Lab Manual

Database Management Systems Lab

Dr K ARUNA KUMARI
D N S B KAVITHA

Department of Computer Science and Engineering
SRKR Engineering College (A), Bhimavaram, India

This lab manual is intended to aid the undergraduate second-year computer science and engineering students for their course *Data Base Management Systems Lab [B20 CS 2204]*.

About the authors

Dr K ARUNA KUMARI: She got her PhD from KL University, M.Tech from Archarya Nagarjuna University and MCA from Bharathidasan University,. Presently working as Assistant Professor in the Department of Computer Science and Engineering, SRKR Engineering College, Bhimavaram, India.

D N S B KAVITHA: She is pursuing PhD from Andhra University, M.Tech and B.Tech from JNTU Kakinada. Presently working as Assistant Professor in the Department of Computer Science and Engineering, SRKR Engineering College, Bhimavaram, India.

Evaluation Scheme	
Examination	Marks
Day to Day Evaluation	5
Record	5
Internal Exam	5
External Exam	35

Preface

Database management systems are now an indispensable tool for managing information, and a course on the principles and practice of database systems is now an integral part of computer science curricula.

Database management has evolved from a specialized computer application to a central component of a modern computing environment, and, as a result, knowledge about database systems has become an essential part of an education in computer science.

This manual is intended for the second-year students of Computer Science and Engineering in the subject of Database Management System. This manual typically contains practical/Lab sessions related to Database Management Systems to enhance understanding.

Students are advised to thoroughly go through this manual rather than only topics mentioned in the syllabus as practical aspects are the key to understanding and conceptual visualization of theoretical aspects covered in the books.

Course Plan

Second Semester

Database Management Systems Lab

Week	Name of The Experiment	Session
1	Introduction to database management system environments, Database Creation	1
2	Creation of tables using DDL and DML commands	2
3	Working with queries using ANY, ALL, IN, IN, EXISTS, NOTEXISTS, UNION, INTERSECT and Constraints	3
4	Working with Aggregate Functions and Views	4
5	Working with queries using Conversion functions	5
6	**Repetition Class**	
7	Theory Mid exams	
8	PL/SQL program for exceptions and implement commit, rollback, savepoint in pl/sql program	6
9	Working with pl/sql control statements, Nullif, Coalesce	7
10	Working with Loop statements, Raise Exceptions	8
11	Working with Procedures using In, Out, In Out Parameters	9
12	Working with Functions	10
13	Working with Cursors	11
14	Working with Triggers	12
15	**Lab Internal**	
16	Theory Mid exams	
17	**Sem End Lab Exam**	

Contents

Session #1

Data Base Creation

Learning Objective

To familiarize the students with database management systems environment (SQL Developer) and to create a new user for daily log in.

Learning Context

DBMS is a software system that allows the user to define, to create and maintain the database and provide control access to the data. The following procedure is to be used to create a user for daily log in.

1. For getting ORACLE SQL DEVELOPER prompt
   - Click start
   - Click all programs
   - Click SQL Developer

2. At Logon window enter the following data.

Username	:	system
Password	:	manager
Host string	:	cse

3. After getting SQL Developer prompt type the following command to create a new user user001 for daily log in with 100 mb of space.

```
create user user001 identified by password 001 quota 100m on users;
```

4. Grant permissions to user using the following command.

```
grant connect, create session, resource to user001;
```

5. Now copy emp and dept tables from scott to user001 using the following commands,

```
create user001.dept as select * from scott. dept;
create user001. emp as select * from scott. emp;
```

6. Now use the following sequence to save the work and quit from SQL prompt.

```
commit;
exit;
```

7. Again, go to ORACLE SQL DEVELOPER prompt.

8. At Logon window enter the following data.

Username	:	user001
Password	:	001
Host string	:	cse

9. Now type the following commands start the work.

```
show user;
set linesize 100;
```

10. Now start doing the queries and note down the queries and results.
It may be noted that after executing every SQL command we have to execute

```
select * from table;
```

and check manually that the result is the desired one or not.

Materials & Resources for the lab sessions

Text Books

1. Abraham Silberschatz, Henry F. Korth, S. Sudarshan, "Database System Concepts", McGraw-Hill, 4th Edition, 2002.
2. Ivan Bayross, “SQL, PL/SQL The programming language of oracle”, BPB publications, 4th Revised Edition, 2010.
3. Ramez Elmasri, Shamkant, B. Navathe, “Database Systems”, Pearson Education, 6th Edition, 2013.
4. Peter Rob, Carles Coronel, “Database System Concepts”, Cengage Learning, 7th Edition, 2008.
5. M L Gillenson, “Introduction to Database Management”, Wiley Student Edition, 2012.

Web References:

1. https://www.tutorialspoint.com/dbms/
2. https://www.javatpoint.com/dbms-tutorial

Session #2

DDL and DML Commands

Learning Objective

Creation of Tables and usage of DML commands insert, update and delete.

Learning Context

Create table: Tables are defined using the **create** command. The simplified form of create command is

```
create table   r
( A1 D 1, A 2 D2 ,…, An Dn,
[ integrit y_ constraint 1 ],…,[ integrity_
constraint n])
| as select_statement;
```

Where create and table is key words.
Create, table and **as** are keywords.

- r is the name of relation
- A1, A2, , An, are attributes of the relation separated by domain types D 1, D2,…
- the way of mentioning integrity constraints are
  - **Primary key** (attribute). It is a set of attributes that uniquely identifies a tuple in a relation and selected by the DBA for the purpose.
  - **foreign key** (Ar) **references** r 1. Or
    constraint c1 **foreign key** (Ar) **references** r1.

Here c1 is constraint name. Foreign key is a combination of attributes with values based on the primary key values from another table. The meaning of this referential integrity here is that for each tupple in the relation, the values of 'Ar' must exist in the referring relation.

- **check** (condition) the optional select statement can be used to create a table from another reference table.

INSERT: Tables are constructed using the insert command. The simplified form of insert command is

```
insert    into    r
[( A1, A2, .., An)]   values( v1 , v2, . ., vn)
| select statement;
```

- **Insert**, **into** and **values** are key words.
- r is the name of relation
- A1, A2, ... , An, are attributes of the relation and v1, v2 ,.., vn are the corresponding values. If the optional attribute list is not given, all the attributes in the relation are assumed.
- The optional select statement can be used to insert values from another table. But here it is to be ensured that both tables possess compatible data types.

UPDATE: The attributes values are updated using the update command. The simplified form of update command is

```
update table_ name | view_ name
set A = v1
where conditions;
```

- **Update**, **set** and **where** are key words
- A is attribute whose new value will be v1.
- The attribute values for the all rows satisfying the conditions are updated.
- If where condition is not given, the attribute values for the all rows are updated.

DROP TABLE: Table can be dropped using drop table. The syntax of the command is

```
drop table [ owner.] table_ name;
```

Drop table is a key word.
Dropping a table in Oracle frees the space used by the table and commits any pending changes to the database. All indexes and grants associated with the table are lost. Objects, such as views, stored procedures, and synonyms built up on the table, are marked invalid and cease to function.

DELETE: Table contents can be deleted using delete command. The syntax of delete command is

```
delete from [ owner.]  table_ name [ where
```

- **Delete**, **from** & **where** are key words
- The attribute values for the all rows satisfying the conditions are deleted.
- If where condition is not given, the attribute values for all rows are deleted.

ALTER TABLE: Table structure can be altered using alters table command. The syntax of alter table command is

```
alter table [ owner_ name.] table_ name
 add A D | modify A D | drop    column A ;
```

- **alter table, add, modify, drop, column** are key words.
- **add, modify**, **drop** are used to append an attribute, to change the data typeof an existing attribute, and to delete an attribute respectively.
- A is attribute and D is domain type.

Procedure

1. Logon into SQL prompt/ livesql.oracle.com
2. Set the line size using the following command.
   set line size 100;
3. Create the tables with create table command.
4. Type the following command after creation or modification of each table to view the structure of the table.
   desc table_name;
5. In between insertions and after all the data is inserted, save the data.
6. After inserting all the data or after modification to data, we can view the data by typing
   Select * from table_name;
7. Use the following commands after all the task is completed.
   commit;
   exit;

Learning Outcome

After undergoing this laboratory module, the student should be able to create, alter and delete tables using DDL commands and modify the table data using DML commands insert, update and delete.

Exercise

1. Create three tables with the following schema
   Sailors(sid:integer, sname:string, rating:integer, age:real)
   Boats(bid:integer, bname:string, color:string)
   Reserves(sid:integer, bid:integer, day:date)
   Impose the appropriate integrity constraints.

2. Insert the following values into the above tables.

Sailors			
sid	sname	rating	age
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	40
85	Art	3	25.5
95	Bob	3	63.5

Reserves		
sid	bid	day
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Boats		
bid	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

3. Change dustin rating to 0
4. Restore dustin rating to 7
5. Increment the rating of all sailors by 10
6. Restore ratings of sailors by decrementing the rating of all sailors by 10
7. Change sailors names to all capital letters
8. Add one row of your own choice and delete that row in reserves table.
9. Alter the sname field of Sailors table to accommodate more characters.
10. Create a table demo with the fields sno, sname, salary and insert 3 rows starting with sno1.
11. Increase the size of sname
12. Update the salary of sno 1 in the demo table to 500
13. Update all the salaries in the demo table to 1000 except sno 1.
14. Delete all the rows in demo table.
15. Add “status” column to demo table
16. Delete the field “status” from dummy table.
17. Create table sailors2 with same data as in sailors.
18. Truncate sailors2.
19. Drop table sailors2.
20. Drop table demo.

Session #3

Working with Queries and Nested Queries

Learning Objective

To practice select queries and nested queries.

Learning Context

Nested Queries

A nested query is a query that has another query embedded within it; the embedded query is called a subquery. Subqueries are an alternate way of returning data from multiple tables.
Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.

Subqueries with the SELECT Statement

SQL subqueries are most frequently used with the Select statement.

Syntax

```
SELECT column_name
FROM table_name
WHERE column_name expression operator
(SELECT column_name from table_name WHERE ...);
```

Example:

```
SELECT sname FROM sailors s
WHERE s.sid = (SELECT DISTINCT r.sid from reserves r WHERE s.sid=r.sid);
```

This nested query displays the names of sailors who have reserved at least one boat.

IN Operator: - The IN operator allows us to test whether a value is in a given set of elements; an SQL query is used to generate the set to be tested.

Syntax

```
SELECT ColumnName(s)
FROM TableName
WHERE ColumnName IN (Value1, Value2...);
```

Example: -

Find the names of sailors who have reserved boat 103 using IN Operator.

```
SELECT S.SNAME FROM SAILORS S WHERE S.SID
IN (SELECT R.SID FROM RESERVES R WHERE R.BID = 103);
```

NOT IN Operator: - The NOT IN is used in an opposite manner to IN.
Example: -
Find the names of sailors who have not reserved boat 103 using NOT IN Operator.

```
SELECT S.SNAME FROM SAILORS S WHERE S.SID
NOT IN (SELECT R.SID FROM RESERVES R WHERE R.BID = 103);
```

EXISTS Operator: - This is a Correlated Nested Queries operator. The EXISTS operator is another set comparison operator, such as IN. It allows us to test whether a set is nonempty, an implicit comparison with the empty set.

Syntax

```
SELECT ColumnName(s)
FROM TableName
WHERE EXISTS
(SELECT ColumnName FROM TableName WHERE condition);
```

Example: - Find the names of sailors who have reserved boat number 103.

```
SELECT S.SNAME FROM SAILORS S WHERE
EXISTS (SELECT * FROM RESERVES R WHERE R.BID = 103 AND R.SID = S.SID );
```

NOT EXISTS Operator: - The NOT EXISTS is used in an opposite manner to EXISTS.

Example: - Find the names of sailors who have not reserved boat number 103.

```
SELECT S.SNAME FROM SAILORS S WHERE NOT EXISTS (SELECT * FROM RESERVES
R WHERE R.BID = 103 AND R.SID = S.SID);
```

ALL Operator: - It is a comparison operator. The ALL operator is used with a WHERE or HAVING clause and returns TRUE if all of the subquery values meet the condition.

Syntax

```
SELECT ColumnName(s)
FROM TableName
WHERE ColumnName operator ALL
(SELECT ColumnName FROM TableName WHERE condition);
```

where the operator is one of the arithmetic, comparison operators {<, <=, =, <>, >=, >}.

Example: - Find the sailor's with the highest rating.

```
SELECT S.SID FROM SAILORS S WHERE S.RATING >= ALL (SELECT S2.RATING FROM
SAILORS S2);
```

ANY Operator: - Similar to the ALL operator, the ANY operator is also used with a WHERE or HAVING clause and returns true if any of the subquery values meet the condition.

Syntax

```
SELECT ColumnName(s)
FROM TableName
WHERE ColumnName operator ANY
(SELECT ColumnName FROM TableName WHERE condition);
```

Example: - Find sailors whose rating is better than some sailor called Horatio

```
SELECT S.SID FROM SAILORS S WHERE S.RATING > ANY (SELECT S2.RATING FROM
SAILORS S2 WHERE S2.SNAME = ' Horatio ');
```

BETWEEN Operator: - The BETWEEN operator is used, when you want to select values within a given range. Since this is an inclusive operator, both the starting and ending values are considered.

Syntax

```
SELECT ColumnName(s)
FROM TableName
WHERE ColumnName BETWEEN Value1 AND Value2;
```

LIKE Operator: - The LIKE operator is used in a WHERE clause to search for a specified pattern in a column of a table.

There are mainly two wildcards that are used in conjunction with the LIKE operator:

% : It matches 0 or more characters.

_ : It matches exactly one character.

Syntax

```
SELECT ColumnName(s)
FROM TableName
WHERE ColumnName LIKE pattern;
```

AND Operator: - This operator is used to filter records that rely on more than one condition. This operator displays the records, which satisfy all the conditions separated by AND, and give the output TRUE.

Syntax

```
SELECT Column1, Column2, ..., ColumnN
FROM TableName
WHERE Condition1 AND Condition2 AND Condition3 ...;
```

OR Operator: - This operator displays all those records which satisfy any of the conditions separated by OR and give the output TRUE.

Syntax

```
SELECT Column1, Column2, ..., ColumnN
FROM TableName
WHERE Condition1 OR Condition2 OR Condition3 ...;
```

NOT Operator: - The NOT operator is used when you want to display the records which do not satisfy a condition.

Syntax

```
SELECT Column1, Column2, ..., ColumnN
FROM TableName
WHERE NOT Condition;
```

SET OPERATIONS

SQL supports the set operations union, intersection and minus with the keyword's union, intersect, and except (minus in oracle).

UNION: The UNION operator returns records from the result of both queries after eliminating the duplicate records which are common in both. There is another option of union, the UNION ALL operator, which returns records from the result of both queries, including duplicate records which are common in both.

Syntax

```
SELECT ColumnName(s) FROM Table1
UNION
SELECT ColumnName(s) FROM Table2;
```

INTERSECT: The INTERSECT operator returns the records which are common in the result of both queries.

Syntax

```
SELECT Column1 , Column2 ....
FROM TableName
WHERE Condition
INTERSECT
SELECT Column1 , Column2 ....
FROM TableName
WHERE Condition;
```

EXCEPT: The EXCEPT operator returns the records which are in the result of the first query but not in the result of the second one.

Syntax

```
SELECT ColumnName
FROM TableName
EXCEPT
SELECT ColumnName
FROM TableName;
```

Learning Outcome

After undergoing this laboratory module, the student should be able to use SQL select command in queries and nested queries by including various operators to select the rows.

Exercise

1. **Create three tables with the following schema**
   Sailors(sid:integer, sname:string, rating:integer, age:real)
   Boats(bid:integer, bname:string, color:string)
   Reserves(sid:integer, bid:integer, day:date)
   Give the appropriate integrity constraints.

2. **Insert the following values into the above tables.**

Sailors				Reserves			Boats		
sid	sname	rating	age	sid	bid	day	bid	bname	color
22	Dustin	7	45.0	22	101	10/10/98	101	Interlake	blue
29	Brutus	1	33.0	22	102	10/10/98	102	Interlake	red
31	Lubber	8	55.5	22	103	10/8/98	103	Clipper	green
32	Andy	8	25.5	22	104	10/7/98	104	Marine	red
58	Rusty	10	35.0	31	102	11/10/98			
64	Horatio	7	35.0	31	103	11/6/98			
71	Zorba	10	16.0	31	104	11/12/98			
74	Horatio	9	40	64	101	9/5/98			
85	Art	3	25.5	64	102	9/8/98			
95	Bob	3	63.5	74	103	9/8/98			

3. Find the names of sailors who have reserved **at least** one boat.
4. Find the ids of sailors who reserved a red boat **and** a green boat.
5. Find the ids of sailors who reserved a red boat **or** a green boat.
6. Find the ids of sailors who reserved a red boat **but not** a green boat.
7. Find the names of sailors who reserved **every** boat.
8. Find the names of sailors who reserved **every** red boat.
9. Find all sids of sailors who have a rating of 10 or reserved boat number 101.
10. Find the ids of sailors whose rating is more than the average rating of all sailors
11. Find the sids of sailors whose rating is better than some sailors called horatio.
12. Find the sailors whose rating is better than every sailor called horatio.
13. Find the names of boats reserved by dustin and by no one else.
14. Display the details of sailors whose names start with letter r.
15. Display the details of sailors whose names end with letter r.
16. Display the name, rating and age of sailors whose rating is between 8 and 10.

Session #4

Aggregate Functions & Views

Learning Objective

To implement Aggregate functions and create, drop views

Learning Context

Aggregate functions: that take a collection of values as input and return a single value.
SQL offers five built-in aggregate functions.

- **MIN (A)** : returns minimum of column values
- **MAX (A)** : returns maximum of column values
- **AVG ([distinct] A)** : returns average of unique column values
- **SUM ([distinct] A)** : returns total of unique column values
- **COUNT ([distinct] A)** : returns number of unique column values

The input to **SUM** and **AVG** must be a collection of numbers but the other operators can operate on collection of non -numeric data types like strings.
For example,

select sum(rating) from **sailors;** displays total rating of all the sailors.
select count(sid) from **sailors**; displays no. of sailors.
select count (*) from **sailors;** displays no. of records in sailors.

All the functions except COUNT (*) ignore null values in their input collection. For SQL null value means unknown or inapplicable.

group by & having Clauses:

The select statement syntax including optional group by and having clauses is as under:

select [distinct] select-list **from** from-list
[where qualification]
[group by group-list]
[**having** group-qualification];

The result of the GROUP BY clause is a grouped table i.e., a set of groups of rows derived from the table by conceptually rearranging it into the minimum number of groups such that within any one group all rows have the same value for the combination of columns identified by GROUP BY clause.

Example

select sum(sal) from emp **group by** job;

Using this statement, we get the total salary for each designation of the employees. There is a restriction on the select clause while using group by. Select item in the select clause must be single valued per group as for the other aggregate functions. Therefore, a statement like

select empno, sum(sal) **from** emp **group by** job;

select * **from** emp **group by** job; are invalid.

But we can give

select job **from** emp **group by** job;

and

select job, avg(sal) **from** emp **group by** job; are valid.

HAVING: clause works very much like a where clause except that its logic is related only to the result of group functions i.e., having clause is used to eliminate groups whereas where clause is used to eliminate rows from the whole table.

Example

select deptno, avg(sal) from emp group by deptno **having** max(sal) > 1500;

Displays average salaries depart wise for departments which have average salary greater than 1500.

VIEWS:

A view is a table whose rows are not explicitly stored in the database, but are computed as needed from the view definition.

Syntax of creating a view

CREATE [OR REPLACE] VIEW view_name

[(column [,...n])]**AS SELECT**_statement [**WITH CHECK OPTION**]

Example

The below command creates a view for dbms teacher to view student name and marks in dbms. But the original table contains sno, sname and all subject marks.

CREATE VIEW dbms (name, dbmarks) AS SELECT sname, s1 FROM Students;

Syntax of deleting view

DROP VIEW view_name;

In SQL, view is said to be updatable (i.e. insert, update or delete) if the following conditions are all satisfied:

- The from clause has only one relation.
- The select clause contains only attributes of the relation and does not have any expressions, aggregate functions, or distinct specification.
- Any attribute not in the select clause can be set to null.

NOTE: *If we insert a row which does not satisfy condition in the where clause of create view then it is allowed and that row doesn't appear in the view. To avoid such insertions, we have to use with check option in the create view.*

Learning Outcome

After undergoing this laboratory module, the student should be able to use aggregate functions and create views

Exercise

1. Count the number of different sailor names.
2. Calculate the average age of all sailors.
3. Find the name and the age of the youngest sailor.
4. Find the name and the age of the oldest sailor.
5. Displays all the sailors according to rating (Topper First), if rating is same then sort according to age (Older First).
6. Displays all the sailors according to rating (Topper First), if rating is same then sort according to age (Younger First).
7. Displays all the sailors according to rating (Lower Rating First), if rating is same then sort according to age (Younger First).
8. Find the average age of sailors for each rating level.
9. Find the average age of sailors for each rating level that has at least two sailors.
10. Find the names of sailors who are older than the oldest sailor with a rating of 10.
11. Find the age of the youngest sailor who is eligible to vote (i.e., is at least 18 years old) for each rating level with at least two such sailors.
12. For each red boat, find the number of reservations for this boat.
13. Find the average age of sailors who are of voting age (i.e., at least 18 years old) for each rating level that has at least two sailors.
14. Create a view for Expert_Sailors (A sailor is an expert sailor if his rating is > 9).
15. Create a view for a subject_teacher (name, s1) from student table which has the fields roll number, name, all subject marks i.e. s1, s2.. Students. Try to insert a row into the view.

Session #5

Queries using Conversion Functions

Learning Objective

To implement conversion functions, String, and Date functions

Learning Context

Date functions

Function	Description
Add_months(d, n)	Returns a date after adding a specified date with the specified number of months
Last_day(d)	Returns the date corresponding to the last day of the month
Months_between(d1, d2)	Returns the number of months between two dates
Next_day(d, day)	Displays the next day
Round(d, [fmt])	Returns the date which is rounded to the unit specified by the format model
Trunc(d, [fmt])	Returns the date with the time portion of day truncated to the unit specified by format model
Greatest(d1, d2, d3…)	Returns the latest date present in the arguments

Character Functions

Function	Example
Initcap(char)	Select initcap('hello') from dual;
Lower(char)	Select lower('FUN') from dual;
Uppder(char)	Select upper('sun') from dual;
Ltrim(char, set)	Select ltrim('xyzadams','xyz') from dual;
Rtrim(char, set)	Select rtrim('xyzadams','ams') from dual;
Translate(char, from, to)	Select translate('jack', 'j', 'b') from dual;
Replace(char, searchstring, [rep string])	Select replace('jack and jue', 'j', 'bl') from dual;
Substr(char, m, n)	Select substr('abcdefg', 3, 2) from dual;

Conversion Functions

Function	Description
To_char(d, [fmt])	Converts date to a value of varchar2 datatype in a form specified by date format fmt. If fmt is neglected then it converts date to varchar2 in the default date format
To_date(char, [fmt])	Converts char or varchar datatype to date datatype.

Numeric Functions

Function	Example
Abs	Select abs(-15) from dual;
Ceil(n)	Select ceil(44.778) from dual;
Cos(n)	Select cos(180) from dual;
Cosh(n)	Select cosh(0) from dual;
Exp(n)	Select exp(4) from dual;
Floor(n)	Select floor(100.2) from dual;
Power(m,n)	Select power(4,2) from dual;
Mod(m,n)	Select mod(10,2) from dual;
Round(m,n)	Select round(100.256,2) from dual;
Trunc(m,n)	Select trunc(100.256,2) from dual;
Sqrt(n)	Select sqrt(4) from dual;

Learning Outcome

After undergoing this laboratory module, the student should be able to write queries using conversion functions, string, number and date functions.

Exercise

1. Convert 12345 to string.
2. Display system date after converting to varchar2 data type.
3. Display system date in 'MON-DD-YYYY' format after converting to varchar2 data type.
4. Convert string '123.45' to number data type.
5. Concate the string 'Rajesh' with 'Raghu'
6. Concate bid & bname of Boats & display along with color.
7. Lpad the string 'Rajesh' to length 30 with the set of characters in string '*'
8. Lpad the string bname to length 20 with '*' set of characters and string color by '/'.
9. Rpad the string 'Rajesh' to length 30 with the set of characters in string '*'

10. Rpad the string sname to length 25 with '*' set of characters and remaining attributes in normal way.
11. Rpad the string bname to length 20 with '*' set of characters and lpad the same to make it length 30 with '#' and remaining attributes in normal way.
12. Display all sailors information by removing characters of sname if starts with 'R'.
13. Display all sailors information by removing characters of sname if starts with 'H'
14. Display all sailors information by removing characters of sname if ends with 'o'.
15. Display all Boats information by removing characters of color if ends with 'r'.
16. Display all Boats information by showing their names in lower case.
17. Display all Sailors information by showing their names in lower case.
18. Display all Sailors information by showing their names in upper case.
19. Display all Boats information by showing their color in Upper case.
20. Display all Sailors information by showing their names in Capitalizing first char.
21. Capitalize first letter of each word in 'rajesh raghu'.
22. Find the number of characters in the string 'Computer Science and Engineering '.
23. Display length of string SID, SNAME from Sailors along with their values.
24. Display boats information by starting their names with 3rd character & show only 4 characters.
25. Display the index of string 'AB' after 2nd character & 3rd occurrence in the given string 'ABCDABCDABABAB'.
26. Replace 'A' with 'D' in the given string 'ABCDABCDABABAB'.
27. Display Sailors information by replacing 'A' with 'I' from SNAME, if any.
28. Display BNAME by replacing 'ER' with 'MA'.
29. Display the system date for a system.
30. Display next date on DAY after date D. Ex: Display date on Thu after 20th Feb, 2018.
31. Display SID, Day of Reservation by adding 20 months to given day.
32. Display SID, Day of Reservation and date corresponding to last date of the month.
33. Display SID, Day of Reservation and months between System Date & day of reservation.
34. Find least value in 12, 53, 17, 2.
35. Find least value in '12', '53', '17', '2'
36. Find least value in 'APPLES', 'ORANGES', and _BANANAS'
37. Find greatest value in 12, 13, 17, 2.
38. Find greatest value in '12', '13', '17', '2'
39. Find greatest value in 'APPLES', 'ORANGES', 'BANANAS'
40. Truncate the 5632.98345 number to 0, 1, 2 decimal places.
41. Round off the 5632.98345 number to 0, 1, 2 decimal places.

Session #6

PL/SQL

Learning Objective

To implement PL/SQL and Commit, Rollback, and Savepoint

Learning Context

PL/SQL stands for Procedural Language/Structured Query Language. PL/SQL is the procedural extension to SQL with design features of programming languages. Data manipulation and query statements of SQL are included within procedural units of code. PL/SQL Block Structure is as follows.

```
< header>
IS | AS
    Declaration section
BEGIN
    Executable section
EXCEPTION (optional)
   /*Exception section*/

  WHEN exception1 THEN
  Statement1;
  WHEN exception2 THEN
  Statement2;
  [WHEN others THEN]
  /*default exception handling code*/
END;
```

Section	Description
Header	Required for named blocks. Specifies the way the program is called by other PL/SQL blocks. Anonymous blocks do not have a header. They start with the DECLARE keyword if there is a declaration section, or with the BEGIN keyword if there are no declarations.
Declaration	It is optional. Declares variables, cursors, TYPEs, and local programs that are used in the block's execution and exception sections.
Execution	Optional in package and TYPE specifications; contains statements that are executed when the block is run.
Exception	Optional; describes error-handling behavior for exceptions raised in the executable section.

As we want output of PL/SQL program on screen, before starting writing anything type (Only Once per session) SET SERVEROUTPUT ON

An example of an anonymous block:

The following code declares today as **date** data type and the default value is system date which can be obtained from the environment variable **SYSDATE.** Display the date after concatenating it with the string.

```
DECLARE
  today DATE DEFAULT SYSDATE;
BEGIN
  DBMS_ OUTP UT. PUT_LINE (' Today is ' || today);
END;
```

Learning Outcome

After undergoing this laboratory module, the student should be able to write PL/SQL block and workout with TCL commands commit, savepoint and rollback.

Exercise

1. Create a simple PL/SQL program which includes a 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. Insert data into student table and use COMMIT, ROLLBACK and SAVEPOINT in PL/SQL block.

Solution:

i). We have to create the student table and insert the records in to the table as follows:

```
SQL> create table student(sid number(10),sname varchar2(20),rank varchar(10));
Table created.
SQL> insert into student values(501,'Ravi','second');
1 row created.
SQL> insert into student values(502,'Raju','third');
1 row created.
SQL> insert into student values(503,'Ramu','');
1 row created.
SQL> select *from student;
SID       SNAME          RANK
---------- -------------------- ----------
501       Ravi           second
502       Raju           third
503       Ramu
```

```
set serveroutput on;
declare
temp1 number(10);
temp2 varchar2(10);
begin
select sid, sname into temp1, temp2 from student where rank='first';
dbms_output.put_line('Student No:'|| temp1 ||' Name:'||temp2||' got first rank');
exception
when no_data_found then
dbms_output.put_line('**********************************************');
dbms_output.put_line('# Error: there is no student got first rank');
end;
/
update student set rank='first' where sid=503;
1 row updated.
SQL> select * from student;
SID        SNAME              RANK
---------- -------------------- ----------
501        Ravi               second
502        Raju               third
503        Ramu               first
Student No:503 Name:Ramu got first rank
PL/SQL procedure successfully completed
```

ii) SQL> select *from student;

```
ID              SNAME        RANK
---------- -------------------- ----------
501         Ravi               second
502         Raju               third
503         Ramu               first
```

PL/SQL CODE:

```
-- Procedure to execute in livesql worksheet
DECLARE
sno student.sid%type;
name student.sname%type;
srank student.rank%type;
BEGIN
sno := 504;
name := 'Karthik';
srank := 'first';
INSERT into student values(sno, name, srank);
```

```
    dbms_output.put_line('One record inserted');
    COMMIT;
    -- adding savepoint
    SAVEPOINT s1;
    -- second time asking user for input
    sno := 505;
    name := 'Manasa';
    srank := 'second';
    INSERT into student values(sno,name,srank);
    dbms_output.put_line('One record inserted');
    ROLLBACK TO SAVEPOINT s1;
    END;
    /
select * from student;
```

-- Procedure to execute in sql editor

```
SQL>ed 5b
Enter the following code into the text editor and save the file with .sql format
        DECLARE
        sno student.sid%type;
        name student.sname%type;
        srank student.rank%type;
        BEGIN
        sno := &sno;
        name := '&name';
        srank := '&srank';
        INSERT into student values(sno, name, srank);
        dbms_output.put_line('One record inserted');
        COMMIT;
        -- adding savepoint
        SAVEPOINT s1;
        -- second time asking user for input
        sno := &sno;
        name := '&name';
        srank := '&srank';
        INSERT into student values(sno,name,srank);
        dbms_output.put_line('One record inserted');
        ROLLBACK TO SAVEPOINT s1;
        END;
        /
SQL> @5b;
Enter value for sno: 504
old 7: sno := &sno;
new 7: sno := 504;
```

```
Enter value for name: ali
old 8: name := '&name';
new 8: name := 'ali';
Enter value for srank: first
old 9: srank := '&srank';
new 9: srank := 'first';
Enter value for sno: 505
old 16: sno := &sno;
new 16: sno := 505;
Enter value for name: haji
old 17: name := '&name';
new 17: name := 'haji';
Enter value for srank: third
old 18: srank := '&srank';
new 18: srank := 'third';
One record inserted
One record inserted

PL/SQL procedure successfully completed.

SQL> select * from student;
        SID          SNAME          RANK
        ---------- --------------------  ----------
        501         Ravi                second
        502         Raju                third
        503         Ramu                first
        504         ali                 first

select sname, case sname
when 'Ravi' then marks+2
when 'Ramu' then marks+4
else marks
end
"new marks"
from student;
```

Session #7

Control Statements, NULLIF, COALESCE

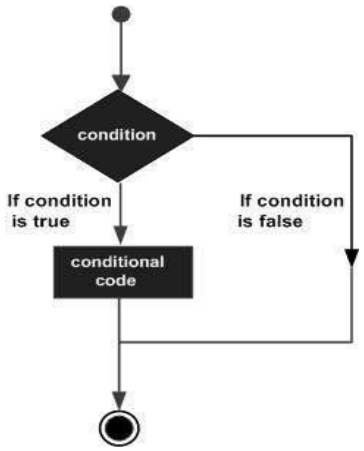
Learning Objective

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

Learning Context

Decision Making Statements in PL/SQL

In PL/SQL. Decision-making structures require that the programmer specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.



IF STATEMENT:

The **IF statement** associates a condition with a sequence of statements enclosed by the keywords **THEN** and **END IF**. If the condition is **TRUE**, the statements get executed, and if the condition is **FALSE** or **NULL**, then the **IF** statement does nothing.

Syntax:

IF *condition* THEN
Executable statement(s)
END IF;

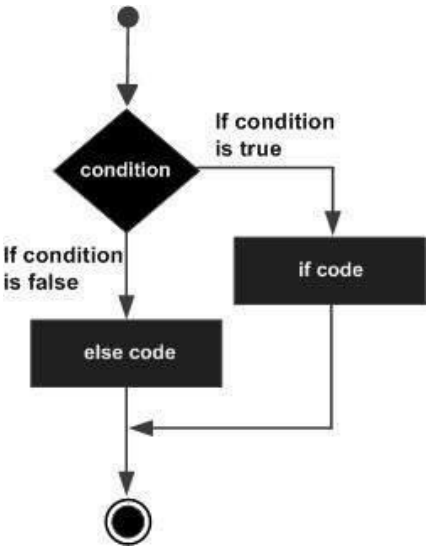
```
DECLARE
   a number(2) := 10;
BEGIN
   a:= 10;
  -- check the boolean condition using if statement
   IF( a < 20 ) THEN
      -- if condition is true then print the following
      dbms_output.put_line('a is less than 20 ' );
   END IF;
   dbms_output.put_line('value of a is : ' || a);
END;
/
```

IF – ELSE STATEMENT

A sequence of **IF-THEN** statements can be followed by an optional sequence of **ELSE** statements, which execute when the condition is **FALSE**.

Syntax

IF *condition* **THEN**
TRUE sequence of_ executable statement(s)
ELSE
FALSE/ NULL sequence_ of_ executable statement(s)
END IF;



ELSIF STATEMENT

The **IF-THEN-ELSIF** statement allows you to choose between several alternatives. An **IF- THEN** statement can be followed by an optional **ELSIF...ELSE** statement. The **ELSIF** clause lets you add additional conditions.

When using **IF-THEN-ELSIF** statements there are a few points to keep in mind.

- It's ELSIF, not ELSEIF.
- An IF-THEN statement can have zero or one ELSE's and it must come after any ELSIF's.
- An IF-THEN statement can have zero to many ELSIF's and they must come before the ELSE.
- Once an ELSIF succeeds, none of the remaining ELSIF's or ELSE's will be tested.

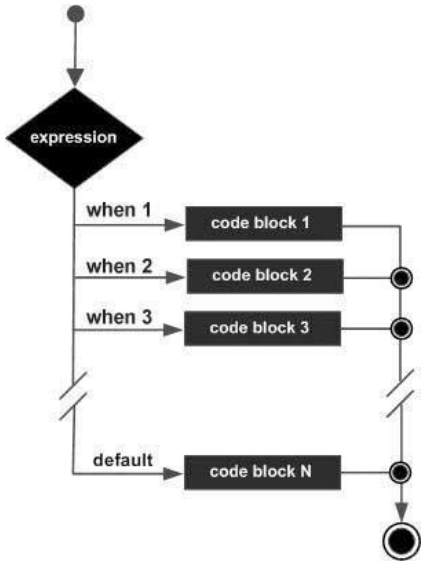
```
IF condition-1 THEN
  Statements-1
ELSIF condition- N THEN
 Statements-N
[ELSE
  ELSE statements]
END IF;
```

CASE STATEMENT

Like the **IF** statement, the **CASE statement** selects one sequence of statements to execute. However, to select the sequence, the **CASE** statement uses a selector rather than multiple Boolean expressions. A selector is an expression, the value of which is used to select one of several alternatives.

Syntax

CASE selector
WHEN expression1 **THEN** result 1
WHEN expression2 **THEN** result 2
...
WHEN expressionN **THEN** result N [**ELSE]** result N+ 1;]
END CASE;



Example

```
DECLARE
  grade char(1) := 'A';
BEGIN
  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('Well done');
    when 'D' then dbms_output.put_line('You passed');
    when 'F' then dbms_output.put_line('Better try again');
    else dbms_output.put_line('No such grade');
  END CASE;
END;
/
```

DATA TYPES

The PL/SQL variables, constants and parameters must have a valid data type, which specifies a storage format, constraints, and a valid range of values.

- **CHAR** [(*maximum_ length*)]
- **VARCHAR2** (*maximum_length*)
- **NUMBER** [(*precision, scale*)]
- DATE
- **% TYPE** Attribute

Declares a variable according to a database column definition or previously declared variable.

Example:

Procedure that displays name of an employee if his empno is given. It uses the % type.

```
CREATE OR REPLACE PROCEDURE pemp1( no emp. empno%type)
IS
emp name emp. ename% type;
BEGIN
     SELECT ename INTO empname FROM emp
     WHERE empno = no;
     DBMS_ OUTPUT. P UT_ LINE (empname);
END;
/
```

NULLIF: Takes two arguments. If the two arguments are equal, then NULL is returned. Otherwise, the first argument is returned.

Syntax

select column_name, NULLIF(argument1,argument2) from table_name;

Example

select ename, nullif('sita','sita1') from emp;

ENAME	NUL
Ram	Sita
Sita	Sita
Lakshman	Sita
Hanuma	Sita

select ename, nullif('Sita','Sita') from emp;

ENAME	NUL
Ram	
Sita	
Lakshman	
Hanuma	

COALESCE: COALESCE () function accepts a list of arguments and returns the first one that evaluates to a non-null value.

Syntax

coalesce("expression1","expression2",...);

Example

SQL> select coalesce(NULL,'DBMS','OS',’JAVA’) from dual;

COALE
DBMS

Learning Outcome

After undergoing this laboratory module, the student should be able to write PL/SQL blocks using control statements (if, case) and the significance of nullif and coalesce functions.

Exercise

1. Write a pl/sql program to find largest of three numbers using nested_if.
2. Write a pl/sql program to find appropriate grade using case statement.
3. Write a pl/sql program to find whether given character is vowel or not using case statement.
4. Implement NULLIF and COALESCE functions.

Session #8

Loop Statements, Exceptions

Learning Objective

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

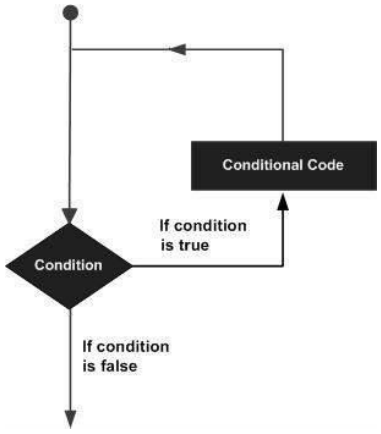
Learning Context

PL/SQL LOOPS

There may be a situation when you need to execute a block of code several number of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.

Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages −



Basic loop structure encloses sequence of statements in between the **LOOP** and **END LOOP** statements. With each iteration, the sequence of statements is executed and then control resumes at the top of the loop.

Syntax

```
LOOP
   Sequence of statements;
END LOOP;
```

Here, the sequence of statement(s) may be a single statement or a block of statements. An **EXIT statement** or an **EXIT WHEN statement** is required to break the loop.

Example

```
DECLARE
   x number := 10;
BEGIN
   LOOP
      dbms_output.put_line(x);
      x := x + 10;
      IF x > 50 THEN
         exit;
      END IF;
   END LOOP;
   -- after exit, control resumes here
   dbms_output.put_line('After Exit x is: ' || x);
END;
/
```

FOR LOOP

A **FOR LOOP** is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

```
FOR loop_index IN [REVERSE] lowest_number.. highest_
number
LOOP
      executable_statement(s)
END LOOP;
```

The REVERSE keyword causes PL/SQL to start with the highest_ number and decrement down to the lowest_ number

Example

```
DECLARE
   a number(2);
BEGIN
   FOR a in 10 .. 20 LOOP
      dbms_output.put_line('value of a: ' || a);
  END LOOP;
END;
/
```

Reverse FOR LOOP Statement

By default, iteration proceeds from the initial value to the final value, generally upward from the lower bound to the higher bound. You can reverse this order by using the **REVERSE** keyword. In such case, iteration proceeds the other way. After each iteration, the loop counter is decremented.

Example

```
DECLARE
   a number(2) ;
BEGIN
   FOR a IN REVERSE 10 .. 20 LOOP
      dbms_output.put_line('value of a: ' || a);
   END LOOP;
END;
/
```

WHILE STATEMENT

A **WHILE LOOP** statement in PL/SQL programming language repeatedly executes a target statement as long as a given condition is true.

```
WHILE condition
LOOP
      executable_statement(s)
END LOOP;
```

Example

```
DECLARE
   a number(2) := 10;
BEGIN
   WHILE a < 20 LOOP
      dbms_output.put_line('value of a: ' || a);
      a := a + 1;
   END LOOP;
END;
/
```

Raising Exceptions

Exceptions are raised by the database server automatically whenever there is any internal database error, but exceptions can be raised explicitly by the programmer by using the command RAISE.

Following is the simple syntax for raising an exception

DECLARE
exception_name **EXCEPTION;**
BEGIN
IF condition THEN
RAISE exception_name;

END IF;
EXCEPTION
WHEN exception_name THEN
statement;
END;

User-defined Exceptions

PL/SQL allows you to define your own exceptions according to the need of your program. A user-defined exception must be declared and then raised explicitly, using either a RAISE statement or the procedure.

DBMS_STANDARD.RAISE_APPLICATION_ERROR.

The syntax for declaring an exception is

```
DECLARE
  my-exception EXCEPTION;
```

Pre-defined 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.

The following table lists few of the important pre-defined exceptions.

Exception	Oracle Error	SQLCODE Value
ACCESS_INTO_NULL	ORA-06530	-6530
CASE_NOT_FOUND	ORA-06592	-6592
COLLECTION_IS_NULL	ORA-06531	-6531
CURSOR_ALREADY_OPEN	ORA-06511	-6511
DUP_VAL_ON_INDEX	ORA-00001	-1
INVALID_CURSOR	ORA-01001	-1001
INVALID_NUMBER	ORA-01722	-1722
LOGIN_DENIED	ORA-01017	-1017
NO_DATA_FOUND	ORA-01403	+100
NOT_LOGGED_ON	ORA-01012	-1012

PROGRAM_ERROR	ORA-06501	-6501
ROWTYPE_MISMATCH	ORA-06504	-6504
SELF_IS_NULL	ORA-30625	-30625
STORAGE_ERROR	ORA-06500	-6500
SUBSCRIPT_BEYOND_COUNT	ORA-06533	-6533
SUBSCRIPT_OUTSIDE_LIMIT	ORA-06532	-6532
SYS_INVALID_ROWID	ORA-01410	-1410
TIMEOUT_ON_RESOURCE	ORA-00051	-51
TOO_MANY_ROWS	ORA-01422	-1422
VALUE_ERROR	ORA-06502	-6502
ZERO_DIVIDE	ORA-01476	-1476

Learning Outcome

After undergoing this laboratory module, the student should be able to write PL/SQL blocks using loop statements (while, for) and the significance of exception handling by creating and raising the exceptions

Exercise

1. Write a PL/SQL program to print the sum of N natural numbers using while loop.
2. Write a PL/SQL program to print the sum of even numbers using for loop.
3. Write a PL/SQL program to print prime numbers upto given N value using for loop.
4. Write a PL/SQL program to print a pyramid of stars using a nested for loop.
5. Write a PL/SQL program to show pre_defined exception ZERO_DIVIDE
6. Write a PL/SQL program to show pre_defined exception CASE_NOT_FOUND
7. Write a PL/SQL program to Raise user_defined exception

Session #9

Procedures

Learning Objective

To create and execute procedures passing IN and OUT parameters in PL/SQL.

Learning Context

Procedures: Procedures are standalone blocks of a program that can be stored in the database.

- Can receive multiple input parameters and return multiple output values (or no output values)
- Can perform an action such as insert, update or delete record

Syntax of the named blocks (procedures and functions)

```
CREATE [OR REPLACE ] PROCEDURE [ owner_ name.]
 procedure_ name [(parameter1 [ IN | OUT | IN OUT ]
datat ype][,... n)]]
{IS | AS }
PL/SQL block ;
```

In an Oracle stored procedure, the specified arguments and parameters include **IN, OUT,** or **IN OUT.** The parameter mode specifies whether a parameter can be read from or write to.

IN

An **IN** parameter is **read-only.** You can reference an IN parameter inside a procedure, but you cannot change its value. Oracle uses IN as the default mode. It means that if you don't specify the mode for a parameter explicitly, Oracle will use the IN mode.

OUT

An **OUT** parameter is **writable**. Typically, you set a returned value for the OUT parameter and return it to the calling program. Note that a procedure ignores the value that you supply for an OUT parameter.

INOUT

An **INOUT** parameter is both **readable and writable.** The procedure can read and modify it.

Note that **OR REPLACE** option allows you to overwrite the current procedure with the new code.

Example:

Procedure that computes and displays tax to be paid if income is passed as parameter

```
CREATE OR REPLACE PROCEDURE tax1( p_ value IN NUMBER)
IS
BEGIN
  DBMS_ OUTP UT. PUT_LINE ( p_value * 0. 08 );
END;
```

Execution

Exec tax1(1000);

output:80

Learning Outcome

After undergoing this laboratory module, the student should be able to write PL/SQL procedures.

Exercise

1. Create a procedure to insert a new record into the emp table using the IN parameter.
2. Create a procedure to print the name of an employee if his id is given using OUT parameter.

Session #10

Functions

Learning Objective

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

Learning Context

A function is similar to a procedure, except that it returns a single value

Syntax of the named blocks (functions)

```
CREATE [OR REPLACE ] FUNCTION [ owner_ name.]
 function_ name [(parameter1 [ IN | OUT | IN OUT]
datat ype][,... n)]]
 RETURN datatype
{IS | AS }
PL/SQL block;
```

Example:

Function that returns tax to be paid if income is passed as parameter

```
CREATE OR REPLACE FUNCTION tax2( p_ value IN NUMBER)
RETURN NUMBER
IS
BEGIN
  RETURN ( p_value * 0.08);
END;
```

Execution

a. select tax2(1000) from dual;

Or

b. Execute Following sequence of statements for getting the value returned by the function into bind variable and print it.

```
variable v1 number;
exec :v1:=tax2(1000);
print v1;
```

In Oracle, user- defined functions and stored procedures are very similar in composition and structure. The primary difference is that stored procedures cannot return a value to the invoking process, while a function may return a single value to the invoking process.

The **IN** qualifier is provided when invoking the function and passes a value in to the function, while **OUT** arguments pass a value back to the invoking process. In other words, the **IN** qualifier is supplied by the user or process that calls the function, while the **OUT** argument is returned by the function. **IN OUT** arguments perform both **IN** and **OUT** functionality.

Learning Outcome

After undergoing this laboratory module, the student should be able to write PL/SQL functions.

Exercise

1. Write a Function in PL/SQL to take your date_of_birth as input to your function and display the calculated age.
2. Write a Function in PL/SQL to find the factorial of a given number using the IN parameter.
3. Write a Function in PL/SQL to find the reverse of a number using the INOUT parameter.

Session #11

CURSORS

Learning Objective

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

Learning Context

CURSORS:
The syntax of cursor declaration is

DECLARE CURSOR cursor_name
IS
Select_ statement
[**FOR UPDATE** [**OF** column_ name [,... n]]}]

The **DECLARE CURSOR** command enables the retrieval and manipulation of records from a table one row at a time. This provides row-by-row processing, rather than the traditional set processing offered by SQL. To use this procedure properly, you should:

- **DECLARE** the cursor
- **OPEN** the cursor
- **FETCH** rows from the cursor
- When finished, **CLOSE** the cursor

The DECLARE CURSOR command works by specifying a SELECT statement. Each row returned by the SELECT statement may be individually retrieved and manipulated. In Oracle, variables are not allowed in the WHERE clause of the SELECT statement unless they are first declared as variables. The parameters are not assigned at the DECLARE. Instead, they are assigned values at the OPEN command.

Example

The following code creates procedure to display empno and ename of all employees in emp table using cursors.

```
CREATE OR REPLACE PROCEDURE pemp2
IS
       CURSOR c IS
       SELECT emp no, ename
       FROM   emp;
       no emp. empno% type;
       name emp. ename % type;
BEGIN
       OPEN c;
  LOOP
       FETCH c into no, name;
       EXIT WHEN c% notfound;
       DBMS_OUTPUT.PUT_LINE  ( TO_CHAR( no)||'-->'||
       name);
  END LOOP;
CLOSE c;
END;
/
```

Example:

The following code creates procedure to update salaries of employees using cursors

```
CREATE OR REPLACE PROCEDURE p2
IS
       CURSOR c1 IS
       SELECT deptno, empno, ename, sal
       FROM   emp FOR UPDAT E OF sal;
BEGIN
       FOR emp_ record IN c1
       LOOP
              IF emp_ record.sal>
              2000 THEN UPDATE emp
              SET  sal= emp_ record. sal*  1. 10
              WHERE CURRENT OF c1 ;
              END IF;
       END LOOP;
 END ;
```

Learning Outcome

After undergoing this laboratory module, the student should be able to manipulate record by record data using cursors.

Exercise

1. Create a procedure to display empno and ename of all employees in emp table using cursors.
2. Create a procedure to update salaries of employees using cursors

Session #12

Triggers

Learning Objective

To create triggers and to study the behavior of triggers.

Learning Context

A trigger is a special kind of stored procedure that fires automatically (hence, the term trigger) when a data- modification statement is executed. Triggers are associated with a specific data- modification statement (INSERT, UPDATE, or DELETE) on a specific table.

Syntax and Description:

```
CREATE [ OR REPLACE] TRIGGER trigger_ name
{ BEFORE | AFTER | INSTEAD OF}
{[DELETE ] [ OR ] [ INSERT ] [ OR ] [ UPDATE [ OF column
[,... n] ]] ON
{table_name | view_ name}
[ FOR EACH { ROW | STATEMENT }]
[ WHEN (conditions)]
code block
```

Triggers, by default, fire once at the statement level. That is, a single INSERT statement might insert 500 rows into a table, but an insert trigger on that table fires only one time. Some vendors allow a trigger to fire for each row of the data- modification operation. So, a statement that inserts 500 rows into a table that has a row- level insert trigger fires 500

In addition to being associated with a specific data- modification statement (INSERT, UPDATE, or DELETE) on a given table, triggers are associated with a specific time of firing. In general, triggers can fire BEFORE the data- modification statement is processed, AFTER it is processed, or (when supported by the vendor) INSTEAD OF processing the statement.

Triggers that fire before or instead of the data- modification statement do not see the changes that the statement renders, while those that fir e afterwards can see and act upon the changes that the data- modification statement renders. It may be noted that Oracle allows INSTEAD OF triggers to process only against views, not / tables.

Examples:

The first trigger is invoked whenever there is an update operation on the emp table rows and if salary is changed. It records these changes in another table to know the changes made to the table. The second trigger is invoked whenever there is an insert, delete or update operation on the emp1 table rows and the changes are recorded in the empl and empl1 tables.

```
CREATE OR REPLACE TRIGGER EMPT1
 BEFORE UPDATE ON EMP
 FOR EACH ROW
 WHEN ( NEW.SAL <> OLD.SAL) BEGIN
 INSERT INTO EMPLOG VALUES (' UPDATED', OLD. EMPNO,
: OLD. SAL,: NEW. EMPNO,: NEW. SAL);
 END;
```

```
 CREATE OR REPLACE TRIGGER EMPT2
 AFTER DELETE OR UPDATE OR INSERT ON EMP 1
 FOR EACH ROW
 BEGIN
 IF DELETING THEN
 INSERT INTO EMPL VALUES (' DELETED', : OLD. EMPNO, :
 OLD. SAL);
 ELSIF INSERTING THEN
 INSERT INTO EMPL VALUES (' ISERTED',: NEW. EMPNO,
 : NEW. SAL);
 ELSE
 INSERT INTO EMPL1 VALUES (' UPDATED',: OLD. EMPNO,
 : OLD. SAL,: NEW. EMPNO,: NEW. SAL);
 END
  IF;
```

Learning Outcome

After undergoing this laboratory module, the student should be able to write Triggers to maintain database consistency at the time of manipulation.

Exercise

1. Create a trigger on 'dbms' table whenever there is an insertion of a row. If the marks are < 0 insert 0 and insert 100 when marks is > 100.
2. Create a trigger which inserts the changes made into a dbms_change table on dbms table before there is a deletion or modification of a row.
3. Create a trigger on account table which is invoked whenever an account is created or an amount is withdrawn such that the balance should be always greater than or equal to 500.
4. Create a trigger on account table which is invoked after an amount is deposited to display the amount of balance in the account with account number.

APPENDIX

Creation of Forms and Reports in MS-Access

Learning Objective

To familiarize with MS Access, open an existing database, create new data base, to create forms and reports in MS - Access.

Learning Context

MS Access is a relational database that facilitates the storage and retrieval of structured information. Access databases are, ideally, a set of tables that are related in one way or another. Along with tables,

Access allows us to create:

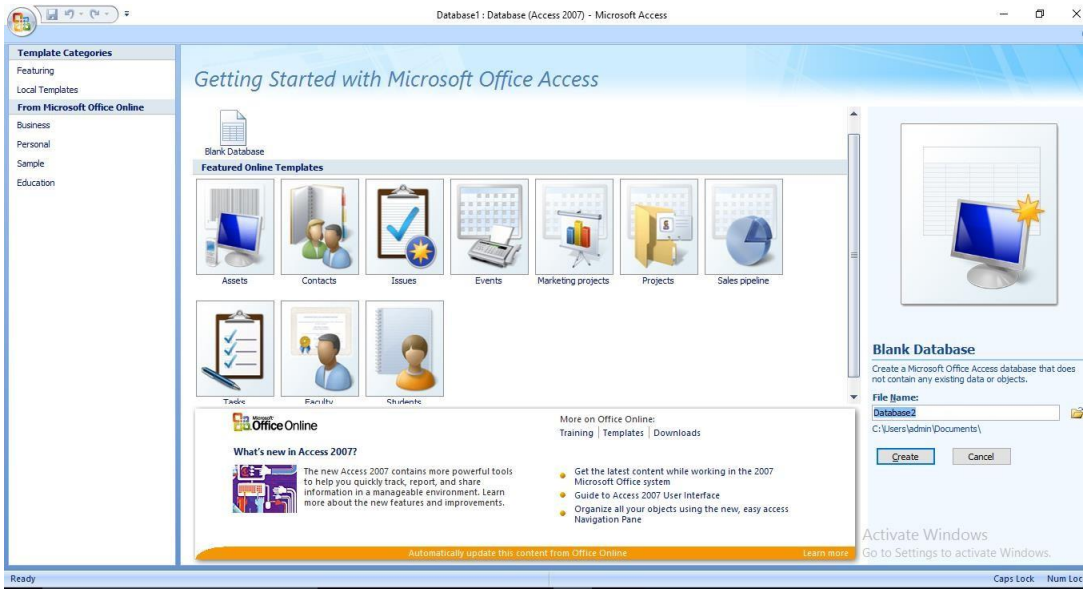
- **Query-**organizing or collecting a particular set of data
- **Form-**interfaces that allow you to maintain or edit records/ data
- **Report-**printable results
- **Macros-**extended functionality for the database, an advanced function all of these objects is stored in a single file with ". mdb" file extension.

PROCEDURE:

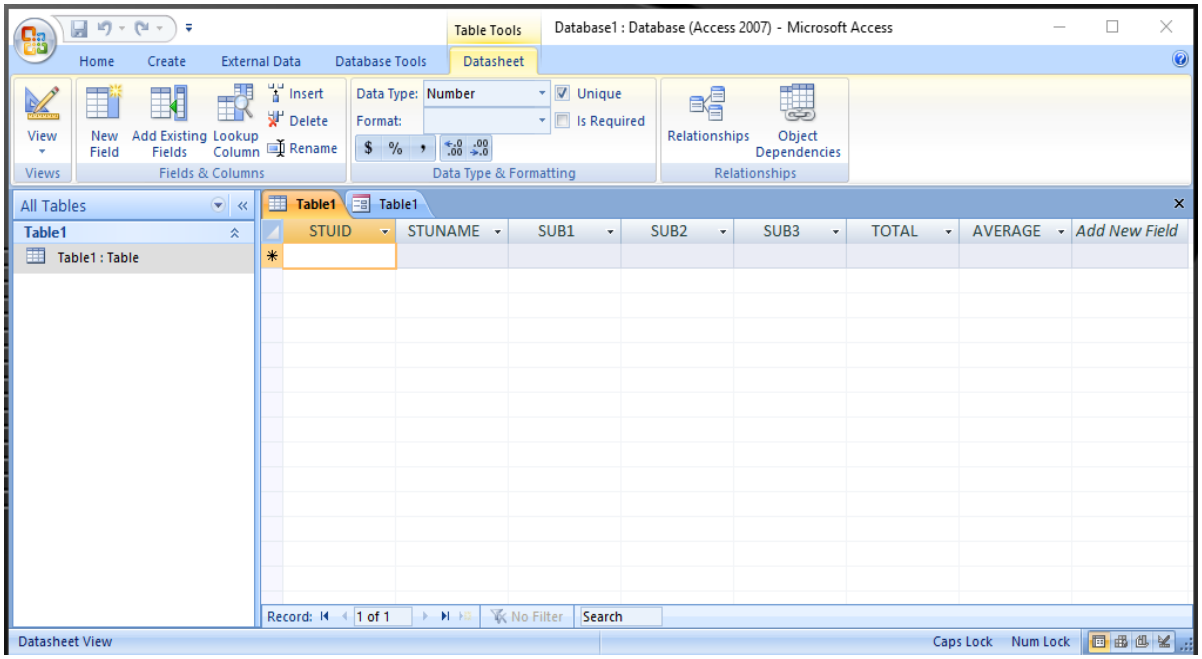
Part-I Opening an Existing database:

- Click on the Windows Start button, select All Programs, and then click Microsoft Access.
- Click on the FILE menu, then open and select the .mdb file to be opened and then click open. Then the following screen appears
- Double click the t able to open the table.
- The t able contents are displayed as shown below.
- Then we navigate to the required record using the record navigation buttons.
- To add a record of data, move to the first empty record and type in the data values and press TAB to move to the next field.
- To save a record of data move to the first empty field below these records. We do not require to do anything else to save your data. When we l eave a record, either by moving to another record or by closing a table, Microsoft access automatically saves the data.
- Edit a record after moving to the required record and use **TAB and SHIFT+TAB** to move to next field and previous field.

- Go to **FILE** menu and click exit to exit from Access.



Data Sheet View



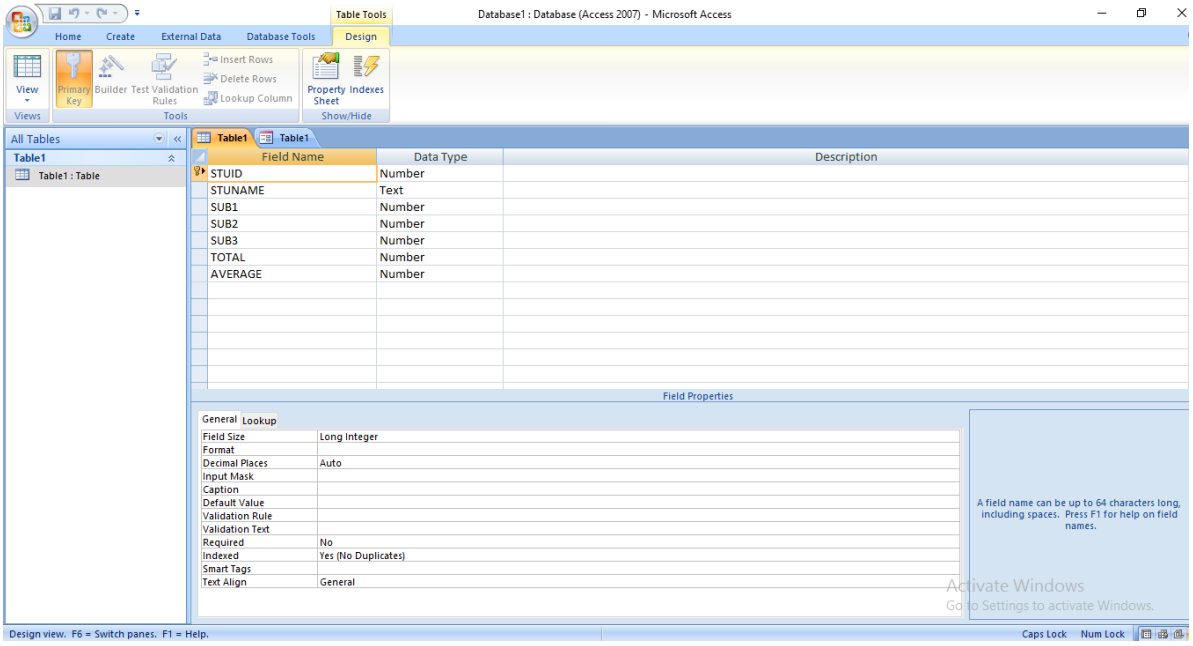
Part II Creating a new database:

1. Click on the Windows Start button, select All Programs, and then click Microsoft Access.
2. Click on the FILE menu, then new.
3. Click blank database option.
4. Name and save the database.
5. Create t able by double- clicking the create table in design view option. Then the following window appears.
6. Enter the field names and data types.

7. Enter the field width in the field property window.
8. Right click the field which uniquely identifies a record in the table and click primary key option.
9. Set the other properties like
   - Required yes or no depending on whether the field value can be set to null or not null.
   - Validation rule like <= 100 for ensuring value in the marks field to be not greater than 100.
10. Now save it with a new table name for example table1.
11. Close the design view and double click the table1 object to enter the data.
12. After editing is completed close the data sheet view and exit from MS Access.

Note: - we can also create table using create table by wizard or create table by entering data.

Design view of a table



`Data Type Property of a table field`

You can use the **Data Type** property to specify the type of data stored in atable field. Each field can store data consisting of only a single data type.The **Data Type** property uses the following settings.

Setting	Type of data	Size
Text	(Default) Text or combinations of text and numbers, as well as numbers that don't require calculations, such as phone numbers.	Up to 255 characters or the length set by the **Field Size** property, whichever is less? Microsoft Access does not reserve space for unused

		portions of a text field.
Memo	Lengthy text or combinations of text and numbers.	Up to 65, 535 characters.(If the Memo field is manipulated through DAO and only text and numbers [not binary data] will b e stored in it, then the size of the Memo field is limited b y the size of the database.)
Number	Numeric data used in mathematical calculations	1 , 2, 4, or 8 bytes (16 bytes if the **Field Size** property is set to Replication ID).
Date/ Time	Date and time values for the years 100 through 9999.	8 bytes.
Currency	Currency values and numeric data used in mathematical calculations involving data with one to four decimal places. Accurate to 15 digits on the left side of the decimal separator and to 4 digits on the right side.	8 bytes.
Auto Number	A unique sequential (incremented by 1) number or random number assigned by Microsoft Access whenever a new record is added toa table. Auto Number fields can't be updated. For more information, see the **New Values** property topic.	4 bytes (16 bytes if the **Field Size** property is set to Replication ID).
Yes/ No	Yes and No values and fields that Contain only one of two values (Yes/ No, True**/** False, or On/ Off).	1 bit.
OLE Object	An object (such as a Microsoft Excel spreadsheet, a Microsoft Word document, graphics, sounds, or other binary data) l inked to or embedded in a Microsoft Access table.	Up to 1 gb disk space)

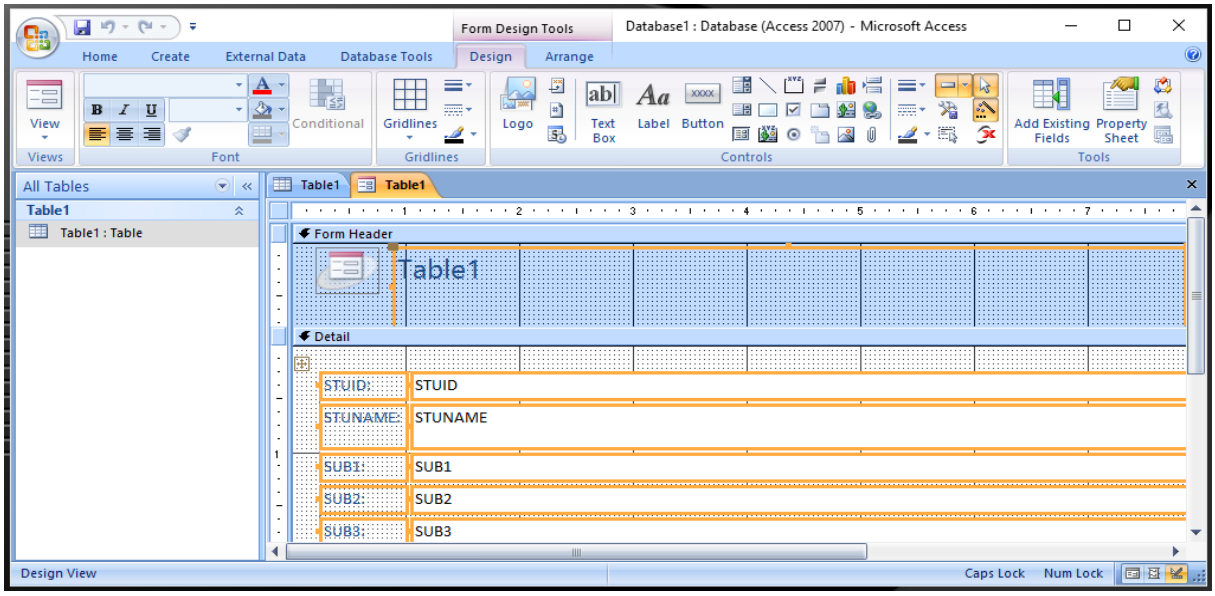
Hyperlink	Text or combinations of text and numbers stored as text and used as A hyperlink address. A hyperlink address can have up to three p arts: text to display – the text that appears in a field or control. Address: the path to a file (UNC path) or page (URL). Sub address: a location within the file or page. Screen tip: the text displayed as a tooltip. The easiest way to insert a hyperlink address in a field or control is to click Hyperlink on The Insert menu.	Each part of the three parts of a Hyperlink data type can contain up to 2048 characters Each part of the three parts of a Hyperlink data type can contain up to 2048 characters..
Lookup Wizard	Creates a field that allows you to choose a value from another table or from a list of values by using a list box or combo box. Clicking this option starts the Lookup Wizard, which creates a Lookup field. After you complete the wizard, Microsoft The same size as the primary key field used to perform the lookup, typically y 4 bytes. Access sets the data type based on the values selected in the wizard.	The same size as the primary key field used to perform the lookup, typically y 4 bytes.

The form is a formatted display where data is entered, displayed and edited.It provides greater flexibility than a table. A report is a like a query but is formatted for printing. As the data is entered data into a form simultaneously data is adding to the table. In man y cases the tables can become quite large and difficult to read, especially when we only want information from individual records. An easier way to view the information in a table is to use a form which is easily created in Access. Microsoft Access divides a form into five sections in the Design view.

The **form header** prints at the top of the first page. When we are viewing data, the form header appears once at the top of the window.

- The **page header** prints at the top of every page
- The p age header only app ears when printed or in print preview.
- The **detail** section contains the fields from the t able. When we are viewing data, the detail section is repeated for each record. When we print the form, the detail section shows as many records as will fit on a page.
- The **page footer** prints at the bottom of every page.
- The p age footer only appears when printed or in print preview.
- The **form footer** prints at the bottom of every page. When we are viewing data, the footer appears once at the bottom of the window.

Design view of Form:



We can use a report to present data in print. With a report, we have greater flexibility to present summary information than with a form. For instance, we can include totals across an entire set of records in a report.

TOOL BOX:

The tool b ox is special kind of tool bar. Control is a graphical user interface object, such as a text box, check box, scroll bar, or command button, that lets users control the program. We use controls to display data or choices, perform an action, or make the user interface easier to read Open a form, report, or data access page in Design view. Toolbox is a set of tools that are availablein Design view to add controls to a form, report, or data access page. To create a control using the tool box click the tool for the control we want to create and then dragit and drop it.

TEXT BOXES

We use text boxes on a form, rep ort, or data access to display data from a record source. This type of text box is called a bound text box because it's bound to data in a field. Text boxes can also be unbound. For example, we can create an unbound text box to display the results of a calculation or to accept input from a user. Data in an unbound text box isn't stored anywhere.

LABELS

We use labels on a form, report, to display descriptive text such as titles, captions, or brief instructions.

LIST BOXES

The list in a list b ox consists of rows of data. In a form, a list box can have one or more columns, which can appear with or without headings. If a multiple- columnlist b ox is bound .Bound control is a control used on a form, rep ort, or data access page to enter or display the contents of a field in the underlying table, query, or SQL statement. The control's Control Source property stores the field name to which the control is bound. Access stores the values from one of the columns. In a data access page, a list box has on e column without a heading.

COMBO BOXES

A combo box is like a text box and a list box combined, so it requires less room. We can type new values in it, as well as select values from a list.

COMMAND BUTTONS

Command buttons provide us with a way of performing action(s) by simply clicking them. When we choose the button, it not only carries out the appropriate action, it also looks as if it's being pushed in and released. We use a command button on a form to start an action or a set of actions. For example, we can create a command button that opens another form. To make a command button do something on a form, we write an event procedure and attach it to the button's On Click property.

CHECK BOXES

We can use a check box on a form, report, or as a stand- alone to display a Yes/No value from an underlying table, query, or SQL statement.

OPTION BUTTONS

We can use an option button on a form, report, or data access page as a stand- alone control to display a Yes/ No value from an underlying record source. We can also use option buttons in an option group to display values to choose from.

The option group is the frame that surrounds the controls inside it. Only one option in an option group can be selected at a time. If an option group is bound to a field, only the group frame itself is bound not the check boxes, toggle buttons, or option buttons inside the frame. Because the Control Source property of the group frameis set to the field that the option group is bound to, we don't set the Control Source property for each in the option group. Instead, we set the Option Value (form or report) or the Value(data access Page) property of each check box, toggle button, or option button. In a form or report, set the control property to a number that's meaningful for the field the group frame is bound to. In a data access page, set the control property to either a number or any text that's meaningful for the field the group frame is bound to. When we select an option in an option group, Access sets the value of the field to which the option group is bound to the value of the selected option's Option Value or Value property.

TOGGLE BUTTONS

We can use a toggle button on a form as a standalone to display a Yes/ No value from an underlying record source.

TABBED PAGES ON FORMS

We can use a tab control to present several pages of information as a single set. This is especially useful when we're working with many controls that can be sorted into two or more categories. For example, we might use a tab control on an Employees form to separate employment history and personal information. Information about employment history is displayed on this page. Personal information, such as home address and phone number, is displayed on this page.

Procedure:-

Part-I

CREATION OF FORMS:

1. Click on **Forms** in the Objects column and select either the wizard or design view.
2. The wizard asks which table the form is to come from, and then which fields you want. Give the table name and select the fields and move to the selected fieldlist by clicking right arrow button.
3. Select a layout, style, and a title to complete your form. This then opens in form view for you to begin entering or viewing information.
4. Change to design view to customize.

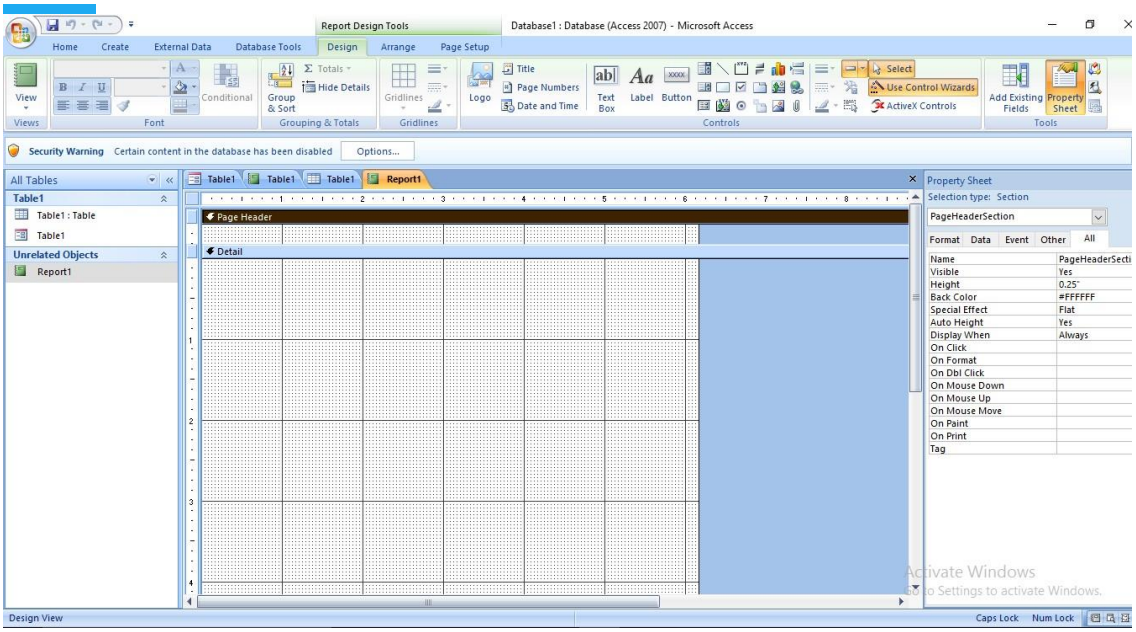
Part II

CREATION OF REPORTS:

1. Click **Report** in the Object column and select the wizard or design view. The wizard asks which t able or query you want to print.
2. Give the table name and then select the fields to print.
3. Select grouping levels **grouping** brings entries together in the same category, say all people living in Dallas.
4. Select fields to sort and whether to sort ascending or descending order. Up to four fields can be sorted.
5. Then select the layout i.e. columnar, tabular, or justified.
6. Finally, select a style and a title.
7. The report opens in print preview.
8. Switch to design view to change the formatting.

DESIGN VIEW OF A REPORT

When we print the report, sections are repeated, as appropriate, until all the data in the report is printed. The controls in each section tell Microsoft Access what data to print in the section. The Report Header prints at the beginning of the report. The Page Header and Footer prints on every page. The Category Header and Footer prints for each category. The Details section prints for each record in the category Controls that we can use to display, enter, filter, or organize data in Access.



Learning Outcome

After undergoing this laboratory module, the student should be able to create database and generate forms and reports in MS-ACCESS.

Exercise

1. Create a form for entering the data into student table which is already created in the previous experiment.
2. Create a form for each subject teacher to enter marks in the concerned subject.
3. Create a report to head of the department to view all subject marks of the students.
4. Create a report displaying roll list (roll number and name of the students) from the student table.

My SQL

Learning Objective

To practice simple MySQL commands for retrieving data.

Learning Context

MySQL is primarily an RDBMS and ships with no GUI tools to administer MySQL databases or manage data contained within the databases. Users may use the included command line tools,
MySQL uses all the standard ANSI SQL numeric data types, so if you're coming to MySQL from a different database system, these definitions will look familiar to you. The following list shows the common numeric data types and their descriptions.

Numeric Data Types:

INT - A normal-sized integer that can be signed or unsigned. If signed, the allowable range is from -2147483648 to 2147483647. If unsigned, the allowable range is from 0 to 4294967295. You can specify a width of up to 11 digits.

TINYINT - A very small integer that can be signed or unsigned. If signed, the allowable range is from -128 to 127. If unsigned, the allowable range is from 0 to 255. You canspecify a width of up to 4 digits.

SMALLINT - A small integer that can be signed or unsigned. If signed, the allowable range is from -32768 to 32767. If unsigned, the allowable range is from 0 to 65535. You can specify a width of up to 5 digits.

MEDIUMINT - A medium-sized integer that can be signed or unsigned. If signed, the allowable range is from -8388608 to 8388607. If unsigned, the allowable range is from 0 to 16777215. You can specify a width of up to 9 digits.

BIGINT - A large integer that can be signed or unsigned. If signed, the allowable range is from -9223372036854775808 to 9223372036854775807. If unsigned, the allowable rangeis from 0 to 18446744073709551615. You can specify a width of up to 11 digits.

FLOAT(M,D) - A floating-point number that cannot be unsigned. You can define the display length (M) and the number of decimals (D). This is not required and will default to 10,2, where 2 is the number of decimals and 10 is the total number of digits (including decimals). Decimal precision can go to 24 places for a FLOAT.

DOUBLE(M,D) - A double precision floating-point number that cannot be unsigned. Youcan define the display length (M) and the number of decimals (D). This is not required and will

default to 16,4, where 4 is the number of decimals. Decimal precision can go to 53 places for a DOUBLE. REAL is a synonym for DOUBLE.

DECIMAL(M,D) - An unpacked floating-point number that cannot be unsigned. In unpacked decimals, each decimal corresponds to one byte. Defining the display length (M) and the number of decimals (D) is required. NUMERIC is a synonym for DECIMAL.

Date and Time Types:

DATE - A date in YYYY-MM-DD format, between 1000-01-01 and 9999-12-31. For example, December 30th, 1973 would be stored as 1973-12-30.

DATETIME - A date and time combination in YYYY-MM-DD HH:MM:SS format, between 1000-01-01 00:00:00 and 9999-12-31 23:59:59. For example, 3:30 in the afternoon on December 30th, 1973 would be stored as 1973-12-30 15:30:00.

TIMESTAMP - A timestamp between midnight, January 1, 1970 and sometime in 2037. This looks like the previous DATETIME format, only without the hyphens between numbers; 3:30 in the afternoon on December 30th, 1973 would be stored as 19731230153000(YYYYMMDDHHMMSS).

TIME - Stores the time in HH:MM:SS format.

YEAR(M) - Stores a year in 2-digit or 4-digit format. If the length is specified as 2 (for example YEAR(2)), YEAR can be 1970 to 2069 (70 to 69). If the length is specified as 4, YEAR can be 1901 to 2155. The default length is 4.

String Types:

Although numeric and date types are fun, most data you'll store will be in string format. This list describes the common string datatypes in MySQL.

CHAR(M) - A fixed-length string between 1 and 255 characters in length (for example CHAR(5)), right-padded with spaces to the specified length when stored. Defining a length is not required, but the default is 1.

VARCHAR(M) - A variable-length string between 1 and 255 characters in length; for example VARCHAR(25). You must define a length when creating a VARCHAR field.

BLOB or TEXT - A field with a maximum length of 65535 characters. BLOBs are "Binary Large Objects" and are used to store large amounts of binary data, such as images or other types of files. Fields defined as TEXT also hold large amounts of data; the difference between the two is that sorts and comparisons on stored data are case sensitive on BLOBs and are not case sensitive in TEXT fields. You do not specify a length with BLOB or TEXT.

TINYBLOB or TINYTEXT - A BLOB or TEXT column with a maximum length of 255 characters. You do not specify a length with TINYBLOB or TINYTEXT.

MEDIUMBLOB or MEDIUMTEXT - A BLOB or TEXT column with a maximum length of 16777215 characters. You do not specify a length with MEDIUMBLOB or MEDIUMTEXT.

LONGBLOB or LONGTEXT - A BLOB or TEXT column with a maximum length of 4294967295 characters. You do not specify a length with LONGBLOB or LONGTEXT. ENUM - An enumeration, which is a fancy term for list. When defining an ENUM, you arecreating a list of items from which the value must be selected (or it can be NULL). Forexample, if you wanted your field to contain " A" or " B" or " C", you would define yourENUM as ENUM ('A', 'B', 'C') and only those values (or NULL) could ever populate thatfield.

SYNTAX FOR PROCEDURE AND FUNCTION MySQL

```
CREATE [DEFINER = { user | CURRENT_USER }] PROCEDURE sp_name
([proc_parameter[,...]]) [characteristic ...]
routine_body


CREATE [DEFINER = { user | CURRENT_USER }] FUNCTION sp_name
([func_parameter[,...]])
RETURNS type
[characteristic ...] routine_body
func_parameter:
        param_name type
type:
        Any valid MySQL data type
```

Characteristic:

```
        COMMENT 'string'
      | LANGUAGE SQL
      | [NOT] DETERMINISTIC
      | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
      | SQL SECURITY { DEFINER | INVOKER }
routine_body:
        Valid SQL routine statement
```

PARAMETERS OF PROCEDURE

The real benefit of a stored procedure is of course when you can pass values to it, as well as receive values back. The concept of parameters should be familiar to anyone who has had experience with any procedural programming experience.
There are three types of parameter:

IN: The default. This parameter is passed to the procedure, and can change inside the procedure, but remains unchanged outside.
OUT: No value is supplied to the procedure (it is assumed to be NULL), but it can be

modified inside the procedure, and is available outside the procedure.
INOUT: The characteristics of both IN and OUT parameters. A value can be passed to the procedure, modified there as well as passed back again.

Example of database creation in My SQL:

Creating a new user and giving all privileges to that user to access
database. C:\xampp\ mysql> cd bin
C:\xampp\ mysql\bin> mysql – u root

Welcome to the My SQL monitor. Commands end with ; or \ g.
Your MySQL connection id is 1
Server version: 5. 1. 41 Source distributions
Type _ help;‘ or _ \h‘ for help. Type _ \c‘ to clear the current input
statement mysql> show databases;

The contents of the databases will be displayed on the screen as follows:

Database
information_ schema cd col
mysql

phpmy
admin
test
5 rows in set (0. 03 sec)

Mysql> use mysql; Database
changed mysql> show
tables;

Tables in mysql
columns_ priv db
event
tables_ priv
time_ zone
time_ zone_leap_ second
time_ zone_ name
time_ zone_ tranition
time_ zone_ transition_ type user
23 rows in set(0. 05 sec)

To create a user in mysql :
mysql> **insert into user(host, user, passwork, select_ priv) values(‘,‘ raju‘, password(_ raju‘),‘ Y‘);**
Query OK, 1 row affected, 3 warnings (0. 01 sec)
mysql> select user, password from user ;
mysql> flush privileges;

Query OK, 0 rows affected(0. 00 sec)
Mysql> create database db;
Query OK, 1 row affected (0. 00 sec)
mysql> show databases;

Database

information_ schema cd col
mysql
php
my
admin
test
db
6 rows in set (0. 00 sec)

mysql> grant usage on db.* to raju@localhost identified by _ raju‘;
Query OK, 0 rows affected (0. 00 sec)

mysql> grant all on sam.* to raju@localhost;
Query OK, 0 rows affected (0. 00 sec)

C:\xampp\ mysql\bin> mysql – u raju – p
Enter password:****
Welcome to the My SQL monitor. Commands end with ; or \ g.
Your MySQL connection id is 2
Serverversion: 5 . 1. 41 Source distribution
Type _ help;‘ or _ \h‘ for help. Type _ \c‘
to clear the current input statement.

mysql> show databases;

Database

information_ schema db
2 rows in set (0. 00 sec)

Example of CREATION AND USING PROCEDURES AND FUNCTIONS in MySQL:
Creating a procedure.
mysql> CREATE PROCEDURE sp_ in(p VARCHAR(10)) SET @x= P;
Query OK, 0 rows affected (0. 00 sec)
mysql> CALL sp_ in(_ CSE_ BTECH‘)
Query OK, 0 rows affected (0. 00 sec)
mysql> SELECT @X;
@X
CSE_ BTECH
1 row in set (0. 00 sec)

An OUT using example

mysql> CREATE PROCEDURE sp_ out(OUT p VARCHAR(10)) SET p=‘ 5th sem‘;
Query OK, o rows affected (0. 00 sec)
mysql> CALL sp_ out(@X);
Query OK, 0 rows affected(0. 00 sec)
mMysql> select @X;
@X
5th sem
1 row in set (0. 00 sec)
(or)
mysql> SET @X=‘ 5th sem‘;
Query OK, 0 rows affected (0. 00 sec)
Mysql> select @X;
@X
5th sem
1 row in set (0. 00 sec)

Creating a Function

mysql> CREATE FUNCTION hello (s CHAR(20))
RETURNS CHAR(50)
DETERMINISTIC
RETURN CONCAT(s);
Query OK, 0 rows affected (0. 00 sec)

mysql> SELECT HELLO(_ CSE‖);
HELLO(‘CST’)
CST
1 row in set (0. 00 sec)

In general, SQL statements can be embedded anywhere in a program that host language statements are allowed.

The following example shows a simple embedded SQL program that retrieves an employee name and salary from the database and prints them on a standard output device. The statements that begin with the words EXEC SQL are embedded SQL statements. The sequence of statements in this example illustrates a pattern common to most embedded SQL programs.

```
begin program
exec sql include sqlca;
exec sql begin declare section;
        name character_string(15);
        salary float;
exec sql end declare section;
exec sql whenever sqlerror stop;
exec sql connect personnel;
exec sql select ename, sal
        into :name, :salary
```

```
        from employee
        where eno = 23;
print name, salary;
exec sql disconnect;
end program
```

Each program statement is described here:
exec sql include sqlca;
They INCLUDE statement incorporates the SQL error and status handling mechanism–the SQL Communications Area (SQLCA)–into the program. The SQLCA is used by the WHENEVER statement, which appears later in the program.

exec sql begin declare section;

Next is an SQL declaration section. Host language variables must be declared to SQL prior to their use in any embedded SQL statements. Host language variables are described in detail in the next section.

exec sql whenever sqlerror stop;

The WHENEVER statement that follows uses information from the SQLCA to control program execution under error or exception conditions. In general, an error handling mechanism must precede all executable embedded SQL statements in a program. For details about error handling, see Error Handling in the chapter “Working with Transactions and Handling Errors.”

exec sql connect personnel;

Next is a series of SQL and host language statements. The first statement initiates access to the personnel database. A CONNECT statement must precede any references to a database.

```
exec sql select ename, sal
        into :name, :salary
        from employee
        where eno = 23;
```

Next is the familiar SELECT statement, containing a clause that begins with the keyword INTO. The INTO clause associates values retrieved by the SELECT statement with host language variables in the program. Following the INTO keyword are the two host language variables previously declared to SQL: name and salary.

print name, salary;
This host language statement prints the values contained in the variables.
exec sql disconnect;

The last SQL statement in the program severs the connection of the program to thedatabase.
Consider a simple C program to illustrate embedded SQL. This program below accepts student

name from the user and queries DB for his student id.

```
#include  <stdio.h>
#include <sqlca.h>
int main()
{
        EXEC SQL INCLUDE SQLCA;
        EXEC SQL BEGIN DECLARE SECTION;
                BASED ON STUDENT.STD_ID SID; // host variable to store the value
returned by query
                char *STD_NAME; // host variable to pass the value to the query
                short ind_sid;// indicator variable
        EXEC SQL END DECLARE SECTION;
        //Error handling
        EXEC WHENEVER NOT FOUND GOTO error_msg1;
        EXEC WHENEVER SQLERROR GOTO error_msg2;
        printf("Enter the Student name:");
        scanf("%s", STD_Name);
        // Executes the query
        EXEC SQL SELECT STD_ID INTO : SID INDICATOR ind_sid FROM STUDENT
WHERE STD_NAME = : STD_NAME;
        printf("STUDENT ID:%d", STD_ID); // prints the result from DB
        exit(0);
 // Error handling labels
error_msg1:
        printf("Student Id %d is not found", STD_ID);
        printf("ERROR:%ld", sqlca->sqlcode);
        printf("ERROR State:%s", sqlca->sqlstate);
        exit(0);
 error_msg2:
        printf("Error has occurred!");
        printf("ERROR:%ld", sqlca->sqlcode);
        printf("ERROR State:%s", sqlca->sqlstate);
        exit(0);
}
```

Learning Outcome

After undergoing this laboratory module, the student should be able to create database in MySQL and access it from a c program.

Exercise

1. Practice the questions in Experiment No. 1 SQL queries in the MYSQLenvironment.
2. Practice the questions in Experiment No. 2 for creation of tables, modification of table structure and inserting values into the tables in the MYSQL environment also.
3. Practice the questions in Experiment No 3 for practicing SQL set operations, joins and nested queries in the MYSQL environment also.

Case Studies

Learning Objective

To develop Database application using DBMS design steps.

The following are some sample applications

1. Accounting Package for Shops
2. Database Manager for Magazine Agency or Newspaper Agency
3. Ticket Booking for Performances
4. Preparing Greeting Cards & Birthday Cards
5. Personal Accounts - Insurance, Loans, Mortgage Payments, Etc.
6. Doctor's Diary & Billing System
7. Personal Bank Account
8. Class Marks Management
9. Hostel Accounting
10. Video Tape Library
11. History of Cricket Scores
12. Cable TV Transmission Program Manager
13. Personal Library
14. Sailors Database
15. Suppliers and Parts Database

Learning Outcome

The student should be able to design a database for the given application by applying design steps.

Sample DB Application: Railway Reservation System

To develop a Railway Reservation application using DBMS concepts.

Requirement Analysis:

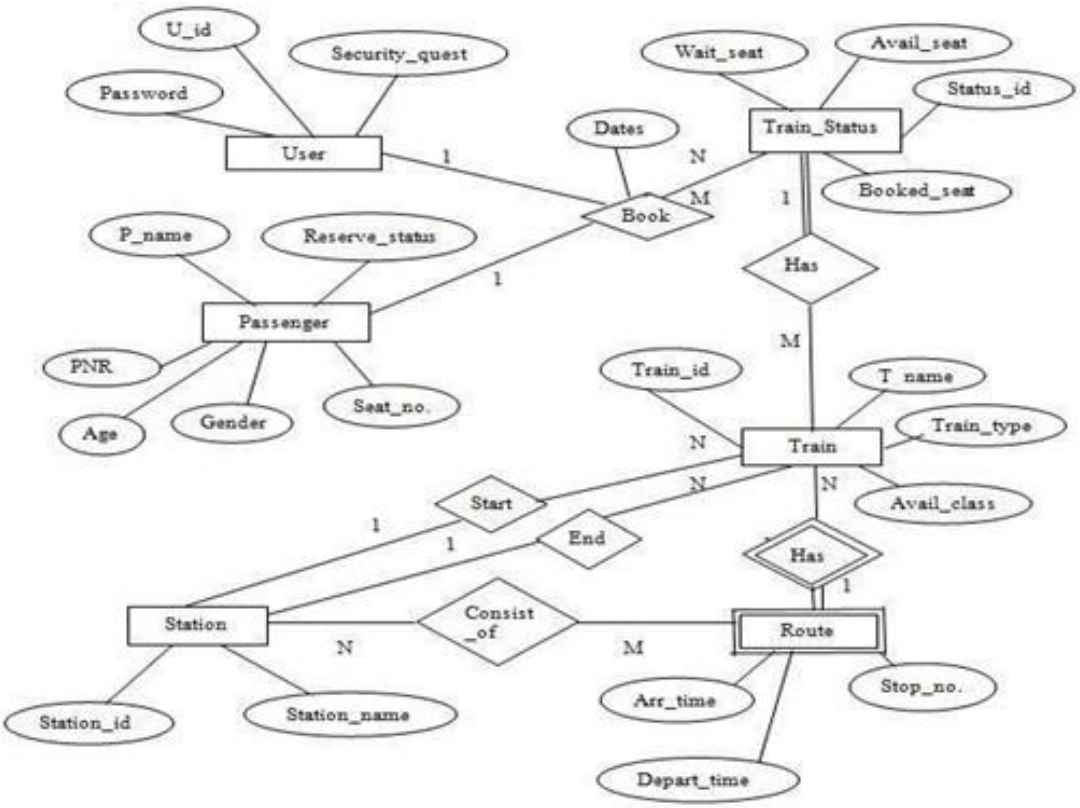
This step includes obtaining the problem statement from user requirements like

- Seat Reservation.
- Cancel Reservation
- Update train information and report generation.
- View Reservation status and train Schedule.

Conceptual Database Design:

This step includes the following tasks

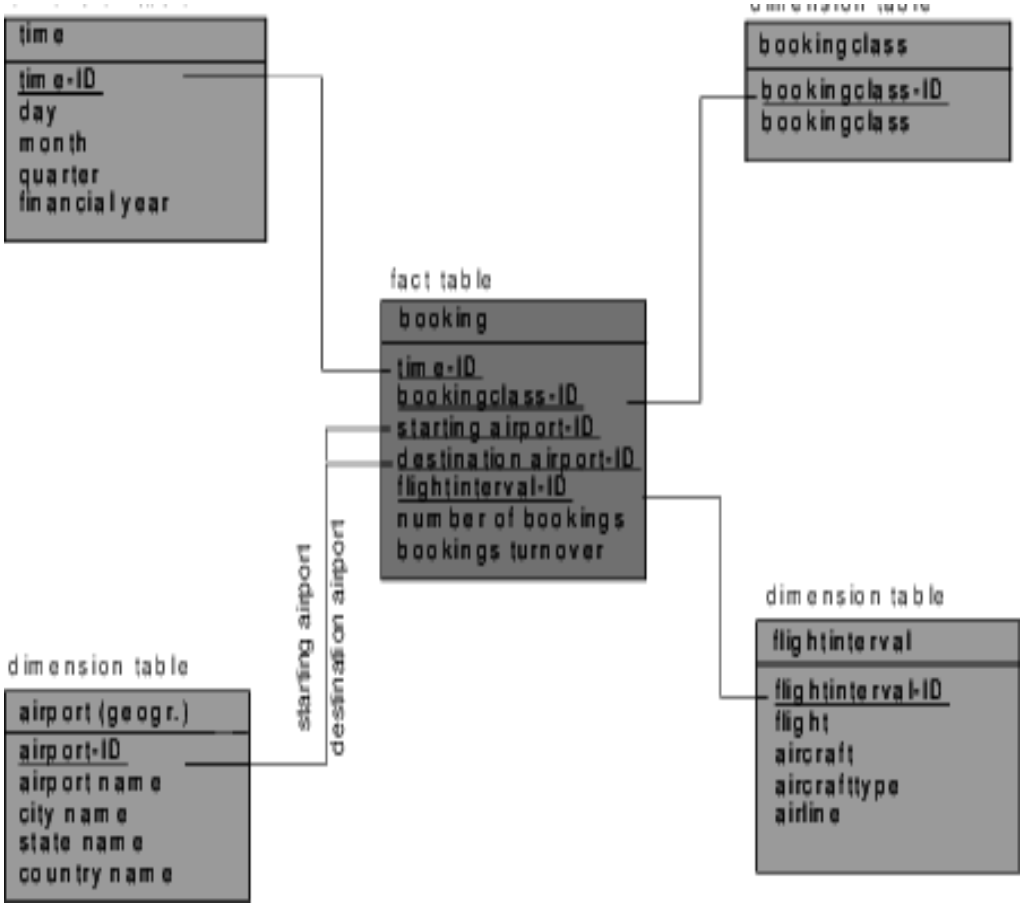
- Identifying entities by noun phrase approach
  (Train, Train Status, Book Ticket, Cancellation, Passenger)
- Identifying attributes.
  (Booking Date, Ticket_Id, Train No, Train Name, Category, Pname, Psex,Page)
- Identifying relationships
  (Booking,Status_of)
- Drawing ER Diagrams.



- Refining ER diagrams: The ER diagrams can be refined and simplified if possible.

Logical Database Design:

This step includes mapping of ER diagrams to tables.



Schema Refinement:

This step includes Normalization. By the process of Normalization, we get Normal forms like 1NF, 2NF, 3NF, 4NF and BCNF.

Relations already in 1NF
USER
STATION
TRAIN_STATUS
ROUTE
RESERVATION
ROUTE_HAS_STATION

Relations not in 1NF
PASSENGER
TRAIN

RELATIONS AFTER FIRST NORMAL FORM(1NF)

(PASSENGER) after 1NF

- Passenger_ticket

PNR	Class Type	Fare	Source_ID	Destination_ID

- Passenger

PNR	Seat number	Passenger_name	Age	Gender

(TRAIN) after 1NF

- Train_days

Train_ID	Days_Available

- Train

Train_ID	Train_Name	Train_type	Source_ID
Destination_ID	Seats_class1	Fare_class1	Seats_class2
Fare_class2	Seats_class3	Fare_class3	

Creation of Tables:

This step includes creation of tables in Oracle.

CREATE TABLE

create table **train_details**(train_name char(15) primary key, total_seats number(3), reserved_seats number(3));
create table **reservation_status**(train_name char(15) references train_details(train_name), seat_id number(3), reserved char(2) check (reserved in('y','n')), customer_name char(15));
create table **waiting_list**(slno number(3), customer_name char(15) primary key, train_name char(15) references train_details(train_name));

Insertion of Data :

This step includes insertion of proper data for the project.

```
Insert into train_details(train_name ,total_seats, reserved_seats) values('circar',45,4);
Insert into reservation_status(train_name, seat_id, reserved, customer_name) values
('circar',34,'C1','arun');
Insert into Waiting_list(slno , customer_name, train_name ) values(34,'arun','circar');
```

VIVA Questions

1. What is SQL?

   Structured Query Language (SQL) being ANSI standard language updates database and commands for accessing.

2. What is database?

   A database is a logically coherent collection of data with some inherent meaning, representing some aspect of real world and which is designed, built and populated with data for a specific purpose.

3. What is DBMS?

   It is a collection of programs that enables user to create and maintain a database. In other words it is general-purpose software that provides the users with the processes of defining, constructing and manipulating the database for various applications.

4. What is a Database system?

   The database and DBMS software together is called as Database system.

5. What are the various kinds of interactions catered by DBMS?

   - Data definition
   - Update
   - Retrieval
   - Administration

6. Advantages of DBMS?

   - Redundancy is controlled.
   - Unauthorized access is restricted.
   - Providing multiple user interfaces.
   - Enforcing integrity constraints.
   - Providing backup and recovery.

7. Who proposed the relational model?

   Edgar F. Codd proposed the relational model.

8. Segregate database technology's development.

The development of database technology is divided into:

- Structure or data model
- Navigational model
- SQL/ relational model

9. Disadvantage in File Processing System?

- Data redundancy & inconsistency.
- Difficult in accessing data.
- Data isolation.
- Data integrity.
- Concurrent access is not possible.
- Security Problems.

10. Describe the three levels of data abstraction?

There are three levels of abstraction:

- Physical level: The lowest level of abstraction describes how data are stored.
- Logical level: The next higher level of abstraction, describes what data are stored in database and what relationship among those data.
- View level:The highest level of abstraction describes only part of entire database.

11. Define the "integrity rules"

There are two Integrity rules.

- Entity Integrity: States that ¯Primary key cannot have NULL value‖
- Referential Integrity: States that ¯Foreign Key can be either a NULL value or should be Primary Key value of other relation.

12. What is Data Independence?

Data independence means that ¯the application is independent of the storage structure and access strategy of data‖. In other words, the ability to modify the schema definition in one level should not affect the schema definition in the next higher level.

Two types of Data Independence:

1. Physical Data Independence: Modification in physical level should not affect the logical level.
2. Logical Data Independence: Modification in logical level should affect the view level.

NOTE: Logical Data Independence is more difficult to achieve.

13. What is a view? How it is related to data independence?

A view may be thought of as a virtual table, that is, a table that does not really exist in its own right but is instead derived from one or more underlying base table. In other words, there is no stored file that direct represents the view instead a definition of view is stored in data dictionary. Growth and restructuring of base tables are not reflected in views. Thus, the view can insulate users from the effects of restructuring and growth in the database. Hence accounts for logical data independence.

14. What is Data Model?

A collection of conceptual tools for describing data, data relationships data semantics and constraints.

15. What is E-R model?

This data model is based on real world that consists of basic objects called entities and of relationship among these objects. Entities are described in a database by a set of attributes.

16. What is Object Oriented model?

This model is based on collection of objects. An object contains values stored in instance variables within the object. An object also contains bodies of code that operate on the object. These bodies of code are called methods. Objects that contain same types of values and the same methods are grouped together into classes.

17. What is an Entity?

It is an 'object' in the real world with an independent existence.

18. What is an Entity type?

It is a collection (set) of entities that have same attributes.

19. What is an Entity set?

It is a collection of all entities of particular entity type in the database.

20. What is an Extension of entity type?

The collections of entities of a particular entity type are grouped together into an entity set.

21. What is an attribute?

It is a particular property, which describes the entity.

22. What is a Relation Schema and a Relation?

A relation Schema denoted by R(A1, A2, …, An) is made up of the relation name R and the list of attributes Ai that it contains. A relation is defined as a set of tuples. Let r be the relation which contains set tuples (t1, t2, t3, ...,tn). Each tuple is an ordered list of n-values t=(v1,v2, ..., vn).

23. What is degree of a Relation?

It is the number of attribute of its relation schema.

24. What is Relationship?

It is an association among two or more entities.

25. What is Relationship set?

The collection (or set) of similar relationships.

26. What is Relationship type?

Relationship type defines a set of associations or a relationship set among a given set of entity types.

27. What is degree of Relationship type?

It is the number of entity type participating. The various relationships of database are:

- One-to-one: Single table having drawn relationship with another table having similar kind of columns.
- One-to-many: Two tables having primary and foreign key relation.
- Many-to-many: Junction table having many tables related to many tables.

28. What is DDL (Data Definition Language)?

A data base schema is specified by a set of definitions expressed by a special language called DDL.

29. What is SDL (Storage Definition Language)?

This language is to specify the internal schema. This language may specify the mapping between two schemas.

30. What is VDL (View Definition Language)?

It specifies user views and their mappings to the conceptual schema.

31. What is Data Storage - Definition Language?

The storage structures and access methods used by database system are specified by a set of definition in a special type of DDL called data storage- definition language.

32. What is DML (Data Manipulation Language)?

This language that enable user to access or manipulate data as organized by appropriate data model.

- Procedural DML or Low level: DML requires a user to specify what data are needed and how to get those data.
- Non-Procedural DML or High level: DML requires a user to specify what data are needed without specifying how to get those data.

33. What is DML Compiler?

It translates DML statements in a query language into low-level instruction that the query evaluation engine can understand.

34. Enlist some commands of DDL.

They are:

CREATE: used in the CREATE TABLE statement. Syntax is:

CREATE TABLE [column name] ([column definitions]) [table parameters]

ALTER: It helps in modification of an existing object of database. Its syntax is:

ALTER objecttype objectname parameters.

DROP: It destroys an existing database, index, table or view. Its syntax is:

DROP objecttype objectname.

35. Define Union All operator and Union.

- Full recordings of two tables is Union All operator.
- A distinct recording of two tables is Union.

36. Define cursor.

A database object which helps in manipulating data row by row representing a result set is called cursor.

37. Enlist the cursor types.

They are:

- Dynamic: it reflects changes while scrolling.
- Static: doesn't reflect changes while scrolling and works on recording of snapshot.
- Keyset: data modification without reflection of new data is seen.

38. Enlist the types of cursor.

They types of cursor are:

- Implicit cursor: Declared automatically as soon as the execution of SQL takes place without the awareness of the user.
- Explicit cursor: Defined by PL/ SQL which handles query in more than one row.

39. Define sub-query.

A query contained by a query is called Sub-query.

40. Why is group-clause used?

Group-clause uses aggregate values to be derived by collecting similar data.

41. Compare Non-clustered and clustered index

Both having B-tree structure, non-clustered index has data pointers enabling one table many nonclustered indexes while clustered index is distinct for every table.

42. Define Aggregate functions.

Functions which operate against a collection of values and returning single value is called aggregate functions.

43. Define Scalar functions.

Scalar function is depended on the argument given and returns sole value.

44. Define "correlated subqueries".

A 'correlated subquery' is a sort of sub query but correlated subquery is reliant on another query for a value that is returned. In case of execution, the sub query is executed first and then the correlated query.

45. What do you mean by Index hunting?

Indexes help in improving the speed as well as the query performance of database. The procedure of boosting the collection of indexes is named as Index hunting

46. Define Join and enlist its types.

Joins help in explaining the relation between different tables. They also enable you to select data with relation to data in another table.

The various types are:

- INNER JOINs: Blank rows are left in the middle while more than equal to two tables are joined.
- OUTER JOINs: Divided into Left Outer Join and Right Outer Join. Blank rows are left at the specified side by joining tables in other side.
- Other joins are CROSS JOINs, NATURAL JOINs, EQUI JOIN and NON-EQUI JOIN.

47. What restrictions can you apply when you are creating views?

Restrictions that are applied are:

- Only the current database can have views.
- You are not liable to change any computed value in any particular view.
- Integrity constants decide the functionality of INSERT and DELETE.
- Full-text index definitions cannot be applied.
- Temporary views cannot be created.
- Temporary tables cannot contain views.
- No association with DEFAULT definitions.
- Triggers such as INSTEAD OF is associated with views.

48. What is Functional Dependency?

A Functional dependency is denoted by X Y between two sets of attributes X and Y that are subsets of R specifies a constraint on the possible tuple that can form a relation state r of R. The constraint is for any two tuples t1 and t2 in r if t1[X] = t2[X] then they have t1[Y] = t2[Y]. This means the value of X component of a tuple uniquely determines the value of component Y.

49. What is normalization?

It is a process of analyzing the given relation schemas based on their Functional Dependencies (FDs) and primary key to achieve the properties

- Minimizing redundance
- Minimizing insertion, deletion and updating anomalies

.

50. Enlist the advantages of normalizing database.

Advantages of normalizing database are:

- No duplicate entries
- Saves storage space
- Boasts the query performances

51. When is a functional dependency F said to be minimal?

- Every dependency in F has a single attribute for its right hand side.
- We cannot replace any dependency X -> A in F with a dependency Y -> A where is a proper subset of X and still have a set of dependency that is equivalent to F.
- We cannot remove any dependency from F and still have set of dependency that is equivalent to F.

52. What is Multivalued dependency?

Multivalued dependency denoted by X Y specified on relation schema R, where X and Y are both subsets of R, specifies the following constraint on any relation r of R: if two tuples t1 and t2 exist in r such that t1[X] = t2[X] then t3 and t4 should also exist in r with the following properties

- t3[x] = t4[X] = t1[X] = t2[X]
- t3[Y] = t1[Y] and t4[Y] = t2[Y]
- t3[Z] = t2[Z] and t4[Z] = t1[Z]

where [Z = (R-(X U Y))]

53. What is Lossless join property?

It guarantees that the spurious tuple generation does not occur with respect to relation schemas after decomposition.

54. What is 1 NF (Normal Form)?

The domain of attribute must include only atomic (simple, indivisible) values.

55. What is Fully Functional dependency?

It is based on concept of full functional dependency. A functional dependency X Y is fully functional dependency if removal of any attribute A from X means that the dependency does not hold any more.

56. What is 2NF?

A relation schema R is in 2NF if it is in 1NF and every non-prime attribute A in R is fully functionally dependent on primary key.

57. What is 3NF?

- A relation schema R is in 3NF if it is in 2NF and for every FD X A either of the following is true X is a Super-key of R.
- A is a prime attribute of R.

In other words, if every non prime attribute is non-transitively dependent on primary key.

58. What is BCNF (Boyce-Codd Normal Form)?

A relation schema R is in BCNF if it is in 3NF and satisfies additional constraints that for every FD X A, X must be a candidate key.

59. What is 4NF?

A relation schema R is said to be in 4NF if for every Multivalued dependency X Y that holds over R, one of following is true

- X is subset or equal to (or) XY = R
- X is a Super-key

60. What is 5NF?

A Relation schema R is said to be 5NF if for every join dependency {R1, R2, ...,Rn} that holds R, one the following is true

- Ri = R for some i
- The join dependency is implied by the set of FD, over R in which the left side is key of R.