**Department of CSE (Data Science)**

**II Year I Semester**

**Data base Management Systems Lab (R20DS21L3)**

**Lab Manual**

**(R20 Regulations)**

A black and purple logo

Description automatically generated

**Narasaraopeta Engineering College**

**Kotappakonda Road, Yellamanda (Post), Narasaraopet – 522601, Guntur District, AP**

**Approved by AICTE, New Delhi & Permanently affiliated to JNTUK, Kakinada, Code: 47,**

**Accredited by NBA & NAAC, RTA Approved Pollution test Centre, ISO 9001: 2008 Certified Institution**

**Phone: 08647-239905 Website: www.nrtec.in**

A close-up of a logo

Description automatically generated

**Institute’s Vision, Mission & Values**

**Vision:**

To emerge as a Centre of excellence in technical education with a blend of effective student centric teaching learning practices as well as research for the transformation of lives and community.

**Mission:**

M1: Provide the best class infra-structure to explore the field of engineering and research.

M2: Build a passionate and a determined team of faculty with student centric teaching, imbibing experiential, innovative skills.

M3: Imbibe lifelong learning skills, entrepreneurial skills and ethical values in students for addressing societal problems.

**Values:**

* **Student-centric education:** Meeting the community’s and student’s needs by developing a world-class educational environment with cultural values.
* **Excellence**: Giving special attention towards the standards of integrity and performance to help the institute in leading academic achievements and professional goals.
* **Collaboration**: Seeking the latest input and working closely with all the industrial sectors and Society for the continuous upgradation of the quality of education.
* **Diversity**: Creating a favourable on-campus environment in which the goals and learning styles of all students are recognised and nurtured.
* **Continuous** Development: Encouraging enthusiastic, innovative thinkers and learners to strive for personal growth in the world of inventions and start-ups.
* **Technological** **Advancement**: Keeping pace with evolving technology and professional trends to prepare all its students to achieve success in the workplace.

A close-up of a logo

Description automatically generated

**Department of CSE (Data Science)**

**Vision:**

To nurture skilled professionals in the Data Science for industry innovation and create meaningful societal impact through advanced analytics, machine learning, and impactful data-driven solutions.

**Mission:**

M1: To develop skilled data scientists who can effectively solve challenges in data analytics through comprehensive education and practical training in statistical analysis, machine learning, data visualization, and data manipulation.

M2: To develop students with strong research capabilities who can revolutionize multiple fields through the application of data science.

M3: To develop ethical data science professionals who utilize data for the welfare of society.

**Program Specific Outcomes:**

**PSO1**: Apply Data Science Techniques, statistical analysis, machine learning algorithms, data visualization, and data manipulation effectively to solve complex data problems.

**PSO2**: Demonstrate proficiency in conducting data collection, preprocessing, analysis, and interpretation, contributing to the advancement of the field.

**PSO3**: Able to independently carry out research and investigation to solve societal problems.

**Program Educational Objectives:**

A close-up of a logo

Description automatically generated

**PROGRAM OUTCOMES (POs)**

|  |  |
| --- | --- |
| P01: | Engineering Knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems. |
| P02: | Problem Analysis: Identify, formulate, review research literature and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences. |
| P03: | Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety and the cultural, societal and environmental considerations. |
| P04: | Conduct investigations of complex problems: Use research based knowledge and research methods including design of experiments, analysis and interpretation of data and synthesis of the information to provide valid conclusions. |
| P05: | Model tool usage: Create, select and apply appropriate techniques, resources and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations. |
| P06: | The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice. |
| P07: | Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development. |
| PO8: | Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice. |
| PO9: | Individual and team work: Function effectively as an individual and as a member or leader in diverse teams, and in multidisciplinary settings. |
| PO10: | Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations and give and receive clear instructions. |
| PO11: | Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments. |
| PO12: | Life-long learning: Recognize the need for and have the preparation and ability to engage in independent and life-long learning in the broad test context of technological change. |

A close-up of a logo

Description automatically generated

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **PROGRAM :** | B.Tech.(CSE-DS) | |  |  |  | | **ACADEMIC YEAR :** | | 2022-23 | |
| **YEAR/SEM :** | II/I |  |  |  |  | | **REGULATION :** | | R20 | |
| **COURSE :** | DBMS LAB | | | | |  | |  | |

**COURSE OUTCOMES**

|  |  |  |
| --- | --- | --- |
| **CO. NO.** | **COURSE OUTCOME** | **BLOOMS TAXANOMY LEVEL** |
| CO 216.1 | Utilize SQL to execute queries for creating database and performing data manipulation operations | Applying – 3 |
| CO 216.2 | Examine integrity constraints to build efficient databases | Analysing – 3 |
| CO 216.3 | Apply Queries using Advanced Concepts of SQL | Applying – 3 |
| CO 216.4 | Build PL/SQL programs including stored procedures, functions, cursors and triggers | Create – 6 |

**Mapping CO with POs/PSOs**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **CO** | POS | | | | | | | | | | | | PSOs | |
| PO1 | PO2 | PO3 | PO4 | PO5 | PO6 | PO7 | PO8 | PO9 | PO10 | PO11 | PO12 | PSO1 | PSO2 |
| CO 216.1 | 3 | 3 | 3 |  |  |  |  |  |  |  |  |  | 3 | 3 |
| CO 216.2 | 3 | 3 | 3 |  |  |  |  |  |  |  |  |  | 3 | 3 |
| CO 216.3 | 3 | 3 | 3 |  |  |  |  |  |  |  |  |  | 3 | 3 |
| CO 216.4 | 3 | 3 | 3 |  |  |  |  |  |  |  |  |  | 3 | 3 |
| **No. of COs mapping with PO** | 4 | 4 | 4 |  |  |  |  |  |  |  |  |  | 4 | 4 |
| **Average** | 3 | 3 | **3** |  |  |  |  |  |  |  |  |  | 3 | 3 |

# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

(**Data Science**)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **II Year – I Semester** | **INTERNAL MARKS**  **15** | **EXTERNAL MARKS**  **35** | **TOTAL**  **MARKS**  **50** | **L** | **T** | **P** | **C** |
|  | **0** | **0** | **3** | **1.5** |
| **DATABASE MANAGEMENT SYSTEMS LAB** | | |  |  | **SUBJECT CODE**  **R20DS21L3** | | |

**Course Objectives:**

This Course will enable students to

* Populate and query a database using SQL DDL/DML Commands
* Declare and enforce integrity constraints on a database
* Writing Queries using advanced concepts of SQL
* Programming PL/SQL including procedures, functions, cursors and triggers

# COURSE OUTCOMES:

# After Completion of this course student must be able to

# CO1: Apply SQL commands like DDL, DML and DCL to perform different Database operations

# [K3].

# CO2: Develop PL/SQL block statements, control statements and cursors. [K3]

# CO3: Develop PL/SQL programs using functions and procedures. [K3]

# CO4: Develop PL/SQL programs using packages and Triggers. [K3

**Programs List**

1. Creation, altering and dropping of tables and inserting rows into a table (use constraints

While creating tables) examples using SELECT command.

2. Queries using Conversion functions (to\_char, to\_number and to\_date), string functions

(Concatenation, lpad, rpad, ltrim, rtrim, lower, upper, initcap, length, substr and instr), date

Functions (Sysdate, next\_day, add months, last\_day, months between, least, greatest, trunc,

round, to\_char, to\_date).

3. Queries using Aggregate functions (COUNT, SUM, AVG, MAX and MIN), GROUP BY, HAVING and Creation and dropping of Views.

4. Queries (along with sub Queries) using ANY, ALL, IN, EXISTS, NOTEXISTS, UNION,

Constraints. Example: - Select the roll number and name of the student who secured fourth rank in the class

5. i) Creation of simple PL/SQL program which includes declaration section, executable

Section and exception –Handling section (Ex. Student marks can be selected from the table

and printed for those who secured first class and an exception can be raised if no records

Were found)

ii) Insert data into student table and use COMMIT, ROLLBACK and SAVEPOINT in

PL/SQL block.

6. Develop a program that includes the features NESTED IF, CASE and CASE expression.

The program can be extended using the NULLIF and COALESCE functions

.

7. Program development using WHILE LOOPS, numeric FOR LOOPS, nested loops using

ERROR Handling, BUILT –IN Exceptions, USE defined Exceptions, RAISE-

APPLICATION ERROR.

8. Programs development using creation of procedures, passing parameters IN and OUT of

PROCEDURES.

9. Program development using creation of stored functions, invoke functions in SQL

Statements and write complex functions.

10. Program development using creation of package specification, package bodies, private

Objects, package variables and cursors and calling stored packages.

11. Develop programs using features parameters in a CURSOR, FOR UPDATE CURSOR,

WHERE CURRENT of clause and CURSOR variables.

12. Develop Programs using BEFORE and AFTER Triggers, Row and Statement Triggers and

INSTEAD OF Triggers.

TEXT BOOKS:

1. SCOTT URMAN, “ORACLE DATA BASE LOG PL/SQL Programming”, Tata Mc-Graw Hill.

REFERENCES:

1. Benjamin Rosenzweig, Elena Silvestrova “ORACLE PL/SQL by example”, Pearson

Education 3/e

2. Dr.P.S. Deshpande, “SQL & PL/SQL for Oracle 10g”, Black Book.

3. Pranab kumar Das Gupta, P Radha Krishna, “Data Base Management System, Oracle

SQL and PL/SQL”, PHI.

WEB REFERENCES:

3. nptel.ac.in/courses/106106093

4. nptel.ac.in/courses/106104135

**5** https://www.edx.org/course/databases-5-sq

**INDEX**

|  |  |  |
| --- | --- | --- |
| **Exercise No.** | **Details** | **Page No’s** |
| 1 | Creation, altering and dropping of tables and inserting rows into a table (use constraints while creating tables) examples using SELECT command | 1-4 |
| 2 | Queries using Conversion functions , string functions, ate  functions | 5-8 |
| 3 | Aggregate Functions, Group by and Having | 9-10 |
| 4 | Queries (along with sub Queries) using any, all, in, exists, not exists, union,  Interest, constraints. | 11-13 |
| 5 | 1. Simple PL/SQL Block 2. Commit, Rollback and SavePoint in PL/SQL Block | 14-16 |
| 6 | Develop a program that includes the features If and Nested If | 17-18 |
| 7 | Program development using while loops, numeric for loops, nested loops using error handling, built –in exceptions, use defined exceptions, raise-  application error | 19-22 |
| 8 | Programs development using creation of procedures, passing parameters in and out of procedures. | 23 |
| 9 | Program development using creation of stored functions, invoke functions in SQL Statements and write complex functions. | 24 |
| 10 | Program development using creation of package specification, package bodies, private objects, package variables and cursors and calling stored packages. | 25-26 |
| 11 | Develop programs using features parameters in a cursor, for update cursor, where current of clause and cursor variables. | 27-28 |
| 12 | Develop programs using before and after triggers, row and statement triggers and instead of triggers | 29-30 |

**Exercise -1**

**Creation, altering and dropping of tables and inserting rows into a table (use constraints while creating tables) examples using SELECT command.**

***Aim:***

***To know how to use DDL commands to create, alter and drop tables, DML commands for inserting rows into tables, and display the information using select statement.***

**DDL Commands (Data Definition Language Commands):** These commands are used to create, alter and drop database objects like table, views, constraints, sequences, etc.

**Create:** This command is used to create database objects.

**Alter:** This command is used to alter the structure of a table.

**Drop:** This command is used to drop the database objects.

**Table:** A Table is a collection of rows and columns. Data in relational model is stored in tables.

**Constraints:** Constraints are the rules that can be applied at column level or table level. The following are the various constraints;

|  |  |
| --- | --- |
| **Constraint** | **Purpose** |
| Unique | It ensures that all the values in a column should be unique. |
| Not Null | It ensures that a value should be entered in a column. |
| Primary Key | It is used to identify a row in the table. A table can contain a single primary key. |
| Check | It is used to check all values in a column should satisfy a condition. |
| Default | It is used to set a default value whenever no value for that column is entered. |
| Foreign Key | It is used to establish relationship between two tables. |

**Creating Tables:**

1. SQL> Create table dept

(deptno number primary key,

Dname varchar2(20));

Table(s) Created.

1. SQL> Create table emp

(empno number primary key,

ename varchar2(20) not null,

job varchar2(10),

mgr number,

hiredate date,

sal number,

comm number,

deptno number references dept(deptno));

Table (s) created

1. SQL> Create table dept

(salgrade number primary key,

Losal number,

Hisal number);

Table(s) created

1. SQL> Create table example

(sno number primary key,

sname varchar2(20),

address varchar2(10));

Table(s) created

**Altering table:**

1. SQL> alter table dept

(add Loc varchar2(10));

**Dropping table:**

1. SQL> drop table example;

1 table dropped

**DML Commands (Data Manipulation Language Commands):** These commands are used for updating, inserting, deleting and querying the data in the database.

**Select, Insert, Update, and Delete**

**Inserting data into tables:** Insert command is used to insert a row into the table.

Syn:

INSERT INTO tablename [(columns list)] {VALUES (value-1,...) | subquery }

SQL> Insert into dept values (&dno, ‘&dnm’, ‘&lc’);

Enter value for dno: 10

Enter value for dnm: Accounting

Enter value for Loc: Newyork

1 row(s) inserted.

In the same way the following rows are inserted into dept, emp and salgrade tables.

**Select : This statement is used to fetch data from the database.**

**Clauses of the SELECT Statement:**

|  |  |
| --- | --- |
| **SELECT** | Lists the columns from base tables or view to be projected into the table that will be the result of the command |
| **FROM** | Identifies the tables or views from which columns will be chooses to appear in the result table, and includes the tables or views needed to join tables to process the query. |
| **WHERE** | Includes the conditions for row selection within a single table or view, and the conditions between tables or views for joining. |

Display the employee names whose designation is manager.

SQL> Select ename from emp here job=’Manager’;

Output:

|  |
| --- |
| **ENAME** |
| JONES |
| BLAKE |
| CLARK |

**Dept:**

|  |  |  |
| --- | --- | --- |
| **DEPTNO** | **DNAME** | **LOC** |
| 10 | ACCOUNTING | NEWYORK |
| 20 | RESEARCH | DALLAS |
| 30 | SALES | CHICAGO |
| 40 | OPERATIONS | BOSTON |

**Emp:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **EMPNO** | **ENAME** | **JOB** | **MGR** | **HIREDATE** | **SAL** | **COMM** | **DEPTNO** |
| 7369 | SMITH | CLARK | 7902 | 17-DEC-80 | 800 |  | 20 |
| 7499 | ALLEN | SALESMAN | 7698 | 20-FEB-81 | 1600 | 300 | 30 |
| 7521 | WARD | SALESMAN | 7698 | 22-FEB-81 | 1250 | 500 | 30 |
| 7566 | JONES | MANAGER | 7839 | 02-APR-81 | 2975 |  | 20 |
| 7654 | MARTIN | SALESMAN | 7698 | 28-SEP-81 | 1250 | 1400 | 30 |
| 7698 | BLAKE | MANAGER | 7839 | 01-MAY-81 | 2850 |  | 30 |
| 7782 | CLARK | MANAGER | 7839 | 09-JUN-81 | 2450 |  | 10 |
| 7788 | SCOTT | ANALYST | 7566 | 19-APR-87 | 3000 | 1500 | 20 |
| 7839 | KING | PRESIDENT |  | 17-NOV-81 | 5000 |  | 10 |
| 7844 | TURNER | SALESMAN | 7698 | 08-SEP-81 | 1500 | 0 | 30 |
| 7876 | ADAMS | CLERK | 7788 | 23-MAY-87 | 1100 |  | 20 |
| 7900 | JAMES | CLERK | 7698 | 03-DEC-81 | 950 |  | 30 |
| 7902 | FORD | ANALYST | 7566 | 03-DEC-81 | 3000 |  | 20 |
| 7934 | MILLER | CLERK | 7782 | 23-JAN-82 | 1300 |  | 10 |

**Salgrade**

|  |  |  |
| --- | --- | --- |
| **GRADE** | **LOSAL** | **HISAL** |
| 1 | 700 | 1200 |
| 2 | 1201 | 1400 |
| 3 | 1401 | 2000 |
| 4 | 2001 | 3000 |
| 5 | 3001 | 9999 |

**Exercise -2**

**Queries using Conversion functions (to\_char, to\_number and to\_date), string functions (Concatenation, lpad, rpad, ltrim, rtrim, lower, upper, initcap, length, substr and instr), date functions (Sysdate, next\_day, add\_months, last\_day, months\_between, least, greatest, trunc, round, to\_char, to\_date)**

***AIM:***

***Practice queries to know the usage of various built-in functions***

SQL has many built-in functions to perform operations on the data. A function may or may not take arguments but it always returns a value as a result. If the function takes arguments, then these arguments are to be given within parentheses followed by the function name.

General Form : functionname [(parameter1, parameter2,....)]

**Type of functions:**

* String Functions
* Numeric Functions
* Data & Time Functions
* Conversion Functions
* Group Functions (Aggrege Functions)
* Miscellaneous Functions

**DUAL Table**

This is a table that is made available to every account in Oracle database. This table contains one row and one column. This table can be used with SELECT when result of the expression is to be displayed only for once.

Conversion Functions: These functions are used to convert from one data type to another.

TO\_CHAR Function

This function is used to convert the given DATE or NUMBER to CHAR type. TO\_CHAR function may also be used to format the given date or number while converting the value to CHAR type. For example, to display date in DD-MM-YYYY format instead of standard format - DD-MON-YY, enter the following:

**SQL>Select to\_char(sysdate,'dd Month yyyy hh24:mi:ss') from dual;**

**Output:**

**TO\_CHAR(SYSDATE,'DDMONTHYY**

**--------------------------**

**15 August 2000 02:18:56**

TO\_DATE function

TO\_DATE is used to convert a CHAR type value to DATE type. If the value is in DD-MON-YY or DD-MM-YYYY format then TO\_DATE is not needed because Oracle implicitly converts the value to DATE type.

TO\_NUMBER function

This function is required in only two occasions. The following are the two cases.

* To convert formatted number to number.
* To sort CHAR data in numeric order.

**SQL> Select to\_number('$333','$999') \* 20 from dual**

**Output:**

**TO\_NUMBER('$333','$999')\*20**

**---------------------------**

**6660**

**String Functions:** String functions are the functions used to manipulate various operations on strings. Some functions are;

Concatenate

|  |  |
| --- | --- |
| **Function** | **Description** |
| **LENGTH (string)** | Returns the number of characters in the *string*. |
| **LOWER (string)** | Returns the string after converting the string to lowercase. |
| **UPPER (string)** | Returns the string after converting the string to uppercase. |
| **INITCAP (string)** | Converts first character of every word to uppercase and remaining to lower case. |
| **LPAD (string, length [, fillstring])** | Makes the *string* of the given *length* by padding the string on the left either with space or with *fillstring*. |
| **RPAD (string, length [, fillstring])** | Same as LPAD but pads on the right. |
| **LTRIM (string [, charset])** | Removes all left most characters of string up to the first character that is not in the *charset*. if *charset* is not given then it defaults to blank. |
| **RTRIM (string [, charset])** | Same as LTRIM, but trims on the right. |
| **TRIM (string)** | Trims space on both sides. |
| **SUBSTR (string, pos , length)** | Extracts *length* number of characters from position *pos* in the string. If *length* is not given then extracts everything from *pos*. |
| **INSTR (s1,s2 [,pos [,occurrence]])** | Finds the starting position of *s2* in *s1*. If *occurrence* is not given then it finds first occurrence. Search starts at *pos*, if given, otherwise at the first character in *s1*. |
| **ASCII (string)** | Returns ASCII code of the first character in the given string |
| **CHR (number)** | Returns ASCII character for the given ASCII code. |
| **TRANSLATE (string, from, to)** | Replaces characters in *from* with *to* in string. |
| **REPLACE (string, source, replace)** | Replaces *source* in *string* with *replace*. |

**Ex: SQL> Select Length(“Rama”), Upper(“Rama”) from dual;**

**Output:**

**Length(“Rama”) Upper(“Rama”)**

**4 RAMA**

**Data & Time Functions:** Date functions operate on values of DATE datatype. Except MONTHS\_BETWEEN all date functions return DATE data type. The following is the list of DATE functions.

Sysdate

|  |  |
| --- | --- |
| **Function** | **Description** |
| ADD\_MONTHS(date, count) | Adds *count* number of months to *date*. |
| MONTHS\_BETWEEN (date1, date2) | Returns number of months between *date1* and *date2*. |
| LAST\_DAY(date) | Returns the last day of the month in which *date* is. |
| NEXT\_DAY(date, ’day’) | Gives the date of next *day* after the date, where *day* name of the week like ‘Monday’. |
| NEW\_TIME(date, ’this’, ’other’) | Returns time in *other* time zone for time of *this* time zone. |
| ROUND(date) | Rounds the date depending upon the time. If time is at or after 12 hours then date is incremented. Time is always set to beginning of the day (0:0:0). |
| TRUNC(date) | Same as ROUND (date) but doesn’t increment date. |

**SQL> Select sysdate from dual;**

**Output:**

**sysdate**

**15-OCT-2020**

**Exercise -3**

**Queries using Aggregate functions (COUNT, SUM, AVG, MAX and MIN), GROUP BY, HAVING and Creation and dropping of Views.**

***Aim:***

***To know the purpose of group by, order by and having clauses in SQL.***

**Group by:** This clause is used in collaboration with select statement, which is used to arrange the identical data into a group for summarizing the information. Group by is followed by where clauses and precedes having clause and then order by clause if any.

**Having:** Having clause is used to check the conditions on aggregate functions and is used in collaboration with group by clause.

**Order by:** Order by clause is used to display the data in an order. By default the data is displayed in ascending order. Desc keyword followed by order by is used to display the data in descending order.

**Ex:**

**Display maximum salary in each department.**

**SQL> Select deptno, max(sal) from emp group by deptno;**

**Output:**

**Deptno max(sal)**

**10 5000**

**20 3000**

**30 2850**

**Display department number, average salary for all the departments having more than three peoples?**

**SQL> Select deptno, avg(sal) from emp group by deptno having count(\*)>3;**

**Output:**

**Deptno avg(sal)**

**20 2175**

**30 1567**

**Find the employees who earn the highest salary in each job type. Sort in ascending salary.**

**SQL> Select ENAME, max(sal) from emp group by job order by max(sal) desc;**

**Output:**

**job max(sal)**

**KING 5000**

**SCOTT 3000**

**JONES 2975**

**ALLEN 1600**

**MILLER 1300**

**SMITH 800**

**Write a query to display total number of employees, total salary, averages salary of 10,30,50 departments.**

**SQL> Select DEPTNO, COUNT(\*), SUM(SAL), AVG(SAL) FROM EMP GROUP BY DEPTNO HAVING DEPTNO IN(10,30,50);**

**Output:**

**Deptno count(\*) sum(sal) avg(sal)**

**10 3 8750 2916.67**

**30 6 9400 1566.67**

**Exercise- 4**

**Queries (along with sub Queries) using ANY, ALL, IN, EXISTS, NOT EXISTS, UNION, INTERSECT, LIKE, in, not in**

**Any will take least value from the subquery.**

1. **List the name,salary,comm. For those employees whose net pay is greater than or equal to any other employee salary of the company.**

**SQL>** select ename,sal,comm. from emp where sal+nvl(comm., 0) > any (select sal from emp);

ENAME SAL COMM

---------- ---------- ----------

ALLEN 1600 300

WARD 1250 500

JONES 2975

MARTIN 1250 1400

BLAME 2850

CLARK 2450

SCOTT 3000 1500

KING 5000

TURNER 1500 0

ADAMS 1100

JAMES 950

FORD 3000

MILLER 1300

1. **List empno, ename, sal of all employees whose salary is greater than all employees in department number 20;**

SQL> select empno, ename, sal from emp where sal > all (select sal from emp where deptno = 20);

EMPNO ENAME SAL

---------- ---------- ----------

7839 KING 5000

1. **List the employees who joined on 1-may-81, 3-dec-81, 17-dec-81,19-jan-80.**

**SQL> select \* from emp where hiredate in (‘1-may-81’, ‘3-dec-81’, ‘17-dec-81’,’19-jan-80’);**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **EMPNO** | **ENAME** | **JOB** | **MGR** | **HIREDATE** | **SAL** | **COMM** | **DEPTNO** |
| 7900 | JAMES | CLERK | 7698 | 03-DEC-81 | 950 |  | 30 |
| 7902 | FORD | ANALYST | 7566 | 03-DEC-81 | 3000 |  | 20 |

2 rows selected

1. **Select all employee names whose name starts with S or M.**

**SQL> select ename from emp where ename like ‘S%’**

**Union**

**select ename from emp where ename like ‘M%’**

ENAME

----------

MARTIN

MILLER

SCOTT

SMITH

1. **Query to display the employees of 10th department and also manager.**

**SQL> select \* from emp where deptno=10 intersect select \* from emp where job=’Manager’;**

**EMPNO ENAME JOB MGR HIREDATE SAL COMM DEPTNO**

**-------------------------------------------------------------------------------------------------------------------------**

**7782 CLARK MANAGER 7839 09-JUN-81 2450 10**

1. **Query to display all employee names whose name starts with S.**

**SQL> Select ename from emp where ename like ‘S%’;**

ENAME

--------

SMITH

SCOTT

1. **Query to display all employee names whose name having exactly four characters.**

**SQL> Select ename from emp where ename like ‘\_\_\_\_’;**

ENAME

----------

WARD

KING

FORD

1. **Query to display all employee names whose name starts with S.**

**SQL> Select ename from emp where ename like ‘S%’;**

ENAME

--------

SMITH

SCOTT

1. **Display the details of departments, those are not having any employees.**

**SQL> select \* from dept where deptno not in (select deptno from dept);**

**DEPTNO DNAME LOC**

**---------- ---------- ----------**

**40 OPERATIONS BOSTON**

**Exercise- 5**

1. **Create a simple PL/SQL program which includes declaration section, executable section and exception –Handling section (Ex. Student marks can be selected from the table and printed for those who secured first class and an exception can be raised if no records were found)**
2. **SQL > Create table student (sno number, snm varchar2(10), m1 number, m2 number, m3 number);**
3. **Insert the following data into student table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sno** | **Snm** | **M1** | **M2** | **M3** |
| **10** | **Rama** | **90** | **87** | **89** |
| **20** | **Raju** | **40** | **45** | **49** |
| **30** | **Amar** | **60** | **60** | **60** |

1. **PL/SQL Procedure**

Declare

Cursor c1 is select sno, snm, m1, m2, m3,(m1+m2+m3)/3 “PER” from student where ((m1+m2+m3)/3)>=60 ;

Begin

For z in c1

loop

If z.per>=60 then

Dbms\_output.put\_line (‘Student Number :’ ||z.sno);

Dbms\_output.put\_line(‘Student Name :’ ||z.snm);

Dbms\_output.put\_line(‘Subject1 Marks :’ ||z.m1);

Dbms\_output.put\_line(‘Subject2 Marks :’ ||z.m2);

Dbms\_output.put\_line(‘Subject3 Marks :’ ||z.m3);

Dbms\_output.put\_line(‘Percentage:’|| z.PER);

End if;

End loop;

Exception

When no\_data\_found then

Dbms\_output.put\_line(‘No Such Data’);

End;

Output:

Student Number: 10

Student Name : Rama

Subject1 Marks:90

Subject2 Marks:87

Subject3 Marks:89

Percentage: 86.66

Student Number: 30

Student Name : Amar

Subject1 Marks:60

Subject2 Marks:60

Subject3 Marks:60

Percentage: 60

1. **Insert data into student table and use COMMIT, ROLLBACK and SAVEPOINT in PL/SQL block.**

***Aim:***

***To know how to control the data using commit, rollback and savepoint.***

**Commit:** The commit command is used to save the changes to the database. It saves all the transactions since the last commit or rollback. The transactions between two DDL commands are committed automatically.

**Syn:** commit;

**Rollback:** The rollback command is used to undo transactions that have not already been saved to the database.

**Syn:** Rollback [to savepoint];

**Savepoint:** Savepoint is used to temporarily save transaction so that you can rollback to that point whenever required.

**Syn:** Savepoint savepoint name;

**SQL>**

Begin

Savepoint a;

Insert into student values (50, ‘Madhu’,56, 67,89);

Savepoint b;

Insert into student values (60, ‘Abc’,56, 67,89);

Rollback to b;

Commit;

End;

SQL> Select \* from student;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sno** | **Snm** | **M1** | **M2** | **M3** |
| **10** | **Rama** | **90** | **87** | **89** |
| **20** | **Raju** | **40** | **45** | **49** |
| **30** | **Amar** | **60** | **60** | **60** |
| **50** | **Madhu** | **56** | **67** | **89** |

**Exercise- 6**

**Develop a program that includes the features NESTED IF, CASE and CASE expression. The program can be extended using the NULLIF and COALESCE functions.**

**Aim: To learn about various selection statements.**

* **Selection:** The selection structure is executed based on the test condition. Based on the test condition sequence of statements may or may not be executed. The following are the selection statements;
  + **If-then**
  + **If-then-else**
  + **If-then-elsif**
  + **Case**

**Nullif:** this function takes two arguments, if both are same then it returns NULL otherwise it returns the value of first argument.

**Coalesce:** It returns first non-null expression from the list.

Ex: select coalesce (null,null,36,44) from dual; 🡪 37

**Ex: 1. Program to check whether the given number is positive or negative.**

**declare**

**a number;**

**b number;**

**begin**

**a:=&a;**

**b:=&b;**

**if nvl(nullif(a,b),0) = 0 then**

**dbms\_output.put\_line(‘Equal numbers’);**

**elsif a>b then**

**dbms\_output.put\_line(a || ' is big’);**

**else**

**dbms\_output.put\_line(b || ' is big);**

**end if;**

**end;**

**Output:**

**Enter value for a: 15**

**Enter value for b: 20**

**20 is big**

**Ex: 2. Program to display the grade text.**

**DECLARE**

**grade CHAR(1);**

**BEGIN**

**grade := '&gd';**

**CASE grade**

**WHEN 'A' THEN DBMS\_OUTPUT.PUT\_LINE('Excellent');**

**WHEN 'B' THEN DBMS\_OUTPUT.PUT\_LINE('Very Good');**

**WHEN 'C' THEN DBMS\_OUTPUT.PUT\_LINE('Good');**

**WHEN 'D' THEN DBMS\_OUTPUT.PUT\_LINE('Fair');**

**WHEN 'F' THEN DBMS\_OUTPUT.PUT\_LINE('Poor');**

**ELSE DBMS\_OUTPUT.PUT\_LINE('No such grade');**

**END CASE;**

**END;**

**Output:**

**Enter value for gd: B**

**Very Good**

**Exercise -7**

**Program development using WHILE LOOPS, numeric FOR LOOPS, nested loops using ERROR Handling, BUILT –IN Exceptions, USE defined Exceptions, RAISE- APPLICATION ERROR.**

* **Iteration:** In this type the sequence of statements are executed repeatedly as long as the condition is true. The following are the iteration statements;
  + **Loop**
  + **While loop**
  + **For loop**

**Ex: 1 Program to display the even numbers below n.**

**DECLARE**

**i number :=0;**

**n number;**

**BEGIN**

**N:=&n;**

**loop**

**dbms\_output.put\_line(i);**

**i:=i+2;**

**exit when i>n;**

**end loop;**

**END;**

**Output:**

**Enter value for n: 10**

**0**

**2**

**4**

**6**

**8**

**10**

**Ex: 2 Program to find factorial of given number.**

**DECLARE**

**n number;**

**i number;**

**BEGIN**

**n:=&n;**

**i:=1;**

**while (n>0)**

**loop**

**i:=i\*n;**

**n:=n-1;**

**end loop;**

**dbms\_output.put\_line(‘Factorial of ‘|| n || ‘ is ‘|| i);**

**END;**

**Output:**

**Enter value for n: 5**

**Factorial of 5 is 120**

**Ex: 3 Program to display multiplication table for n.**

**DECLARE**

**n number;**

**I number;**

**BEGIN**

**n:=&n;**

**for I in 1 .. 10**

**loop**

**dbms\_output.put\_line(n||’\*’||i||’=’||n\*i);**

**end loop;**

**END;**

**Output:**

**Enter value for n: 10**

**10\*1=10**

**10\*2=20**

**10\*3=30**

**10\*4=40**

**10\*5=50**

**10\*6=60**

**10\*7=70**

**10\*8=80**

**10\*9=90**

**10\*10=100**

**Ex 4: Write a Prg for Division (c=a/b) using error handling. The conditions are;**

* **B should not be zero**
* **B should not >a**

SQL> declare

a number;

b number;

c number;

e1 exception;

e2 exception;

begin

a:=&a;

b:=&b;

if b=0 then

raise e1;

elsif b>a then

raise e2;

else

c:=a/b;

dbms\_output.put\_line('Quotient =' || c);

end if;

exception

when e1 then

dbms\_output.put\_line('B should not be zero');

when e2 then

dbms\_output.put\_line('B should not be > a');

end;

**Output:**

Enter value for a: 10

old 8: a:=&a;

new 8: a:=10;

Enter value for b: 2

old 9: b:=&b;

new 9: b:=2;

Quotient =5

PL/SQL procedure successfully completed.

SQL> /

Enter value for a: 10

old 8: a:=&a;

new 8: a:=10;

Enter value for b: 0

old 9: b:=&b;

new 9: b:=0;

B should not be zero

PL/SQL procedure successfully completed.

SQL> /

Enter value for a: 10

old 8: a:=&a;

new 8: a:=10;

Enter value for b: 11

old 9: b:=&b;

new 9: b:=11;

B should not be > a

PL/SQL procedure successfully completed

**Exercise -8**

**Programs development using creation of procedures, passing parameters IN and OUT of PROCEDURES.**

**SQL –Routines** can be either **procedures** or **functions.**  A **Function** returns one value and has only input parameters. A **procedure** may have input parameters, output parameters, and parameters that are both input and output parameters.

**Syn:**

**{CREATE PROCEDURE | CREATE FUNCTION }** routine-name

**([parameter [{,parameter}…]]}**

**[RETURNS data\_type result\_cast]**

**routine\_body**

**Ex: Write a procedure to find out name of the given employee number;**

**SQL>Create or replace PROCEDURE findname(eno in number, enm out varchar2) IS**

**BEGIN**

**Select ename into enm from emp where empno=eno;**

**END;**

**/**

**Output:**

**Procedure created**

**Executing procedure:**

**SQL> Declare**

**N number;**

**M varchar2(20);**

**Begin**

**N:=&n;**

**Findname(n,m);**

**Dbms\_output.put\_line(‘The name is:’||m);**

**End;**

**Output:**

**Enter value for n:7788**

**The name is: Scott**

**Exercise -9**

**Program development using creation of stored functions, invoke functions in SQL Statements and write complex functions.**

**Ex: function to display the total bonus (10% of the salary) for all employees.**

**SQL >** create or replace function bonus return number is

b number;

begin

select (sum(sal)+sum(nvl(comm,0)))\*10/100 into b from emp;

return b;

end;

**Function Created**

**SQL>** declare

b number;

begin

b:=bonus();

dbms\_output.put\_line('Total Bonus =' || b);

end;

**Output:**

**Total Bonus = 3272.50**

**Exercise -10**

**Program development using creation of package specification, package bodies, private objects, package variables and cursors and calling stored packages.**

(1) create a table dept1

->create tabke dept1 (dname varchar2(10),deptno number);

->insert into dept values('accounting',10);

->insert into dept values('hr',20);

(2)create a table dept

->create table dept(dno number,vt varchar2(10),dloc varcar2(20));

(3)creating package header

create or replace package test

is

procedure savedept

(dno in number,dloc in varchar);

end;

/

(4)creating package body

create or replace package body test

is

function getdno(dno in number)

return varchar

is

dnum varchar(20);

begin

select dname into dnum from dept

where deptno=dno;

return dnum;

end;

procedure savedept

(dno in number,dloc in varchar)

is

vt varchar(20)

begin

vt:=getno(dno);

insert into dept values(dno,vt,dloc);

exception

when dup\_val\_on\_index then

raise\_application\_error(-2007,'duplicate');

end;

end;

/

(5)Executing procedure

exec test.savedept(10,'vijayawada');

(6)Display the table

->select \* from dept;

**Exercise- 11**

**Develop programs using features parameters in a CURSOR, FOR UPDATE CURSOR, WHERE CURRENT of clause and CURSOR variables.**

**Cursors:** Cursor is a work area in memory where Oracle stores the current SQL statement. Usage of only select statement did not place any locks on selected rows.

* Select … for update statement obtains an exclusive row-level lock on all the rows identified by the select statement. Select … for update of column statement obtains an exclusive colum-level lock on all the rows identified by the select statement.
* No one will be able to change any of these records until you perform a rollback or commit.
* Where current of clause allows to make the changes (update and delete) to the most recently fetched row of the data.
* There are two types of cursors available in PL/SQL.
  + Implicit Cursor, automatically created whenever a DML statement is executed. (Ex: %found, %count, etc)
  + Explicit cursors, are declared by the user by using OPEN, FETCH and CLOSE keywords or using for loop.

**Ex: Cursor to increase the salary by 10% those who are having above 2000 salary otherwise increase 5% salary.**

declare

cursor c1 is select \* from emp for update of sal;

sl number;

begin

select sum(Sal) into sl from emp;

dbms\_output.put\_line('Before update total salary is:' ||sl);

for z in c1

Loop

select sal into sl from emp where empno=z.empno;

If sl>2000 then

update emp set sal = sal+ (sal\*10/100) where empno=z.empno;

else

update emp set sal=sal+ (sal\*5/100) where empno=z.empno;

end if;

end loop;

select sum(Sal) into sl from emp;

dbms\_output.put\_line('After update total salary is:' ||sl);

end;

**Output:**

Before update total salary is: 42542.125

After update total salary is: 46258.86875

PL/SQL procedure successfully completed.

**Exercise- 12**

**Develop Programs using BEFORE and AFTER Triggers, Row and Statement Triggers and INSTEAD OF Triggers**

Triggers are the stored programs. Triggers are automatically executed whenever an event occurs.

**Levels of Triggers:**

* Row level trigger: It gets executed for each row got updated by a DML statement.
* Statement level trigger: It gets executed only once by the event statement.

Categorization of Trigger timing:

* Before: It gets executed prior to the specific event that has taken place.
* After: It gets executed post to the specific event that has taken place.
* Instead of: It is for creating a trigger that has a view.

Trigger Events:

* DML Trigger: It gets executed if a DML event like insert, update or delete is performed.
* DDL Trigger: It gets executed if a DDL event like create, alter or drop is performed.
* Database Trigger: It gets executed if a database event like shutdown, startup, logoff, and logon has taken place.

**Ex: Create a trigger to display a message after performing operations on emp table.**

**After inserting :New row inserted**

**After deleting : One row deleted**

**After updating : Salary difference is : <<difference amount>>**

SQL> create or replace trigger t1 after insert or update or delete on emp for each row

Declare

diff number;

begin

if inserting then

dbms\_output.put\_line('New row inserted');

elsif updating then

diff:=:new.sal-:old.sal;

dbms\_output.put\_line('Difference is:' ||diff);

else

dbms\_output.put\_line('One Row deleted');

end if;

end;

/

Trigger created.

Output:

SQL> insert into emp (empno, sal) values (77,5000);

New row inserted

1 row created.

SQL> update emp set sal = 7000 where empno=77;

Difference is :2000

1 row updated.

SQL> delete from emp where empno=77;

One Row deleted

1 row deleted.