined in the subquery.

```
SELECT S_ID FROM STUDENT
WHERE S_ID NOT IN (
  SELECT S_ID FROM STUDENT_COURSE
  WHERE C_ID IN (
    SELECT C_ID FROM COURSE WHERE C_NAME IN ('DSA', 'DBMS')
  )
);
```

EXISTS Operator

The EXISTS operator checks for the existence of rows in a subquery. It returns true if the subquery produces any rows, making it efficient for conditional checks. This operator is often used to test for relationships between tables.

```
SELECT S_NAME FROM STUDENT S
WHERE EXISTS (
  SELECT 1 FROM STUDENT_COURSE SC
  WHERE S.S_ID = SC.S_ID AND SC.C_ID = 'C1'
);
```

ANY and ALL Operators

ANY: Compares a value with any value returned by the subquery.
ALL: Compares a value with all values returned by the subquery.

Example using ANY:

```
SELECT S_NAME FROM STUDENT
WHERE S_AGE > ANY (
  SELECT S_AGE FROM STUDENT WHERE S_ADDRESS = 'DELHI'
);
```

BETWEEN Operator

The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.
The BETWEEN operator is inclusive: begin and end values are included.

Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

LIKE Operator

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.
There are two wildcards often used in conjunction with the LIKE operator:
The percent sign % represents zero, one, or multiple characters
The underscore sign _ represents one, single character

Syntax

```
SELECT column1, column2, ... FROM table_name
WHERE column LIKE pattern;
```

Return all customers from a city that starts with 'L' followed by one wildcard character, then 'nd' and then two wildcard characters:

```
SELECT * FROM Customers
WHERE city LIKE 'L_nd__';
```

Return all customers from a city that contains the letter 'L':

```
SELECT * FROM Customers
WHERE city LIKE '%L%';
```

Return all customers that starts with 'La':

```
SELECT * FROM Customers
WHERE CustomerName LIKE 'La%';
```

Return all customers that starts with 'a' or starts with 'b':

```
SELECT * FROM Customers
WHERE CustomerName LIKE 'a%' OR CustomerName LIKE 'b%';
```

Return all customers that ends with 'a':

```
SELECT * FROM Customers
WHERE CustomerName LIKE '%a';
```

Return all customers that starts with "b" and ends with "s":

```
SELECT * FROM Customers
WHERE CustomerName LIKE 'b%s';
```

Return all customers that starts with "a" and are at least 3 characters in length:

```
SELECT * FROM Customers
WHERE CustomerName LIKE 'a__%';
```

Return all customers that have "r" in the second position:

```
SELECT * FROM Customers
WHERE CustomerName LIKE '_r%';
```

Equal to (=)

Select Persons whose department is 'Sales'

```
SELECT * FROM Persons WHERE Department = 'Sales';
```

Greater than (>)

Select Persons whose Age is greater than 35

```
SELECT * FROM Person WHERE Age> 35;
```

Less than (<)

Select Persons whose age is less than 35

```
SELECT * FROM Persons
WHERE Age < 35;
```

Greater than or equal to (>=)

Select Persons whose salary is greater than or equal to 60000

```
SELECT * FROM Persons
WHERE salary >= 60000;
```

Less than or equal to (<=)

Select Persons whose Salary is less than or equal to 60000

```
SELECT * FROM Persons
WHERE Salary <= 60000;
```

Combining Relational Operators with Logical Operators

Relational operators can also be combined with logical operators (AND, OR, NOT) to create complex conditions in SQL queries:

```
-- Select products with a price between 50 and 100
SELECT * FROM Persons
WHERE PersonsID >= 2 AND PersonsID <= 4;
```

SQL Joins (Inner, Left, Right and Full Join)

An SQL JOIN clause is used to query and access data from multiple tables by establishing logical relationships between them.

SQL INNER JOIN

The INNER JOIN keyword selects all rows from both the tables as long as the condition is satisfied. This keyword will create the result set by combining all rows from both the tables where the condition satisfies i.e value of the common field will be the same.

Syntax

```
SELECT table1.column1,table1.column2,table2.column1,.... FROM table1 INNER JOIN table2
ON table1.matching_column = table2.matching_column;
```

SQL LEFT JOIN

A LEFT JOIN returns all rows from the left table, along with matching rows from the right table. If there is no match, NULL values are returned for columns from the right table. LEFT JOIN is also known as LEFT OUTER JOIN.

Syntax

SELECT table1.column1,table1.column2,table2.column1,.... FROM table1 LEFT JOIN table2 ON table1.matching_column = table2.matching_column;

SQL RIGHT JOIN

RIGHT JOIN returns all the rows of the table on the right side of the join and matching rows for the table on the left side of the join. It is very similar to LEFT JOIN for the rows for which there is no matching row on the left side, the result-set will contain null. RIGHT JOIN is also known as RIGHT OUTER JOIN.

Syntax

SELECT table1.column1,table1.column2,table2.column1,.... FROM table1 RIGHT JOIN table2 ON table1.matching_column = table2.matching_column;

SQL FULL JOIN

FULL JOIN creates the result-set by combining results of both LEFT JOIN and RIGHT JOIN. The result-set will contain all the rows from both tables. For the rows for which there is no matching, the result-set will contain NULL values.

Syntax

SELECT table1.column1,table1.column2,table2.column1,....FROM table1 FULL JOIN table2 ON table1.matching_column = table2.matching_column;

FULL JOIN Example

This example demonstrates the use of a FULL JOIN, which combines the results of both LEFT JOIN and RIGHT JOIN. The query retrieves all rows from the Student and StudentCourse tables. If a record in one table does not have a matching record in the other table, the result set will include that record with NULL values for the missing fields

Query:

SELECT Student.NAME,StudentCourse.COURSE_ID FROM Student FULL JOIN StudentCourse ON StudentCourse.ROLL_NO = Student.ROLL_NO;

CROSS JOIN Keyword

The CROSS JOIN keyword returns all records from both tables (table1 and table2).

CROSS JOIN Syntax

SELECT column_name(s)

EXERCISE DEMO

a) Write simple SQL queries to perform the following

i)Retrieve all the records from the student’s table.

```
SELECT * FROM students;
```

ii)Display only the name and grade of each student.

```
SELECT name, grade FROM students;
```

iii)Select all students who have scored more than 80.

```
SELECT * FROM students
WHERE score > 80;
```

iv)Display all student records sorted by grade in descending order.

```
SELECT * FROM students
ORDER BY grade DESC;
```

b) Nested Queries

i)Retrieve the employee(s) who earn the highest salary in the company using a subquery

```
SELECT *
FROM employees
WHERE salary = (
   SELECT MAX(salary)
 FROM employees
);
```

ii)Display the names and salaries of employees whose salary is above the average salary of all employees

```
SELECT name, salary
FROM employees
WHERE salary > (
 SELECT AVG(salary)
 FROM employees
);
```

iii)Find the department(s) where the lowest salary is paid, using a nested query

```
SELECT DISTINCT department_id
FROM employees
WHERE salary = (
SELECT MIN(salary)
FROM employees
);
```

iv)List all employees whose department_id is found in a list of departments where at least one employee earns more than 100000, using a multi-row subquery

```
            SELECT *
FROM employees
WHERE department_id IN (
            SELECT DISTINCT department_id
            FROM employees
            WHERE salary > 100000
);
```

c) Joins (Inner Join, Left Join, Right Join, Full Join, Cross Join)

Table Assumptions:

employees(emp_id, name, department_id, ...)

departments(department_id, department_name, ...)

i) Write a SQL query to list the names of all employees along with their department names, only for those employees who have a corresponding department in the departments table. Use an INNER JOIN.

```
SELECT e.name AS employee_name, d.department_name
FROM employees e
INNER JOIN departments d
ON e.department_id = d.department_id;
```

ii) Write a SQL query to retrieve a list of all employees and their department names. If an employee does not belong to any department, show the department name as NULL. Use a LEFT JOIN.

```
SELECT e.name AS employee_name, d.department_name
FROM employees e
LEFT JOIN departments d
ON e.department_id = d.department_id;
```

iii)Write a SQL query to list all departments along with the names of employees working in those departments. If a department has no employees, display NULL for employee names. Use a RIGHT JOIN.

```
SELECT e.name AS employee_name, d.department_name
FROM employees e
RIGHT JOIN departments d
ON e.department_id = d.department_id;
```

iv)Write a SQL query to list all employees and all departments, even if some employees are not assigned to a department and some departments have no employees. Use a FULL OUTER JOIN.

```
SELECT e.name AS employee_name, d.department_name
FROM employees e
LEFT JOIN departments d ON e.department_id = d.department_id
```

v)Retrieve only the first two records from the table.

```
SELECT * FROM students
LIMIT 2;
```

Result

Thus the SQL Queries has been successfully executed.

Ex.No:3 **BASIC PL/SQL PROGRAM**

AIM:

To studyand implement PL/SQL using simple programs.

STRUCTURE OF PL/SQL:

```
declare
  <<declar active statements>>;
begin
 <<executable statements>>;
exception
  <<exception handling>>;
end;
```

SQL BLOCK STRUCTURE:

DECLARE:
Declarations of memory variables used later.

BEGIN:
SQL executable statements for manipulating table data.

EXCEPTIONS:
SQL and/or **PL/SQL** code to handle errors that may crop up
During the execution of the above code block.

END;

1. Conditional Control:

A sequence of statements can be executed based on some condition using **IF.**

Syntax:

```
If<condition> then <Action>
Else <Action>
End if;
```

2. Iterative Control:

A sequence of statements can be executed any number of times using loop constructs.

a) Simple loop

Syntax:

```
Loop
Statements;
End loop;
```

b) While loop

Syntax:

```
While<condition>
Loop
Statements;
End loop;
```

c) ***For loop***

Syntax:

```
For counter in[reverse] lower bound .. upper bound
Loop
statements
End loop;
```

PROGRAM TO ADD TWO NUMBERS AND DISPLAYING THE SUM

ALGORITHM

```
    *Declare a,b and c.
    *Get input of a and b.
    *Do arithmetic addition operation between a and b.
    *And store the value in c.
    *Display the sum as output.
```

PROGRAM

```
declare
a number(3);
b number(3);
c number(3);
begin
a:=&a;
b:=&b;
c:=a+b;
dbms_output.put_line('Sum='||c);
end;
```

OUTPUT

```
Enter value for a: 34
Enter value for b: 52
Sum=86
PL/SQL procedure successfully completed.
```

RESULT:

Thus the PL/SQL for adding two numbers has been executed successfully.

Ex.No:4 SERIES OF NUMBERS (FOR LOOP)

AIM

To study and implement the PL/SQL to print a series of „n" numbers using for loop.

ALGORITHM

```
*Declare i and n.
*Get input of n.
*Initialize the value of i=1.
*Inside a for loop of i in n, print the value of i.
*Display output.
```

PROGRAM

```
declare
i number(2):=1;
n number(3):=&n;
begin
for i in 1..n
loop
dbms_output.put_line(i);
end loop;
end;
```

OUTPUT

```
Enter value for n: 6
1
2
3
4
5
6
PL/SQL procedure successfully completed.
```

RESULT:

Thus the PL/SQL to print a series of „n" numbers has been executed successfully.

Ex.No:5 SERIES OF NUMBERS (WHILELOOP)

AIM

To studyand implement the PL/SQL to print a series of „n‟ numbers using while loop.

ALGORITHM

*****Declare i and n.
*Get input of n.
*Initialize the value of i=1.
*Inside a while loop with condition i<=n, print the value of i.
*Increment the value of i.
*Display output.

PROGRAM

```
declare
i  number(2):=1;
n number(3):=&n;
begin
while(i<=n)
loop
dbms_output.put_line(i);
i:=i+1;
end loop;
end;
```

OUTPUT

```
Enter value for n: 15
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
PL/SQL procedure successfully completed.
```

RESULT: Thus the PL/SQL to print a series of „n‟ numbers using whileloop has been executed successfully.

Ex.No:6 **DEBIT FROM ACCOUNT**

AIM

To study and implement the PL/SQL for debiting an amount from an entry in a table by this program

ALGORITHM

```
*Declare accno,balance,debit,minbal as 500.
*Get input of accno and debit from user .
*Debit the amount from entry in the table account33 using select command.
*Using if condition, update bal attribute in table only if it is greater than minbal.
*Else print that the account is too low to debit.
*End the program.
```

PROGRAM

```
declare
accno number(5);
balance number(5);
debit number(5);
minbal number(5);
begin
minbal:=500;
accno:=&accno;
debit:=&debit;
select bal into balance from account33 whereaccno=Ano;
balance:=balance-debit;
if(balance>minbal)then
update account33 set bal=balance where accno=Ano;
else
dbms_output.put_line('Your account is too low to debit');
end if;
end;
```

OUTPUT

```
SQL> create table account33(Ano number(2),name varchar(5),bal number(7));
Table created.

SQL> select * from account33;
ANO NAME BAL
_____________________
    1   A      5000
    2   B      1000
  3     C      4000
  4     D      5000
-------------------------------------------------
```

```
Enter value for accno: 1
Enter value for debit: 2000
PL/SQL procedure successfully completed.

SQL> select * from account33;
  ANO  NAME    BAL
-----------------------
    1     A          3000
    2     B          1000
    3     C          4000
    4     D          5000
-----------------------------------------------------------
Enter value for accno: 2
Enter value for debit: 2000
Your account is too low to debit
PL/SQL procedure successfully completed.
```

RESULT:

Thus the PL/SQL for debiting an amount from an entry in atable has been executed successfully.

Ex.No:7 **SUM OF A SERIES**

AIM

To study and implement the PL/SQL to determine the sum of a series of „n‟ numbers.

ALGORITHM

*****Declare n,i and x.
*Get input of n.
*Initialize the values of i=1 and x=0.
*Inside a simple loop, calculate the sum of numbers upto „n‟.
*Display the sum as output.

PROGRAM

```
declare
n number(3):=&n;
i number(3):=1;
x number(3):=0;
begin
loop
x:=x+i;
exit when i=n;
i:=i+1;
end loop;
dbms_output.put_line('Sum = '||x);
end;
```

OUTPUT

```
Enter value for n: 6
Sum = 21
PL/SQL procedure successfully completed.
```

RESULT:

Thus the PL/SQL to determine the sum of a series of „n‟numbers has been executed

Ex.No:8 FACTORIAL OF A NUMBER

AIM

To study and implement the PL/SQL to determine the factorial of the given number.

ALGORITHM

```
*Declare n,i and x.
*Get input of n.
*Initialize the values of i=1 and x=1.
*Inside a while loop, calculate the product of numbers upto „n‟.
*Displaythe factorial as output.
```

PROGRAM

```
declare
n number(3):=&n;
i number(3):=1;
x number(3):=1;
begin
while(i<=n)
loop
x:=x*i;
i:=i+1;
end loop;
dbms_output.put_line('Factorial = '||x);
end;
```

OUTPUT

```
Enter value for n: 5
Factorial = 120
PL/SQL procedure successfully completed.
```

RESULT:

Thus the PL/SQL to determine the factorial of the given number has been executed

Ex.No:9 **REVERSING A STRING**

AIM

To studyand implement PL/SQL for reversing a string using the for loop.

ALGORITHM

*Get the input string.
*Find the length of the string.
*Extract the characters one by one from the end of the string.
*Concatenate the extracted characters.
*Displaythe concatenated reversed string.
*Stop the program.

PROGRAM

```
declare
b varchar2(10):='&b';
c varchar2(10);
l  number(2);
i  number(2);
g number(2);
d varchar2(10);
begin
l:=length(b);
g:=l;
for i in 1..l
loop
c:=substr(b,g,1);
g:=g-1;
d:=d||c;
end loop;
dbms_output.put_line('Reversed string is:');
dbms_output.put_line(d);
end;
```

OUTPUT

```
Enter value for b: hello
Reversed string is
olleh
PL/SQL procedure successfully completed.
```

RESULT:

Thus the PL/SQL for reversing a string using the for loop has been executed successfully.

Ex.No:10 **FIBONACCI SERIES**

AIM:

To studyand implement PL/SQL for Fibonacci series.

ALGORITHM:

*Initialise a,b,c,n where a=-1,b=1,c=0.
*Use "n" for „for"loop
*Add a,b and store it in c.
*Exchange b to a and c to b.
*Get output from c.

PROGRAM :

```
declare
a number(3);
b number(3);
c number(3);
n number(3);
i number(3);
begin
n:=&n;
c:=0;
b:=1;
a:=-1;
for i in 1..n
loop
c:=a+b;
a:=b;
b:=c;
dbms_output.put_line(c);
end loop;
end;
```

OUTPUT

```
Enter value for n: 6
0
1
1
2
3
5
PL/SQL procedure successfully completed.
```

RESULT: Thus the PL/SQL for Fibonacci series has been executed successfully.

Ex.No:11 DIPLAY SPECIFIC ATTRIBUTE FROM TABLE

AIM

To studyand implement the PL/SQL for displaying a particular attribute from a table.

ALGORITHM

```
*Declare a,b,c,d,e,f,g.
*Get input of a and b.
*Do arithmetic operations between a and b.
*And store it in c,d,e,f,g.
*Display output.
```

PROGRAM

```
declare
vsal number(6); rid
number(6):=11;
begin
select salary into vsal from emp33 where id=rid;
dbms_output.put_line(vsal);
dbms_output.put_line('The employee '||rid||'‟ has salary „||vsal);
end;
```

OUTPUT

```
SQL> create table emp33(id number(3),name varchar(5),salary number(5));
Table created.

SQL> select * from emp33;

   ID NAME  SALARY
 ------------------
    9   AA     5000
   10   BB    10000
   11   CC     3000

3000
The employee 11 has salary 3000.
PL/SQL procedure successfully completed.
```

RESULT:

Thus the PL/SQL for displaying a particular attribute from atable has been executed successfully.

Ex.No.12 Mini Project(Banking System)

AIM:

To Connect Oracle Database from Visual Basic 6.0 for Banking System

PROCEDURE:

1. Start the process.
2. Create the table using oracle,SQL and insert some tuples in it.
3. Commit the table and exit.
4. For ODBC connectivity, select control panel -> administrative tools->ODBC data sources, select drivers menu, select Microsoft ODBC driver, click ok.
5. In VB 6.0, choose project menu, under that references.
6. Select Microsoft DAO 3.51 object library and Microsoft DAO 3.6 object library
7. Create forms for the project for the appropriate requirements specified.
8. Execute and terminate the project.

CODING:

GENERAL:

```
Dim conn As ADODB.Connection
Dim rs As ADODB.Recordset
```

INSERT:

```
Private Sub Command1_Click()
Set rs = New ADODB.Recordset
rs.Open "select * from emp890", conn, adOpenDynamic, adLockOptimistic
rs.AddNew
rs(0) = Val(Text1.Text)
rs(1) = Text2.Text
rs.Update
rs.Close
MsgBox "record created"
End Sub
```

DELETE:

```
Private Sub Command2_Click()
Set rs = New ADODB.Recordset
rs.Open "select * from emp890 where eno=" & Val(Text1.Text), conn, adOpenDynamic,
adLockOptimistic
rs.Delete
Text1.Text = ""
MsgBox "deleted"
rs.Close
End Sub
```

VIEW:

```
Private Sub Command3_Click()
Set rs = New ADODB.Recordset

rs.Open "select * from emp890", conn, adOpenDynamic, adLockOptimistic
rs.MoveFirst
With rs
Text1.Text = rs(0)
```

```
Text2.Text = rs(1)
End  With
rs.Close
End Sub
```

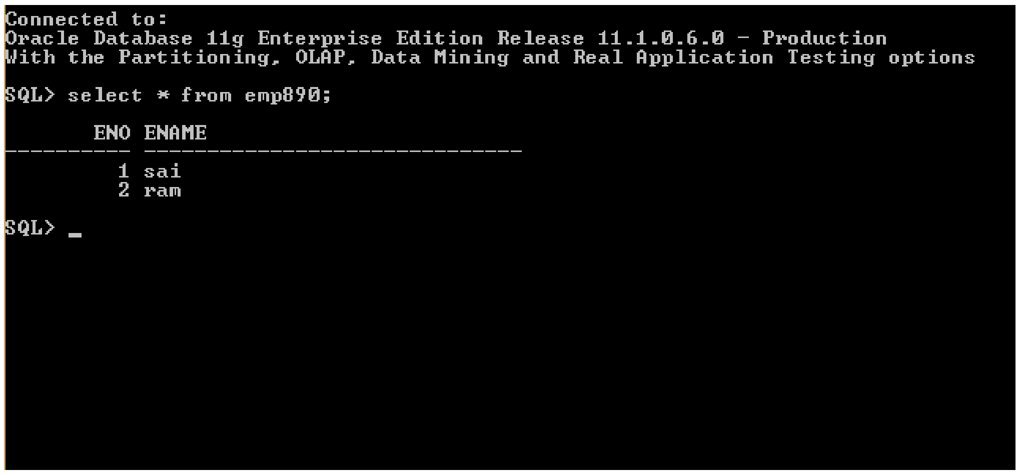
CLEAR:

```
Private Sub Command3_Click()
Text1.Text = ""
Text2.Text = ""
End Sub
```

EXIT:

```
Private Sub Command4_Click()
unload me
End Sub
```

BACK END:



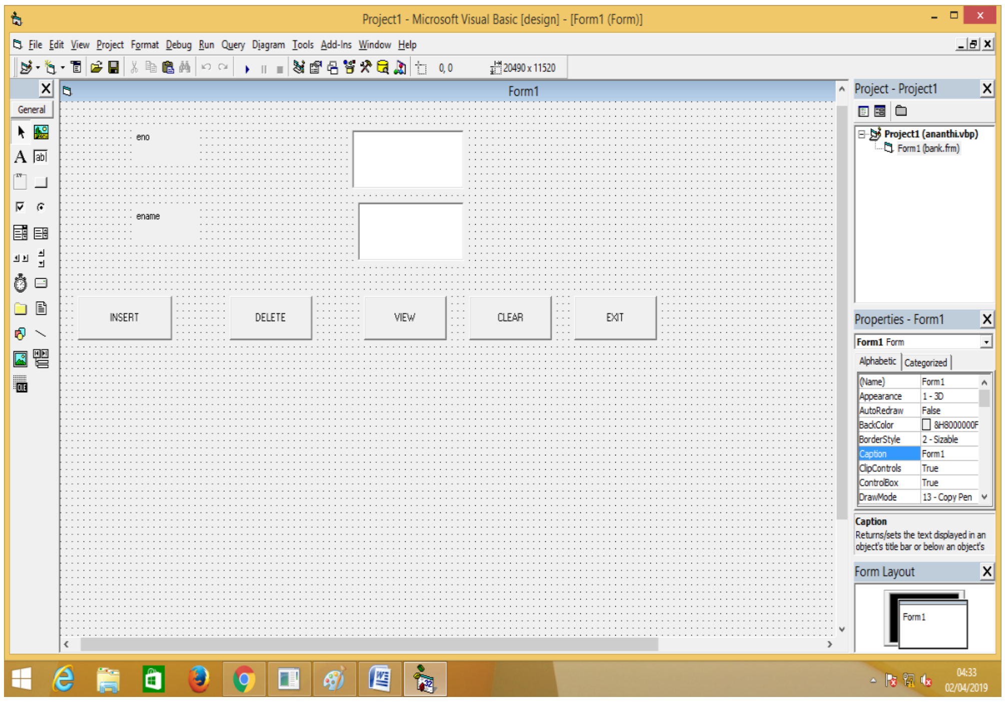
```
Connected to:
Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> select * from emp890;

       ENO ENAME
---------- ------------------------------
         1 sai
         2 ram

SQL> _
```

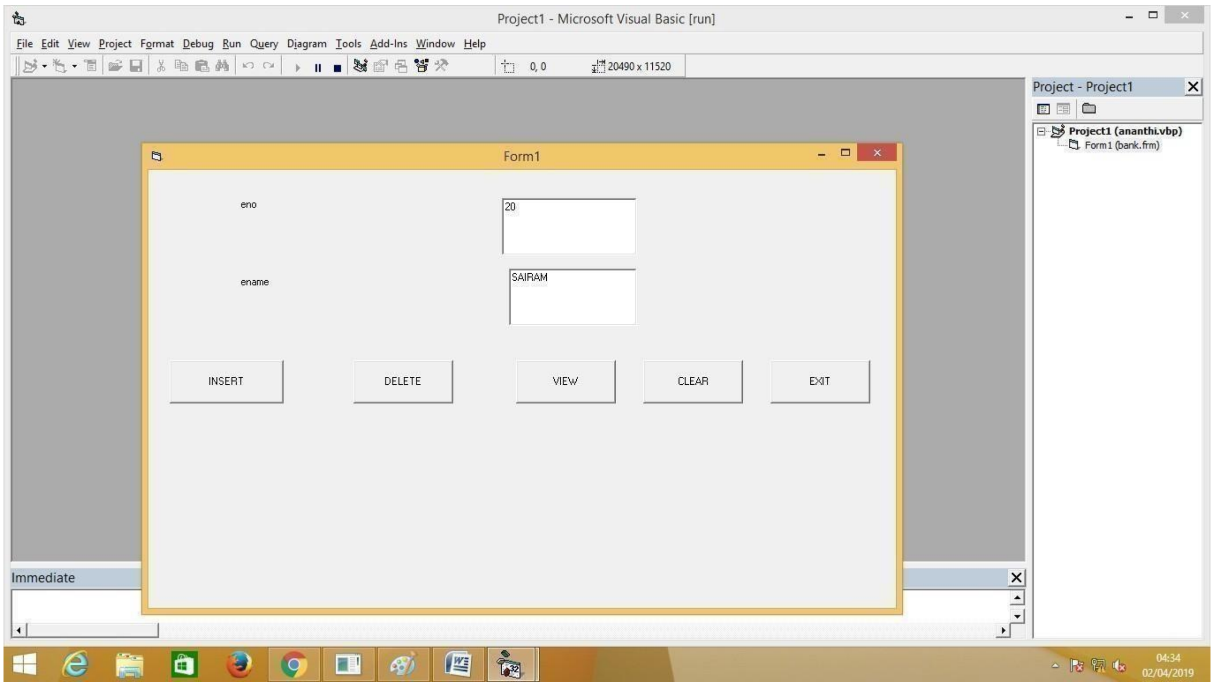
FORM DESIGN:

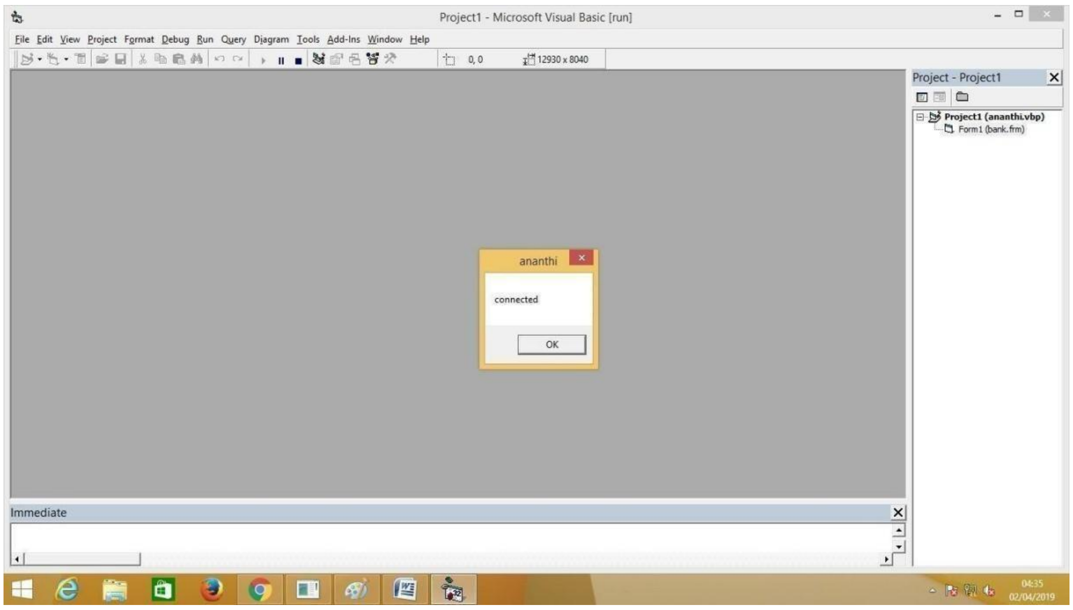


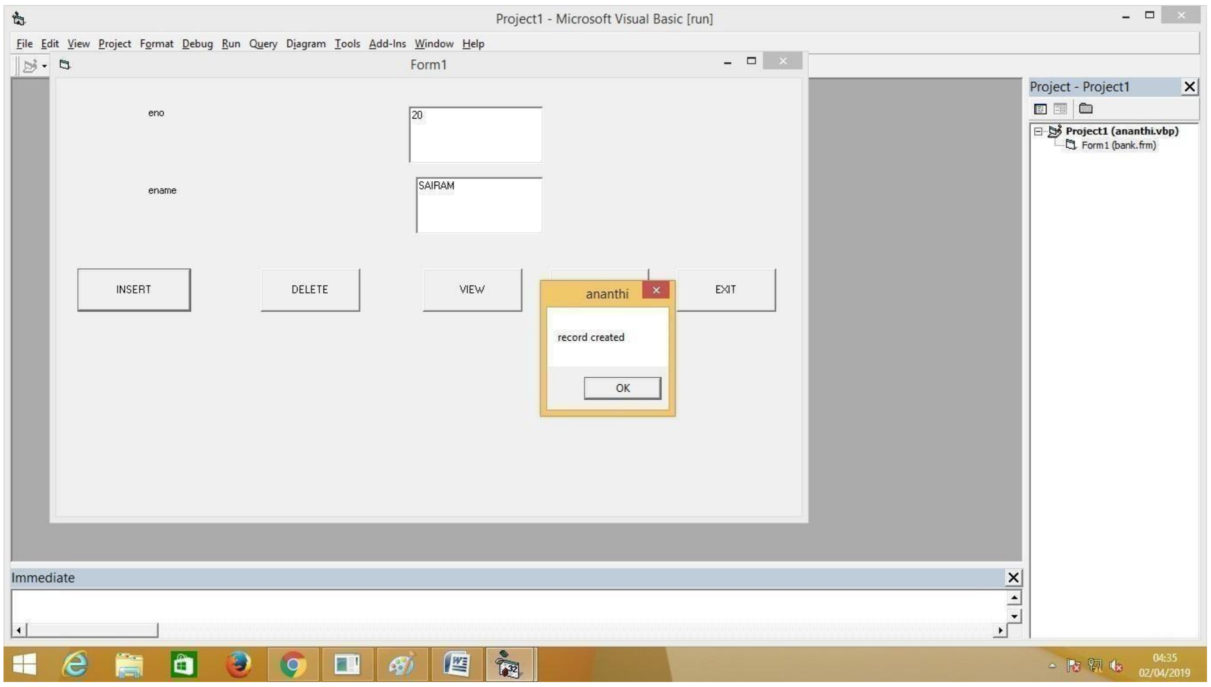
```
(Name)        Form1
Appearance    1 - 3D
AutoRedraw    False
BackColor     &H8000000F
BorderStyle   2 - Sizable
Caption       Form1
ClipControls  True
ControlBox    True
DrawMode      13 - Copy Pen
```

Caption
Returns/sets the text displayed in an object's title bar or below an object's

INSERTING A TUPLE:


Project1 - Microsoft Visual Basic [run]
File Edit View Project Format Debug Run Query Diagram Tools Add-Ins Window Help
0, 0
20490 x 11520
Project - Project1
Project1 (ananthi.vbp)
Form1 (bank.frm)
Form1
eno
20
ename
SAIRAM
INSERT
DELETE
VIEW
CLEAR
EXIT
Immediate
04:34
02/04/2019


Project1 - Microsoft Visual Basic [run]
File Edit View Project Format Debug Run Query Diagram Tools Add-Ins Window Help
0, 0
12930 x 8040
Project - Project1
Project1 (ananthi.vbp)
Form1 (bank.frm)
ananthi
connected
OK
Immediate
04:35
02/04/2019


Project1 - Microsoft Visual Basic [run]
File Edit View Project Format Debug Run Query Diagram Tools Add-Ins Window Help
Form1
Project - Project1
Project1 (ananthi.vbp)
Form1 (bank.frm)
Immediate

eno 20

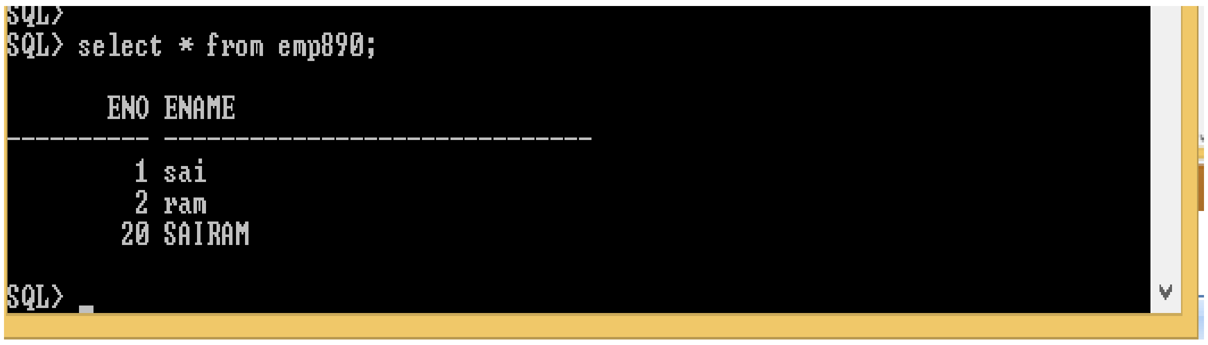
ename SAIRAM

INSERT DELETE VIEW EXIT

ananthi

record created

OK



```
SQL>
SQL> select * from emp890;

       ENO ENAME
---------- ------------------------------
         1 sai
         2 ram
        20 SAIRAM

SQL> _
```

RESULT:

Thus the Oracle Database from Visual Basic 6.0 for Banking System has been executed successfully.