RDBMS (Diploma 3rd Computer Engineering)

Unit : 2

SQL inbuilt functions and joins

Prepared by:

Mr. Uresh N. Parmar

Lecturer, Computer Engg. Dept.
C U Shah Govt. Polytechnic
Surendranagar

CO (Course Outcome)
&
UOs (Unit Outcomes)

CO (a)

Perform joins, subqueries and nested queries on multiple tables using SQL*plus

UOs

2a. Execute various SQL operators and functions.

2b. Perform queries on 'Group by', 'Having' and 'Order by' clause

2c. Implement joins

2d. Perform different types of sub queries

2.1 SQL Operators and Functions

❖ Concept of Dual table

- Mainly used to check mathematical expressions, to know system date-time and to know some manipulations. Kind of work table for Oracle.
- It is a dummy table provided by Oracle.
- It is a special one-row, one-column table.
- Dual table is a part of data dictionary and you can't make change with its schema.
- **Description of Dual table:**

```
SQL> DESC dual;
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------
 DUMMY                                              VARCHAR2(1)
```

2.1 SQL Operators and Functions

- A simple query for Dual table is like:

```
SQL> SELECT * FROM dual;

D
-
X
```

D stands for Dummy and X is a dummy single character

Example (*Calculate the result of mathematical expression 10 * 5*)

```
SQL> SELECT 10 * 5 FROM dual;

      10*5
----------
        50
```

Example (*Calculate the result of mathematical expression 10 * 5*)

```
SQL> SELECT USER FROM dual;

USER
------------------------------
SYSTEM
```

2.1.1 SQL Operators

- Basic concepts of operators.
- SQL operator can be divided into main six categories:
  1. ***Arithmetic Operators***
  2. ***Relational Operators (Comparison Operator)***
  3. ***Logical Operators***
  4. ***Range searching Operators***
  5. ***Set searching Operators***
  6. ***Character Operators***

First three operators are there in GTU syllabus

2.1.1 SQL Operators

- Consider following table '**person**' on which we will apply further operations.

SR	NAME	SURNAME	BDATE	CITY	BALANCE
1	bhagya	parikh	15-JAN-10	surendranagar	15000
2	vihan	sagar	22-FEB-12	rangpur	10000
3	shreeja	parikh	22-JUL-10	rajkot	5000
4	samay	saxena	10-AUG-10	ahmedabad	1000
5	tushar	parmar	13-NOV-90	palanpur	19000
6	aarush	parikh	15-JAN-15	surendranagar	22000
7	avani	solanki	15-MAY-10	ahmedabad	1000
8	misha	sutariya	22-DEC-12	ahmedabad	55000
9	parth	parikh	15-SEP-95	rajkot	5000
10	aarav	solanki	11-SEP-10	surendranagar	15000

2.1.1 SQL Operators

1. Arithmetic Operators

- Perform arithmetic operations on numerical operands. Also can be performed on the data stored in a table.
- Different arithmetic operators are:

- Addition	- Subtraction	- Multiplication
- Division	- Modulus (%)	- Enclosed operation ( )

Example (*on Dual table*)

```
SELECT 5+3, 5-3, 5*3, 5/3 FROM dual;

  5+3        5-3        5*3        5/3
----- ---------- ---------- ----------
    8          2         15 1.66666667
```

2.1.1 SQL Operators

Example (*on person table*)

Display serial number, name and balance from table 'person' by adding 100 rupees in balance of every person.

```
SELECT sr,name,balance "CURRENT BAL", balance+100 "NEW BAL"
FROM person;
```

SR	NAME	CURRENT BAL	NEW BAL
1	bhagya	15000	15100
2	vihan	10000	10100
3	shreeja	5000	5100
4	samay	1000	1100
5	tushar	19000	19100
6	aarush	22000	22100
7	avani	200	300
8	misha	55000	55100
9	parth	5000	5100
10	aarav	1200	1300

2.1.1 SQL Operators

2. Relational Operators

- Used to compare one expression with other.
- In result, it returns → TRUE / FALSE / unknown.
- SQL supports following relational operators.

Operator	Description
=	Equals
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
!= or <>	Not equals

Example (*on Dual table*)

2.1.1 SQL Operators

Example (*Display details of persons whose balance is greater than 15000*)

```
SQL> SELECT * FROM person WHERE balance > 15000;

        SR NAME       SURNAME    BDATE     CITY              BALANCE
---------- ---------- ---------- --------- -------------- ----------
         5 tushar     parmar     13-NOV-90 palanpur            19000
         6 aarush     parikh     15-JAN-15 surendranagar       22000
         8 misha      sutariya   22-DEC-12 ahmedabad           55000
```

Example (*Give the details of those persons who are living in surendranagar city*)

```
SQL> SELECT * FROM person
  2  WHERE city = 'surendranagar';

        SR NAME       SURNAME    BDATE     CITY              BALANCE
---------- ---------- ---------- --------- -------------- ----------
         1 bhagya     parikh     15-JAN-10 surendranagar       15000
         6 aarush     parikh     15-JAN-15 surendranagar       22000
        10 aarav      solanki    11-SEP-10 surendranagar       15000
```

2.1.1 SQL Operators

3. Logical Operators

- Compare two or more conditions at a time and generate result.
- SQL provides three logical operators → AND, OR, NOT

Operator	Description
AND	Logical AND compares between two Booleans as expression and returns True when both expressions are true. False otherwise.
OR	Logical OR compares between two Booleans as expression and returns True when one of the expressions is true. False otherwise.
NOT	Not takes a single Boolean as an argument and changes its value from *false to true* or from *true to false*.

2.1.1 SQL Operators

➥ **AND operator**

- It compares between two Booleans as expression and returns True when both expressions are true. False otherwise
- If you want to select rows that must satisfy all the given conditions, you can use the logical operator - AND.

Example (*Display name, surname and balance of all persons whose city=Ahmadabad and balance is less than 5000*)

```
SQL> SELECT name,city,balance FROM person
  2  WHERE city='ahmedabad' AND balance < 5000;

NAME       CITY           BALANCE
---------- ---------- ----------
samay      ahmedabad        1000
avani      ahmedabad         200
```

2.1.1 SQL Operators

➥ **OR operator**

- It compares between two Booleans as expression and returns True when one of the expressions is true. False otherwise.
- If you want to select rows that satisfy one of the given conditions, you can use the logical operator, OR.

  Example (*Display name, surname and balance of all persons who are living in Ahmadabad or balance is less than 5000*)

```
SQL> SELECT name,city,balance FROM person
  2  WHERE city='ahmedabad' OR balance < 5000;

NAME       CITY           BALANCE
---------- ---------- ----------
samay      ahmedabad        1000
avani      ahmedabad         200
misha      ahmedabad       55000
aarav      snagar           1200
```

2.1.1 SQL Operators

➥ **NOT operator**

- It takes a single argument as input and changes its value from False to True or from True to False.
- It is used to negate a condition.
- If you want to find rows that do not satisfy a particular condition, you can use the logical operator NOT.
- NOT results in the reverse of a condition. That is, if a condition is satisfied, then that particular row is not returned.

Example (*Display records of all the persons who do not belong to the city Ahmadabad*)

```
SQL> SELECT * FROM person WHERE NOT city = 'ahmedabad';

        SR NAME       SURNAME    BDATE     CITY                BALANCE
---------- ---------- ---------- --------- --------------- ----------
         1 bhagya     parikh     15-JAN-10 surendranagar        15000
         2 vihan      sagar      22-FEB-12 rangpur              10000
         3 shreeja    parikh     22-JUL-10 rajkot                5000
         5 tushar     parmar     13-NOV-90 palanpur             19000
         6 aarush     parikh     15-JAN-15 surendranagar        22000
         9 parth      parikh     15-SEP-95 rajkot                5000
        10 aarav      solanki    11-SEP-10 surendranagar        15000
```

2.1.1 SQL Operators

4. Range searching operator (BETWEEN) *Not in GTU syllabus*

- It is used to check between two values or specific range.

Syntax

.... ColumnName BETWEEN lower limit AND upper limit;

Example (*Display persons having balance between 10000 and 20000*)

```
SELECT * FROM PERSON WHERE balance BETWEEN 10000 AND 20000;

   SR NAME       SURNAME    BDATE     CITY              BALANCE
----- ---------- ---------- --------- -------------- ----------
    1 bhagya     parikh     15-JAN-10 surendranagar         15000
    2 vihan      sagar      22-FEB-12 rangpur               10000
    5 tushar     parmar     13-NOV-90 palanpur              19000
   10 aarav      solanki    11-SEP-10 surendranagar         15000
```

2.1.1 SQL Operators

5. Set searching operator (IN) *Not in GTU syllabus*

- It is used to select rows that match one of the values in a list.
- This is similar to '='. But, '=' compares single value to another single value, while IN compares single value to a list (set) of values provided with IN predicate.
- It is like '*multiple OR*'

Syntax

.... ColumnName IN (value1, value2,..., valueN);

Example (*Display persons who belong to city 'ahmedabad', 'rangpur' or 'palanpur'*)

```
SELECT * FROM person
WHERE city IN ('ahmedabad', 'rangpur', 'palanpur');

   SR NAME       SURNAME    BDATE     CITY              BALANCE
----- ---------- ---------- --------- -------------- ----------
    2 vihan      sagar      22-FEB-12 rangpur             10000
    4 samay      saxena     10-AUG-10 ahmedabad            1000
    5 tushar     parmar     13-NOV-90 palanpur            19000
    7 avani      solanki    15-MAY-10 ahmedabad            1000
    8 misha      sutariya   22-DEC-12 ahmedabad           55000
```

2.1.1 SQL Operators

➥ **NOT IN** ***Not in GTU syllabus***

- It works exactly opposite of IN operator.

Syntax

…. ColumnName NOT IN (value1, value2,…, valueN);

Example (*Display record of all the persons who are not living in the city 'ahmedabad', 'rangpur' or 'palanpur'*)

```
SQL> SELECT * FROM person
  2  WHERE city NOT IN ('ahmedabad', 'rangpur','palanpur');

        SR NAME       SURNAME    BDATE     CITY              BALANCE
---------- ---------- ---------- --------- -------------- ----------
         1 bhagya     parikh     15-JAN-10 surendranagar       15000
         3 shreeja    parikh     22-JUL-10 rajkot               5000
         6 aarush     parikh     15-JAN-15 surendranagar       22000
         9 parth      parikh     15-SEP-95 rajkot               5000
        10 aarav      solanki    11-SEP-10 surendranagar       15000
```

2.1.1 SQL Operators

6. Character operator (LIKE) *Not in GTU syllabus*

- It is mainly used for pattern matching within string.
- This is similar to '='. BUT, the '=' operator compares for exact matching. While LIKE compares for pattern similarity.

Syntax

.... ColumnName LIKE pattern;

- It is used with two special characters → % and _

Example1 (*Display name of persons starting with 'a'*)

```
SQL> SELECT name FROM person WHERE name LIKE 'a%';

NAME
----------
aarush
avani
aarav
```

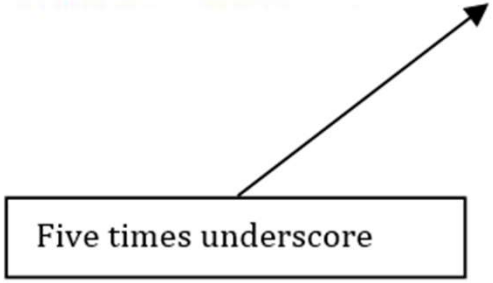
2.1.1 SQL Operators

Example2 (*Display the name of persons have 'a' as second character in their name*)

```
SQL> SELECT name FROM person WHERE name LIKE '_a%';

NAME
----------
samay
aarush
parth
aarav
```

Example3 (*Display name and surname of persons having 5 characters in their name*)

```
SQL> SELECT name, surname FROM person WHERE name LIKE '_____';

NAME       SURNAME
---------- ----------
vihan      sagar
samay      saxena
avani      solanki
misha      sutariya
parth      parikh
aarav      solanki

6 rows selected.
```



2.1.1 SQL Operators

➥ **Concatenation operator (|| operator)** ***Not in GTU syllabus***

- It is used to link two or more columns.
- It also concatenates two expressions.
- It is independent of the data types of a column.

Example (*Display person full name (combing name and surname) from 'person' table*)

```
SQL> SELECT name || surname "FULL NAME"
  2  FROM person;

FULL NAME
--------------------
bhagyaparikh
vihansagar
shreejaparikh
samaysaxena
tusharparmar
aarushparikh
avanisolanki
mishasutariya
parthparikh
aaravsolanki

10 rows selected.
```

2.1.1 SQL Operators

➥ **IS NULL (Extra)** ***Not in GTU syllabus***

- In it, we can get the records from table where NULL values are available in particular column.
- Mainly used for data analysis.

Syntax

....Column_name IS NULL;

Example (Consider the employee table)

EMPLOYEE_ID	EMPLOYEE_NAME	HIRE_DATE	SALARY	MANAGER_ID	DEPT_ID
101	bhagya	12-JAN-20	60000		1
102	aarush	15-JAN-21	11000	101	2
103	avani	02-JUL-10	50000	104	3
104	aarav	05-AUG-22	6000		2
105	shreeja	18-JUL-18	5000	101	4
106	kahaan	18-FEB-15	16000	107	4
107	dhairya	08-NOV-20	18000		5
108	tanzil	08-FEB-15	45000	107	2
109	ghata	03-DEC-18	4600	104	5
110	samay	29-SEP-20	41500		3

10 rows selected.

2.1.1 SQL Operators

Now if we want to find the details of all the managers, we can see, in a record if manager_id is null then he/she is the manager. To get that data we can use IS NULL operator on manager_id column, so that we get only those records which have NULL value in manager_id column.

```
SQL> SELECT * FROM employees
  2  WHERE manager_id IS NULL;

EMPLOYEE_ID EMPLOYEE_NAME   HIRE_DATE     SALARY MANAGER_ID    DEPT_ID
----------- -------------- --------- ---------- ---------- ----------
        101 bhagya         12-JAN-20      60000                     1
        104 aarav          05-AUG-22       6000                     2
        107 dhairya        08-NOV-20      18000                     5
        110 samay          29-SEP-20      41500                     3
```

2.1.2 SQL In-built Functions

- These are pre defined functions by Oracle. Also called *Oracle functions.*
- These functions are useful while performing mathematical calculations, string concatenations, sub-strings etc.
- The general format for SQL function is:

Function_name (argument1, argument2 ,..., argumentN)

- SQL functions divided into two broad categories:
  - ***i. Aggregate functions (Group functions)***
  - ***ii. Scalar functions (Single row functions)***

2.1.2 Aggregate Functions

- Also called group functions as they operate on multiple records and returns a single value for group of values processed. For example MAX( ) is an aggregate function which finds maximum number out of set of numbers and returns single value as an answer.

- Different aggregate functions in SQL are:
  - ***MAX ( )***
  - ***MIN ( )***
  - ***SUM ( )***
  - ***AVG ( )***
  - ***COUNT (*)***
  - ***COUNT ( )***

2.1.2 Aggregate Functions (MAX)

❑ MAX ( )

- Returns maximum value for the given column.

Syntax

```
MAX (ColumnName)
```

Example (*Find out maximum balance from the table 'person'*)

```
SQL> SELECT MAX(balance) FROM person;

MAX(BALANCE)
------------
       55000
```

- It can be applied on character and date data type also.

```
SQL> SELECT MAX(NAME) FROM person;

MAX(NAME)
----------
vihan
```

2.1.2 Aggregate Functions (MIN)

❑ MIN ( )

- Returns minimum value for the given column.

Syntax

MIN (ColumnName)

Example (*Find out minimum balance from the table 'person'*)

```
SQL> SELECT MIN(balance) FROM person;

MIN(BALANCE)
------------
         200
```

- It can be applied on character and date data type also.

```
SQL> SELECT MIN(NAME) FROM person;

MIN(NAME)
----------
aarav
```

2.1.2 Aggregate Functions (SUM)

❑ SUM ( )

- It returns the sum of all the values of the selected column.

Syntax

SUM (ColumnName) OR SUM ([Distinct] ColumnName)

Example (*Find out some of all balance and distinct balance from person table*)

```
SQL> SELECT SUM(balance), SUM(DISTINCT balance) FROM person;

SUM(BALANCE) SUM(DISTINCTBALANCE)
------------ --------------------
      133400                128400
```

2.1.2 Aggregate Functions (AVG)

❑ AVG ( )

- It returns the average of all the values of the given input column.

Syntax

AVG ([Distinct] ColumnName)

Example (*Find average and distinct average of Balance from person table*)

```
SQL> SELECT AVG(balance), AVG(DISTINCT balance)
  2  FROM person;

AVG(BALANCE) AVG(DISTINCTBALANCE)
------------ --------------------
       14800              18142.8571
```

2.1.2 Aggregate Functions (COUNT(*))

❑ COUNT (*)

- It returns total number of rows including NULL in a table.

Syntax

```
COUNT (*)
```

Example (*Find the number of rows in person table*)

```
SQL> SELECT COUNT(*) FROM person;

  COUNT(*)
----------
        10
```

2.1.2 Aggregate Functions (COUNT( ))

❑ COUNT ( )

- It returns total number of rows in a column of a table, do not consider NULL values. So, it returns total number of Non-NULL records (rows) in a column.
- It can be used with DISTINCT to find out total number of unique values in a column. By default, 'All' is considered.

Syntax

COUNT([Distinct] ColumnName)

Example (*Find number of city and distinct number of cities from person table*)

```
SQL> SELECT COUNT(city), COUNT(DISTINCT city) FROM person;

COUNT(CITY) COUNT(DISTINCTCITY)
----------- -------------------
         10                   5
```

2.1.2 Scalar Functions

- Functions operate on a single row (or value) are called scalar or single row function.
- Operate on a single record and return single value for each single record processed
- Some important scalar functions are:
  - **Date functions**
  - **Numeric functions**
  - **Character functions**
  - **Conversion functions**

2.1.2 Scalar Functions (Date functions)

- These functions are kind of single row functions or scalar functions.
- Date should be in the form of 'DD-MON-YY' while passed as argument and it should be enclosed with single quotes.
- SYSDATE can also be used.
- Now we will discuss some important date functions.

❖ ADD_MONTHS ( )

Syntax

ADD_MONTHS (date, n)

- This function is a scalar function or single row function.
- It is a kind of date function.
- This function returns new date after adding n months in the specified date.
- If n is a negative, then n month will be subtracted.

2.1.2 Scalar Functions (Date functions)

Example

```
SQL> SELECT ADD_MONTHS(sysdate, 1), ADD_MONTHS(sysdate, -1)
  2  FROM dual;

ADD_MONTH ADD_MONTH
--------- ---------
31-MAY-24 31-MAR-24
```

❖ MONTHS_BETWEEN ( )

Syntax

MONTHS_BETWEEN (date1, date2)

- This function is a scalar function or single row function.
- It is a kind of date function.
- This function returns number of months between date1 and date2.

2.1.2 Scalar Functions (Date functions)

- date2 will be subtracted from date1 in that case.

Example

```
SQL> SELECT MONTHS_BETWEEN(sysdate, '13-NOV-99')
  2  FROM dual;

MONTHS_BETWEEN(SYSDATE,'13-NOV-99')
-----------------------------------
                          293.563973

SQL> SELECT MONTHS_BETWEEN('13-NOV-99', sysdate)
  2  FROM dual;

MONTHS_BETWEEN('13-NOV-99',SYSDATE)
-----------------------------------
                          -293.56398
```

Example2

Calculate the age of all persons from the table person.

```
SQL> SELECT name, bdate, FLOOR(MONTHS_BETWEEN(SYSDATE,bdate)/12) "AGE"
  2  FROM person;
```

2.1.2 Scalar Functions (Date functions)

❖ ROUND ( )

Syntax

ROUND (date, unit)

- This function is a scalar function or single row function.
- It is a kind of date function.
- This function returns rounded date value to a specific unit of time, such as rounding to the nearest day, month, or year. Unit is optional.

Example

```
SQL> SELECT ROUND(sysdate) "Rounded Date" FROM dual;

Rounded D
---------
30-APR-24

SQL> SELECT ROUND(sysdate, 'year') FROM dual;

ROUND(SYS
---------
01-JAN-24
```

2.1.2 Scalar Functions (Date functions)

❖ TRUNC ( )

Syntax

TRUNC (date, unit)

- This function is a scalar function or single row function.
- It is a kind of date function.
- This function returns truncated date value to a specific unit of time, effectively rounding it down to a specific unit of time such as day, month, or year. Unit is optional.

Example

```
SQL> SELECT TRUNC(sysdate, 'day') FROM dual;

TRUNC(SYS
---------
28-APR-24

SQL> SELECT TRUNC(sysdate, 'month') FROM dual;

TRUNC(SYS
---------
01-APR-24
```

2.1.2 Scalar Functions (Date functions)

❖ NEXT_DAY ( ) *Not in GTU syllabus*

Syntax

NEXT_DAY (date, day)

- This function is a scalar function or single row function.
- It is a kind of date function.
- This function returns the date of next named week-day specified by day relative to date.

Example

```
SQL> SELECT NEXT_DAY(sysdate, 'monday') FROM dual;

NEXT_DAY(
---------
06-MAY-24

SQL> SELECT NEXT_DAY(TO_DATE('13-nov-81'), 'monday') FROM dual;

NEXT_DAY(
---------
16-NOV-81
```

2.1.2 Scalar Functions (Date functions)

❖ LAST_DAY ( ) *Not in GTU syllabus*

Syntax

LAST_DAY (date)

- This function is a scalar function or single row function.
- It is a kind of date function.
- This function returns the last date of the month specified by date.

Example

```
SQL> SELECT LAST_DAY(sysdate) FROM dual;

LAST_DAY(
---------
30-APR-24

SQL> SELECT LAST_DAY(TO_DATE('13-nov-81')) FROM dual;

LAST_DAY(
---------
30-NOV-81
```

2.1.2 Scalar Functions (Date functions)

❖ EXTRACT ( ) *Not in GTU syllabus*

Syntax

EXTRACT (part_of_date from Date value)

- This function is a scalar function or single row function.
- It is a kind of date function.
- This function used to extract any part of the date (i.e. day, month, year) from the given date value as input. The input should be a date data type value.

Example

```
SQL> SELECT EXTRACT (day from sysdate), EXTRACT (month from sysdate),
  2  EXTRACT (year from sysdate) FROM DUAL;

EXTRACT(DAYFROMSYSDATE) EXTRACT(MONTHFROMSYSDATE) EXTRACT(YEARFROMSYSDATE)
----------------------- ------------------------- -------------------------
                     12                         5                      2024
```

2.1.2 Scalar Functions (Numeric functions)

- Numeric Functions are used to perform operations on numbers and return numbers in output.
- These functions are scalar functions or single row functions.
- Some important numeric functions are explained below.

❖ ABS ( )

Syntax

ABS (n)

- This function is a scalar function or single row function.
- It is a kind of numeric function.
- It returns absolute value of 'n'. Negative numbers are converted into positive.

Example

```
SQL> SELECT ABS(-10),ABS(10),ABS(10.5) FROM dual;

  ABS(-10)    ABS(10)  ABS(10.5)
---------- ---------- ----------
        10         10       10.5
```

2.1.2 Scalar Functions (Numeric functions)

❖ SQRT ( )

Syntax

SQRT (n)

- This function is a scalar function or single row function.
- It is a kind of numeric function.
- It returns square root of n. Here n cannot be negative.

Example

```
SQL> SELECT SQRT(25), SQRT(24) FROM dual;

  SQRT(25)   SQRT(24)
---------- ----------
         5 4.89897949
```

2.1.2 Scalar Functions (Numeric functions)

➥ POWER

Syntax:

POWER (m, n)

- This function is a scalar function or single row function.
- It is a kind of numeric function.
- It returns '*m*' raised to the n^{th} power.

Example:

```
SQL> SELECT POWER(3,2) FROM dual;

POWER(3,2)
----------
         9
```

➥ MOD

Syntax:

MOD (m, n)

- This function is a scalar function or single row function.
- It is a kind of numeric function.
- It returns remainder of '*m*' divide by '*n*'.

Example:

```
SQL> SELECT MOD(8,3) FROM dual;

  MOD(8,3)
----------
         2
```

2.1.2 Scalar Functions (Numeric functions)

➥ CEIL *(not in GTU syllabus)*

Syntax:

```
CEIL (n)
```

- This function is a scalar function or single row function.
- It is a kind of numeric function.
- It returns the smallest integer (whole number) value that is greater than or equal to '***n***'.

Example:

```
SQL> SELECT CEIL(10.6), CEIL(10.2), CEIL(-10.6)
  2  FROM dual;

CEIL(10.6) CEIL(10.2) CEIL(-10.6)
---------- ---------- -----------
        11         11         -10
```

➥ FLOOR *(not in GTU syllabus)*

Syntax:

```
FLOOR (n)
```

- This function is a scalar function or single row function.
- It is a kind of numeric function.
- It returns the largest integer (whole number) value that is less than or equal to '***n***'.

Example:

```
SQL> SELECT FLOOR(10.6), FLOOR(10.2), FLOOR(-10.6)
  2  FROM dual;

FLOOR(10.6) FLOOR(10.2) FLOOR(-10.6)
----------- ----------- ------------
         10          10          -11
```

2.1.2 Scalar Functions (Numeric functions)

❖ ROUND ( )

Syntax

ROUND (m, n)

- It is a scalar function and a kind of numeric function.
- It returns rounded of m to n decimal places to the right.
- If n is omitted, m is rounded to 0 places.
- If n is negative, m is rounded to n places to the left of a decimal point.
- n must be an integer value.

Example

```
SQL> SELECT ROUND (123.45,2) FROM dual;

ROUND(123.45,2)
---------------
         123.45

SQL> SELECT ROUND (123.45) FROM dual;

ROUND(123.45)
-------------
          123
```

```
SQL> SELECT ROUND (123.45,-1) FROM dual;

ROUND(123.45,-1)
----------------
             120

SQL> SELECT ROUND (123.45,-2) FROM dual;

ROUND(123.45,-2)
----------------
             100
```

2.1.2 Scalar Functions (Numeric functions)

❖ TRUNC ( )

- This function is a scalar function or single row function.
- It is a kind of numeric function.
- This function returns a number truncated value to a certain number of decimal places.

Syntax

TRUNC (m, n)

- If n is positive, m is truncated to n places to the right of a decimal point.
- If n is omitted, m is truncated to 0 places.
- If n is negative, m is truncated to n places to the left of a decimal point.

2.1.2 Scalar Functions (Numeric functions)

Example

```
SQL> SELECT TRUNC(123.456,2) FROM dual;

TRUNC(123.456,2)
----------------
          123.45

SQL> SELECT TRUNC(123.456) FROM dual;

TRUNC(123.456)
--------------
           123

SQL> SELECT TRUNC(123.456,-2) FROM dual;

TRUNC(123.456,-2)
-----------------
              100
```

2.1.2 Scalar Functions (Numeric functions)

Difference between TRUNC and ROUND by example.

```
SQL> select ROUND(456.789,2), TRUNC(456.789,2) from dual;

ROUND(456.789,2) TRUNC(456.789,2)
---------------- ----------------
          456.79           456.78

SQL> select ROUND(135.356,2), TRUNC(135.356,2) from dual;

ROUND(135.356,2) TRUNC(135.356,2)
---------------- ----------------
          135.36           135.35

SQL> select ROUND(153.456,-2), TRUNC(153.456,-2) from dual;

ROUND(153.456,-2) TRUNC(153.456,-2)
----------------- -----------------
              200               100
```

2.1.2 Scalar Functions (Character functions)

- These are scalar or single row functions.
- Accept character as inputs and can return either characters or number values as output.
- Character functions are of two types:

1. **Case-Manipulative Functions**

   LOWER, UPPER and INITCAP

2. **Character-Manipulative Functions**

   CONCAT, LENGTH, SUBSTR, INSTR, LPAD, RPAD, TRIM, TRANSLATE and REPLACE

Important note: *if any character function asked in the examination, first two lines you have to write like:*

- This function is a scalar function or single row function.

- It is a kind of character function.

And accordingly you can write in any type of scalar function.

2.1.2 Scalar Functions (Character functions)

❖ LENGTH ( )

- This function returns the length of the string (number of characters of string).

Syntax

LENGTH (string)

Example

```
SQL> SELECT LENGTH('hello') FROM dual;

LENGTH('HELLO')
---------------
              5

SQL> SELECT name, LENGTH(name) FROM person;

NAME       LENGTH(NAME)
---------- ------------
bhagya                6
vihan                 5
shreeja               7
samay                 5
tushar                6
aarush                6
avani                 5
misha                 5
parth                 5
aarav                 5
```

2.1.2 Scalar Functions (Character functions)

❖ LOWER

Syntax:

LOWER (string)

- This function is a scalar function or single row function.
- It is a kind of character function.
- It returns ***string*** with all letters in lower case.

Example:

```
SQL> SELECT LOWER('BhagYa AarUSh')
  2  FROM dual;

LOWER('BHAGYA
-------------
bhagya aarush
```

❖ UPPER

Syntax:

UPPER (string)

- This function is a scalar function or single row function.
- It is a kind of character function.
- It returns ***string*** with all letters in upper case.

Example:

```
SQL> SELECT UPPER('BhagYa AarUSh')
  2  FROM dual;

UPPER('BHAGYA
-------------
BHAGYA AARUSH
```

2.1.2 Scalar Functions (Character functions)

❖ INITCAP ( ) ~ Initial Capital

Syntax

INTICAP (string)

- This function returns string with first letter of each word in capital case.

Example

```
SQL> SELECT INITCAP('uresh n parmar') "INITCAP"
  2  FROM dual;

INITCAP
--------------
Uresh N Parmar
```

2.1.2 Scalar Functions (Character functions)

❖ SUBSTR ( )

Syntax

SUBSTR (string, position, length)
OR
SUBSTR (string, SP, L)

- This function returns substring of input ***string***, beginning at ***position*** and going up to ***length*** number of characters.

Example

```
SQL> SELECT SUBSTR('Uresh Parmar',3,3)
  2  FROM dual;

SUB
---
esh
```

2.1.2 Scalar Functions (Character functions)

❖ LPAD ( )

Syntax

LPAD (string, n, string2)

- This function returns ***string***, left padded with ***string2*** up to length ***n***.
- Default string2 is blank.

Example

```
SQL> SELECT LPAD('aarush',10,'*') "LPAD"
  2  FROM dual;

LPAD
----------
****aarush
```

2.1.2 Scalar Functions (Character functions)

❖ RPAD ( )

Syntax

RPAD (string, n, string2)

- This function returns ***string***, right padded with ***string2*** up to length ***n***.
- Default string2 is blank.

Example

```
SQL> SELECT RPAD('aarush',10,'*') "RPAD"
  2  FROM dual;

RPAD
----------
aarush****
```

2.1.2 Scalar Functions (Character functions)

❖ TRIM

- The TRIM function removes the spaces OR other specified characters from the start or end of a string.
- By default, the TRIM function removes leading and trailing spaces from a string.

Example

```
SQL> SELECT TRIM('     aarush     ') NAME FROM dual;

NAME
------
aarush

SQL> SELECT TRIM('aarush     ') NAME FROM dual;

NAME
------
aarush

SQL> SELECT TRIM('aar  ush     ') NAME FROM dual;

NAME
--------
aar  ush
```

2.1.2 Scalar Functions (Character functions)

❖ LTRIM ( )

Syntax

LTRIM (string, set)

- This function removes (trims) the characters from the left of string.
- Characters will be removed from left, up to the first character not in set. (or up to the first unmatched character)

Example

```
SQL> SELECT LTRIM('123BHAGYA',123)
  2  FROM dual;

LTRIM(
------
BHAGYA

SQL> SELECT LTRIM('xyzxyzBHAGYA','xyz')
  2  FROM dual;

LTRIM(
------
BHAGYA
```

```
SQL> SELECT LTRIM('786BHAGYA',0123456789)
  2  FROM dual;

LTRIM(
------
BHAGYA

SQL> SELECT LTRIM('BHAGYA','HABXYA')
  2  FROM dual;

LTR
---
GYA

SQL> SELECT LTRIM('BHAGYA','AB')
  2  FROM dual;

LTRIM
-----
HAGYA
```

2.1.2 Scalar Functions (Character functions)

❖ RTRIM ( )

Syntax

RTRIM (string, set)

- This function removes (trims) the characters from the right of string.
- Characters will be removed from right, up to the first character not in set. (or up to the first unmatched character)

Example

```
SQL> SELECT RTRIM('BHAGYA','AYPB')
  2  FROM dual;

RTRI
----
BHAG

SQL> SELECT RTRIM('BHAGYA','1A2G')
  2  FROM dual;

RTRIM
-----
BHAGY

SQL> SELECT RTRIM('BHAGYA','YGA3')
  2  FROM dual;

RT
--
BH
```

2.1.2 Scalar Functions (Character functions)

❖ REPLACE ( ) *Not in GTU syllabus*

Syntax

REPLACE (string, from_string, to_string)

- This function is similar to translate function, but it works on strings rather than set of characters.
- It replaces entire substring instead of individual characters as in translate function.

Example

```
SQL> SELECT REPLACE ('abc12ef3', '12', 'XY') FROM dual;

REPLACE(
--------
abcXYef3

SQL> SELECT REPLACE ('abc786', '786', 'XYZ') FROM dual;

REPLAC
------
abcXYZ

SQL> SELECT REPLACE ('abc78def786', '786', 'XYZ') FROM dual;

REPLACE('AB
-----------
abc78defXYZ
```

2.1.2 Scalar Functions (Character functions)

❖ INSTR ( )

- It is also called a character index function which is used to find the position of the specific substring from input string. It detects the first occurrence of a substring or a character from the given input string.
- 0 will be returned if the substring is not found in input string.

Syntax

INSTR (string1, string2)

*(In above syntax, **string1** is the input string and **string2** is the substring whose position we want to find in the string1. After executing the function, we get the numerical value in output which shows the position of the first occurrence of the **string2** in **string1**.)*

2.1.2 Scalar Functions (Character functions)

❖ ASCII ( ) *Not in GTU syllabus*

Syntax

ASCII (char)

- This function returns the ASCII code of a character.

Example

```
SQL> SELECT ASCII('a'), ASCII('A') FROM dual;

ASCII('A') ASCII('A')
---------- ----------
        97         65
```

2.1.2 Scalar Functions (Character functions)

Example

```
SQL> SELECT INSTR ('aarush','r') FROM dual;

INSTR('AARUSH','R')
-------------------
                  3

SQL> SELECT INSTR ('aarush parikh','r') FROM dual;

INSTR('AARUSHPARIKH','R')
-------------------------
                        3

SQL> SELECT INSTR ('aarush parikh','parikh') FROM dual;

INSTR('AARUSHPARIKH','PARIKH')
------------------------------
                             8

SQL> SELECT INSTR ('aarush parikh','X') FROM dual;

INSTR('AARUSHPARIKH','X')
-------------------------
                        0
```

The position of first occurrence of 'r' is shown in the result.

0 is returned if substring 'X' is not found the string 'aarush parikh'.

2.1.2 Scalar Functions (Conversion functions)

- These functions convert a value from one data type to another.
- Here we will discuss four important categories of conversion functions.

1. ***Converting Character to Number (TO_NUMBER)***
2. ***Converting Number to Character (TO_CHAR)***
3. ***Converting Date to Character (TO_CHAR)***
4. ***Converting Character to Date (TO_DATE)***

Important note: *if any conversion function asked in the examination, first two lines you have to write like:*

- This function is a scalar function or single row function.

- It is a kind of conversion function.

And accordingly you can write in any type of scalar function.

2.1.2 Scalar Functions (Conversion functions)

❖ TO_NUMBER ( ) : Converting Character to Number

Syntax

TO_NUMBER (string)

- This function converts char or varchar2 data types to number.
- In short, it returns number value equivalent to input string.
- String must consists of 0-9, decimal point and '+' or '-' sign.

Example

```
SQL> SELECT TO_NUMBER('123'), TO_NUMBER('123.45') FROM dual;

TO_NUMBER('123') TO_NUMBER('123.45')
---------------- --------------------
             123                123.45
```

2.1.2 Scalar Functions (Conversion functions)

❖ TO_CHAR ( ) : Converting Number to Character

Syntax

TO_NUMBER (string)

- This function converts a numerical value n to a numeric character data type using operational format.
- A format should be valid numeric format consisting of zero '0', nine '9' and comma (,).

Example

```
SQL> SELECT TO_CHAR(123456, '09,99,999') FROM dual;

TO_CHAR(12
----------
 01,23,456

SQL> SELECT TO_CHAR(12345, '99,999') FROM dual;

TO_CHAR
-------
 12,345
```

2.1.2 Scalar Functions (Conversion functions)

❖ TO_CHAR ( ) : Converting Date to Character

Syntax

TO_CHAR (date, format)

- This function converts date value to character value using format.
- A format must be a valid date format.
- If format is omitted, the default format is 'DD-MON-YY' is used.

Example

```
SQL> SELECT TO_CHAR(sysdate, 'dd-mm-yy') FROM dual;

TO_CHAR(
--------
17-09-23

SQL> SELECT TO_CHAR(sysdate, 'dd-mon-yyyy') FROM dual;

TO_CHAR(SYSDATE,'DD-
--------------------
17-sep-2023
```

```
SQL> SELECT TO_CHAR(sysdate, 'day-mon-yyyy') FROM dual;

TO_CHAR(SYSDATE,'DAY-MON-YYYY')
-------------------------------------------------------
sunday    -sep-2023

SQL> SELECT TO_CHAR(sysdate, 'day-month-year') FROM dual;

TO_CHAR(SYSDATE,'DAY-MONTH-YEAR')
-------------------------------------------------------------
sunday    -september-twenty twenty-three
```

2.1.2 Scalar Functions (Conversion functions)

✍ (આ function, date ને string ના particular format માં convert કરે છે.)

- (Some useful date functions that can be used in these functions)

Format	Specification	Format	Specification
MM	Number of month (1-12)	YYY	Last 3 digits of year
MON	3 – letter month (AUG)	YYYY	Last 4 digits of year
MONTH	FULL month name (AUGUST)	YEAR	Spelling of year
D	No. of day of week	WW	No. of week in a year
DD	No. of day of month	W	No. of week in a month
DAY	Name of the day	HH	Hour
DDD	No. of day of year	MI	Minutes
Y	Last I digit of year	SS	Second
YY	Last 2 digits of year		

2.1.2 Scalar Functions (Conversion functions)

❖ TO_DATE ( ) : Converting Character to Date

Syntax

TO_DATE (string, format)

- This function converts character value i.e. string to a DATE value.
- A format specifies the date format in which string contains value.
- The resultant date will be in the default date format → 'DD-MON-YY'.

Example

```
SQL> SELECT TO_DATE('13 november 2019','DD MON YY') from dual;

TO_DATE('
---------
13-NOV-19

SQL> SELECT TO_DATE('13 november 2019','DD MONTH YY') from dual;

TO_DATE('
---------
13-NOV-19

SQL> SELECT TO_DATE('13 november 19','YY MONTH DD') from dual;

TO_DATE('
---------
19-NOV-13
```

2.1.2 Scalar Functions (Conversion functions)

```
SQL> SELECT TO_DATE('070903', 'MMDDYY') FROM dual;

TO_DATE('
---------
09-JUL-03

SQL> SELECT TO_DATE('070903', 'DDMMYY') FROM dual;

TO_DATE('
---------
07-SEP-03

SQL> SELECT TO_DATE('070903', 'YYDDMM') FROM dual;

TO_DATE('
---------
09-MAR-07

SQL> select TO_DATE('20070903', 'ddYYYYmm') from dual;

TO_DATE('
---------
20-MAR-09
```

Here, 07 considered as Month (MM), 09 as Day (DD) and 03 as Year (YY).

(નોંધ: આ function માં output તો fixed format 'DD-MON-YY' માં જ આવે છે, પરંતુ તમે string માં જે value આપો છો તેને date format સાથે match કરી output આવે છે.)

2.1.2 MISCELLANEOUS FUNCTIONS (*not in GTU syllabus*)

❖ GREATEST

Syntax

GREATEST (value1, value2,...., valueN)

- This function returns the greatest value from a list of arguments.
- It can be used with numerical, character as well as with date data.

Example

```
SQL> SELECT GREATEST (13, 5, 15, 1) FROM DUAL;

GREATEST(13,5,15,1)
-------------------
                 15
```

2.1.2 MISCELLANEOUS FUNCTIONS (*not in GTU syllabus*)

❖ LEAST

Syntax

LEAST (value1, value2,...., valueN)

- This function returns the lowest value from a list of arguments.
- It can be used with numerical, character as well as with date data.

Example

```
SQL> SELECT LEAST (13, 5, 15, 1) FROM DUAL;

LEAST(13,5,15,1)
----------------
               1
```

2.1.2 MISCELLANEOUS FUNCTIONS (*not in GTU syllabus*)

❖ VSIZE

Syntax

VSIZE (expression)

- This function returns the storage size of expression in Oracle.
- It returns the number of bytes of expression.
- If expression is NULL, it returns NULL.

Example

```
SQL> SELECT VSIZE ('bhagya'), VSIZE(25), VSIZE (sysdate)
  2  FROM DUAL;

VSIZE('BHAGYA')  VSIZE(25) VSIZE(SYSDATE)
---------------- ---------- -------------
              6          2             7
```

```
SQL> SELECT VSIZE (''), VSIZE(' ') FROM DUAL;

 VSIZE('')  VSIZE('')
---------- ----------
                    1
```

CASE Statement

- Used to perform conditional logic within SQL queries.
- It operates like if-then-else type of logical queries.
- It can be used with SELECT, INSERT or DELETE.
- This statement in SQL is always followed by at least one pair of WHEN and THEN statements and always finished with the END keyword.

Syntax

Write explanation of the syntax

```
CASE <expression>
WHEN condition1 THEN result1
WHEN condition2 THEN result2…
WHEN consitionN THEN resultN
ELSE result
END;
```

CASE Statement

Example (*Suppose we have a table 'employee' like:***)**

E_ID	E_NAME	E_SALARY
1	dinesh	25000
2	ashok	35000
3	gita	12500
4	naresh	15000
5	kamlesh	5000
6	haresh	8000
7	babulal	55000
8	akash	33500
9	jitu	5500
10	piyush	8000

Now grade the salary to each employee as if their salary is less than 10000 then grade them as 'Low Salary', if they have salary greater than or equal to 10000 and less than 25000 then grade them as 'Medium Salary' and if they are earning greater than or equals to 25000 salary then grade them as 'High Salary'.

CASE Statement

```
select e_id,e_name,
case
when e_salary <10000 then 'Low Salary'
when e_salary <=25000 then 'Medium Salary'
when e_salary >=25000 then 'High Salary'
end
FROM employee;
```

E_ID	E_NAME	CASEWHENE_SAL
1	dinesh	Medium Salary
2	ashok	High Salary
3	gita	Medium Salary
4	naresh	Medium Salary
5	kamlesh	Low Salary
6	haresh	Low Salary
7	babulal	High Salary
8	akash	High Salary
9	jitu	Low Salary
10	piyush	Low Salary

2.2 Group By, Having and Order by clause

❖ ORDER BY

(Sorting data of a table in ascending or descending order)

- SELECT does not return result in a particular order. ORDER BY clause does it.
- We can sort the data in either ascending or descending order. By default, ORDER BY sorts the data in ascending order. Use ***DESC*** keyword to get the data in descending order.

Syntax

```
SELECT columnNames
FROM tabel_name
ORDER BY columnName [DESC];
```

2.2 Group By, Having and Order by clause

Example (*Display name and balance of all persons according to their balance in ascending order*)

```
SQL> SELECT name, balance FROM person ORDER BY balance;

NAME           BALANCE
---------- ----------
avani             200
samay            1000
aarav            1200
shreeja          5000
parth            5000
vihan           10000
bhagya          15000
tushar          19000
aarush          22000
misha           55000
```

Example (*Display name and balance of all persons according to their balance in descending order*)

```
SQL> SELECT name, balance FROM person ORDER BY balance DESC;
```

2.2 Group By, Having and Order by clause

❖ GROUP BY

(Grouping data of a table)

- Used to group rows that have the same values.
- It is used with SELECT statement. GROUP BY is placed after WHERE condition and if we use ORDER BY clause in our statement, GROUP BY is placed before ORDER BY.
- GROUP BY returns one records for each group. GROUP BY typically also involves aggregate functions: COUNT, MAX, SUM, AVG, etc.

Syntax

```
SELECT ColumnNames, Aggregate Function (argument)
FROM table_name WHERE condition
GROUP BY ColumnNames;
```

OR

```
SELECT column-names
  FROM table-name
 WHERE condition
 GROUP BY column-names
```

2.2 Group By, Having and Order by clause

Example (*Display the balance of all grouped surnames***)**

```
SQL> SELECT surname, sum(balance) "Total Balance"
  2  FROM person
  3  GROUP BY surname;

SURNAME    Total Balance
---------- -------------
sutariya             55000
parikh               47000
parmar               19000
saxena                1000
solanki               1400
sagar                10000

6 rows selected.
```

2.2 Group By, Having and Order by clause

Example2 (*Display the count of all surnames based on their group***)**

```
SQL> SELECT surname,COUNT(surname)
  2  FROM person
  3  GROUP BY surname;

SURNAME     COUNT(SURNAME)
---------- --------------
sutariya                1
parikh                  4
parmar                  1
saxena                  1
solanki                 2
sagar                   1

6 rows selected.
```

2.2 Group By, Having and Order by clause

❖ HAVING

- Place the condition, defined by GROUP BY. It simply filtering the records summarized by GROUP BY.
- HAVING requires that a GROUP BY clause is present.
- Only the groups that meet the HAVING criteria will be returned in the output.

Syntax

```
SELECT ColumnNames,
FROM table_name
WHERE condition
GROUP BY ColumNames
Having condition;
```

2.2 Group By, Having and Order by clause

Example (*Display how many persons having surname as 'parmar'***)**

```
SQL> SELECT surname,COUNT(surname)
  2  FROM person
  3  GROUP BY surname;

SURNAME     COUNT(SURNAME)
---------- --------------
sutariya                1
parikh                  4
parmar                  1
saxena                  1
solanki                 2
sagar                   1

6 rows selected.

SQL> SELECT surname,COUNT(surname)
  2  FROM person
  3  GROUP BY surname
  4  HAVING surname = 'parmar';

SURNAME     COUNT(SURNAME)
---------- --------------
parmar                  1
```

2.3 JOINS

- Used to combine records from two or more tables in a database.
- Multiple tables can be joined using common columns.
- The output of a join is a new wider and bigger table in which each row is a concatenation of two rows, one from each column.
- Various joins can be classified as:
  - ✓ ***CROSS JOIN (Cartesian product or Simple join)***
  - ✓ ***INNER JOIN (Equi join and non-Equi join)***
  - ✓ ***SELF JOIN***
  - ✓ ***OUTER JOIN (Left outer, Right outer, Full outer joins)***
- Consider these two tables.

Table name: *test1*

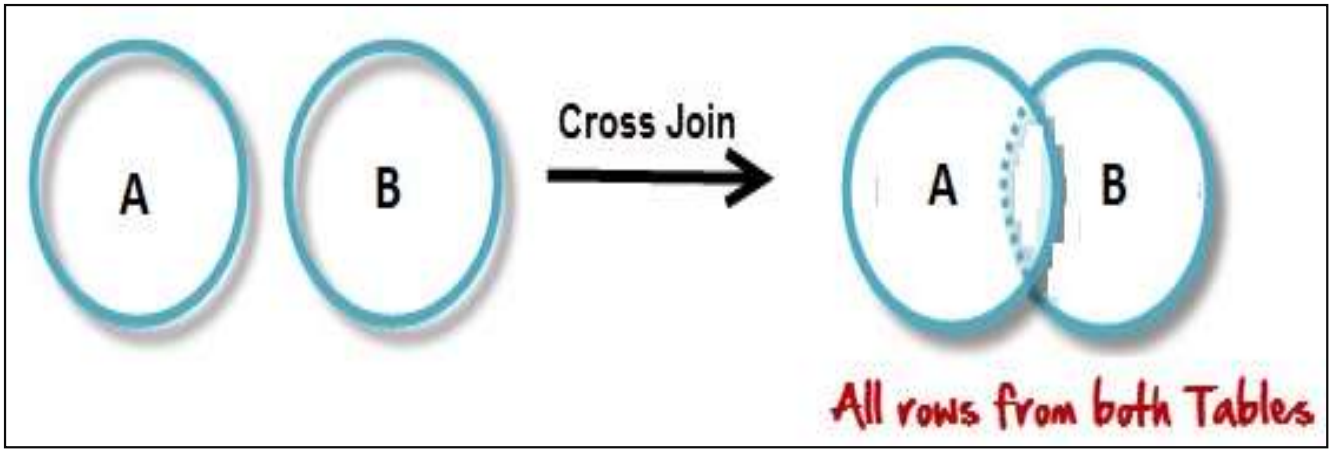
ID	NAME
1	bhagya
2	aarush
3	shreeja

Table name: *test2*

ID	CITY
2	surendranagar
3	rajkot
4	ahmedabad

2.3 JOINS (CROSS JOIN)

❖ CROSS JOIN (Simple Join OR Cartesian Product)



- It is simplest join. Also referred as Cartesian product.
- It combines every row from one table with every row of another table.

Syntax

```
SELECT column1, column2, .., column
FROM table1, table2;
```

2.3 JOINS (CROSS JOIN)

Example (*SIMPLE JOIN on table 'test1' and 'test2'***)**

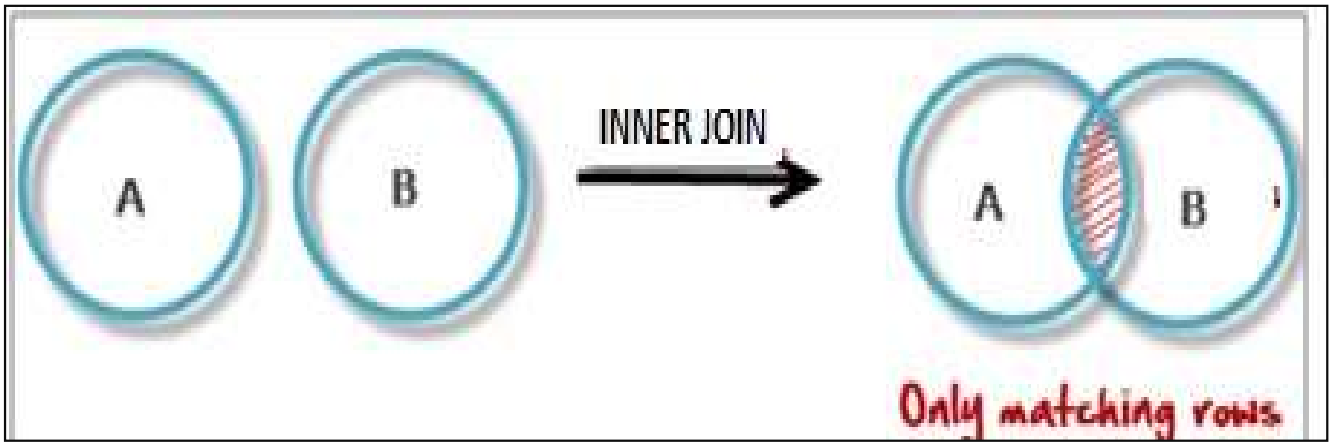
*SELECT * FROM test1, test2;*

```
SQL> SELECT * FROM test1, test2;

        ID NAME                         ID CITY
---------- -------------- ---------- --------------
         1 bhagya                     2 surendranagar
         1 bhagya                     3 rajkot
         1 bhagya                     4 ahmedabad
         2 aarush                     2 surendranagar
         2 aarush                     3 rajkot
         2 aarush                     4 ahmedabad
         3 shreeja                    2 surendranagar
         3 shreeja                    3 rajkot
         3 shreeja                    4 ahmedabad

9 rows selected.
```

Only consistent and true information

2.3 JOINS (INNER JOIN)

❖ INNER JOIN



- This operation provides filtering on the output of cross join operation.
- It provides consistent information.
- The INNER JOIN combines only those records which are common in both tables.
- It returns records that have matching values in both tables.

2.3 JOINS (INNER JOIN)

Syntax

Syntax of Inner join:

```
SELECT column_Names
FROM table1, table2
WHERE table1.ColumnName OP table2.ColumnName;
```

Here OP can be any relational operator which is used to compare two values of column1 and column2.

OR

Any of this syntax can be written in the exam.

Syntax of Inner join:

```
SELECT column_Names
FROM table1 INNER JOIN table2
ON table1.column = table2.column;
```

Concept of 'Equi-join' and 'Non-equi join'

2.3 JOINS (INNER JOIN)

Example *(Apply inner join on table test1 and test2 tables)*

```
SQL> SELECT test1.id, name, city
  2  FROM test1, test2
  3  WHERE test1.id = test2.id;

        ID NAME       CITY
---------- ---------- --------------
         2 aarush     surendranagar
         3 shreeja    rajkot
```

OR

```
SQL> SELECT test1.id, name, city
  2  FROM test1 INNER JOIN test2
  3  ON test1.id = test2.id;

ID       NAME       CITY
-------  ---------- --------------
2        aarush     surendranagar
3        shreeja    rajkot
```

2.3 JOINS (SELF JOIN)

❖ SELF JOIN

- The process (OR operation) of joining a table with itself is called '***Self join***'.
- In it, each row of a table is combined with other rows of the same table to produce result.
- At that time, two different copies of the same table are opened in the memory.
- To distinguish two different instances of the same table, we have to create alias of the table.

Syntax

```
SELECT column1, column2, ..., columnN
FROM table1 alias1, table1 alias2
WHERE condition;
```

2.3 JOINS (SELF JOIN)

Example *(Consider the below table 'test_self' and display the name of persons along with their sister's name)*

NO	NAME	BR_NO	SIS_NO
1	bhagya		4
2	aarav	1	4
3	aarush	1	4
4	avani	1	

(*In this table, the column br_no tells that the particular number person is the brother of that record and sis_no tell sister's serial number. So that we can identify that br_no = 1 means 'bhagya' is the brother of other three persons and sis_no = 4 means 'avani' is the sister of first three persons.*)

```
SQL> SELECT a.name, b.name "SisterName"
  2  FROM test_self a, test_self b
  3  WHERE a.sis_no = b.no;

NAME       SisterName
---------- ----------
aarush     avani
aarav      avani
bhagya     avani
```

Here, we have created alias for the table.

2.3 JOINS (OUTER JOIN)

❖ OUTER JOIN

- It is extension of inner join operation.
- Prevent the loss of data occurred in inner join.

An example that shows the problem occurred in inner join:

Suppose that we are having two tables → 'test1' and 'test2' like below:

Table: *test1*

ID	NAME
1	bhagya
2	aarush
3	shreeja

Table: *test2*

ID	CITY
2	surendranagar
3	rajkot
4	ahmedabad

Now consider the inner join operation on above relations.

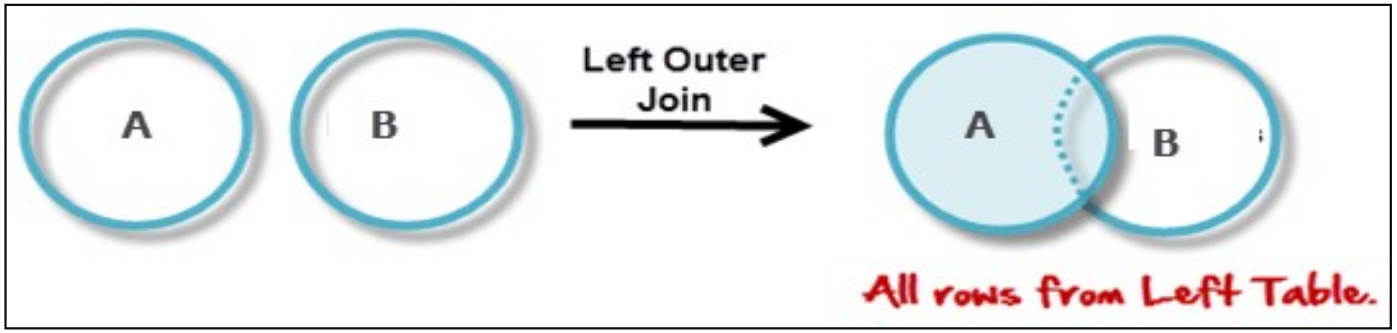
```
SELECT test1.id, test1.name, test2.city
from test1, test2
WHERE test1.id = test2.id;
```

ID	NAME	CITY
2	aarush	surendranagar
3	shreeja	rajkot

2.3 JOINS (OUTER JOIN)

- There are three kinds of outer join.
  1. ***Left outer join***
  2. ***Right outer join***
  3. ***Full outer join***

1. Left outer join



- Also known as left join.
- Simply we can say → "This join returns all the rows of the table on the left side of the join and matching rows for the table on the right side of join". Non matching rows will contain NULL values.
- It preserves the records of the 'left table'.

2.3 JOINS (OUTER JOIN)

Syntax

Syntax of Left outer join (Left join):

```
SELECT column_names
FROM table1, table2
WHERE table1.column = table2.column(+);
```

OR

Syntax of Left outer join (Left join):

```
SELECT column_names
FROM table1 LEFT JOIN table2
ON table1.column = table2.column;
```

અહી આપેલ બંને syntax માંથી કોઈ પણ એક exam માં લખી શકાય

2.3 JOINS (OUTER JOIN)

Example (*Perform the left outer join operation on 'test1' and 'test2' tables***)**

Table: *test1*

ID	NAME
1	bhagya
2	aarush
3	shreeja

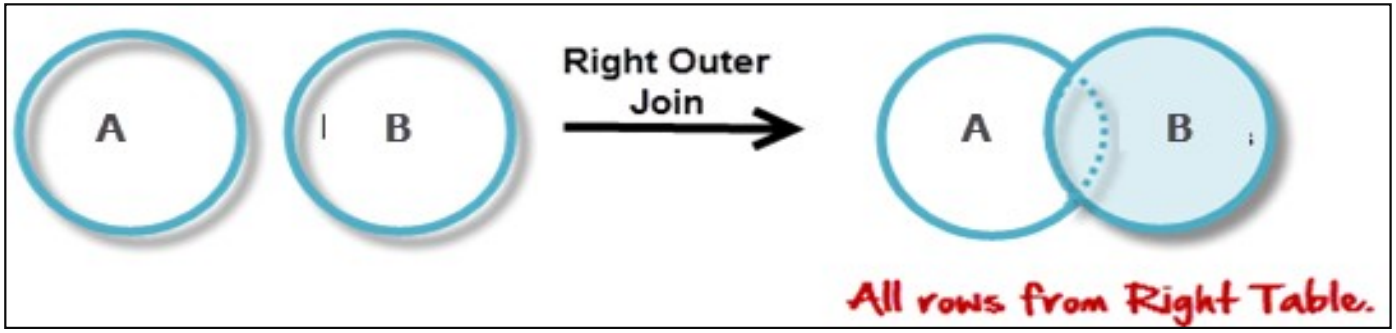
Table: *test2*

ID	CITY
2	surendranagar
3	rajkot
4	ahmedabad

```
SQL> SELECT * FROM test1, test2
  2  WHERE test1.id = test2.id(+);

        ID NAME                 ID CITY
---------- ---------- ---------- ---------------
         2 aarush                2 surendranagar
         3 shreeja               3 rajkot
         1 bhagya

SQL> SELECT * FROM test1 LEFT JOIN test2
  2  ON test1.id = test2.id;

        ID NAME                 ID CITY
---------- ---------- ---------- ---------------
         2 aarush                2 surendranagar
         3 shreeja               3 rajkot
         1 bhagya
```

2.3 JOINS (OUTER JOIN)

2. Right outer join



- Also known as right join.
- Simply we can say → "This join returns all the rows of the table on the right side of the join and matching rows for the table on the left side of join". Non matching rows will contain NULL values.
- It preserves the records of the 'right table'.

2.3 JOINS (OUTER JOIN)

Syntax

Syntax of Right outer join (Right join):

```
SELECT column_names
FROM table1, table2
WHERE table1.column(+) = table2.column;
```

OR

Syntax of Right outer join (Right join):

```
SELECT column_names
FROM table1 RIGHT JOIN table2
ON table1.column = table2.column;
```

અહી આપેલ બંને syntax માંથી કોઈ પણ એક exam માં લખી શકાય છે.

2.3 JOINS (OUTER JOIN)

Example (*Perform the right outer join operation on 'test1' and 'test2' tables***)**

Table: *test1*

ID	NAME
1	bhagya
2	aarush
3	shreeja

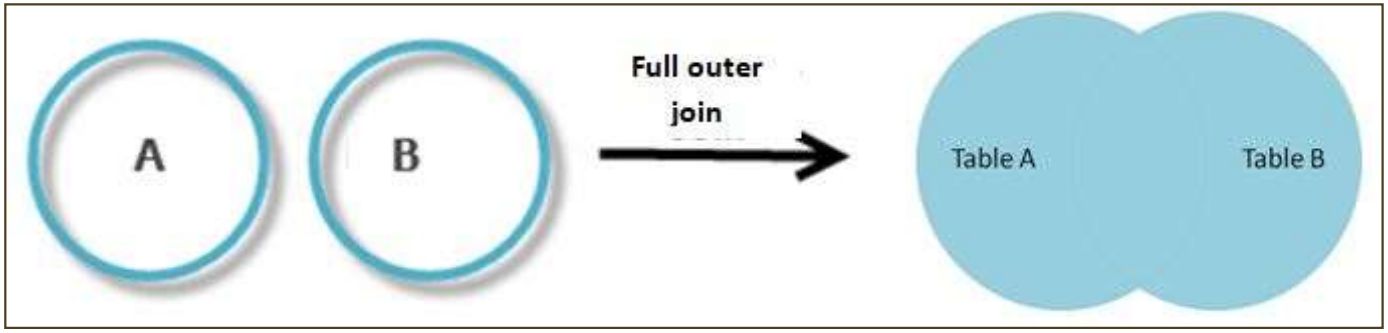
Table: *test2*

ID	CITY
2	surendranagar
3	rajkot
4	ahmedabad

```
SQL> SELECT * FROM test1, test2
  2  WHERE test1.id(+) = test2.id;

        ID NAME                 ID CITY
---------- ---------- ---------- ---------------
         2 aarush                2 surendranagar
         3 shreeja               3 rajkot
                                 4 ahmedabad

SQL> SELECT * FROM test1 RIGHT JOIN test2
  2  ON test1.id = test2.id;

        ID NAME                 ID CITY
---------- ---------- ---------- ---------------
         2 aarush                2 surendranagar
         3 shreeja               3 rajkot
                                 4 ahmedabad
```

2.3 JOINS (OUTER JOIN)

2. Full outer join



- Also known as full join.
- Simply we can say → " This join returns all the rows from both the tables which left and right of the join give". Non matching rows will contain NULL values.
- It preserves the records of the both right and left tables.

2.3 JOINS (OUTER JOIN)

Syntax

Syntax of Full outer join (Full join):

```
SELECT column_names
FROM table1, table2
WHERE table1.column = table2.column(+)
UNION
SELECT column_names
FROM table1, table2
WHERE table1.column(+) = table2.column;
```

Full outer join can be performed using UNION on left outer join and right outer join.

OR

Syntax of Full outer join (Full join):

```
SELECT column_names
FROM table1 FULL JOIN table2
ON table1.column = table2.column;
```

2.3 JOINS (OUTER JOIN)

Example (*Perform the full outer join operation on 'test1' and 'test2' tables***)**

Table: ***test1***

ID	NAME
1	bhagya
2	aarush
3	shreeja

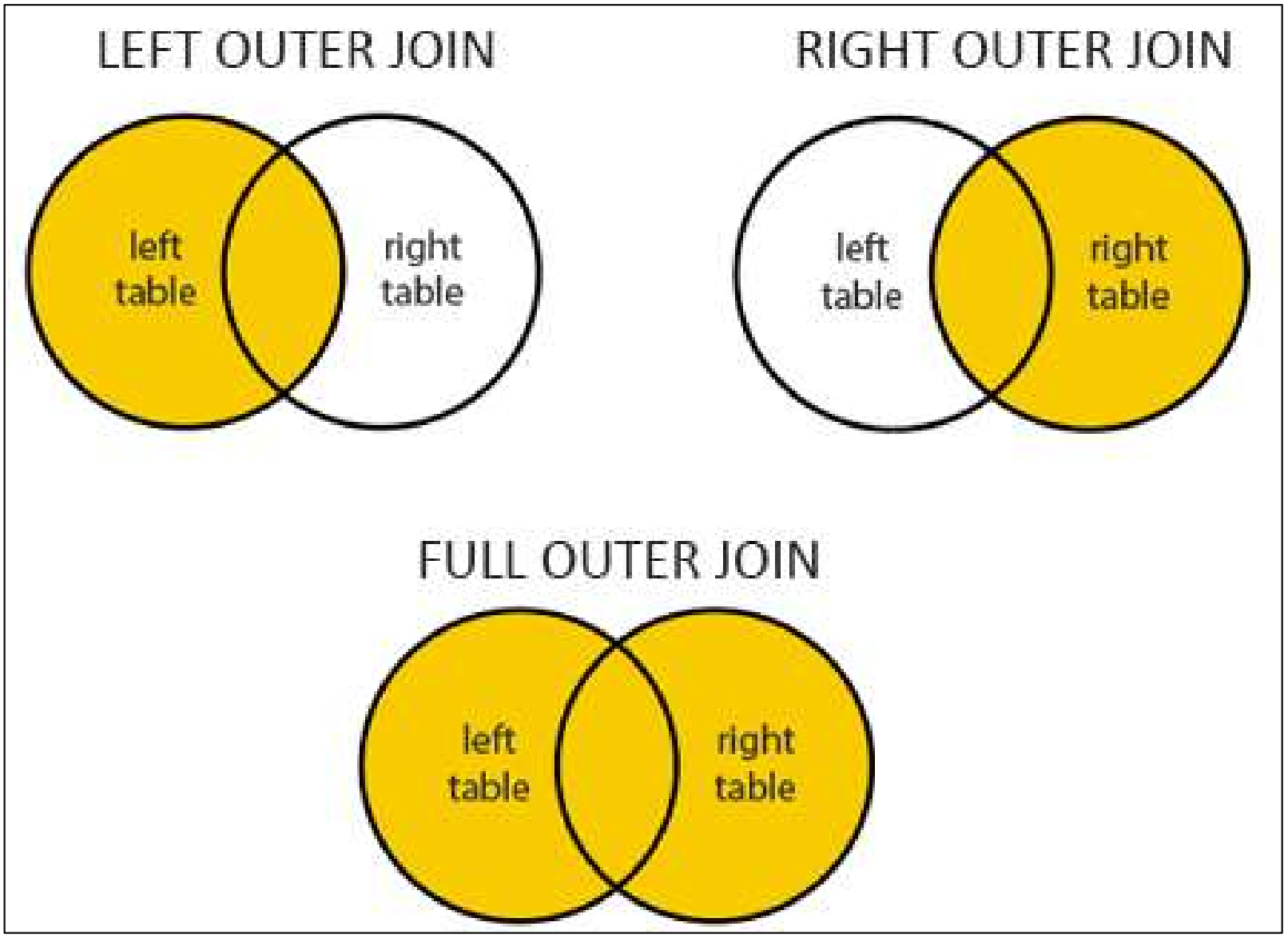
Table: ***test2***

ID	CITY
2	surendranagar
3	rajkot
4	ahmedabad

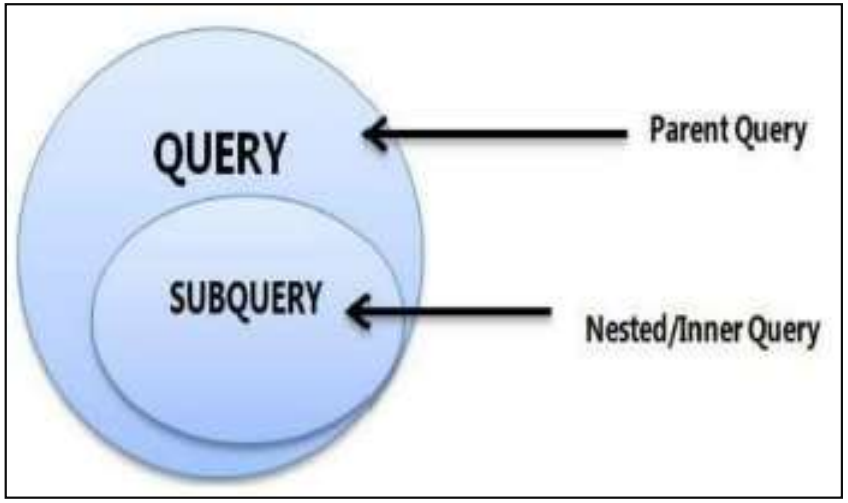
```
SQL> SELECT * FROM test1,test2
  2  WHERE test1.id = test2.id(+)
  3  UNION
  4  SELECT * FROM test1,test2
  5  WHERE test1.id(+) = test2.id;

        ID NAME                 ID CITY
---------- ---------- ---------- ---------------
         1 bhagya
         2 aarush                2 surendranagar
         3 shreeja               3 rajkot
                                 4 ahmedabad

SQL> SELECT * FROM test1 FULL JOIN test2
  2  ON test1.id = test2.id;

        ID NAME                 ID CITY
---------- ---------- ---------- ---------------
         2 aarush                2 surendranagar
         3 shreeja               3 rajkot
         1 bhagya
                                 4 ahmedabad
```

2.3 JOINS (OUTER JOIN)

Summary of Outer join operations



2.4 SUB-QUERIES

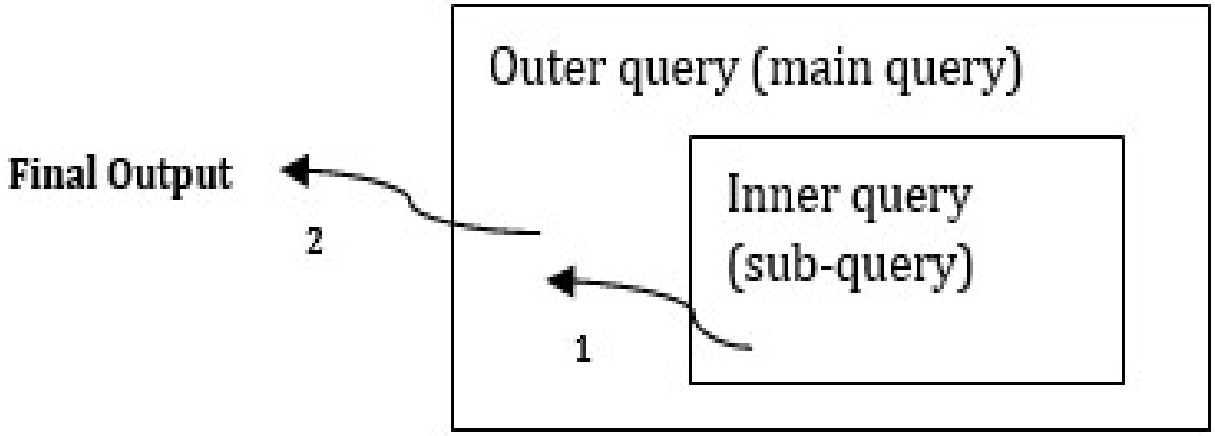


- A sub-query is also called 'inner-query' or 'nested-query'.
- Simply we can say → *'a sub-query' is a query within another query'*.

➥ **Characteristics of sub-query**

- The statement that contains a sub-query is called 'parent query' or 'outer query' or 'main query'.
- There can be a chain of sub-query.
- Sub-query must be enclosed in parentheses.
- ORDER BY command cannot be used within a sub-query.

2.4 SUB-QUERIES



Description of above figure.

Syntax

```
SELECT columnNames
FROM tableName1
WHERE value ExpressionOperator
        (SELECT columnNames FROM tablename2 WHERE condition..);
```

2.4 SUB-QUERIES

Example (we have two tables as following):

Table: *test1*

ID	NAME
1	bhagya
2	aarush
3	shreeja

Table: *test2*

ID	CITY
2	surendranagar
3	rajkot
4	ahmedabad

Find out the city of a person whose name is 'aarush' from above tables.

```
SQL> SELECT city FROM test2 WHERE id IN (SELECT id FROM test1 WHERE name='aarush');

CITY
---------------
surendranagar
```

Sub query

2.4 SUB-QUERIES

Another example (consider the below three tables):

Table name: test11	
id	**name**
1	unp
2	bdl
3	akp
4	srb

(* id = faculty id)

Table name: test22	
name	**subject**
bdl	c++
cck	malp
akp	os
unp	dbms
srb	ds

Table name: test33	
subject	**subid**
ds	3330701
dbms	3330703
os	3330701
c++	3330702
malp	3330705
cns	3350704

(in test11, faculty names and their id are given while in test22 faculty names and subject which they are teaching is given and in test33 subjects along with their subject id given)

Find the name of the faculty who is teaching the subject 3330703.

```
SQL> SELECT name FROM test22
  2  WHERE subject = (SELECT subject FROM test33
  3  WHERE subid = 3330703);

NAME
---------------
unp
```

2.4 SUB-QUERIES

Example *(Chain of sub-query.)*

Say for example, if we want to find out the faculty id of who is teaching the subject 3330701, then we have to concern with all three tables, like:

```
SQL> SELECT id FROM test11
  2  WHERE name = (SELECT name from test22
  3  WHERE subject = (SELECT subject FROM test33
  4  WHERE subid = 3330701));

        ID
----------
         3
```

2.4 SUB-QUERIES

❖ EXIST operator in Sub-query

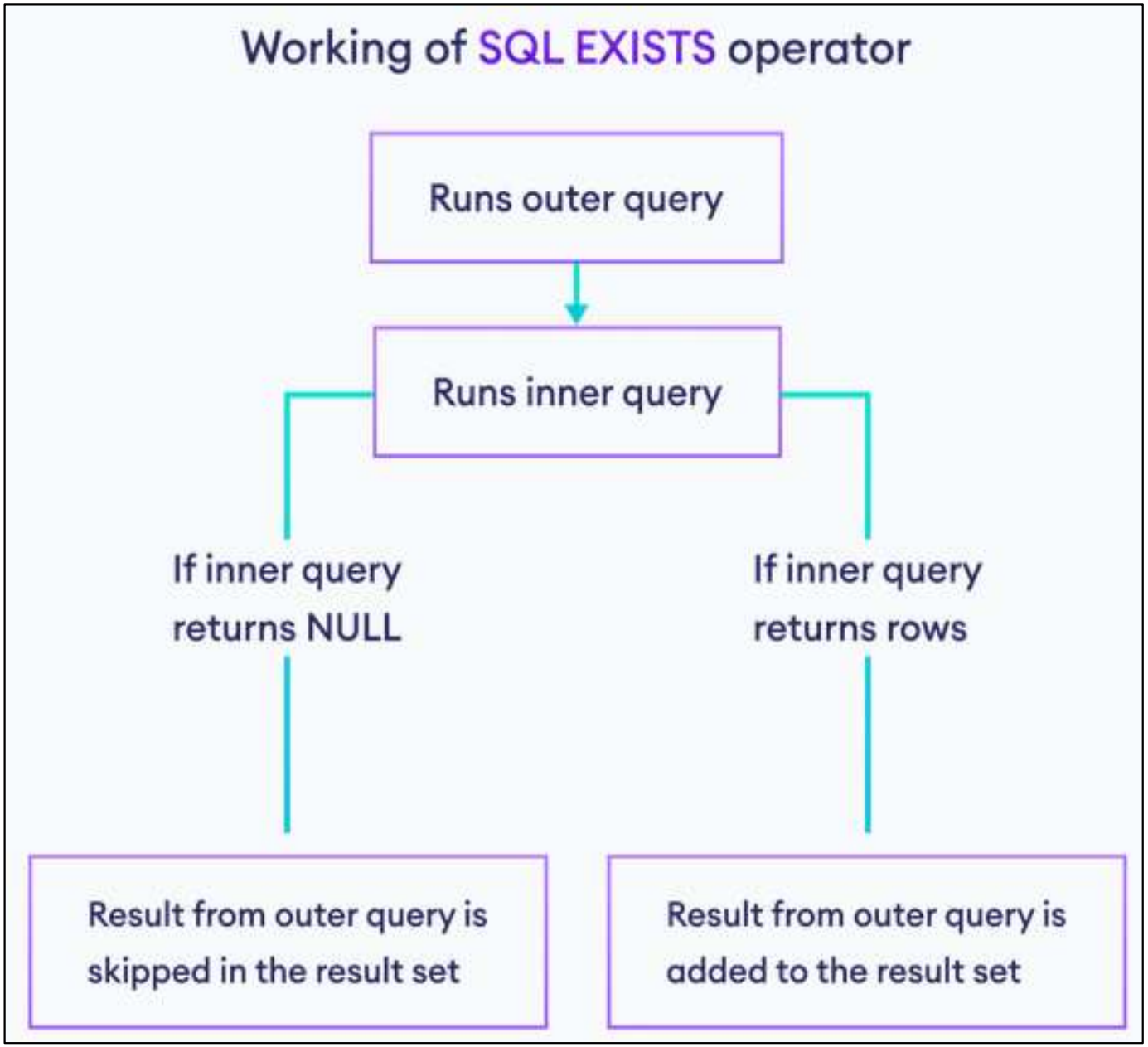
- It is used to check for the existence of rows returned by a subquery.
- It executes the outer query only if the subquery is not null.
- The result of EXIST is a Boolean value.
- It can be used with SELECT, UPDATE, INSERT or DELETE statement.

Syntax

```
SELECT columnNames
FROM tableName
WHERE EXISTS (Sub-Query);
```

2.4 SUB-QUERIES

The working of EXIST operator



2.4 SUB-QUERIES

Example

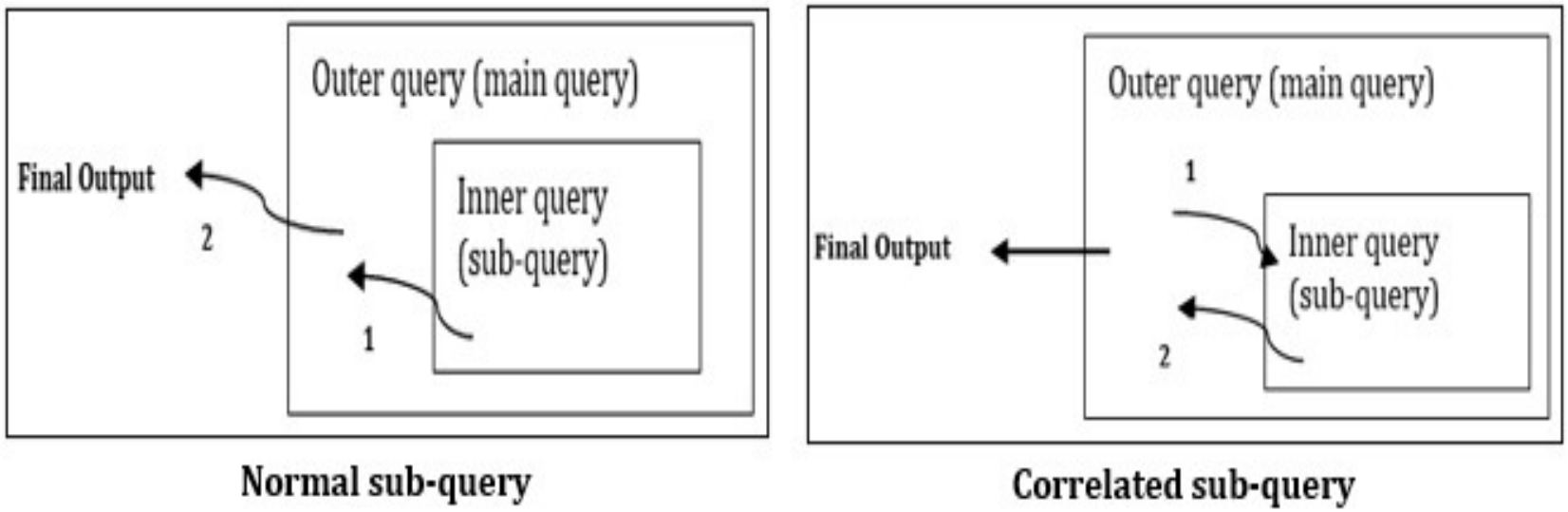
(Consider previous tables (i.e. test11 and test22), and provide the name of subjects for which we are having the faculty id data. (Simply generate the name of only those subjects who are taught by the faculties whose id are available))

```
SQL> SELECT subject FROM test22
  2  WHERE EXISTS (SELECT name FROM test11 WHERE test11.name=test22.name);

SUBJECT
---------------
dbms
c++
os
ds
```

✍ ***(Note: NOT EXISTS working exactly opposite of EXIST. Do practice yourself)***

2.4 SUB-QUERIES

❖ Correlated sub-query



- With a normal nested sub-query, the inner query runs first and executes once and returns result, that result is used by the main query and then main query generates the final output.
- In case of correlated sub-query, the inner query using the value from outer query and executing once for every row of the outer query.
- In correlated sub-query → *'a sub-query which is using values from outer queries'.*

2.4 SUB-QUERIES

- It is also called *synchronized sub-query*.
- It is executed in top-down manner.
- In the working of non-correlated sub-query, it is executed only once. While in case of correlated sub-query, it is executed multiple times, precisely once for each row returned by the outer query.
- The main difference between a correlated sub-query and a simple sub-query is that correlated sub-queries reference columns from the outer table.

2.4 SUB-QUERIES

Example *(Consider the person table in slide 6 and Find out the name, surname and highest balance (surname wise) from 'person' table. Means if more than one person having same surname, then find the maximum balance for that surname.)*

```
SQL> SELECT name, surname, balance
  2  FROM person P
  3  WHERE balance IN (SELECT max(balance) FROM person WHERE surname=P.surname);

NAME       SURNAME       BALANCE
---------- ---------- ----------
vihan      sagar           10000
samay      saxena           1000
tushar     parmar          19000
aarush     parikh          22000
misha      sutariya        55000
aarav      solanki          1300

6 rows selected.
```

Here, we have created the alias P for the table 'person'.

2.4 SUB-QUERIES

Example *(Consider the tables)*

Table: *test1*

ID	NAME
1	bhagya
2	aarush
3	shreeja

Table: *test2*

ID	CITY
2	surendranagar
3	rajkot
4	ahmedabad

Find the names of the persons for whom we are having the data of their cities. (એટલે કે તેવા માણસોના નામ આપો, જેમના city ના data આપણાં પાસે છે. જે માણસનું city નથી તેમના નામ output માં ન આવવા જોઈએ.)

```
SQL> SELECT name FROM test1
  2  WHERE EXISTS (SELECT * FROM test2 WHERE test2.id = test1.id);

NAME
---------------
aarush
shreeja
```

Here inner query is using the data of outer query

2.5 SQL SET Operators

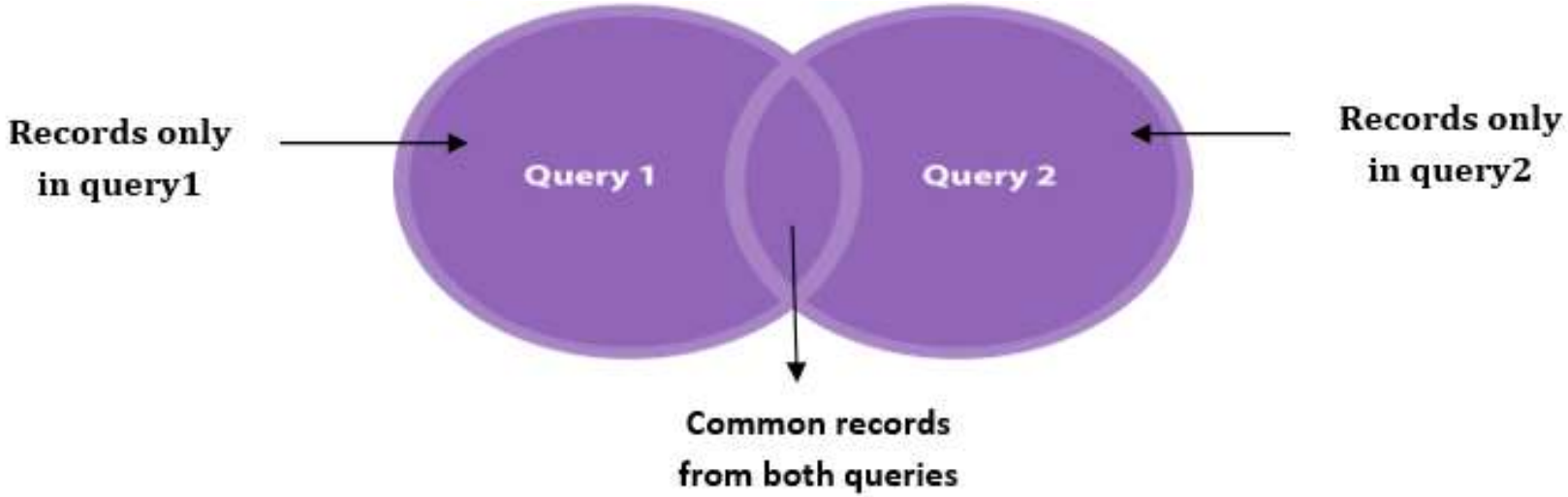
- These operators are performed on table data.
- The SQL Set operation is used to combine two or more SQL SELECT statements. It appears between two independent queries.
- The queries containing set operators are called compound queries.
- Different types of SET operators in SQL, are:

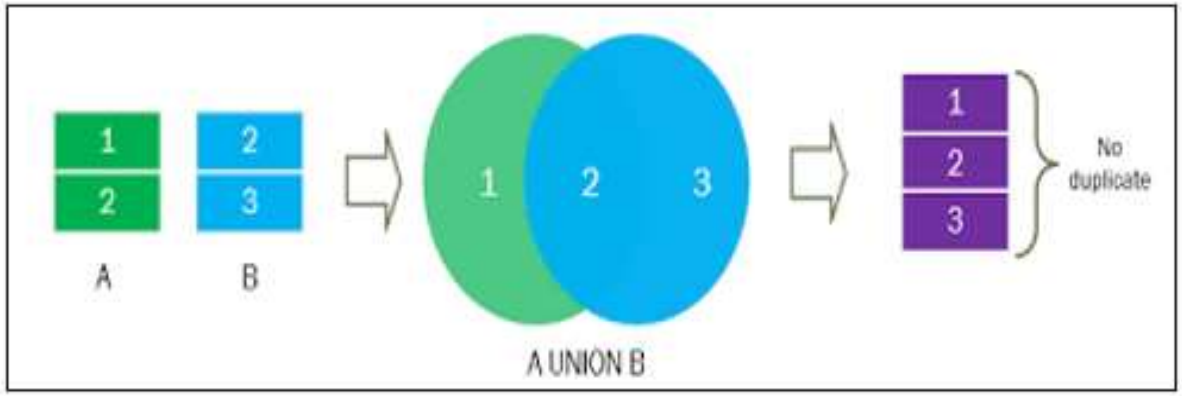
UNION	It combines two results sets, but *removes* any duplicates.
UNION ALL	It combines two results sets, but *preserves* any duplicates.
INTERSECT	It is used to show the *commonalities* between two result sets.
MINUS	It is used to produce the difference between two result sets.

2.5 SQL SET Operators (UNION)

❖ UNION

- .





- It is used to combine the result of two or more SQL SELECT queries.
- In the UNION operation, all the datatype and columns must be same in both the tables on which UNION operation is being applied.

2.5 SQL SET Operators (UNION)

- It eliminates duplicate rows in result.
- NULL values will be ignored.
- Output will be sorted records in ascending order.

Example (*Suppose we have two tables named → 'test1' and 'test2'. Now we are performing UNION operation on them***)**

table: *test1*

ID	NAME
1	bhagya
2	aarush
3	shreeja

Table: *test2*

ID	NAME
3	shreeja
4	avani
5	aarav

test1 UNION test2

```
SQL> SELECT * FROM test1 UNION SELECT * FROM test2;

        ID NAME
---------- ----------
         1 bhagya
         2 aarush
         3 shreeja
         4 avani
         5 aarav
```

Here we have two different SELECT queries, which are combined using UNION set operator.

2.5 SQL SET Operators (UNION ALL)

❖ UNION ALL

- Doing the operation like UNION operation, but it preserves the duplicates.

Example (*If the same queries (like UNION) we apply using UNION ALL, we get the output like***)**

Table: *test1*

ID	NAME
1	bhagya
2	aarush
3	shreeja

Table: *test2*

ID	NAME
3	shreeja
4	avani
5	aarav

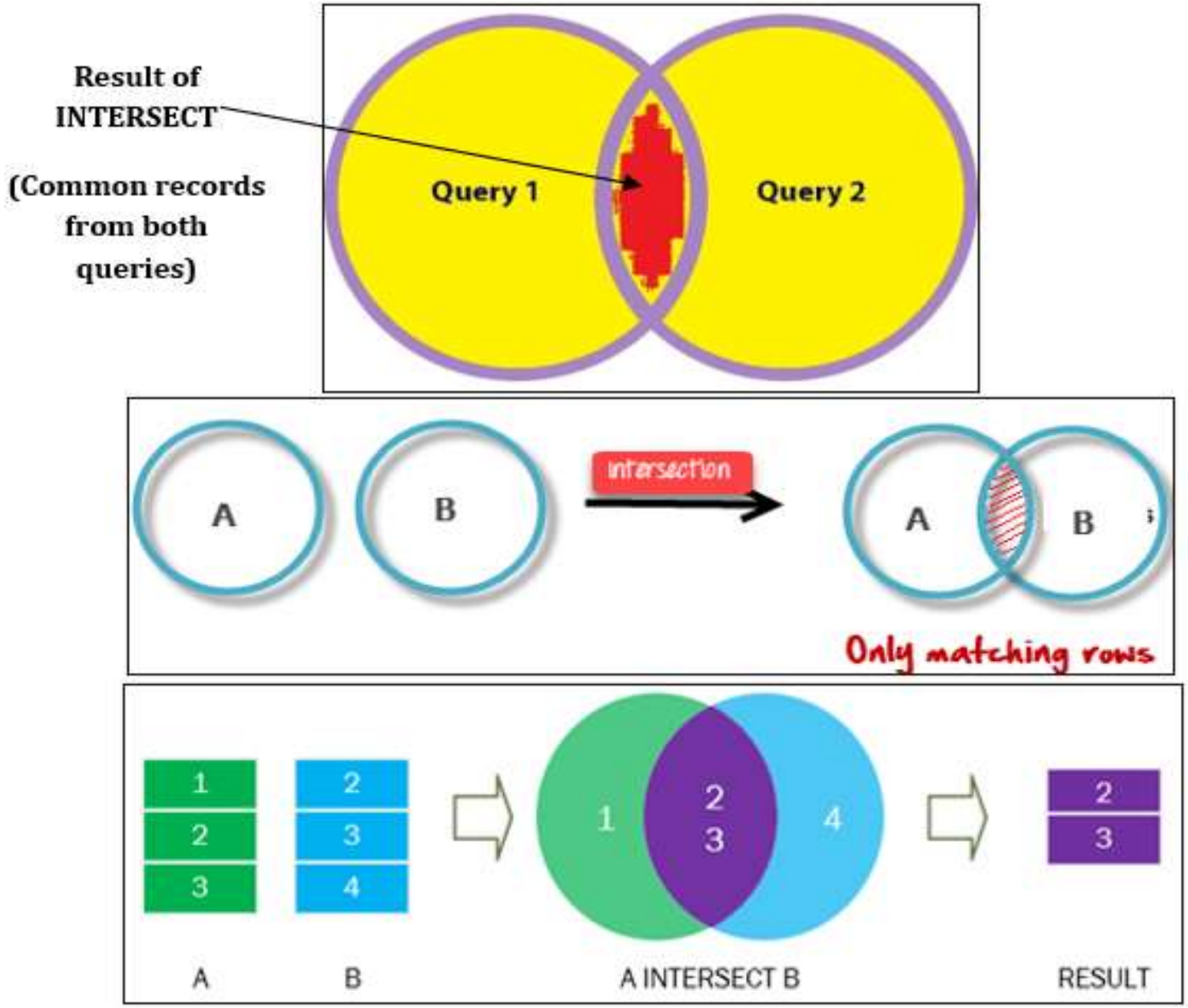
test1 UNION ALL test2

```
SQL> SELECT * FROM test1 UNION ALL SELECT * FROM test2;

        ID NAME
---------- ----------
         1 bhagya
         2 aarush
         3 shreeja
         3 shreeja
         4 avani
         5 aarav

6 rows selected.
```

(here we can see the duplicate rows which is removed from UNION)

2.5 SQL SET Operators (INTERSECT)

❖ INTERSECT



2.5 SQL SET Operators (INTERSECT)

- It retrieves the information which is common in both tables.
- Intersect operation is used to combine two SELECT statements, but it only returns the records which are common from both SELECT statements.
- All the data types and columns must be same in both the tables.
- Duplicates are ignored and result is generated in ascending order.
- NULL values are ignored.

Example (*apply intersect on both the tables test1 and test2***)**

Table: *test1*

ID	NAME
1	bhagya
2	aarush
3	shreeja

Table: *test2*

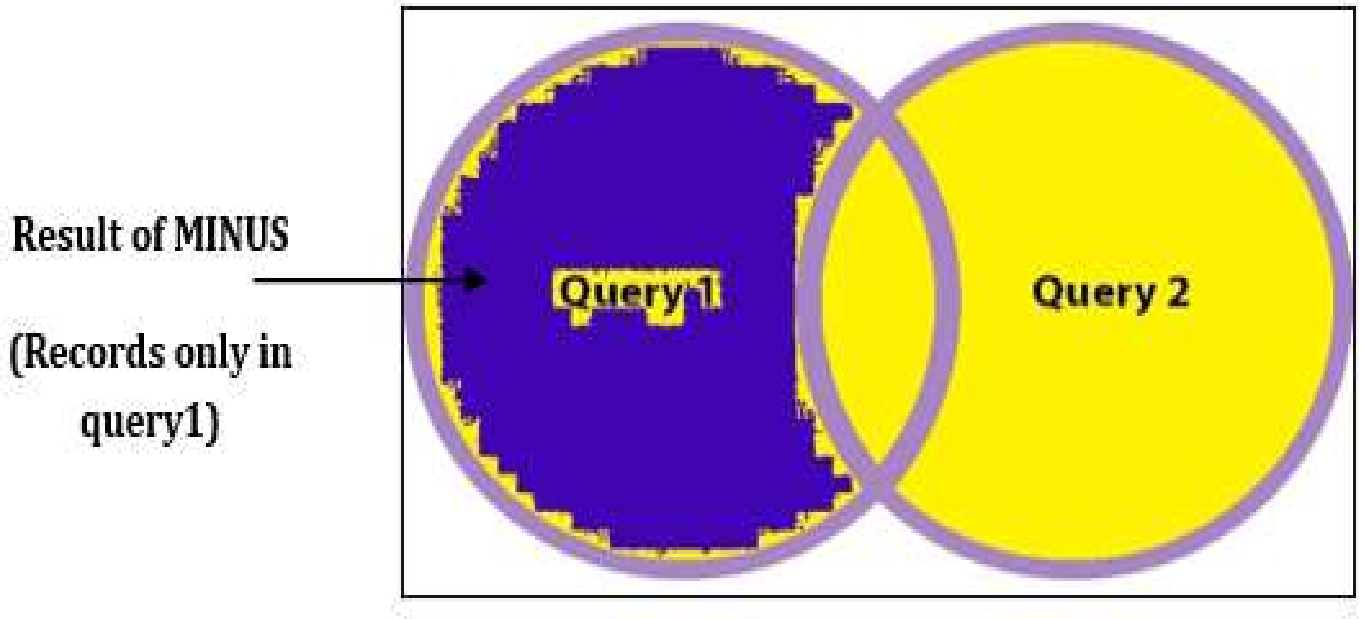
ID	NAME
3	shreeja
4	avani
5	aarav

test1 INTERSECT test2

```
SQL> SELECT * FROM test1
  2  INTERSECT
  3  SELECT * FROM test2;

        ID NAME
---------- ----------
         3 shreeja
```

2.5 SQL SET Operators (DIFFERENCE)

❖ DIFFERENCE (MINUS)



- It combines two SELECT queries.
- It is used to display the rows which are present in the first query but absent in the second query.
- Data types and columns must be same in both the tables.
- Duplicates are ignored and result is generated in ascending order.
- NULL values are ignored.

2.5 SQL SET Operators (DIFFERENCE)

Example (*apply difference on both the tables test1 and test2***)**

Table: *test1*

ID	NAME
1	bhagya
2	aarush
3	shreeja

Table: *test2*

ID	NAME
3	shreeja
4	avani
5	aarav

test1 MINUS test2

```
SQL> SELECT * FROM test1
  2  MINUS
  3  SELECT * FROM test2;

        ID NAME
---------- ----------
         1 bhagya
         2 aarush
```

2.5 SQL SET Operators (DIFFERENCE)

Another example (*consider the following tables***)**

collegeA	
id	**name**
a1	Aone
a2	Atwo
a3	Athree
a4	Afour

collegeB	
id	**name**
b1	Bone
b2	Btwo
b3	Bthree
b4	Bfour

hostelABCD	
id	**name**
a1	Aone
b2	Btwo
a3	Athree
a4	Afour
c1	Cone
d2	Dtwo

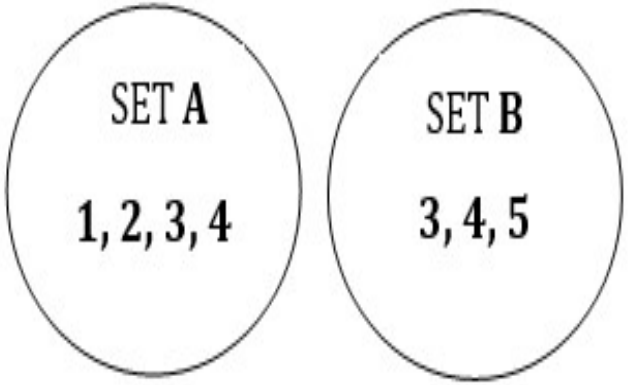
Display records of all the students of collegeA who are also living in hostel.

```
SQL> SELECT * FROM collegeA
  2  INTERSECT
  3  SELECT * FROM hostelABCD;

ID     NAME
-----  ---------------
a1     Aone
a3     Athree
a4     Afour
```

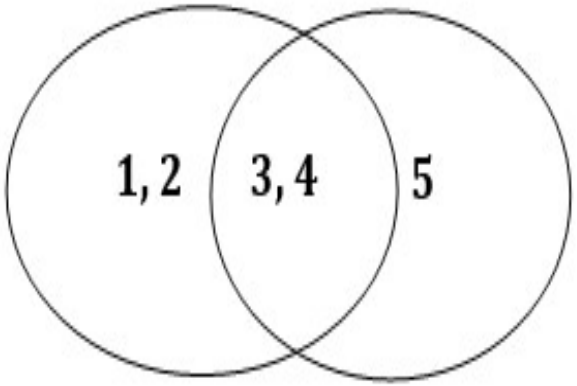
Display records of only those students of collegeA and collegeB who are living in hostel. (try it yourself)

2.5 SQL SET Operators

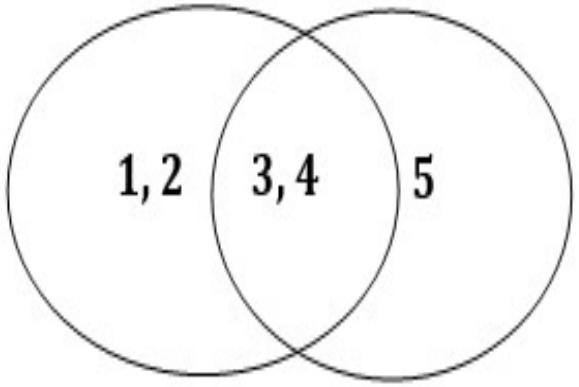
SUMMARY OF ALL SET OPERATORS



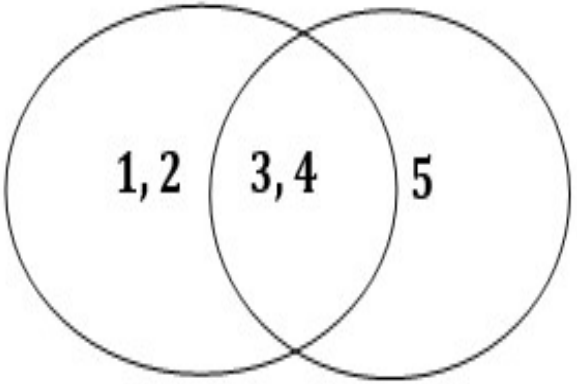
A UNION B = (1,2,3,4,5)



A INTERSECT B = (3,4)



A MINUS B = (1,2)



A UNION ALL B = (1,2,3,3,4,4,5)

Assignment (MCQs)

1 Among the following which is not an SQL operator?

a) Logical operator b) Character operators

c) Range operators d) Set operators

2 What does the SQL operator "IN" do?

a) Performs pattern matching within strings

b) Checks if a value exists in a given set

c) Retrieves rows based on a condition applied to more than one column

d) Checks if a value exists in a given set

3 Which of the following is NOT the SQL Arithmetic Operator?

a) Modulus b) Addition

c) Subtraction d) Unary

4 Which SQL operator is used to sort the result set in descending order?

a) DESC b) ASC

c) ORDER BY d) GROUP BY

5 Which type of join returns all rows from both tables, joining records where available and inserting NULLs where there is no match?

a) INNER JOIN b) LEFT OUTER JOIN

c) RIGHT OUTER JOIN d) FULL OUTER JOIN

6 In a LEFT JOIN, if there is no match for a row from the left table in the right table, what value will the columns from the right table contain?

a) NULL b) Blank

c) Error d) 0

Assignment (MCQs)

7 Which SQL join type is also known as an "equi-join"?

a) OUTER-JOIN
b) INNER-JOIN
c) CROSS-JOIN
d) SELF-JOIN

8 What does a CROSS JOIN do in SQL?

a) Returns only the rows that have matching values in both tables
b) Returns all possible combinations of rows from both tables
c) Returns all rows from the left table, and the matched rows from the right table
d) Returns only the unmatched rows from the left table

9 Among the following, which is not a type of join in SQL?

a) OUTER JOIN
b) SELF JOIN
c) FULL JOIN
d) EMPTY JOIN

10 Relation R1 has 5 tuples and 5 attributes. Relation R2 has 0 tuples and 5 attributes. When a CROSS JOIN is achieved between R1 and R2, how many tuples would the resultant set have?

a) 0
b) 25
c) 5
d) 10

11 Which join is to be used between two tables A and B when the resultant table needs rows from A and B that matches the condition and rows from A that does not match the condition?

a) CROSS JOIN
b) INBER JOIN
c) OUTER JOIN
d) CROSS PRODUCT

Assignment (MCQs)

12 Which is from the following is character operator?

a) SELECT
b) IN
c) STRING
d) LIKE

13 _____ is a query within another query.

a) Meta-query
b) Sub-query
c) Master-query
d) Super-query

Assignment (TRUE / FALSE)

1 In case of inner join between two tables in SQL, INNER keyword is optional.

2 The SQL operator "AND" is used to concatenate two or more strings together.

3 Right outer join is better than left outer join.

4 INNER JOIN is giving better results than CROSS JOIN.

5 Scalar functions are also called single raw functions.

6 We can change the schema of the dual table.

7 count ( ) is a scalar function.

8 < (less than) is a logical operator in SQL.

9 A NATURAL JOIN in SQL is based on the columns with the same name and datatype in both tables.

10 A CROSS JOIN combines each row from the first table with every row in the second table.

Assignment (Short Questions)

Sr. No	Questions	Bloom's Taxonomy Level
1	Define operator. List various types of SQL operators. *[S23]*	R
2	List various aggregate functions.	R
3	Find the output for the following: 1. select round (to_date ('13-nov-81'), 'year') from dual; 2. select abs (-1.2) from dual; 3. select round (654.351,2), trunc (654.351, 2) from dual; 4. select round (654.351,-2), trunc (654.351, -2) from dual; 5. select rtrim ('aarush', 'h1a2sg') from dual; 6. select replace ('bh1a2gya', '12a', 'XYZ') from dual; select to_char (to_date ('13-nov-'81'), 'day-mon-yyyy') from dual;	A
4	Explain SUM, GREATEST, POWER, SUBSTRING, TO_DATE, TO_CHAR functions of SQL. *[W22] [S23]*	U
5	List various types of joins. *[W22] [S23]*	R
6	Write the syntax of INNER JOIN. And explain Equi-join and non-equi join with example. *[W22]*	R, U
7	Explain SQRT, INITCAP, ROUND, LTRIM functions of SQL. *[W22]*	U
8	Explain IN and NOT IN operators in SQL. *[W23]*	U

Assignment (Short Questions)

Sr. No	Questions	Bloom's Taxonomy Level
1	List different logical operators. Explain logical operators with example. *[S23]*	R, U
2	Give syntax and Examples of any 4 SQL Character Functions. *[S24]*	U
3	Explain relational (comparison operator) with example. *[S23]*	U
4	Explain add_months, months_between, floor functions of SQL with example. *[S23]* **OR** Give syntax and examples of any 4 SQL date functions *[S24]*	U
5	Explain GROUP BY and HAVING clause of SQL with syntax and example. *[W22] [W23]*	U
6	Explain outer join operation of SQL with example. *[S23] [S24]* **OR** List out types of outer joins. Illustrate each type with example. *[W23]*	U
7	List various SQL SET operations. Explain UNION, INTERSECT and MINUS operation with example. *[W22]* **OR** List and explain SQL set operators with examples. *[S24]*	R, U
8	Explain sub-query concept with syntax and example. *[W23]* **OR** Define sub-query. List various types of sub-queries. Provide each type with example. *[S24]*	U
9	What is self-join? Give an example with a query. *[S24]*	R, A

Bloom's Taxonomy Levels: R (Remember), U (Understanding), A (Apply)

Thank you...

Prepared by:

Uresh N. Parmar

Lecturer, Computer Dept.
C U Shah Govt. Polytechnic
Surendranagar