**Sub Queries, Join and Indexing**

♣ sub queries:- use, example

♣ Set Operations- Union, Union all, Minus and Intersect

♣ MySQL join: Advantages and disadvantages of Join, Types of Joins

♣ Indexing: Advantages and disadvantages of Indexing, creating index(simple, composite, unique),

multiple indexing, drop index

♣ MySQL View: why view, Create, Alter and Drop view

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

* **MySQL View**
* A view is a virtual table. A view consists of rows and columns like a table.
* We can create a view by selecting fields from one or more tables present in the database. But it does not contain any data. At run time it contains the data from table and after execution it gets free.
* If data is changing in the original table, the same change is reflected in the view. A view can be built on top of a single table or multiple tables & another view.
* View requires no memory to store data but some memory required for storing definition of view.
* View is defined on which table this table is called as Base table.
* If view is already exit the View can be replaced using ‘replace’ keyword.
* We can create forcefully view on without existing the table.
* View only allows a user to retrieve data.

**Types of view:**

* ***Simple view :*** 
  + 1. The simple view is created from only one single table.
    2. A simple view also does not have any groups of data.
    3. Simple view will not have any expression or group function.
    4. All DML allow in simple views.
    5. If table drop then view will not drop but view become invalid. View become valid when table again created with same as dropped table column name.
* ***Complex view:***

1. The complex view is created from than one table and also has group function.
2. A complex view contains groups of data.
3. DML are not allowed.
4. Through view we can insert data into child table not in parent table.

**Why Views?**

* Provides additional level of security by restricting access to a fixed set of rows and columns of a table.
* Hide the data complexity.
* Ease of use and Space savings
* To provide data independence
* To present different views of the same data
* **Creating Views**

Suppose we have EMP and DEPT table. We want to see emp details i.e. department names where they are working. We have to write a long join query.

select e.eno,e.ename,e.sal,e.dno,d.dname,d.loc From emp e, dept d where e.dno=d.dno;

Instead of writing this join query again and again, we can create a view on these tables by using a CREATE VIEW command:

create view vw\_empdept as select e.empno, e.ename, e.sal, e.deptno, d.dname, d.loc from emp e, dept d where e.deptno=d.deptno;

Now to see the employee details and department names:

select \* from vw\_empdept; // This will show same result as join query.

* **View with Check Option Constraint**

Creates view with the constraint that apply on INSERT and UPDATE, DELETE statements.

**Example-**suppose all the employee working in Department No. 10 belongs to accounts department.

create view vw\_accounts\_staff as select eno, ename, dno from emp where dno=10 with check option constraint ica\_accounts\_cnst;

select \* from vw\_accounts\_staff;  
select sum(sal) from vw\_accounts\_staff;  
select max(sal) from vw\_accounts\_staff;

In “vw\_accounts\_staff” view you can insert only rows in these departments number 10.

INSERT statement successfully inserts a row into the EMP table through the vw\_accounts\_staff view: insert into vw\_accounts\_staff values (110, 'ASHI', 10);

If you insert a row with department number 30, this INSERT statement is rolled back and raise error because with check constraints apply on vw\_accounts\_staff view it allow only dept number 10:

insert into vw\_accounts\_staff values(111, 'SAMI', 30);

* **Creating View without Having the Base Table**

You can create a view without base table but once the base table was created then the view is validated.

Create force view vw\_stud as select \* from student;

* **Replacing/Altering Views**
* A view can be drop or recreate. When a view is dropped, all grants of corresponding view rights are revoked from users. After the view is re-created, necessary rights must be regranted.
* A view can be replaced or redefining by CREATE OR REPLACE VIEW statement. This option replaces the current definition of a view, but preserves the present security authorizations.
* If base table was dropped then we will get the message like “view has errors”.
* If base table has been altered but still the view was with the initial definition, we

have to replace the view to affect the changes.

create or replace view vw\_accounts\_staff as select empno, ename, deptno from emp where deptno = 30 with check option constraint ica\_accounts\_cnst;

INSERT statement successfully inserts a row into the EMP table through the accounts\_staff view because we are change view definition:

insert into vw\_accounts\_staff values (199, 'ABID', 30);

* **Dropping Views :** Remove view from database

Example: drop view vw\_accounts\_staff;

* **Views without DML**
* **Read only view:** Create view vw\_dept as select \* from dept with read only;
* View with aggregate functions & group by clause:

Create view vw\_sumsal as select dno, sum(sal) t\_sal from emp group by dno;

* View with distinct: Create view vw\_stud as select distinct no,name from student;
* **Updating Views with DML**

EMP base table that cause the view changes and employee 109's name changes from ASHI to SHRI in the EMP table.

update vw\_empdept set ename='shri' where eno=109;

* **Creating View Join (complex view)**

To provide users with convenient access to information from multiple tables, create a complex view.

create view vw\_emploc as select e.eno, e.ename, e.dno, e.sal, d.dname, d.loc from emp e, dept d  where e.dno=d.dno and d.loc in ('HYD', 'BOM', 'DEL');

* **Listing Information about VIEWS**

To see how many views are there in your database:

select \* from user\_views;

select \* from user\_updatable\_columns where table\_name='vw\_empdept';

* **Advantages of View**
* View provides data security for the same base tables.
* It allows different users to view the same data in different ways at the same time.
* It is used to represent additional information like derived columns.
* It is used to hide complex queries.
* It presents a consistent, unchanged image of the database structure, even if the tables are split or renamed.
* It does not allow direct access to the tables of the data dictionary.
* **Disadvantages of View**
* We cannot use DML operations on View, if there is more than one table.
* When table is dropped view becomes inactive.
* View is a database object, so it occupies the space.
* Without table, view will not work.
* Updation is possible for simple view but not for complex views, they are read only type of views.
* **INDEX**
* SQL INDEX is used to find and retrieve data fast in a table.
* It is like a pointer or index of a book. So, when we want to search some data in a table.
* Index searches very quickly without searching the whole table which improves query performance especially in large tables.
* Indexes need to be created in tables for the columns which are used very frequently for searching (i.e. in WHERE clause)
* When a select statement is fired to search for a particular record, the Oracle engine must first locate the table on the hard disk.
* The oracle engine then performs a sequential search to locate records that matches user-defined criteria as specified in the select.
* Indexing completely independent of the table on which the index is being created.
* This two dimensional matrix will have a single column, which will hold sorted data, extracted from the table columns on which the index is created.
* Another column is called the address field identifies the location of the record in the oracle database.
* Indexes can be created on a single column or a group of columns. When an index is created, it first sorts the data and then it assigns a ROWID for each row.
* When there are thousands of records in a table, retrieving information will take a long time. Indexes are created on columns which are accessed frequently, so that the information can be retrieved quickly.

**Benefits**

* Improves performance and data access is faster.
* Ensures uniqueness.
* Index is that it greatly speeds the execution of SQL statement with search and that refer to the indexed column.
* Index must be updated every time row added to the table.
* When user drops primary key or unique key constraints table, oracle automatically drop indexes on primary key column or unique key column or table itself.
* One disadvantage is it requires additional disk space.

**When to use INDEX**

1. If a table has more than a few thousand rows then index it.
2. Index only simple columns and Index frequently used columns.
3. Try not to create more than two or three indexes on a table.

**Creation of an Index**

An index can be created on one more columns. Based on the number columns included

in the index, an index can be:

**1. Simple index:** The single column index is created on only one column of a table.

This type of index allows inserting duplicate records in the table

**Syntax** Create index <IndexName> ON <TableName> (<ColumnName>);

**Example** Create index indx\_emp on emp (eno);

**2. Composite index:** An index created on more than one column.

**Syntax:** Create index <IndexName> ON <TableName> ColumnName1>,<ColumnName2>);

**Example:** Create index indx\_emp on emp (eno,dno);

Oracle allows the creation of two types of indexes.

1. ***Duplicate Index:*** It is an Index that allows duplicate values for the indexed columns

2. ***Unique Index:*** It is an Index that rejects duplicate values for the indexed columns

* **UNIQUE INDEX:**
* Unique index is created on only one column of a table.
* This type of index does not allow inserting duplicate records in the table
* If duplicate values present in particular column then oracle won’t allow creating index on that column.

**Syntax:** create unique index <index name> on <table> (column nm1,…);

**Example:** create unique index indx\_uid on dept(dname)

* **Bit map index:** Bitmap indexes are normally used to index low [cardinality](http://www.orafaq.com/wiki/Cardinality) [columns](http://www.orafaq.com/wiki/Column) in a warehouse environment.(statics)

**Example:** reate bit map index bmid on emp(deptno);

* **Function based index:** This function is created arithmetic expression on columns of table.

**Example:** Create index fid on emp(sal+comm.);

* **Reverse index:** This is created on columns containing data in sequential order.

**Example:** create index rid on emp(empno) reverse;

* **Descending index:** This is created on columns which are displayed in sorted order.

**Example:** create index dixd on emp(ename asc, sal desc);

* **Cluster Index:** Clustered index physically rearrange the data that users inserts in your tables. A table can have only one clustered index.
* **Non-Cluster Index:** The data is present in arbitrary order, but the logical ordering is specified by the index. The logical order of the index does not match the physical sorted order of the row on the disk. If primary key is a cluster index it must be foreign key is non cluster index.
* **Dropping an Index**

**Syntax:** Drop Index <Index Name>; **Example:** Drop index indx\_emp

* **Sub-query and Nested Sub-queries**

**A subquery is a query within another query** i.e. A query embedded in another query is called a subquery. The outer query is called as **main query** and inner query is called as**subquery.** It is also known as nested query. The subquery can be nested inside a SELECT, INSERT, UPDATE, or DELETE statement or inside another subquery.  The results of an inner query can be passed to the outer query. The main query (outer query) use the subquery result. We can define any number of sub queries with in a query. But the system executes the inner most query first. If we are using relational operators between the queries then the sub query must return a single value. Main / Outer query is called Parent query. Sub / Inner query is called Child query. The parent query uses the rows returned by the child query i.e. sub query.

**Syntax:** select <column\_name> from <table\_name> where <column\_name>

**expression operator** (select column\_name from table\_name where <condition>);

The Following rules using Sub Queries

1. The inner Query must be enclosed in parentheses
2. A subquery must be placed on the right side of the comparison operator.
3. The sub query may not have an order by clause
4. The Order By clause appears at the end of the main select statement
5. Sub queries are always executed from the most deeply but nested is least deeply.

* **Single-Row Operator** (>, =, <=,>=, <>,!=)– sub query must return only one value.
* **Multiple Row Operator** (IN, ANY, ALL)– sub query can return more than one value
* **Types 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 i.e. subquery is related to the outer SQL statement.
* Nested subqueries: Subqueries are placed within another subquery.

## 1. Subqueries with the Select Statement

SQL subqueries are most frequently used with the Select statement.

select \*  from emp where eno in (select eno from emp where salary > 19000);

**2. Subqueries with the INSERT Statement**

In the insert statement, data returned from the subquery is used to insert into another table.

insert into emp\_bkp select \* from emp where eno in (select eno  from emp where salary between 15000 and 16000);

**3. Subqueries with the UPDATE Statement**

When a subquery is used with the Update statement, then either single or multiple columns in a table can be updated.

update emp\_bkp set sal=sal \* 0.25  where dno in(select dno from dept where dname=’Sales’);

**4. Subqueries with the DELETE Statement**

With the DELETE statement to delete the single or multiple rows in a table

delete from emp\_bkp where eno in (select eno from emp where eno>=9);

* **Single Row Subqueries:** In single row sub query, it will return one value.

**Example:** select \* from emp where sal>(select sal from emp where eno=102);

* **Multi Row Subqueries**

It will return more than one value. In such cases we should use operators like, any,

all, in, or, not in, between, comparison operator

**Example:** find all employees whose salaries are equal to the average salary of their dept.

select ename, sal from emp where sal=any(select avg(sal) from emp group by dno);

finds all employees whose salaries are greater than between 12500 and 16000

select \* from emp where sal >all(select sal from emp where sal between 12500 and 16000);

* **Multiple Sub queries**

There is no limit on the number of subqueries included in a where clause. It allows nested query within a sub query.

**Example:** Find second highest salary paid employee.

select \* from emp where sal=(select max(sal) from emp where sal<(select max(sal) from emp));

**Example Sub Queries**

* **List the employee’s Details who belong to the department of “Sales”.**

select \* from emp where job=’Sales’; **and**  select \* from emp where dno=3;

##### Sub Queries: select \* from emp where dno=(select dno from dept where dname=’sales’);

* **List the emp’s details whose salary is greater than the average salary of all the employees**

##### select avg(sal) from emp; and select \* from emp where sal>=8125;

##### Sub Queries:select \* from emp where sal>(select avg(sal) from emp)

**List the details of all the employee’s who earn lowest or highest salary in each department**

select \* from emp where sal in(select min(sal) from emp group by dno);

select \* from emp where sal in(select max(sal) from emp group by dno); **(max salary)**

* **Retrieve those employee records who are working under sales and HR dept.**

select \* from emp where dno in(select dno from dept where dname in('Sales','HR'));

* **Retrieve those employee record who salary is lowest than all employee salary.**

select \* from emp where sal=(select min(sal) from emp);

select \* from emp where sal=(select max(sal) from emp); **(highest salary)**

* **Retrieve those employee records who salary is highest than employee ‘rajkumar’.**

select \* from emp where sal>(select sal from emp where ename='rajkumar');

* **Retrieve that dname in which dname does not have any employee.**

select \* from dept where dno not in(select distinct(dno) from emp);

* **Dept wise min sal**

select \* from emp where sal in(select min(sal) from emp group by dno);

select \* from emp where (dno,sal) in(select dno,min(sal) from emp group by dno);

* **Job wise junior**

select \* from emp where hiredate in(select max(hiredate) from emp group by job)

* **Some another sub queries to find explanation**

select \* from emp where hiredate in(select max(hiredate) from emp group by job)

select \* from emp where hiredate < (select hiredate from emp where ename='amar');

select \* from emp where job <>(select job from emp where ename='amar');

select \* from emp where dno=(select dno from dept where loc='pune');

select \* from emp where hiredate=(select min(hiredate) from emp where dno=3 );

* **ANY operator**

The ANY operator is a [logical operator](http://www.sqltutorial.org/sql-logical-operators/) that compares a value with a set of values returned by a subquery. It’s like [OR operator](https://www.tutlane.com/tutorial/sql-server/sql-or-operator). The ANY operator must be preceded by a [comparison operator](http://www.sqltutorial.org/sql-comparison-operators/) >, >=, <, <=, =, <> and followed by a [subquery](http://www.sqltutorial.org/sql-subquery/).

**Syntax:** select <column\_name> from <table\_name> where <column\_name>

comparison operator **Any**(select <column\_name> from <table\_name>);

* **List All the employee details whose salary is greater than the lowest salary of an employee belonging to department number 20**

select \* from emp where sal > any(select sal from emp where dno=20);

* **List All the employee details whose salary is same any of anil or sanjay**

select \* from emp where sal=any(select sal from emp where ename in('anil','sanjay'));

* **List the employee details of those employees whose salary is greater than any of the managers**

select \* from emp where sal > any(select sal from emp where job=’manager’);

* **ALL operator**

ALL operator is a [logical operator](http://www.sqltutorial.org/sql-logical-operators/) that compares a single value with a single-column set of values returned by a [subquery](http://www.sqltutorial.org/sql-subquery/). It’s like [AND operator](https://www.tutlane.com/tutorial/sql-server/sql-or-operator). The ALL operator must be preceded by a [comparison operator](http://www.sqltutorial.org/sql-comparison-operators/) >, >=, <, <=, =, <> and followed by a [subquery](http://www.sqltutorial.org/sql-subquery/).

**Syntax:** select <column\_name> from <table\_name> where <column\_name>

comparison operator **all**(select <column\_name> from <table\_name>);

* **List the employee names whose salary is greater than the highest salary of all employees belonging to department no 2**

Select \* from emp where sal >all(select sal from emp where dno=2);

* **List the details of the employees earning more than the highest paid managers**

select \* from emp where sal > ALL (select sal from emp where job=’manager’);

select \* from emp where dno=3 and sal>all(select sal from emp where dno=4);

* **SOME operator**

This operator is similar to Any Operator, **SOME** operator is used to compare a value with single column set of values returned by subquery. The **SOME** operator in sql must match at least one value in subquery and that value must be preceded by [comparison operators](https://www.tutlane.com/tutorial/sql-server/sql-comparison-operators).

**Syntax:** select <column\_name> from <table\_name> where <column\_name>

comparison operator **some**(select <column\_name> from <table\_name>);

* **Find details of employee earning highest salary**

select \* from emp where sal=some (select max(sal) from emp);

* **CORRELATED SUBQUERY**
* A correlated subquery is the subquery which value depends on the outer query. The correlated subquery gets executed repeatedly, once for each row that is selected by the subquery.
* Correlated subqueries are used for row-by-row processing. Each subquery is executed once for every row of the outer query.
* A correlated subquery is also known as repeating subquery or synchronized subquery.

**Example:** Find all employees whose salary is greater than average of all the employee’s salaries in their department.

select ename, sal, dno from emp a where a.sal >(select avg(sal) from emp b where a.dno=b.dno) order by dno;

* **Using a correlated sub query in an update**

Give a raise to those employees whose salary is less than their department's average.

update emp a set a.sal = a.sal + 500 where a.sal < (select avg (sal) from emp b where a.dno = b.dno);

* **Using a correlated subquery in a delete**

***Delete the highest earning employees in each department.***

delete from emp a where a.sal = (select max(sal) from emp b where a.dno = b.dno);

select distinct dno from emp e where 5<=(select count(ename)from emp where e.dno=dno);

**Top five salary**

select \* from (select \* from emp order by sal desc) where rownum<=5;

**Write a co-related sub query to retrieve second highest salary**

select \* from emp where sal=(select distinct e.sal from emp e where 2=(select count(distinct Sal) from emp where e.sal<=sal));

* **EXISTS OPERATOR**

Return type of exists is Boolean. It returns TRUE if sub query contains any rows. Exists function is a test for existence. This is a logical test for the return of rows from a query.

* WHERE EXISTS tests for the existence of any records in a sub query.
* EXISTS returns true if the sub query returns one or more records.
* EXISTS is commonly used with correlated sub queries.

**Syntax:** SELECT column-names FROM table-name WHERE EXISTS (SELECT column-name FROM table-name WHERE condition)

The EXISTS operator is used to search for the presence of a row in a specified table

that meets certain criteria.

select ename from emp where exists (select ename from emp where sal > 6500);

* **Suppose we want to display the department numbers which has more than 4 employees.**

select dno,count(\*) from emp group by dno having count(\*) > 4;

* **Retrieve those dno in which dept does not have any employee**

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

* **From the above query can you want to display the names of employees?**

select dno,ename, count(\*) from emp group by dno,ename having count(\*) > 4;

The above query returns nothing because combination of dno and ename never return more than one count. The solution is to use exists which follows.

select dno,ename from emp e1 where exists (select \* from emp e2 where e1.dno=e2.dno group by e2.dno having count(e2.ename) > 4) order by dno,ename;

* **NOT EXISTS OPERATOR**

Sub queries using NOT EXISTS will return true only if the sub query returns no rows from the table.

select dno,ename from emp e1 where not exists (select \* from emp e2 where e1.dno=e2.dno group by e2.dno having count(e2.ename) > 4) order by dno,ename;

**Use a correlated NOT EXISTS subquery to find the departments that have no employees:**

select dno from dept d where not exists (select \* from emp e where e.dno = d.dno);