Error Details:

Not enough values: given columns is not matching with values given recheck it.

DAY 1 video notes:

SQL developer tool to connect oracle.

SQL is not case sensitive #we can write query in both small or capital letters

We can call query as sql statement

A query should have two important keywords

1.SELECT 2. FROM

SELECT/FROM/WHERE #these are clauses

Syntax:

Select column\_name from table\_name;

\*astersick is a oracle operator which denotes all\_columns;

Eg;

To take a single column from a table

Use: select names from Employee; here names is one of the column in Employee

table

To show more than one column in a table use , to divide

Use: select names,salary from Employee;

Syntax:

To select a particular row or value or data in the table use where condition

Select column\_name from table\_name where condition;

Use: select names,salary from Employee where salary>10000;

SQL sometimes act a case sensitive to fetch the data

Eg:8

A table named Employee with two columns named names and salary

Here names are characters which may be capital letter or small letter

But the values cannot have any case sensitive case.

You have to fetch data from Naveen which is in the table

Select \* from Employee where names=’Naveen’;

Here Naveen first letter is capital so it will fetch correctly

When you give naveen instead of Naveen it will not fetch data because sql act as case sensitive while accessing the data.

Need to mention in a single quote for case sensitive

Eg: select \* from employee where names=’Naveen’;

CHARACTER/DATE/SPCL CHAR – these consider under case sensitive

OPERATOR:

We can write a multiple condition to a query for that we need logical operator.

Logical operator:

1.AND Operator- it must satisfy all the given conditions

Eg:

Select names,salary from Employee where first\_name=’naveen’ and salary>1100;

Here AND operator acts like both the conditions should satisfy

2.OR Operator- neither any condition given

Select names,salary from Employee where first\_name=’naveen’ or salary=300;

RELATIONAL OPERATOR:

= Equal too

> greater than

<smaller than

>=greater than equal too

<=smaller than equal too

!= (or) <> not equal too

in #operator which selects multiple data in table

not in #operator which deselects multiple data in table

between #show the range or limit

not between #does not show the range or limit

like #used to display first or last letter to fetch data

not like #not used to display first or last letter to fetch data

% shows the remaining letters

\_ #underscore matches only one letter

= equal too which matches only one value

is null #display the value which is null # undefined/unassigned/unknown value

is not null #display the value which is not null # undefined/unassigned/unknown value

Eg: we cannot give like this

Select \* from Employee where names=’naveen’,’praveen’,’satvik’;

Here = is relational operator which matches only one value.

We can use

Select \* from Employee where names in (’naveen’,’praveen’,’satvik’);

To show the data that is without the names given, we can use

Select \* from Employee where names not in (’naveen’,’praveen’,’satvik’);

Eg for between/not between:

Select \* from Employee where salary between 1000 and 2000;

This displays the data from salaried names with salary between 1000 and 2000.

Select \* from Employee where salary not between 1000 and 2000;

This displays the data from salaries names w65ithout salary between 1000 and 2000.

Like:

Select \* from Employee where first\_name like ‘a%’;

This would display the names that are starting with character ‘a’

% shows the remaining letters

To show the data of last letters

Use this:

Select \* from Employee where first\_name like ‘%n’;

This will display the names that are ending with character ‘n’

To display the exact third word starting letter

Use this:

Select \* from Employee where first\_name like ‘\_\_n%’; #underscore shows the position of the character.

Null: #display the value which is null # undefined/unassigned/unknown value

null<>null

null<>-

null<>0

null<>blank

to find a value which is null we use ‘is null’

Use this:

Select first\_name,details from Employee where details is null;

CONCATENATED OPERATOR:

Used to join two columns

For Eg:

You need to joint first\_name and last\_name in a single column

Use this:

Select first\_name||last\_name from Employee;

#it will merge both the columns but no space in between

Use this:

Select first\_name||’ ’||last\_name from Employee;

Single quoted in between shows the space so it will give the space

What and all you give in between the single quoted it will display

ALIAS ‘as’

#to change the column name we use as

Use this:

Select first\_name,last\_name as names from Employee;

Here names will be displayes instead of first\_name and last\_name.

DAY 2 Video notes:

Group by functions:

Max

Min

Count

Sum

Avg

Group by functions are single row function shows only result in one row.

Max(arg) function:

Shows the maximum value of given column

Use this: shows only rows format

select max(salary) from employees;

min(arg) function:

Shows the minimum value of given column.

Use this:

select min(salary) from employees;

count(arg) function:

count function counts the value in a numeric format

use this:

select count(salary) from employees;

sum(arg) function:

Add the total number of values in the column and shows in a row with the value

Use this:

Select sum(salary) from employees;

Avg(arg) function:

Shows in a average format like total values divide by number of counts.

Use this:

Select avg(salary) from employees;

Group function with empty column:

Group by: clause

We cannot get data when group by function with empty column

Eg:

Select first\_name,max(salary) from employees;

Here first\_name is a empty column shows all the columns in the database,but max(salary) is a group by function shows only one column.

For that we are using ‘group by’ - clause.

Use this:

Select first\_name,max(salary) from employees group by first\_name;

Having clause:

Having clause is used when group function needs a condition.

For normal column we can use where clause for condition.

Select employee\_id,max(salary) from employees group by employee\_id having max(salary)>10000;

1.Strings, number data type works only on max, min, count and sum, avg only works on numbers.

2.’Condition’ with group function need to mention in ‘having clause’.

3.Only ‘one argument’ is possible in ‘group function’.

4.\* asterisk only allowed on ‘count function’.

E.g., for \*asterisk:

ANALYTICAL FUNCTION: order by clause

Rank

Dense\_rank

Row\_number

Lead

Lag

Listagg

Rank:

Rank is one of the analytical functions which comes with order by clause, and a keyword over.

Syntax:

Rank()over(order by sort\_expression (asc|desc))

Order by is a clause

Select employee\_id,salary, rank()over(order by salary desc) as rank from employees;

It will show a table with employee\_id,salary and rank columns resulting with giving rank position based on their highest salary starting.

Dense\_rank:

Dense\_rank is almost same as rank function but to achieve consecutive ranking dense\_rank is one of the best methods to do so….

Select employee\_id,salary dense\_rank()over(order by salary desc) from employees;

Row\_number:

select employee\_id,row\_number()over(order by employee\_id asc) as sno from employees;

Row\_number is used to align the column in row manner by giving the unique number.

Lead:

select first\_name,lead(first\_name,1,'aaa')over(order by first\_name asc) as updated\_lead\_name from employees;

The lead function returns the value of a column from the next row in the window.

Lag:

The lag function returns the value of a column from the previous row in the window.

select employee\_id as ID,lag(employee\_id,1,999)over(order by employee\_id asc) as updated\_lag from employees;

subtracting to columns:

select first\_name,salary,lead(salary,1,100)over(order by salary desc) as deduction,salary-lead(salary,1,100)over(order by salary desc) as hand\_on\_salary from employees;

listagg:

This is used to concatenate values from multiple rows into a single string.It is particularly useful when you want to combine values from a column across multiple rows into a comma-separated list or any other delimiter-separated list.

Select listagg(first\_name,’-’)within group(order by first\_name asc) from employees;

DAY 3 Video notes:

Table creation:

To create a table in oracle database it should have 3 needs:

1.table structure

2.data type

3.query table creation

Table structure:

It is nothing but table name, column name

Create table table\_name(column\_name data\_type(length));

Data type:

Data type is a kind of data that a oracle understands.

Some of the data types are:

Number Numbers(0-9)

Char Alphanumeric(0-9,A-Z,!@#$%^&\* spcl characters)

Varchar Alphanumeric(0-9,A-Z,!@#$%^&\* spcl characters)

Long Alphanumeric(0-9,A-Z,!@#$%^&\* spcl characters)

Clob Alphanumeric(0-9,A-Z,!@#$%^&\* spcl characters)

Blob Alphanumeric(0-9,A-Z,!@#$%^&\* spcl characters)

Date Date

Timestamp Date+time

Xml type XML

Difference between char() and varchar():

Char takes along with space, but varchar takes only the space allotted.

See this, after creating the table:

Here name with char(10) and varchar(10)

Char(10)

Name total\_space taken free\_space\_taken shown

Java 10 4 6 10

C 10 1 9 10

Varchar(10)

Name total\_space taken free\_space\_taken shown

Java 10 4 0 4

C 10 1 0 1

For, varchar no space is taken

Query for creating the table:

Create table table\_name(column\_name data\_type(length));

DDL- Data definition language:

* Create
* Rename
* Truncate
* Alter
* Drop

DDL is auto commit which mean auto save no need to save the query personally

Create:

Create table student\_details(SID number(5),SNAME varchar(20),CID number(10),ADDRESS varchar(40),GENDER varchar(6),EMAIL varchar(20),PH\_NO number(10),DOB varchar(20),DOJ varchar(10),RESUME varchar(10));

Rename:

Rename works in table name renames

Rename student\_details to stu\_details;

Truncate:

Truncate is used to delete the data in the table.

Truncate table stu\_details;

Alter:

Alter is used to add, drop, rename, modify the data type of the column table.

Alter table stu\_details add lap\_or\_pc varchar(20);

Alter table stu\_details drop lap\_or\_pc;

Alter with rename to change the name of the column

Alter table stu\_details rename column SID to ID;

Alter table stu\_details modify GENDER char(5);

Drop:

Drop is used the entire table

Drop table stu\_details;

DML-Data Manipulation Language

It is not auto committed which means it will not save automatically

User need to commit explicity

* Insert
* Update
* Delete
* Merge

INSERT:

Insert is used to insert the values in the table,we can two method to insert the values

Insert into stu\_details(id,sname) values(5,’naveen’);

(or)

Insert into stu\_details values(5,’naveen’);

Inserting multiple values:

Insert all into stu\_details values(1,’pravin’)

into stu\_details values(2,nivin’)

into stu\_details values(3,’kelvin’)

select \* from dual;

Update:

Update is used to change the valued in the row

Update stu\_details set cname=’c++’ where id=2;

If you didn’t use where condition than the whole column will update as given, always try to use a where condition to avoid those mistakes.

Delete:

Same as update delete is used to delete a row in a table

Delete from stu\_details where id=6;

If you didn’t use where condition than the whole column will update as given, always try to use a where condition to avoid those mistakes.

DAY 4 Video notes:

DDL-Data definition language

DDL are table and column level changes and it is auto commit no need to save oracle itself save the changes.

* Create
* Truncate
* Alter
* Rename
* Drop

DML-Data manipulation Language

DML it is not auto commit user need to implicit the changes

* Insert
* Update
* Delete
* Merge

DML-Merge

Merge-Update+insert which is also called upsert

Merging normally known as merging to tables,but in sql it is different

Syntax for merging:

Merge into target\_table

Using source\_table

On condition

When matched then

Update

When matched then

Insert;

For eg:

Take two table named x1 and x2;

X1 table X2 table

A B A B

1 A 1 X

2 B 2 Y

3 C 3 Z

4 D

In merging one table should be a target table which is data that we put on table is target\_table

Another one table is source\_table which is the used to merge with target\_table.

Merging operation using syntax:

Merge into x2 #here x2 is target table

Using x1 #here x1 is source table

On x1.A=x2.A #when x1.A value is matched with x2.A then go to next query

When matched then #if both the valued are matched

Update set x2.B=x2.A #put the valued of x2.A into x2.B

When not matched then #if both the values are not matched

Insert (x2.A,x2.B) values (x1.A,x1.B); #puth the valued of x1 to x2

Output:

Select \* from x2;

X2 table

A B

1 A

2 B

3 C

4 D

TCL-Transaction control language

This language is unable when DML is used

* Commit-used to change the status from pending to permanent
* Rollback-used to reverse the changes like ctrl+Z/discart the pending changes.

If any DML operation is done TCL need to be there.

Eg for understanding:

create table dummy(sno number(10),Mobile\_name varchar(40),Price number(20));

rename dummy\_details to mobile\_details;

desc mobile\_details;

drop table mobile\_details;

insert into mobile\_details values(1,'Apple',80000);

insert into mobile\_details values(2,'Samsung',60000);

insert into mobile\_details values(3,'Redmi',40000);

insert into mobile\_details values(4,'Blackberry',30000);

commit; # temporary to permanent

delete from mobile\_details where sno=4;

rollback; #reverse the deletion like ctrl+Z

select \* from mobile\_details;

DAY 5 Video notes:

JOINS:

When we need to get/fetch data from two or more table at a same time joins are used.

Joins are of four types:

* Inner join
* Outer join

1. Outer left join
2. Outer right join
3. Full outer join

* Cross join
* Self join

Inner join: in oracle file saved as ‘joins-inner join’

Inner join is used to get the data only with related value/records.

Syntax is same as normal query.

Select column\_name from table\_name where condition;

Inner join with two table : table 1 name: students & table2 name: courses

Students has three columns sno,course\_id,stu\_name and

Course has two columns course\_id and course\_name

Way 1:

select students.stu\_name,course.course\_name from students,course where students.course\_id=course.course\_id; #only matched data shown

Way 2:

Select s.stu\_name,c.course\_name from students s,course c where s.course\_id=c.course\_id;

Way 3:

Select s.stu\_name,c.course\_name from students s inner join course c on s.course\_id=c.course\_id;

Outer join:

Outer join are of three types:

* Left outer join
* Right outer join
* Full outer join

Left outer join:

It shows the data for both matched records and unmatched left table records.

Here, we have two table named students and course,

Students has three columns sno,course\_id and stu\_name and

Course has two columns course\_id and course\_name

Way 1(implicit method):

select students.stu\_name,course.course\_name from students,course where students.course\_id=course.course\_id(+); #Need to indicate + sign for right side

Way 2(ansi method):

select s.stu\_name,c.course\_name from students s left outer join course c on s.course\_id=c.course\_id;

Right Outer join:

It shows matched records and unmatched right table record.

Way1:

Select s.stu\_name,c.course\_name from students s, course c where s.course\_id(+)=c.course\_id;

Way2:

Select stu\_name,course\_name from students s right outer join course c on s.course\_id=c.course\_id;

Full outer join:

Shows the data for both matched and unmatched records for both the table.

Way1:

Select s.stu\_name,c.course\_name from students s full outer join course c on s.course\_id=c.course\_id;

Error way don’t use this way:

Select s.stu\_name,c.course\_name from students s, course c where s.course\_id(+)=c.course\_id(+); #there is no special used in both ways.

Cross join:

When the user ignores the where clause the cartesian product executes.

Cartesian product is nothing but number of rows in the first\_table is multiplied by number of rows in the second\_table.

For eg:

You have 3 rows in first\_table and 5 rows in second\_table then it will display

3 \* 5 = 15 total of 15 rows.

Self join:

The join that join itself is named as self join.

Hint:

Formula condition=table-1 or C=T-1,based on if we use two table for data then,

C=2-1=1 one condition mandatory

Taken 4 table; C=4-1=3 condition mandatory.

For eg:

You have a table named employees contains more columns.

Check fir employee\_id,first\_name,manager\_id

|  |  |  |
| --- | --- | --- |
| **EMPLOYEE\_ID** | **FIRST\_NAME** | **MANAGER\_ID** |
| 100 | Steven | - |
| 101 | Neena | 100 |
| 102 | Lex | 100 |
| 103 | Alexander | 102 |
| 104 | Bruce | 103 |
| 105 | David | 103 |
| 106 | Valli | 103 |
| 107 | Diana | 103 |
| 108 | Nancy | 101 |
| 109 | Daniel | 108 |
| 110 | John | 108 |
| 111 | Ismael | 108 |

|  |  |  |
| --- | --- | --- |
| **EMPLOYEE\_ID** | **FIRST\_NAME** | **MANAGER\_ID** |
| 100 | Steven | - |
| 101 | Neena | 100 |
| 102 | Lex | 100 |
| 103 | Alexander | 102 |
| 104 | Bruce | 103 |
| 105 | David | 103 |
| 106 | Valli | 103 |
| 107 | Diana | 103 |
| 108 | Nancy | 101 |
| 109 | Daniel | 108 |
| 110 | John | 108 |
| 111 | Ismael | 108 |

We need to get an output like this:

|  |  |
| --- | --- |
| **EMPLOYEE\_NAME** | **MANAGER\_NAME** |
| Neena | Steven |
| Lex | Steven |
| Den | Steven |
| Matthew | Steven |
| Adam | Steven |
| Payam | Steven |
| Shanta | Steven |

First\_name from one table and same first\_name from another table using employee\_id format.

Use this query:

select e1.first\_name as Employee\_name,e2.first\_name as Manager\_name

from employees e1,employees e2

where e1.manager\_id=e2.employee\_id

and e1.manager\_id=100;

ASSIGNMENT:

Link three columns in three different tables using joins.

Table 1: Employee\_details:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **SNO** | **FIRST\_NAME** | **PH\_NO** | **SALARY** | **DEPARTMENT\_ID** | **LOCATION\_ID** |
| 1 | Naveen | 55665566 | 10000 | 30 | 21 |
| 2 | Kavin | 23568941 | 20000 | 50 | 23 |
| 3 | Pravin | 47826359 | 42000 | 60 | 22 |
| 4 | Christian\_Bale | 21369547 | 2500 | 100 | 25 |
| 5 | Chris\_Evans | 44765933 | 3500 | 90 | 26 |
| 6 | Robert\_Downey | 33695874 | 8000 | 60 | 24 |
| 7 | Gal\_Gadot | 69873305 | 2000 | 90 | 27 |
| 8 | Tom\_Holland | 10105689 | 4500 | 50 | 26 |

Table 2: Department:

|  |  |  |  |
| --- | --- | --- | --- |
| **SNO** | **DEPARTMENT\_ID** | **DEPARTMENT\_NAME** | **LOCATION\_ID** |
| 1 | 30 | IT | 21 |
| 2 | 50 | BPO | 22 |
| 3 | 60 | NETWORKING | 23 |
| 4 | 90 | UI/UX\_DESIGNER | 24 |
| 5 | 100 | MARKETING | 25 |

Table 3: Location:

|  |  |  |
| --- | --- | --- |
| **SNO** | **LOCATION\_ID** | **CITY** |
| 1 | 21 | INDIA |
| 2 | 22 | US |
| 3 | 23 | UK |
| 4 | 24 | SINGAPORE |
| 5 | 25 | MALAYSIA |
| 6 | 26 | FINLAND |
| 7 | 27 | SWEDEN |

We need to fetch first\_name,department\_name,city from these table using joins.

Using parent-child relationship:

Select e.First\_name,d.department\_name,l.city

From employee\_details e,department d,location l

Where e.department\_id=d.department\_id

And d.location\_id=l.location\_id;

Using joins:

select e.first\_name,d.department\_name,l.city

from employee\_details e inner join department d

on e.department\_id=d.department\_id inner join location l

on d.location\_id=l.location\_id;

Output:

|  |  |  |
| --- | --- | --- |
| **FIRST\_NAME** | **DEPARTMENT\_NAME** | **CITY** |
| Naveen | IT | INDIA |
| Kavin | BPO | US |
| Pravin | NETWORKING | UK |
| Christian\_Bale | MARKETING | MALAYSIA |
| Chris\_Evans | UI/UX\_DESIGNER | SINGAPORE |
| Robert\_Downey | NETWORKING | UK |
| Gal\_Gadot | UI/UX\_DESIGNER | SINGAPORE |
| Tom\_Holland | BPO | US |

ASSIGNMENT:

Table1 with values inserted as:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

|  |
| --- |
| **ONE** |
| 1 |
| 0 |
| 1 |
| 0 |

Table 2 with values inserted as:

|  |
| --- |
| **ALL\_ONE** |
| 1 |
| 1 |
| 1 |
| 1 |

Need to use all types of joins to get the result:

Inner join

Query:

Inner join:

select t1.one,t2.all\_one

from table1 t1,table2 t2

where t1.one=t2.all\_one;

select table1.one,table2.all\_one

from table1,table2

where table1.one=table2.all\_one;

select t1.one,t2.all\_one

from table1 t1 inner join table2 t2

on t1.one=t2.all\_one;

Output:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

|  |  |
| --- | --- |
| **ONE** | **ALL\_ONE** |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |

Outer Join:

Left outer Join:

select t1.one,t2.all\_one

from table1 t1,table2 t2

where t1.one=t2.all\_one(+);

select t1.one,t2.all\_one

from table1 t1 left outer join table2 t2

on t1.one=t2.all\_one;

Output:

|  |  |
| --- | --- |
| **ONE** | **ALL\_ONE** |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 0 | - |
| 0 | - |

Right Outer Join:

select t1.one,t2.all\_one

from table1 t1,table2 t2

where t1.one(+)=t2.all\_one;

select t1.one,t2.all\_one

from table1 t1 right outer join table2 t2

on t1.one=t2.all\_one;

Output:

|  |  |
| --- | --- |
| **ONE** | **ALL\_ONE** |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |

Full outer Join:

select t1.one,t2.all\_one

from table1 t1 full outer join table2 t2

on t1.one=t2.all\_one;

output:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

|  |  |
| --- | --- |
| **ONE** | **ALL\_ONE** |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 0 | - |
| 0 | - |

cross join:

select t1.one,t2.all\_one

from table1 t1,table2 t2;

|  |  |
| --- | --- |
| **ONE** | **ALL\_ONE** |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 0 | 1 |
| 0 | 1 |
| 0 | 1 |
| 0 | 1 |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 0 | 1 |
| 0 | 1 |
| 0 | 1 |
| 0 | 1 |

DAY 6 Video notes:

SET OPERATOR:

* To obtain data by using two or more queries is to be were set operator used.
* It gets executed based on first query.
* Set operator is used in between the queries.

Types of set operator:

UNION, UNION ALL, INTERSECT, MINUS.

Union operator:

It will eliminate the duplicate.

By default, it will place in an ascending order.

t1 t2

|  |
| --- |
| **A** |
| 1 |
| 2 |
| 3 |
| 5 |

|  |
| --- |
| **A** |
| 1 |
| 2 |
| 3 |
| 4 |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

Here, we have two tables t1, t2 where union operator eliminates the duplicate.

Use this:

Select \* from t1

UNION

Select \* from t2;

|  |
| --- |
| **A** |
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |

Output:

ro

Union all;

It does not eliminate duplicate

Shows the value as it is in the table

Eg:

t1 t2

|  |
| --- |
| **A** |
| 1 |
| 2 |
| 3 |
| 5 |

|  |
| --- |
| **A** |
| 1 |
| 2 |
| 3 |
| 4 |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

Here, we have two tables t1, t2

Use this:

Select \* from t1

Union all

Select \* from t2;

|  |
| --- |
| **A** |
| 1 |
| 2 |
| 3 |
| 5 |
| 1 |
| 2 |
| 3 |
| 4 |

Output:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

Why the table values are one by one because it will execute based on first query.

Intersect:

It will fetch only common data.

Eg:

t1 t2

|  |
| --- |
| **A** |
| 1 |
| 2 |
| 3 |
| 5 |

|  |
| --- |
| **A** |
| 1 |
| 2 |
| 3 |
| 4 |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

Here, we have two tables t1, t2

We can find that there are some values are common for both the tables.

|  |
| --- |
| **A** |
| 1 |
| 2 |
| 3 |

Output:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

Here, 1,2,3 are common values on both tables.

Minus:

Minus is opposite to intersect

It will show data only for uncommon values.

Eg:

t1 t2

|  |
| --- |
| **A** |
| 1 |
| 2 |
| 3 |
| 5 |

|  |
| --- |
| **A** |
| 1 |
| 2 |
| 3 |
| 4 |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

Here, we have two tables t1, t2

We can see that there are some unmatching values found on both the tables.

|  |
| --- |
| **A** |
| 5 |

Output:

We can see that 1,2,3 are common values and 5 is uncommon for t2 and also 4 is uncommon for t1,

But it will display only 5, because set operator works based on first query.

Combines set operators:

Select \* from t1

Union

Select \* from t2 #here union for t1 & t2 will show their output

Union all

Select \* from t1 #output for the second query is union all with the t2

Intersect

Select \* from t2

Minus

Select \* from t1;

Output:

t1 t2

|  |
| --- |
| **A** |
| 1 |
| 2 |
| 3 |
| 5 |

|  |
| --- |
| **A** |
| 1 |
| 2 |
| 3 |
| 4 |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

|  |
| --- |
| **A** |
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |

|  |
| --- |
| **A** |
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
| 1 |
| 2 |
| 3 |
| 5 |

Union of t1 and t2 is and this output is union all with t1 this output intersects with t2

|  |
| --- |
| **A** |
| 1 |
| 2 |
| 3 |
| 4 |

|  |
| --- |
| **A** |
| 4 |

Give and then minus with t1 gives

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

Set operator not used cases:

1.When both the data type of values differs or expression must have same data type.

Eg:

Select 5 from t1

Union

Select ‘A’ from t2;

Here there are 2 different data type is used 5 number and ‘A’ character

2.When columns mismatches.

Eg:

Select a,b,c from t1

Union all

Select a,b from t2;

Here, in t1 we have three columns named a,b,c and table t2 has two columns a,b

Both the column numbers were mismatched so in this scenario set operator will not work.

PSEUDO TABLES AND COLUMNS:

Oracle has its own tables and columns are known as pseudo tables and columns.

Dual is one of the oracle’s pre-defined tables

Some of the oracle’s predefined columns are mentioned below:

Sysdate:

* Displays the current date

Query: select sysdate from dual;

Output: Shows the current date

Systimestamp:

* Displays the current date with time

Query: select systimestamp from dual;

Output: shows the current date and time

User:

* Displays the current user id login name

Query: select user from dual;

Uid:

Also called a userid

* Displays the user id’s a unique number

Query: select uid from dual;

Output: shows a unique number of current login user id

Rownum:

* Displays the rownumber colums

Query: select first\_name,rownum from employee\_details;

Output: displays the rownumber wise numbers in a column

Rowid:

* Displays a 16-digit unique characters to a row\_number

Query: select first\_name,rownum,rowid from employee\_details;

Output: displays a sequence of unique number to the rownum.

SEQUENCE:

* It is a database object
* It is used to generate sequence of numbers
* It only works for number data type
* It is used in primary key columns.

Some of the database objects are tables, sequence

Every database object needs a ‘create’ clause.

Syntax:

Create sequence sequence\_name

Start with number

Increment by number

Maxvalue number;

There are two types of attributes in sequence:

* Nextval – shows the next value in the sequence
* Currval – shows the current value in the sequence

To view the nextval in the sequence:

Syntax:

Select sequence\_name.nextval from dual;

Every time you run the nextval it will return next value till max value.

Currval:

To view the currval In the sequence:

Syntax:

Select sequence\_name.currval from dual;

Every time you run the currval it will return only current value.

ALTER SEQUENCE:

We can also alter a sequence using

Syntax:

Alter sequence seq\_1

Increment by 2

Maxvalue 10;

But, were the previous sequence ends the alter sequence also starts from the same.

For example the previous sequence ends with 5 and now you altered the sequence and increment by 2 maxvalue 10; when you run select seq\_1.nextval from dual; it will show output as 7 because previous value ends with 5.

DROP SEQUENCE:

To drop or delete a sequence use this:

Syntax:

Drop sequence sequence\_name;

Drop sequence seq\_1;

DATA DICTIONARY TABLE:

Select \* from user\_sequences; - displays the total number of sequences in the database.

Select \* from user\_tables; - displays the total number of tables in the database.

DAY 7 Video notes:

CONSTRAINTS:

* Enforcing a rule to a table

Types of constraints:

* Primary key constraints
* Unique constraints
* Not null constraints
* Check constraints
* Foreign constraints

Primary key constraints:

* It will not allow duplicates and null values
* Only one primary key column table is allowed in a table

Unique constraints:

* It will allow only null values does not allow duplicates.

Not null constraints:

* It will allow only duplicates and does not allow null values.

Check constraints:

* It checks for the condition
* If condition satisfies 🡪 it allows

Foreign constraints:

* Referential integrity (refers the base table, base table means another table related to parental table, our parental table has cid and base table has cid and cname)
* It refers the column base table and base table column should be primary key or unique key
* It accepts both duplicate and null values.

Creating table using constraints:

CREATE TABLE t\_students (

Id number(10),

Name varchar(30) NOT NULL,

Gender char(5),

Email varchar(30),

Phno number(20),

cid number(10),

CONSTRAINT const\_id\_pk PRIMARY KEY (Id),

CONSTRAINT const\_email\_u UNIQUE (Email),

CONSTRAINT const\_phno\_chk CHECK (length(Phno)=10),

CONSTRAINT const\_Gender\_chk CHECK (Gender IN ('M', 'F')),

CONSTRAINT const\_cid\_fk FOREIGN KEY (cid) REFERENCES course1(cid)

);

Altering constraints in table:

Dropping constraint:

Alter table t\_students drop constraint const\_email\_u;

Adding constraint:

Alter table t\_students add constraint const\_email\_u unique(email);

DAY 8 Video notes:

SUBQUERY:

* A query that is embedded within the query is known as sub query
* Two parts of subquery
  + Inner query
  + Outer query

Types of subquery:

* Nested subquery
* Correleated subquery

For eg:

Need to find the max salary with their first\_name and dep\_id of the employee table

We will use like select max(salary) from employees; to find max salary

And select first\_name,salary,dep\_id from employees where salary=24000;

Here, we have used two queries to get a single data

For this sub query is mostly used

SINGLE ROW SUBQUERY:

* The data which only provides single row of data.

Use this:

Select first\_name,salary,dep\_id from employees where salary=(select max(salary) from employees);

MULTIPLE ROW SUBQUERY:

* If a subquery returns more than the single or multiple query it is names as multiple row subquery

There are three different ways to performs multiple subquery:

* In
* >any
* >all

For eg:

We need to get data from first\_name,salary,department\_id from a table that should be matched with a department\_id=60.

Select salary from employees where department\_id=60;

Output:

|  |
| --- |
| **SALARY** |
| 9000 |
| 6000 |
| 4800 |
| 4800 |
| 4200 |

Here it will display multiple rows of data.

Using ‘in’:

Select first\_name,salary,department\_id from empoyees where salary in (Select salary from employees where department\_id=60);

|  |  |  |
| --- | --- | --- |
| **FIRST\_NAME** | **SALARY** | **DEPARTMENT\_ID** |
| Allan | 9000 | 80 |
| Peter | 9000 | 80 |
| Daniel | 9000 | 100 |
| Alexander | 9000 | 60 |
| Pat | 6000 | 20 |
| Bruce | 6000 | 60 |
| Valli | 4800 | 60 |
| David | 4800 | 60 |
| Nandita | 4200 | 50 |
| Diana | 4200 | 60 |

Displays based on department\_id=60 salary, please compare it with department\_id=60 table.

Using ‘>any’:

Select first\_name,salary,department\_id from empoyees where salary >any (Select salary from employees where department\_id=60);

|  |  |  |
| --- | --- | --- |
| **FIRST\_NAME** | **SALARY** | **DEPARTMENT\_ID** |
| Steven | 24000 | 90 |
| Neena | 17000 | 90 |
| Lex | 17000 | 90 |
| John | 14000 | 80 |
| Karen | 13500 | 80 |
| Michael | 13000 | 20 |
| Nancy | 12000 | 100 |
| Alberto | 12000 | 80 |
| Shelley | 12000 | 110 |
| Lisa | 11500 | 80 |
| Ellen | 11000 | 80 |
| Gerald | 11000 | 80 |
| Den | 11000 | 30 |
| Eleni | 10500 | 80 |
| Clara | 10500 | 80 |
| Janette | 10000 | 80 |
| Peter | 10000 | 80 |
| Hermann | 10000 | 70 |
| Harrison | 10000 | 80 |
| Tayler | 9600 | 80 |
| Danielle | 9500 | 80 |
| David | 9500 | 80 |
| Patrick | 9500 | 80 |
| Peter | 9000 | 80 |
| Alexander | 9000 | 60 |
| Allan | 9000 | 80 |
| Daniel | 9000 | 100 |
| Alyssa | 8800 | 80 |
| Jonathon | 8600 | 80 |
| Jack | 8400 | 80 |
| William | 8300 | 110 |
| Adam | 8200 | 50 |
| John | 8200 | 100 |
| Matthew | 8000 | 50 |
| Lindsey | 8000 | 80 |
| Christopher | 8000 | 80 |
| Payam | 7900 | 50 |
| Jose Manuel | 7800 | 100 |
| Ismael | 7700 | 100 |
| Louise | 7500 | 80 |
| Nanette | 7500 | 80 |
| William | 7400 | 80 |
| Elizabeth | 7300 | 80 |
| Mattea | 7200 | 80 |
| Oliver | 7000 | 80 |
| Kimberely | 7000 | - |
| Sarath | 7000 | 80 |
| Luis | 6900 | 100 |
| David | 6800 | 80 |
| Susan | 6500 | 40 |
| Shanta | 6500 | 50 |
| Sundar | 6400 | 80 |
| Charles | 6200 | 80 |
| Amit | 6200 | 80 |
| Sundita | 6100 | 80 |
| Pat | 6000 | 20 |
| Bruce | 6000 | 60 |
| Kevin | 5800 | 50 |
| Valli | 4800 | 60 |
| David | 4800 | 60 |
| Jennifer | 4400 | 10 |

Display a table with least salary than department\_id=60 table.

Using ‘>all’:

Select first\_name,salary,department\_id from empoyees where salary >all (Select salary from employees where department\_id=60);

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

|  |  |  |
| --- | --- | --- |
| **FIRST\_NAME** | **SALARY** | **DEPARTMENT\_ID** |
| Steven | 24000 | 90 |
| Neena | 17000 | 90 |
| Lex | 17000 | 90 |
| Nancy | 12000 | 100 |
| Den | 11000 | 30 |
| John | 14000 | 80 |
| Karen | 13500 | 80 |
| Alberto | 12000 | 80 |
| Gerald | 11000 | 80 |
| Eleni | 10500 | 80 |
| Peter | 10000 | 80 |
| David | 9500 | 80 |
| Janette | 10000 | 80 |
| Patrick | 9500 | 80 |
| Clara | 10500 | 80 |
| Danielle | 9500 | 80 |
| Lisa | 11500 | 80 |
| Harrison | 10000 | 80 |
| Tayler | 9600 | 80 |
| Ellen | 11000 | 80 |
| Michael | 13000 | 20 |
| Hermann | 10000 | 70 |
| Shelley | 12000 | 110 |

Display a table with more salary than department\_id=60 table.

MULTIPLE COLUMNS SUBQUERY:

When an inner subquery contains more than one column we use multiple column subquery.

For eg:

Need a data which display first\_name,salary,department with max salary of every name according to department\_id.

If department\_id is 60 then max salary need to be shown, applies to every department\_id.

Output will be like:

|  |  |  |
| --- | --- | --- |
| **FIRST\_NAME** | **DEPARTMENT\_ID** | **SALARY** |
| Nancy | 100 | 12000 |
| Den | 30 | 11000 |
| Steven | 90 | 24000 |
| Michael | 20 | 13000 |
| Hermann | 70 | 10000 |
| Shelley | 110 | 12000 |
| Adam | 50 | 8200 |
| John | 80 | 14000 |
| Susan | 40 | 6500 |
| Alexander | 60 | 9000 |
| Jennifer | 10 | 4400 |

select department\_id,salary from employees; #displays both columns

select department\_id,max(salary) from employees group by department\_id;

#displays max salary according to department\_id

select first\_name,department\_id,salary from employees where (department\_id,salary) in (select department\_id,max(salary) from employees group by department\_id);

#displays first\_name,department\_id with max salary.

SCALAR SUBQUERY:

The subquery that is placed with in the select clause is termed as scalar subquery.

Query:

Select 1 from dual;

select 1,2+2,(select 3+4 from dual) from dual; #subquery in select clause

INLINE VIEW SUBQUERY:

* The subquery that placed in table\_name of the query.
* That subquery that is used in the from clause of a sql query.It is also known as derived tables.

Query:

Select first\_name,salary\_date,hire\_date from employees;

Select first\_name from (Select first\_name,salary\_date,hire\_date from employees);

#here the from clause subquery act as table we can able to fetch data.

NESTED SUBQUERY:

* Outer query depends on inner query

On top all subquery are knows nested subqueries.

As per BARD there are three main subqueries:

1. Scalar subquery
2. Multiple column subquery
3. Correlated subquery

CORRELATED SUBQUERY:

* The inners query depends the data of the outer query.

Eg: we need to display a table with first\_name,salary,department\_id,department\_name here first\_name,salary,department\_id are from employees table and department\_name is from department table.

Select first\_name,salary,department\_id from employees;

select first\_name,salary,(select department\_name from department d where d.department\_id=e.department\_id) as Department\_name from employees e;

for this subquery inner query will not run separately as it depends on data of outer query.

DAY 9 Video notes:

VIEWS:

* It is a database object
* Stores the query in the database
* It is a virtual table

For eg:

Here we have a query

Select first\_name from employees;

It will show output as only first\_name column instead of putting this query we can create a view to this query to run in need

Syntax:

Create or replace view view\_name

As

Needed\_query;

Create or replace view v1

As

Select first\_name from employees;

Here, we have created a view v1 to the query instead of running the first\_name query we can run v1 query because view stores the query in database.

To see the view: Select \* from v1;

When we run this, it will display the first\_name query of the employees table.

ADVANTAGES OF VIEWS:

* To restrict data access-it will not allow access to see other data
* Complex query into simple- more lines of query by creating a simple view table to run.

TYPES OF VIEWS:

* Simple views
* Complex views

SIMPLE VIEW:

* Only one table involves by getting the data than it is simple view.
* DML is possible

For eg:

Create or replace view v2

Select hire\_date from employees;

Here, only one table employees involves in getting the data so it is simple view.

COMPLEX VIEW:

* If my query consists of
* JOINS
* SUBQUERY
* SETOPERATOR
* PSEUDO COLUMNS
* DISTINCT/UNIQUE
* FUNCTIONS then they are complex view.
* DML is not possible in complex view

For eg:

Create or replace view v3

Select unique first\_name from employees;

It is a complex view as it contains unique.

select max(salary) from employees

Union

Select min(salary) from employees;

It is also complex view as it contains union a set operator.

DML IN SIMPLE VIEW:

A test table created for reference:

create table test(A number(5));

desc test;

insert into test values(1);

select \* from test;

create or replace view v1

as

select a from test;

select \* from v1;

insert into v1 values(2); #insert dml is possible in simple view

DML IN COMPLEX VIEW:

select a from v1;

create or replace view v1

as

select unique a from test; #unique comes as it is complex view

insert into v1 values(2); # insert not possible in complex view as it is a dml

Error throw as: **ORA-01732: data manipulation operation not legal on this view**

RESTRICT METHODS:

* With check option
* With read only

Restrict methods are some of the restrict methods used in view for data.

With check option:

With check option used to restrict the data which is given in the query.

create or replace view v1

as

select A from test where a in (1,2) #here column A has two rows 1,2

with check option;

insert into v1 values(3); # when we give values like 3 or any other values than 1,2 it will throw the error as **ORA-01402: view WITH CHECK OPTION where-clause violation**

With read only:

* With read only is used to view the table or read the table
* We cannot able to insert values in with read only query.

Query:

Create or replace view v1

As

Select a from test

With read only;

insert into v1 values(3); #no insertion will not done here as it read only it will throw error as **ORA-01733: virtual column not allowed here**

FORCE VIEW:

We can able to create view for a non-existing table.

View creates table for non-existing table but it will not execute.

For eg:

There a table name ‘tabu’ which is not existing in the database

Create or replace force view tv3

As

Select \* from tabu;

The view tv3 created but it will not execute.

DROPPING VIEW:

* To delete a view database we use drop method.

Drop view view\_name;

DAY 10 & 11 Video notes:

MATERIALIZED VIEW:

View is used to store query in the database

* Materialized view is used to store result of the query in the database.
* It is a database object
* It is like a snapshot 🡪takes the photo of the result
* DML is not possible
* Materialized view is faster than view.

For eg:

Select first\_name from employees where department\_id=60;

Output:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

|  |
| --- |
| **FIRST\_NAME** |
| Alexander |
| Bruce |
| David |
| Valli |
| Diana |

Syntax:

Create or replace materialized view materialized\_view\_name

As

The selected\_query;

Create or replace materialized view mv\_1

As

Select first\_name from employees where department\_id=60;

Here, the result with a table output is taken by mviews a screenshot.

When we run select \* from mv\_1 it shows the result of above table.

Difference between views and materialized view:

After creating view we can insert values to certain table and can be viewed after showing select \* from v1;

But, in materialized view we can insert values in certain table or query but cannot be viewed after running select \* form mv\_1;

REFRESH METHODS:

* Complete
* Fast -need dba permission to access this
* Force

These methods are used to refresh the materialized view to get the snap from recent insertion done or not, because materialized view doesn’t allow insertion.

COMPLETE METHOD:

The complete method is used to refresh the whole materialized view method and show the recent changes in the table.

Syntax:

BEGIN

DBMS\_MVIEW.REFRESH(mv\_name, method => 'C');

END;

Begin

Dbms\_mview.refresh(mv\_1,’c’)

End;

/ 🡺no need to mention this,it is used to terminate the sql statement.

FAST METHOD:

Fast method needs a log table to execute

Log table syntax:

Create materialized view log on table\_name;

Fast method syntax:

Begin

Dbms\_mview.refresh(mv\_name,method=>’F’)

End;

FORCE METHOD:

Force method knows oracle to see what type of refresh it is needed.

Syntax:

Begin

Dbms\_mview.refresh(mv\_name,method=>’?’)

End;