10th Jan. 2022
Monday.

Unit - 4.

Advanced SQL.

* Relational set operators.

- SQL provides following relational set operators:
1.) UNION
2.) UNION ALL
3.) INTERSECT
4.) MINUS

1. UNION.

- The UNION statement combines rows from two or more queries without including duplicate rows.
- The UNION statement combines output of two ~~selected~~ SELECT queries.
- The SELECT statement must be union compatible which means that the number of attributes must be same with their data types also.

Syntax:-

```
    query UNION query;
              OR
SELECT columnname FROM table1
UNION
SELECT columnname FROM table2;
```

e.g.,

```
select * from STU
union
select * from STU1;
```

STU

sid	name
1	A
2	B
3	C

Union

STU1

sid	name
3	C
4	D
5	E

=>

sid	name
1	A
2	B
3	C
4	D
5	E

2. UNION ALL.

- The UNION ALL statement combines rows from two or more queries including duplicate rows.
- The UNION ALL operator must be union compatible which means that both the tables must have same columns as well as data types.

Syntax:-

```
query UNION ALL query;

        OR

SELECT columnname FROM table1
UNION ALL
SELECT columnname FROM table2;
```

e.g.,

```
select * from STU
union all
select * from STU1;
```

STU

sid	name
1	A
2	B
3	C

Union all

STU1

sid	name
3	C
4	D
5	E

=>

sid	name
1	A
2	B
3	C
3	C
4	D
5	E

3. INTERSECT.

- The INTERSECT operator can be used to combine rows from two queries returning only the rows that appear in both tables.

Syntax:-

```
query INTERSECT query;
```

OR

```
SELECT columnname FROM table1
INTERSECT
SELECT columnname FROM table2;
```

e.g.,

```
select * from STU
INTERSECT
select * from STU1;
```

STU

sid	name
1	A
2	B
3	C

INTERSECT

STU1

sid	name
3	C
4	D
5	E

=>

sid	name
3	C

4. MINUS.

→ The MINUS statement in SQL combines rows from two queries and returns only the rows that appear in first table but not in the second.

Syntax:-

```
query MINUS query;
```

OR

```
SELECT colummname FROM table1
MINUS
SELECT colummname FROM table2;
```

e.g.,

```
select * from STU
minus
select * from STU1;
```

* SQL functions.

1. Numeric functions.

- SQL provides following numeric functions:

1.) ABS
2.) ROUND
3.) CEILING
4.) FLOOR

1) ABS

- It returns absolute value of the specified number.

Syntax:-

```
   ABS (numeric-value)
```

e.g.,

```
Select  -1.93, ABS (-1.93)  from Dual;
```

Output:-

```
     -1.93    ABS (-1.93)
     -1.93      1.93
```

2) ROUND.

- It rounds a value to a specified precision.

Syntax:-

```
ROUND (numeric-value, P)
```

Where P = Precision.

e.g.,
Ro ROUND 2.34 with 1, 0 precision.

```
select round (2.34, 1),
       round (2.34, 0) from dual;
```

Output:-

```
    2.3     2
```

e.g.,
Display price of product with 2 decimal places.

select round ~~f2~~ (price, 2) from product;

3.) CEILING.

- It returns smallest integer greater than or equal to a number.

Syntax:-

```
CEIL (numeric-value);
```

e.g.,

```
select ceil (2.3) as ceil value from dual;
```

Output:-

```
        ceil(2.3)
        ---------
           3
```

4.) FLOOR.

- It returns nearest largest integer less than or equal to a number.

Syntax:-

```
FLOOR (numeric-value)
```

e.g.,

```
SELECT floor (2.3) as floor value from
                        dual;
```

Output:-

```
floor (2.3)
-----------
    ~~3~~ 2
```

2. String functions.

- SQL provides following string function:

1.) Concatenation (||)
2.) UPPER & LOWER
3.) LENGTH
4.) SUBSTR

1.) Concatenation (||)

- It combines data from two different character columns and returns single column.

Syntax:-

```
str-value1 || str-value2;
```

e.g.,
List id and name of students in a single column.

```
select sid||name from student;
```

OR

```
select sid||','||name from student;
```

Output:-

```
sid||','||name
     1, A
     2, B
     3, C
```

2.) UPPER & LOWER.

- The UPPER function returns a string in capital letters.

Syntax:-

```
UPPER (str-value);
```

e.g.,
List all students name in capital letters.

```
select upper(name) from student;
```

- The LOWER function returns string in lower case letters.

Syntax:-

```
LOWER (str-value);
```

e.g.,

- List all students name in lower case letters.

```
select lower(name) from student;
```

3.) LENGTH.

- It returns the number of characters in a string value.

Syntax:-

```
LENGTH (str-value);
```

e.g.,

Display name and its length of all students.

```
select name, length(name) from student;
```

4.) SUBSTR.

- SUBSTR returns a sub string or a part of a string of a given string value.

Syntax:-

~~St~~

```
SUBSTR (str-value, p, L);
```

Where

p = start position

L = length of characters.

e.g.,

Display first 3 characters of student's name.

```
select name, substr(name, 1, 3) from
        student;
```

24th Jan. 2022

Monday.

3. Date / Time functions.

1.) TO_CHAR

- It returns a character string or a formatted string from a date value.

Syntax:

```
TO_CHAR (date_value, fmt)
```

where fmt (format) can be

MONTH - name of the month

MON - 3 letter of the month name

MM - 2 digit month name

D - number for day of the week

DD - number for day of the month

DAY - name of day of week

YYYY - 4 digit year value
YY - 2 digit year value

e.g., ~~Di~~

- Display year from student's date of birth.

```
select sid, dob, to_char(dob, 'yyyy') as
   year from student;
```

- Display students who born in 1997.

```
select * from student
where dob = to_char(dob, 'yyyy') = '1997';
```

2.) SYSDATE.

- It returns system's today's date.

Syntax:
SYSDATE

e.g.,
Display current date of system.

```
select sysdate from dual;
```

3.) ADD_MONTHS

- It adds a number of months or years to a date.

Syntax:
ADD_MONTHS (date_value, n)
where
n = number of month.

e.g.,
Add 2 years in student's date of birth. and rename column as years added.

```
select sid, sname, dob, add_months(dob,24)
    as years_added from student;
```

4.) LASt_DAY

- It returns the last day of the given month in a date.

Syntax:

```
LAST_DAY(date_value)
```

e.g.,
Display last day of the month from student's date of birth.

```
select sname, dob, last_day(dob) from
        student;
```

5.) TO_DATE

- It returns a date value using a character string and date format mask.

Syntax:

```
TO_DATE(char_value, fmt)
```

where fmt can be
MONTH - name of the month
MON - 3 letter of the month name
MM - 2 digit month name
D - number for day of the week
DD - number for day of the month

DAY - name of day of week
YYYY - 4 digit year value
YY - 2 digit year value

~~e.g.,~~
NOTE :- '11/25/2012' is a text string, not a date.

4. Conversion function.

1.) TO_CHAR
- It returns a character string from numeric value.

Syntax:-
TO_CHAR (numeric_value, fmt)

e.g.,
Display student name and mark of the student with the format 99.99

```
select sname, mark, to_char (marks, '99.99')
       from student;
```

2.) TO_CHAR
- It returns a character string from a date value.

Syntax:
TO_CHAR (date_value, fmt)

e.g.,
Display date of birth in yyyy/mm/dd format of student.

```
select sid, sname, dob, to_char (dob,
       'yyyy/mm/dd') from student;
```

3.) TO_NUMBER.

- It returns a formatted number from a character string.

Syntax:
TO_NUMBER (char-value, fmt)
where format =

9 - displays a digit
0 - displays leading zero
, - displays comma
. - displays decimal point
$ - displays dollar sign
B - displays leading blank
S - leading sign
MI - prailing - (minus) sign

* Oracle sequences.

- Oracle does not support AutoNumber datatype or the identity column property.
- We can use a sequence to assign values to a column in a table.

→ Properties:

- It is not a data type. It is dependant object in database.
- Sequences have a name and can be used anywhere a value is expected.
- Sequences are not tied to a table or a column.
- It generate a numeric value that can be assigned to any column in any table.
- Sequence can be created and deleted any time.

Syntax:

```
CREATE SEQUENCE Name [START WITH n]
[INCREMENT BY n] [CACHE /NOCACHE];
```

where

Name - name of the sequence

Start with - specifies initial sequence value (default 1)

Increment by - determines value by which the sequence is incremented (default 1)

Cache or Nocache - indicates whether oracle will preallocate sequence numbers in memory (default 20)

25th Jan. 2022
Tuesday.

To view created sequence,
select * from USER_SEQUENCES;

- To use sequence during data entry, two keywords are used:
  - NEXTVAL - ~~retrieves~~ retrieves next available number.
  - CURRVAL - retrieves the current value of sequence
- To insert data in a table using sequence, following syntax is used,

```
INSERT INTO tablename
VALUES (sequences_name.NEXTVAL,
        'value1', 'value2', 'valuen');
```

e.g.,
Create a sequence demo_seq that starts with 101 and increment by 2. Insert data into demo table using this sequence.

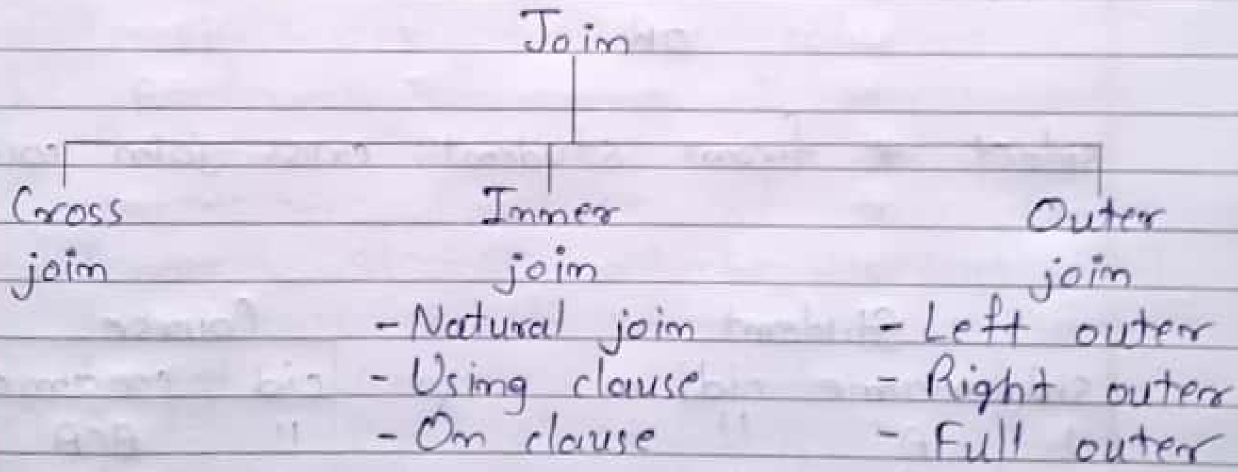
```
Create table demo(
        did number(3),
      dname varchar2(10));


insert into demo values (demo_seq.
                    nextval, 'A');
```

31st Jan. 2024
Monday.

* SQL joint operator.

- The relational join operator merges rows from two tables and returns the row with one of the following conditions:
  - Have common values in common column [Natural join]
  - Meet a given join condition [Inner join]
  - Have common values in common columns or have no matching values [Outer join]

→ Types of join.



Definition
- The inner join is a traditional join in which only rows that meet a given criteria are selected.
- An outer join returns not only the matching rows but the rows with unmatched value from both tables.

1. Cross join

- A cross join performs relational PRODUCT [Cartesian PRODUCT] of two tables.

Syntax:-

```
SELECT columnlist FROM table1, table2;

                OR

SELECT columnlist FROM table1 CROSS
       JOIN table2;
```

e.g.,

```
select * from student, course;

                OR

select * from student cross join course;
```

Student

sid	name	cid
1	A	11
2	B	11
3	C	12

Course

cid	cname
11	BCA
12	BCOM
13	BBA

sid	name	student.cid	course.cid	cname
1	A	11	11	BCA
1	A	11	12	BCOM
1	A	11	13	BBA
2	B	11	11	BCA
2	B	11	12	BCOM
2	B	11	13	BBA
3	C	12	11	BCA
3	C	12	12	BCOM
3	C	12	13	BBA

2. Natural join

- A natural join returns all rows with matching values in the matching columns and eliminates duplicate columns.
- It performs following 3 steps:

1) Perform PRODUCT which displays cartesian PRODUCT of two tables.
2) Perform SELECT which displays only the rows with common values in the common column.
3) Perform PROJECT which eliminates duplicate column.

Syntax:-

```
SELECT columnlist FROM table1
    NATURAL JOIN table2;
```

e.g.,
select * from student natural join course;

Student

sid	name	cid
1	A	11
2	B	11
3	C	12

Course

cid	cname
11	BCA
12	BCOM
13	BBA

Step-I: Perform PRODUCT.

sid	name	student. cid	course. cid	cname
1	A	11	11	BCA
1	A	11	12	Bcom
1	A	11	13	BBA
2	B	11	11	BCA
2	B	11	12	Bcom
2	B	11	13	BBA
3	C	12	11	BCA
3	C	12	12	Bcom
3	C	12	13	BBA

Step-2: Perform SELECT.

sid	name	student. cid	course. cid	cname
1	A	11	11	BCA
2	B	11	11	BCA
3	C	12	12	Bcom

Step-3: Perform PROJECT.

sid	name	cid	course
1	A	11	BCA
2	B	11	BCA
3	C	12	Bcom

3. Join USING CLAUSE.

- It returns only the rows with matching values in the matching column.

Syntax:-

```
SELECT columnlist FROM table1 JOIN
 table2 USING (common-column);
```

e.g.,

```
select * from student join course
 using (cid);
```

Note:- Join USING CLAUSE returns same output as natural join.

4. Join ON CLAUSE.

- It returns only the rows that meet the specified join condition.
- It is same as old style join.
- It includes equality comparition expression of two columns.

Syntax:-

```
SELECT columnlist FROM table1 JOIN
  table2 ON table1.column = table2.column;
```

e.g.,

```
select * from student join course on
    student.cid = course.cid;
```

sid	name	student. cid	course. cid	cname
1	A	11	11	BCA
2	B	11	11	BCA
3	C	12	12	Bcom

Ex. Display sid, name and course name of student.

```
select sid, name, cname from student, course
where student.cid = course.cid;
```

Ex. List sname, marks and cname of male student.

```
select name, marks, cname from
 student, course
where gender = 'm' and student.cid =
                        course.cid;
```

Ex. Display sid, name, marks, dob, cname of students who born in April month.

5. Left outer join

- The left outer join displays all the rows of first table and matching as well as not matching values of the another table.

Syntax:-

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

e.g.,

```
select * from student left outer join
course on student. cid = course. cid;
```

sid	name	cid	cname
1	A	11	BCA
2	B	11	BCA
3	C		

6. Right outer join

- The right outer join displays all the rows of second table including rows that do not have matching value in the first table.

Syntax:-

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

e.g.,

```
select * from student right outer
join course on student.cid = course.cid;
```

sid	name	cid	cname
1	A	11	BCA
		12	BCOM
		13	BBA
2	B	11	BCA

7. Full outer join

- The full outer join returns matching and not matching values from both tables.
- It is combination of left and right outer join.

Syntax:-

SELECT columnlist FROM table1 ~~full~~ FULL
OUTER JOIN table2 ON
  table1.column = table2.column;

e.g.,

```
select * from student full outer join
course on student.cid = course.cid;
```

sid	name	cid	cname
1	A	11	BCA
		12	BCOM
		13	BBA
2	B	11	BCA
3	C		

1st Feb. 2022
Tuesday.

* Sub queries and co-related queries.

1. WHERE sub query.

- The most common type of subquery uses the inner SELECT sub query on the right side of WHERE expression

Syntax:-
SELECT columnname FROM tablename
WHERE (subquery);

e.g.,
Display sid, sname, dob of students whose marks are greater than the average marks.

select sid, name, dob from student
where marks > (select avg(marks) from
student);

2. IN sub query.

- If we want to display multiple values based on the specified value from a table them the special operator IN is used.

Symtax:-

```
SELECT colummmame FROM tablemame
WHERE colummmame IN ('value1', 'value2',...);
```

e.g.,

List students whose mame either start with letter A or emds with letter A using subquery.

```
select * from student
where name like 'A%' or name like
        '%a';
```

(or)

```
select * from student
where name im (select mame from
   student where mame like 'A%' or
     mame like 'a%');
```

3. Multi row sub query operators - ANY, ALL.

- The use of ALL operator allows to compare a single value with a list of values return by first sub query using a comparisom operator other tham equal.

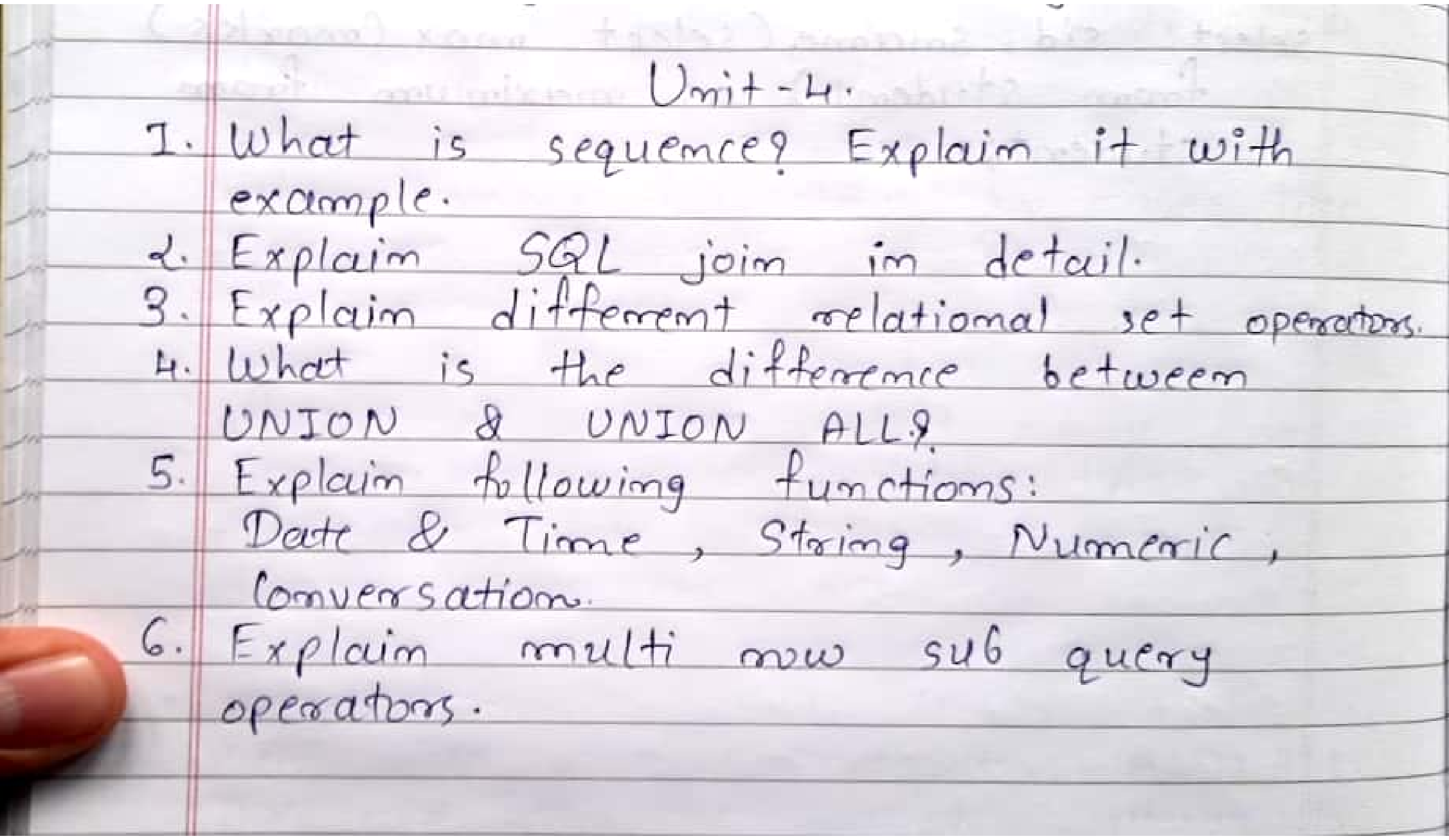
- The ANY operator allows user to compare a single value to a list of values and select only the rows which fulfill the given condition

4. Attribute list sub queries.

- The attribute list can include a sub query expression which is called inline sub query.
- A sub query in the attribute list must return one value.

e.g.,

```
select sid, sname, (select max (marks)
     from student) as maximum from
     student;
```



Unit-4.

1. What is sequence? Explain it with example.
2. Explain SQL join in detail.
3. Explain different relational set operators.
4. What is the difference between UNION & UNION ALL?
5. Explain following functions:
   Date & Time, String, Numeric, Conversation.
6. Explain multi row sub query operators.