### Joins

Cartesian Product

When the join condition is invalid or is omitted. All rows in the first table are joined to all rows in the 2nd table

**select** \* **from** emp,dept

### or

Select \* from emp, dept where 1>0

or

select empno, ename, dname from emp cross join dept

### Equi Join

Also called simple join/Inner join

The relation between an Employee and Department tables involves as Equi-Join. The values in the common field in both the tables must be equal.

select \* from emp e , dept d where e.deptno=d.deptno

select \* from emp join dept on dept.deptno=emp.deptno

select \* from emp join dept using(deptno);

### LEFT OUTER JOIN

select ename, empno, dname, loc from dept d, emp e where E.Deptno(+)=D.DEPTNO;

For every record in D the corresponding records in E are displayed, including those in D who do not have a corresponding record in E

OR

select ename, empno, dname, loc from dept left outer join emp on dept.deptno=emp.deptno;

All Records from department including those who do not have a corresponding employee;

Or

select deptno, dname, ename from dept d left outer join emp e using(deptno)

select d.deptno, e.empno, e.ename, c.childname from dept d left outer join emp e on d.deptno=e.deptno left outer join empchild c on e.empno=c.empno

### RIGHT OUTER JOIN

select \* from deptmast d, employee e where D.DEPTNO(+)=E.DEPTNO

For every record in E the corresponding records in D are displayed, including those in E who do not have a corresponding record in D

OR

select \* from dept RIGHT outer join emp on dept.deptno=emp.deptno;

All Records from employee including those who do not have a corresponding department;

### FULL OUTER JOIN

select \* from dept FULL outer join emp on dept.deptno=emp.deptno;

OR

select \* from dept d, emp e where E.DEPTNO(+)=D.DEPTNO

union

select \* from dept d, emp e where D.DEPTNO(+)=E.DEPTNO

All Records from both the tables, including those who do not have a corresponding department/employee in the other table;

Non-Equi Join

It involves an operator other than the equality(=) operator. It uses operators like <=, >= or Between.

Example

Employee Sal\_Grade

Empno Empname Salary Grade Min\_Sal Max\_sal

10 hhh 3300 A 3000 5000

23 ggg 4500 B 5001 7500

35 jjj 7800 C 7501 10000

67 lll 8700 D 10001 15000

89 ppp 9600

67 ooo 10200

Select e.empno, e.empname, e.salary, j.grade from employee e, Sal\_Grade j where e.salary BETWEEN j.Min\_Sal and j.Max\_sal

SELECT e.empname , e.salary, s.grade FROM emp e JOIN sal\_grade s ON e.salary BETWEEN s.min\_sal AND s.max\_sal

### Inner Join

select \* from emp e inner join dept d using(deptno)

same as equi join, but the column deptno is not repeated as in euqi join

### Natural Join

SELECT \* FROM STUDENT1 NATURAL JOIN STUDENTS;

Here it will display the records from the 2 tables who have the same values for all the columns with the same name in both the tables. It is a type of EQUI JOIN on the basis of all the columns in the two tables with same names.

Table : DEPT

DEPTNO

DNAME

ADDRESS

Table : EMP

EMPNO

ENAME

DEPTNO

ADDRESS

### UNION

select deptname from deptmast

union

select dept\_desc from dept\_mast

will not display duplicate values

### UNION ALL

select deptname from deptmast

union all

select dept\_desc from dept\_mast

will display all values from both the table including duplicate values

### INTERSECT

select deptname from deptmast

intersect

select dept\_desc from dept\_mast

common values(or intersection) from both the tables

### MINUS

select studname from students

minus

select name1 from student1;

present in 1st table but not in second;

select name1 from student1

minus

select studname from students

present in 1st table but not in second;

**self join**

consider the table emp in scott/tiger

**select** x.ename worker, y.ename manager **from** emp x, emp y **where** x.mgr=y.empno;

select x.ename worker, y.ename manager from emp x join emp y on x.mgr=y.empno;

### IN Operator

select \* from students where studname NOT IN ('Hemant','Laxman')

select \* from students where studname IN ('Hemant','Laxman')

### Like Operator

select \* from students where studname like '\_\_\_m'

select \* from students where studname NOT like '\_\_\_m'

select \* from students where studname like 'He%

### Between

select \* from students where maths between 43 and 70

select \* from students where maths NOT between 43 and 70

### Exists

select deptno,deptname from deptmast d where

exists(select \* from employee e where e.deptno=d.deptno

group by e.deptno

having count(e.deptno)=3)

order by d.deptno

Exists is same as In, but it is a logical operator

### Subqueries

A subquery is a SELECT statement in the WHERE, HAVING or FROM clause of another SELECT statement.

Select \* from students where maths>(select maths from students where studname='Laxman')

The Inner query is also called Nested query or Sub query. The inner or nested or sub query generally executes first and it’s output is used as the input for the outer or main query.

Enclose subqueries inside parenthesis. Oracle 8i onwards “Order by” clause can be used in the inner query, only for inline views(sub query in the FROM part of the main query).

There are two type of comparison operators : single row operators and multiple row operators, and hence there are two types of sub-queries Single row sub-queries(Query that returns only one row from the sub-query) and Multiple-Row sub queries(Query that returns more than one row from the sub-query)

Single-Row comparison operators are =,>,>=,<,<=,<>

Select rollno, firstname from students where std=(select std from students where rollno=101) and div=( select div from students where rollno=101)

The outer and inner queries can get data from different tables.

Group functions can be used in a sub-query

Select rollno, firstname from students where maths=(select max(maths) from students);

HAVING clause with subqueries

Select deptno, min(salary) from employees group by deptno having min(salary)>(select min(salary) from employees where deptno=50);

To find the Department with the lowest average salary

Select deptno, avg(salary) from employees group by deptno having avg(salary)=(select min(avg(salary)) from employees group by deptno);

If the inner query returns more than one row, then it would generate an error. To correct such errors we use the IN, ANY or ALL operators.

If the inner query returns no rows or NULL value, then the outer query returns no rows.

Multiple Row Subqueries or Multiple Row Comparison operators

IN(Equal to any number in the list), ANY(Compares value to each value returned by the subquery), ALL(Compares value to every value returned by the subquery)

select \* from students where studname IN (select studname from students where rollno=45 or rollno=35) ;

To display the employees who are not Programmers and whose salary is less than any of the Programmer

select empno, empname, job, salary from employees where salary<ANY(select salary from employees where job=’Programer’) and job<>’Programmer’;

<ANY means less than the maximum.

>ANY means more than the minimum.

=ANY is equivalent to IN

To display the employees who are not Programmers and whose salary is less than all the Programmer

select empno, empname, job, salary from employees where salary<ALL(select salary from employees where job=’Programer’) and job<>’Programmer’;

<ALL means less than the minimum.

>ALL means more than the maximum.

### VIEWS

Views are virtual tables. It actually contains a query. It is a pointer to the underlying base table(s). It does not take any memory in the database. When a table is dropped, the views on it are not accessible

To restrict access of data because the view can display selective columns from table/tables. To present different views of the same data

To make complicated queries simpler by storing them as a view.

Simple views:

Derives data from a single table. Dose not contain calculated columns or expressions, group-functions, is not a join. Allows you to perform DML operations, subject to non-violation of constraints in the base table

Complex views :

A view is called a complex view when it is join or contains calculated columns or expressions or contains group functions. Allows restricted DML operations depending upon the complexity of the view

create or replace view v1

as

select \* from students where maths>70;

select \* from v1;

update students set english=70 where rollno=1;

select \* from v1;

insert into v1(rollno, studname) values(29,'Pappu');

values can be inserted into the base table through views

Cannot update or delete records which cannot be seen through the view.

select \* from v1;

When select is run on a view it will show those records which satisfy the condition.

select \* from students;

select view\_name, text from user\_views where view\_name='V12'

create or replace view V2 as select \* from students order by rollno;

insert into stud2(rollno, studname) values(31,'Sunil');

cannot insert values in this view because of the order by clause. Update and Delete allowed.

create or replace view v1(employee\_no, employee\_name, sal) as select empno, empname, salary from employee where salary<=500

-to add an alias to each column.

Complex view :

create view v2(name, minsal, maxsal, avgsal) as select d.deptname, min(e.salary), max(e.salary), avg(e.salary) from employee e, deptmast d   
where e.deptno=d.deptno group by d.deptname;

create view v44 as select deptno, sum(sal) as tsal from emp group by deptno

### READ only views

They do not allow to insert/update/delete values into the base table thru the view

Create or replace view v5 as select \* from employee where deptno=1 with read only

**To drop a view**

Drop view viewname

**Inline Views**

An inline view is created by placing a subquery in the FROM clause and giving that subquery an alias so that it behaves like a separate table. Example to find all the employees who earn a salary less than the maximum salary in that department.

Select a.empname, a.salary, a.deptno, b.maxsal from employee a, (select deptno, max(salary) maxsal from employee group by deptno) b where a.deptno=b.deptno and a.salary<b.maxsal

**Top N analysis**

select rownum as rank, empno, sal from (select empno, salary from emp where sal is not null order by sal desc) where rownum<=3

Rownum is pseudo column assigned to each row returned by the inner query. The inner query contains the order by clause to get the ranking in the desired order. The outer query limits the number of rows returned by using the where clause which uses a < or <=operator

**commit and rollback**

How to commit

1) commit;

Implicit Commit

2) A DDL Command : Create, Drop, Alter, Audit

3) When you say :Exit or Quit; Connect or Disconnect

How to Rollback

1) Rollback;

Implicit Rollback

3) when you switch off;

4) hard disk crash

5) Power failure

### Index

It is a schema object for faster data access or speed up retrieval of rows by using a pointer. Indexes are maintained automatically by the Oracle Server. It reduces disk I/O by using a rapid path access method

If you do not have an index on a column, then a full table scan occurs.

Indexes are created 1)Automatically : A unique index is created automatically when you define a Primary Key or a Unique constraint in a table definition. The name of the index is the name given to the constraint. Indexes created by creation of PK/UK cannot be dropped, since they are required for enforcement of the PK/UK constraint 2) Manually : Users can create non-unique indexes on columns to speed up access to the rows. You may also manually create unique indexes by specifying the keyword Unique at the time of index-creation. Manually created unique indexes do not create a corresponding constraint.

Indexes are logically and physically independent of the table that they index. Hence, indexes can be created or dropped at any time and have no effect on the base tables or other indexes.

When you drop a table, corresponding indexes are also dropped.

ORA cannot be forced to use indexes

Suppose you have the following table

Rowid Empno Name Salary

1 5 aaa

2 2 bbb

3 3 ggg

4 7 mmm

5 9 pon

6 1 lon

7 4 jhg

8 6 jkl

create index empno\_idx on emp(empno)

then the internal storage is as follows

select \* from user\_indexes

Rowid Empno Name Salary

6 1

2 2

3 3

7 4

1 5

8 6

4 7

5 9

Now if u use a select command like

Select \* from emp where empno=4

The search will start from the 4th record onwards and not the 1st record, as empno 4 occurs at the 4th record, if there is no index the search starts from the first record

Create Index emp\_last\_name\_idx on employee(last\_name);

Create Index emp\_last\_first\_name\_idx on employee(last\_name, first\_name);

When to create an Index :

A Table is very large and most queries are expected to retrieve less than 2 to 4 percent of rows.

Some of the table’s columns are frequently used together in a WHERE clause of a join condition

A column of a table which is frequently used as a search criteria contains either a large number of NULL values or a wide range of values.

The table is not subject to a lot of DML operations, since every DML operation is followed by updation(of reorgansation of all the indexes on the table)

When not to create an Index :

The table is quite small

The table is frequently updated by DML operations

The columns of the table are not frequently used as a search condition in a query

**SubQueries :**

Create table employees

(empid varchar2(3),

empname varchar2(20),

mgrid varchar2(3),

deptid varchar2(3));

insert into employees values('E1','Hemant',NULL,'10');

insert into employees values('E2','Hetal','E1','10');

insert into employees values('E3','Sheetal','E1','10');

insert into employees values('E4','Ram','E1','10');

insert into employees values('E5','Shyam','E2','10');

insert into employees values('E6','Tom','E2','20');

insert into employees values('E7','Harry','E2','20');

insert into employees values('E8','Ryan','E2','20');

insert into employees values('E9','Sheryl','E1','20');