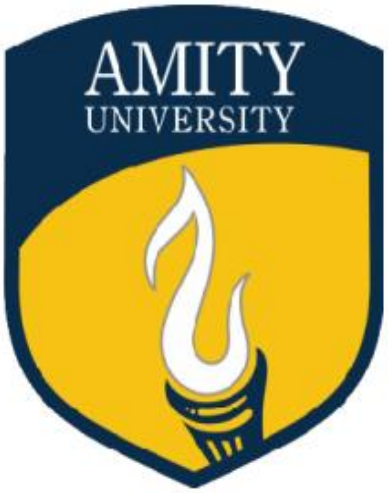
AMITY UNIVERSITY GWALIOR
MADHYA PRADESH



Practical file (2021-2025)
Database management system
(CSE-324)

Submitted by:
Priyanshi Garg
B.Tech(CSE)-D
Enroll no-A60205221261

Submitted To:
Dr. Deepak Motwani
Associate Professor
AUMP, Gwalior

INDEX

p.NO	TOPICS	PAGE NO.
1.	DDL	4-6
2.	DML	6-7
3.	DQL	7-8
4.	TCL	8
5.	DCL	8-9
6.	Constraints	9-10
7.	keys	10-11
8.	Conversion functions	11-12
9.	Oracle functions	12-18
10.	Group by - Having	18-19
11.	Order by	19
12.	Distinct	19-20
13.	Operators(all 3 types)	20-22
14.	Dual table	23
15.	View	23
16.	On delete cascade	24
17.	Subquery	24-25
18.	Correlated subquery	25-26
19.	sequence	26-27
20.	Index	27-28
21.	Where	28-29

22.	Joins	29-31
23.	Pl\sql	32 – 38
24.	B2 LAB Exercise	39 - 55
25.	SQL [SET-B]	56 - 65
26.	DBA Commands	66-68

SQL

DDL (Data Definition Language) – Commands such as CREATE , DROP , ALTER , TRUNCATE

1. CREATE: The CREATE statement is used to create a new table in a database.

Syntax

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
   ....
);
```

EXAMPLE:

```
create table btechstudent1(studentID number(10) not null, Fname varchar(10),
lname varchar(10), Address varchar(30), marks number(5), primary key(studentID));
```

Results Explain Describe Saved SQL History

```
Table created.

0.01 seconds
```

2. ALTER: This command is used to add, delete, or modify columns in an existing table.

Syntax

To add a new column in the existing table

```
ALTER TABLE table_name
ADD column_name datatype;
```

EXAMPLE:

```
alter table btechstudent1 add (subject varchar(20) unique);
```

Results Explain Describe Saved SQL History

STUDENTLD	FNAME	LNAME	ADDRESS	MARKS	SUBJECT
61	priyanshi1	g	gwl	98	-

1 rows returned in 0.00 seconds CSV Export

3. TRUNCATE: This statement deletes all the rows from the table. This is different from the DROP command, the DROP command deletes the entire table along with the table schema, however, TRUNCATE just deletes all the rows and leaves an empty table.

Syntax

```
TRUNCATE TABLE table_name;
```

EXAMPLE:

```
TRUNCATE TABLE BTECHSTUDENT1;
```

Results Explain Describe Saved SQL History

```
Table truncated.
```

0.11 seconds

After truncating the table, if you write

```
TRUNCATE TABLE BTECHSTUDENT1;
select * from btechstudent1;
```

Results Explain Describe Saved SQL

no data found

4. DROP: This command is used to drop tables.

Syntax

```
DROP TABLE table_name;
```

EXAMPLE

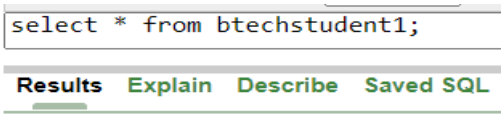


Table dropped.

0.04 seconds

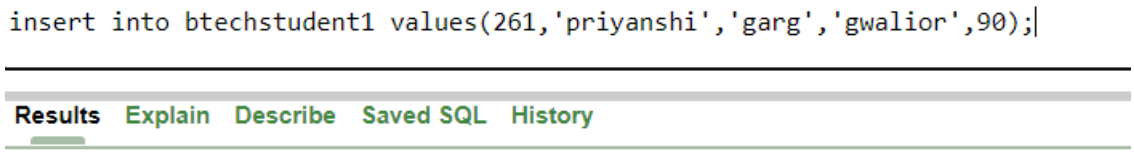
DML (Data Manipulation Language) – Commands such as INSERT, UPDATE, DELETE

5. INSERT: The INSERT INTO statement is used to insert new records in a tab

Syntax

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

EXAMPLE



1 row(s) inserted.

0.01 seconds

6. DELETE: The DELETE statement is used to delete existing records in a table.

Syntax

```
DELETE FROM table_name WHERE condition
```

EXAMPLE

```
delete from btechstudent1 where studentlD=261;
```

Results Explain Describe Saved SQL History

```
1 row(s) deleted.

0.00 seconds
```

7. UPDATE: The UPDATE statement is used to modify the existing records in a table.

Syntax

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

EXAMPLE

```
update btechstudent1 set marks=98 where studentld=61;
```

Results Explain Describe Saved SQL History

```
1 row(s) updated.
```

DQL (Data Query Language) – Commands such as SELECT

8. SELECT: The SELECT statement is used to select data from a database.

Syntax

```
SELECT expressions
FROM TABLE_NAME
WHERE conditions;
```

EXAMPLE

```
select *from btechstudent1
```

Results Explain Describe Saved SQL History

STUDENTLD	FNAME	LNAME	ADDRESS	MARKS
61	priyanshi1	g	gwl	80

1 rows returned in 0.00 seconds CSV Export

TCL (Transaction Control Language) – Commands such as COMMIT, ROLLBACK , SAVEPOINT

9. COMMIT: Commit command is used to permanently the save the record from database from Buffer to Data file.

Syntax

```
COMMIT;
```

10. ROLLBACK: Rollback command is used to undo the changes that have been made to the database temporarily. The important point to note here is that the changes saved using the `COMMIT` command cannot be undone using the `ROLLBACK` command.

Syntax

```
ROLLBACK;
```

11.SAVEPOINT: This command helps in roll backing the transactions till a certain point. For example, a transaction consists of several DML commands and we want to undo the changes till a certain point and not completely, this can be achieved by using `SAVEPOINT.`

Syntax

```
SAVEPOINT SAVEPOINT_NAME;
```

DCL (Data Control Language) – Commands such as GRANT, REVOKE

12. GRANT: It is used to give user access privileges to a database.

Example:

The following command will grant users `USER1` & `USER2`, the select, update and delete access to the table `TABLE1` .

```
GRANT SELECT, UPDATE, DELETE ON TABLE1 TO USER1, USER2;
```

13. REVOKE: It revokes the given access to the user.

Example:

Let's revoke the access from `USER1` & `USER2` given above using `GRANT` command.

```
REVOKE SELECT, UPDATE, DELETE ON TABLE1 FROM USER1, USER2;
```

14. CONSTRAINTS

Constraints enforce limits to the data or type of data that can be inserted/updated/deleted from a table. The whole purpose of constraints is to maintain the **data integrity** during an update/delete/insert into a table.

CONSTRAINTS TYPES:

- NOT NULL
- UNIQUE
- DEFAULT
- CHECK
- Key Constraints – PRIMARY KEY, FOREIGN KEY

Syntax

Create table table_name(

Column1 datatype constraint,

Column1 datatype constraint,

…….);

Example:

```
create table p_mahesh_6( id number(30) PRIMARY KEY, name varchar(20) NOT NULL UNIQUE,
AGE1  NUMBER (20) NOT NULL CHECK (AGE1>=29), AGE2 NUMBER (10) DEFAULT NULL )
desc p_mahesh_6
```

Results Explain **Describe** Saved SQL History

Object Type **TABLE** Object **P_MAHESH_6**

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
P_MAHESH_6	ID	Number	-	30	0	1	-	-	-
	NAME	Varchar2	20	-	-	-	-	-	-
	AGE1	Number	-	20	0	-	-	-	-
	AGE2	Number	-	10	0	-	✓	NULL	-

1 - 4

** Note: foreign key should be applied on other table taking reference of 1st table. See point no.15 for an example of a foreign key.

15. KEYS

Key plays an important role in a relational database; it is used for identifying unique rows from tables. It also establishes the relationships among tables.

- PRIMARY KEY (PK): A primary is a column or set of columns in a table that uniquely identifies tuples (rows) in that table.

Syntax

```
CREATE TABLE table_name
(
  column_name1 datatype [ NULL | NOT NULL ],
  column_name2 datatype [ NULL | NOT NULL ],
  ...

  CONSTRAINT constraint_name PRIMARY KEY (column_nameX, column_nameY..)
);
```

Example:

```
CREATE TABLE STUDENTS
( stu_id int NOT NULL
  first_name VARCHAR(30) NOT NULL,
  last_name VARCHAR(25) NOT NULL,
  dob DATE,
  CONSTRAINT student_pk PRIMARY KEY (stu_id)
);
```

- SUPER KEY (SK): A super key is a set of one of more columns (attributes) to uniquely identify rows in a table.

- CANDIDATE KEY (CK): A super key with no redundant attribute is known as the candidate key.

- COMPOSITE KEY (CK): A key that consists of more than one attribute to uniquely identify rows (also known as records & tuples) in a table is called a composite key.

- ALTERNATE KEY (AK): Out of all candidate keys, only one gets selected as the primary key, the remaining keys are known as alternate or secondary keys.

- FOREIGN KEY (AK): Foreign keys are the columns of a table that points to the primary key of another table. They act as a cross-reference between tables.

Example:

Create Table emp162_C2 (empno number(5) primary key, Ename varchar2(20) Not NULL, sal number(9,2), deptno references Dept162_C2);

16. CONVERSION FUNCTIONS

The CONVERT() function **converts a value (of any type) into a specified datatype**.

- TO_NUMBER: To convert string into numeric format.

Example:

SQL > Select '12' + '12' from dual;

Ans. 24

- TO_CHAR: To convert numeric into char and used to set the date format.

Example:

SQL > Select to_char(sal, '9,999.00') from emp;

- TO_DATE: Convert a character string to a date format using the **TO_DATE** function

Example:

```
SELECT last_name, hire_date
FROM employees
WHERE hire_date = TO_DATE('May 24, 1999', 'fxMonth DD, YYYY');
```

OUTPUT:

LASTNAME	HIREDATE
Kumar	24-MAY-99

17. ORACLE FUNCTIONS

Oracle Functions are categorized into two parts. **Aggregate function** and group function set groups of value and **Scalar function**. [Individuals work on a single row] or each row.

AGGREGATEFUNCTION

These functions are used to do operations from the values of the column and a single value is returned.

- **AVG()**: It returns the average value after calculating from values in a numeric column.

Syntax:

```
SELECT AVG(column_name) FROM table_name;
```

Example:

- Computing average marks of students.

```
SELECT AVG(MARKS) AS AvgMarks FROM Students;
```

Output:

AvgMarks
80

- **COUNT():** It is used to count the number of rows returned in a SELECT statement. It can't be used in MS ACCESS.

Syntax:

```
SELECT COUNT(column_name) FROM table_name;
```

Example:

- Computing number of students with unique/distinct age.

```
SELECT COUNT(DISTINCT AGE) AS NumStudents FROM Students;
```

Output:

NumStudents
4

- **FIRST():** The FIRST() function returns the first value of the selected column.

Syntax:

```
SELECT FIRST(column_name) FROM table_name;
```

Example:

- Fetching marks of first student from the Students table.

```
SELECT FIRST(MARKS) AS MarksFirst FROM Students;
```

Output:

MarksFirst
90

- **LAST():** The LAST() function returns the last value of the selected column. It can be used only in MS ACCESS.

Syntax:

```
SELECT LAST(column_name) FROM table_name;
```

Example:

- Fetching marks of last student from the Students table.

```
SELECT LAST(MARKS) AS MarksLast FROM Students;
```

Output:

MarksLast
85

- **MIN():** The MIN() function returns the minimum value of the selected column.

Syntax:

```
SELECT MIN(column_name) FROM table_name;
```

Example:

- Fetching minimum age among students from the Students table.

```
SELECT MIN(AGE) AS MinAge FROM Students;
```

Output:

MinAge
18

- **SUM():** The SUM() function returns the sum of all the values of the selected column.

Syntax:

```
SELECT SUM(column_name) FROM table_name;
```

Example:

- Fetching summation of total marks among students from the Students table.

```
SELECT SUM(MARKS) AS TotalMarks FROM Students;
```

Output:

TotalMarks
400

- **MAX():** The MAX() function returns the maximum value of the selected column.

Syntax:

```
SELECT MAX(column_name) FROM table_name;
```

Example:

- Fetching maximum marks among students from the Students table.

```
SELECT MAX(MARKS) AS MaxMarks FROM Students;
```

Output:

MaxMarks
95

SCALAR FUNCTION

These functions are based on user input, these too return a single value.

- **UCASE()**: It converts the value of a field to uppercase.

Syntax:

```
SELECT UCASE(column_name) FROM table_name;
```

Example:

- Converting names of students from the table Students to uppercase.

```
SELECT UCASE(NAME) FROM Students;
```

Output:

NAME
HARSH
SURESH

- **LCASE()**: It converts the value of a field to lowercase.

Syntax:

```
SELECT LCASE(column_name) FROM table_name;
```

Example:

- Converting names of students from the table Students to lowercase.

```
SELECT LCASE(NAME) FROM Students;
```

Output:

NAME
harsh
suresh

- **MID():** The MID() function extracts texts from the text field.

Syntax:

```
SELECT MID(column_name,start,length) AS some_name FROM table_name;
```

**specifying length is optional here, and start signifies start position (starting from 1)

Example:

- Fetching first four characters of names of students from the Students table.

```
SELECT MID(NAME,1,4) FROM Students;
```

Output:

NAME
HARS
SURE

- **LEN():** The LEN() function returns the length of the value in a text field.

Syntax:

```
SELECT LENGTH(column_name) FROM table_name;
```

Example:

- Fetching the length of names of students from the Students table.

```
SELECT LENGTH(NAME) FROM Students;
```

Output:

NAME
5
6

- **NOW():** The NOW() function returns the current system date and time.

Syntax:

```
SELECT NOW() FROM table_name;
```

Example:

- Fetching current system time.

```
SELECT NAME, NOW() AS DateTime FROM Students;
```

Output:

NAME	DateTime
HARSH	1/13/2017 1:30:11 PM
SURESH	1/13/2017 1:30:11 PM

- **FORMAT():** The FORMAT() function is used to format how a field is to be displayed.

Syntax:

```
SELECT FORMAT(column_name,format) FROM table_name;
```

Example:

- Formatting current date as 'YYYY-MM-DD'.

```
SELECT NAME, FORMAT(Now(),'YYYY-MM-DD') AS Date FROM Students;
```

Output:

NAME	Date
HARSH	2017-01-13
SURESH	2017-01-13

- **ROUND():** The ROUND() function is used to round a numeric field to the number of decimals specified.

  **NOTE: Many database systems have adopted the IEEE 754 standard for arithmetic operations, which says that when any numeric .5 is rounded it results to the nearest even integer i.e, 5.5 and 6.5 both gets rounded off to 6.

Syntax:

```
SELECT ROUND(column_name,decimals) FROM table_name;
```

```
**decimals- number of decimals to be fetched.
```

Example:

- Fetching maximum marks among students from the Students table.

```
SELECT ROUND(MARKS,0) FROM table_name;
```

Output:

MARKS
90
50
80

18. GROUP BY – HAVING

GROUP BY: The GROUP BY statement groups rows that have the same values into summary rows

Syntax

SELECT column_name(s)
FROM table name
WHERE condition
GROUP BY column_name(s);

Example:

IF WE WRITE A QUERY FOR THIS TABLE:

ROLL_NO	NAME	D_O_B	COURSE_ID	AGE
10	Aditya	2004	1	18
12	Amit	2001	2	21
14	Abhay	2003	3	19
16	Ankit	2002	4	20
18	Santosh	2004	5	18

select count(roll_no) from student group by AGE;

COUNT(ROLL_NO)
1
1
2
1

HAVING: The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions.

Syntax

SELECT column_name(s)
FROM table name

GROUP BY column_ name(s)

HAVING condition;

Example:

IF WE WRITE A QUERY FOR THIS TABLE:

ROLL_NO	NAME	D_O_B	COURSE_ID	AGE
10	Aditya	2004	1	18
12	Amit	2001	2	21
14	Abhay	2003	3	19
16	Ankit	2002	4	20
18	Santosh	2004	5	18

select name from student group by name having max(age)>19;

NAME
Ankit
Amit

19. ORDER BY

The ORDER BY keyword is used to sort the result set in ascending or descending order.

Syntax

SELECT columni, column2,. . .

FROM table name

ORDER BY columni, column2 ,…….. ASC I DESC;

Example:

IF WE WRITE A QUERY FOR THIS TABLE:

ROLL NO	NAME	DOB	COURSE ID	AGE
10	Priyanshi	2003	1	19
12	purvi	2005	2	27
14	princy	2008	3	14
16	goli	2022	4	01
18	prinsu	2006	5	16

select name,age from student order by AGE ASC;

NAME	AGE
Goli	01
princy	14
prinsu	16
priyanshi	19
Purvi	27

20. DISTINCT

The DISTINCT statement is used to return only distinct (different) values.

Syntax

SELECT DISTINCT columni, column2, . . .FROM table name;

Example:

IF WE WRITE A QUERY FOR THIS TABLE:

ROLL NO	NAME	DOB	COURSE ID	AGE
10	Priyanshi	2003	1	18
12	purvi	2005	2	21
14	princy	2008	3	20
16	goli	2022	4	18
18	prinsu	2006	5	19

select DISTINCT age from student;

AGE
21
20
18
19

21. OPERATORS

Arithmetic Operators

OPERATOR	DESCRIPTION
+	Addition
–	Subtraction
*	Multiplication
/	Division
%	Modulus

Examples:

```
This can be simply used to add two numbers:
SELECT 50 + 30;
Result: 80

subtraction operator can be used:
SELECT 30 - 20 -10;
Result: 0

Multiplication operator:
SELECT 10* 10 * 10;
Result: 1000

Division Operator: returns the quotient
```

```
SELECT 20 / 5;
Result: 4

Modulo operator: returns the remainder
SELECT 21 % 5;
Result: 1
```

Comparison Operators

OPERATOR	DESCRIPTION
=	Equal to operator
>	Greater than operator
<	Less than operator
>=	Greater than or equal to operator
<=	Less than or equal to operator
<>	Not equal to operator

Examples:

```
This will fetch all the records of employees from EMPLOYEE table where
employee age is equal to 18:
SELECT * FROM EMPLOYEE WHERE age = 18;

This will fetch the records of those employees from EMPLOYEE table who have
salary greater than 10000:
SELECT * FROM EMPLOYEE WHERE salary > 10000;

This will fetch the records of those employees from EMPLOYEE table who have
salary less than 30000:
SELECT * FROM EMPLOYEE WHERE salary < 30000;

This will fetch the records of employee who have salary equal to or greater
than 15000:
SELECT * FROM EMPLOYEE WHERE salary >= 15000;

This will fetch the records of employee who have salary equal to or greater
than 25000:
SELECT * FROM EMPLOYEE WHERE salary <= 25000;

This will fetch the records from EMPLOYEE table where the employee age is not
equal to 18, which means it will get all the records except the records where
employee age is 18:
SELECT * FROM EMPLOYEE WHERE age <> 18;
```

Logical Operators

ALL	Returns true if all the subquery values fulfil the condition. This operator is generally use along with a subquery. Values generated by the subquery is used by the main query in where clause. Example will make it clear, refer example below.
AND	Returns if all the conditions that are separated by AND operator are true.
ANY	Returns true if all the subquery values fulfil the condition.
EXISTS	Returns true if the subquery returns one or more records.
BETWEEN	Used in SQL query where we need to check the condition for values between a certain range, these ranges can be defined in where condition using BETWEEN operator.
IN	Compares the value to a specified list of values.
LIKE	It compares the value to similar values using wildcard operator.
OR	OR operator is used to combine multiple conditions in SQL query.
NOT	Used along with other logical operators and reverses their meaning for example, NOT IN will compares the values except the specified list of values.
SOME	Returns true if any of the subquery value fulfil the condition.

Examples:

```
The subquery will fetch the student ids for all the students from SCHOOL table where age is greater than 18, these ids will be used by main query that will fetch the name of these students from STUDENT table:
SELECT name FROM STUDENT
WHERE studentID = ALL (
SELECT studentID FROM SCHOOL WHERE age > 18
);

This will fetch the records of students where city is 'Agra' and country is 'INDIA'. This will only fetch those records where both of these conditions are true, for example it will not fetch the record where country is 'INDIA' but city is other than 'Agra':
SELECT * FROM STUDENT
WHERE City = "Agra" AND Country = "INDIA";

This will fetch all the records where student age lies between 18 and 22:
SELECT * FROM STUDENT
WHERE age BETWEEN 18 AND 22;

The following query demonstrate the use of NOT and LIKE operator. It will fetch the names of the students from STUDENT table where name does not start with letter 'a' or 'A':
SELECT name FROM STUDENT
WHERE name NOT LIKE 'a%';
```

22. DUAL TABLE

It is a table that is automatically created by Oracle Database along with the data dictionary. Also used to get the operating systems functions (like date, time, arithmetic expression., etc.)

Examples:

(a)select * from dual;

DUMMY
X

It has one column, DUMMY, defined to be VARCHAR2(1), and contains one row with a value X. **(automatically created by Oracle Database)**

(b) select sysdate from dual;

SYSDATE
18-OCT-22

23. VIEW

Views in SQL are a kind of virtual tables. A view also has rows and columns as they are in a real table in the database. We can create a view by selecting fields from one or more tables present in the database. A View can either have all the rows of a table or specific rows based on certain conditions

Syntax

CREATE OR REPLACE VIEW view name AS
SELECT columni, column2, . . .FROM
table name
WHERE condition;

Examples:

IF WE CREATE A VIEW FOR THIS TABLE:

ROLL NO	NAME	DOB	COURSE ID	AGE
10	Priyanshi	2003	1	19
12	purvi	2005	2	27
14	princy	2008	3	14
16	goli	2022	4	01
18	prinsu	2006	5	16

CREATE OR REPLACE VIEW STUDENT VU AS SELECT ROLL NO, NAME FROM STUDENT;

View created

24. ONDELETECASCADE

ON DELETE CASCADE constraint is used in MySQL to delete the rows from the child table automatically when the rows from the parent table are deleted.

Examples:

STEP 1: Create PARENT table:

```
CREATE TABLE PARENT(
PNO NUMBER(2) CONSTRAINT PK_PRT PRIMARY KEY, NAME
VARCHAR2(14));
```

STEP 2: Create CHILD table:

```
CREATE TABLE CHILD(
CNO NUMBER(4) CONSTRAINT PK_CLD PRIMARY KEY, "NO NUMBER(2) CONSTRAINT FK_PNO
REFERENCES PARENT ON DELETE CASCADE);
```

After inserting some values in both the table, we have

PNO	NAME
1	PRT_1
2	PRT_2
3	PRT_3

(PARENT)

CNO	PNO
10	1
20	2
30	3

(CHILD)

If we delete a row from the PARENT table, then it must affect the CHILD table also.

```
DELETE FROM PARENT WHERE PNO=3;
```

PNO	NAME
1	PRT_1
2	PRT_2

CNO	PNO
10	1
20	2

Hence, from both the table 1 row is deleted where PNO=3.

25. SUBQUERY

In SQL a Subquery can be simply defined as a query within another query. In other words, we can say that a Subquery is a query that is embedded in the WHERE clause of another SQL query.

Syntax:

SELECT column_name(s)
FROM table name
WHERE column_ name operator
(SELECT column_name
FROM table name WHERE condition);

Examples:

IF WE WRITE A QUERY FOR THIS TABLE:

ROLL NO	NAME	DOB	COURSE ID	AGE
10	Priyanshi	2003	1	19
12	purvi	2005	2	27
14	princy	2008	3	14
16	goli	2022	4	01
18	prinsu	2006	5	16

select name , age from student where age > (select age from student where name='princy');

NAME	AGE
purvi	27
prinsu	16
priyanshi	19

26. CORRELATEDSUBQUERY

Correlated subqueries are used for row-by-row processing. Each subquery is executed once for every row of the outer query. A correlated subquery is evaluated once for each row processed by the parent statement. The parent statement can be a SELECT, UPDATE, or DELETE statement.

Syntax:

SELECT column_name(s)
FROM table 1 outer
WHERE column_ name operator
(SELECT column_name
FROM table 2
WHERE exprl=outer. expr2);

Examples:

IF WE WRITE A QUERY FOR THIS 2 TABLE:

ROLL NO	NAME	ADDRESS	AGE
1	Priyanshi	GWL	19
2	purvi	DELHI	27
3	princy	USA	14
4	goli	NODIA	01
5	prinsu	MUMBAI	16

(TABLE 1)

COURSE	ROLL_NO
javascript	3
Sql	4
C++	5
HTML	9
Sql	11

(TABLE2)

SELECT NAME , AGE FROM TABLE1 OUTER WHERE ROLL NO=(SELECT ROLL_NO FROM TABLE2 WHERE ROLL_NO=OUTER.ROLL_NO)

NAME	AGE
Princy	14
Goli	01
prinsu	16

27.SEQUENCE

Sequence is a set of integers 1, 2, 3,.. that are generated and supported by some database systems to produce unique values on demand. A sequence is a user-defined schema bound object that generates a sequence of numeric values.

Syntax:

```
CREATE SEQUENCE sequence_name
START WITH initial value INCREMENT BY
increment value MINVALUE minimum value
MAXVALUE maximum value
CYCLE | NOCYCLE;
```

Examples:

Step 1:

```
CREATE SEQUENCE seq1
start with 1
increment by 1
minvalue 1
maxvalue 100
cycle;
```

Step 2:

```
CREATE TABLE
SEQUENCE
( ID number(10),
NAME VARCHAR2(20)
);
```

Step 3:

```
INSERT into SEQUENCE VALUES(seq1.nextval,'priyanshi');
INSERT into SEQUENCE VALUES(seq1.nextval,'purvi');
INSERT into SEQUENCE VALUES(seq1.nextval,'princy');
```

Output:-

ID	NAME
1	Priyanshi
2	Purvi
3	Princy

28.INDEX

The CREATE INDEX statement is used to create indexes in tables. Indexes are used to retrieve data from the database more quickly.

Syntax

- Creates an index on a table. Duplicate values are allowed:

```
CREATE INDEX index name
ON table_name (columnl, column2, ...);
```

Examples:

Step 1:

```
CREATE TABLE Colleges (
 college_id INT PRIMARY KEY,
 college_code VARCHAR2(20) NOT NULL,
 college_name VARCHAR2(50)
);
```

Step 2:

```
insert into colleges values(1,'a2','amity')
insert into colleges values(2,'a2','mits')
insert into colleges values(3,'a3','itm')
```

Step 3:

```
CREATE INDEX college_index
ON Colleges(college_code);
```

```
Index created.
```

The users cannot see the indexes, they are just used to speed up searches/queries.

Note: Updating a table with indexes takes more time than updating a table without (because the indexes also need an update). So, only create indexes on columns that will be frequently searched against.

29. WHERE

The `WHERE` clause is used to filter records. It is used to extract only those records that fulfill a specified condition.

Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Note: The `WHERE` clause is not only used in `SELECT` statements, it is also used in `UPDATE`, `DELETE`, etc.!

Examples:

- The following SQL statement selects all the customers from the country "Mexico", in the "Customers" table:

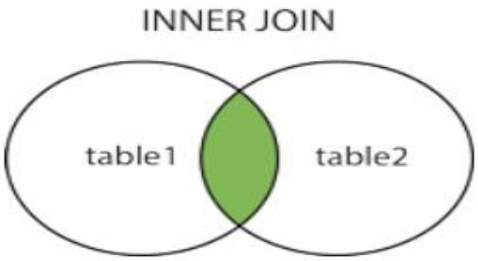
```
SELECT * FROM Customers
WHERE Country='Mexico';
```

30. JOINS

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

Here are the different types of JOINs in SQL:

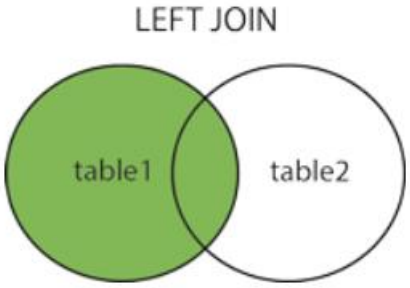
- (INNER) JOIN: Returns records that have matching values in both tables



Syntax

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

- LEFT (OUTER) JOIN: Returns all records from the left table, and the matched records from the right table

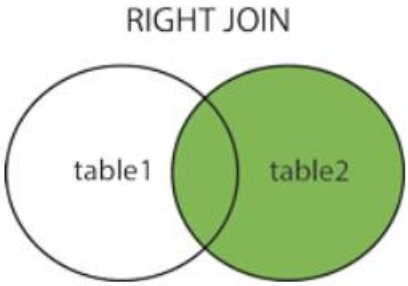


Note: In some databases LEFT JOIN is called LEFT OUTER JOIN.

Syntax

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

- RIGHT (OUTER) JOIN: Returns all records from the right table, and the matched records from the left table

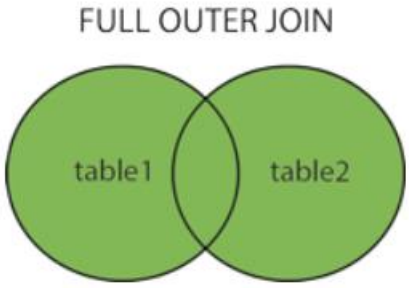


Note: In some databases RIGHT JOIN is called RIGHT OUTER JOIN.

Syntax

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

- FULL (OUTER) JOIN: Returns all records when there is a match in either left or right table



Note: `FULL OUTER JOIN` can potentially return very large result-sets!

Syntax

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```

- `(SELF) JOIN:` A self join is a regular join, but the table is joined with itself.

Syntax

```
SELECT column_name(s)
FROM table1 T1, table1 T2
WHERE condition;
```

PL\SQL :

1. WAP in PL/SQL for addition of 1 to 100 numbers.

```
declare
i number(10);
n number(10);
begin
n:=0;
i:=1;
while i<100
loop
n:=n+i;
i:=i+1;
end loop;
dbms_output.put_line('Sum is:'||n);
end;
```

Results Explain Describe Saved SQL History

```
Sum is:4950

Statement processed.

0.03 seconds
```

2. WAP in PL/SQL to inverse a number, e.g. 5467 must be inverted to 7645.

```
DECLARE
num number;
reverse_num number:=0;

begin
num:=6461;
while num>0
loop
reverse_num:=(reverse_num*10) + mod(num,10);
num:=trunc(num/10);
end loop;
dbms_output.put_line(' Reversed number is : '|| reverse_num);
end;
```

Results Explain Describe Saved SQL History

```
 Reversed number is : 1646

Statement processed.


0.00 seconds
```

3. WAP in PL/SQL for changing th eprice of product ‘p00001’ to 7000 if it’s price is less then 7000 and update this transation by updating the table.

```
Declare
 name varchar2(30);
 max_sal number(5):=7000;
 sal_updt number(20);
 BEGIN
 name:=:name;
select salary into sal_updt from agency where salesman=name;
if (sal_updt < 7000) then
```

```
 update agency
 set salary=salary+3000
where
salesman = name;
sal_updt:=sal_updt+3000;
dbms_output.put_line('Rs 3000 is added
                          and current salary is '|| sal_updt);
ELSE
dbms_output.put_line('Current salary is '||sal_updt);
END IF;
 END;
```

OUTPUT:

```
Current salary is 15000

Statement processed.
```

SALESMAN	LOC	SALARY	AGE
ROHAN	AGRA	8000	20
AMIR	GWL	7000	50
ANIKET	SAGAR	9000	40
MOHIT	MUMBAI	10000	30
TARUN	DELHI	15000	38
Tapashi	Dubai	1500	61

4. Create a view which shows the detail of salesman with his salary. (Salesmanname, salary)

```
create table grocery (salesman varchar (30), loc varchar (20), salary number, age number)
desc grocery
insert into grocery values ('Ajay', 'gwl', 10000, 20)
insert into grocery  values ('Priyanshi', 'delhi', 9999, 50)
insert into grocery  values ('Vaishu', 'Suraj', 3004, 40)
insert into grocery  values ('shivu', 'porsa', 100000, 30)
insert into grocery  values ('Aditya', 'kolkata', 15000, 38)
insert into grocery  values ('Kirti', 'Dubai', 2800, 61)
select * from grocery
create view v1 as select salesman, salary from grocery where salary <10000
select * from v1
```

Results Explain Describe Saved SQL History

SALESMAN	SALARY
Priyanshi	9999
Vaishu	3004
Kirti	2800

3 rows returned in 0.01 seconds CSV Export

5. Create a trigger on the sailors table after updating a row into the table.

```
create or replace Trigger tname after Update on emp
Declare
Begin
if updating then
Dbms_output.put_line('updation is Successfully Completed');
else
Dbms_output.put_line('updation is not Successfully    Completed');
End if;
End;
```

6. Write the pl/SQL program to find the division of two numbers and store it in a table

```
declare
    a number(5):= 8;
    b number(5):= 2;
    div number(10);
begin
     div:= a/b;

     dbms_output.put_line('Division value is '||div);
end;
```

Results Explain Describe Saved SQL History

```
Division value is 4

Statement processed.
```

7. WAP in pl/SQL to implement whether the given number is palindrome or not

```
DECLARE
   n number;
   m number;
   temp number:=0;
   rem number;
BEGIN
   n :=12321;
   m :=n;
   while n>0
   loop
      rem := mod(n,10) ;
      temp := (temp*10) + rem ;
      n := trunc(n/10);
   end loop;
   if m = temp
   then
      dbms_output.put_line('Palindrome');
   else
      dbms_output.put_line('Not Palindrome');
   end if;
END;
```

Results Explain Describe Saved SQL History

```
Palindrome

Statement processed.

0.01 seconds
```

8. WAP in P L/SQL to write a Cursor to display the list of employees and total salary departments.

```
declare
cursor c1 is select * from emp order by sal desc;
r c1%rowtype;
begin
open c1;
loop
fetch c1 into r;
exit when c1%notfound;
dbms_output.put_line('NAME - '||r.ename||'SALARY - '||r.sal ||'
department - '||r.deptno);
end loop;
close c1;
end;
```

```
NAME - KINGSALARY - 5000 department - 10
NAME - FORDSALARY - 3000 department - 20
NAME - SCOTTSALARY - 3000 department - 20
NAME - JONESSALARY - 2975 department - 20
NAME - BLAKESALARY - 2850 department - 30
NAME - CLARKSALARY - 2450 department - 10
NAME - ALLENSALARY - 1600 department - 30
NAME - TURNERSALARY - 1500 department - 30
NAME - MILLERSALARY - 1300 department - 10
NAME - WARDSALARY - 1250 department - 30
NAME - MARTINSALARY - 1250 department - 30
NAME - ADAMSSALARY - 1100 department - 20
NAME - JAMESSALARY - 950 department - 30
NAME - SMITHSALARY - 800 department - 20
```

9. WAP in PL/SQL to write a Cursor to display the list of employees who are working as Managers or Analyststs.

```
declare
cursor emp_cur IS select ename from emp where job='MANAGER'or
job='ANALYST';
my_cur emp_cur%rowtype;
begin
open emp_cur;
loop
FETCH emp_cur INTO my_cur;
   EXIT WHEN emp_cur%NOTFOUND;
dbms_output.put_line('employees name: '||my_cur.ename);
end loop;
close emp_cur;
end;
```

B2 LAB Exercise

Q1. Perform Constraints on field of tables (UNIQUE, NOT NULL, IN, DEFAULT)

1.NOT NULL-

The columns have a default value of NULL. In SQL, a NOT NULL constraint is used to prevent the insertion of NULL values into the specified column by treating them as invalid input. Given that the column will always have data, you should provide it a valid SQL NOT NULL value in the INSERT or UPDATE commands.

SQL QUERY-

```
Create Table Dept162_C2
(deptno number(5) primary key,
dname varchar2(20) Not NULL)

insert into Dept162_C2 values(3,'cse')
insert into Dept162_C2 values(2,' ')
select*from Dept162_C2
```

Results Explain Describe Saved SQL History

DEPTNO	DNAME
3	cse
2	

2 rows returned in 0.01 seconds CSV Export

2.UNIQUE-

IN SQL, the unique constraint is used to prevent duplicate values from being placed into a particular column or group of columns that are not part of the primary key but are instead participating in the unique constraint.

SQL QUERY-

```
CREATE TABLE DEPT_C2 (DEPTNO NUMBER(5) PRIMARY KEY, DNAME VARCHAR2(20)
UNIQUE);
INSERT INTO DEPT_C2 VALUES(11,'ECE')
INSERT INTO DEPT_C2 VALUES(12,'MAT')
SELECT*FROM DEPT_C2
```

Results Explain Describe Saved SQL History

DEPTNO	DNAME
11	ECE
12	MAT

2 rows returned in 0.00 seconds CSV Export

3.IN-

You can specify several values in a WHERE clause when using the IN c ommand.

SQL QUERY-

```
SELECT * FROM EMP WHERE DEPTNO IN(SELECT DEPTNO FROM DEPT W
HERE LOC = 'NEW YORK');
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7782	CLARK	MANAGER	7839	09-JUN-81	2450	-	10
7839	KING	PRESIDENT	-	17-NOV-81	5000	-	10
7934	MILLER	CLERK	7782	23-JAN-82	1300	-	10

4.DEFAULT-

If no value is supplied for that column in the INSERT statement, a DEFAU LT constraint is utilised to provide a default column value for the added r ows.

SQL QUERY-

```
create table empd_c21 (empno number(8),ename varchar(20) , sal number(9,2) default 2000);
insert into empd_c21 values (7, 'prijay' ,default)
select * from empd_c21
```

Results Explain Describe Saved SQL History

EMPNO	ENAME	SAL
7	prijay	2000
7	prijay	2000

2 rows returned in 0.01 seconds CSV Export

Q2. Check constraint

In order to restrict the range of values that may be put into a column or set of columns using a preset condition, a check constraint is established on the column or group of columns in question. When a value is inserted or updated, the check constraint is activated to assess it. If the value fulfils the requirement, it is added to the table; otherwise, the insert operation is rejected. Multiple CHECK restrictions may be specified for a single column.

SQL QUERY-

```
CREATE TABLE EMPD_C21 (EMPNO NUMBER(5), ENAME VARCHAR2(20), SAL NUMBER (9));

ALTER TABLE EMPD_C21 ADD CHECK(SAL>0);

INSERT INTO EMPD_C21 VALUES(1, 'LUCIFER', 0);
```

```
ORA-02290: check constraint (SYSTEM.SYS_C004192) violated
```

Q3. How to use on delete cascade at the time of creating table?

ON DELETE CASCADE clause is used to automatically remove the matching records from the child table when we delete the rows from the parent table. It is a kind of referential action related to the foreign key.

Suppose we have created two tables with a FOREIGN KEY in a foreign key relationship, making both tables a parent and child. Next, we define an ON DELETE CASCADE clause for one FOREIGN KEY that must be set for the other to succeed in the cascading operations. If the ON DELETE CASCADE is defined for one FOREIGN KEY clause only, then cascading operations will throw an error.

SQL QUERY-

```
CREATE TABLE EMP0078_C2 (EMPNO NUMBER(5) PRIMARY KEY, ENAME
 VARCHAR2(20), DEPNO REFERENCES DEPT0078_C2 ON DELETE CASCAD
E);

CREATE TABLE DEPT0078_C2 (DEPTNO NUMBER(10) PRIMARY KEY, LOC
VARCHAR2(20));

DROP TABLE DEPT0078_C2;
```

```
ORA-02449: unique/primary keys in table referenced by foreign keys
```

```
DROP TABLE DEPT0078_C2 CASCADE CONSTRAINTS;
```

```
Table dropped.
```

Q4. Oracle functions;(nvl() , count(),avg(), sum())

1.COUNT-

The number of objects in a group is returned by the aggregate Oracle COUNT () method.

The COUNT () function accepts a clause which can be either ALL, DISTINCT, or *:

SQL QUERY-

```
SELECT COUNT(ENAME), COUNT(EMPNO) FROM EMP
```

COUNT(ENAME)	COUNT(EMPNO)
15	15

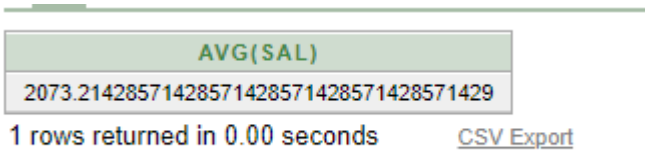
1 rows returned in 0.10 seconds CSV Export

2.AVG ()-

The average is returned by the Oracle AVG () method after accepting a lis t of values. Either a DISTINCT or ALL clause can be sent to the AVG () met hod. While the ALL clause forces the function to take into account all du plicate values, the DISTINCT clause directs the function to disregard dupl icate values.

SQL QUERY-

```
SELECT AVG(SAL) FROM EMP
```



AVG(SAL)
2073.21428571428571428571428571428571429

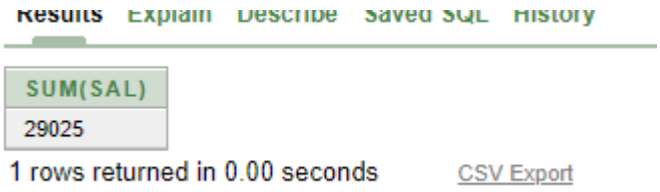
1 rows returned in 0.00 seconds CSV Export

3.SUM ()-

The aggregate function SUM () in Oracle returns the total of all or specific values in a collection of data. A clause that can be either DISTINCT or ALL is accepted by the Oracle SUM () function. The SUM () function is compelled by the DISTINCT clause to calculate the sum of distinct values. All values, including duplicates, are added together using the SUM () function thanks to the ALL clause.

SQL QUERY-

SELECT SUM(SAL) FROM EMP



Results Explain Describe Saved SQL History

SUM(SAL)
29025

1 rows returned in 0.00 seconds CSV Export

4.NVL ()-

When a null value is detected, you may replace it with another value using the Oracle/PLSQL NVL function.

SQL QUERY-

SELECT NVL (supplier_city, 'n/a') from supplier

Q5. like (%, _) between and operator with example?

1.LIKE (%, _)-

In SQL, the LIKE Operator is used to retrieve entries that match a specific pattern. The LIKE operator is used in a WHERE clause to s earch for a specific pattern in a column. There are two wildcard ch aracters in SQL, including:

SQL QUERY-

SELECT * from EMP where ENAME LIKE('J___S')

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7566	JONES	MANAGER	7839	02-APR-81	2975	-	20
7900	JAMES	CLERK	7698	03-DEC-81	950	-	30

SELEC
T * FROM EMP WHERE JOB LIKE('M%')

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7566	JONES	MANAGER	7839	02-APR-81	2975	-	20
7698	BLAKE	MANAGER	7839	01-MAY-81	2850	-	30
7782	CLARK	MANAGER	7839	09-JUN-81	2450	-	10

2.BETWEEN-

In SQL, the BETWEEN operator is used to choose values from a specified range. For instance, we must use the SQL BETWEEN query to retrieve just those entries where the person's age is between 20 and 25.

SQL QUERY-

SELECT * FROM EMP WHERE SAL BETWEEN '2000' AND '3000'

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7566	JONES	MANAGER	7839	02-APR-81	2975	-	20
7698	BLAKE	MANAGER	7839	01-MAY-81	2850	-	30
7782	CLARK	MANAGER	7839	09-JUN-81	2450	-	10
7788	SCOTT	ANALYST	7566	19-APR-87	3000	-	20
7902	FORD	ANALYST	7566	03-DEC-81	3000	-	20

Q6. Explain (% type attribute, %row type, %row count, % is open).

1.%type attribute-

The %TYPE attribute let's use the datatype of a field, record, nested table, database column, or variable in your own declarations, rather than hardcoding the type names. You can use the %TYPE attribute as a datatype specifier when declaring constants, variables, fields, and parameters

SQL QUERY-

TYPE type_record_type IS RECORD (emp_id employees.employee_id%TYPE)

2.%row type-

The %ROWTYPE attribute lets you declare a record that represents either a full or partial row of a database table or view.

For every visible column of the full or partial row, the record has a field with the same name and data type. If the structure of the row changes, then the structure of the record changes accordingly.

3.%Row count-

SQL Server @@ROWCOUNT is a system variable that is used to return the number of rows that are affected by the last executed statement in the batch.

The rows affecting statement can be any INSERT, UPDATE, DELETE or SELECT statement that is executed directly before the @@ROWCOUNT execution, taking into consideration that both the rows affecting statement and the system variable calling query are in the same execution.

Q7. Explain Pragma Exception.

Pragma is a keyword directive to execute proceed at compile time. pragma EXCEPTION_INIT function take these two arguments,

exception_name

error_number

You can define pragrma EXCEPTION_INIT in DECLARE BLOCK on your program.

exception_name and error_number define on yourself, where exception_name is character string up to 2048 bytes suppot and error_number is a negative integer range from -20000 to -20999.

Example: -

edit user-named_exp

```
DECLARE
   myex EXCEPTION;
   PRAGMA EXCEPTION_INIT(myex,-20015);
   n NUMBER := &n;
BEGIN
```

```
   FOR i IN 1..n LOOP
      dbms_output.put.line(i);
      IF i=n THEN
         RAISE myex;
      END IF;
   END LOOP;
EXCEPTION
   WHEN myex THEN
      dbms_output.put.line('loop finish');
END;
```

Q8. Create the following tables:

Table 1: DEPT

DEPTNO (NOT NULL, NUMBER (2)), DNAME (VARCHAR2 (14)),

LOC (VARCHAR2 (13)

Table 2: EMP

EMPNO (NOT NULL, NUMBER (4)), ENAME (VARCHAR2 (10)),

JOB (VARCHAR2 (9)), MGR (NUMBER (4)), HIREDATE (DATE),

SAL (NUMBER (7, 2)), COMM (NUMBER (7, 2)), DEPTNO (NUMBER (2))

MGR is the empno of the employee whom the employee reports to. DEPTNO is a foreign key.

1. List all the employees who have at least one person reporting to them.

```
SELECT ENAME FROM EMP WHERE EMPNO IN (SELECT MGR FROM EMP)
```

Results Ex

ENAME
JONES
BLAKE
CLARK
SCOTT
KING
FORD

2. List the department details in which more than 1 employees.

SELECT * FROM DEPT WHERE DEPTNO IN(SELECT DEPTNO FROM EMP GROUP BY DEPTNO HAVING COUNT(*) >1);

DEPTNO	DNAME	LOC
30	SALES	CHICAGO
20	RESEARCH	DALLAS
10	ACCOUNTING	NEW YORK

3. List the name of the employees with their immediate higher authority.

SELECT A.ENAME "EMPLOYEE", B.ENAME "REPORTS TO" FROM EMP A,EMP B WHERE A.MGR=B.EMPNO

EMPLOYEE	REPORTS TO
SMITH	FORD
ALLEN	BLAKE
WARD	BLAKE
JONES	KING
MARTIN	BLAKE
BLAKE	KING
CLARK	KING
SCOTT	JONES
TURNER	BLAKE
ADAMS	SCOTT

4. List all the employees who do not manage any one.

SELECT * FROM EMP WHERE EMPNO IN (SELECT EMPNO FROM EMP MINUS SELECT MGR FROM EMP)

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1987	Dep	-	-	-	-	-	10
7369	SMITH	CLERK	7902	17-DEC-80	800	-	20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7876	ADAMS	CLERK	7788	23-MAY-87	1100	-	20
7900	JAMES	CLERK	7698	03-DEC-81	950	-	30
7934	MILLER	CLERK	7782	23-JAN-82	1300	-	10

5. List the employee details whose salary is greater than the lowest salary of an employee belonging to deptno 20.

SELECT * FROM EMP WHERE SAL > (SELECT MIN(SAL) FROM EMP GROUP BY DEPTNO HAVING DEPTNO=20)

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975	-	20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850	-	30
7782	CLARK	MANAGER	7839	09-JUN-81	2450	-	10
7788	SCOTT	ANALYST	7566	19-APR-87	3000	-	20
7839	KING	PRESIDENT	-	17-NOV-81	5000	-	10

6. List the details of the employee earning more than the highest paid manager.

SELECT * FROM EMP WHERE SAL > (SELECT MAX(SAL) FROM EMP GROUP BY JOB HAVING JOB = 'MANAGER')

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7788	SCOTT	ANALYST	7566	19-APR-87	3000	-	20
7839	KING	PRESIDENT	-	17-NOV-81	5000	-	10
7902	FORD	ANALYST	7566	03-DEC-81	3000	-	20

7. List the highest salary paid for each job.

SELECT JOB, MAX(SAL) FROM EMP GROUP BY JOB;

CLERK	1300
SALESMAN	1600
PRESIDENT	5000
MANAGER	2975
ANALYST	3000

8. Find the most recently hired employee in each department.

SELECT * FROM EMP WHERE (DEPTNO, HIREDATE) IN (SELECT DEPTNO,MAX(HIREDATE) FROM EMP GROUP BY DEPTNO);

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7900	JAMES	CLERK	7698	03-DEC-81	950	-	30
7876	ADAMS	CLERK	7788	23-MAY-87	1100	-	20
7934	MILLER	CLERK	7782	23-JAN-82	1300	-	10

9. Which department has the highest annual remuneration bill?

SELECT DEPTNO, LPAD(SUM(12*(SAL+NVL(COMM,0))),15)"COMPENSATION" FROM EMP GROUP BY DEPTNO HAVINGSUM(12*(SAL+NVL(COMM,0))) = (SELECTMAX(SUM(12*(SAL+NVL(COMM,0)))) FROM EMP GROUP BY DEPTNO)

10. Write a correlated sub-query to list out the employees who earn more than the average salary.

SELECT ENAME,SAL FROM EMP E WHERE SAL > (SELECT AVG(SAL)FROM EMP F WHERE E.DEPTNO = F.DEPTNO)

ENAME	SAL
ALLEN	1600
JONES	2975
BLAKE	2850
SCOTT	3000
KING	5000
FORD	3000

Q9. Delete duplicate row from the table.

You should take the following actions to remove the duplicate rows from the SQL Server table:

Using the GROUP BY clause or the ROW NUMBER() function, locate duplicate records.

To get rid of the duplicate rows, use the DELETE command.

EXAMPLE: -

```
CREATE TABLE contacts(
   contact_id INT IDENTITY(1,1) PRIMARY KEY,
   first_name NVARCHAR(100) NOT NULL,
   last_name NVARCHAR(100) NOT NULL,
   email NVARCHAR(255) NOT NULL,
);
INSERT INTO contacts (first_name,last_name,email) VALUES
   ('sadu','rajawat','sadu.rajawat@example.com'),
   ('priyanshi','garg','priyanshi.garg@example.com'),
   ('vaishnavi','sharma','vaishnavi.sharma@example.com'),
   ('vaishnavi','sharma','vaishnavi.sharma@example.com'),
   ('vaishnavi','sharma','vaishnavi.sharma@example.com'),
   ('shivu','garg','shivu.garg@example.com'),
   ('shivu','garg','shivu.garg@example.com'),
```

WITH cte AS (SELECT contact_id, first_name, last_name, email, ROW_NUMBER() OVER (PARTITION BY first_name, last_name, email ORDER BY first_name, last_name, email) row_num FROM contacts)

DELETE FROM cte

WHERE row_num > 1;

Q.10. Create the following tables:

Employee (Emp_no: varchar2, Emp_name: varchar2, Emp_city varchar2) Company (Emp_no:Varchar2, Company_name: varchar2,Salary number)

CREATE TABLE EMP1(EMPNO Varchar2(12) PRIMARY KEY, ENAME varchar2(20), ECITY Varchar2(10))

CREATE TABLE COMP1(EMPNO REFERENCES EMP1, CNAME VARCHAR2(8), SALARY NUMBER(6))

INSERT INTO EMP1 VALUES(:EMPNO, :ENAME, :ECITY)

INSERT INTO COMP1 VALUES(:EMPNO, :CNAME, :SAL)

SELECT * FROM EMP1

EMPNO	ENAME	ECITY
4534	rahul	gwalior
5643	manoj	lucknow
9876	garima	delhi

SELECT * FROM COMP1

EMPNO	CNAME	SALARY
4534	suresh	56000
5643	shivam	45000
9876	deergha	57000

Q1.Write the query to display all employees working in ‘XYZ’ company.

SELECT ENAME FROM EMP1,COMP1 WHERE CNAME = 'rahul' AND EMP1.EMPNO = COMP1.EMPNO

ENAME
rahul

Q2. Write the query to display names of employees whose salary is the largest.

Q3. Perform ALTER and DELETE, TRUNCATE, and DROP commands.

ALTER TABLE COMP1 ADD (AGE INT)

EMPNO	CNAME	SALARY	AGE
4534	suresh	56000	-
5643	shivam	45000	-
9876	deergha	57000	-

DELETE FROM COMP1 WHERE CNAME ='SURESH'

1 row(s) deleted.

EMPNO	CNAME	SALARY	AGE
5643	shivam	45000	-
9876	deergha	57000	-

TRUNCATE TABLE COMP1

Table truncated.

SELECT * FROM COMP1

no data found

DROP TABLE COMP1

Table dropped.

SQL ASSIGNMENT IV

SET-B

EMPLOYEES6 TABLE

```
CREATE TABLE employees6 (
  EMPLOYEE_ID numeric(6) NOT NULL primary key,
  FIRST_NAME varchar2(20) DEFAULT NULL,
  LAST_NAME varchar2(25) NOT NULL,
  EMAIL varchar2(25) NOT NULL,
  PHONE_NUMBER varchar2(20) DEFAULT NULL,
  HIRE_DATE date NOT NULL,
  JOB_ID varchar2(10) NOT NULL,
  SALARY decimal(8,2) DEFAULT NULL,
  COMMISSION_PCT decimal(2,2) DEFAULT NULL,
  MANAGER_ID numeric(6) DEFAULT NULL,
  DEPARTMENT_ID numeric(4) DEFAULT NULL
);
```

TABLE OUTPUT

```
select * from employees6
```

Results Explain Describe Saved SQL History

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
100	Steven	King	SKING	515.123.4567	17-JUN-03	AD_PRES	24000	0	0	90
103	Ajay	Rajawat	THAKUR	590.423.4567	10-DEC-00	IT_PROG	9000	0	1	60
104	Bruce	sharma	pandit	590.423.4568	21-MAY-07	IT_PROG	6000	0	2	60
105	chitra	agrawal	charu	590.423.4569	04-NOV-05	IT_PROG	4800	0	2	60
106	Deepika	drexler	sis	590.423.4560	03-FEB-98	IT_PROG	4800	0	2	60
107	Eshita	bansal	buddy	590.423.5567	16-FEB-07	IT_PROG	4200	0	2	60
108	fanah	goyel	movie	515.124.4569	17-AUG-02	FI_MGR	12000	0	2	100
109	girja	drexler	mg	515.124.4169	01-OCT-78	FI_ACCOUNT	9000	0	4	100
110	honey	jha	hj	515.124.4269	28-SEP-05	FI_ACCOUNT	8200	0	4	100
111	Ismael	Sciarra	ISCIARRA	515.124.4369	30-SEP-05	FI_ACCOUNT	7700	0	4	100

More than 10 rows available. Increase rows selector to view more rows.

10 rows returned in 0.00 seconds CSV Export

DEPT_6 TABLE

```
CREATE TABLE dept_6 (
dept_id NUMBER(4) PRIMARY KEY,
dept_name VARCHAR2(30) NOT NULL ,
manager_id NUMBER(6) NOT NULL ,
loc_id NUMBER(4));
desc dept_6

insert into dept_6 values (1 , 'btech' , 66, 464)
insert into dept_6 values (2 , 'barc' , 62, 461)
insert into dept_6 values (3 , 'bba' , 63, 461)
```

```
insert into dept_6 values (4 , 'bsc' , 64, 462)
insert into dept_6 values (5 , 'bcom' , 65, 463)
insert into dept_6 values (6 , 'mba' , 66, 464)
insert into dept_6 values (7 , 'mtech' , 67, 465)
insert into dept_6 values (8 , 'btech' , 68, 466)
insert into dept_6 values (9 , 'barc' , 69, 467)
insert into dept_6 values (10 , 'bba' , 70, 468)
insert into dept_6 values (11 , 'bsc' , 71, 469)
insert into dept_6 values (12 , 'bcom' , 72, 470)
insert into dept_6 values (13 , 'mba' , 73, 471)
insert into dept_6 values (14, 'mtech' , 74, 472)
insert into dept_6 values (15 , 'mcom' , 75, 473)
insert into dept_6 values (16 , 'bed' , 76, 474)
insert into dept_6 values (17 , 'ba' , 77, 475)
```

TABLE OUTPUT

```
SELECT * FROM DEPT_6
```

Results Explain Describe Saved SQL History

DEPT_ID	DEPT_NAME	MANAGER_ID	LOC_ID
1	btech	66	465
2	barc	62	461
3	bba	63	461
4	bsc	64	462
5	bcom	65	463
6	mba	66	464
7	mtech	67	465
8	btech	68	466
9	barc	69	467
10	bba	70	468

More than 10 rows available. Increase rows selector to view more rows.

10 rows returned in 0.00 seconds CSV Export

QUESTIONS

1. Write a query that displays the last name (with the first letter uppercase and all other letters lowercase) and the length of the last name for all employees whose name starts with the letters J, A, or M. Give each column an appropriate label. Sort the results by the employees' last name.

```
select initcap(last_name) "Name", length(last_name) "Length of Name"
from employees6
where last_name like 'J%' or last_name like 'A%' or last_name like 'M%'
order by last_name;
```

Results Explain Describe Saved SQL History

no data found

2. Create a query to display the last name and salary for all employees. Format the salary to be 15 characters long, left-padded with the $ symbol. Label the column SALARY.

```
select last_name, lpad(salary,15,'$') Salary
from employees6;
```

Results Explain Describe Saved SQL History

LAST_NAME	SALARY
King	$$$$$$$$$24000
Rajawat	$$$$$$$$$$9000
sharma	$$$$$$$$$$6000
agrawal	$$$$$$$$$$4800
drexler	$$$$$$$$$$4800
bansal	$$$$$$$$$$4200
goyel	$$$$$$$$$12000
drexler	$$$$$$$$$$9000
jha	$$$$$$$$$$8200
Sciarra	$$$$$$$$$$7700

More than 10 rows available. Increase rows selector to view more rows.

10 rows returned in 0.01 seconds CSV Export

3. Find the highest, lowest, sum, and average salary of all employees for each job type. Label the columns Maximum, Minimum, Sum, and Average, respectively. Round your results to the nearest whole number.

```
SELECT ROUND(MAX(salary),0) "Maximum",
       ROUND(MIN(salary),0) "Minimum",
       ROUND(SUM(salary),0) "Sum",
       ROUND(AVG(salary),0) "Average"
FROM   employees6;
```

Results Explain Describe Saved SQL History

Maximum	Minimum	Sum	Average
24000	2200	185800	7146

1 rows returned in 0.00 seconds CSV Export

4. The HR department needs a report on job grades and salaries. To familiarize yourself with the JOB_GRADES table, first show the structure of the JOB_GRADES table. Then create a query that displays the name, job, department name, salary, and grade for all employees.

```
DESC JOB_GRADES
SELECT e.last_name, e.job_id, d.department_name,
e.salary, j.grade_level
```

```
FROM   employees e JOIN departments d
ON     (e.department_id = d.department_id)
JOIN   job_grades j
ON    (e.salary BETWEEN j.lowest_sal AND j.highest_sal);
```

5. The HR department needs a report that displays the last name, department number, and job ID of all employees whose department location ID is 1700.

```
select last name, department id, job id
from employees6
where department id in (select dept id from dept_6 where loc id =1700);
```

Results Explain Describe Saved SQL History

no data found

6. Create a report that displays the employee number, last name, and salary of all employees who earn more than the average salary. Sort the results in order of ascending salary.

```
select employee_id, last_name, salary
from employees6
where salary > (select avg(salary) from employees6)
order by salary;
```

Results Explain Describe Saved SQL History

EMPLOYEE_ID	LAST_NAME	SALARY
160	Doran	7500
111	Sciarra	7700
112	drexler	7800
122	tomar	7900
159	share	8000
110	jha	8200
103	Rajawat	9000
158	agrawal	9000
109	drexler	9000
157	jha	9500
More than 10 rows available. Increase rows selector to view more rows.		

10 rows returned in 0.00 seconds CSV Export

7. The HR department needs a list of department IDs for departments that do not contain the job ID ST_CLERK. Use set operators to create this report.

```
SELECT dept_id
FROM   dept_6
MINUS
SELECT department_id
FROM
  employees6
WHERE  job_id = 'ST_CLERK';
```

Results Explain Describe Saved SQL History

DEPT_ID
1
2
3
4
5
6
7
8
9
10
More than 10 rows available. Increase rows selector to view more rows.

10 rows returned in 0.00 seconds CSV Export

8. **Change the last name of employee 3 to Drexler**.

```
update employees6
set last_name = 'drexler'
where last_name = 'garg'
```

Results Explain Describe Saved SQL History

4 row(s) updated.

9. **The staff in the HR department wants to hide some of the data in the EMPLOYEES table. They want a view called EMPLOYEES_VU based on the employee numbers, employee names, and department numbers from the EMPLOYEES table. They want the heading for the employee name to be EMPLOYEE.**

```
CREATE OR REPLACE VIEW employees_vu AS
    SELECT employee_id, last_name employee, department_id
    FROM employees6;
```

Results Explain Describe Saved SQL History

View created.

0.00 seconds

10. **The HR department needs a report that displays the last name and hire date for all employees who were hired in 1973.**

```
select last_name, hire_date
from employees6
where hire_date like '%73';
```

Results Explain Describe Saved SQL History

LAST_NAME	HIRE_DATE
drexler	23-JAN-73

1 rows returned in 0.00 seconds CSV Export

11. Department 50 needs access to its employee data. Create a view named DEPT50 that contains the employee numbers, employee last names, and department numbers for all employees in department 50. You have been asked to label the view columns EMPNO, EMPLOYEE, and DEPTNO.

```
CREATE VIEW dept50 AS
   SELECT   employee_id empno, last_name employee,
            department_id deptno
   FROM     employees6
   WHERE    department_id = 50
   WITH CHECK OPTION CONSTRAINT emp_dept_50;
```

Results Explain Describe Saved SQL History

```
View created.
```

DBA COMMANDS

1 connect to dbA

SQL> conn system/UP93bj2448();

```
SQL> conn system/UP93bj2448();
Connected.
```

2 CREATE USER

SQL> create user karan1 identified by 1256;

```
SQL> create user karan1 identified by 1256;

User created.
```

2.2 change user password

SQL> alter user karan1 identified by 1234;

```
SQL> alter user karan1 identified by 1234;

User altered.
```

3 CHANGE GRACE TIME OF USER

SQL> Create profile prorest1 limit Failed_Login_Attempts 3;

SQL> Alter user karan1 profile prorest;

```
SQL> Create profile prorest1 limit Failed_Login_Attempts 3;

Profile created.

SQL> Alter user karan1 profile prorest;

User altered.
```

4 UNLOCK USER IF LOCKED AFTER GIVEN ATTEMPTS

```
SQL> conn karan1/98675;
ERROR:
ORA-01017: invalid username/password; logon denied


Warning: You are no longer connected to ORACLE.
SQL> conn karan1/98678;
ERROR:
ORA-01017: invalid username/password; logon denied


SQL> conn karan1/98643;
ERROR:
ORA-01017: invalid username/password; logon denied


SQL> conn karan1/98634;
ERROR:
ORA-28000: the account is locked
```

SQL> alter user karan1 account lock;

```
SQL> conn system/UP93bj2448();
Connected.
SQL> alter user karan1 account lock;

User altered.
```

5 GRANT PERMISSIION TO USER USING GRANT KEYWORD

a) **to access specified table**

SQL> Grant select on relatives_12 to karan1;

```
SQL> Grant select on relatives_12 to karan1;

Grant succeeded.
```

b) **to allow it to update the table**

SQL> Grant select , insert on relatives_12 to karan1;

```
SQL> Grant select , insert on relatives_12 to karan1;

Grant succeeded.
```

c) **to allow it to change schema of table**

SQL> Grant all on relatives_12 to karan1;

```
SQL> Grant all on relatives_12 to karan1;

Grant succeeded.
```