Unit – 2
SQL In built Functions and Joins

Y.A.HATHALIYA

CO and Topics Covered

CO2: Perform joins, subqueries and nested queries on multiple tables using SQL*plus

Topics:

- Introduction of SQL Operators.
- SQL Single Row & Date Functions.
- SQL Numeric & Conversion Functions.
- SQL Character Functions.
- Introduction of Queries on ‘Group by’, ‘Having’ & ‘Order by’ Clause.
- Introduction of Joins (Simple, Equi, Non Equi, Self and Outer).
- Introduction of Other Types of Queries : Sub, Multiple, Correlated.
- Implementation of Queries Using SQL Set Operators(Union, Union All, Intersect, Minus).

- Create Table: person

SR	NAME	SURNAME	BDATE	CITY	SPI	BALANCE
1	bhagya	parikh	15-JAN-10	snagar	9.3	15000
2	vihan	sagar	22-FEB-12	rangpur	7.1	10000
3	shreeja	parikh	22-JUL-10	rajkot	9	5000
4	samay	saxena	10-AUG-10	ahmedabad	8.3	1000
5	tushar	parmar	13-NOV-90	palanpur	6.3	19000
6	aarush	parikh	15-JAN-15	snagar	8.12	22000
7	avani	solanki	15-MAY-10	ahmedabad	7.3	200
8	misha	sutariya	22-DEC-12	ahmedabad	5.5	55000
9	parth	parikh	15-SEP-95	rajkot	9.3	5000
10	aarav	solanki	11-SEP-10	snagar	9.3	1200

Operators in SQL

- Arithmetic Operators
  - It is used to perform various arithmetic operations with SQL Query.
  - The Arithmetic Operators are listed below,

Operator	Specifies
+	Addition
-	Subtraction
*	Multiplication
/	Division
( )	Enclosed Operation
**	Exponentiation

Example:

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

Input command →

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

Output →

```
   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
```

- Relational/Comparison Operators
  - It performs comparison among values and It returns three values only; true, false, unknown (when performed with NULL value).
  - The Relational Operators are listed below,

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

Example:

Display details of persons whose balance is greater than 15000.

Input command →

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

Output →

```
   SR NAME       SURNAME    BDATE     CITY            SPI    BALANCE
----- ---------- ---------- --------- ---------- ---------- ----------
    5 tushar     parmar     13-NOV-90 palanpur          6.3      19000
    6 aarush     parikh     15-JAN-15 snagar           8.12      22000
    8 misha      sutariya   22-DEC-12 ahmedabad         5.5      55000
```

- Logical Operators

❑ AND Operator

- It compares two or more expressions and returns true if all conditions are true, otherwise false
- All conditions must be true.
- Generally used in HAVING and WHERE clause

Example:

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

Input command →

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

Output →

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

OR Operator

- It compares two or more expressions and returns true if anyone conditions are true, otherwise false
- Any one condition required true.
- Generally used in HAVING and WHERE clause

Example:

Display name, surname and balance of all persons whose city=Ahmadabad or balance is less than 5000.

Input command →

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

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

Output →

NOT Operator

- It is used to negate the result of any condition or group of conditions.

Example:

Display records of all the persons who do not belong to the city Ahmadabad.

Input command →

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

Output →

SR	NAME	SURNAME	BDATE	CITY	SPI	BALANCE
1	bhagya	parikh	15-JAN-10	snagar	9.3	15000
2	vihan	sagar	22-FEB-12	rangpur	7.1	10000
3	shreeja	parikh	22-JUL-10	rajkot	9	5000
5	tushar	parmar	13-NOV-90	palanpur	6.3	19000
6	aarush	parikh	15-JAN-15	snagar	8.12	22000
9	parth	parikh	15-SEP-95	rajkot	9.3	5000
10	aarav	solanki	11-SEP-10	snagar	9.3	1200

```
7 rows selected.
```

BETWEEN Operator

- Used to select data that belong to some particular range.

Example:

Display persons having balance between 10000 and 20000.

Input command →

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

Output →

```
    SR NAME       SURNAME    BDATE     CITY            SPI    BALANCE
------ ---------- ---------- --------- ---------- ---------- ----------
     1 bhagya     parikh     15-JAN-10 snagar            9.3      15000
     2 vihan      sagar      22-FEB-12 rangpur           7.1      10000
     5 tushar     parmar     13-NOV-90 palanpur          6.3      19000
```

IN Operator

- Used to select data that belongs to some particular set of values.
- This is similar to =, but = compares single value to another single value while IN compares a single value to a list of values provided with IN predicate.
- It can be used when there is a need to use multiple OR conditions.

Example:

Display persons who belong to city 'ahmedabad', 'rangpur' or 'palanpur'.

Input command →

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

Output →

SR	NAME	SURNAME	BDATE	CITY	SPI	BALANCE
2	vihan	sagar	22-FEB-12	rangpur	7.1	10000
4	samay	saxena	10-AUG-10	ahmedabad	8.3	1000
5	tushar	parmar	13-NOV-90	palanpur	6.3	19000
7	avani	solanki	15-MAY-10	ahmedabad	7.3	200
8	misha	sutariya	22-DEC-12	ahmedabad	5.5	55000

LIKE Operator

- Used for character operator.
- This is similar to = , but = operator compares for exact matching while LIKE compares for pattern similarity.
- The LIKE operator is used with two special characters that is % and _(underscore)
- % allows matching with any string having any number of characters
- _(underscore) allows matching with single character

Examples of LIKE operator are explained below:

1. *Display name of persons starting with 'a'.*

Input command →

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

Output →

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

2. *Display the name of persons have 'a' as second character in their name.*

Input command →

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

Output →

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

3. *Display name and surname of persons having 5 characters in their name.*

Input command →

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

Output →

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

6 rows selected.
```

Concatenation Operator

- Used for combine two string or more string.

Example:

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

Input command →

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

Output →

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

10 rows selected.
```

- SQL Functions
  - SQL provides many functions to perform various operations on data.
  - It can be divided in two parts:
    - Aggregate Functions (Group Functions)
      - A Functions that are operate on set of rows (values).
      - It returns single values after performing operations.
      - Example:
        - » MIN( ),MAX( ),SUM( ),AVG( ),COUNT( ),COUNT(*)
    - Scaler Functions (Single Row Functions)
      - A Functions that are operate on a single of row (value).
      - It returns single value from a input value.
      - Example:
        - » Date, Numeric, Character and Conversion Function

Aggregate/Group Function

Function Name	Description	Query	Output
MIN( )	It returns the minimum value in the specified column. Syntax: MIN (columnName)	select min (balance) from person;	200
MAX ( )	It returns the maximum value in the specified column. Syntax: MAX (columnName)	select max (balance) from person;	55000
SUM ( )	It returns the sum of values of the specified column. Syntax: SUM (columnName)	select sum (balance) from person;	133400

Function Name	Description	Query	Output
AVG ( )	It returns average value of the specified column. Syntax: AVG(columnName)	select avg (spi) from person;	7.952
COUNT (*)	Returns number of rows in a table including duplicate and null values Syntax: COUNT (*)	select count (*) from person;	10
COUNT	Returns number of rows where column does not contain null values Syntax: COUNT (columnName)	select count (city) from person;	10

Date Function

Function Name	Description	Query	Output
ADD_MONTHS ( )	It returns the date after adding the number of months specified within the function. Syntax: ADD_MONTHS(date,n)	select add_months('01-OCT-2017','5') from dual;	01-MAR-18
MONTHS_BETWEEN ( )	It Finds the number of months between date1 and date2 Syntax: MONTHS_BETWEEN(date1, date2)	Select months_between('01-OCT-19','01-OCT-18') from dual;	12

Function Name	Description	Query	Output

ROUND ( )	This function round a date in specified format.
	
	Syntax:
	ROUND (date,format)
		Select round(SYSDATE) from dual;
		
		select round(TO_DATE('01-MAR-19'),'year') from dual;
		
		select round(TO_DATE('17-MAR-19'),'MONTH') from dual;
			14-SEP-23
			
			01-JAN-19
			
			01-APR-19

Function Name	Description	Query	Output

TRUNC ( )	This function trunc a date in specified format.
	
	Syntax:
	TRUNC (date,format)
		select trunc(SYSDATE) from dual;
		
		select trunc(TO_DATE('13 November 2023')) from dual;
		
		select trunc(TO_DATE('17 November 2023'),'month') from dual;
		
		select trunc(TO_DATE('13 November 2023'),'year') from dual;
			13-SEP-23
			
			13-NOV-23
			
			01-NOV-23
			
			01-JAN-23

Function Name	Description	Query	Output
NEXT_DAY ( )	Find the date of the next specified day of the week Syntax: NEXT_DAY(date,day)	Select next_day(SYSDATE,'Thursday') from dual; select next_day('01-MAR-23','FRIDAY') from dual;	14-SEP-23 03-MAR-23

LAST_DAY ( )	Find the date of the last day of the month that contains date
	
	Syntax:
	LAST_DAY (date)
		select last_day(SYSDATE) from dual;
		
		select last_day('01-MAR-23') from dual;
			30-SEP-23
			
			31-MAR-23

Function Name	Description	Query	Output
GREATEST ( )	It displays the latest date from a list of dates. Syntax: GREATEST (date1, date2, date3 …)	select GREATEST ('02-MAR-15', '02-DEC-08') from dual;	02-MAR-15

Numeric Function

Function Name	Description	Query	Output

ABS ( )	Returns absolute value of n
	
	Syntax:
	abs (n)
		select abs(-15) from dual;
		
		select abs(10.5) from dual;
			15
			
			10

POWER ( )	It returns the value raised to a given positive exponent. Syntax: power (value, exponent)	select power (2,3) from dual;	8
MOD ( )	It divides a value by a divisor and returns the remainder. Syntax: mod (value, divisor)	select mod (8,3) from dual;	2

Function Name	Description	Query	Output
ROUND ( )	It rounds a number to given number of digits of precision Syntax: round (value, precision)	select round (5.6356,2) from dual; select round (5.6356) from dual; select round (5.6356,-2) from dual;	5.64 6 0

TRUNC ( )	It truncates digits of precision from a number
	
	Syntax:
	trunc (value, precision)
		select trunc (8.656,2) from dual;
		
		select trunc(8.656) from dual;
		
		select trunc(8.656,-2) from dual;
			8.65
			
			8
			
			0

Function Name	Description	Query	Output
COS ( )	It returns the cosine of a given value. Syntax: cos (n)	select cos (0) from dual;	1
COSH ( )	It returns hyperbolic cosine of a given value. Syntax: cosh (n)	select cosh (0) from dual;	1
EXP ( )	It returns e raised to the n th power, where e = 2.71828183 Syntax: exp (n)	select exp (2) from dual;	7.3890561

Function Name	Description	Query	Output
CEIL ( )	It returns the smallest integer that is greater than or equal to a specific value. Syntax: ceil (n)	select ceil (25.2) from dual;	26
FLOOR ( )	It returns the greatest integer that is less than or equal to a specific value. Syntax: floor (n)	Select floor (25.2) from dual;	25
SQRT ( )	It returns the square root given number Syntax: sqrt (n)	select sqrt (16) from dual;	4

Character Function

Function Name	Description	Query	Output
INITCAP ( )	Returns a string with the first letter of each word in UPPER CASE. Syntax: initcap (string)	select initcap ('yagnik') from dual;	Yagnik
LOWER ( )	It takes any string as a input and converts it into lowercase string Syntax: lower (string)	select lower ('YAGNIK') from dual;	yagnik
UPPER ( )	It takes any string as a input and converts it into uppercase string Syntax: upper (string)	select upper ('yagnik') from dual;	YAGNIK

Function Name	Description	Query	Output

LTRIM ( )	This function removes (trims) the character from the left of the string.
	
	Characters will be removed up to the first character not in set.
	
	Syntax:
	ltrim (string, set)
		select ltrim('123yagnik',123) from dual;
		
		select ltrim('bhagya','habxya') from dual;
			Yagnik
			
			gya

RTRIM ( )	This function removes (trims) the character from the right of the string.
	
	Characters will be removed up to the first character not in set.
	
	Syntax:
	ltrim (string, set)
		select rtrim('123yagnik','ik') from dual;
		
		select rtrim('bhagya','a') from dual;
			123yagn
			
			bhagy

Function Name	Description	Query	Output
REPLACE ( )	It replaces a character in a string with zero or more character. Syntax: replace (string, character to be replaced, characters)	select replace ('yagnik12thaliya','12','ha') from dual;	yagnikha thaliya
TRANSLATE ( )	Replace a sequence of character in a string with another set of character Syntax: translate (string,string to replace, replacement string)	select translate ('1sct523','123','7a9') from dual;	7sct5a9

Function Name	Description	Query	Output
LENGTH ( )	Returns the number of character in a given string Syntax: length (string)	select length ('YAGNIK') from dual;	6
SUBSTRING ( )	It returns a substring from a given string Syntax: Substr (string,position,length)	select substr('yagnik',1,4) from dual;	yagn

Conversion Function

Function Name	Description	Query	Output
TO_NUMBER ( )	Converting Character to number It returns the number of value equivalent to input string String must consist 0-9,decimal point, + or - Syntax: TO_NUMBER ('char')	select TO_NUMBER('100') from dual;	100

✍ *(TO_NUMBER() function માં string ની value single quote માં નંબર value હોવી જોઈએ.)*

Function Name	Description	Query	Output
TO_CHAR( )	Converting Number to Character Syntax: TO_CHAR (n, format)	select TO_CHAR(12345,'99,999') from dual;	12,345

✍ (આ function નંબર value ને ફોરમેટ વાળા character value માં convert કરે છે, જે format માં ફક્ત 0, 9 કે comma (,) જ હોવા જોઈએ.)

TO_CHAR( )	Converting date to Character Syntax: TO_CHAR (date, format)	select TO_CHAR(SYSDATE,'dd-mon-yyyy') from dual;	15-sep-2023

✍ (આ function, date ને string ની 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
DDD	No. of day of year	HH	Hour
Y	Last I digit of year	MI	Minutes
YY	Last 2 digits of year	SS	Second

Function Name	Description	Query	Output
TO_DATE ( )	Converting character to date Syntax: TO_DATE (string, format)	select TO_DATE(140692,'DD-MM-YY') from dual;	14-JUN-92

✍ *(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.)*

ORDER BY Clause

- Sorting data of a table in ascending or descending order.
- Rows are sorted based on column specified with ORDER BY
- By default rows are sort in ascending order
- To sort rows in descending order we have to write DESC

Syntax:

```
SELECT columnNames
FROM tabel_name
ORDER BY columnName;
```

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
```

Display name and balance of all persons according to their balance in descending order.

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

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

GROUP BY Clause

- Used to group rows that have a same values.
- Used with select statement.
- It is placed after WHERE condition and before ORDER BY Clause.

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
```

Example1:

Display the balance of all grouped surnames.

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

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

6 rows selected.
```

Example2:

Display the count of all surnames based on their group.

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

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

6 rows selected.
```

HAVING Clause

- It places the condition in the group defined by GROUP BY Clause in select statement.
- It filters records that work on summarized GROUP BY result.

Syntax:

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

Example:

Display how many persons having surname as 'parmar'.

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

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

6 rows selected.
```

Input command →

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

Output →

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

JOINs

- Purpose of join clause is to combine record from two or more tables from database.
- Multiple tables can be join using common columns.
- It 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 and Full Outer Join)

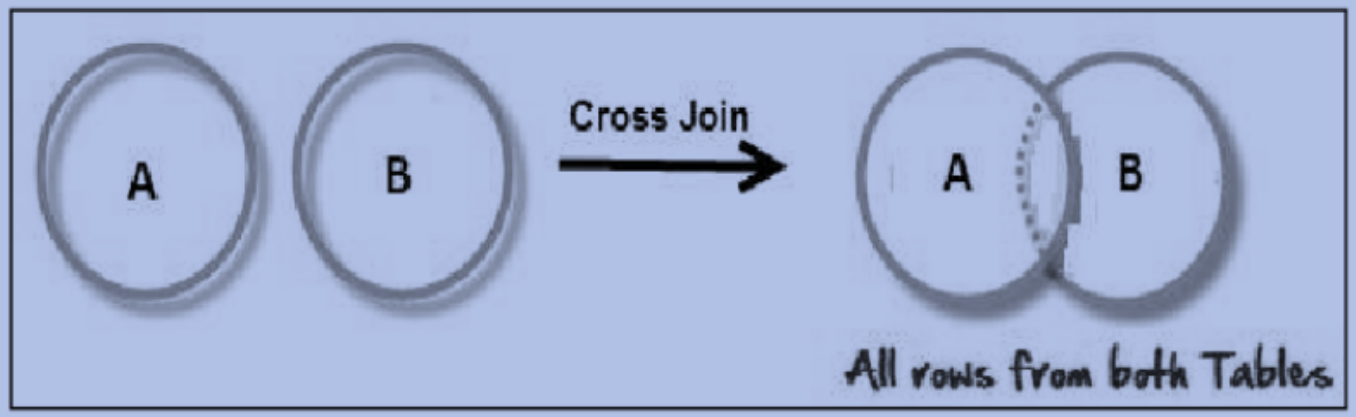
Table name: *account*

ANO	BALANCE	BNAME
a01	5000	vvn
a02	6000	ksad
a03	7000	anand
a04	8000	ksad
a05	6000	vvn

Table name: *branch*

BNAME	BADDRESS
vvn	mota bazzar,VVNagar
ksad	chhota bazzar, karamsad
anand	nana bazzar,Anand

Cross Join (Cartesian Product or Simple Join)



- Cross join is the simplest form of all joins so it is called Simple join.
- It combines every row from one table with every row of another table.
- The output of cross join is similar to Cartesian product and that is why this operation is referred as Cartesian product.

Syntax of Simple join:

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

(*Note: if two tables have same column names, then these columns can be separated using **tableName.columnName**)*

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

Table: *test1*

ID	NAME
1	bhagya
2	aarush
3	shreeja

Table: *test2*

ID	NAME
3	shreeja
4	avani
5	aarav

SELECT * FROM test1, test2;

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

Only consistent and true information

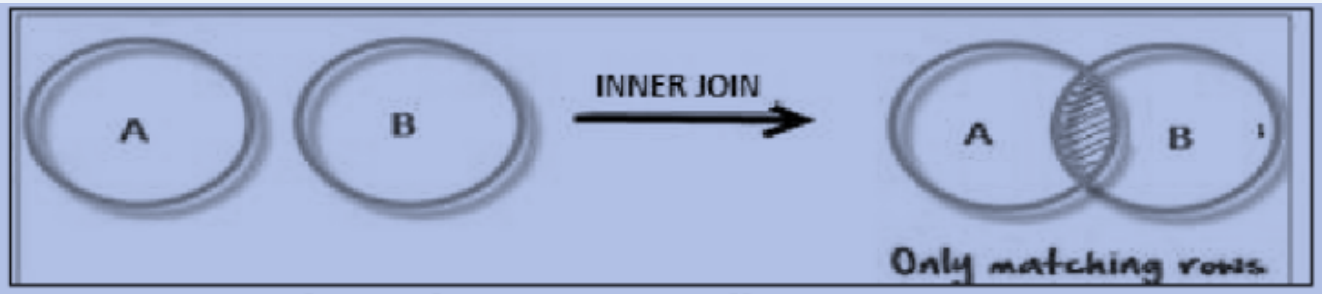
✍ (નોંધ: ઉપરના example પરથી આપણે જોઈ શકીએ છીએ કે record 7 સીવાયના બધાજ records irrelevant છે એટલે કે તે ખોટી માહિતી આપે છે. Cartesian product માં આ problem જોવા મળે છે. Due to this reason cross join is rarely used. And the solution for this problem is INNER JOIN

Combine information from 'account' and 'branch' tables.

SELECT * FROM account, branch;

ANO	BALANCE	BNAME	BNAME	BADDRESS
a01	5000	vvn	vvn	mota bazzar,VVNagar
a02	6000	ksad	vvn	mota bazzar,VVNagar
a03	7000	anand	vvn	mota bazzar,VVNagar
a04	8000	ksad	vvn	mota bazzar,VVNagar
a05	6000	vvn	vvn	mota bazzar,VVNagar
a01	5000	vvn	ksad	chhota bazzar, karamsad
a02	6000	ksad	ksad	chhota bazzar, karamsad
a03	7000	anand	ksad	chhota bazzar, karamsad
a04	8000	ksad	ksad	chhota bazzar, karamsad
a05	6000	vvn	ksad	chhota bazzar, karamsad
a01	5000	vvn	anand	nana bazzar,Anand
a02	6000	ksad	anand	nana bazzar,Anand
a03	7000	anand	anand	nana bazzar,Anand
a04	8000	ksad	anand	nana bazzar,Anand
a05	6000	vvn	anand	nana bazzar,Anand

Inner Join (Equi Join and Non Equi Join)



- This join is most commonly used joins among all joins.
- This operation provides filtering on the output of cross join operation.
- The INNER JOIN combines only those records which are common in both tables.
- It returns records that have matching values in both tables
- So, in contrast to cross join, inner join provides consistent information.

Syntax of Inner join:

```
SELECT column_Names
FROM table1, table2
WHERE table1.column1 OP
table2.column2;
```

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

OR

Syntax of Inner join:

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

- In INNER JOIN, if both the columns are compared for equality (i.e. OP is '='), then this operation is referred as ***EQUI-JOIN***. This join produces only those records which have same values for both columns.
- If both the columns are compared for non-equality (operation, other than '=') then this operation is referred as ***NON-EQUI JOIN***.

Example:

Combine only consistent information from 'account' and 'branch' tables. (for that we have to use INNE JOIN)

```
SQL> SELECT ano, balance, account.bname, baddress
     FROM account, branch
     WHERE account.bname=branch.bname;

ANO        BALANCE BNAME      BADDRESS
------ ---------- ---------- ----------------------
a01            5000 vvn        mota bazzar,VVNagar
a02            6000 ksad       chhota bazzar, karamsad
a03            7000 anand      nana bazzar,Anand
a04            8000 ksad       chhota bazzar, karamsad
a05            6000 vvn        mota bazzar,VVNagar
```

(The result of inner join can be considered as a separate table)

Another example (apply INNER JOIN on table *'test1' and 'test2'*)

Table: *test1*

ID	NAME
1	bhagya
2	aarush
3	shreeja

Table: *test2*

ID	CITY
2	surendranagar
3	rajkot
4	ahmedabad

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

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

Self Join

- As we have seen, join operation combines two different tables, but in some situation there is a need to join a table with itself.
- *The process (OR operation) of joining a table with itself is called '**Self join**'.*
- In self-join, the each row of a table is combined with other rows of the same table to produce result.
- It is generally used when two rows from the same table is combined to form a 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 can use alias of tables.

Syntax of Self-join:

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

Example:

Eid	Name	Mngr_id
E01	Kina	E02
E02	Mona	NULL
E03	Heena	E02
E04	Rekha	E02

```
SELECT emp.Eid, emp.name, mgnr.name
FROM Employee emp, Employee mngr
WHERE emp.mgnr_id = mgnr.eid
```

emp.Eid	emp.Name	mgnr.Name
E01	Kina	Mona
E03	Heena	Mona
E04	Rekha	Mona

Outer Join

- The outer join operation is an extension of the inner join operation.
- Inner join operation gives consistent information, but it may happen in some situations that in the result of inner join, there is a loss of information. The outer join solve that problem.

An example that shows the problem occurred in inner join:

Suppose that we are having two tables → 'college' and 'hostel' like below:

Table: college

NAME	ID	DEPT
manisha	s01	computer
anisha	s02	computer
nisha	s03	IT

Table: hostel

NAME	HOSTEL_NAME	ROOM_
anisha	kaveri hostel	k01
nisha	godavari hostel	g07
isha	kaveri hostel	k02

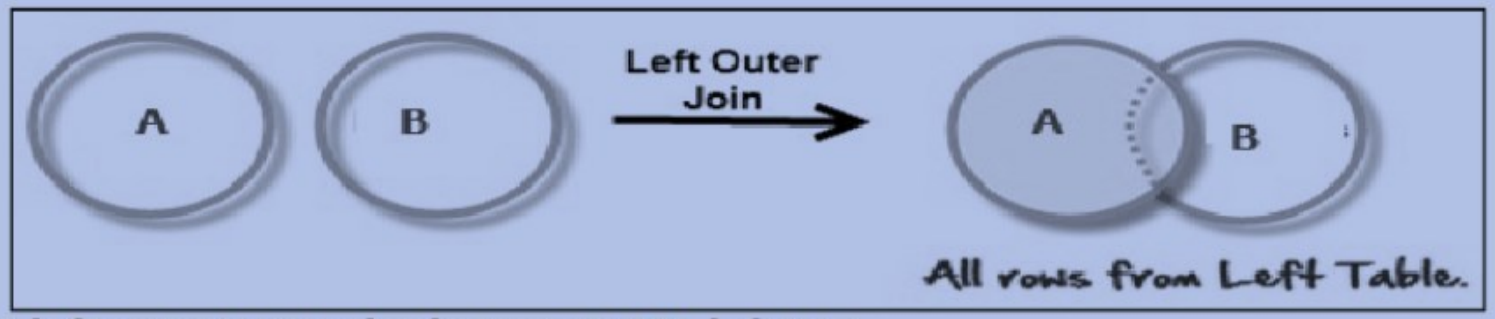
Now consider the inner join operation on above relations.

```
SELECT college.name, id, dept, hostel_name, room_no
FROM college, hostel
WHERE college.name = hostel.name;
```

NAME	ID	DEPT	HOSTEL_NAME	ROOM_
anisha	s02	computer	kaveri hostel	k01
nisha	s03	IT	godavari hostel	g07

- SO we can say that an outer join returns a set of records (or rows) that include what an inner join would return but also includes other rows for which no corresponding match is found in the other table.
- There are three kinds of outer joins:
  1. **Left Outer Join**
  2. **Right Outer Join**
  3. **Full Outer Join**

1. Left Outer Join



- SQL left outer join is also known as SQL left join.
- Suppose, we want to join two tables: A and B. SQL left outer join returns all rows in the left table (A) and all the matching rows found in the right table (B).
- 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."
- The rows for which there is no matching row on right side, the result-set will contain NULL value.

Syntax of Left outer join (Left join):

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

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

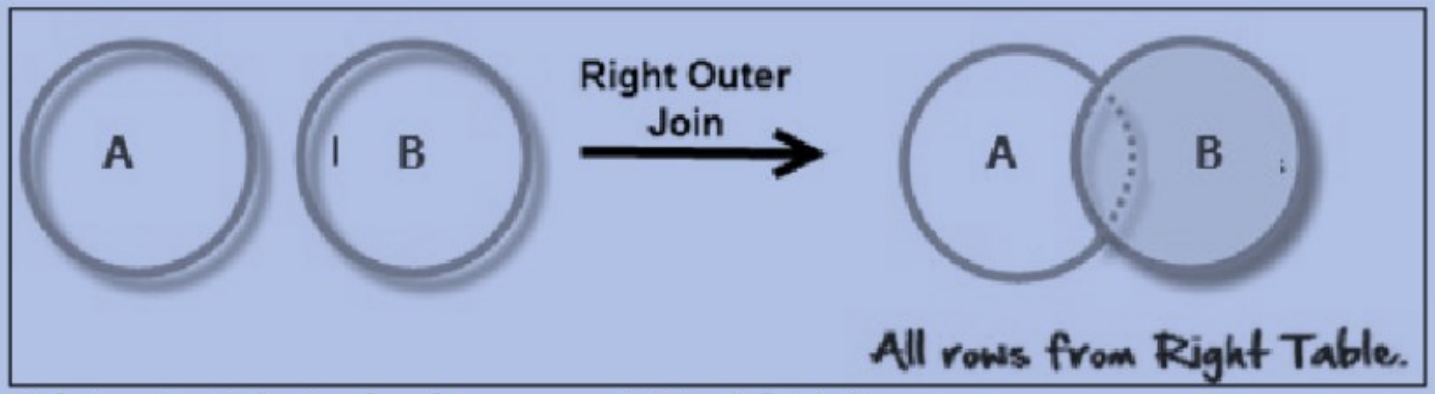
```
SQL> SELECT * FROM test1 LEFT JOIN test2
     ON test1.id = test2.id;

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

Another Example

Perform the left outer join operation on 'college' and 'hostel' tables.

```
SQL> SELECT * FROM college LEFT JOIN hostel
     ON college.name = hostel.name;

NAME       ID    DEPT       NAME       HOSTEL_NAME      ROOM_
---------- ----- ---------- ---------- ---------------- -----
anisha     s02   computer   anisha     kaveri hostel    k01
nisha      s03   IT         nisha      godavari hostel  g07
manisha    s01   computer
```

2. Right Outer Join



- SQL right outer join is also known as SQL right join.
- Suppose, we want to join two tables: A and B. SQL right outer join returns all rows in the right table (B) and all the matching rows found in the left table (A).
- 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."
- The rows for which there is no matching row on left side, the result-set will contain *NULL* value.

Syntax of Right outer join (Left join):

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

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

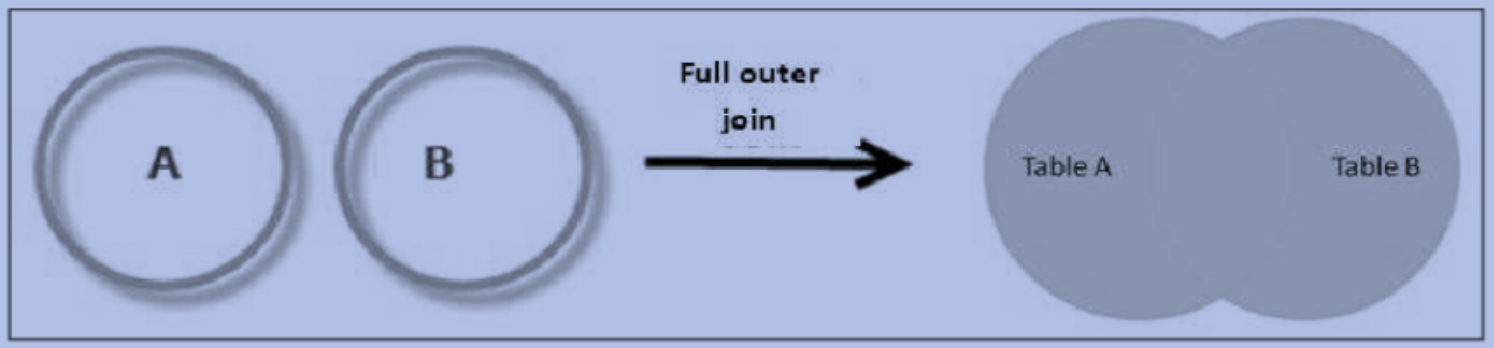
```
SQL> SELECT * FROM test1 RIGHT JOIN test2
     ON test1.id = test2.id;

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

Another Example

Perform the right outer join operation on 'college' and 'hostel' tables.

```
SQL> SELECT * FROM college RIGHT JOIN hostel
     ON college.name = hostel.name;

NAME       ID    DEPT       NAME       HOSTEL_NAME      ROOM_
---------- ----- ---------- ---------- ---------------- -----
anisha     s02   computer   anisha     kaveri hostel    k01
nisha      s03   IT         nisha      godavari hostel  g07
                            isha       kaveri hostel    k02
```

3. Full Outer Join



- SQL full outer join is also known as SQL full join.
- Suppose, we want to join two tables: A and B. SQL full outer join returns all the matching rows from both the tables, and also returns non-matching rows from both the tables with NULL padded values.
- Full join combines the result of left outer join and right outer join.
- Simply we can say→"This join returns all the rows from both the tables which left and right of the join give"
- The rows for which there is no matching row on left side and right side, the result-set will contain *NULL* value.

Syntax of Full outer join:

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

Example

Perform 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

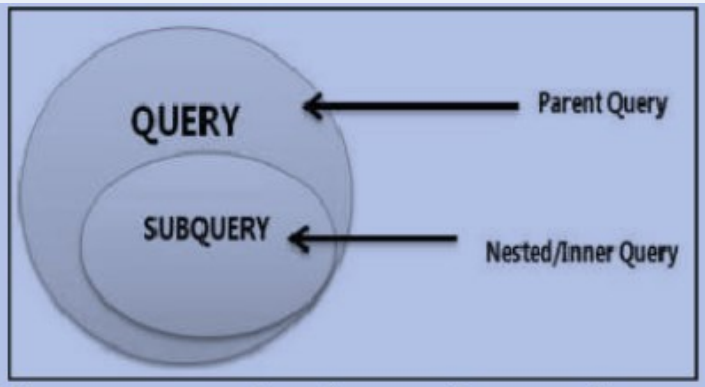
```
SQL> SELECT * FROM test1 FULL JOIN test2
     ON test1.id = test2.id;

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

Another Example

Perform the full outer join operation on 'college' and 'hostel' tables.

```
SQL> SELECT * FROM college FULL JOIN hostel
     ON college.name = hostel.name;

NAME       ID    DEPT       NAME       HOSTEL_NAME      ROOM_
---------- ----- ---------- ---------- ---------------- -----
anisha     s02   computer   anisha     kaveri hostel    k01
nisha      s03   IT         nisha      godavari hostel  g07
manisha    s01   computer
                            isha       kaveri hostel    k02
```

Sub Query



- 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, means one parent query contains sub-query and that sub-query may contains another sub-query and so on.
- You can place the sub-query in a number of SQL clauses: WHERE clause, HAVING clause, FROM clause.
- The sub-query generally executes first, and its output is used to complete the query condition for the main or outer query.
- Sub-query must be enclosed in parentheses.
- ORDER BY command cannot be used within a sub-query.
- BETWEEN operator cannot be used with a sub-query, but it can be used within sub-query.

Syntax of Sub-query:

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

Example:

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
```

To understand the concept of sub-query, here we will take three different tables, named '*customer*', '*account_holder*' and '*account*'.

Table: customer

CID	NAME
c01	riya
c02	jiya
c03	diya
c04	taral
c05	saral

Table: account_holder

CID	ANO
c01	a01
c02	a02
c03	a03
c04	a04
c05	a05
c02	a04

Table: account

ANO	BALANCE	BNAME
a01	5000	vvn
a02	6000	ksad
a03	7000	anand
a04	8000	ksad
a05	6000	vvn

Another example

Find out the balance of an account which belongs to customer 'c01'.

```
SQL> SELECT balance FROM account WHERE ano IN (SELECT ano FROM account_holder WHERE cid='c01');

   BALANCE
----------
      5000
```

- **Correlated sub-queries**
  - Correlated sub-queries are used for row-by-row processing. Each sub query is executed once for every row of the outer query.
  - The sub-query is known as a correlated because the sub-query is related to the outer query and vice-versa OR simply we can say 'a sub-query which is using values from outer queries'.
  - With a normal nested sub-query, the inner SELECT query runs first and executes once, returning values to be used by the main query.
  - 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.

Example:

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
     FROM person P
     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.
```

Another example

Find out the account having maximum balance in branch to which it belongs.

```
SQL> SELECT ano, balance, bname FROM account ACC
     WHERE balance IN (SELECT max(balance) FROM account
                       WHERE bname=ACC.bname);

ANO        BALANCE BNAME
------ ---------- ----------
a03           7000 anand
a04           8000 ksad
a05           6000 vvn
```

SQL Set Operators

- A table in relational database and relation in mathematics (algebra) both are same, as both represent one kind of set.
- SQL supports few Set operations which can be performed on the table data (records).
- 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 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.

Now, before understanding in details, first we take some tables as examples so that we can apply set operators on them.

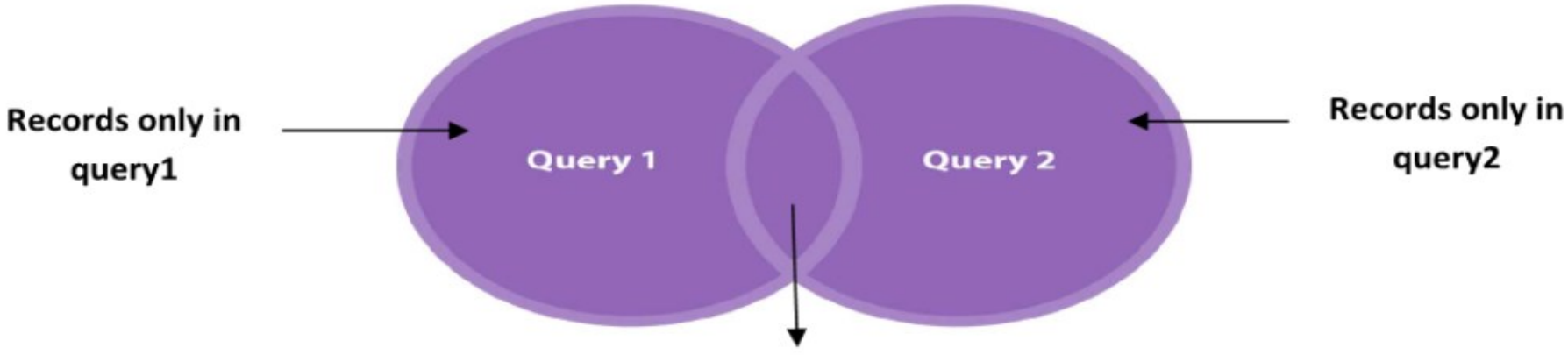
Table: customer

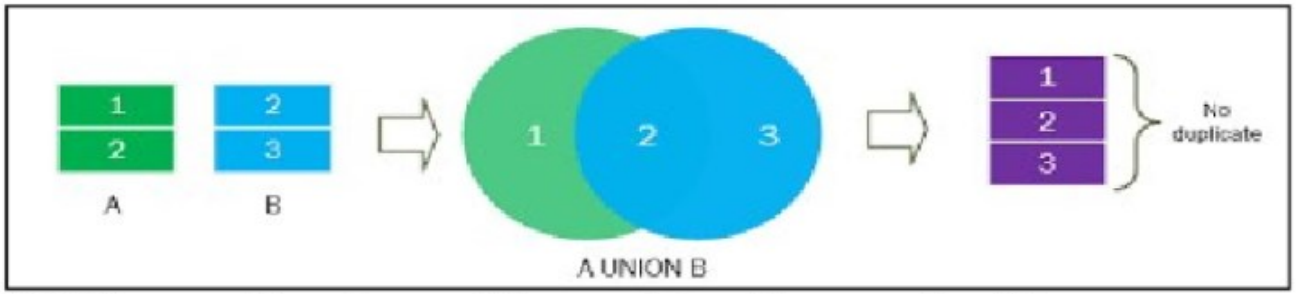
CID	NAME
c01	riya
c02	jiya
c03	diya
c04	taral
c05	saral

Table: employee

EID	NAME	MGR_ID
e01	palak	e02
e02	zalak	NULL
e03	falak	e02
e04	taral	e02
e05	saral	e02

◆ UNION





- The SQL UNION operation 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.
- The UNION operation eliminates the duplicate rows from its result set.
- The output will be as a single set of rows and columns.
- NULL values will not be ignored while comparing record from both queries.
- The output will contains sorted records in ascending order according to the column of first SELECT statement.

Example:

Suppose we have two tables named → '*test1*' and '*test2*'. Now we are doing 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
```

Another example

List out name of persons who are either customers or employees.

```
SQL> SELECT name FROM customer
     UNION
     SELECT name FROM employee;

NAME
----------
diya
falak
jiya
palak
riya
saral
taral
zalak

8 rows selected.
```

♦ UNION ALL

- SQL UNION operation combines the results of two SELECT queries and in output, duplicate values are removed, while in case of UNION ALL → it also results the output of two SELECT queries but it preserves the duplicates, means duplicates values are not removed.
- So, in a single statement we can say "*This operation is similar to Union. But it also shows the duplicate rows*".

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

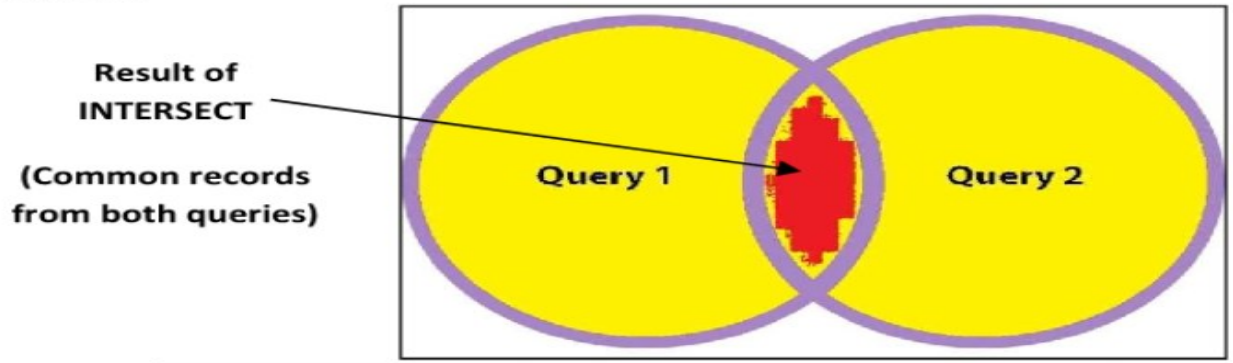
Table: *test2*

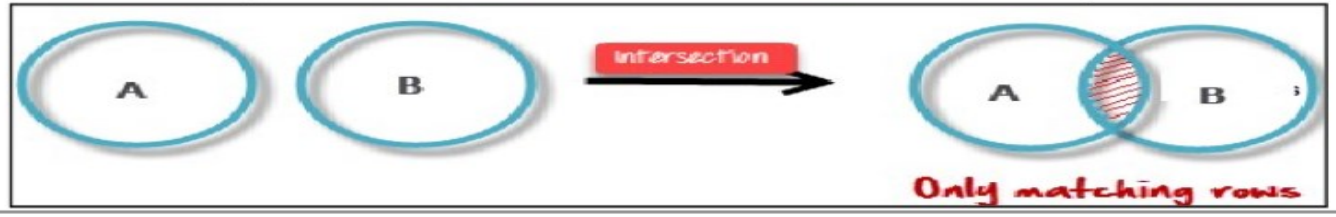
ID	NAME
3	shreeja
4	avani
5	aarav

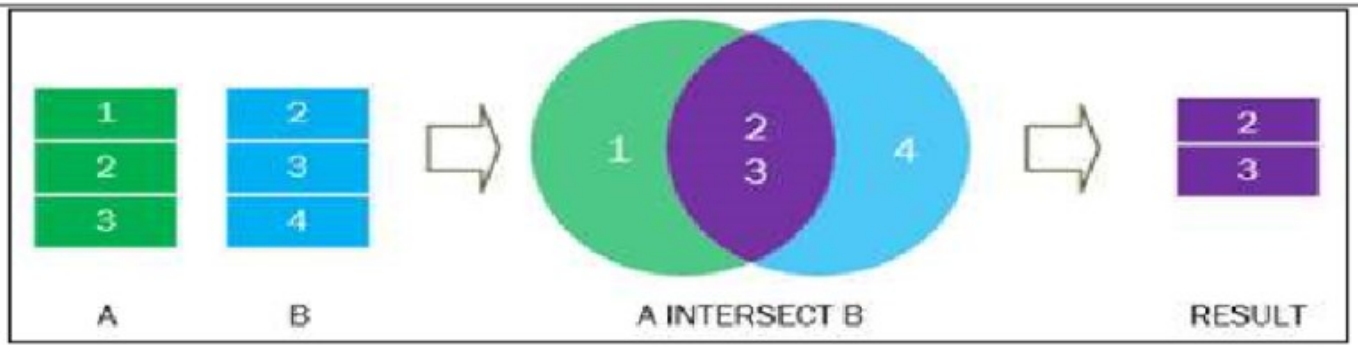
test1 UNION ALL test2

```
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.
```

- **INTERSECT**







- This set operator is used to retrieve 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.
- In the INTERSECT operation, all the datatype and columns must be same in both the tables on which INTERSECT operation is being applied.
- It has no duplicates and it arranges the data in ascending order by default.
- NULL values will not be ignored while comparing record from both queries.

Example (Apply intersect on test1 and test2 tables):

table: *test1*

ID	NAME
1	bhagya
2	aarush
3	shreeja

Table: *test2*

ID	NAME
3	shreeja
4	avani
5	aarav

test1 INTERSECT test2

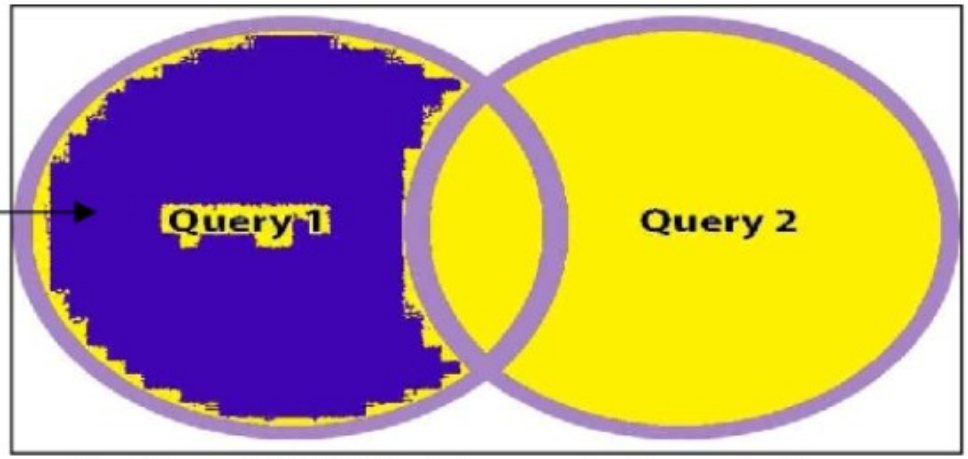
```
SQL> SELECT * FROM test1
     INTERSECT
     SELECT * FROM test2;

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

Another example

List out all the names of persons who are customers as well as employees.

```
SQL> SELECT name FROM customer
     INTERSECT
     SELECT name FROM employee;

NAME
----------
saral
taral
```

♦ DIFFERENCE (MINUS)



- It combines the result of two SELECT statements (queries).
- Minus operator is used to display the rows which are present in the first query but absent in the second query.
- In this operation, all the datatype and columns must be same in both the tables on which MINUS operation is being applied.
- It has no duplicates and data arranged in ascending order by default.
- NULL values will not be ignored while comparing record from both queries.

Example (apply difference on table *'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
     MINUS
     SELECT * FROM test2;

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

Another example

*List out all the names of persons who are customers **but not** employees.*

```
SQL> SELECT name FROM customer
     MINUS
     SELECT name FROM employee;

NAME
----------
diya
jiya
riya
```