**“Shree Ganeshay Namah !!!”**

**Questions:-**

1. **What is SQL?**
2. **Difference between CHAR and VARCHAR?**
3. **Difference between VARCHAR and VARCHAR2?**
4. **Data type NUMBER(p,s)?**
5. **What is Data Integrity?**
6. **What is Referential Integrity?**
7. **What is CONSTRAINTS/Integrity constraints?**
8. **Difference between UNIQUE and PRIMARY KEY?**
9. **What are Keys?**
10. **Difference between Unique key and Primary key?**
11. **What is Index?**
12. **Types of Indexes in oracle?**
13. **Difference between b-tree and bitmap index?**
14. **How many indexes are created in a table?**
15. **Advantage and Disadvantage of Index?**
16. **DDL?**
17. **DML?**
18. **DCL?**
19. **TCL?**
20. **Syntax of all SQL statements?**
21. **Difference between DELETE, TRUNCATE, DROP?**
22. **Use of Merge?**
23. **Type of Subqueries?**
24. **Pseudo Columns?**
25. **Difference between ROWID and ROWNUM?**
26. **How to find all db tables, Index, index keys?**
27. **What are String/Char functions?**
28. **What are Numeric/Math Functions?**
29. **What are Date/Time Functions?**
30. **What are Conversion Functions?**
31. **What are Analytical functions?**
32. **What are Advanced Functions?**
33. **What are Miscellaneous Functions?**
34. **Types of RANKING functions?**
35. **NVL, NVL2 function?**
36. **Difference between TRUNC(), ROUND(Date),TO\_DATE(), TO\_CHAR() functions?**
37. **What is difference between SUBSTR() and INSTR() functions?**
38. **What is difference between TRANSLATE() and REPLACE() functions?**
39. **What is DECODE() function?**
40. **What is MOD() and REMAINDER() function?**
41. **Difference between RANK(), ROW\_NUMBER(), DENSE\_RANK() functions?**
42. **What is LISTAGG() function?**
43. **PERCENT\_RANK() function?**
44. **What is FIRST() and LAST() function?**
45. **What is FIRST\_VALUE() and LAST\_VALUE() function?**
46. **What is LEAD() function?**
47. **What is LAG() function?**
48. What is NTH\_VALUE() function?
49. **WITH clause?**
50. **IN Clause?**
51. **EXISTS Clause?**
52. **Which clause we use first GROUP BY or ORDER BY?**
53. **Difference between IN and EXISTS clause?**
54. **Difference between WHERE and HAVING clause?**
55. **Can we use having clause without GROUP BY?**
56. **Can we use WHERE and GROUP BY in single query and what is the order of execution?**
57. **Difference between GROUP BY and PARTITION BY clause?**
58. Difference between GROUP BY and OVER clause?
59. **What are Set Operations?**
60. **What is Joins?**
61. **Types of join?**
62. **Types of Equi joins?**
63. **What is CROSS JOIN/ Cartesian Product?**
64. **What is SELF JOIN?**
65. **What is Antijoins?**
66. **What is Semijoins?**
67. **Join with third table?**
68. **Can join compare NULL values?**
69. **What is Normalization?**
70. **Types of Normalization?**
71. **Difference between 2NF and 3 NF?**
72. **Difference between Normalization and De-normalization?**
73. **Oracle SQL Query Writing and Performance Tuning?**
74. Performance Tuning like Indexes, Hints, Partitioning?
75. **What is Parallel hint in Oracle?**
76. **What is Query Optimization?**
77. What are Exceptions (user defined and oracle raised)?
78. **What is Database partitioning?**
79. **Difference between Explain plan and Execution plan?**
80. **SQL Query is case sensitive or not?**
81. **How to select new table structure from existing table without coping data?**
82. **Can we insert multiple NULLs in UNIQUE constraints column?**
83. **How many ways to create table in oracle?**
84. **How many ways to insert records in the table?**
85. **What is the max size of table?**
86. **How to find number of rows in table other than count(\*)?**
87. **Versions of Oracle?**
88. Difference between 11g and 12c?
89. **Difference between Primary Key and Unique Key?**
90. **Difference between UNIQUE Constraint and UNIQUE index?**
91. **Difference between Primary Key and UNIQUE Constraint?**
92. **Difference between Primary Key and UNIQUE INDEX?**
93. **Difference between Primary key and Foreign key?**
94. **What is Global Temporary Table?**
95. **Difference between COUNT(\*), COUNT(Expression) and COUNT(DISTINCT Expression)?**
96. **What is Execution plan and Explain plan?**
97. **What is Hint?**
98. **Types of Hints?**
99. What is CONNECT BY LEVEL?
100. What is FULL TABLE SCAN?
101. What is PIVOT operator?

**Answers:-**

1. **What is SQL?**Structured Query Language (SQL) is a language that provides an interface to relational database systems.
2. **Difference between CHAR and VARCHAR?**The CHAR data type specifies a fixed-length character string.

The VARCHAR2 data type specifies a variable-length character string.

The difference between a CHAR and a VARCHAR is that a CHAR(n) will ALWAYS be N bytes long, it will be blank padded upon insert to ensure this.

A varchar2(n) on the other hand will be 1 to N bytes long, it will NOT be blank padded.

CHAR(10) --"abc "

VARCHAR2(10) --"abc"

1. **Difference between VARCHAR and VARCHAR2?**The difference between Varchar and Varchar2 is both are variable length

but only 2000 bytes of character of data can be store in varchar

where as 4000 bytes of character of data can be store in varchar2.

1. **Data type NUMBER(p,s)?**Number having precision p and scale s. The precision p can range from 1 to 38. The scale s can range from -84 to 127. Both precision and scale are in decimal digits.

For example, numeric(7,2) is a number that has 5 digits before the decimal and 2 digits after the decimal.

1. **What is Data Integrity?**This chapter explains how to use integrity constraints to enforce the business rules associated with your database and prevent the entry of invalid information into tables.
2. **What is Referential Integrity?**Referential integrity refers to the accuracy and consistency of data within a relationship.

In relationships, data is linked between two or more tables. This is achieved by having the foreign key (in the associated table) reference a primary key value (in the primary – or parent – table). Because of this, we need to ensure that data on both sides of the relationship remain intact.

So, referential integrity requires that, whenever a foreign key value is used it must reference a valid, existing primary key in the parent table.

1. **What is CONSTRAINTS/Integrity constraints?**

Integrity Constraints are used to apply business rules for the database tables.

**NOT NULL constraints:-** To ensure that no null values are allowed.

**UNIQUE constraints:-** To ensure that a given column is unique.

**PRIMARY KEY constraints:-** This constraint defines a column or combination of columns which uniquely identifies each row in the table.

**FOREIGN KEY constraints:-** To ensure that two keys share a primary key to foreign key relationship.

This constraint identifies any column referencing the PRIMARY KEY in another table.

**CHECK Constraint:-** This constraint defines a business rule on a column. All the rows must satisfy this rule. The constraint can be applied for a single column or a group of columns.

1. **Difference between UNIQUE and PRIMARY KEY?**  
   PRIMARY KEY fields contains UNIQUE and NOT NULL values. It cannot allow any NULL values. Ex:- 1 2 3.UNIQUE contains unique values and allowing multiple NULL values as well. Ex.:- 1 2 3 NULL NULL.
2. **What are Keys?**

Key:- Key is a set of columns that is used to uniquely identify the record in a table.

Database supports the following types of keys-

1) Candidate Key

2) Primary Key

3) Alternate Key

4) Unique Key

5) Composite Key

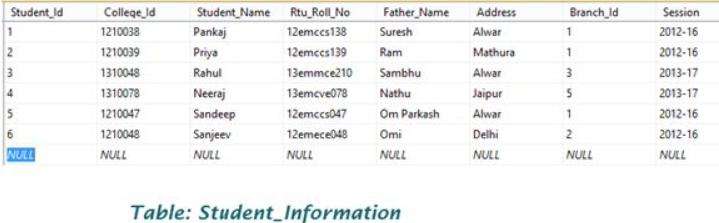
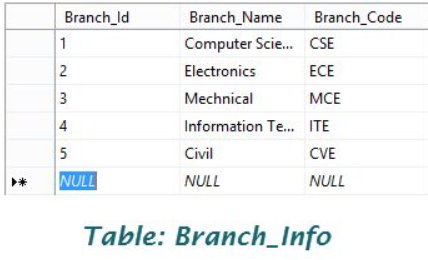
6) Super Key

7) Minimal Super Key

8) Natural Key

9) Surrogate Key

10) Foreign Key



**1) Candidate Key:-** A Candidate key is an attribute or set of attributes that uniquely identifies a record. Among the set of candidate, one candidate key is chosen as Primary Key. So a table can have multiple candidate key but each table can have maximum one primary key.

**Ex.:-** Possible Candidate Keys in Branch\_Info table:- 1) Branch\_Id 2) Branch\_Name 3) Branch\_Code

Possible Candidate keys in Student\_Information table:- 1) Student\_Id 2) College\_Id 3) Rtu\_Roll\_No

**2) Primary key:-** A primary key is a combination of fields which uniquely identifies each record in a table.

A table can have only one primary key and one candidate key can select as a primary key.

Primary key is a special kind of unique key, and it has implicit NOT NULL constraint. It means, Primary key values cannot be NULL.

\* A primary key is unique. A key value cannot occur twice in one table.

\* PRIMARY KEY = (UNIQUE + NOT NULL). Unique:- not allow duplicate, Not null:- at least some value in column.

The primary key should be chosen such that its attributes are never or rarely changed, for example, we can’t select Student\_Id field as a primary key because in some case Student\_Id of student may be changed.

Ex.:- Primary Key in Branch\_Info table:- Branch\_Id

Primary Key in Student\_Information Table:- College\_Id

**3) Alternate Key:-** Alternate keys are candidate keys that are not selected as primary key. Alternate key can also work as a primary key. Alternate key is also called “Secondary Key”.

Ex.:- Alternate Key in Branch\_Info table:- 1) Branch\_Name 2) Branch\_Code

Alternate Key in Student\_Information table: 1) Student\_Id 2) Rtu\_Roll\_No

**4) Unique key:-** A Unique key constraint uniquely identified each record in the database. This provides uniqueness for the column or set of columns.

\* A Primary key constraint has automatic unique constraint defined on it. But not, in the case of Unique Key.

\* There can be many unique constraint defined per table, but only one Primary key constraint defined per table.

Ex.:- Possible Unique Key in Branch\_Info table:- Branch\_Name

Possible Unique Key in Student\_Information table:- Rtu\_Roll\_No

**5) Composite Key:-** Composite key is a combination of more than one attributes that can be used to uniquely identity each record. It is also known as “Compound” key. A composite key may be a candidate or primary key.  
Ex.:- Composite Key in Branch\_Info table:- { Branch\_Name, Branch\_Code}

Composite Key in Student\_Information table:- { Student\_Id, Student\_Name }  
**6) Super Key:-** Super key is a set of on e or more than one keys that can be used to uniquely identify the record in table. A Super key for an entity is a set of one or more attributes whose combined value uniquely identifies the entity in the entity set. A super key is a combine form of Primary Key, Alternate key and Unique key and Primary Key, Unique Key and Alternate Key are subset of super key. A Super Key is simply a non-minimal Candidate Key, that is to say one with additional columns not strictly required to ensure uniqueness of the row. A super key can have a single column.

Ex.:- Super Keys in Branch\_Info Table:- 1)Branch\_Id 2)Branch\_Name 3)Branch\_Code 4){Branch\_Id,Branch\_Code} 5){Branch\_Name, Branch\_Code}

Super Keys in Student\_Information Table:- 1)Student\_Id 2)College\_Id 3)Rtu\_Roll\_No 4){Student\_Id, Student\_Name} 5){College\_Id,Branch\_Id } 6){Rtu\_Roll\_No, Session}  
**7) Minimal Super Key:-** A minimal super key is a minimum set of columns that can be used to uniquely identify a row. In other wordsm the minimum number of columns that can be combined to give a unique value for every row in the table.

Ex.:- Minimal Super Keys in Branch\_Info Table:- 1)Branch\_Id 2)Branch\_Name 3)Branch\_Code

Minimal Super Keys in Student\_Information Table:- 1)Student\_Id 2)College\_Id 3)Rtu\_Roll\_No

**8) Natural Keys:-** A natural key is a key composed of columns that actually have a logical relationship to other columns within a table. For example, if we use Student\_Id, Student\_Name and Father\_Name columns to form a key then it would be “Natural Key” because there is definitely a relationship between these columns and other columns that exist in table. Natural keys are often called “Business Key ” or “Domain Key”.

**9) Surrogate Key:-** Surrogate key is an artificial key that is used to uniquely identify the record in table. For example, in SQL Server or Sybase database system contain an artificial key that is known as “Identity”. Surrogate keys are just simple sequential number. Surrogate keys are only used to act as a primary key.

Ex.:- Branch\_Id is a Surrogate Key in Branch\_Info table and Student\_Id is a Surrogate key of Student\_Information table.

**10) Foreign Keys:-** Foreign key is used to generate the relationship between the tables. Foreign Key is a field in database table that is Primary key in another table. A foreign key can accept null and duplicate value.

Ex.:- Branch\_Id is a Foreign Key in Student\_Information table that primary key exist in Branch\_Info(Branch\_Id) table.

1. **Difference between Unique key and Primary key?**Unique key is similar to primary key but unique key field can contain a “Null” value but primary key doesn’t allow “Null” value.

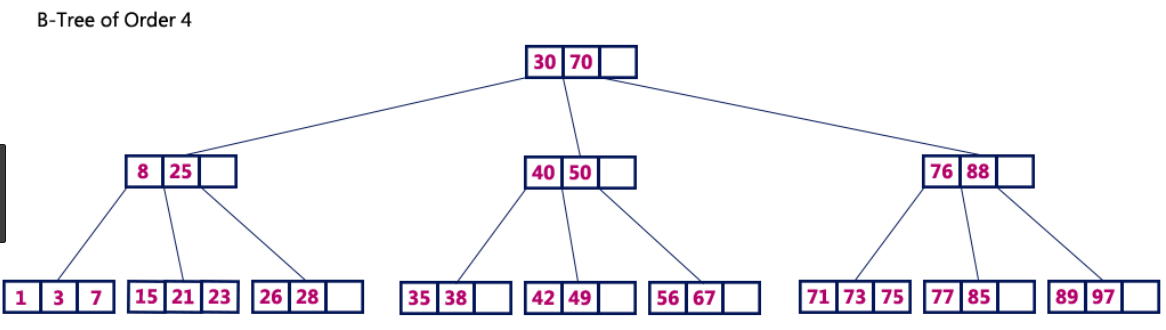
Other difference is that primary key field contain a clustered index and unique field contain a non-clustered index.

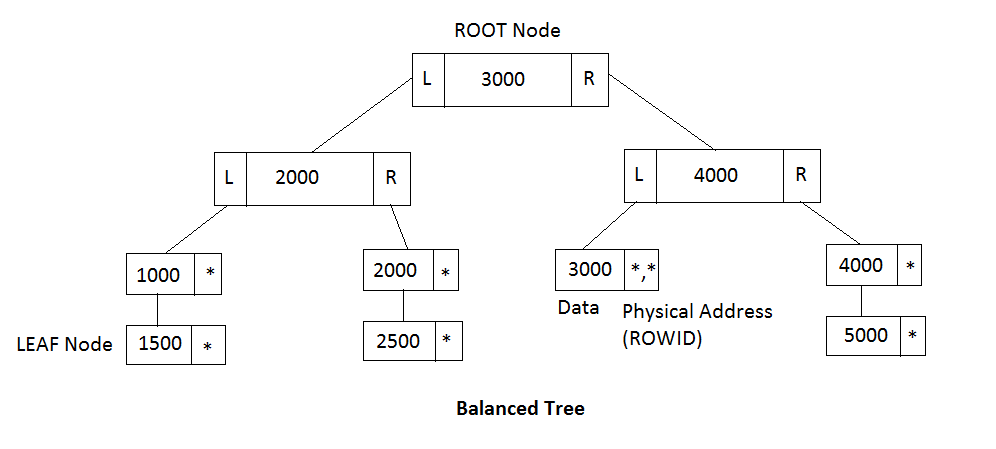
1. **What is Index?**An index is a performance-tuning method of allowing faster retrieval of records.

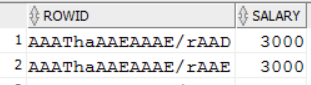
Index internally sort the data in ascending order.

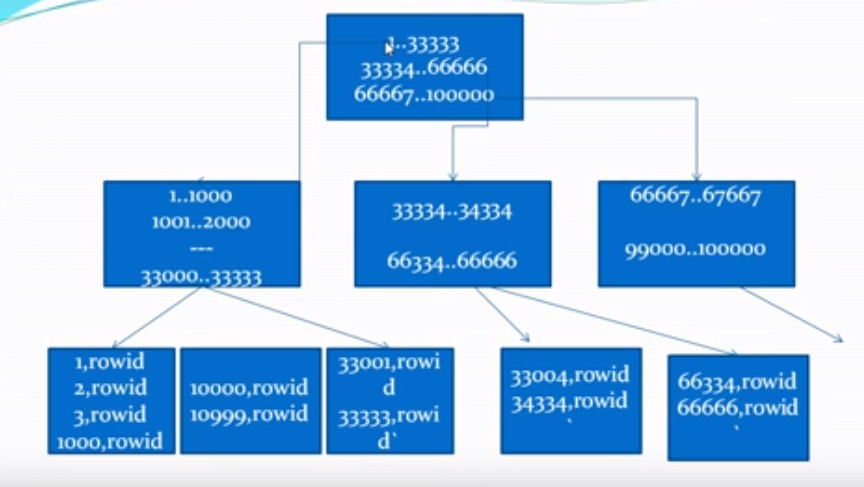
By default, Oracle creates index based on B-tree (Balanced Tree) concept.

Note:- There is no B-tree Index in Oracle. Also B-tree is not a Binary tree.



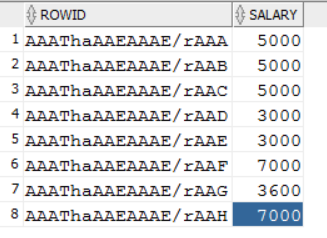
Searching is starting from ROOT node then LEFT/RIGHT node(L to R).  






Example:-

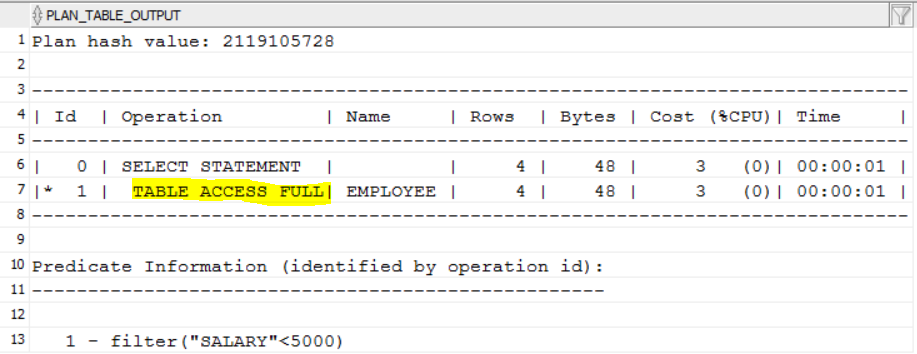
SELECT ROWID, SALARY FROM EMPLOYEE;



SELECT \* FROM employee WHERE salary<5000;

EXPLAIN PLAN FOR SELECT \* FROM EMPLOYEE WHERE SALARY<5000;

SELECT \* FROM TABLE(DBMS\_XPLAN.DISPLAY);



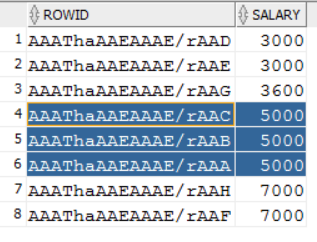
It takes more time to compare each salary using full scan to the table.

Let’s create index on employee table-

CREATE INDEX IDX\_SAL ON EMPLOYEE (SALARY);

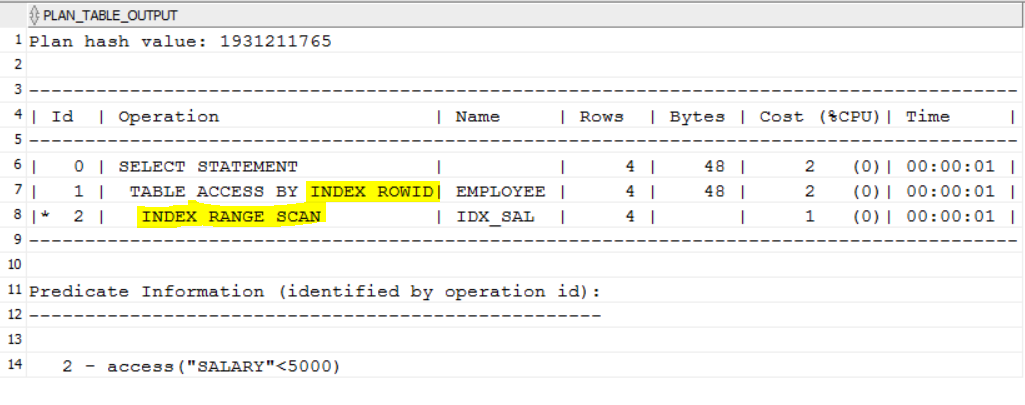
After creating index, Oracle internally creates ordered data into ascending order.

SELECT ROWID, SALARY FROM EMPLOYEE;



EXPLAIN PLAN FOR SELECT \* FROM EMPLOYEE WHERE SALARY<5000;

SELECT \* FROM TABLE(DBMS\_XPLAN.DISPLAY);



It takes less time because data is already in sorted order using index.

First Oracle compare values based on index and then access data by index rowid.

1. **Types of Indexes in oracle?**

There are 6 types of index in Oracle. For every type of index internally oracle create B-tree. **1) Simple Index**

**2) Composite Index**

**3) UNIQUE Index**

**4) NONUNIQUE Index**

**5) REVERSE Index**

**6) Function based Index**

**7) Bitmap Index**

**1) Simple Index:-** Index on single column in a table.

CREATE INDEX idx\_salary ON employee(salary);

**2) Composite Index:-** Index on multiple columns in a table.

CREATE INDEX idx\_empinfo ON employee(emp\_id, salary);

**3) UNIQUE Index:-** Index on unique column only.

We cannot create unique index on columns which contains duplicate values.

CREATE UNIQUE INDEX idx\_mobile ON employee(mobile\_no);

Unique indexes guarantee that no two rows of a table have duplicate values in the key column or columns. For example, no two employees can have the same employee ID. Thus, in a unique index, one rowid exists for each data value. The data in the leaf blocks is sorted only by key.

**4) NONUNIQUE Index:-** Nonunique indexes permit duplicates values in the indexed column or columns. For example, the first\_name column of the employees table may contain multiple Mike values. For a nonunique index, the rowid is included in the key in sorted order, so nonunique indexes are sorted by the index key and rowid (ascending).

CREATE INDEX idx\_salary ON employee(salary);

CREATE INDEX idx\_empinfo ON employee(emp\_id, salary);

**5) REVERSE Index:-** Used when looking for highest value.

Every node in B-tree has 2 subnodesI(LEFT and RIGHT). It search from left to right.

But in case of REVERSE Index it search from right to left. While traverse in right side so it will find highest value as much as fast.

CREATE INDEX idx\_highest\_salary ON employee(salary) REVERSE;

**6) Function based Index:-**

CREATE INDEX idx\_commission ON employee(salary,(0.10\*salary),INITCAP(emp\_nm));

**7) Bitmap Index:-**

Bits- 0 1

CREATE BITMAP INDEX idx\_gender ON employee(gender);

Here 0-Male 1-Female

**------------------------------------------------------------------**

**1) B-tree indexes:-** These are balanced. This means that all the leaf nodes are at the same depth in the tree.

**2) Bitmap indexes:-** which store rowids associated with a key value as a bitmap.

**3)Partitioned indexes:-** which consist of partitions containing an entry for each value that appears in the indexed column(s) of the table.

**4) Function-based indexes:-** which are based on expressions. They enable you to construct queries that evaluate the value returned by an expression, which in turn may include built-in or user-defined functions.

**5) Application Domain indexes:-** which are instances of an application-specific index of type indextype.  
  
**1) B-tree indexes:-**

These are balanced. This means that all the leaf nodes are at the same depth in the tree.

B-tree indexes are used to avoid large sorting operations.

For example, a SQL query requiring 10,000 rows to be presented in sorted order will often use a b-tree index to avoid the very large sort required to deliver the data to the end user.

Oracle offers several options when creating an index using the default b-tree structure. It allows you to index on multiple columns (concatenated indexes) to improve access speeds. Also, it allows for individual columns to be sorted in different orders.

For example, we could create a b-tree index on a column called last\_name in ascending order and have a second column within the index that displays the salary column in descending order.

create index name\_salary\_idx on person

( last\_name asc,

salary desc

);

**2) Bitmap indexes:-**

In bitmap structures, a two-dimensional array is created with one column for every row in the table being indexed. Each column represents a distinct value within the bitmapped index. This two-dimensional array represents each value within the index multiplied by the number of rows in the table. At row retrieval time, Oracle decompresses the bitmap into the RAM data buffers so it can be rapidly scanned for matching values. These matching values are delivered to Oracle in the form of a Row-ID list, and these Row-ID values may directly access the required information.

The real benefit of bitmapped indexing occurs when one table includes multiple bitmapped indexes. Each individual column may have low cardinality. The creation of multiple bitmapped indexes provides a very powerful method for rapidly answering difficult SQL queries.

For example, assume there is a motor vehicle database with numerous low-cardinality columns such as car\_color, car\_make, car\_model, and car\_year. Each column contains less than 100 distinct values by themselves, and a b-tree index would be fairly useless in a database of 20 million vehicles. However, combining these indexes together in a query can provide blistering response times a lot faster than the traditional method of reading each one of the 20 million rows in the base table. For example, assume we wanted to find old blue Toyota Corollas manufactured in 1981.

select license\_plat\_nbr from vehicle and make = 'toyota' and year = 1981;

Oracle uses a specialized optimizer method called a bitmapped index merge to service this query. In a bitmapped index merge, each Row-ID, or RID, list is built independently by using the bitmaps, and a special merge routine is used in order to compare the RID lists and find the intersecting values. Using this methodology, Oracle can provide subsecond response time when working against multiple low-cardinality columns.

1. **Difference between b-tree and bitmap index?**

**1) Syntax differences:-**

By default, Oracle creates B-tree indexes.

The bitmap index includes the "bitmap" keyword. The b-tree index does not say "bitmap".

B-tree index:-

CREATE INDEX idx\_emp\_id ON employee (id);

Bitmap index:-

CREATE BITMAP INDEX idx\_emp\_gender ON employee(gender);

**2) Cardinality differences:-**

B-tree indexes are suitable for columns where there is a high number of unique values (e.g. ID numbers, names).

Bitmap indexes are suitable for columns where there is a low number of unique values (e.g. gender, category).

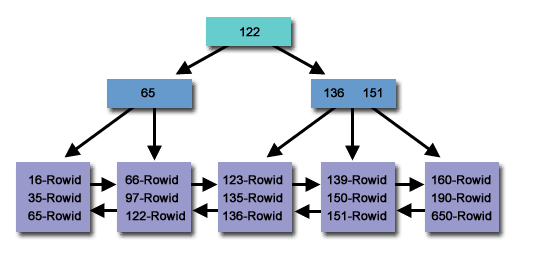
B-tree indexes are useful when there is less or no duplicate values (high cardinality).

Bitmap indexes are used on the columns which has lots of duplicate values(low cardinality).

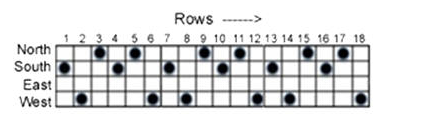
**3) Internal structure differences**:-

B-tree index uses a tree-like structure with index nodes, where Bitmap index uses a two dimensional array with zeros and ones (bits) values.

A b-tree index has index nodes (based on data block size)-



A bitmap index looks like a two-dimensional array with zero and one (bit) values-



1. **How many indexes are created in a table?**Unlimited Indexes can be created per Oracle Table.

Unlimited constraints can be created per Oracle Table.

Total number of columns per table that can be indexed is 32 columns for B-Tree index and 30 columns for Bit Map index.

Maximum 1000 columns created per table.

1. **Advantage and Disadvantage of Index?**Advantages:-

1) Speed up SELECT query.

2) Helps to make a row unique or without duplicates(primary,unique).

3) If index is set to fill-text index, then we can search against large string values. for example to find a word from a sentence etc.

4) Pre-sorted data: The data in the leaf nodes is already sorted by the value of the primary key.

Disadvantages:-

1) Indexes take additional disk space.

2) Indexes slow down INSERT,UPDATE and DELETE, but will speed up UPDATE if the WHERE condition has an indexed field. INSERT, UPDATE and DELETE becomes slower because on each operation the indexes must also be updated.

1. **What is DDL?**

Data Definition Language-  
CREATE, ALTER, DROP, TRUNCATE, COMMENT, RENAME.

1. **What is DML?**Data Manipulation Language-

INSERT, UPDATE, DELETE, MERGE, CALL, EXPLAIN PLAN, LOCK TABLE.  
Note:- SELECT is Data Retrieval Statement

1. **What is DCL?**Data Control Language-

GRANT, REVOKE.

1. **What is TCL?**Transaction Control Language-

COMMIT, ROLLBACK, SAVEPOINT, SET TRANSACTION.

1. **Syntax of all SQL statements?**

CREATE TABLE EMPLOYEE (EMPNO NUMBER(5) PRIMARY KEY,NAME VARCHAR2(20),HIREDATE DATE);

ALTER TABLE EMPLOYEE ADD (DESIGNATION VARCHAR2(45),CITY VARCHAR2(40));

DROP TABLE EMPLOYEE;

TRUNCATE TABLE EMPLOYEE;

COMMENT ON COLUMN EMPLOYEES.JOB\_ID IS ’Abbreviated job title';

ALTER TABLE EMPLOYEE RENAME TO EMPLOYEE;

INSERT INTO EMPLOYEES VALUES(101,'Mohit');

UPDATE EMPLOYEES SET SALARY=70000 WHERE EMPNM='Mohit';

DELETE FROM EMPLOYEE WHERE EMPNAME = 'Smith';

SELECT \* FROM EMPLOYEE;

GRANT SELECT, INSERT, UPDATE, DELETE ON EMPLOYEE TO USERNAME\_MOHIT;

REVOKE DELETE ON EMPLOYEE FROM USERNAME\_MOHIT;

COMMIT;

ROLLBACK;

SET TRANSACTION READ ONLY NAME 'RO\_example';

1. **Difference between DELETE, TRUNCATE, DROP?  
   DELETE:-** The DELETE command is used to remove some or all rows from a table. A WHERE clause can be used to only remove some rows. If no WHERE condition is specified, all rows will be removed.

After performing a DELETE operation you need to COMMIT or ROLLBACK the transaction to make the change permanent or to undo it.

**TRUNCATE:-** TRUNCATE removes all rows from a table. The operation cannot be rolled back and no triggers will be fired. As such, TRUNCATE is faster and doesn't use as much undo space as a DELETE.

**DROP:-** The DROP command removes a table from the database. All the tables' rows, indexes and privileges will also be removed. No DML triggers will be fired. The operation cannot be rolled back.

DROP and TRUNCATE are DDL commands, whereas DELETE is a DML command. Therefore DELETE operations can be rolled back (undone), while DROP and TRUNCATE operations cannot be rolled back.

1. **Use of Merge?**Use the MERGE statement to select rows from one or more sources for update or insertion into a target table. You can specify conditions to determine whether to update or insert into the target.

The MERGE statement is used to conditionally insert or update data depending on its presence, a process also known as an "upsert".

The MERGE statement reduces table scans and can perform the operation in parallel if required.

Consider the following example where data from the HR\_RECORDS table is merged into the EMPLOYEES table.

Target Table- EMPLOYEES

Source Table -HR\_RECORDS

MERGE INTO EMPLOYEES E

USING HR\_RECORDS H

ON (E.ID = H.EMP\_ID)

WHEN MATCHED THEN

UPDATE SET E.ADDRESS = H.ADDRESS

WHEN NOT MATCHED THEN

INSERT (ID, ADDRESS)

VALUES (H.EMP\_ID, H.ADDRESS);

The source can also be a query-

MERGE INTO EMPLOYEES E

USING (SELECT \* FROM HR\_RECORDS WHERE START\_DATE > ADD\_MONTHS(SYSDATE, -1)) H

ON (E.ID = H.EMP\_ID)

WHEN MATCHED THEN

UPDATE SET E.ADDRESS = H.ADDRESS

WHEN NOT MATCHED THEN

INSERT (ID, ADDRESS)

VALUES (H.EMP\_ID, H.ADDRESS);

1. **Type of Subqueries?**

**Single row subquery :-** Returns zero or one row.

**Multiple row subquery:-** Returns one or more rows.

**Multiple column subqueries:-** Returns one or more columns.

**Correlated subqueries :-** Reference one or more columns in the outer SQL statement. The subquery is known as a correlated subquery because the subquery is related to the outer SQL statement.

**Nested subqueries :-** Subqueries are placed within another subquery.

1. **Pseudo Columns?**  
   ROWID, ROWNUM, LEVEL, USER.
2. **Difference between ROWID and ROWNUM?**

ROWID:- AAASf2AABAAAVWRAAA ROWNUM:- 1

1. **How to find all db tables, Index, index keys?**With the help of SYSTEM tables-  
   How to find index columns of table? --ALL\_IND\_COLUMNS

How to find all constraints? --ALL\_CONSTRAINTS

Descriptions of indexes on tables accessible to the user? --ALL\_INDEXES

Description of relational tables accessible to the user? --ALL\_TABLES

Columns of users tables, views and clusters? --ALL\_TAB\_COLUMNS

Information about all users of the database? --ALL\_USERS

1. **What are String/Char functions?**

CONCAT,Concat with ||,CONVERT,INITCAP,INSTR,LENGTH,LOWER,LPAD,LTRIM, REPLACE,RPAD,RTRIM, SUBSTR,TRANSLATE,TRIM,UPPER.

Others:- ASCII,ASCIISTR,CHR,COMPOSE,DECOMPOSE,DUMP,INSTR2,INSTR4,INSTRB,INSTRC,LENGTH2,LENGTH4 ,LENGTHB,LENGTHC,NCHR,REGEXP\_INSTR,REGEXP\_REPLACE,REGEXP\_SUBSTR,SOUNDEX,VSIZE.

1. **What are Numeric/Math Functions?**

COUNT,AVG,SUM,MAX,MIN,TRUNC (numbers),ROUND (numbers), MOD,REMAINDER,POWER,LEAST,ROWNUM,

Others:- ABS,ACOS,ASIN,ATAN,ATAN2,BITAND,CEIL,COS,COSH,EXP,FLOOR,GREATEST,LN,LOG, MEDIAN,REGEXP\_COUNT,SIGN,SIN,SINH,SQRT,TAN,TANH.

1. **What are Date/Time Functions?**

SYSDATE,SYSTIMESTAMP,TRUNC (dates),ROUND (dates),ADD\_MONTHS,LAST\_DAY,NEXT\_DAY,

Others:- CURRENT\_DATE,CURRENT\_TIMESTAMP,DBTIMEZONE,EXTRACT, LOCALTIMESTAMP,MONTHS\_BETWEEN,NEW\_TIME,SESSIONTIMEZONE,TZ\_OFFSET.

1. **What are Conversion Functions?**

CAST,TO\_CHAR,TO\_DATE,TO\_NUMBER,TO\_TIMESTAMP,

Others:- BIN\_TO\_NUM,CHARTOROWID,FROM\_TZ,HEXTORAW,NUMTODSINTERVAL, NUMTOYMINTERVAL,RAWTOHEX,TO\_CLOB,TO\_DSINTERVAL,TO\_LOB, TO\_MULTI\_BYTE, TO\_NCLOB, TO\_SINGLE\_BYTE,TO\_TIMESTAMP\_TZ,TO\_YMINTERVAL.

1. **What are Analytical functions?**

RANK,DENSE\_RANK,FIRST,LAST,FIRST\_VALUE,LAST\_VALUE,LAG,LEAD,LISTAGG.

Others:- CORR,COVAR\_POP,COVAR\_SAMP,CUME\_DIST,NTH\_VALUE,STDDEV,VAR\_POP,VAR\_SAMP,VARIANCE.

1. **What are Advanced Functions?**

CASE,DECODE,NVL,NVL2,

Others:- BFILENAME,CARDINALITY,COALESCE,EMPTY\_BLOB, EMPTY\_CLOB,GROUP\_ID,LNNVL,NANVL,NULLIF,SYS\_CONTEXT,UID,USER,USERENV.

1. **What are Miscellaneous Functions?**

SQLCODE,SQLERRM,Retrieve user ID from the current Oracle session,Retrieve the session ID for the user logged in.

1. **Types of RANKING functions?**

RANK and DENSE\_RANK, CUME\_DIST, PERCENT\_RANK, NTILE ,ROW\_NUMBER

1. **NVL, NVL2 function?  
   NVL:-** NVL ( expr1 , expr2 ): If expr1 is null, then NVL returns expr2.

If expr1 is not null, then NVL returns expr1.

Ex:- select nvl(commission\_pct,0) from employees;

**NVL2:-** NVL2 ( expr1 , expr2 , expr3 ): If expr1 is null, then NVL2 returns expr3.

If expr1 is not null, then NVL2 returns expr2.

Ex:- select nvl2(commission\_pct,commission\_pct,0) from employees;

1. **Difference between TRUNC(), ROUND(Date),TO\_DATE(), TO\_CHAR() functions?  
   TRUNC() Function with dates:-** TRUNC function returns a date truncated to a specific unit of measure.

Ex:- TRUNC(TO\_DATE('22-AUG-03'), 'YEAR')

Result: '01-JAN-03'

TRUNC(TO\_DATE('22-AUG-03'), 'Q')

Result: '01-JUL-03'

TRUNC(TO\_DATE('22-AUG-03'), 'MONTH')

Result: '01-AUG-03'

TRUNC(TO\_DATE('22-AUG-03'), 'DDD')

Result: '22-AUG-03'

**TRUNC() Function with numbers:-** TRUNC function returns a number truncated to a certain number of decimal places.

Ex:-

TRUNC(125.815) --Result: 125

TRUNC(125.815, 0) --Result: 125

TRUNC(125.815, 1) --Result: 125.8

TRUNC(125.815, 2) --Result: 125.81

**ROUND Function (with numbers):-** ROUND function returns a number rounded to a certain number of decimal places.

ROUND(125.315) --Result: 125

ROUND(125.315, 0) --Result: 125

ROUND(125.315, 1) --Result: 125.3

ROUND(125.315, 2) --Result: 125.32

**TO\_DATE() function:-** TO\_DATE function converts a string to a date.

SELECT TO\_DATE('2003/07/09', 'yyyy/mm/dd') FROM dual; --Result: 09-JUL-2003

SELECT TO\_DATE('070903', 'MMDDYY') FROM dual; --Result: 09-JUL-2003

SELECT TO\_DATE('20020315', 'yyyymmdd') FROM dual; --Result: 15-MAR-2002

1. **What is difference between SUBSTR() and INSTR() functions?**  
   **SUBSTR():-** SUBSTR function returns the section of the specified string, specified by numeric character positions.

Ex:- Select SUBSTR('The Three Musketeers',1,3) from dual;

Output:-'The'

**INSTR():-** INSTR function finds the numeric starting position of a string within a string.

Ex:- Select INSTR('Mississippi','i',3,3) test1,

INSTR('Mississippi','i',1,3) test2,

INSTR('Mississippi','i',-2,3) test3 from dual;

Output:- Test1 --11 Test2 --8 Test3 –2  
Select SUBSTR('MOHIT,PUNE', 1, 5 ) Test1,

SUBSTR('MOHIT,PUNE', 7) Test2

from dual;

--Tes1 MOHIT

--Test2 PUNE

Select SUBSTR('MOHIT,PUNE', 1, 5 ) Test1,

INSTR('MOHIT,PUNE', ',' , 1 , 1 ) Test2,

SUBSTR('MOHIT,PUNE', 1 , INSTR('MOHIT,PUNE',',',1,1)-1 ) Test3,

SUBSTR('MOHIT,PUNE', 7, 10 ) Tes4,

SUBSTR('MOHIT,PUNE', 7) Test5,

INSTR('MOHIT,PUNE', ',' , 1 , 1 ) Test6,

SUBSTR('MOHIT,PUNE', INSTR('MOHIT,PUNE',',',1,1)+1 ) Test7

from dual;

--Test1 MOHIT

--Test2 6

--Test3 MOHIT

--Test4 PUNE

--Test5 PUNE

--Test6 6

--Test7 PUNE

1. **What is difference between TRANSLATE() and REPLACE() functions?**REPLACE:- Replace function searches for a string and replaces with the given string.

The Replace function replaces one value in a string with another.

For example, you can replace each occurrence of a letter with matching number.

Syntax:- REPLACE (char,search\_string,replace\_string)

If value for replace\_string is not specify, the search\_string value is when found then it is removed.

SELECT REPLACE('COMPUTER','OM','AB') FROM dual;

Output:- CABPUTER

TRANSLATE:-Translate function searches for a character and it replaces in occurrence of the character.

Translate does an orderly character-by-character substitution in a string.

Syntax:- TRANSLATE (string,if,then)

SELECT TRANSLATE(1256364,2345678,'BDEFGHI') FROM dual;

Output:-BFGDGE

1. **What is DECODE() function?**

The DECODE function has the functionality of an IF-THEN-ELSE statement.

SELECT supplier\_name,

DECODE(supplier\_id, 10000, 'IBM',

10001, 'Microsoft',

10002, 'Hewlett Packard',

'Gateway') result

FROM suppliers;  
select DECODE(1,1,1,0) from dual; --1

select DECODE(1,0,1,0) from dual; --0

select DECODE(1,2,1,0) from dual; --0

1. **What is MOD() and REMAINDER() function?**SELECT MOD(10,1) FROM DUAL; --0

SELECT MOD(10,2) FROM DUAL; --0

SELECT MOD(10,3) FROM DUAL; --1

SELECT MOD(10,4) FROM DUAL; --2

SELECT MOD(10,5) FROM DUAL; --0

SELECT MOD(10,0) FROM DUAL; --10

SELECT REMAINDER(10,1) FROM DUAL; --0

SELECT REMAINDER(10,2) FROM DUAL; --0

SELECT REMAINDER(10,3) FROM DUAL; --1

SELECT REMAINDER(10,4) FROM DUAL; --2

SELECT REMAINDER(10,5) FROM DUAL; --0

SELECT REMAINDER(11,1) FROM DUAL; --0

SELECT REMAINDER(11,2) FROM DUAL; --(-1)

SELECT REMAINDER(11,3) FROM DUAL; --(-1)

1. **Difference between RANK(), ROW\_NUMBER(), DENSE\_RANK() functions?  
   ROW\_NUMBER():-** The ROW\_NUMBER analytic function is similar to the ROWNUM pseudocolumn in that it assigns a unique number for each row returned. It assigns unique numbers to each row within the PARTITION given the ORDER BY clause.

**RANK():-** The RANK analytic function assigns a sequential rank for each distinct value in the specified window.

It behaves like ROW\_NUMBER(), except that “equal” rows are ranked the same.

**DENSE\_RANK():-** The DENSE\_RANK analytic function is similar to the RANK analytic function in that it assigns a sequential rank for each distinct value in the specified window. The difference being the ranks are compacted, so there are no gaps.

DENSE\_RANK() is a rank with no gaps, i.e. it is “dense”.  
Example 1:-  
SELECT v, ROW\_NUMBER() OVER(ORDER BY v),

RANK() OVER(ORDER BY v),

DENSE\_RANK() OVER(ORDER BY v) FROM t;

| V | ROW\_NUMBER | RANK | DENSE\_RANK |

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

| a | 1 | 1 | 1 |

| a | 2 | 1 | 1 |

| a | 3 | 1 | 1 |

| b | 4 | 4 | 2 |

| c | 5 | 5 | 3 |

| c | 6 | 5 | 3 |

| d | 7 | 7 | 4 |

| e | 8 | 8 | 5 |  
**Example 2:-**  
DROP TABLE EMPLOYEE;

CREATE TABLE EMPLOYEE

( EMP\_ID VARCHAR2(5 BYTE),

EMP\_NAME VARCHAR2(20 BYTE),

DEPT\_ID VARCHAR2(5 BYTE),

EXPERTISE VARCHAR2(50 BYTE),

SALARY NUMBER(10,2),

RESULTS VARCHAR2(10 BYTE)

);

Insert into EMPLOYEE (EMP\_ID,EMP\_NAME,DEPT\_ID,EXPERTISE,RESULTS,SALARY) values ('5003','ABINASH','1','SCIENCE','PASS',50000);

Insert into EMPLOYEE (EMP\_ID,EMP\_NAME,DEPT\_ID,EXPERTISE,RESULTS,SALARY) values ('5003','ABINASH','1','ENGLISH','PASS',50000);

Insert into EMPLOYEE (EMP\_ID,EMP\_NAME,DEPT\_ID,EXPERTISE,RESULTS,SALARY) values ('5003','ABINASH','1','MATH','PASS',50000);

Insert into EMPLOYEE (EMP\_ID,EMP\_NAME,DEPT\_ID,EXPERTISE,RESULTS,SALARY) values ('107','AMARESH','2','MATH','PASS',50000);

Insert into EMPLOYEE (EMP\_ID,EMP\_NAME,DEPT\_ID,EXPERTISE,RESULTS,SALARY) values ('107','AMARESH','2','ENGLISH','PASS',50000);

Insert into EMPLOYEE (EMP\_ID,EMP\_NAME,DEPT\_ID,EXPERTISE,RESULTS,SALARY) values ('105','JYOTI','3','MATH','FAIL',75000);

Insert into EMPLOYEE (EMP\_ID,EMP\_NAME,DEPT\_ID,EXPERTISE,RESULTS,SALARY) values ('105','JYOTI','3','ENGLISH','PASS',75000);

Insert into EMPLOYEE (EMP\_ID,EMP\_NAME,DEPT\_ID,EXPERTISE,RESULTS,SALARY) values ('7003','NISHAD','2','ENGLISH','FAIL',70000);

Insert into EMPLOYEE (EMP\_ID,EMP\_NAME,DEPT\_ID,EXPERTISE,RESULTS,SALARY) values ('7003','NISHAD','2','MATH','PASS',70000);

Insert into EMPLOYEE (EMP\_ID,EMP\_NAME,DEPT\_ID,EXPERTISE,RESULTS,SALARY) values ('6003','RAKESH','2','MATH','PASS',50556);

Insert into EMPLOYEE (EMP\_ID,EMP\_NAME,DEPT\_ID,EXPERTISE,RESULTS,SALARY) values ('6003','RAKESH','2','ENGLISH','FAIL',50556);

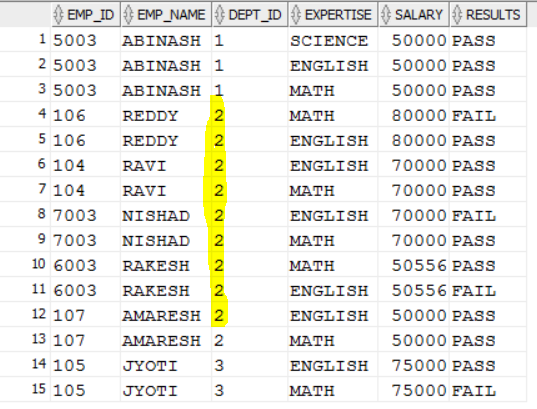
Insert into EMPLOYEE (EMP\_ID,EMP\_NAME,DEPT\_ID,EXPERTISE,RESULTS,SALARY) values ('104','RAVI','2','MATH','PASS',70000);

Insert into EMPLOYEE (EMP\_ID,EMP\_NAME,DEPT\_ID,EXPERTISE,RESULTS,SALARY) values ('104','RAVI','2','ENGLISH','PASS',70000);

Insert into EMPLOYEE (EMP\_ID,EMP\_NAME,DEPT\_ID,EXPERTISE,RESULTS,SALARY) values ('106','REDDY','2','MATH','FAIL',80000);

Insert into EMPLOYEE (EMP\_ID,EMP\_NAME,DEPT\_ID,EXPERTISE,RESULTS,SALARY) values ('106','REDDY','2','ENGLISH','PASS',80000);

SELECT \* FROM EMPLOYEE order by DEPT\_ID,SALARY DESC;



SELECT EMP\_ID, DEPT\_ID,EXPERTISE,SALARY,

SUM(SALARY) OVER (PARTITION BY DEPT\_ID) AS DEPT\_SALARY,

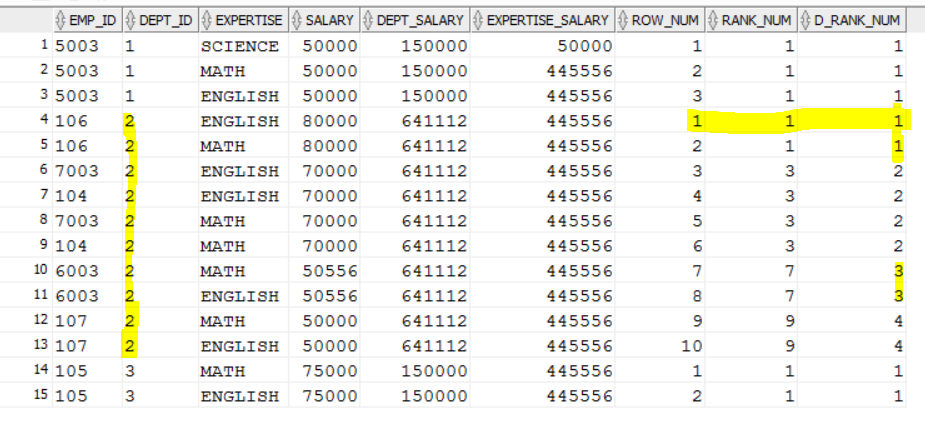
SUM(SALARY) OVER (PARTITION BY EXPERTISE) AS EXPERTISE\_SALARY,

ROW\_NUMBER() OVER (PARTITION BY DEPT\_ID ORDER BY SALARY DESC ) AS ROW\_NUM ,

RANK() OVER (PARTITION BY DEPT\_ID ORDER BY SALARY DESC ) AS RANK\_NUM,

DENSE\_RANK() OVER (PARTITION BY DEPT\_ID ORDER BY SALARY DESC ) AS D\_RANK\_NUM

FROM EMPLOYEE

ORDER BY DEPT\_ID, SALARY DESC;  


1. **What is LISTAGG() function?**LISTAGG() function is used to aggregate strings. Also it allows us to order the elements in the concatenated list.

SELECT deptno, LISTAGG(ename, ',') WITHIN GROUP (ORDER BY ename) AS employees

FROM emp GROUP BY deptno;

DEPTNO EMPLOYEES

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

10 CLARK,KING,MILLER

20 ADAMS,FORD,JONES,SCOTT,SMITH

30 ALLEN,BLAKE,JAMES,MARTIN,TURNER,WARD

1. **PERCENT\_RANK() function?**

The PERCENT\_RANK analytic function assigns value between 0-1 which represents the position of the current row relative to the set as a percentage. The following example displays the top 80% of the rows based on the value.

SELECT val

FROM (SELECT val,

PERCENT\_RANK() OVER (ORDER BY val) AS val\_percent\_rank

FROM rownum\_order\_test)

WHERE val\_percent\_rank >= 0.8;

VAL

----------

9

9

10

10

1. **What is FIRST() and LAST() function?**

The FIRST and LAST functions can be used to return the first or last value from an ordered sequence. Say we want to display the salary of each employee, along with the lowest and highest within their department we may use something like.

SELECT empno,

deptno,

sal,

MIN(sal) KEEP (DENSE\_RANK FIRST ORDER BY sal) OVER (PARTITION BY deptno) AS lowest,

MAX(sal) KEEP (DENSE\_RANK LAST ORDER BY sal) OVER (PARTITION BY deptno) AS highest

FROM emp

ORDER BY deptno, sal;

EMPNO DEPTNO SAL LOWEST HIGHEST

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

7934 10 1300 1300 5000

7782 10 2450 1300 5000

7839 10 5000 1300 5000

7369 20 800 800 3000

7876 20 1100 800 3000

7566 20 2975 800 3000

7788 20 3000 800 3000

7902 20 3000 800 3000

1. **What is FIRST\_VALUE() and LAST\_VALUE() function?**

**FIRST\_VALUE() function:-** The FIRST\_VALUE analytic function is similar to the FIRST analytic function, allowing you to return the first result from an ordered set.

SELECT empno,

deptno,

sal,

FIRST\_VALUE(sal) IGNORE NULLS

OVER (PARTITION BY deptno ORDER BY sal) AS lowest\_in\_dept

FROM emp;

EMPNO DEPTNO SAL LOWEST\_IN\_DEPT

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

7934 10 1300 1300

7782 10 2450 1300

7839 10 5000 1300

7369 20 800 800

7876 20 1100 800

7566 20 2975 800

7788 20 3000 800

7902 20 3000 800

**LAST\_VALUE() function:-** The LAST\_VALUE analytic function is similar to the LAST analytic function, allowing you to return the last result from an ordered set. Using the default windowing clause the result can be a little unexpected

SELECT empno,

deptno,

sal,

LAST\_VALUE(sal) IGNORE NULLS

OVER (PARTITION BY deptno ORDER BY sal ROWS BETWEEN

UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS highest\_in\_dept

FROM emp;

EMPNO DEPTNO SAL HIGHEST\_IN\_DEPT

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

7934 10 1300 5000

7782 10 2450 5000

7839 10 5000 5000

7369 20 800 3000

7876 20 1100 3000

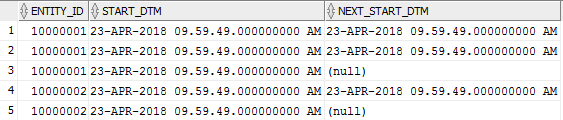
7566 20 2975 3000

7788 20 3000 3000

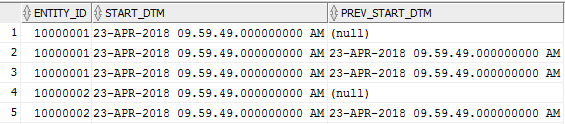
7902 20 3000 3000

This is because the default windowing clause is "RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW", which in this example means the first row with the same value as that of the current row will always be the last row considered. Altering the windowing clause to "ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING" gives us the result we probably expected.

1. **What is LEAD() function?**The LEAD function returns values from the next row in the table.

SELECT ENTITY\_ID,START\_DTM, LEAD(START\_DTM,1) OVER (PARTITION BY ENTITY\_ID ORDER BY START\_DTM) AS NEXT\_START\_DTM FROM RDO.ENTITY\_EVENT;  


1. **What is LAG() function?**The LAG function returns values from a previous row in the table.

SELECT ENTITY\_ID,START\_DTM, LAG(START\_DTM,1) OVER (PARTITION BY ENTITY\_ID ORDER BY START\_DTM) AS PREV\_START\_DTM FROM RDO.ENTITY\_EVENT;  


1. **What is NTH\_VALUE() function?**
2. **Oracle WITH clause?**Oracle With Clause is similar to temporary tables, where you store the data once and read it multiple times in your sql query. Oracle With Clause is used when a sub-query is executed multiple times. In simple With Clause is used to simply the complex SQL. You can improve the performance of the query by using with clause.

**Syntax:-**

With query\_name As

(

SQL query

)

SELECT \* FROM query\_name;

Ex:- We will do a math operation by dividing the salary of employee with the total number of employees in each department.

WITH CNT\_DEPT AS

(

SELECT DEPARTMENT\_ID,

COUNT(1) NUM\_EMP

FROM EMPLOYEES

GROUP BY DEPARTMENT\_ID

)

SELECT EMPLOYEE\_ID,

SALARY/NUM\_EMP

FROM EMPLOYEES E,

CNT\_DEPT C

WHERE E.DEPARTMENT\_ID = C.DEPARTMENT\_ID;

1. **IN Clause?**  
   The Oracle IN condition is used to help reduce the need to use multiple OR conditions in a SELECT, INSERT, UPDATE, or DELETE statement.

SELECT \* FROM customers WHERE customer\_name IN ('IBM', 'Hewlett Packard', 'Microsoft');

1. **EXISTS Clause?**The Oracle EXISTS condition is used in combination with a subquery and is considered "to be met" if the subquery returns at least one row. It can be used in a SELECT, INSERT, UPDATE, or DELETE statement.

SELECT \* FROM customers WHERE

EXISTS (SELECT \* FROM order\_details WHERE customers.customer\_id = order\_details.customer\_id);

1. **Which clause we use first GROUP BY or ORDER BY?**

First will use GROUP BY then ORDER BY.

select emp\_id from employee group by emp\_id order by emp\_id;

1. **Difference between IN and EXISTS clause?**1) In IN clause, The inner query is executed first and the list of values obtained as its result is used by the outer query.The inner query is executed for only once.

In EXISTS clause, The first row from the outer query is selected ,then the inner query is executed and , the outer query output uses this result for checking.This process of inner query execution repeats as many no.of times as there are outer query rows. That is, if there are ten rows that can result from outer query, the inner query is executed that many no.of times.

2) The Exists keyword evaluates true or false, but the IN keyword will compare all values in the corresponding subuery column. If you are using the IN operator, the SQL engine will scan all records fetched from the inner query. On the other hand, if we are using EXISTS, the SQL engine will stop the scanning process as soon as it found a match.

3)The EXISTS clause is much faster than IN when the subquery results is very large. Conversely, the IN clause is faster than EXISTS when the subquery results is very small. Also, the IN clause can't compare anything with NULL values, but the EXISTS clause can compare everything with NULLs.

\*\*\* The EXISTS subquery is used when we want to display all rows where we have a matching column in both tables.

In most cases, this type of subquery can be re-written with a standard join to improve performance.

select book\_key from book where exists (select book\_key from sales);

1. **Difference between WHERE and HAVING clause?**WHERE clause is used to filter the data from table.

HAVING clause is used to filter data from aggregating data. Used to filter data from GROUP BY data.  
When WHERE and HAVING clause are used together in a SELECT query with aggregate function, WHERE clause is applied first on individual rows and only rows which pass the condition is included for creating groups. Once group is created, HAVING clause is used to filter groups based upon condition specified.

If WHERE and HAVING clause is used together, first WHERE clause is applied to filter rows and only after grouping HAVING clause is applied.

1. **Can we use having clause without GROUP BY?**

You often use the HAVING clause with the GROUP BY clause. The GROUP BY clause groups a set of rows into a set of summary rows or groups.

Then the HAVING clause filters groups based on specified conditions.

If you use a HAVING clause without the GROUP BY clause, the HAVING clause behaves like a WHERE clause.

select \* from rdo.dim\_person having 1=1;

1. **Can we use WHERE and GROUP BY in single query and what is the order of execution?**Yes, we can use WHERE and GROUP BY in single query.

SELECT titles.pub\_id, AVG(titles.price)

FROM titles INNER JOIN publishers ON titles.pub\_id = publishers.pub\_id

WHERE publishers.state = 'CA'

GROUP BY titles.pub\_id

HAVING AVG(price) > 10

ORDER BY titles.pub\_id;  
Order of execution:- FROM🡪WHERE🡪GROUP BY🡪HAVING🡪SELECT🡪ORDER BY

1. **Difference between GROUP BY and PARTITION BY?**1. In Group By Clause:- reduces the no. of records.

In Partition By Clause:- No. of records will not be reduced. Instead of that it will add one extra column.

2. In Group By Clause:- Any non group by column is not allowed in the select clause.

Ex.:- SELECT deptno ,COUNT(\*) DEPT\_COUNT FROM emp GROUP BY deptno;

In Partition By Clause:- Any non group by column is allowed in the select clause.

Ex.:- SELECT empno, deptno ,COUNT(\*) OVER (PARTITION BY deptno) DEPT\_COUNT FROM emp;

3. In Group By Clause:- Group by actually groups the result set returning one row per group.

In Partition By Clause:- Analytic functions (Partition by) give aggregate result they do not group the result set means They return the group value multiple times with each record.

4. In Group By Clause:- In filter condition we need to use having clause instead of where clause.

In Partition By Clause:- We can use where clause in filter condition apart from partition column.

5. In Group By Clause:- In select we need to use only columns which are used in group by. but we can use aggregate functions.

In Partition By Clause:- In select we can use N no. of columns. No restrictions.

Ex:- Select caller ,count(\*) as outgoing from callerlog group by caller;

CALLER OUTGOING

123 3

456 2

567 1

678 1

789 1

select caller ,count(\*) over (partition by caller ) as outgoing from callerlog;

CALLER OUTGOING

123 3

123 3

123 3

456 2

456 2

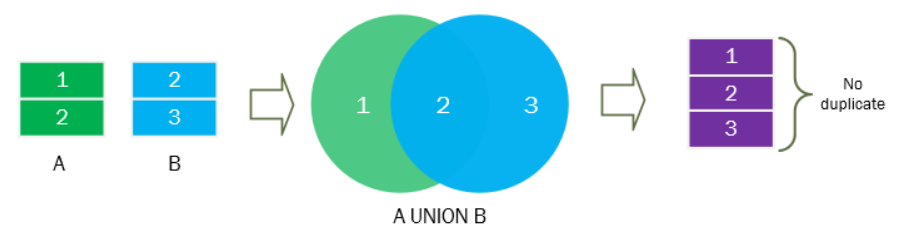
567 1

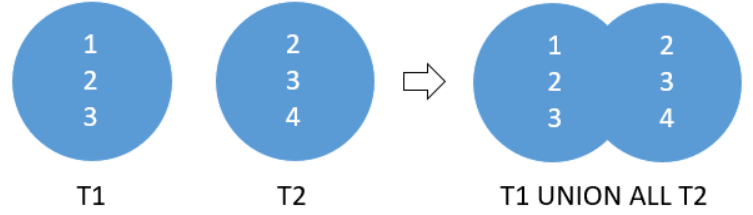
678 1

789 1

1. **Difference between GROUP BY and OVER clause?**

???

1. **What are Set Operations?**Union, Union All, Intersect, Minus.  
   



\* UNION contains distinct records but UNION ALL contains duplicate records as well.

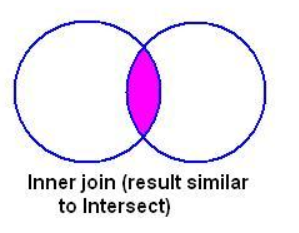
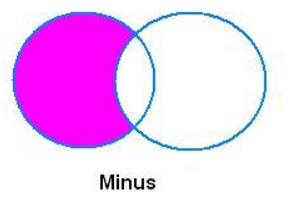
\* If there are multiple blocks of UNION then we can use ORDER BY in the last block of UNION.  
SELECT FIRST\_NAME FROM TEST

UNION

SELECT MIDDLE\_NAME FROM TEST

UNION

SELECT LAST\_NAME FROM TEST ORDER BY 1;

1. **What is Joins?**JOIN is used to combine rows from two or more tables, based on a common field between them.
2. **Types of join?**

Two types of Join:-

1. Equi Join:- (i) Inner Join [ NATURAL JOIN/INNER JOIN / JOIN ]

(ii) Left Outer Join (LEFT OUTER JOIN / LEFT JOIN)

(iii) Right Ouer Join (RIGHT OUTER JOIN / RIGHT JOIN)

(iv) Full Outer Join

2. Non-equi Join  
**1) EQUI JOIN :-** The SQL EQUI JOIN is a simple SQL join uses the equal sign(=) as the comparison operator for the condition.

SQL EQUI JOIN performs a JOIN against equality or matching column(s) values of the associated tables. An equal sign (=) is used as comparison operator in the where clause to refer equality.

SELECT \* FROM Customers,Orders WHERE Customers.CustomerID=Orders.CustomerID;

You may also perform EQUI JOIN by using JOIN keyword followed by ON keyword and then specifying names of the columns along with their associated tables to check equality.

SELECT \* FROM Customers JOIN Orders on Customers.CustomerID=Orders.CustomerID;   
**2) NON EQUI JOIN :**

The SQL NON EQUI JOIN uses comparison operator instead of the equal(=) sign like >, <, >=, <= along with conditions.

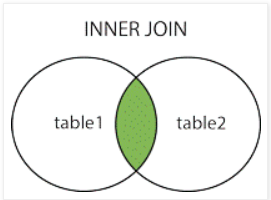
SELECT \* FROM table\_name1, table\_name2 WHERE table\_name1.column > table\_name2.column;  
SELECT \* FROM Customers c,Orders o where c.CustomerID between o.CustomerID and o.CustomerID;

1. **Types of Equi joins?  
   (i) INNER JOIN:-** Matching records from both table.

**(ii) LEFT OUTER JOIN:-** Matching records from both table + Non-matching from Left table.

**(iii) RIGHT OUTER JOIN:-** Matching records from both table + Non-matching from Right table.

**(iv) FULL OUTER JOIN:-** All rows from both tables.(Matching records from both table+ Non-matching from Left table+ Non-matching from Right table)  
  
**(i) Inner Join [ NATURAL JOIN/INNER JOIN / JOIN ]:-**

Inner join returns matching rows from both the tables.  


SELECT Customers.CustomerName, Customers.CustomerID, Orders.OrderID FROM Customers

INNER JOIN Orders ON Customers.CustomerID=Orders.CustomerID;

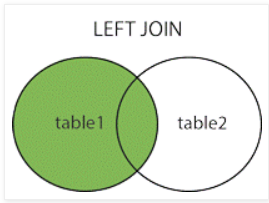
or/otherwise

Select \* from Customers natural join Orders;

or/otherwise

Select \* from Customers join Orders using (CustomerID);

or/otherwise

Select \* from Customers join Orders on (Customers.CustomerID =Orders.CustomerID );  
**(ii) Left Outer Join (LEFT OUTER JOIN / LEFT JOIN)**:-  
  
Left join return rows which are common between the tables and all rows of Left hand side table. Simply, it returns all the rows from Left hand side table even though there are no matches in the Right hand side table.

The LEFT JOIN keyword returns all rows from the left table (table1), with the matching rows in the right table (table2). The result is NULL in the right side when there is no match.

SELECT Customers.CustomerName, Customers.CustomerID,Orders.OrderID FROM Customers

LEFT OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID;

or/otherwise

SELECT Customers.CustomerName, Orders.OrderID FROM Customers

LEFT JOIN Orders ON Customers.CustomerID=Orders.CustomerID;

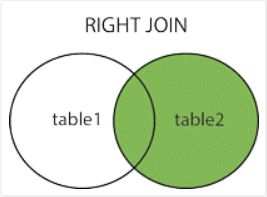
or/otherwise

Select Customers.CustomerID, Customers.CustomerName, Orders.OrderID from Customers,Orders

where Customers.CustomerID =Orders.CustomerID (+);

or/otherwise

SELECT \* FROM Customers LEFT OUTER JOIN Orders USING(CustomerID);

**(iii) Right Ouer Join (RIGHT OUTER JOIN / RIGHT JOIN)**  
Right join return rows which are common between the tables and all rows of Right hand side table. Simply, it returns all the rows from the right hand side table even though there are no matches in the left hand side table.

The RIGHT JOIN keyword returns all rows from the right table (table2), with the matching rows in the left table (table1). The result is NULL in the left side when there is no match.

SELECT Customers.CustomerName, Customers.CustomerID, Orders.OrderID FROM Customers

RIGHT OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID;

or/otherwise

SELECT Orders.OrderID, Employees.FirstName FROM Orders

RIGHT JOIN Employees ON Orders.EmployeeID=Employees.EmployeeID;

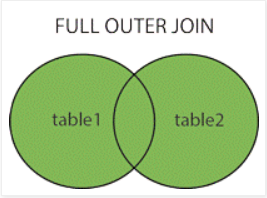
or/otherwise

Select Customers.CustomerID, Customers.CustomerName, Orders.OrderID from Customers,Orders

where Customers.CustomerID (+)=Orders.CustomerID ;

or/otherwise

SELECT \* FROM Customers RIGHT OUTER JOIN Orders USING(CustomerID);

**(iv) Full Outer Join**Full join return rows when there are matching rows in any one of the tables. This means, it returns all the rows from the left hand side table and all the rows from the right hand side table.

The FULL OUTER JOIN keyword returns all rows from the left table (table1) and from the right table (table2).

The FULL OUTER JOIN keyword combines the result of both LEFT and RIGHT joins.

SELECT Customers.CustomerName, Customers.CustomerID, Orders.OrderID FROM Customers

FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID;

or/otherwise

SELECT Customers.CustomerName, Orders.OrderID FROM Customers

FULL JOIN Orders ON Customers.CustomerID=Orders.CustomerID;

or/otherwise

SELECT \* FROM Customers FULL OUTER JOIN Orders USING(CustomerID);

1. **What is CROSS JOIN/ Cartesian Product?**The CROSS JOIN clause produces the cross-product of two tables. A cross join or Cartesian product is formed when every row from one table is joined to all rows in another. Suppose, the source and target tables have four and three rows, respectively, a cross join between them results in (4 × 3 = 12) rows being returned provided by there is no WHERE clause have been applied with the cross join statement.

Select \* from Customers cross join Orders;

1. **What is SELF JOIN?**  
   A self join is a join in which a table is joined with itself.

To join a table itself means that each row of the table is combined with itself and with every other row of the table.

The table appears twice in the FROM clause and is followed by table aliases that qualify column names in the join condition.

The self join can be viewed as a join of two copies of the same table. The table is not actually copied, but SQL performs the command as though it were.  
Select e.e\_id, e.mgr, m.e\_id, e.e\_nm ||' works for manager '|| m.e\_nm from emp e join emp m

on (e.mgr=m.e\_id);

1. **What is Antijoins?**   
   An antijoin between two tables returns rows from the first table where no matches are found in the second table. Anti-Joins are only available when performing a NOT IN sub-query.

SELECT \* FROM emp\_mast WHERE dept\_no NOT IN (SELECT dept\_no FROM dep\_mast);

1. **What is Semijoins?**A semi-join is such a join where the EXISTS clause is used with a subquery. It can be called a semi-join because even if duplicate rows are returned in the subquery, only one set of matching values in the outer query is returned.

SELECT \* FROM dep\_mast a WHERE EXISTS (SELECT \* FROM emp\_mast b WHERE a.dept\_no = b.dept\_no);

1. **Join with third table?**SELECT a.ord\_num,b.cust\_name,a.cust\_code, c.agent\_code,b.cust\_city FROM agents c,customer b,orders a

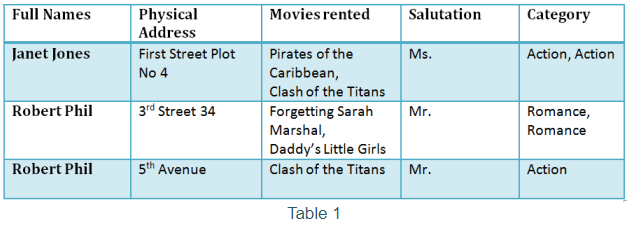
WHERE b.cust\_city=c.working\_area AND a.cust\_code=b.cust\_code AND a.agent\_code=c.agent\_code;

1. **Can Join compare NULL values?**In Oracle Null = Null is evaluated to False.
2. **What is Normalization?**Normalization is a database design technique which organizes tables in a manner that reduces redundancy and dependency of data.

It divides larger tables to smaller tables and links them using relationships.

1. **Types of Normalization?**In most practical applications, normalization achieves its best in 3rd Normal Form.

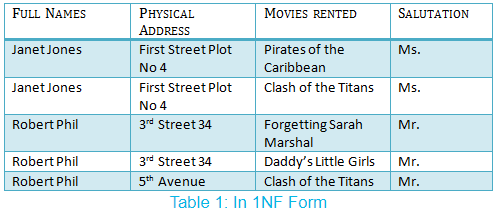
1NF->2NF->3NF->BCNF->4NF->5NF



**1NF (First Normal Form) Rules:-**

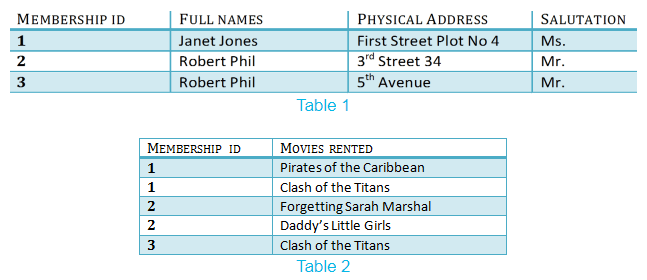
Rule 1- Each table cell should contain a single value.

Rule 2- Each record needs to be unique.



**2NF (Second Normal Form) Rules:-**

Rule 1- Be in 1NF.

Rule 2- Single Column Primary Key.  
  
  
We have introduced a new column called Membership\_id which is the primary key for table 1.

Records can be uniquely identified in Table 1 using membership id

In Table 2, Membership\_ID is the Foreign Key.  
**3NF (Third Normal Form) Rules:-**

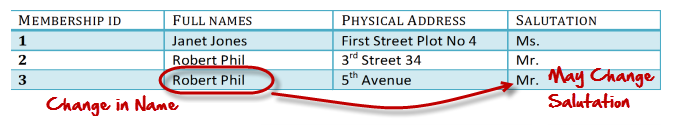
Rule 1- Be in 2NF.

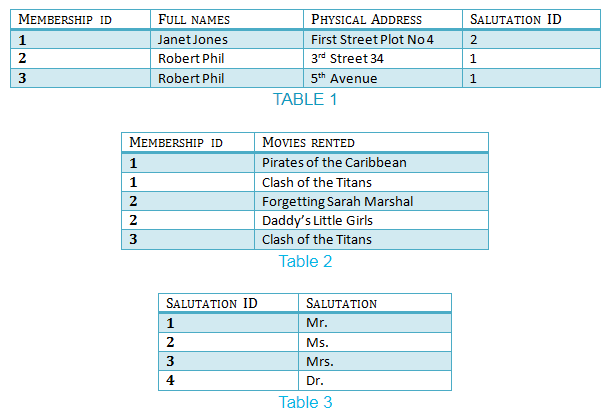
Rule 2- Has no transitive functional dependencies.

What are transitive functional dependencies?

A transitive functional dependency is when changing a non-key column, might cause any of the other non-key columns to change

Consider the table 1. Changing the non-key column Full Name may change Salutation-





We have again divided our tables and created a new table which stores Salutations.

There are no transitive functional dependencies, and hence our table is in 3NF

In Table 3 Salutation ID is primary key, and in Table 1 Salutation ID is foreign to primary key in Table 3.

**Boyce-Codd Normal Form (BCNF):-** Even when a database is in 3rd Normal Form, still there would be anomalies resulted if it has more than one Candidate Key.

Sometimes is BCNF is also referred as 3.5 Normal Form.

**4NF (Fourth Normal Form) Rules:-** If no database table instance contains two or more, independent and multivalued data describing the relevant entity, then it is in 4th Normal Form.

**5NF (Fifth Normal Form) Rules:-** A table is in 5th Normal Form only if it is in 4NF and it cannot be decomposed into any number of smaller tables without loss of data.

1. **Difference between 2NF and 3NF?**See answer in above question Types of Normalization?
2. **Difference between Normalization and De-normalization?**1) Normalization is the process of dividing the data into multiple tables, so that data redundancy and data integrities are achieved.

De-Normalization is the opposite process of normalization where the data from multiple tables are combined into one table, so that data retrieval will be faster.

2) Normalization removes data redundancy i.e.; it eliminates any duplicate data from the same table and puts into a separate new table.

De-Normalization creates data redundancy i.e.; duplicate data may be found in the same table.

3) Normalization maintains data integrity i.e.; any addition or deletion of data from the table will not create any mismatch in the relationship of the tables.

De-Normalization may not retain the data integrity.

4) Normalization increases the number of tables in the database and hence the joins to get the result.

De-Normalization reduces the number of tables and hence reduces the number of joins. Hence the performance of the query is faster here compared to normalized tables.  
5) Normalization is used in **OLTP** system, which emphasizes on making the insert, delete and update anomalies faster.

As against, Denormalization is used in **OLAP** system, which emphasizes on making the search and analysis faster.

1. **Oracle SQL Query Writing and Performance Tuning?**1. Do not use the set operator UNION if the objective can be achieved through an UNION ALL. UNION incurs an extra sort operation which can be avoided.

2. Select ONLY those columns in a query which are required. Extra columns which are not actually used, incur more I/O on the database and increase network traffic.

3. Do not use the keyword DISTINCT if the objective can be achieved otherwise. DISTINCT incurs an extra sort operation and therefore slows your queries down.

4. Avoid doing an ORDER BY on a large data set especially if the response time is important.

5. When writing sub-queries make use of the EXISTS operator where possible as Oracle knows that once a match has been found it can stop and avoid a full table scan (it does a SEMI JOIN).

6. If the selective predicate is in the sub query, then use IN.

7. If the selective predicate is in the parent query, then use EXISTS.

8. Use equi-joins whenever possible, they improve SQL efficiency.

9. Avoid the following kinds of complex expressions:

\* NVL (col1,-999) = ….

\* TO\_DATE(), TO\_NUMBER(), and so on

These expressions prevent the optimizer from assigning valid cardinality or selectivity estimates and can in turn affect the overall plan and the join method.

10. If Query requires quick response rather than good throughput is the objective, try to avoid sorts (group by, order by, etc.). For good throughput, optimizer mode should be set to ALL ROWS.

11. Do not modify indexed columns with functions such as RTRIM, TO\_CHAR, UPPER, TRUNC as this will prevent the optimizer from identifying the index. If possible perform the modification on the constant side of the condition. If the indexed column is usually accessed through a function (e.g NVL), consider creating a function based index.

12. Try to use an index if less than 5% of the data needs to be accessed from a data set. The exception is a small table (a few hundred rows) which is usually best accessed through a FULL table scan irrespective of the percentage of data required.

13. If it is required to use a composite index, try to use the “Leading” column in the “WHERE” clause. Though Index skip scan is possible, it incurs extra cost in creating virtual indexes and may not be always possible depending on the cardinality of the leading columns.

14. There should not be any Cartesian product in the query unless there is a definite requirement to do so. I know this is a silly point but we all have done this mistake at one point.

15. If possible use bind variables instead of constant/literal values in the predicate filter conditions to reduce repeated parsing of the same statement.

16. Use CASE statements instead of DECODE (especially where nested DECODEs are involved) because they increase the readability of the query immensely.

17. Do not use HINTS unless the performance gains clear.

18. Consider using the PARALLEL hint (only when additional resources can be allocated) while accessing large data sets.

1. **Performance Tuning like Indexes, Hints, Partitioning?**
2. **What is Parallel hint in Oracle?**Parallel SQL enables a SQL statement to be processed by multiple threads or processes simultaneously.

Parallel processing can improve the performance of suitable SQL statements to a degree that is often not possible by any other method.

Query:-

SELECT /\*+ PARALLEL(MATURITY\_ID,8)\*/ MATURITY\_ID FROM PIT\_STG.MATURITY\_RATING WHERE MATURITY\_ID=800238793;

1. **What is Query Optimization?**A single query can be executed through different algorithms or re-written in different forms and structures.

Hence, the question of query optimization comes into the picture – Which of these forms or pathways is the most optimal?

The query optimizer attempts to determine the most efficient way to execute a given query by considering the possible query plans.

Importance: The goal of query optimization is to reduce the system resources required to full-fill a query, and ultimately provide the user with the correct result set faster.

First, it provides the user with faster results, which makes the application seem faster to the user.

Secondly, it allows the system to service more queries in the same amount of time, because each request takes less time than unoptimized queries.

Thirdly, query optimization ultimately reduces the amount of wear on the hardware (e.g. disk drives), and allows the server to run more efficiently (e.g. lower power consumption, less memory usage).

A query optimizer is a critical database management system (DBMS) component that analyzes Structured Query Language (SQL) queries and determines efficient execution mechanisms. A query optimizer generates one or more query plans for each query, each of which may be a mechanism used to run a query. The most efficient query plan is selected and used to run the query.

Database users do not typically interact with a query optimizer, which works in the background.

**Query Optimizer-**

SQL queries may be simple or complex statements. Each SQL statement requires minimal use of valuable resources, such as disk reads and server memory. The query optimizer ensures this, as well as expedited execution of each SQL query. For example, a query optimizer may generate a series of query plans based on resource costs. One query plan may involve reading a table to retrieve a subset of its data, while another may involve using table indexes for quick data reading. These are known as cost-based optimizers.

A query optimizer may select different query plans for the same query, depending on environmental circumstances. For example, a user runs a query that selects approximately half of a table's data. The user runs the query when the server is heavily tasked with multiple simultaneous connections. In this scenario, the query optimizer may decide to use a query plan that calls on the created table indexes to satisfy the query, based on limited resources. This ensures minimal server drain by the query. By running the same query at another time with more resources, the query optimizer could determine that resource limitation is not an issue. In this instance, table indexes would not be used, and the query optimizer would allow full table loading to server memory.

1. **What are Exceptions (user defined and oracle raised)?**
2. **What is Database partitioning?**Partitioning is powerful functionality that allows tables, indexes, and index-organized tables to be subdivided into smaller pieces, enabling these database objects to be managed and accessed at a finer level of granularity. Oracle provides a comprehensive range of partitioning schemes to address every business requirement. Moreover, since it is entirely transparent in SQL statements, partitioning can be used with any application, from packaged OLTP applications to data warehouses.

**When to Partition a Table?**

Tables greater than 2 GB should always be considered as candidates for partitioning.

Tables containing historical data, in which new data is added into the newest partition. A typical example is a historical table where only the current month's data is updatable and the other 11 months are read only.

When the contents of a table need to be distributed across different types of storage devices.

**Key Benefits of Partitioning:**

\* Increases performance by only working on the data that is relevant.

\* Improves availability through individual partition manageability.

\* Decreases costs by storing data in the most appropriate manner.

\* Is easy as to implement as it requires no changes to applications and queries.

\* Is a mature, well proven feature used by thousands of Oracle customers.

**Types of Partitioning:-**

**1. Range partitions:-** The data is distributed based on a range of values.

A table that is partitioned by range is partitioned in such a way that each partition contains rows for which the partitioning expression value lies within a given range.

CREATE TABLE EMPLOYEE(EMP\_NO NUMBER(2),EMP\_NAME VARCHAR(2)) PARTITION BY RANGE(EMP\_NO) (PARTITION P1 VALUES LESS THAN(100), PARTITION P2 VALUES LESS THAN(200), PARTITION P3 VALUES LESS THAN(300),PARTITION P4 VALUES LESS THAN(MAXVALUE));

INSERT INTO EMPLOYEE VALUES(101,’A’); -- THIS WILL GO TO P1

INSERT INTO EMPLOYEE VALUES(201,’B’); -- THIS WILL GO TO P2

SELECT \*FROM EMPLOYEE PARTITION(P1);

**2. List partitions:-** The data distribution is defined by a discrete list of values. One or multiple columns can be used as partition key.

List partitioning enables you to explicitly control how rows map to partitions by specifying a list of discrete values for the partitioning key in the description for each partition.

CREATE TABLE EMPLOYEE (EMP\_NO NUMBER(2),EMP\_NAME VARCHAR(2)) PARTITION BY LIST(EMP\_NO) (PARTITION P1 VALUES(1,2,3,4,5), PARTITION P2 VALUES(6,7,8,9,10),PARTITION P3 VALUES(11,12,13,14,15), PARTITION P4 VALUES(16,17,18,19,20));

INSERT INTO EMPLOYEE VALUES(4,’XXX’); -- THIS WILL GO TO P1

INSERT INTO EMPLOYEE VALUES(8,’YYY’); -- THIS WILL GO TO P2

SELECT \*FROM EMPLOYEE PARTITION(P1);

**3. Hash partitions:-** An internal hash algorithm is applied to the partitioning key to determine the partition.

Hash partitioning maps data to partitions based on a hashing algorithm that Oracle applies to the partitioning key that you identify.

CREATE TABLE EMPLOYEE(EMP\_NO NUMBER(2),EMP\_NAME VARCHAR(2)) PARTITION BY HASH(EMP\_NO) PARTITIONS 5;

Here oracle automatically gives partition names like sys\_p1, sys\_p2.

INSERT INTO EMPLOYEE VALUES(5,’A’);

INSERT INTO EMPLOYEE VALUES(8,’B’);

SELECT \*FROM EMPLOYEE PARTITION(SYS\_P2);

1. **Difference between Explain plan and Execution plan?**The plan for a SQL statement is a set of instructions. This tells the database how to access the data and join it together.

Plans come in two varieties:

\* Explain Plan F10

\* Execution Plan

An explain plan predicts how Oracle will process your query.

An execution plan describes the steps it actually took.

The EXPLAIN PLAN statement displays execution plans chosen by the Oracle optimizer for SELECT, UPDATE, INSERT, and DELETE statements. A statement's execution plan is the sequence of operations Oracle performs to run the statement.

The row source tree is the core of the execution plan. It shows the following information:

\* An ordering of the tables referenced by the statement

\* An access method for each table mentioned in the statement

\* A join method for tables affected by join operations in the statement

\* Data operations like filter, sort, or aggregation

\* In addition to the row source tree, the plan table contains information about the following:

\* Optimization, such as the cost and cardinality of each operation

\* Partitioning, such as the set of accessed partitions

\* Parallel execution, such as the distribution method of join inputs

The EXPLAIN PLAN results let you determine whether the optimizer selects a particular execution plan, such as, nested loops join. It also helps you to understand the optimizer decisions, such as why the optimizer chose a nested loops join instead of a hash join, and lets you understand the performance of a query.

1. **SQL Query is case sensitive or not?**SQL is NOT case sensitive but Data in sql query is case sensitive.
2. **How to select new table structure from existing table without coping data?**CREATE TABLE NEW\_EMPLOYEE AS SELECT \* FROM EMPLOYEE WHERE 1=2;

**Note:-** CREATE TABLE NEW\_EMPLOYEE AS SELECT \* FROM EMPLOYEE WHERE 1=1; --coping data as well.

1. **Can we insert multiple NULLs in UNIQUE constraints column?**Yes we can insert multiple NULL values in UNIQUE key constraint.
2. **How many ways to create table in oracle?**CREATE TABLE EMPLOYEE (EMPNO NUMBER(5) PRIMARY KEY,NAME VARCHAR2(20),HIREDATE DATE);  
   CREATE TABLE NEW\_TABLE AS SELECT \* FROM EMPLOYEE; --Copy table structure with data.  
   CREATE TABLE NEW\_TABLE AS SELECT \* FROM EMPLOYEE WHERE 1=1; --Copy table structure with data.  
   CREATE TABLE NEW\_TABLE AS SELECT \* FROM EMPLOYEE WHERE 1=2; --Copy table structure only.
3. **How many ways to insert records in the table?**INSERT INTO EMPLOYEES VALUES(101,'Mohit');  
   INSERT INTO NEW\_TABLE SELECT \* FROM EMPLOYEE WHERE EMPLOYEE\_ID=101;  
   INSERT ALL INTO EMPLOYEE VALUES(101,'MOHIT')

INTO EMP VALUES(102,'RAM')

SELECT \* FROM DUAL;

1. **What is the max size of table?**Maximum 1000 columns created per table.
2. **How to find number of rows in table other than count(\*)?**select count(\*) from emp;

select count(1) from emp;

1. **Versions of Oracle?**  
   Oracle 8i,9i,10g,11g,12c. (i-INTERNET , g-grid architecture, c-cloud)
2. **Difference between 11g and 12c?**
3. **Difference between Primary Key and Unique Key?**

Primary key and unique key are entity integrity constraints. The main difference between the two keys is that there can be multiple unique key columns with null values, whereas that there can only be one primary key column with no null values on a table.

1. **Difference between UNIQUE Constraint and UNIQUE index?**

When you create a UNIQUE constraint, the database automatically creates a UNIQUE index.

1. **Difference between Primary Key and UNIQUE Constraint?**

Primary Key constraints are not nullable. UNIQUE constraints may be nullable.

Per table you may only have one Primary Key but you may define more than one UNIQUE constraints.

1. **Difference between Primary Key and UNIQUE INDEX?**

A primary key also implies NOT NULL, but a unique index can be nullable.

There can be only one primary key, but there can be multiple unique indexes.

If there is no clustered index defined then the primary key will be the clustered index.

1. **Difference between Primary key and Foreign key?**

1) It ensures rows in one table have corresponding rows in another.

2) Foreign keys can be null even though primary keys cannot NULL.

3) Foreign keys cannot be unique but primary keys should by Unique.

1. **What is Global Temporary Table?**

It’s worth mentioning up front that the table itself is not temporary, but rather the data within it. The data in such a table is stored only as long as the session or transaction lasts and is private for each session, however the definition is visible to all sessions. Of course, after commit or disconnection, the data is lost but the definition of the table remains (it’s not necessary to perform many ddl operations – especially create table – which is a good practice). Likewise, other structures related to the temporary table like synonyms or views won’t disappear after the end of the transaction or session. Indexes created on a temporary table behave similarly.

Temporary tables have all of the features that ordinary tables have like triggers or the statistics about table access cost, join cardinality, etc. as well as information about rows and blocks. It’s important to point out that temporary tables can’t have foreign keys related to other temporary/permanent tables.

Creating a Temporary Table

The statement to create a global temporary table (GTT) is similar to the definition of an ordinary table with the addition of the keywords GLOBAL TEMPORARY. In the clause ON COMMIT, you specify if a table is bound to a transaction (DELETE ROWS) or to a session (PRESERVE ROWS).

**Global Temporary Table transaction-specific:-**

CREATE GLOBAL TEMPORARY TABLE table\_name

( column\_name column\_data\_type

...

...

) ON COMMIT DELETE ROWS;

**Global Temporary Table session-specific:-**

CREATE GLOBAL TEMPORARY TABLE table\_name

( column\_name column\_data\_type

...

...

) ON COMMIT PRESERVE ROWS;

Example:-

CREATE GLOBAL TEMPORARY TABLE temp\_users (

id number(5) NOT NULL,

name varchar2(50) NOT NULL,

surname varchar2(50) NOT NULL ) ON COMMIT DELETE ROWS;

INSERT INTO temp\_users VALUES (1, 'John', 'Smith');

INSERT INTO temp\_users VALUES (2, 'Anne', 'Parker');

INSERT INTO temp\_users VALUES (3, 'Kate', 'Doe');

SELECT COUNT(\*) FROM temp\_users;

-- RESULT: 3

COMMIT;

SELECT COUNT(\*) FROM temp\_users;

-- RESULT: 0

1. **Difference between COUNT(\*), COUNT(Expression) and COUNT(DISTINCT Expression)?**

COUNT(\*):- Returns number if rows in a table including duplicates rows and rows containing NULL values in any of the columns.

select count(\*) from employee; --10

COUNT(Expression):- Returns the number of NOT NULL values in the column identified by column.

select count(emp\_no) from employee; --10

COUNT(DISTINCT Expression):- Returns the number of unique, NOT NULL values in the column identified by column.

select count(distinct emp\_no) from employee; --7

1. **What is Execution plan and Explain plan?**The plan for a SQL statement is a set of instructions. This tells the database how to access the data and join it together.

Plans come in two varieties:

1) Execution Plan

2) Explain Plan F10

For Example:-

You’re making a car journey. You plan it beforehand. This is the route you expect to take.

But just as you’re going to leave, you hear on the news that there’s been an accident on your chosen route. This will make it much slower. So you go a different way.

There are two routes here. The one you expected to take and the one you actually took.

After you arrive you wonder whether you could have completed the journey faster. To figure this out, you need to look at where you went. Not where you planned to go.

An explain plan predicts how Oracle will process your query.

An execution plan describes the steps it actually took.

Just as in the driving example above, Oracle may use a different route than the one it predicted.

**1) Execution Plan:-**

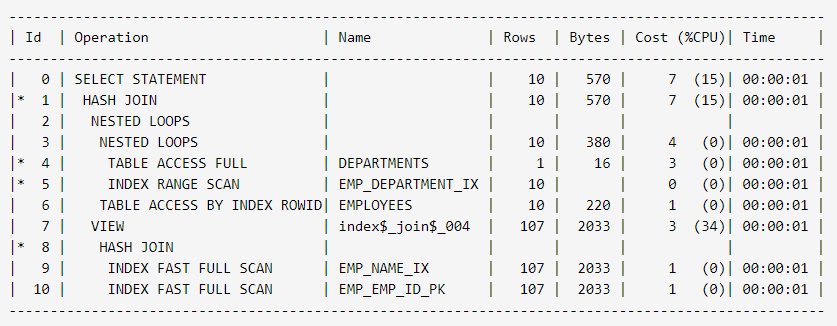
There are three simple rules for reading execution plans-

1. The first operation, or starting point, is the first leaf node, when reading from the top to the bottom. That is, the first element without an indented entry below it. You read from that point backwards.

2. Join operations always require two sets. The order you read the sets is top down, so the first set is the driving set and the second is the probed set. In the case of a nested loop, the first set is the outer loop. In the case of a hash join, the first set is used to build the hash table.

3. One join is performed at a time, so you only need to consider two sets and their join operation at any one time.

Looking at the following execution plan, the order of the operations is 4, 5, 3, 6, 2, 9, 10, 8, 7, 1, 0.

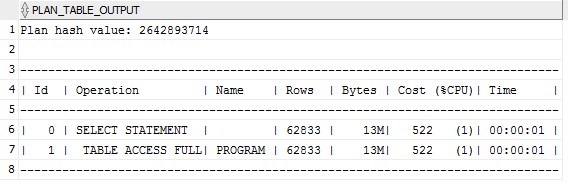


Example:-

EXPLAIN PLAN FOR SELECT \* FROM EDW.PROGRAM;

--Plan FOR succeeded.

SELECT \* FROM TABLE(dbms\_xplan.display);



**2) Explain plan:-**

The EXPLAIN PLAN statement displays execution plans chosen by the Oracle optimizer for SELECT, UPDATE, INSERT, and DELETE statements.

The EXPLAIN PLAN results let you determine whether the optimizer selects a particular execution plan, such as, nested loops join. It also helps you to understand the optimizer decisions, such as why the optimizer chose a nested loops join instead of a hash join, and lets you understand the performance of a query.

A statement's execution plan is the sequence of operations Oracle performs to run the statement.

The row source tree is the core of the execution plan. It shows the following information-

\* An ordering of the tables referenced by the statement

\* An access method for each table mentioned in the statement

\* A join method for tables affected by join operations in the statement

\* Data operations like filter, sort, or aggregation

\* In addition to the row source tree, the plan table contains information about the following:

\* Optimization, such as the cost and cardinality of each operation

\* Partitioning, such as the set of accessed partitions

\* Parallel execution, such as the distribution method of join inputs

Ex.:-

SELECT \* FROM EDW.PROGRAM\_EVENT P

LEFT JOIN EDW.ELKP\_COMMON\_REF EC

ON P.PROGRAM\_EVENT\_CD=EC.LKP\_CD

AND EC.lkp\_type\_cd = 'LKP\_PRGM\_EVENT'

AND P.PROGRAM\_EVENT\_CD = '25453'

LEFT JOIN EDW.PROGRAM PR

ON P.SOURCE\_PROGRAM\_ID=PR.PROGRAM\_ID

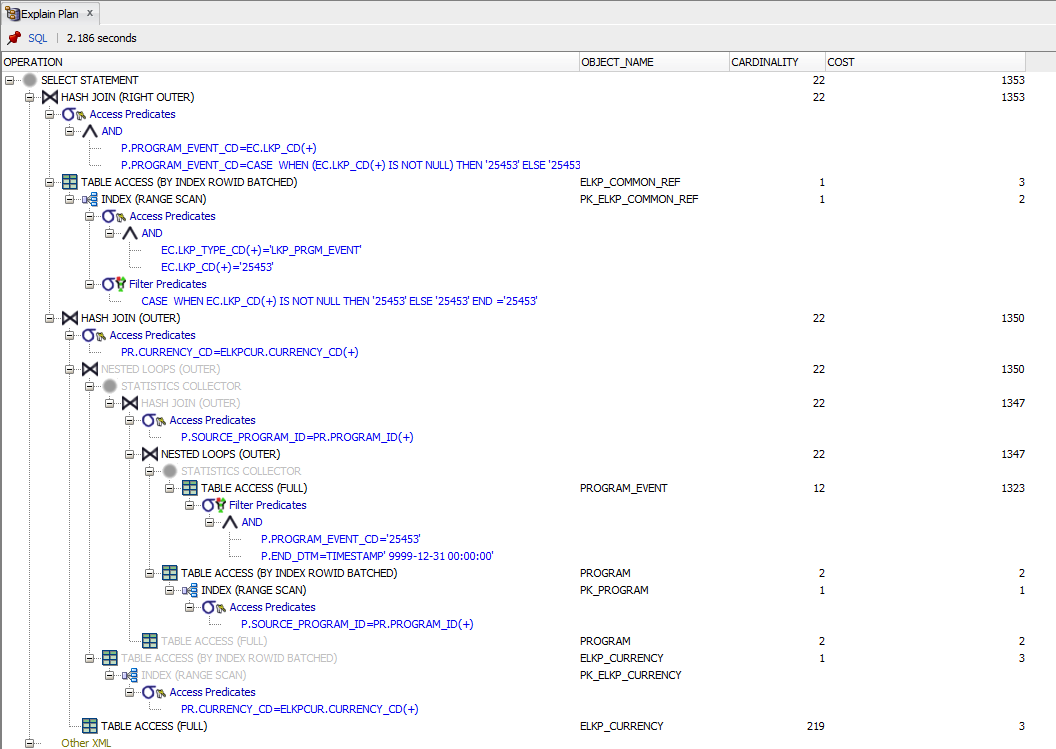
LEFT JOIN EDW.ELKP\_CURRENCY ELKPCUR

ON PR.CURRENCY\_CD=ELKPCUR.CURRENCY\_CD

WHERE P.END\_DTM=to\_date( '31-DEC-9999', 'DD-MON-YYYY' )

AND P.PROGRAM\_EVENT\_CD = '25453';

Press F10 to see Explain Plan.



1. **What is Hint?**A hint is an addition to the SQL standard that instructs the database engine on how to execute the query. For example, a hint may tell the engine to use or not to use an index (even if the query optimizer would decide otherwise).
2. **Types of Hints?**

1. FULL Hint

2. PARALLEL Hint

3. NO\_INDEX Hint

4. FIRST\_ROW Hint

5. ALL\_ROWS Hint

6. ROWID Hint

7. INDEX Hint

**1. FULL Hint:-**

The FULL hint explicitly chooses a full table scan for the specified table.

Table specifies the name or alias of the table on which the full table scan is to be performed. If the statement does not use aliases, then the table name is the default alias.

Example:-

SELECT /\*+ FULL(e) \*/

employee\_id, last\_name FROM employees e WHERE last\_name LIKE :b1;

Oracle performs a full table scan on the employees table to execute this statement, even if there is an index on the last\_name column that is made available by the condition in the WHERE clause.

**2. PARALLEL hint:-**

Parallel SQL enables a SQL statement to be processed by multiple threads or processes simultaneously.

Parallel processing can improve the performance of suitable SQL statements to a degree that is often not possible by any other method.

Query:-

SELECT /\*+ PARALLEL(MATURITY\_ID,8)\*/

\* FROM EDW.PROGRAM;

The PARALLEL hint lets you specify the desired number of concurrent servers that can be used for a parallel operation. The hint applies to the SELECT, INSERT, UPDATE, and DELETE portions of a statement, as well as to the table scan portion.

The PARALLEL hint must use the table alias, if an alias is specified in the query. The hint can then take two values, separated by commas after the table name. The first value specifies the degree of parallelism for the given table, and the second value specifies how the table is to be split among the Oracle Real Application Clusters instances. Specifying DEFAULT or no value signifies that the query coordinator should examine the settings of the initialization parameters to determine the default degree of parallelism. In the following example, the PARALLEL hint overrides the degree of parallelism specified in the employees table definition:

SELECT /\*+ FULL(hr\_emp) PARALLEL(hr\_emp, 5) \*/ last\_name

FROM hr.employees hr\_emp;

In the next example, the PARALLEL hint overrides the degree of parallelism specified in the employees table definition and tells the optimizer to use the default degree of parallelism determined by the initialization parameters. This hint also specifies that the table should be split among all of the available instances, with the of parallelism on each instance.

SELECT /\*+ FULL(hr\_emp) PARALLEL(hr\_emp, DEFAULT,DEFAULT) \*/ last\_name

FROM hr.employees hr\_emp;

**3. NO\_INDEX Hint:-**  
The NO\_INDEX hint explicitly disallows a set of indexes for the specified table.

\* If this hint specifies a single available index, then the optimizer does not consider a scan on this index. Other indexes not specified are still considered.

\* If this hint specifies a list of available indexes, then the optimizer does not consider a scan on any of the specified indexes. Other indexes not specified in the list are still considered.

\* If this hint specifies no indexes, then the optimizer does not consider a scan on any index on the table. This behavior is the same as a NO\_INDEX hint that specifies a list of all available indexes for the table.

The NO\_INDEX hint applies to function-based, B-tree, bitmap, cluster, or domain indexes. If a NO\_INDEX hint and an index hint (INDEX, INDEX\_ASC, INDEX\_DESC, INDEX\_COMBINE, or INDEX\_FFS) both specify the same indexes, then both the NO\_INDEX hint and the index hint are ignored for the specified indexes and the optimizer considers the specified indexes.

For example:-

SELECT /\*+NO\_INDEX(employees emp\_empid)\*/ employee\_id

FROM employees

WHERE employee\_id > 200;

SELECT /\*+ FULL(P) NO\_INDEX(PR)+\*/

\* FROM EDW.PROGRAM\_EVENT P

LEFT JOIN EDW.PROGRAM PR

ON P.SOURCE\_PROGRAM\_ID=PR.PROGRAM\_ID;

**4. FIRST\_ROW Hint:-**  
The hints FIRST\_ROWS(n) (where n is any positive integer) or FIRST\_ROWS instruct Oracle to optimize an individual SQL statement for fast response. FIRST\_ROWS(n) affords greater precision, because it instructs Oracle to choose the plan that returns the first n rows most efficiently. The FIRST\_ROWS hint, which optimizes for the best plan to return the first single row, is retained for backward compatibility and plan stability.

For example, the optimizer uses the cost-based approach to optimize this statement for best response time:

SELECT /\*+ FIRST\_ROWS(10) \*/ employee\_id, last\_name, salary, job\_id

FROM employees

WHERE department\_id = 20;

In this example each department contains many employees. The user wants the first 10 employees of department #20 to be displayed as quickly as possible.

**5. ALL\_ROWS Hint:-**

The ALL\_ROWS hint explicitly chooses the cost-based approach to optimize a statement block with a goal of best throughput (that is, minimum total resource consumption).

For example, the optimizer uses the cost-based approach to optimize this statement for best throughput:

SELECT /\*+ ALL\_ROWS \*/ employee\_id, last\_name, salary, job\_id

FROM employees

WHERE employee\_id = 7566;

**6. ROWID Hint:-**

The ROWID hint explicitly chooses a table scan by rowid for the specified table.

where table specifies the name or alias of the table on which the table access by rowid is to be performed.

For example:

SELECT /\*+ROWID(employees)\*/ \*

FROM employees

WHERE rowid > 'AAAAtkAABAAAFNTAAA' AND employee\_id = 155;

**7. INDEX Hint:-**

The INDEX hint explicitly chooses an index scan for the specified table. You can use the INDEX hint for domain, B-tree, bitmap, and bitmap join indexes. However, Oracle recommends using INDEX\_COMBINE rather than INDEX for bitmap indexes, because it is a more versatile hint.

where table specifies the name or alias of the table associated with the index to be scanned.

index specifies an index on which an index scan is to be performed.

SELECT /\*+ INDEX(patients sex\_index) use sex\_index because there are few

male patients \*/ name, height, weight

FROM patients

WHERE sex = 'm';

SELECT /\*+ INDEX(employee)\*/

\*

FROM employee;

1. **What is CONNECT BY LEVEL?**???
2. **What is FULL TABLE SCAN?**???
3. **What is PIVOT operator?**???