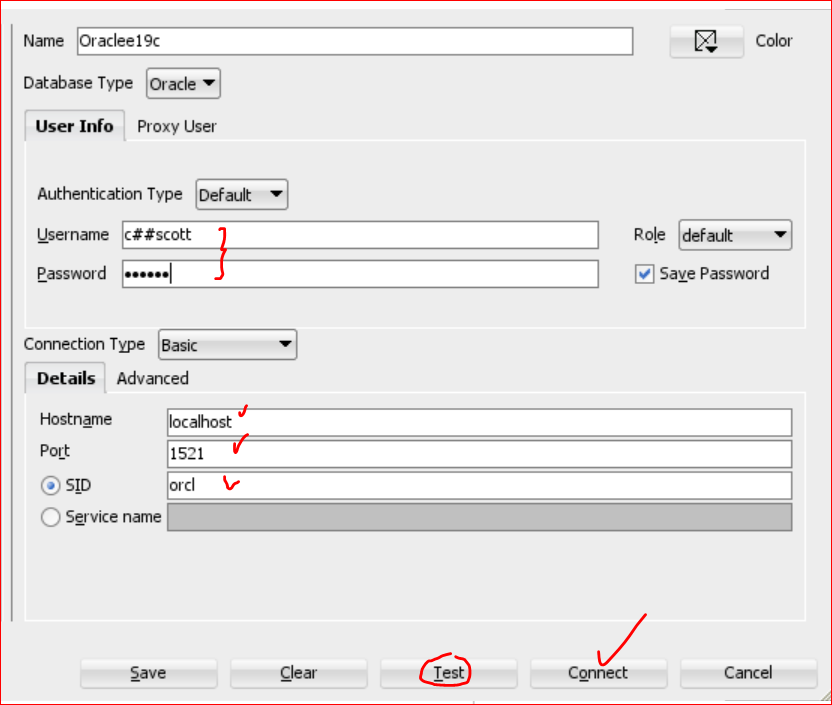
**1) how to connect to the Oracle Database?**

We have to provide the Database connection details and then we can able to connect like below are the details

* Server name
* Port Number
* Database Name
* Username
* Password



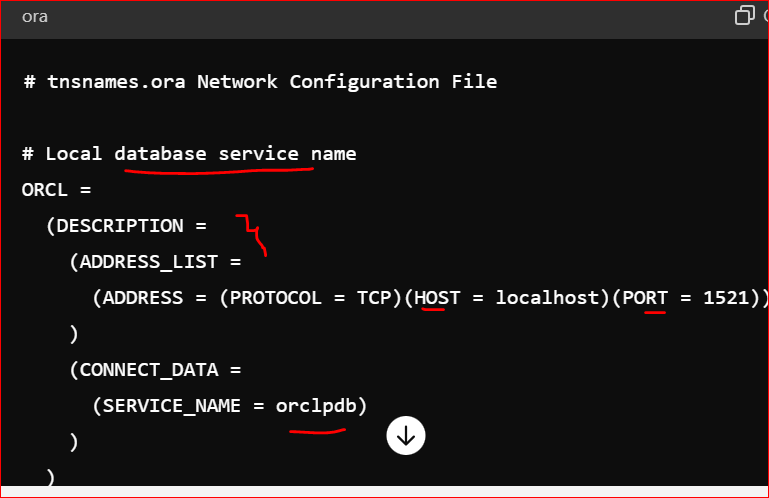
**2) what is ODBC.ini file or tnsnames.ora and what it contains?**

**tnsnames.ora** *is a configuration file used by Oracle clients to connect to Oracle databases.* It contains network service names and corresponding connect descriptors, which define how to connect to an Oracle database.

**Contents of tnsnames.ora**

The tnsnames.ora file is typically located in the $ORACLE\_HOME/network/admin directory on Unix/Linux systems or %ORACLE\_HOME%\network\admin on Windows systems. Here’s what the tnsnames.ora file contains:

1. **Network Service Name**: This is a user-defined alias that clients use to connect to the database. Each service name entry starts with the service name in square brackets.
2. **Connect Descriptor**: This section contains details about the database connection, including the protocol, host, port, and database service or system identifier (SID).



The **odbc.ini** file is a configuration file used by ODBC (Open Database Connectivity) drivers to define the data sources that applications can connect to. It contains information about the Data Source Names (DSNs) and their associated database connection details. This file is essential for applications that use ODBC to interact with databases, providing a standard way to define database connections.

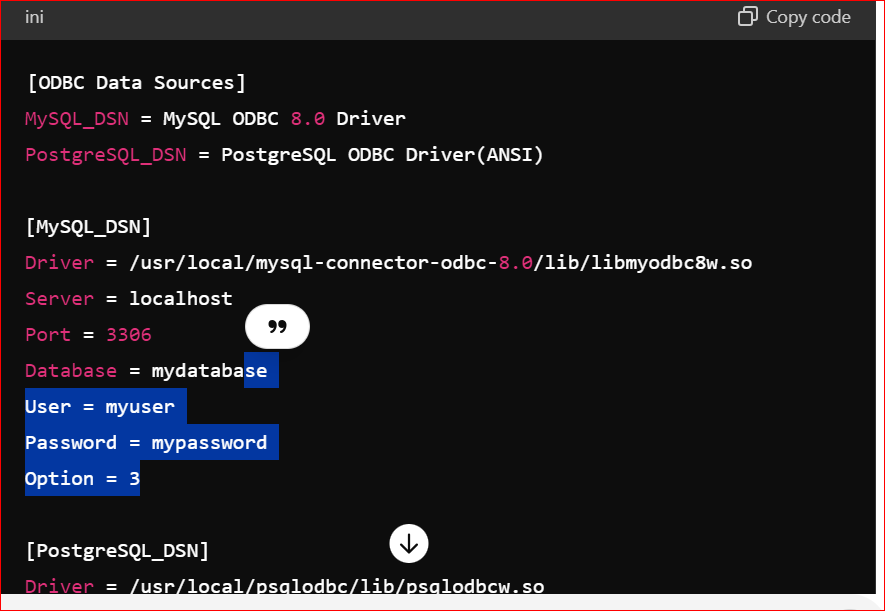
**Contents of odbc.ini**

The odbc.ini file typically contains sections for each DSN, along with the parameters needed to connect to the corresponding database. Here’s a breakdown of the key components:

1. **ODBC Data Sources Section**: Lists the DSNs and their descriptions.
2. **Data Source Sections**: Each DSN has its own section with specific connection details like driver, database name, server, user, and password.

**Example odbc.ini File**

Here’s an example of what an odbc.ini file might look like:



**Configuration Example for Linux**

On Linux systems, the odbc.ini file is typically located in one of the following locations:

* /etc/odbc.ini
* $HOME/.odbc.ini

To configure an ODBC DSN on a Linux system, you might follow these steps:

1. **Install the ODBC driver**: Ensure the appropriate ODBC driver for your database is installed.
2. **Edit the odbc.ini file**: Add the DSN entries as shown in the example above.
3. **Test the DSN**: Use a tool like isql to test the DSN connection.

isql MySQL\_DSN myuser mypassword

**Configuration Example for Windows**

On Windows systems, the odbc.ini file is typically managed through the ODBC Data Source Administrator, but the configuration principles remain the same.

1. **Open ODBC Data Source Administrator**: Go to Control Panel > Administrative Tools > Data Sources (ODBC).
2. **Add a new DSN**: Select the appropriate driver and enter the connection details.

**3) what is means by** **Data Redundancy?**

In Flat files sometimes we are maintaining multiple copy of the same data in different location.

*. this data is also called as duplicate data or redundant data.*

**4) what are all the ACID properties?**

 **Atomicity**: The order placement and payment processing are part of a single transaction. If the payment processing fails, the order should not be placed.

 **Consistency**: The transaction ensures that the order status and inventory levels remain consistent. If an item is ordered, the inventory count is reduced accordingly.

 **Isolation**: Multiple customers can place orders concurrently without interfering with each other's transactions. Each order is processed as if it were the only transaction happening.

 **Durability**: Once an order is confirmed and payment is processed, the order details are saved permanently. Even if the system crashes, the order information remains intact.

**5) what are all the DDL commands?**

1. **DDL (Data Definition Language)** 
   1. *. Create*
   2. *. Alter*
   3. *. Drop*
   4. *. Truncate*
   5. *. Rename (Oracle 9i)*

6) **DML (Data Manipulation Language)?**

* 1. *Insert*
  2. *Update*
  3. *Delete*
  4. *Merge (Oracle 9i)*

**7) TCL (Transaction Control Language)?**

* 1. *Commit*
  2. *Rollback*
  3. *Savepoint*

**8) DCL (Data Control Language) ?**

* 1. Grant
  2. Revoke

**9) what is the difference between char and varchar?**

**Char:** It is used to store fixed length alpha numeric data in bytes. Maximum limit is up to 2000 bytes.

**varchar2**: datatype it is used to store variable length of alphanumeric data in bytes. Max size is 4000 bytes.

**10) how to add new column in a existed table?**

**11) how to delete a column from the existed table?**

**12) how can you rename the table name?**

**13) how to create a new table/duplicate table by using existed table. It should contain data as well?**

a) Create table <new table> as select \* from <existed table> where 1=1;

**14) how to create a new table by using existed table to get only metadata?**

a) Create table <new table> as select \* from <existed table> where 1=2;

**15) Difference between delete and truncate:  
 Delete:**

1.using **Delete** command we can *delete entire table* and also we can delete a portion of the data using **where clause**. Once the table data is deleted, deleted data internally stored in buffer, hence using rollback command we can roll back the delete data until **commit** command is executed

2. In **Delete** command we can use the where clause

**Truncate:**

1. Whenever we are using **truncate** table, then all rows are permanently deleted we can’t get it back this data by using rollback also because truncate is a DDL command and ddl command **transactions are automatically committed**.

2. truncate command will not support **where** clause condition.

**16) what is the output of ‘select 2 from emp’?**

It will return all the rows as ‘2’ itself. How many records are there in the table those many time ‘2’ will repeat.

**17) what is the output of ‘select \* from emp order by 3’?**

It will sort the data base for 3rd columns and give the output.

**18) Waq to get the below data by using substr and instr functions? \*\*\***input data:  
[saishaik123@gmail.com](mailto:saishaik123@gmail.com)  
[izan11@gmail.com](mailto:izan11@gmail.com)  
[shahid@gmail.com](mailto:shahid@gmail.com)  
[anil1@gmail.com](mailto:anil1@gmail.com)  
[Radha@gmail.com](mailto:Radha@gmail.com)  
  
**output:**saishaik123  
izaz11  
shahid  
anil  
Radha

**Answer:**   
SQL> select mailname, substr(mailname,1,instr(mailname,'@')-1) name from email;

**19)** **write a query to display the data which is start and end character is same? \*\*\***

select \* from frstlstmtch where substr(col1,1,1)=substr(col1,-1,1);



**20) I have a data like below the table contain -- single row comment**

/\* -- multirow comment

Tab\_1 -- this is table name

col1

1

1

null

\*/

select count(\*) from tab\_1; -- what is the output

select count(1) from tab\_1; -- what is the output

select count(col1) from tab\_1; -- what is the output

create table tab\_1(col1 varchar(2)) ;

insert into tab\_1 values ('1');

insert into tab\_1 values ('1');

insert into tab\_1 values (null);

select col1 from tab\_1;

select rowid,col1 from tab\_1;

select count(\*) from tab\_1; -- o/p: 3

select count(1) from tab\_1; -- o/p: 3

select count(col1) from tab\_1; -- o/p :2

select \* from emp

select 2 from emp;

select count(1) from emp;

select empno,1,3 from emp;

if a column contain ‘null’ and if you use that column in ‘count()’ clause then it will not consider in the count.

**21) Order of the SQL query Execution: (interview question)**

1) From  
2) Where   
3) Group by   
4) Having   
5) Select   
6) order by

**22) what is Natural Join:**

This join also returns matching rows only. this is similar to inner join, but it gives high performance compared to inner join.

In this join not required to used join condition explicitly. But in this case resource tables must have a common column based on this common column oracle server internally establish joining condition.

*Syn:*

*Select \* from table1* ***natural join*** *table2;*

Eg: select \* from emp natural join dept;

23) **Cross join:**

Cross join will return Cartesian result.

Which mean left dataset completely multiply with right dataset.

When emp table – 14

Dept table – 4

24) **SELF JOIN: (imp interview question)**

. joining a table itself is called self join

**25) WAQ to display the employees who are getting more salary than their manager salary from emp table by using self-join?**

SQL> select

e1.ename "employees",

e2.ename "manager",

e1.sal emp\_Sal,

e2.sal mgr\_sal

from

emp e1,

emp e2

where e1.mgr=e2.empno

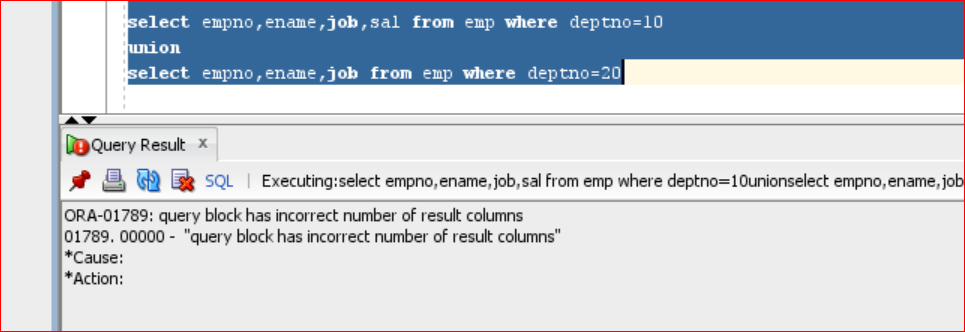
and e1.sal>e2.sal;

**26) what is the difference between union and union all?**

**Union** : it returns unique values from both datatsets and also sorting those values

**Union all :** Unique + duplicate values

**27) my first table contain 5 columns 2nd table contain 4 columns, what will happen if I do union?**



It will through error. Make sure how many colums are there in table1 those many columns should be in the 2nd table as well. While doing the union operation.

**28) what is mean by intersect?**

**Intersect:** it returns common values or marched records with distinct values (unique)

select \* from test1 (1,2,3,3,3,4,5)

intersect

select \* from test3; (4,5,5,5,6,7)

output: (4,5)

**29) Minus:?**

It returns values from first query those values are not in second query.

select \* from test1

minus

select \* from test3;

test1{1,2,3,4)  
test2{2,5,7}

tests Minus Test2 {1,3,4}

30) **what is decode():**

Decode() is a conversion function which is used to decoding the values. Decode conversion function Is same as **if-then-else** construct in pl-sql.

**Syn:** *decode(colum\_name, value1, stmts1, value2, stmt2, stmts);*

Eg:

select deptno, decode(deptno,10,'ten',20,'twenty',30,'thirty','others') dcode from emp;

**31) in my source table Male>>’f’ and Female >>’M’ wanted to convert where ‘f’ is as ‘Male’ and “M’ female? How to write a query?**

gender:

Male

M

Female

F

Male  
Female

F

Answer) Select case when gender =’M’ then ‘Female’   
 when gender =’f’ then ‘Male’

Else Gender end from table\_name;

**32)** **Difference between nvl(), coalesce(): \*\*\***

* **NVL** is an oracle function where as **coalese** is an ANSI function and also **coalese** performance is very high as compared to NVL function.
* NVL function internally uses implicit conversions i.e NVL function returns a value if the exp1,exp2 is not belong to same datatype also if exp2 automatically converted into exp1 where as in **coalese** function exp1,exp2 must belongs to same datatype.

**33) How to handle the null values in a table?**

*By using* ***Nvl*** *or* ***coalesce*** *()*

**34) how many types of constraints we do have in SQL?**

1) NOT NULL

2) UNIQUE

3) Primary Key

4) Foreign Key

5) Check

6) DEFAULT

**34) what is the difference between NOT NULL and UNIQUE constrain?**

*-NOT NULL doesn’t allow null values but it allows duplicate values*.

- unique constraints doesn’t accept duplicate value but accept null values.

**35) what is the difference between PRIMARY KEY and FOREIGN KEY?**

**primary key: *doesn’t accept duplicate values & null values***

And primary key constraint we can use only once in a table.  
Whenever we are creating primary key also oracle server automatically creates B Tree index on those columns.

Primary Key = NOT NULL + UNIQUE

**foreign key:**

if you want to establish relationship on base tables then we are using referential integrity constraints foreign key.

One Table foreign key must belong to another table primary key and also these two columns must belong to same datatype.

Always foreign key values based on primary key values only.

Generally primary key doesn’t accept duplicate, null values whereas foreign key accepts duplicates & null values.

Syntax:

|  |
| --- |
| Create table table\_name (col1 datatype(size), col2 datatype(size),….  foreign key(col1,col2…..)  references master\_table\_name(primary\_key col)); |

**36) can we delete the master table records if they are present in the child table?**

No, we can’t delete and it will through an error like, child records are found.

Whenever we are establishing relation between table by using foreign key then oracle server automatically violates following 2 rules, these are,

**37) what is sub query and co-related Sub query?**

**Sub- Query:**

in subquery first child query will execute and get the result, then that result will assign to the parent query where clause, based on that result parent query will execute.

**Syntax:** select \* from <parant\_tabel> where <operators> (select \* from child\_qry);

**Co-related subquery:**

. Generally, in no-correlated subquery, child query is executed first then parent query is executed.

. Whereas in co-related subquery first parent query executed first, then after child query is executed, that result will assign to parent query again.

. Generally, in non-corelated subquery child query is executed only once per parent query.

whereas co-related subquery child query is executed for each row per parent query table.

. In co-relate subquery we must create an alias name for parent query table and then pass the alias name in child query in “where” condition.

|  |
| --- |
| **Syntax:**  Select \* from table\_name aliasname  Where column\_name=(select \* from table\_name where columnname=aliasname.colname); |

**38)** **waq to display 1st highest salary employee from emp table by using co-related subquery?**

select \* from emp e1 where 1=(select count(\*) from emp e2 where e2.sal>=e1.sal);

**39)** **WAQ to display nth highest salary employee from the emp table by using corelates subquery.**

SQL> select \* from emp e1 where &n=(select count(distinct sal) from emp e2 where e2.sal>=e1.sal);

**40) what is the Inline View?**

Inline views are a special type of query having a sub query in **FROM** clause of the parent query.

Generally, we are not allowed to use order by clause in child query , to over come this problem, oracle introduced sub-queries. In from Clause in parent query.

these types of queries are also called as inline views

**syntax:**

select \* from (subquery);

**41) what is the difference between ROWNUM, ROWID?**

**Rownum:**

* . Rownum is a pseudo column which behaves like a table column.
* . rownum is used to restrict no.of rows in a table.
* . These pseudo columns are belongs to all databases.
* . it will generate temporary individual unique values for each select row

**ROWID:**

* . rowid is an pseudo column, it behaves like a table column.
* . generally rownum having tempary values where as rowid having fixed values.
* . when ever we are inserting data into a table then oracle server automatically generates an unique identification number for identify the record uniquely,for each and every row this is called rowid.

**43) what is the difference between ROW\_NUMBER(), RANK(), DENSE\_RANK()?**

* **Row\_number()** analytical function automatically assigns different rank\_number when values are same
* **rank()** analytical function automatically assigns same rank numbers when values are same and also skip the next consecutive number(1,1,3,3,5)
* **dense\_rank()** analytical function assign same number and doesn’t skip the sequence.

**Row\_number:**

Dno ename sal rn

20 a 100 1

20 b 100 2

20 c 100 3

30 d 200 4

**Rank():**

Dno ename sal rnk

20 a 100 1

20 b 100 1

20 c 100 1

30 d 200 4

30 e 100 4  
40 f 100 6

**Dense\_Rank():**

Dno ename sal dnsernk

20 a 100 1

20 b 100 1

20 c 100 1

30 d 200 2

40 e 300 3

40 e 600 3

50 e 600 4

**44)** **WAQ to display the employee details highest salary to lowest salary and also automatically assign rank to each dept from table by using analytical function?**

SQL> select deptno,ename,sal,row\_number() over(partition by deptno order by sal desc) rn from emp;

**45)** **WAQ to display 2nd highest salary employee in each dept from emp table by using analytical function?**

SQL> select \* from (

select deptno,empno,ename,sal, row\_number() over(partition by deptno order by sal desc) rn from emp) where rn=2;

correct result is:

SQL> select \* from (select deptno,ename,sal,dense\_rank() over(partition by deptno order by sal desc) rn from emp) where rn=2;

DEPTNO ENAME SAL RN

---------- ---------- ---------- ----------

10 CLARK 2450 2

20 JONES 2975 2

30 ALLEN 1600 2

**46) WAQ to display duplicate records from the table? \*\*\*\*\*\*\*\*\***

**Method1:**

select deptno,count(\*) from emp1 group by deptno having count(\*)>1;

deptno count(\*)

30 6

10 3

20 5

**Method2:**

select \* from (

select deptno,ename,row\_number() over(partition by deptno order by deptno) dnk from emp1 ) where dnk>1;

**47) waq to find the unique records from the table?**

**Method1:**

select deptno,count(\*) from emp1 group by deptno having count(\*)=1;

deptno count(\*)

30 6

10 3

20 5

**Method2:**

select \* from (

select deptno,ename,row\_number() over(partition by deptno order by deptno) dnk from emp1 ) where dnk=1;

**48) WAQ to DELETE duplicate data from emp? \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Method1:**

SQL> delete from emp\_dup where (deptno,rowid) not in (select deptno,min(rowid) from emp\_dup group by deptno);

**Method2:**

delete from emp\_test where rowid in (

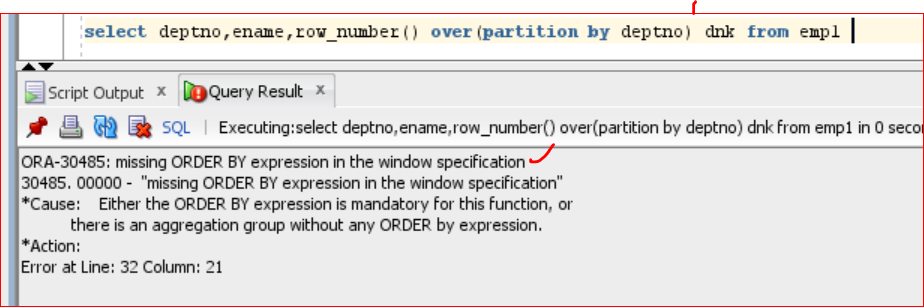
select rowid from (select rowid,deptno,ename,row\_number() over(partition by deptno order by rowid) rn from emp\_test ) where rn>1

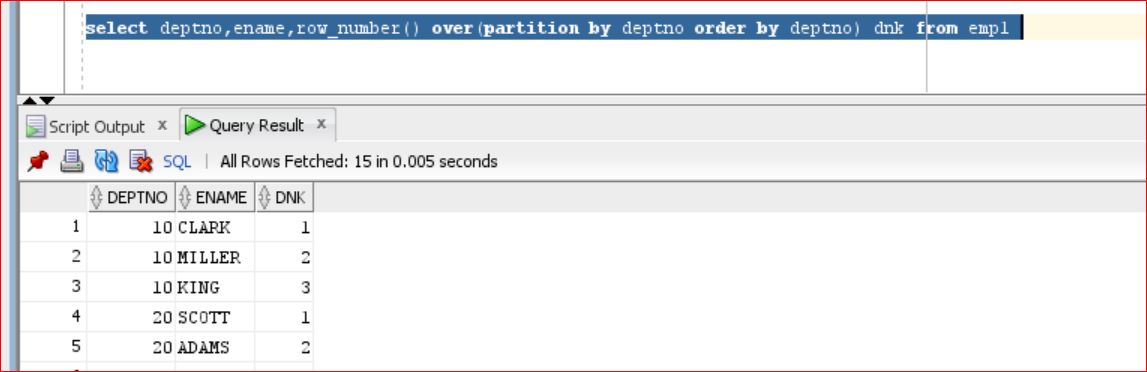
);

**49) order by clause is mandatory or not in the analytical functions?**

“Order By” is mandatory in the analytical functions, but ‘**partition by’** is not mandatory

Below is the example





**Delete duplicates scenarios:**

|  |  |
| --- | --- |
| **data** 1 1 1 2 2 3 4 4 5  1) remove deplicates?  case1 - Complete duplicates  1) find duplicates?  query1) select data,**count**(\*) from tab1 group by data having **count**(\*)>1;  **data** 1 1 1 2 2 4 | 3  **2) find only unique records**  query2) select data,**count**(\*) from tab1 group by data having **count**(\*)=1;  3,5  **3) delete duplicates?**  delete from tab1 where data in (selet data,count(\*) from tab1 group by data having count(\*)>1);  **case2 - keep 1 common records and delete subsequence records?**  **1) find the duplicates**  select \* from (select rnm, **row\_number**() over(partition by rnm order by rnm) rnk from tab11\_test) where rnk>1;  **2) delete the duplicats (in this scenario must use rowid to delete the duplicates)**  delete from tab11\_test where rowid in         (select rowid from             (select rnm, **row\_number**() over( partition by rnm order by rnm) rnk from tab11\_test) where rnk>1); |

**50) what is exist operator and what is the use of it?**

**Exists operator:**

. Exists operator always return Boolean value either **true** or **false**

. Exists operator performance is very high compared to **IN** operator

. Exists operator used in co-related subquery

Syntax:

Select \* from table\_name aliasname where exists

(select \* from table\_name where tablename.col\_name=aliasname.col\_name);

Eg: exists {1,2,3,4} = True

Exists { } = false

**Q) WAQ to display department from dept table those dept having employees in emp table by using corelated subquery exists operator.**

Select \* from dept d where exists (select \* from emp where deptno=d.deptno);

**51) what is mean by CTE? And how you can use that?**

**CTE - (common table expression)**

**Syntax:**

A common table expression (CTE) is a named temporary result set that exists within the scope of a single statement and that can be referred to later within that statement, possibly multiple times

WITH

cte1 AS (SELECT a, b FROM table1),

cte2 AS (SELECT c, d FROM table2)

SELECT b, d FROM cte1 JOIN cte2

WHERE cte1.a = cte2.c;

Example:

with  
     e1 as (select \* from emp),  
     d1 as (select \* from dept)  
    select \* from e1 inner join d1 on e1.deptno=d1.deptno;

**52) what is Temp table?**

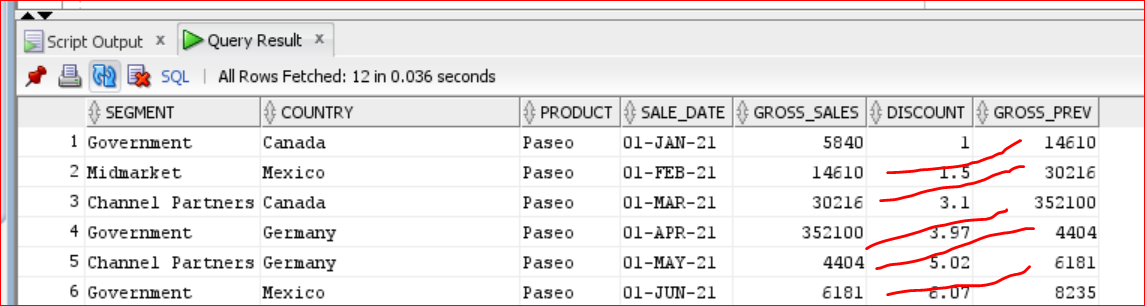
**Temp Table:**

A TEMPORARY table is visible only within the current session, and is dropped automatically when the session is closed. This means that two different sessions can use the same temporary table name without conflicting with each other or with an existing non- TEMPORARY table of the same name.

## **53) LEAD() function**

LEAD() is an [analytic function](https://www.oracletutorial.com/oracle-analytic-functions/) that allows you to access the following row from the current row without using a [self-join](https://www.oracletutorial.com/oracle-basics/oracle-self-join/).

select segment,country,product,sale\_date,gross\_sales,discount,  
**lead**(gross\_sales) over(order by sale\_DAte,gross\_Sales) gross\_prev from Financial;



**Lag:**

LAG() is an [analytic function](https://www.oracletutorial.com/oracle-analytic-functions/) that allows you to access the row at a given offset prior to the current row without using a [self-join](https://www.oracletutorial.com/oracle-basics/oracle-self-join/)

select segment,country,product,sale\_date,gross\_sales,discount,**lag**(gross\_sales) over(order by sale\_DAte,gross\_Sales) gross\_prev  
 from Financial;

# **LISTAGG()**

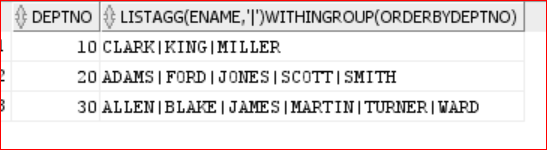
LISTAGG() function is an [aggregation function](https://www.oracletutorial.com/oracle-aggregate-functions/) that transforms data from multiple rows into a single list of values separated by a specified delimiter.

LISTAGG (

[ALL] column\_name [,delimiter]

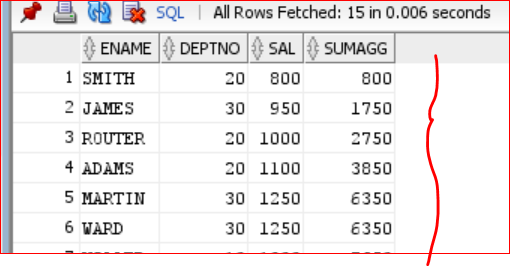
) WITHIN GROUP( ORDER BY sort\_expressions);

select deptno,listagg(ename,'|') within group (order by deptno) from emp group by deptno



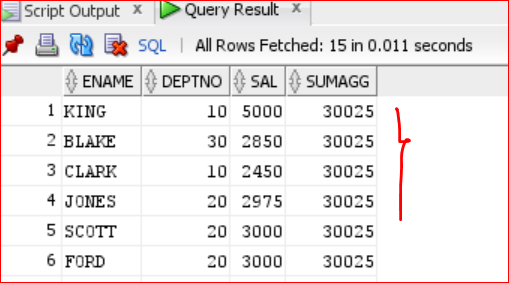
**54) waq to Cumulative sum on salary?**

select ename,deptno,sal,**sum**(sal) over( order by sal) sumagg from emp;



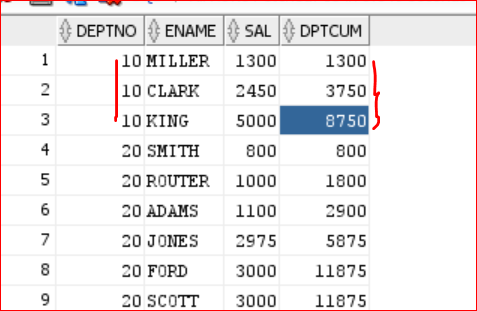
**55) Total sum for each row?**

select ename,deptno,sal,**sum**(sal) over() sumagg from emp;

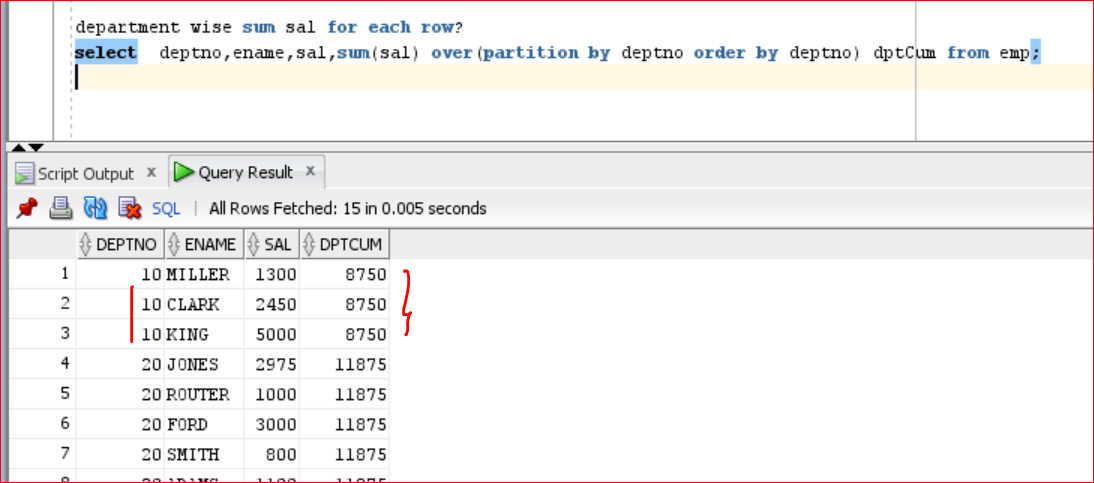


**54) waq to department wise Cumulative sum salary?**

select  deptno,ename,sal,**sum**(sal) over(partition by deptno   
order by sal) dptCum from emp;



**55) waq to department wise sum salary?**



**56) department wise avg,min,max sal for each row?**

select deptno,ename,sal,avg(sal) over(partition by deptno order by deptno) dptCum from emp;

select deptno,ename,sal,min(sal) over(partition by deptno order by deptno) dptCum from emp;

select deptno,ename,sal,max(sal) over(partition by deptno order by deptno) dptCum from emp;

**57) what is view? Types of views and usage of views?**

a **view** is a virtual table that is based on the result set of a SQL query. *Views do not store data physically but present data from one or more tables*. They provide a way to simplify complex queries, enhance security by restricting access to specific data, and present data in a particular format.

There are 2 types of view

1) Simple View  
2) Complex View

**Simple view** is a view which is created from only one base table whereas

**DML Operation on simple view?**

we can also perform DML operator through simple view to base table based on following restrictions.

1) If a simple view having group functions group by clause, rownum, distinct , set operators, join then we can’t perform DML operations through simple view to the base table.

2) we must include base table not null column into the view then only we are allow to perform insertion operation through simple view to the base table.

**Syntax:**

Create or replace view view\_name

As

Select statement from base\_table;

**Eg:**

SQL> create or replace view v1

as

select \* from emp1 where deptno=10;

select \* from v1;

**complex view** is a view which is created from no.of base tables.

create or replace view v5 as   
select ename,sal,dname,loc from emp,dept   
where emp.deptno=dept.deptno;

SQL> select \* from emp;

Generally, we can’t perform DML operation through complex view to base table.

SQL> update v5 set ename='abc' where ename='SMITH';

**Materialized view:**

* M.View used in data warehousing application.
* Generally views doesn’t stored data whereas materialized views store data
* Generally materialized views are used to improved performance of the joined or aggregable queries
* M.Views are store data same like base table but when are refreshing M.views synchronize the data based on base table.

Syntax:

Create materialized view view\_name

As

Select \* from tables

Ex:

Create materialized view mv1

As

Select \* from emp1;

Sql> select \* from mv1;

**Materialized view refesh:**

execute dbms\_mview.refresh('EMP\_mz1');

**58) what is sequence in database how it will generate the sequence?**

**SEQUENCE:**

* Sequence is a database object, which is used to generates sequence number automatically.
* Generally, sequences are used to generates primary key values automatically.
* once sequence are created, then no.of users simultaneously access that sequence.
* Sequence is an independent database object.

***Syntax:***

*Create sequence sequence\_name*

*Start with n*

*Increment by n*

*Minvalue n*

*Maxvalue n*

*Cycle/nocycle*

*Cache/nocache;*

* If we want to generate sequence value or access sequence value then we are using following 2 pseudo columns.

1) currval

2) nextval

Syntax: sequence\_name.currval

Syntax: sequence\_name.nextval

**Note**: pseudo columns are used in select,insert,update,delete statement in oracle. If we want to generates sequence values by using select statements then we must use dual table.

Syntax: select sequence\_name.curval from dual;

Syntax: Select sequence\_name.nextval from dual;

### 59) What is an Index?

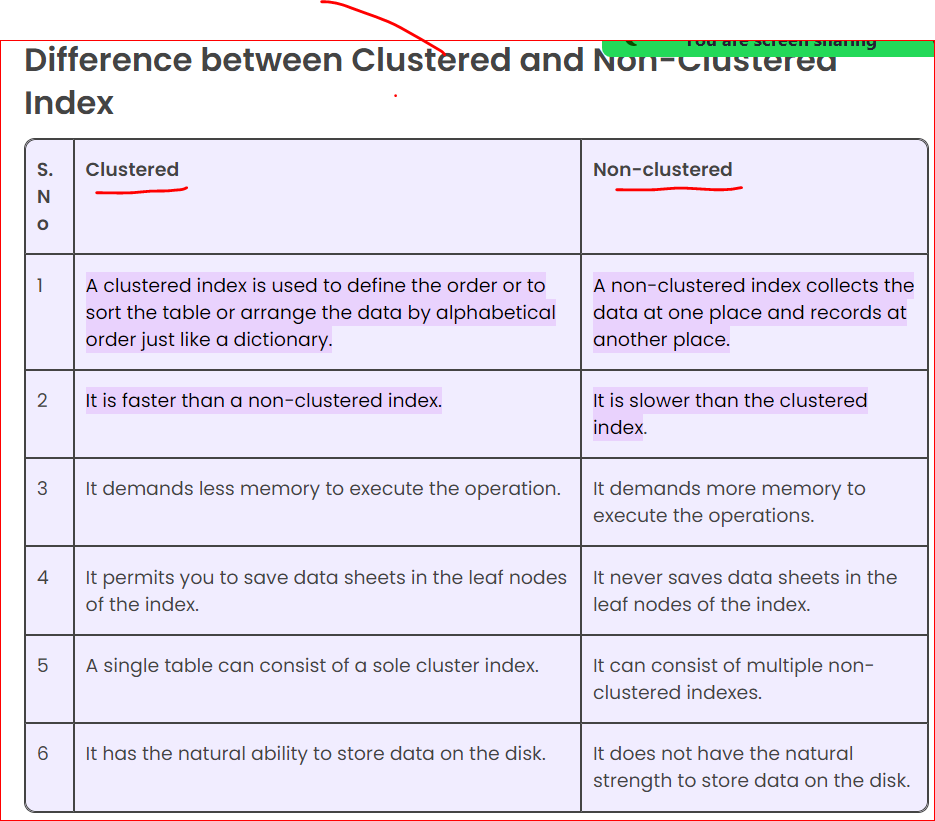
an **index** is a database object that improves the speed of data retrieval operations on a table. Indexes are used to enhance the performance of queries by providing a more efficient way to access rows. However, indexes also have an overhead during data modifications like INSERT, UPDATE, and DELETE, as the indexes must be maintained

### Uses of Indexes

1. **Improve Query Performance**: By providing a faster way to locate rows based on key values.
2. **Support for Constraints**: Indexes are often used to enforce primary key and unique constraints.
3. **Facilitate Sorting**: Indexes can be used to speed up sorting operations (e.g., ORDER BY and GROUP BY clauses).

### Types of Indexes in DB2

1. **B-tree Index**
   * **Structure**: Balanced tree structure.
   * **Use Case**: Suitable for a wide range of queries, especially those that involve exact matches or range queries.
   * **Difference**: Efficient for most general purposes, but may not be the best choice for columns with very low cardinality.
2. **Bitmap Index**
   * **Structure**: Uses bitmaps for each key value.
   * **Use Case**: Ideal for columns with low cardinality, such as gender or Boolean flags.
   * **Difference**: Efficient in data warehousing environments with complex ad-hoc queries, but less efficient for transactional systems due to overhead in maintaining bitmaps.
3. **Clustered Index**
   * **Structure**: Physically orders the rows in the table based on the index key.
   * **Use Case**: Beneficial for range queries and queries that retrieve large chunks of data.
   * **Difference**: Improves performance of range-based queries and impacts the physical storage order of data.
4. **Non-clustered Index**
   * **Structure**: Separate index structure that does not affect the physical order of data in the table.
   * **Use Case**: Useful for enhancing query performance without changing the physical storage order.
   * **Difference**: Multiple non-clustered indexes can be created on a table, whereas typically only one clustered index is allowed.
5. **Composite Index**
   * **Structure**: Index on multiple columns.
   * **Use Case**: Optimizes queries that filter on multiple columns.
   * **Difference**: Combines multiple columns into a single index, improving performance for queries that involve those columns.
6. **Unique Index**
   * **Structure**: Enforces uniqueness of values in the indexed column(s).
   * **Use Case**: Used to enforce primary key or unique constraints.
   * **Difference**: Ensures no duplicate values, unlike other indexes which do not enforce uniqueness.



**60) types of Partitions in Oracle?**

**Partitions table**

Partition tables are used to improve the query performance of the application in backup and recovery process.(DBA)

Partition tables are created by database administrator in **every large database**.

Partition tables are used in data warehousing applications; partitions table are created based on **key column**; this column is also called partition key.

In oracle if we want to view particular partitions then we are use following sysntax

Select \* from table\_name partition (partion\_name1,partition\_name2….)

1) Range Partitions

2) list partitions

3) hash partitions.

**61) what is the normalization?**

**NORMALIZATION:**

. Normalization is specific process which is used to decomposing a table into no.of tables. This process automatically reduces duplicate/redundant data and also automatically avoids insertion, updating & deletion problems.

**Normal Form:**

1) 1st Normal Form

2) 2nd Normal From

3) 3rd Normal Form

4) BCNF Normal Form (Boyee Code)

5) 4th Normal Form

6) 5th Normal Form

**62) What is the De-normalization?**

Traditionally data is stored in normalized databases, in which multiple separate tables are maintained in a relational database to minimize the redundant data. Therefore, whenever we have to access data from multiple tables, we need to perform complex and costly join operations on the required tables.

**63)** **Performance tuning Techniques:**

* [12 Query optimization tips for better performance](https://blog.devart.com/how-to-optimize-sql-query.html#query-optimization-tips-for-better-performance)
  + [Tip 1: Add missing indexes](https://blog.devart.com/how-to-optimize-sql-query.html#missing-indexes)
  + [Tip 2: Check for unused indexes](https://blog.devart.com/how-to-optimize-sql-query.html#Non-used-indexes)
  + [Tip 3: Avoid using multiple OR in the FILTER predicate](https://blog.devart.com/how-to-optimize-sql-query.html#or-in-join-predicate)
  + [Tip 4: Use wildcards at the end of a phrase only](https://blog.devart.com/how-to-optimize-sql-query.html#use-wildcards)
  + [Tip 5: Avoid too many JOINs](https://blog.devart.com/how-to-optimize-sql-query.html#high-table-count)
  + [Tip 6: Avoid using SELECT DISTINCT](https://blog.devart.com/how-to-optimize-sql-query.html#avoid-using-select-distinct)
  + [Tip 7: Use SELECT fields instead of SELECT \*](https://blog.devart.com/how-to-optimize-sql-query.html#use-select-fields-instead-of-select-all)
  + [Tip 9: Run the query during off-peak hours](https://blog.devart.com/how-to-optimize-sql-query.html#run-query-during-offpeak-hours)
  + [Tip 11: Minimize large write operations](https://blog.devart.com/how-to-optimize-sql-query.html#minimize-large-write-operations)
  + [Tip 12: Create joins with INNER JOIN (not WHERE)](https://blog.devart.com/how-to-optimize-sql-query.html#create-joins-with-inner-join)
  + Tip 13: Use **explain plan** and identify the bottle neck

**64) What is an Index?**

An index is a data structure that improves the speed of data retrieval operations on a database table at the cost of additional storage space and processing time for insertions, updates, and deletions. In simpler terms, an index helps you find data quickly without having to search every row in a database table.

**Types of Indexes**

1. **Primary Index/B-Tree Index (Balanced Tree):**
   * **Description:** A primary index/B-Tree Index is automatically created when a primary key is defined on a table. It ensures that the column or set of columns defined as the primary key will have unique and not-null values.
   * **Example:** In a table employees, if the employee\_id is the primary key, an index is automatically created on employee\_id.
2. **Unique Index**
   * **Description:** A unique index ensures that the values in the indexed columns are unique across the table.
   * **Example:** If you have a column email in the users table and you want to ensure that no two users can have the same email address, you would create a unique index on the email column.
3. **Clustered Index**
   * **Description:** A clustered index determines the physical order of data in a table. There can be only one clustered index per table, as the data rows themselves are stored in the index.
   * **Example:** If a table orders has a order\_id column and it is set as the clustered index, the rows in the table will be stored in the order of order\_id.
4. **Non-clustered Index**
   * **Description:** A non-clustered index creates a separate structure from the data rows. The index contains pointers to the data rows that contain the indexed value.
   * **Example:** In a table products, you might create a non-clustered index on the product\_name column to speed up searches on product names.
5. **Composite Index**
   * **Description:** A composite index, also known as a multi-column index, is an index on two or more columns of a table.
   * **Example:** If a table sales has columns customer\_id and product\_id, you can create a composite index on both columns to speed up queries involving both columns.
6. **Full-text Index**
   * **Description:** A full-text index is used to improve the performance of full-text searches on large text fields.
   * **Example:** In a books table with a description column, a full-text index can be used to efficiently search for specific words or phrases within the descriptions.
7. **Bitmap Index**
   * **Description:** A bitmap index uses bitmaps and is particularly efficient for columns with a limited number of distinct values (low cardinality).
   * **Example:** In a customers table, if there is a gender column with values 'Male' and 'Female', a bitmap index would be suitable.

**Differences Between Index Types**

* **Primary vs. Unique Index:**
  + Both ensure uniqueness, but a primary index also prevents null values and is automatically created with the primary key.
* **Clustered vs. Non-clustered Index:**
  + Clustered index dictates the physical storage order of the data, while non-clustered does not.
  + Only one clustered index can be created per table, but multiple non-clustered indexes can exist.
* **Composite vs. Single-column Index:**
  + Composite indexes span multiple columns, providing faster access for queries involving those columns.
* **Full-text vs. Regular Index:**
  + Full-text indexes are optimized for text search, while regular indexes (like B-trees) are optimized for exact matches and range queries.
* **Bitmap vs. B-tree Index:**
  + Bitmap indexes are more efficient for columns with low cardinality, while B-tree indexes are more general-purpose and suited for high cardinality.

**Advantages of Indexes**

1. **Speed Up Data Retrieval:**
   * Indexes can significantly reduce the time it takes to retrieve data from a table.
2. **Efficient Search Operations:**
   * They help in quickly locating rows based on indexed column values.
3. **Improved Performance for Joins:**
   * Indexes can improve the performance of join operations between tables.
4. **Enhanced Sorting:**
   * Indexes can be used to speed up sorting operations on large datasets.
5. **Reduced Disk I/O:**
   * By narrowing down the search range, indexes reduce the number of disk I/O operations required.

**Example**

Let's consider a table employees with columns employee\_id, name, department, and email.

**Without Index:**

sql

Copy code

SELECT \* FROM employees WHERE name = 'John Doe';

* The database would perform a full table scan, checking each row to find John Doe.

**With Index on name Column:**

sql

Copy code

CREATE INDEX idx\_name ON employees(name);

* Now, the same query will use the index idx\_name, directly jumping to rows where name is John Doe, speeding up the retrieval process.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Cardinality in Databases Cardinality refers to the uniqueness of data values contained in a column. It is an important concept in database design and optimization. Understanding the cardinality of a column helps in choosing the right type of index and optimizing query performance. Low Cardinality Low cardinality means that a column contains a limited number of unique values. In other words, the same value is repeated many times across different rows. Examples of Low Cardinality:  1. **Gender Column:**    * In a table customers, the gender column might only contain values 'Male' and 'Female'.    * Example Data:  | **customer\_id** | **name** | **gender** | | --- | --- | --- | | 1 | Alice | Female | | 2 | Bob | Male | | 3 | Charlie | Male | | 4 | Diana | Female |  1. **Status Column:**    * In a table tasks, the status column might contain values 'Pending', 'In Progress', 'Completed'.    * Example Data:  | **task\_id** | **description** | **status** | | --- | --- | --- | | 101 | Fix bugs | In Progress | | 102 | Develop feature X | Pending | | 103 | Code review | Completed | | 104 | Write tests | Pending |  High Cardinality High cardinality means that a column contains a large number of unique values. Each value is unique or nearly unique across different rows. Examples of High Cardinality:  1. **Email Column:**    * In a table users, the email column is expected to have a unique email address for each user.    * Example Data:  | **user\_id** | **name** | **email** | | --- | --- | --- | | 1 | Alice | alice@example.com | | 2 | Bob | bob@example.com | | 3 | Charlie | charlie@example.com | | 4 | Diana | diana@example.com |  1. **Order ID Column:**    * In a table orders, the order\_id column will have a unique identifier for each order.    * Example Data:  | **order\_id** | **product\_name** | **quantity** | | --- | --- | --- | | 1001 | Laptop | 1 | | 1002 | Smartphone | 2 | | 1003 | Headphones | 3 | | 1004 | Monitor | 1 |  Significance of Cardinality  1. **Index Selection:**    * Low Cardinality: Bitmap indexes are more efficient for columns with low cardinality because they use less space and are faster for such scenarios.    * High Cardinality: B-tree indexes are suitable for high cardinality columns as they efficiently handle a large number of unique values. 2. **Query Optimization:**    * Knowing the cardinality helps the database optimizer choose the best execution plan for queries. For instance, a high cardinality column might be better suited for equality searches, while a low cardinality column might be better for aggregation operations. 3. **Storage and Performance:**    * High cardinality columns may require more storage for indexes, but they provide faster lookups for unique searches.    * Low cardinality columns use less storage for indexes but may not significantly speed up searches unless combined with other columns in composite indexes.   **How Indexes Improve Performance**  Indexes improve the performance of database queries by allowing the database management system (DBMS) to locate data without scanning every row in a table. They act as lookup tables, enabling quick data retrieval and efficient searching, sorting, and filtering operations.  **How Indexes Search for a Matching Record**  Indexes use various data structures (such as B-trees, hash tables, or bitmaps) to organize and manage the data for fast access. Here’s a general overview of how indexes work, with a focus on the B-tree index, which is one of the most common types:   1. **B-tree Index Structure:**    * A B-tree index is a balanced tree data structure that maintains sorted data and allows for efficient insertion, deletion, and search operations.    * The tree consists of nodes, where each node contains keys (indexed values) and pointers to child nodes. The keys in each node are sorted.    * The root node is the top node, and the leaf nodes are at the bottom level of the tree. 2. **Searching in a B-tree Index:**    * **Step 1: Start at the Root Node**      + The search begins at the root node of the B-tree.    * **Step 2: Compare Keys**      + The keys in the current node are compared to the search value.      + If the search value matches a key in the node, the search is successful.      + If the search value is less than a key, the search follows the pointer to the left child node.      + If the search value is greater than a key, the search follows the pointer to the right child node.    * **Step 3: Traverse Down the Tree**      + This process is repeated, traversing down the tree, until the leaf nodes are reached.    * **Step 4: Find the Record**      + Once the search reaches the appropriate leaf node, the corresponding record or pointer to the record is retrieved.   **Example**  Consider a table employees with columns employee\_id, name, and department. We create an index on the name column.  **Creating the Index:**  sql  Copy code  CREATE INDEX idx\_name ON employees(name);  **Searching for a Record Without an Index:**  sql  Copy code  SELECT \* FROM employees WHERE name = 'John Doe';   * Without an index, the DBMS performs a full table scan, checking each row one by one until it finds the row where name is 'John Doe'. This can be slow, especially for large tables.   **Searching for a Record With an Index:**  sql  Copy code  SELECT \* FROM employees WHERE name = 'John Doe';   * With the idx\_name index, the DBMS performs a search in the B-tree index:   + **Step 1:** Starts at the root node.   + **Step 2:** Compares 'John Doe' with the keys in the current node.   + **Step 3:** Traverses down the tree following the appropriate pointers.   + **Step 4:** Locates the leaf node containing 'John Doe' and retrieves the pointer to the actual record in the table.   **Advantages of Using Indexes**   1. **Faster Data Retrieval:**    * Indexes allow the DBMS to quickly locate the data without scanning the entire table, reducing query execution time. 2. **Efficient Sorting:**    * Indexes can speed up sorting operations by providing pre-sorted data. 3. **Improved Join Performance:**    * Indexes on columns used in join conditions can significantly enhance the performance of join operations between tables. 4. **Optimized Aggregation:**    * Indexes can improve the performance of aggregate functions (e.g., COUNT, SUM) by providing quick access to the relevant data.   **Disadvantages of Using Indexes**   1. **Additional Storage:**    * Indexes require additional storage space to maintain the index data structure. 2. **Maintenance Overhead:**    * Indexes need to be updated whenever data is inserted, updated, or deleted, which can add overhead to these operations. 3. **Performance Degradation for Frequent Updates:**    * Tables with frequent insertions, updates, or deletions may experience performance degradation due to the need to maintain indexes. |

**65) What is Merge and How it works?**

The MERGE statement in Oracle SQL is used to perform an "upsert" operation, which means it can update existing rows or insert new rows depending on whether a condition is met. This is particularly useful for synchronizing tables or updating them based on the results of another table or query.

### Syntax

sql

Copy code

MERGE INTO target\_table

USING source\_table

ON (condition)

WHEN MATCHED THEN

UPDATE SET column1 = value1, column2 = value2, ...

WHEN NOT MATCHED THEN

INSERT (column1, column2, ...)

VALUES (value1, value2, ...);

### Example

Let's consider two tables: employees and employees\_update. We want to synchronize employees with the new data from employees\_update.

#### Table Structures

**employees**

| **employee\_id** | **name** | **department** | **salary** |
| --- | --- | --- | --- |
| 1 | Alice | HR | 50000 |
| 2 | Bob | IT | 60000 |
| 3 | Charlie | Sales | 55000 |

**employees\_update**

| **employee\_id** | **name** | **department** | **salary** |
| --- | --- | --- | --- |
| 1 | Alice | HR | 52000 |
| 2 | Bob | IT | 60000 |
| 4 | Diana | Marketing | 58000 |

### Using MERGE

We want to update the employees table with the data from employees\_update. If an employee\_id exists in employees\_update but not in employees, we insert a new row. If an employee\_id exists in both tables, we update the existing row.

MERGE INTO employees e

USING employees\_update eu

ON (e.employee\_id = eu.employee\_id)

WHEN MATCHED THEN

UPDATE SET e.name = eu.name,

e.department = eu.department,

e.salary = eu.salary

WHEN NOT MATCHED THEN

INSERT (e.employee\_id, e.name, e.department, e.salary)

VALUES (eu.employee\_id, eu.name, eu.department, eu.salary);

### Result

After executing the MERGE statement, the employees table will be updated as follows:

**employees**

| **employee\_id** | **name** | **department** | **salary** |
| --- | --- | --- | --- |
| 1 | Alice | HR | 52000 |
| 2 | Bob | IT | 60000 |
| 3 | Charlie | Sales | 55000 |
| 4 | Diana | Marketing | 58000 |

### Explanation

1. **Matched Rows (employee\_id = 1 and 2):**
   * For employee\_id = 1, the name, department, and salary are updated to Alice, HR, and 52000 respectively.
   * For employee\_id = 2, since the data is the same in both tables, no change is made.
2. **Not Matched Rows (employee\_id = 4):**
   * A new row is inserted for employee\_id = 4 with values Diana, Marketing, and 58000.

### Key Points

* **Target Table (employees):** The table being updated.
* **Source Table (employees\_update):** The table providing the new data.
* **ON Condition:** The condition to match rows between the target and source tables.
* **WHEN MATCHED:** The action to take when rows match the condition.
* **WHEN NOT MATCHED:** The action to take when no matching rows are found in the target table.

**66) what is partition and how it works in Oracle database?**

### Partitions in Oracle

Partitioning in Oracle refers to the process of dividing a large table or index into smaller, more manageable pieces called partitions. Each partition can be managed and accessed independently, improving performance, manageability, and availability. Oracle supports several types of partitioning strategies, each suitable for different types of data and queries.

### Types of Partitioning

1. **Range Partitioning**
   * Divides data based on a range of values.
   * Example: Partitioning sales data by year.
2. **List Partitioning**
   * Divides data based on a list of discrete values.
   * Example: Partitioning customers by region.
3. **Hash Partitioning**
   * Divides data based on a hash function.
   * Example: Evenly distributing data to avoid skew.
4. **Composite Partitioning**
   * Combines multiple partitioning methods.
   * Example: Range-Hash or Range-List partitioning.

### How Partitions Work

Partitions allow Oracle to manage large tables and indexes more efficiently. When a query accesses a partitioned table, Oracle can quickly locate the relevant partitions and process only the necessary data, rather than scanning the entire table. This improves query performance and resource utilization.

### Example of Partitioning

Let's create an example using range partitioning. We'll create a sales table partitioned by year.

#### Step 1: Create the Partitioned Table

sql

Copy code

CREATE TABLE sales (

sale\_id NUMBER,

sale\_date DATE,

amount NUMBER,

customer\_id NUMBER

)

PARTITION BY RANGE (sale\_date) (

PARTITION p\_2019 VALUES LESS THAN (TO\_DATE('01-JAN-2020', 'DD-MON-YYYY')),

PARTITION p\_2020 VALUES LESS THAN (TO\_DATE('01-JAN-2021', 'DD-MON-YYYY')),

PARTITION p\_2021 VALUES LESS THAN (TO\_DATE('01-JAN-2022', 'DD-MON-YYYY')),

PARTITION p\_2022 VALUES LESS THAN (TO\_DATE('01-JAN-2023', 'DD-MON-YYYY'))

);

In this example:

* The sales table is partitioned by the sale\_date column.
* Four partitions are created, one for each year from 2019 to 2022.

#### Step 2: Insert Data into the Partitioned Table

sql

Copy code

INSERT INTO sales VALUES (1, TO\_DATE('15-MAR-2019', 'DD-MON-YYYY'), 100, 101);

INSERT INTO sales VALUES (2, TO\_DATE('20-JUL-2020', 'DD-MON-YYYY'), 200, 102);

INSERT INTO sales VALUES (3, TO\_DATE('10-DEC-2021', 'DD-MON-YYYY'), 150, 103);

INSERT INTO sales VALUES (4, TO\_DATE('05-MAY-2022', 'DD-MON-YYYY'), 250, 104);

#### Step 3: Query the Partitioned Table

When you query the sales table, Oracle uses partition pruning to limit the partitions it needs to scan.

sql

Copy code

SELECT \* FROM sales WHERE sale\_date >= TO\_DATE('01-JAN-2021', 'DD-MON-YYYY');

In this query:

* Oracle only scans the partitions p\_2021 and p\_2022 since the sale\_date condition only involves data from these partitions.

### Benefits of Partitioning

1. **Improved Query Performance:**
   * Partition pruning allows Oracle to scan only relevant partitions, reducing the amount of data processed.
2. **Manageability:**
   * Partitions can be managed independently, allowing for operations like loading, indexing, and backup to be performed on individual partitions.
3. **Enhanced Availability:**
   * Partitions can be moved, archived, or maintained without affecting the entire table.
4. **Maintenance Operations:**
   * Maintenance tasks such as rebuilding indexes and gathering statistics can be performed at the partition level, reducing downtime.
5. **Scalability:**
   * Partitioning helps in scaling the database to handle large volumes of data more efficiently.

**String, Number, Date & Type Conversion Functions:**

**String Manipulation Questions**

1. How do you concatenate strings in Oracle SQL?

Answer: You can use the || operator to concatenate strings. Example:

SELECT 'Hello' || ' ' || 'World' AS concatenated\_string FROM dual;

1. **How do you convert a string to uppercase and lowercase in Oracle SQL?**

Answer: Use the UPPER() and LOWER() functions.

SELECT UPPER('hello') AS upper\_string, LOWER('WORLD') AS lower\_string FROM dual;

1. **How do you extract a substring from a string in Oracle SQL?**

Answer: Use the SUBSTR() function. Example:

SELECT SUBSTR('Hello World', 1, 5) AS substring FROM dual;

1. **How do you find the length of a string in Oracle SQL?**

Answer: Use the LENGTH() function. Example:

SELECT LENGTH('Hello World') AS string\_length FROM dual;

1. How do you replace characters in a string in Oracle SQL?

Answer: Use the REPLACE() function. Example:

SELECT REPLACE('Hello World', 'World', 'Oracle') AS replaced\_string FROM dual;

**Date Manipulation Questions**

1. How do you get the current date and time in Oracle SQL?

Answer: Use the SYSDATE function. Example:

SELECT SYSDATE AS current\_date FROM dual;

* **How do you format a date in Oracle SQL?**

Answer: Use the TO\_CHAR() function with a date format. Example:

SELECT TO\_CHAR(SYSDATE, 'YYYY-MM-DD HH24:MI:SS') AS formatted\_date FROM dual;

1. How do you convert a string to a date in Oracle SQL?

Answer: Use the TO\_DATE() function. Example:

SELECT TO\_DATE('2023-07-15', 'YYYY-MM-DD') AS converted\_date FROM dual;

1. **How do you add days to a date in Oracle SQL?**

Answer: Use arithmetic operations with dates. Example:

SELECT SYSDATE + 7 AS date\_next\_week FROM dual;

1. How do you find the difference between two dates in Oracle SQL?

Answer: Subtract one date from another. Example:

SELECT TO\_DATE('2023-07-15', 'YYYY-MM-DD') - TO\_DATE('2023-07-10', 'YYYY-MM-DD') AS date\_difference FROM dual;

**Type Conversion Questions**

1. **How do you convert a number to a string in Oracle SQL?**

Answer: Use the TO\_CHAR() function. Example:

SELECT TO\_CHAR(12345) AS string\_number FROM dual;

1. **How do you convert a string to a number in Oracle SQL?**

Answer: Use the TO\_NUMBER() function. Example:

SELECT TO\_NUMBER('12345') AS number\_string FROM dual;

1. **How do you handle implicit data type conversions in Oracle SQL?**

Answer: Oracle SQL automatically performs implicit conversions when possible, but it's best to use explicit conversion functions like TO\_CHAR(), TO\_DATE(), and TO\_NUMBER() to avoid errors.

1. What is the difference between CAST and TO\_\* conversion functions in Oracle SQL?

Answer: The CAST function is ANSI SQL standard and can be used for converting between various data types. The TO\_\* functions are Oracle-specific and provide more flexibility for formatting.

SELECT CAST('2023-07-15' AS DATE) AS cast\_date FROM dual;

SELECT TO\_DATE('2023-07-15', 'YYYY-MM-DD') AS to\_date FROM dual;

1. **How do you convert a timestamp to a date in Oracle SQL?**

Answer: Use the CAST function or TRUNC() function. Example:

SELECT CAST(SYSTIMESTAMP AS DATE) AS date\_from\_timestamp FROM dual;

SELECT TRUNC(SYSTIMESTAMP) AS truncated\_date FROM dual;

**Practical Scenario Questions**

1. **How would you extract the year, month, and day from a date in Oracle SQL?**

Answer: Use the EXTRACT() function. Example:

SELECT EXTRACT(YEAR FROM SYSDATE) AS year, EXTRACT(MONTH FROM SYSDATE) AS month, EXTRACT(DAY FROM SYSDATE) AS day FROM dual;

1. **How do you handle date format mismatches when converting strings to dates?**

Answer: Ensure the format model in TO\_DATE() matches the string format. Example:

SELECT TO\_DATE('15-JUL-2023', 'DD-MON-YYYY') AS correct\_date FROM dual;

1. **How would you handle a requirement to display dates in different time zones?**

Answer: Use the NEW\_TIME() function or FROM\_TZ() and AT TIME ZONE clauses. Example:

SELECT NEW\_TIME(SYSDATE, 'PST', 'EST') AS est\_time FROM dual;

SELECT FROM\_TZ(CAST(SYSDATE AS TIMESTAMP), 'UTC') AT TIME ZONE 'US/Eastern' AS est\_time FROM dual;

**Advanced String Manipulation Questions**

1. **How do you handle and clean up strings that contain extra spaces or unwanted characters in Oracle SQL?**

Answer: Use TRIM(), LTRIM(), RTRIM(), and REPLACE() functions.

SELECT TRIM(' Hello World ') AS trimmed\_string, LTRIM(' Hello World ') AS left\_trimmed\_string, RTRIM(' Hello World ') AS right\_trimmed\_string, REPLACE('Hello World!!', '!', '') AS cleaned\_string FROM dual;

1. **How do you find and extract email domains from a list of email addresses?**

Answer: Use the INSTR() and SUBSTR() functions to locate and extract the domain.

SELECT SUBSTR(email, INSTR(email, '@') + 1) AS domain FROM (SELECT 'user@example.com' AS email FROM dual UNION ALL SELECT 'admin@oracle.com' AS email FROM dual);

1. **How do you handle case-insensitive string comparisons in Oracle SQL?**

Answer: Use the UPPER() or LOWER() functions to standardize case before comparison.

SELECT \* FROM employees WHERE UPPER(last\_name) = UPPER('smith');

**Advanced Date Manipulation Questions**

1. **How do you calculate the number of business days between two dates excluding weekends and holidays?**

Answer: Create a function or use a subquery to exclude weekends and holidays from the date range.

SELECT COUNT(\*) FROM (SELECT TRUNC(SYSDATE, 'MM') + LEVEL - 1 AS day FROM dual CONNECT BY LEVEL <= TRUNC(LAST\_DAY(SYSDATE)) - TRUNC(SYSDATE, 'MM') + 1) WHERE TO\_CHAR(day, 'DY') NOT IN ('SAT', 'SUN') AND day NOT IN (SELECT holiday\_date FROM holidays);

1. **How do you find overlapping date ranges in a table?**

Answer: Use a self-join to check for overlapping conditions.

SELECT a.id, b.id FROM date\_ranges a JOIN date\_ranges b ON a.id <> b.id   
WHERE a.start\_date < b.end\_date AND a.end\_date > b.start\_date;

1. **How do you convert time zones and handle daylight saving time changes?**

Answer: Use FROM\_TZ(), AT TIME ZONE, and TZ\_OFFSET().

SELECT FROM\_TZ(CAST(SYSDATE AS TIMESTAMP), 'UTC') AT TIME ZONE 'America/New\_York' AS est\_time FROM dual;

**Advanced Type Conversion Questions**

1. **How do you handle converting a comma-separated string into rows of data?**

Answer: Use REGEXP\_SUBSTR() or a combination of string functions.

WITH temp AS (SELECT 'apple,banana,cherry' AS str FROM dual) SELECT REGEXP\_SUBSTR(str, '[^,]+', 1, LEVEL) AS fruit FROM temp CONNECT BY REGEXP\_SUBSTR(str, '[^,]+', 1, LEVEL) IS NOT NULL;

1. **How do you convert a date with a time component to just the date part, ignoring the time?**

Answer: Use the TRUNC() function.

SELECT TRUNC(SYSDATE) AS date\_only FROM dual;

1. **How do you handle implicit conversions and avoid potential errors with date formats?**

Answer: Always use explicit conversion functions like TO\_CHAR() and TO\_DATE() with appropriate format models.

SELECT TO\_DATE('15-JUL-2023', 'DD-MON-YYYY') AS date\_converted FROM dual;

1. **How do you convert a number to a formatted string with leading zeros?**

Answer: Use the TO\_CHAR() function with a format model.

SELECT TO\_CHAR(123, '000000') AS formatted\_number FROM dual;

**Complex Scenario Questions**

1. **How would you write a query to find the most recent order for each customer, where orders are stored with date and time?**

Answer: Use the ROW\_NUMBER() window function.

SELECT customer\_id, order\_id, order\_date FROM (SELECT customer\_id, order\_id, order\_date, ROW\_NUMBER() OVER (PARTITION BY customer\_id ORDER BY order\_date DESC) AS rn FROM orders) WHERE rn = 1;

1. **How do you pivot a table that has dates in rows to have dates as columns in Oracle SQL?**

Answer: Use the PIVOT clause.

SELECT \* FROM (SELECT customer\_id, order\_date, amount FROM orders) PIVOT (SUM(amount) FOR order\_date IN (DATE '2023-07-01' AS "2023-07-01", DATE '2023-07-02' AS "2023-07-02"));

1. **How do you handle multi-language string comparisons where case and accents must be ignored?**

Answer: Use the NLS\_COMP and NLS\_SORT parameters.

ALTER SESSION SET NLS\_COMP = LINGUISTIC;

ALTER SESSION SET NLS\_SORT = BINARY\_AI;

SELECT \* FROM employees WHERE last\_name = 'garcía';