**/ CONTENTS**

[INTRODUCTION **2**](#_heading=h.gjdgxs)

[BUILT-IN FUNCTIONS **8**](#_heading=h.1fob9te)

[JOINS, SET OPERATORS, NESTED QUERIES **11**](#_heading=h.3znysh7)

VIEW, SEQUENCE, SYNONYM, INDEX**……………….………………………. .13**

TCL COMMANDS**……………………………………………………………….14**

DCL COMMANDS**………………………………………………….. ………….15**

[EX 1. DDL COMMANDS](#_heading=h.2et92p0)**16**

[EX 2. DML COMMANDS **1**](#_heading=h.tyjcwt)**7**

[EX 3. BUILT-IN FUNCTIONS **18**](#_heading=h.3dy6vkm)

[EX 4. JOINS, SET OPERATORS, NESTED QUERIES **19**](#_heading=h.1t3h5sf)

EX 5. VIEW, SEQUENCE, SYNONYM, INDEX**..………………………………… 21**

EX 6. TCL COMMANDS**…………………………………………………………21**

EX 7. DCL COMMANDS**……………………………...…………………………21**

[PL/SQL - INTRODUCTION **22**](#_heading=h.2s8eyo1)

[CURSORS **25**](#_heading=h.17dp8vu)

[TRIGGERS **29**](#_heading=h.3rdcrjn)

EXCEPTION**……………………………………………………………………30**

[PROCEDURES AND FUNCTIONS **33**](#_heading=h.26in1rg)

[EX 8. SIMPLE PL/SQL PROGRAMS **36**](#_heading=h.lnxbz9)

[EX 9. PL/SQL PROGRAMS - CURSORS **37**](#_heading=h.35nkun2)

[EX 10. PL/SQL PROGRAMS - TRIGGERS **38**](#_heading=h.1ksv4uv)

EX 11.PL/SQL PROGRAMS-EXCEPTION**…………………………………….…..39**

[EX 12. PROCEDURES AND FUNCTIONS **39**](#_heading=h.44sinio)

[MINI PROJECT **40**](#_heading=h.2jxsxqh)

LIST OF PROJECTS**……………………………………………………………. 43**

PRACTICE EXERCISES**…………………………………………………………44**

**DBMS**

**INTRODUCTION**

A data base management system is a collection of programs that enables users to create and maintain a database. DBMS is a general purpose software system that facilitates the process of defining, constructing,manipulating databases for various applications.

## Functions of DBMS

Database Definition - how data is to be stored and organized

Database Creation - storing data in a defined database

Data Retrieval - Querying and reporting

Updating - changing the contents of the data base

Programming user facilities for system development.

Database revision and restructuring

Database integrity control.

Performance monitoring.

**RDBMS**

RDBMS is a database management system where all data visible to the user is organized strictly as tables of data values and all database operations work on these tables.

**SQL**

A standard query language that is commonly used with all relational databases for data manipulation and definition.

Based on the type of action that each command performs, SQL commands can be broadly classified as follows:

| **Classifications** | **Description** | **Commands** |
| --- | --- | --- |
| DDL(Data definition language) | Used to define the structure of a table, or modify the structure. | CREATE,ALTER,DROP,  TRUNCATE,RENAME |
| DML(Data manipulation language) | Used to manipulate with the data. | INSERT,UPDATE,DELETE |
| DCL(Data control language) | Used to restrict or grant access to tables | GRANT,REVOKE |
| TCL(Transaction control language) | Used to complete fully or undo the transactions | COMMIT,SAVEPOINT,  ROLLBACK |
| Queries | Used to select records from the tables or other objects. | SELECT |

**Data Types**

* Character
  + Char(n)
  + Varchar2(n)
* Numeric
  + Number(p,s)
* Date
  + Default format – DD-MON-YY
* Raw
* Long
* Long Raw
* Lob

**SYNTAX FOR SQL COMMANDS**

## 1. *To create a table*

**Create table** <table name>(columnname1 data type(size),………

Columnname N data type(size));

## 2. *To view table structure*

**Desc** <table name>;

## 3. *To insert data*

**a.** ***Adding a row to the table***

**Insert into** <table name> **values** (value1 for column1,value2 for column2,….Value n for column n);

### b. Inserting records through user interaction

**Insert into** <table name>**values**(‘&column name1’,’& column name2’,’&

column name3’,…………. ………..,’& column name4’);

## c. *Insert a partially filled record into a table*

**Insert into** <table name>(column1,column3,…… …..column n) **values**(value1 for column1,value3 for column3,…. … value n for

column n);

## 4. *To view the entire table*

**Select \* from** <table name>;

## 5. *To view selective columns of the corresponding table*

**Select** column name1, column name2 **from** <table name>;

**6. *To view existing tables of the user***

**Select \* from tab;**

**7. *To modify table structure***

### a. Adding a new column

**Alter table** <table name> **add** (new column name datatype(size));

#### b. Modifying existing column’s datatype or size

**Alter table** <table name> **modify**(existing column name data

type(size));

##### 8. *To edit data*

**Update** <tablename> **set** <column name1> = <value1>,<column name2> = <value2>,……. [**where** <condition>];

**9. *To delete a record***

**Delete** <table name> [**where <**condition>];

**10. *To create a duplicate table***

**Create table** <new table name> as <**Select** statement>

## 11. *To delete a table*

**Drop table** <table name>

**12. *To display Virtual or calculated column***

Columns with values that are calculated from the values of the existing column are known as **virtual columns.**

**Select**  column name1,column name4, [calculation on existing columns]

<alias name of virtual column> **from** <table name>;

## 13. *Conditional retrieval of data*

We can retrieve data that satisfies certain conditions. These conditions are framed by using the relational operators and logical operators. Besides these Oracle8 has its own operators such as Like, NULL, IN etc. classified as OTHERS which can be used in conditional retrieval of data.

**Relational operators** - = , <> , < , <= , > , >=

**=** operator is used to compare two values for equality

**<>** operator is used for inequality comparison

**Others** - in , between and , like% , like “\_” , is NULL

**in** - operator is used to retrieve data which matches a specific list of values

**between and** - operator is used when data needs to be retrieved for a specific

range of values

##### Like

Using simple = comparison test we are able to retrieve rows where the contents of a text column match some particular text. Suppose we may not be able to remember the entire pattern, if it is a lengthy one, Like operator can be used to retrieve rows which will match a specified pattern. The pattern is a string that may include one or more wildcard characters % or \_ .

###### Like %

When data needs to be retrieved which matches any sequence

of zero or more characters, wildcard % is used.

###### Like “\_”

To match a single character in a pattern, wild card “\_” is used.

**Is NULL**

Null value is an indicator that tells that the data is inapplicable,

unknown or missing.

**Logical operators** - and , or , not

Used to produce a single result by combining two separate conditions.

**And**

When you want the rows to be retrieved should satisfy both the

conditions then AND operator can be used.

###### Or

When you want the rows to be retrieved to satisfy either of

the conditions then OR operator can be used.

###### Not

Sometimes you need to negate a condition . For this NOT

operator can be used. Not can be used with all the operators

discussed so far, such as IS NOT NULL,NOT BETWEEN ….

And, NOT LIKE.

##### *Ordering data*

Data that is displayed using SELECT statement can be arranged in a particular Column’s order i.e. ascending or descending .By default columns are ordered in ascending order . To change the order of display in reverse direction (descending) “Desc” keyword is used in the order by.

**Select** <column list> **from** <table name> **order by** <column name> asc/desc

# *The Syntax for a simple query is*

**Select** <column list> **from** <table name> [**where** <condition>] [**group by**

<column name>] [**having** <condition>] [**order by** <expression>];

***14. To create constraints on the table***

Constraints can be defined at two levels- Column or table.

##### Column level constraints

**Create table** <table name>(column name1 data type(size) **constraint** <constraint name> **primary key**, column name2 data type(size) **constraint**  <constraint name> **references** referenced table name [(primary column name of referenced table)], column name3 data type(size) **constraint** <constraint name> **check**(<condition>), column name data type(size) **NOT NULL**);

##### Table level constraints

Those constraints which refer other columns of the table should be defined at the table level.

**Create table** <table name>

(column name1 data type(size),

……

…….

Column name N data type(size),

**Constraint** <constraint name> **primary key**(column name1),

**Constraint** <constraint name> **foreign key**(foreign column name) **references**

referenced table name[(primary column name of referenced table)],

**constraint** <constraint name> **check** (<condition>));

**BUILT-IN FUNCTIONS**

**Introduction**

Built-in functions are predefined functions that perform a specific task. Built-in functions based on the values that they take to perform a task, can be classified into two types. They are

1. Scalar or single row function

2. Aggregate or group function

**Scalar functions**

* Number functions
* Character functions
* Date functions
* Conversion functions
* Other functions

**Number functions**

* + ABS(n)
  + FLOOR(n)
  + CEIL(n)
  + EXP(n)
  + LN(n)
  + LOG(n)
  + MOD(n)
  + POWER(m,n)
* ROUND(n[,m])
* SIGN(n)
* SQRT(n)
* TRUNC(n[,m])

**Character Functions**

**a. returning number values**

* + - * ASCII(char)
      * INSTR(char1,char2[,n[,m]])
      * INSTRB(char1,char2[,n[,m]])
      * LENGTH(char)
      * LENGTHB(char)

**b. returning char value**

* + - * CHR(n)
      * CONCAT(char1,char2)
      * INITCAP(char)
      * LPAD(char1,n[,char2])
      * LTRIM(char[,set])
      * REPLACE(char,search-string[,replacement-string])
      * RPAD(char1,n[,char2])
      * RTRIM(char[,set])
      * SOUNDEX(char)
      * SUBSTR(char,m[,n])
      * SUBSTRB(char,m[,n])
      * TRANSLATE(char,from,to)
      * UPPER(char)
      * LOWER(char)

**Date functions**

* + - * ADD\_MONTHS(d,n)
      * Last\_day(d)
      * MONTHS\_BETWEEN(d1,d2)
      * NEXT\_DAY(d,char)
      * ROUND(d[,fmt])
      * TRUNC(d[,fmt])

**Date format elements**

| Element | Meaning |
| --- | --- |
| YYYY | 4 digit year |
| YY | Last 2 digits of year |
| MM | Month(01-12 ; Jan=01) |
| MONTH | Name of month, |
| MON | Abbreviated name of month |
| DDD | Day of year(1-366) |
| DD | Day of month(1-31) |
| D | Day of week(1-7) |
| DAY | Name of day |
| DY | Abbreviated name of day |
| HH or HH12 | Hour of day(1-12) |
| HH24 | Hour of day(0-23) |
| MI | Minute(0-59) |
| SS | Second(0-59) |

**Conversion Functions**

* + - * TO\_CHAR(d[,fmt])
      * TO\_CHAR(n[,fmt])
      * TO\_DATE(char[,fmt])
      * TO\_NUMBER(char[,fmt[,'nlsparams']])

**Other functions**

* + - * greatest(expr[,expr])
      * least(expr[,expr])
      * Nvl(expr1,expr2)
      * UID(user)
      * Sysdate

**Aggregate functions**

* + - * AVG()
      * MAX()
      * MIN()
      * COUNT()
      * SUM()

**JOINS, SET OPERATORS, NESTED QUERIES**

**JOINS**

If any information needs to be queried from more than one table, a concept called joins is used. This allows data to be selected from one or more tables and combine the selected data into a single result table.

Joins can be of different types

* Equi-join
* Self join
* Outer join
* Non-equi join

**Equi-join**

This is the most common type of joins. This join contains a condition containing an equality operator. An equi-join combines rows that have equivalent values for the columns specified in the join.

**Self join**

Self joins are a type of joins where the join operation takes place within the same table. To differentiate the two tables, alias name for the table is used.

**Outer join**

The Outer join extends the result of a simple join or equi-join. Outer join returns all the rows that satisfy the join condition and those rows from one table for which no rows from the other table satisfy the join condition.

**Non-Equi join**

Non-equi joins specify the relationship between the tables not in terms of columns but in the terms of relational operators or any comparison operators used.

**SET OPERATORS**

SET operators combine the results of two component queries into a single result. Queries containing set operators are called Compound queries.The set operators are

* Union
* Union all
* Intersect
* Minus

**Union**

Union operation performed on two tables yields records from both the tables without the values being repeated.

**Union All**

Union operation performed on two tables yields records from both the tables with duplication.

**Intersect**

Intersect operation performed on two tables yields records that are common and distinct.

**Minus**

All the distinct rows selected by the first query but which are not in the second query are listed.

**SUB QUERIES**

A sub query is a query that can contain multiple query statements each nested within another. It is also called as a nested query. Statement containing a sub query is called the parent statement and the query inside is called the child query. Parent statement uses the rows returned by the child query.

If the inner query returns only one value the outer query can process and display values that match the condition. If more than one value is selected, Oracle8 requires an operator that can check and display the values. To do this operation the following operators are used.

* Any
* All
* Some
* Exists

**VIEW, SEQUENCE, SYNONYMN AND INDEX**

**View:**

A **view** is a subset of a database that is generated from a query and stored as a permanent object. Although the **definition** of a **view** is permanent, the data contained therein is dynamic depending on the point in time at which the **view** is accessed. **Views** represent a subset of the data contained in a table.

Syntax: create view view-name as select column-name from table-name

**Sequence:**

A sequence is database object basically used for generating primary key number and many users can share the sequence. A sequence generates a list of unique numbers that identify the rows and columns of a table.

Syntax: create sequence seq-name start with 1 increment by 1 maxvalues 999

**Synonym:**

A **synonym** is an alias or alternate name for a [table](https://en.wikipedia.org/wiki/Table_(database)), [view](https://en.wikipedia.org/wiki/View_(database)), [sequence](https://en.wikipedia.org/wiki/Sequence) or other schema object. They are used mainly to make it easy for users to access database objects owned by other users. They hide the underlying object's identity and make it harder for a malicious program or user to target the underlying object. Because a synonym is just an alternate name for an object, it requires no storage other than its definition.

Syntax: create synonym synonym-name for table-name

**Index:**

Index is a copy of selected columns of data from a table that can be searched very efficiently. An index can be created in a table to find data more quickly and efficiently. The users cannot see the indices, they are just used to speedup searches/queries.

Syntax: create index index-name on table-name(column-name);

**TCL COMMANDS**

**TCL commands:**

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

* Commit

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

Syntax: commit;

* Rollback

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

Syntax: rollback to savepoint-name;

* Savepoint

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

Syntax: savepoint savepoint-name;

**DCL COMMANDS**

**DCL commands:**

A Data Control Language (DCL) is a syntax similar to a computer programming language used to control access to data stored in a database. The Data Control Language(DCL) component of the SQL language is used to create privileges to allow users access to, and manipulation of, the database.

There are some main commands in DCL and TCL. They are,

* Grant

Gives user access privileges to database.

#### Syntax:

#### To Allow a User to create Session

grant create session to username;

* To allow user to create table

grant create table to username;

* To grant all privilege to a user

grant sysdba to username;

* To grant permission to create any table

grant create any table to username;

* To grant permission to drop any table

grant drop any table to username;

* Revoke

Take back permissions from user.

#### Syntax:

#### To take back Permissions

revoke create table from username;

**EX 1. DDL COMMANDS**

1. Create a table **‘Department’** with the followingstructure (**Column level constraints**)

Depno Number(2) Primary key

Depname Varchar2(15)

Deplocation Varchar2(10)

1. Create a table **‘Employee’** with the following details(**Column level constraints**)

Empno Number(5) Primary key

Empname Varchar2(20)

Designation Varchar2(10)

Date\_of\_join Date NOT NULL

Salary Number(9,2)

Depno Number(2) Foreign key(Reference **‘Department’** table)

1. Create a table **‘Course’** with the following details(**Table level constraints**)

Coursecode Number(2) Primary key

Coursename Varchar2(15)

1. Create a table **‘Student’** with the following details(**Table level constraints**)

Rollno Number(5) Primary key

Name Varchar2(15)

Coursecode Number(2) Foreign key(Reference ‘**Course’** table)

Mark1 Number(3) Check constraint value between 0&100

Mark2 Number(3) Check constraint value between 0&100

1. Display the structure of the tables **‘Department’, ‘Employee’, ’Course’, ’Student’**
2. Add the column ‘Phoneno’ number (10), ’Grade’ varchar2 (1) in the **‘Employee’** table
3. Change the column size of ‘Deplocation’ column in the **‘Department’** table from 10 to 15.
4. Change the data type of the column ‘**Rollno**’ from numeric to character.
5. Drop the column ‘**Mark2**’ in **student** table.
6. Display the names of the table created by the current user
7. Delete the contents of the table **‘Student’**.
8. Delete the table **‘Student’**
9. Rename the table **‘Employee’** as **’Emp’**.
10. Create not null constraint on ‘**Name**’ column of the table ‘**Student**’
11. Create default constraint on ‘**Work\_location**’ column of the table ‘**Employee**’

**EX 2. DML COMMANDS**

1. Insert into the table **‘Department’** the following values

| Depno | Depname | Deplocation |
| --- | --- | --- |
| 01 | SALES | Blocka |
| 02 | PURCHASE | Blockd |
| 03 | PRODUCTION | Blockb |
| 04 | MARKETING | Blocka |
| 05 | ACCOUNTS | Blockc |
| 06 | SOFTWARE | Blocke |

1. Insert into the table **‘Employee’** the following values

| Empno | Empname | Designation | Date\_of\_join | Salary | Depno | Grade |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Siva | manager | 05-Oct-1987 | 15000.00 | 05 | A |
| 2 | Mani | salesman | 12-Apr-1987 | 5000.75 | 01 | F |
| 3 | Raju | Clerk | 30-Nov-1989 | 7000.00 | 02 | E |
| 4 | Babu | Clerk | 04-Jan-1995 | 5000.50 | 03 | E |
| 5 | Ram | salesman | 08-Dec-2000 | 3000.25 | 01 | F |
| 6 | Velu | programmer | 24-Feb-2002 | 10000.50 | 06 | D |
| 7 | Ravi | accountant | 12-Sep-1991 | 8000.25 | 05 | G |
| 8 | Balan | manager | 07-Jun-1993 | 12000.75 | 03 | A |
| 9 | Mahesh | Officer | 18-Mar-1997 | 10000.50 | 02 | B |
| 10 | Kumar | Analyst | 15-Jan-1995 | 14500.00 | 06 | C |

1. Display all the columns of the table **‘Employee’, ‘Department’**
2. Insert values for the column ‘Phoneno’ in the **‘Employee’** table and Display empname , Phoneno for all the employees.(Use “Update” command)
3. Create a duplicate employee table called **‘Emp’**
4. Display Empname, Designation, Years-of-service(Virtual column or calculated column) for all the employees. Calculate Years-of-service by subtracting date-of-join from current date.
5. Create a view with the fields Empname , Designation, Depname
6. List the employees whose salary is equal to Rs.10,000.50(Use ‘=’ operator)
7. List the employees whose depno is not equal to ‘01’(Use ‘<>’ operator)
8. List the employees who have joined after ’04-Apr-1995’ (Use ‘>’ operator)
9. List the employees who belongs to the department 2, 3, 5(Use ‘IN’ operator)
10. List the employees who have joined after ’01-Jan-1988’ and before ’01-Jan-1998’(Use ‘Between and’ operator)
11. List the employees whose name starts with the letter ‘R’(Use ‘LIKE’ operator)
12. List the employees who don’t have phone numbers(Use ‘IS NULL’ operator)
13. List the employees who belong to department ‘01’ .Also salary equal to 5000.75(Use ‘AND’ operator)
14. List the employees either belongs to department ‘01’ or salary equal to 5000.50(Use ‘OR’ operator)
15. List the employees whose designation is other than ‘Manager’, ‘Officer’(Use ‘NOT’ operator)
16. List the department details in the descending order of department name.
17. Delete the records of the employees who have joined before ’01-Jan-1989’.

**EX 3. BUILT-IN FUNCTIONS**

**String Functions**

1. Find the length of the column ‘Designation’
2. Display the first three characters of Department name
3. Change all ‘BlockA’ in Deplocation column as ‘BlockF’
4. Remove the ‘0’ s in the strings ‘ABC0000’, ‘000ABC’
5. Display the distinct Designations in title case
6. Display the Depname in lower case
7. Display the Deplocation in upper case
8. Translate Grade to the corresponding designation

**Numeric Functions**

1. Find the total number of employees.
2. Find the employee with maximum salary, minimum salary in each department in the ascending order of depno.
3. Find the total salary paid to the employees.
4. Find the average salary paid to the employees.
5. Display the rounded value of the ‘salary’ column in **‘Employee’** table.
6. Display the salary such that it contains no decimal places.

**Date Functions**

15.Display the system date in the format mentioned below “27th October 1996”.

16.Display “20th September 1996” in the date format.

17.Display the date two months after the date-of-join for all the employees.

18.Display the last date of the month in the date-of-join for all the employees.

19.Display the months between the current date and the date-of-join.

20.Display the next occurrence of ‘Friday’ to the current date.

21.Display the first day of the year 2000.

\* Use **‘Employee’** and **‘Department’** tables .

**EX 4. JOINS, SET OPERATORS, NESTED QUERIES**

**Joins**

1. List the Employee names and their department names.

(Use Equi-Join)

1. List the Empolyees who earn more than Raju. (Use Self-Join)
2. List the Employee names and their department names.(Use Outer join)
3. List the employees who belongs to the department “Production”,”Sales”,”Software”.

(Use Non-Equi join)

\* Use **‘Employee’** and **‘Department’** tables .

# Set Operators

1. List the products available in both the tables without duplication.
2. List the products available in both the tables with duplication.
3. List the products common to both the tables .
4. List the products available in ‘Product 1’ table but not available in ‘Product 2’ table.

\* Use the following tables

**Product 1**  **Product 2**

Item code Varchar2(5) Item code Varchar2(5)

Item name Varchar2(10) Item name Varchar2(10)

**Sub Queries**

1. Find the employee with the maximum salary.
2. List the employees having salary greater than the lowest of the average salary of departments.(Use ANY operator)
3. List the employees having salary lower than the greatest of the average salary of all departments.(Use ALL operator).
4. List the departments where there are employees functioning. (Use EXISTS operator)
5. List the details of the employees belonging to ‘Software’ department.
6. Display designation, department number, total number of employees designation wise and department wise and the number of employees in each department and for each job must greater than 2 (use group by having)

**EX 5 . VIEW, SEQUENCE, SYNONYMN AND INDEX**

1. Create a view ‘emp desn’ for employee table with the column empname,designation,dname.
2. Display empname,designation and dname for all the employee using view.
3. Create a sequence for the column empno on ‘employee’ table.
4. Insert values for the column empno using sequence.
5. Create alias name for employee table called s-emp
6. Display all the columns of table employee using synonymn.
7. Update values for the column phoneno in the employee table using synonymn.
8. Create index on employee table named ‘isalary’ with field empno and salary.

**EX 6. TCL COMMANDS**

1. Write a query to implement save point named si.
2. Delete the records of table employee whose name starts with letter “R”.
3. Display all the records of table employee.
4. Write a query to implement savepoint named s2.
5. Delete the records of employee whose name starts with letter ‘s’.
6. Display all the records of table ‘employee’
7. Write a query to implement rollback to savepoint s2.
8. Display all the records of table employee.
9. Write a query to implement commit.
10. Create two users with password.
11. Create session for the users.
12. Create a table in one user and insert some records.

**EX 7. DCL COMMANDS**

1. Provide various privileges for other user on that table and check it.
2. Take back the privileges from the other user and check it.

**PL/SQL - INTRODUCTION**

Oracle's PL/SQL is a procedural language extension of SQL.PL/SQL is a block-structured language. The basic unit of PL/SQL is called a BLOCK, which contains declarative statements and error handling statements. These blocks can be nested into one or more blocks.

**ADVANTAGES OF PL/SQL**

* Sql support
* Better performance
* Support for object oriented programming
* Portability
* Higher productivity
* Integration with oracle

**DATA TYPES**

Every constants and variable has a datatype that specifies a storage format and valid range of values. Apart from the data types that are available in SQL,PL/SQL has its own set of datatypes that can be used in PL/SQL blocks.

**Numbered datatypes**

For numeric data and to represent quantities, calculations can be performed on this datatype .

**Binary\_integer**

For storing signed integers, its magnitute ranges from -2147483647 to 2147483647.It has various subtypes:

* NATURAL
* NATURALN
* POSITIVE
* POSITIVEN
* SIGNTYPE

**Collection datatypes**

A collection is an ordered group of elements ,all of the same type. Each element has a unique subscript that determines the position of the collection.

**Boolean datatypes**

Logical values TRUE, FALSE or NULL can be stored using BOOLEAN datatypes. The datatype has no parameters.

**Exception datatypes**

This is a datatype that is used to define exception or error handlers, which is defined by the user.

**A Simple Program**

DECLARE

X NUMBER;

BEGIN

X:=26;

DBMS\_OUTPUT.PUT\_LINE(X);

END;

The above program displays the value 26 which is stored in the variable X.

**Using the SELECT statement inside the PL/SQL**

The SELECT statement selects a set of records from the database. To display a condition based record, the SELECT statements requires an INTO clause. A SELECT statement must hold an INTO clause where the output of the query is assigned to the variable given in the INTO clause.

DECLARE

s NUMBER;

BEGIN

SELECT sal INTO s FROM emp WHERE empno=7369;

DBMS\_OUTPUT.PUT\_LINE(!the salary is !||s);

END;

The variable s is assigned the salary of the employee number 7369 and the result would be displayed.

**DECLARING VARIABLES**

Variables can be declared in various ways.They are:

**1)Using DEFAULT:**

DECLARE

s NUMBER DEFAULT 10;

BEGIN

DBMS\_OUTPUT.PUT\_LINE(s);

END;

This program displays the default value of s i.e 10 is printed.

**2)Using CONSTANT:**

Pi CONSTANT REAL:=3.14;

**3)Using NOTNULL:**

s VARCHAR(10) NOTNULL:='RADIANT';

**CONTROL STRUCTURE**

There are three forms of IF statements:

IF...THEN

IF...THEN...ELSE

IF...THEN...ELSEIF

**ITERATIVE CONTROL**

Iterative statements are the statements that are performed a number of times depending o the value in the LOOP statement. There are three various forms of LOOP statements:

1)LOOP

Sequence\_of\_statements;

EndLOOP

2)While condition loop

Sequence\_of\_statements;

ENDLOOP;

3)FOR counter IN [REVERSE] lower\_ bound.. higher\_ bound LOOP

sequence\_ of\_ statements;

ENDLOOP;

4)BEGIN

....

EXIT WHEN<CONDITION>;

In the LOOP statement, each loop can be given different names:

<<label\_ name>>

LOOP

Sequence\_ of\_ statements;

ENDLOOP;

**CURSORS**

Oracle uses work areas to execute SQL statements and store processing results. A PL/SQL construct called Cursor allows a work area to be named, to store and access its information. They are typically used in areas where the query inside the PL/SQL block retrieves more than one record. A cursor is a pointer to handle the context area or the memory area or work area. The result set of information is called Result Set.

When a query inside a PL/SQL block returns more than one record or one set of data, Oracle requires a placeholder to place the values. The variables provided in the INTO clause can contain only one value at a time. In order to process multiple records, CURSORS are used.

**Types of Cursors**

PL/SQL implicitly declares a cursor for all SQL data manipulation statements, including queries that return only one row. Cursors can be of 4 types.

* Implicit Cursors
* Explicit Cursors

**Implicit Cursors**

When the executable part of PL/SQL blocks are known as implicit cursor, which has the identifier SQL.

**Explicit Cursors**

Explicit cursors has two types. They are

* Static Cursors
* Dynamic Cursors

Static Cursors:

It is a type of cursor where the SELECT statement is given at compile time itself. That is the table from which the data are coming and the records that are going to be selected are predetermined at compile time itself.

Dynamic Cursors:

It is a set of cursors where the records from the tables are selected at run time rather than at compile time.

Explicit cursor manipulation is performed using four commands.

* DECLARE
* OPEN
* FETCH
* CLOSE

**ATTRIBUTES OF CURSORS:**

| **Attribute** | **Description** |
| --- | --- |
| %FOUND | Returns TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows or a SELECT INTO statement returned one or more rows. Otherwise, it returns FALSE. |
| %NOTFOUND | The logical opposite of %FOUND. It returns TRUE if an INSERT, UPDATE, or DELETE statement affected no rows, or a SELECT INTO statement returned no rows. Otherwise, it returns FALSE. |
| %ISOPEN | Always returns FALSE for implicit cursors, because Oracle closes the SQL cursor automatically after executing its associated SQL statement. |
| %ROWCOUNT | Returns the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement. |

**Declaring a Cursor**

Syntax:

CURSOR <cursor name> IS <query>

Example

CURSOR C1 is SELECT ename, job FROM emp ;

**Opening a Cursor**

Syntax:

OPEN <cursor name>

Example

OPEN C1;

**Fetching Records from the cursor**

Syntax:

FETCH <cursor name> INTO <variable list>

Example

FETCH C1 INTO x, y ;

**Closing a Cursor**

CLOSE statement releases the data within the cursor and closes it. The cursor is reopened to refresh its data.

Example

This example is used to select all the employees names and their jobs from the table.

DECLARE

CURSOR c1 IS SELECT ename, job FROM emp;

Mname VARCHAR2 (20) ;

Mjob VARCHAR2 (20) ;

BEGIN

OPEN c1;

LOOP

FETCH c1 INTO mname, mjob;

DBMS\_OUTPUT.PUT\_LINE(mname || ` is a ` || mjob);

END LOOP;

CLOSE c1;

END;

The output of this program is,

KING is a PRESIDENT.

**TRIGGERS**

A database trigger is a stored PL/SQL block that is associated with a table. Triggers are automatically executed when a specified SQL statement is issued against the table. Triggers are mainly used for the following purposes:

* + - * To automatically generate values.
      * To provide auditing.
      * To prevent invalid transaction.
      * To check for complex integrity constraints that cannot be given in constraints.
      * To maintain replicate tables.

Trigger contain PL/SQL constructs and they can be fired only for DML statements like INSERT, UPDATE and DELETE. Triggers are event based.

**Parts Of Trigger**

A Trigger contains three parts:

* + - * Triggering event or statement
      * Trigger restriction
      * Trigger action

**Types Of Trigger**

Triggers are classified into different types depending on when the trigger is to be fired.

They are:

* + - * BEFORE
      * AFTER
      * INSTEAD OF

**Creating Triggers**

Triggers are created using CREATE TRGGER statement. The name of the trigger must be unique with respect to other triggers in the same schema. The syntax for creating the trigger is:

Syntax:

CREATE OR REPLACE TRIGGER <trigger name> [BEFORE/AFTER]

[INSERT/UPDATE/DELETE] ON <table name> [FOR EACH ROW][WHEN <condition>]

Example:

CREATE OR REPLACE TRIGGER instremp BEFORE INSERT ON emp

BEGIN

DBMS\_OUTPUT.PUT\_LINE(`inserting record`);

END;

The trigger gets created. When an insert statement is issued,the output will look like:

INSERT INTO emp (empno, ename) VALUES (1,’RAM’);

This would display, Inserting Record.

1 row created.

**EXCEPTION**

An error condition during a program execution is called an exception in PL/SQL. PL/SQL supports programmers to catch such conditions using EXCEPTIONblock in the program and an appropriate action is taken against the error condition. An exception must be defined before it can be raised.

Structure of exception:

DECLARE

Declaration section

BEGIN

Execution section

EXCEPTION

Exception section

END;

Syntax: EXCEPTION\_NAME EXCEPTION;

**Types of exception**

* Pre-defined exceptions
* Non pre-defined exceptions
* User-defined exceptions

## Pre-defined Exceptions

Predefined exceptions are defined by the system and we have some predefined names to handle these exceptions. PL/SQL provides many pre-defined exceptions, which are executed when any database rule is violated by a program. For example, the predefined exception NO\_DATA\_FOUND is raised when a SELECT INTO statement returns no rows.

| **Exception** | **Description** |
| --- | --- |
| ACCESS\_INTO\_NULL | It is raised when a null object is automatically assigned a value. |
| CASE\_NOT\_FOUND | It is raised when none of the choices in the WHEN clauses of a CASE statement is selected, and there is no ELSE clause. |
| COLLECTION\_IS\_NULL | It is raised when a program attempts to apply collection methods other than EXISTS to an uninitialized nested table or varray, or the program attempts to assign values to the elements of an uninitialized nested table or varray. |
| DUP\_VAL\_ON\_INDEX | It is raised when duplicate values are attempted to be stored in a column with unique index. |
| INVALID\_CURSOR | It is raised when attempts are made to make a cursor operation that is not allowed, such as closing an unopened cursor. |
| INVALID\_NUMBER | It is raised when the conversion of a character string into a number fails because the string does not represent a valid number. |
| LOGIN\_DENIED | It is raised when s program attempts to log on to the database with an invalid username or password. |
| NO\_DATA\_FOUND | It is raised when a SELECT INTO statement returns no rows. |
| NOT\_LOGGED\_ON | It is raised when a database call is issued without being connected to the database. |
| PROGRAM\_ERROR | It is raised when PL/SQL has an internal problem. |
| ROWTYPE\_MISMATCH | It is raised when a cursor fetches value in a variable having incompatible data type. |
| SELF\_IS\_NULL | It is raised when a member method is invoked, but the instance of the object type was not initialized. |
| STORAGE\_ERROR | It is raised when PL/SQL ran out of memory or memory was corrupted. |
| TOO\_MANY\_ROWS | It is raised when s SELECT INTO statement returns more than one row. |
| VALUE\_ERROR | It is raised when an arithmetic, conversion, truncation, or size-constraint error occurs. |
| ZERO\_DIVIDE | It is raised when an attempt is made to divide a number by zero. |

**Non Predefined Exceptions**

Non-predefined exceptions are also raised by the system only but we don’t have any predefined names we have to trap the exceptions with error code defined by the system. These Exceptions have a code and an associated message.

**User Defined Exceptions**

User defined exceptions are defined by the user. He has to declare the exception and he has to raise the exception.

**PROCEDURES AND FUNCTIONS**

A procedure or function is a named PL/SQL block - they are normally stored in the database within package specifications (which is a wrapper for a group of named blocks) but they may be stored on the database individually. The advantage of this is that when a block is placed on the database it is parsed at the time it is stored. When it is subsequently executed Oracle already has the block compiled and it is therefore much faster. It is also a good way of grouping application functionality together and exposing only function calls (not the code itself). It is possible to invoke a stored procedure or function from most Oracle tools including SQL \*Plus. It is also possible to attach a block to a database or Form trigger.

Benefits of using procedures and functions include :-

1. Improved data security and integrity.
   * Control indirect access to objects from non privileged users with security privileges.
   * Ensure that related actions are performed together, or not at all, by funneling actions for related tables through a single path.
2. Improved performance.
   * Avoid reparsing for multiple users by exploiting shared SQL.
   * Avoid PL/SQL parsing at run time by parsing at compile time.
   * Reduce the number of calls to the database and decrease network traffic by bundling commands.
3. Improved maintenance.
   * Modify routines online without interfering with other users.
   * Modify one routine to affect multiple applications.
   * Modify one routine to eliminate duplicate testing.

Procedures are simply a named PL/SQL block, they are created by a particular schema and like other database objects are then owned by that schema. Rights to the procedure are granted / revoked just like any other object except that you grant / revoke the EXECUTE right and not SELECT, UPDATE etc.

An example of an unnamed block is given below :-

DECLARE

TEMP\_COST NUMBER(10,2);

BEGIN

SELECT COST INTO TEMP\_COST FROM BOOK WHERE ISBN = 21;

IF TEMP\_COST > 0 THEN

UPDATE BOOK SET COST = (TEMP\_COST\*1.175) WHERE ISBN = 21;

ELSE

UPDATE BOOK SET COST = 21.32 WHERE ISBN = 21;

END IF;

COMMIT;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

INSERT INTO ERRORS (CODE, MESSAGE) VALUES(99, ‘ISBN 21 NOT FOUND’);

END;

This block could be saved to a command file and executed from there but it would be parsed each and every time it was called. To save this block to the database we need to name it.

CREATE OR REPLACE PROCEDURE MYPROC1

(TEMP\_COST NUMBER(10,2))

IS BEGIN

SELECT COST INTO TEMP\_COST FROM BOOK WHERE ISBN = 21;

IF TEMP\_COST > 0 THEN

UPDATE BOOK SET COST = (TEMP\_COST\*1.175) WHERE ISBN = 21;

ELSE

UPDATE BOOK SET COST = 21.32 WHERE ISBN = 21;

END IF;

COMMIT;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

INSERT INTO ERRORS (CODE, MESSAGE) VALUES(99, ‘ISBN 21 NOT FOUND’);

END MYPROC1;

The procedure (named MYPROC1) is defined in the first line. CREATE OR REPLACE asks Oracle to create a new procedure or if a procedure with this name already exists in this schema to replace it (you may leave out the OR REPLACE if you don't want this to happen).

The DECLARE section is now implicitly the section between the procedure definition and the IS BEGIN statement - it is not explicitly named but is enclosed within brackets. The END statement now ends a named block rather than an unnamed block. Nothing else is required for a procedure to run. To execute the stored procedure simply call it by name.

To get values into and out of the procedure, we can define variables in the implicit declare section as IN, OUT or IN OUT.

CREATE OR REPLACE PROCEDURE MYPROC1

(REQISBN IN NUMBER,

MYVAR1 IN OUT CHAR,

TCOST OUT NUMBER)

TEMP\_COST NUMBER(10,2))

IS BEGIN

SELECT COST INTO TEMP\_COST FROM BOOK WHERE ISBN = REQISBN;

IF TEMP\_COST > 0 THEN

UPDATE BOOK SET COST = (TEMP\_COST\*1.175) WHERE ISBN = REQISBN;

ELSE

UPDATE BOOK SET COST = 21.32 WHERE ISBN = REQISBN;

END IF;

TCOST := TEMP\_COST;

COMMIT;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

INSERT INTO ERRORS (CODE, MESSAGE) VALUES(99, ‘ISBN NOT FOUND’);

END MYPROC1;

EXECUTE MYPROC1 (21, :PCOST)- will execute the procedure in SQL \*PLUS (PCOST is the name of a SQL \*PLUS bind variable). 21 will be passed into the procedure, the value of TCOST will be passed out of the procedure and stored in :PCOST**.**

An IN OUT variable can pass values both in and out of a procedure.

A procedure can read in a list of values but won't directly return any value (it may indirectly return values). A function can also read in a list of values but will explicitly return a single result which is normally assigned to a variable or used in a program control structure condition test (IF statements etc).

The function below demonstrates the syntax of a PL/SQL function block, note that as with a procedure the OR REPLACE clause can be left out if you don't want the replacement of an existing function with the same name. Function definitions vary from procedure definitions in that you must explicitly name a variable to return and you must return a value in the variable via the RETURN statement.

CREATE OR REPLACE FUNCTION MYFUNC1

RETURN NUMBER

IS RCOST NUMBER(10,2);

BEGIN

SELECT COST INTO RCOST FROM BOOK WHERE ISBN = 21;

RETURN (RCOST);

END MYFUNC1;

EXECUTE :PCOST := MYFUNC1- will execute the procedure in SQL \*PLUS (PCOST is the name of a SQL \*PLUS bind variable). The purchase value of book 21 will be passed out of the function and stored in :PCOST.

**EX 8. SIMPLE PL/SQL PROGRAMS**

1. Write a PL/SQL **block** to find whether the given number is odd or even
2. Write a PL/SQL **block** to generate the fibonacci series
3. Write a PL/SQL **block** to find the sum of N natural numbers
4. Write a PL/SQL **block** to find the factorial of a given number
5. Write a PL/SQL block to find whether a year is leap year or not
6. Write a PL/SQL **block** to generate the armstrong numbers
7. Write a PL/SQL **block** to check the text is palindrome or not.





1. Write a PL/SQL **block** to display a particular item details. Display appropriate message if the item is not found.

**EX 9. PL/SQL PROGRAMS - CURSORS**

1. Write a PL/SQL **block** using **cursor** to display empname, designation and department name
2. Write a PL/SQL **block** using **cursor** to display empname, designation and department location (Use for loop)
3. Write a PL/SQL **block** using **cursor** to display depno, location, depname, deploc salary(Use %rowtype)
4. Write a PL/SQL **block** using **cursor** to display the total number of records, first, second, last, last but previous records and delete a record
5. Write a PL/SQL **block** using **cursor** to update the employee table. Increment salary of manager by 500 and salary of clerk by 200
6. Write a PL/SQL **block** using **cursor** to display the contents of various tables based on the options(Using Constrained variables)
7. Write a PL/SQL **block** using **cursor** to display the contents of various tables based on the options(Using UnConstrained variables)





1. Write a PL/SQL **block** using **cursor** to display the order details
2. Write a PL/SQL **block** using **cursor** to update the order status as ‘p’ if the order date is less than the delivery date

**EX 10. PL/SQL PROGRAMS - TRIGGERS**

1. Write a database **trigger** before insert or update for each row on the table employee not allowing to enter salary > 10,000

1. Write a database **trigger** before delete for each row on table department, delete corresponding department employees from employee table
2. Write a database **trigger** to display the total number of records after inserting a record into the table
3. Write a database **trigger** to check for the availability of department number in the department table, before inserting or updating a record in the employee table.
4. Write a database **trigger** to check for the job equal to manager and the corresponding salary between 4000 and 6000 after inserting into the table with allowing the record to be inserted.
5. Write a database **trigger** to check for the job equal to manager and the corresponding salary between 4000 and 6000 after inserting into the table without allowing the record to be inserted.
6. Deactivate a trigger and then activate it again.
7. Delete a trigger.
8. Disable all triggers in a particular table and then enable them again.
9. Create a trigger on table employ for not allowing insert/update/delete operations on Friday.
10. Write a trigger to ensure that empno in employee table is maintained as primary key.
11. Write a trigger to ensure that no DML operatios are allowed on employee table.

**EX 11. PL/SQL PROGRAMS- EXCEPTION**

1. Write a PL/SQL to handle ‘no data found’ predefined exception.
2. Write a PL/SQL to handle ‘zero divide’ predefined exception.
3. Write a PL/SQL to handle ‘duplicate value on index’ predefined exception.
4. Write a PL/SQL to handle user defined exception.

**EX 12. PROCEDURES AND FUNCTIONS**

1. Write a PL/SQL **procedure** using **cursor** to display the employee name, salary, depno for a given depno
2. Write a PL/SQL **procedure** which accepts empno as input and displays the employee name
3. Write a PL/SQL **procedure** which accepts empno as input and displays the employee name, salary
4. Write a PL/SQL **procedure** which accepts vendor code as input and displays the vendor name
5. Write a PL/SQL **procedure** to update the total field by adding the 3 marks
6. Write a PL/SQL **procedure** to display the details of the employees from employee table whose designation is ‘manager’
7. Write a PL/SQL **procedure** which accepts depno as input and increase the salary of the corresponding department employees by 10% in the employee table
8. Write a PL/SQL **procedure** to find the grade of the employees according to their salary

Salary > 20,000 Grade A

Salary > 15,000 Grade B

Salary > 10,000 Grade C

Salary < 10,000 Grade D

1. Write a PL/SQL **function**  to count the number of employees in a particular department
2. Write a PL/SQL **function** which accepts empno as input and return the salary as output
3. Write a PL/SQL **function** which accepts depno as input and return the department name as output
4. Write a PL/SQL **function** to calculate the total salary for a given department
5. Write a PL/SQL **function** to display the Fibonacci series

**MINI PROJECT**

**Back End :**

Back end is used to store the data.

The following are the examples of Back ends :

1. Oracle8
2. Ms Access
3. SQL Server

**Front End :**

Front end provide user friendly interface to accept input commands and data

and to display the results in the required Format.

The following are the examples of Front ends :

* + - 1. Power Builder
      2. Visual Basic6.0
      3. Developer 2000

**VISUAL BASIC6.0**

**Visual Basic6.0** is a front end tool that allows you to develop Windows

(Graphical User Interface - **GUI**) applications.

**Visual Basic6.0** is **event-driven**, means that code remains idle until called

upon to respond to some event (button pressing, menu selection, ...). Visual Basic6.0 is governed by an event processor. Nothing happens until an event is detected. Once an event is detected, the code corresponding to that event (event procedure) is executed. Program control is then returned to the event processor.

**ODBC : (Open DataBase Connectivity)**

ODBC Interface allows front end applications to access the data from the backend.

**Data Access Features:**

Visual Basic6.0 provides a rich set of data access features such as

* Data control
* Data Access object(DAO)
* Remote Data Control(RDC)
* Remote Data Object(RDO)
* ActiveX Data Object Data Control(ADODC)
* ActiveX Data Object(ADO)

**Data control :**

A data control binds data aware to Microsoft access or to other ODBC data sources, enabling to move from record to record and to display and manipulate data from the records in bound controls.A bound control is a control that is assigned to a field in a database.

Most data access operations can be performed with a data control without writing any code. If a database and a record source – a table, a view, a SQL statement is set to a data control, then it is automatically populated with data from that has the data control, is activated.

**Data Access object(DAO):**

Data Access objects communicate with Microsoft access and other complaint data sources through the jet engine. They provide properties and methods that allow to perform all the operations necessary to manage such a system.

**When to use DAO :**

* DAO is the only data access technology that supports 16 - bit operations. If an application must run within a 16 – bit environment, the DAO is the only choice.
* If the application must access both native Microsoft Jet and ODBC resources, DAO provides a consistent programming model.

**Remote Data Control(RDC)**

The Remote Data Control is another way for accessing remote data in Visual Basic6.0 applications. The user can interact data through the user interface controls like browsing the data, performing updates and adding records. Remote Data Control binds data aware controls to an ODBC remote database, and the remote data object. The Remote data control is similar to data control except that it creates and manipulates RDO objects.

**Remote Data Object(RDO):**

Remote data objects and collections provide a framework for using code to create and manipulate components of a remote ODBC database system.

**When to use RDO:**

* To access any 32 – bit level ODBC data source.
* To Virtually access remote databases such as SQL Server and Oracle8.

**ActiveX Data Object Data Control(ADODC):**

The ADODC uses Microsoft ActiveX Data Objects to quickly create connection between data bound controls and data providers(Any controls that feature a DataSource property).Data providers can be any source written to the OLE DB specification.

**ActiveX Data Object (ADO):**

ADO is not a database connector like DAO or ODBC, rather it is an extensible set of data access objects that is a programming model. These objects are based on OLE DB, which operates at the basic API(Application Programming Interface) level. ADO wraps this functionality into an easy to use flexible package that will be the basis of all of Microsoft’s future data access developments.

**List of Projects**

* 1. Inventory Control System
  2. Text Editor
  3. Railway Reservation System
  4. Material Requirement System
  5. Hospital Management System
  6. Personal Information System
  7. Web Based User Identification System
  8. Timetable Management System
  9. Hotel Management System

**Practice Exercises**

1. Create the following tables with the mapping given below.

a. **stu\_details** (reg\_no, stu\_name, DOB, address, city)

b. **mark\_details** (reg\_no, mark1, mark2, mark3, total)

(i) Alter the table mark\_details to add a column average with data type as long.

(ii) Display the months between the DOB and till date.

(iii) Using alter command drop the column address from the table stu\_details.

(iv) Display only those rows whose total ranges between 250 and 300.

(v). Drop the table **mark\_details**.

(vi). Delete the row whose reg\_no=161.

(vii). Display all details whose names begins with 'a'.

(viii)Find out the name of all students.

(ix)List all the student detail that who are all located in Chennai.

(x) Find the name of the student whose reg\_no is’107’.

(xi)Write a pl/sql program to find the address of a particular student using functions.

(xii)Design Student Details from the table using Front end tool.

(xiii)Display the details of a particular student whose name is ‘MATHU’.

(xiv)Rename the table **mark\_details** as ’**academics’.**

(xv)Write a pl/sql program to check whether the given number is prime or not.

(xvi)Write a pl/sql program to find the sum &avg marks of all the student using procedures.

2. Create the following tables with the mapping given below.

a. **emp\_details** (emp\_no, emp\_name, DOB, address, doj, mobile\_no, dept\_no, salary).

b. **dept\_details** (dept\_no, dept\_name, location).

(i) Display the months between the doj and till date.

(ii) Alter the table emp\_details to add a primary key constraint on emp\_no.

(iii)Write a pl/sql program to display the salary of a particular employee using functions.

(iv) Create a view emp1 from emp\_details such that it contains only emp\_no and

emp\_name.

(v) Select dept\_no from **dept\_details** and not in **emp\_details** using both the tables.

(vi) Truncate the table **dept\_details.**

(vii) Display the structure of the table **emp\_details.**

(viii) Convert the first letter of emp\_name into capitals.

(ix) Display the emp\_name getting highest salary.

(x)Select dept\_no from dept\_details and not in emp\_details using both the tables.

(xi) Create a table named as student and insert values into the table.

(xii) Write a pl/sql program to display the salary of a particular employee using functions.

(xiii)Select dept\_no from dept\_details and not in emp\_details using both the tables.

(xiv)Display the structure of the table emp\_details.

(xv)Display the emp\_namegetting highest salary

(xvi)Creating Trigger On Table Employ For Not Allowing Insert/Update/Delete Operations On Friday.

3. Design and implement payroll system to insert data's in DB and update Data’s in DB using VB as front end and Ms-Access as back end.

4. Write a PL/SQL program to find the greatest of 3 numbers.

5. Design a Menu Editor in Visual Basic.

6.Create the following tables with the mapping given below.

**a. book** (book\_name,author,price,quantity).

**b. customer** (Cust\_id, Cust\_name, Addr, ph\_no,pan\_no)

(i) Truncate the table **customer.**

(ii) List the author of the book which one have the price of 200.

(iii).List the price of the book which one is between the price of 175 & 250.

(iv).Retrieve all the details from the table book whose author name start with K.

7 .Write a pl/sql program to find the largest of two numbers.

8. Create the following tables with the mapping given below.

a. **Customer** (Cust\_id, Cust\_name, Addr, ph\_no,pan\_no)

b. **Loan** (Loan\_id, Amount, Interest, Cust\_id)

(i)Display the Cust\_name having both Loan and Account .

(ii) Display number of Loans, the sum of Loan Amount of a ParticularCustname(“LEENA”)

(iii)Display the Custname doesn’t hold any Account nor taken any Loan

(iv)Add a column nol(number of loans)

(v)Design a Report for Banking Process.

9.Write PL/SQL Program to generate even numbers.

10. Create the following tables with the mapping given below.

**Phone\_book**(ph\_no,name,door\_no,street,place).

(i)Display all names along withph\_no.

(ii)Add a column pin\_no.

(iii)Write a pl/sql program to find the address of a particular customer using functions.

11. Create the following tables with the mapping given below.

**emp\_details** (emp\_no, emp\_name, DOB, address, doj, mobile\_no, salary).

(i)Add a column dept\_no(department number).

(ii)Drop the column salary by altering the table.

(iii)Rename the table as’Employee’.

(iv) Design Employee Details Form using Front end tool.

12. Create the following table with the mapping given below.

**Customer** (Cust\_id, Cust\_name, Addr, ph\_no,pan\_no).

(i)Delete the row where cust\_name=’NANCY’.

(ii)Update the addr where cust\_name=’MATHIK’.

(iii)Display the details of a customer named ‘LITHUANA’.

(iv) Write a pl/sql program to swap two numbers.

(v) Write a pl/sql program to find the summation of odd numbers using for loop.

13. Create the following table with the mapping given below.

**book**(book\_name,author\_name,price,quantity).

(i) write a query to update the quantity by double in the table book.

(ii)List all the book\_namewhose price is greater than Rs.400.

(iii)Retrieve the list of author\_name whose first letter is ’a’ along with the book\_name and price.

14. Design Arithmetic Operations Form using Front end tool.

15. Create the following tables with the mapping given below.

a. **assessment**(reg\_no,name, mark1, mark2, mark3, total)

b.**dept\_details** (dept\_no, dept\_name, location).

(i) Using alter command drop the column location from the table dept\_details.

(ii) Display all dept\_name along withdept\_no.

(iii)Drop the table **dept\_details.**

(iv)Write a pl/sql program to find the sum &avg marks of all the student using procedures.

(v)Design a Report for Student Details.

16. Write a pl/sql code block to calculate the area & circumference of a circle

17. Create the following tables with the mapping given below.

**Phone\_book**(ph\_no,name,door\_no,street,place).

(i)List the price of the book which one is between the price of 400 & 500.

(ii)Creating Trigger To Insert Values To Another Table.

18. Create the following table with the mapping given below.

**retailor**(Cust\_id, Cust\_name, place, ph\_no,pan\_no).

(i) Alter the table to add a column pin\_no.

(ii) Display the details of customer who are all living in bangalore.

(iii) Display the customer name whose first letter is ‘M’.

(iv) Write a pl/sql program in procedure using in & out parameter.

(v) Design Customer Details Form using Front end tool.

19. Create the following table with the mapping given below.

**a.Product\_master**(product\_name,purchase\_prize,sell\_prize,profit,quantity,balance)

**b. Customer** (Cust\_id, Cust\_name, Addr, ph\_no,pan\_no).

(i)Display all the customer names along with their address.

(ii)Drop the table **customer.**

(iii) Change thesell\_price to 5000&purchase\_price amount to 4000 for any one of the product in product\_master.

(iv) Create a trigger to update the balance in product\_ master table whenever transaction table will have a new entry.

20. Create the following table with the mapping given below.

a. **stu\_details** (reg\_no, stu\_name, DOB, address, city)

b. **mark\_details** (reg\_no, mark1, mark2, mark3, total)

(i)Find out the name of all students along with their total marks.

(ii)Change the mark1 as ‘78’ from ‘59’ and alter the total for a particular student.

(iii) Delete all the records and its memory space from the table student.

(iv) Design Student Details Form using Front end tool.

21. Create the following table with the mapping given below.

**a.Administration**(employee\_salary, development \_cost, fund\_amount, turn\_over,bonus)

**b**. **Emp\_details** (emp\_no, emp\_name, DOB, address, doj, mobile\_no, dept\_no, salary).

(i)Calculate the total and average salary amount of the administration table.

(ii)Display total salary spent for employees.

(iii)Display total fundamount.

22. Create the following table with the mapping given below.

**emp\_details** (emp\_no, emp\_name, DOB, address, doj, mobile\_no, dept\_no, salary).

(i)List all employees which starts with either B or C.

(ii)Display the names and dob of all employees who were born in Feburary.

(iii)List out the employee names whose salary is greater than 15000.

(iv)Write a pl/sql program to find the sum of 1-100 numbers.

23.Create the following tables with the mapping given below.

a. **Customer** (Cust\_id, Cust\_name, Addr, ph\_no,pan\_no)

b. **Loan** (Loan\_id, Amount, Interest, Cust\_id,branch)

(i)Display the entire loan relation in descending order of the amount.

(ii)Find the names of all branches in loan relation.

(iii)Find all customer id's for loan's with loan amount between 5000 and 15000.

(iv)write a pl/sql code to accept the text and check the text is palindrome or not.

(v)Design a Report for Banking Process.

24.Create the following table with the mapping given below.

**book**(book\_name,author\_name,price,quantity).

(i)Write a query to find out the minimum quantity of books available in the table book.

(ii)Write a query to find the total price of all the books present in the table.

(iii)Write a query to find the average amount of all the books.

(iv)Write a pl/sql program to generate numbers from 0 to 25 in step of 5

(v)Data’s in DB using VB/Java as front end and Ms-Access as back end.

25.Create the following table with the mapping given below.

**Product\_master**(product\_name,product\_no,purchase\_prize,sell\_prize,profit,quantity,balance)

(i)Change the purchase\_prize into 551 for the product\_no pd3.

(ii)Delete all products from product\_master where the Quantity is equal to 40.

(iii)Find the product no & product name of the product which have a purchase amount equal to Rs.5500.

(iv) Design and implement payroll system to insert data's in DB and delete Data from DB using VB as front end and Ms-Access as back end.

26. Consider the insurance database given below.

PERSON(driver\_id**,** name, address) CAR(regno, model,year ) ACCIDENT(report\_number,accd\_date,location) OWNS(driver\_id,regno) PARTICIPATED(driver\_id,regno,report\_number,damage\_amount)

i. Create the above tables by properly specifying the primary keys and foreign keys and enter at least five tuples for each relation.

ii. Update the damage amount for the car with specific regno in the accident with report number 12 to 25000.

iii. Add a new accident to the database.

iv. Find the total number of people who owned cars that were involved in accidents in the year 2008.

v. Find the number of accidents in which cars belonging to a specific model were involved.

27. a. Consider the following employee and department tables.

EMPLOYEE(empno, ename, designation, manager, hiredate, salary, commission,

deptno)

DEPARTMENT(deptno, dname, location)

i. Create the above tables by properly specifying the primary keys and foreign keys and enter at least five tuples for each relation.

ii. List the names of employees whose name contain substring ‘LA’.

iii. List the details of employees of salary are greater than or equal to the average salary of

employee table.

iv. Create a view which consists of details of all ‘SALESMAN’.

b. i. Write a PL/SQL to display the empno,job,salary of all employees in employee table.

ii. Write a trigger to ensure that empno in employee table is maintained as primary key.

28. Consider the following tables.

SAILOR(sid, sname, rating, age)

BOATS(bid, bname, colour)

RESERVES(sid, bid, day)

i. Create the above tables by properly specifying the primary keys and foreign keys and enter at least five tuples for each relation.

ii. List the sailors in the descending order of their rating.

iii. List the sailors whose youngest sailor for each rating and who can vote.

iv. List the sailors who have reserved for both ‘RED’ and ‘GREEN’ boats.

v. List the details of the oldest sailor for each rating level.

29. Consider the following relations for order processing database application in a company.

CUSTOMER(custno, cname, city) ORDER(orderno, odate, custno, ord\_amt ) ORDER\_ITEM(orderno, itemno, quantity) ITEM(itemno, unitprice) SHIPMENT(orderno, warehouseno, ship\_date) WAREHOUSE(warehouseno, city)

i. Create the above tables by properly specifying the primary keys and foreign keys and enter at least five tuples for each relation.

ii. Produce a listing: custname , No\_of\_orders , Avg\_order\_amount , where the middle column is the total number of orders by the customer and the last column is the average order amount for that customer.

iii. List the orderno for orders that were shipped from ***all*** the warehouses that the company has in a specific city.

iv. Demonstrate the deletion of an item from the ITEM table and demonstrate a method of handling the rows in the ORDER\_ITEM table that contains this particular item.

30. Consider the following employee and department tables.

EMPLOYEE(empno, ename, designation, manager, hiredate, salary, commission,

deptno)

DEPARTMENT(deptno, dname, location)

i. Create the above tables by properly specifying the primary keys and foreign keys and enter at least five tuples for each relation.

ii.. List the different job titles of employee table.

iii. List the details of employees of with minimum salary of employee table.

31. Develop an application for Library Management System

32. Develop an application for Bank Management System

33. Develop a software for a game.

34. Develop an application for student information system

35. Develop an application for payroll system

36. **Table 1 :** STUDIES  
PNAME (VARCHAR), SPLACE (VARCHAR), COURSE (VARCHAR), CCOST (NUMBER)  
**Table 2 :** SOFTWARE  
PNAME (VARCHAR), TITLE (VARCHAR), DEVIN (VARCHAR), SCOST (NUMBER), DCOST (NUMBER), SOLD (NUMBER)  
**Table 3 :** PROGRAMMER  
PNAME (VARCHAR), DOB (DATE), DOJ (DATE), SEX (CHAR), PROF1 (VARCHAR), PROF2 (VARCHAR), SAL (NUMBER)  
**LEGEND :**  
PNAME – Programmer Name, SPLACE – Study Place, CCOST – Course Cost, DEVIN – Developed in, SCOST – Software Cost  
, DCOST – Development Cost, PROF1 – Proficiency 1  
**QUERIES :**  
1. Find out the selling cost average for packages developed in Oracle.  
2. Display the names, ages and experience of all programmers.  
3. Display the names of those who have done the PGDCA course.  
4. What is the highest number of copies sold by a package?  
5. Display the names and date of birth of all programmers born in April.  
6. Display the lowest course fee.  
7. How many programmers have done the DCA course.  
8. How much revenue has been earned through the sale of packages developed in C.  
9. Display the details of software developed by Rakesh.  
10. How many programmers studied at Pentafour.  
11. Display the details of packages whose sales crossed the 5000 mark.  
12. Find out the number of copies which should be sold in order to recover the development cost of each package.  
13. Display the details of packages for which the development cost has been recovered.  
14. What is the price of costliest software developed in VB?  
15. How many packages were developed in Oracle ?  
16. How many programmers studied at PRAGATHI?  
17. How many programmers paid 10000 to 15000 for the course?  
18. What is the average course fee?  
19. Display the details of programmers knowing C.  
20. How many programmers know either C or Pascal?  
21. How many programmers don’t know C and C++?  
22. How old is the oldest male programmer?  
23. What is the average age of female programmers?  
24. Calculate the experience in years for each programmer and display along with their names in descending order.  
25. Who are the programmers who celebrate their birthdays during the current month?   
26. How many female programmers are there?  
27. What are the languages known by the male programmers?  
28. What is the average salary?  
29. How many people draw 5000 to 7500?  
30. Display the details of those who don’t know C, C++ or Pascal.  
31. Display the costliest package developed by each programmer.  
32. Produce the following output for all the male programmers

37. **Table 1 :** DEPT  
DEPTNO (NOT NULL , NUMBER(2)), DNAME (VARCHAR2(14)),  
LOC (VARCHAR2(13)  
**Table 2 :** EMP  
EMPNO (NOT NULL , NUMBER(4)), ENAME (VARCHAR2(10)),  
JOB (VARCHAR2(9)), MGR (NUMBER(4)), HIREDATE (DATE),  
SAL (NUMBER(7,2)), COMM (NUMBER(7,2)), DEPTNO (NUMBER(2))  
MGR is the empno of the employee whom the employee reports to. DEPTNO is a foreign key.

**QUERIES**

1. List all the employees who have at least one person reporting to them.  
2. List the employee details if and only if more than 10 employees are present in department no 10.  
3. List the name of the employees with their immediate higher authority.  
4. List all the employees who do not manage any one.  
5. List the employee details whose salary is greater than the lowest salary of an employee belonging to deptno 20.  
6. List the details of the employee earning more than the highest paid manager.  
7. List the highest salary paid for each job.  
8. Find the most recently hired employee in each department.  
9. In which year did most people join the company? Display the year and the number of employees.  
10. Which department has the highest annual remuneration bill?  
11. Write a query to display a ‘\*’ against the row of the most recently hired employee.  
12. Write a correlated sub-query to list out the employees who earn more than the average salary of their department.  
13. Find the nth maximum salary.  
14. Select the duplicate records (Records, which are inserted, that already exist) in the EMP table.  
15. Write a query to list the length of service of the employees (of the form n years and m months).