**5 ways to remove duplicates they are**

**1)rowid**

**2)row\_number()**

**3)self join**

**4)group by**

**5)dense\_rank() or rank()**

**We can keep distinct to prevent duplicates from entering into the data.the above and below are the methods to remove duplicates already present in data**

**. Using dense\_rank()**

SQL > delete from emp where rowid in  
(  
select rid from  
(  
select rowid rid,  
dense\_rank() over(partition by empno order by rowed desc) rn  
from emp  
)  
where rn > 1  
);

Here you can use both rank() and dens\_rank() since both will give unique records when order by rowid.

**To remove duplicates**

**WITH prod AS (select m\_product\_id, upc from m\_product where upc='7094')**

**DELETE FROM m\_productprice B**

**USING prod C**

**WHERE B.m\_product\_id = C.m\_product\_id**

**AND B.m\_pricelist\_version\_id = '1000020';**

delete r1 a where rowid<(select max(rowid) from r1 b where a.sno=b.sno);

WITH CTE AS

(

SELECT call\_id,phone\_key,ROW\_NUMBER() OVER(PARTITION BY call\_id,phone\_key ORDER BY call\_id) AS Rnum

FROM ravi

)

DELETE FROM CTE WHERE Rnum <> 1

All queries combined using a UNION, INTERSECT or EXCEPT operator must have an equal number of expressions in their target lists.

**Query to find the duplicates**

with x as   (select  \*,rn = row\_number()  
            over(PARTITION BY OrderNo,item  order by OrderNo)  
            from    #temp1)  
  
select \* from x  
where rn > 1

**Deleting Duplicate records based on record order**

WITH CTE AS(

   SELECT ReportMonth,MainCategory,category,Reportvalue,

       RN = ROW\_NUMBER()OVER(PARTITION BY reportmonth ORDER BY edw\_create\_date desc)

   FROM edwmaster.dbo.executive\_summary\_r where MainCategory='ACTIVE SAAS ENCOMPASS USERS'

)

DELETE FROM CTE WHERE RN > 1

WITH CTE AS(

   SELECT ReportMonth,MainCategory,category,Reportvalue,

       RN = ROW\_NUMBER()OVER(PARTITION BY reportmonth ORDER BY edw\_create\_date desc)

   FROM edwmaster.dbo.executive\_summary\_r where MainCategory='ACTIVE SAAS ENCOMPASS USERS'

)

DELETE FROM CTE WHERE RN > 1

(rownum,rowid )see liked video youtube

explain plan for select \* from r1;

select \* from table(dbms\_xplan.display());

**Query to find the count of duplicates**

select sno, count(sno)

from r1

group by sno

having count (sno) > 1;

**Query to find the 2nd max in a table**

select max(sno) from r1

where sno<(select max(sno) from r1) order by sno desc;

----

**SELECT** TOP 1 salary **FROM** ( **SELECT** TOP 2 salary **FROM** employees **ORDER** **BY** salary **DESC**) **AS** emp **ORDER** **BY** salary **ASC**

**----**  
**SELECT** salary  **FROM** (**SELECT** salary **FROM** Employee **ORDER** **BY** salary **DESC** **LIMIT** 2) **AS** emp **ORDER** **BY** salary **LIMIT** 1

**Query to find Nth max in a table**

select sno from (select sno ,dense\_rank() over (order by sno desc) as r from r1) where r=3;

**note**:If we want max(i.e highest) keep desc.If u want min(i.e lowest) keep asc.By default asc;

**Query to find duplicates**

select sno from r1 group by sno having count(sno)>1;

**Query to find the duplicates**

1)SELECT \* FROM r1 A WHERE EXISTS ( SELECT 1 FROM r1 WHERE sno = A.sno

AND ROWID < A.ROWID);

2)select \* from r1 a where rowid>(select min(rowid) from r1 b where a.sno=b.sno);

Remove duplicates using row\_number()

delete from r1 where rowid in

(

select rid from

(

select rowid rid,

row\_number() over(partition by sno order by sno) rn

from r1

)

where rn > 1

);

U can also select the duplicate records using the above query

**Pictorial Presentation of Cross Join syntax**



1. A **CROSS JOIN** is a **JOIN** operation that produces the Cartesian product of two tables.

select \* from ravi1 natural join ravi2;

<http://www.w3resource.com/slides/sql-joins-slide-presentation.php>

**Query to join 3 tables at a time**

select \* from ravi1,ravi2,ravi3 where ravi1.no=ravi2.no and ravi1.no=ravi3.no;

**left join of 3 tables**

select \* from ravi2 left join ravi1 on ravi1.no=ravi2.no left join ravi3 on ravi2.no=ravi3.no;

One significant difference between INNER JOIN and NATURAL JOIN is the number of columns returned.

Consider:

TableA TableB

Column1 Column2 Column1 Column3

1 2 1 3

The INNER JOIN of TableA and TableB on Column1 will return

a.column1 a.column2 b.column1 b.column3

1 2 1 3

SELECT \* FROM TableA INNER JOIN TableB USING (Column1)

SELECT \* FROM TableA INNER JOIN TableB ON TableA.Column1 = TableB.Column1

The NATURAL JOIN of TableA and TableB on Column1 will return:

column1 column2 column3

1 2 3

SELECT \* FROM TableA NATURAL JOIN TableB

The repeated column is avoided

Another thing is the column names should be same for natural join otherwise by default it goes to cross join

RANK gives you the ranking within your ordered partition. Ties are assigned the same rank, with the next ranking(s) skipped. So, if you have 3 items at rank 2, the next rank listed would be ranked 5.

DENSE\_RANK again gives you the ranking within your ordered partition, but the ranks are consecutive. No ranks are skipped if there are ranks with multiple items.

EMP DEPTNO SAL R DR1 DR2

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

xxx 10 1 1 4

rrr 10 10000 2 2 1

fff 10 40000 3 3 2

ddd 10 40000 3 3 2

ccc 10 50000 5 4 3

bbb 10 50000 5 4 3

mmm 11 5000 1 1 1

nnn 11 20000 2 2 2

kkk 12 30000 1 1 1

+---+------------+------+------------+

| 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 |

+---+------------+------+------------+

In words

* ROW\_NUMBER() attributes a unique value to each row
* RANK() attributes the same row number to the same value, leaving "holes"
* DENSE\_RANK() attributes the same row number to the same value, leaving no "holes"

SELECT RANK(15500, .05) WITHIN GROUP

(ORDER BY salary, commission\_pct) "Rank"

FROM employees;

Rank

----------

105

SELECT RANK(15500) WITHIN GROUP

(ORDER BY salary DESC) "Rank of 15500"

FROM employees;

Rank of 15500

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

4

SELECT department\_id, last\_name, salary, commission\_pct,

RANK() OVER (PARTITION BY department\_id

ORDER BY salary DESC, commission\_pct) "Rank"

FROM employees WHERE department\_id = 80;

DEPARTMENT\_ID LAST\_NAME SALARY COMMISSION\_PCT Rank

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

80 Russell 14000 .4 1

80 Partners 13500 .3 2

80 Errazuriz 12000 .3 3

80 Ozer 11500 .25 4

80 Cambrault 11000 .3 5

80 Abel 11000 .3 5

80 Zlotkey 10500 .2 7

80 Vishney 10500 .25 8

80 Bloom 10000 .2 9

80 Tucker 10000 .3 10

## <http://docs.oracle.com/cd/B19306_01/server.102/b14200/functions043.htm>

**COALESCE** returns the first non-null ***expr*** in the expression list. You must specify at least two expressions. If all occurrences of ***expr*** evaluate to null, then the function returns null.

The COALESCE function takes two or more compatible arguments and returns the first argument that is not null.

**Example**

**ij> -- create table with three different integer types**

**ij> create table temp(smallintcol smallint, bigintcol bigint, intcol integer);**

**0 rows inserted/updated/deleted**

**ij> insert into temp values (1, null, null);**

**1 row inserted/updated/deleted**

**ij> insert into temp values (null, 2, null);**

**1 row inserted/updated/deleted**

**ij> insert into temp values (null, null, 3);**

**1 row inserted/updated/deleted**

**ij> select \* from temp;**

**SMALL&|BIGINTCOL |INTCOL**

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

**1 |NULL |NULL**

**NULL |2 |NULL**

**NULL |NULL |3**

**3 rows selected**

**ij> -- the return data type of coalesce is bigint**

**ij> select coalesce (smallintcol, bigintcol) from temp;**

**1**

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

**1**

**2**

**NULL**

**3 rows selected**

**ij> -- the return data type of coalesce is bigint**

**ij> select coalesce (smallintcol, bigintcol, intcol) from temp;**

**1**

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

**1**

**2**

**3**

**3 rows selected**

You could use the coalesce function in a SQL statement as follows:

SELECT COALESCE( address1, address2, address3 ) result

FROM suppliers;

The above COALESCE function is equivalent to the following IF-THEN-ELSE statement:

IF address1 is not null THEN

result := address1;

ELSIF address2 is not null THEN

result := address2;

ELSIF address3 is not null THEN

result := address3;

ELSE

result := null;

END IF;

NVL always evaluates both arguments, while COALESCE stops evaluation whenever it finds first non-NULL:

I think coalesce doesn’t work in oracle

MONTHS\_BETWEEN(SYSDATE,DATE'2011-04-18')

So, number of years between two dates can also be achieved with the same function and dividing by 12...

SQL> select months\_between(sysdate, date '2011-04-18')/12 from dual;

MONTHS\_BETWEEN(SYSDATE,DATE'2011-04-18')/12

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

                                 1.53087867

SELECT TO\_DATE('2000-01-02', 'YYYY-MM-DD') -

TO\_DATE('2000-01-01', 'YYYY-MM-DD') AS DateDiff

FROM dual;

select date '2000-01-02' - date '2000-01-01' as dateDiff

from dual

SELECT SYSDATE - TO\_DATE('20081205','YYYYMMDD') datediff FROM DUAL;

SELECT sysdate - INTERVAL '8' MONTH FROM DUAL;

Datediff is not oracle specific instead you can use the above all

Rename a column name in SQL

alter table ravi2 rename column no to id;

rollback;

create table scd as select \* from scd1;

Select NVL(max(s\_key),0) as maxskey,1 as dummy from table;

* **SELECT** 1+1
* **FROM** DUAL;
* **SELECT** 1
* **FROM** DUAL;
* **SELECT** **USER**
* **FROM** DUAL;
* **SELECT** SYSDATE
* **FROM** DUAL;
* **SELECT** \*
* **FROM** DUAL

***The DUAL table is a pseudo table, not a real table. The DUAL table has only one column named DUMMY***

SELECT**COUNT(\*)**FROM**DUAL;**

1. **DELETE** **FROM** DUAL;
2. **TRUNCATE** **TABLE** DUAL

Dual is a table that is created by Oracle together with data dictionary. It consists of exactly one column named “dummy”, and one record. The value of that record is X.

Dual is a virtual table ...it do not eists .

It is used to query fro pseudo columns.

. Select NVL(max(s\_key),0) as maxskey,1 as dummy from table

**To get the maximum salary from each city**

Select city,max(salary) from ravi1 group by city;

**Having**

Select city,max(salary) from ravi1 group by city,salary having salary>500;

### Concatenate More Than 2 Values

SELECT CONCAT(CONCAT('A', 'B'),'C')

FROM dual;

**select concat('a','b') from dual;**

SELECT CONCAT('Let''s', ' learn Oracle')

FROM dual;

**1)Select into in oracle**

create table ravi\_bkp as select \* from ravi

**2)Concat**

select concat(id,name) from ravi

**3)||**

select (id||'-'||name||salary) v from ravi

**Note** :1)Now in Oracle

select sysdate from dual

2) SELECT TO\_CHAR(TO\_DATE(sysdate,'dd-mon-yy'), 'mmddyyyy')

FROM dual

3) SELECT sysdate,

TO\_CHAR(sysdate,'YYYY-MM-DD HH:MI:SS')

FROM dual;

4) **COALESCE** returns the first non-null *expr* in the expression list.

COALESCE (expr1, expr2, ..., expr*n*)

5) SELECT SUBSTR('ABCDEFG',3,3) v

FROM DUAL;

Dateadd

Datediff

**13. What are all the different normalizations?**

The normal forms can be divided into 5 forms, and they are explained below -.

**First Normal Form (1NF):.**

This should remove all the duplicate columns from the table. Creation of tables for the related data and identification of unique columns.

**Second Normal Form (2NF):.**

Meeting all requirements of the first normal form. Placing the subsets of data in separate tables and Creation of relationships between the tables using primary keys.

**Third Normal Form (3NF):.**

This should meet all requirements of 2NF. Removing the columns which are not dependent on primary key constraints.

**Fourth Normal Form (3NF):.**

Meeting all the requirements of third normal form and it should not have multi- valued dependencies.

**Intersect**

|  |
| --- |
| What are the benefits of Normalization? |
| Some of the benefits of normalization include:   * Faster sorting and index creation. * Narrower and more compact indexes. * Fewer indexes per table, which improves the performance of INSERT, UPDATE, and DELETE statements. * Fewer null values and less opportunity for inconsistency, which increase database compactness. |

CREATE TABLE EMPL\_DEMO AS SELECT \* FROM employees WHERE 1=2;

This type of command is usually used to copy the structure of one table to another. In this case, EMPL\_DEMO will have the same column structure of employees, ***except for the keys or constraints***.

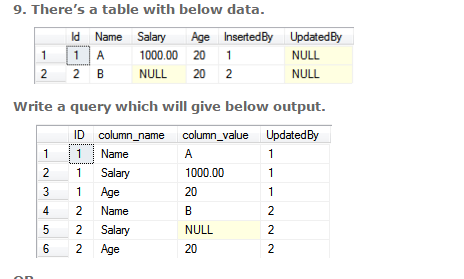
The 1=2 always evaluates to False which prevents you from copying any of the rows.

Convert in sql

Left

Dateadd,datediff

Cast



**How to Include NULL using UNPIVOT.**  
**Below is query using UNPIVOT:**  
declare @employee as table(  
       Id int,  
       Name varchar(40),  
       Salary decimal(18,2),  
       Age int,  
       InsertedBy bigint,  
       UpdatedBy bigint  
)  
  
insert into @employee values(1,'A',1000,20,1,null)  
insert into @employee values(2,'B',null,20,2,null)  
  
select \* from @employee

Select

       Id,

       ColumnName,

       ColumnValue,

       UpdatedBy

From

(

Select

       Id,

       Cast(Name as nvarchar(max)) Name,

       Cast(Salary as nvarchar(max)) Salary,

       Cast(Age as nvarchar(max)) Age,

       isnull(UpdatedBy,InsertedBy) UpdatedBy

From

       @employee

)T

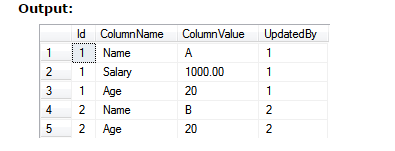
Unpivot

(

       ColumnValue for ColumnName in (Name,Salary, Age)

) as UNP

**Output:**



Here, we can see that, it's excluding results for NULL. We will see how can we achieve it using CROSS JOIN to include NULL column. Below is query for same.

select

       a.ID,

       b.column\_name,

       column\_value =

    case b.column\_name

      when 'Name' then a.Name

      when 'Salary' then a.Salary

      when 'Age' then a.Age

    end,

       UpdatedBy

from (

       Select

              Id,

              Cast(Name as nvarchar(max)) Name,

              Cast(Salary as nvarchar(max)) Salary,

              Cast(Age as nvarchar(max)) Age,

              isnull(UpdatedBy,InsertedBy) UpdatedBy

       From

              @employee

  ) a

cross join (

  select 'Name' union all

  select 'Salary' union all

  select 'Age'

  ) b (column\_name)

**2.** Write a query to display only friday dates from Jan, 2000 to till now?  
  
**Solution:**

SELECT C\_DATE,

TO\_CHAR(C\_DATE,'DY')

FROM

(

SELECT TO\_DATE('01-JAN-2000','DD-MON-YYYY')+LEVEL-1 C\_DATE

FROM DUAL

CONNECT BY LEVEL <=

(SYSDATE - TO\_DATE('01-JAN-2000','DD-MON-YYYY')+1)

)

WHERE TO\_CHAR(C\_DATE,'DY') = 'FRI';

<http://www.folkstalk.com/2011/10/oracle-scenario-based-questions-with.html>

<https://docs.oracle.com/cd/B19306_01/server.102/b14200/queries003.htm>

AVG() - Returns the average value

COUNT() - Returns the number of rows

FIRST() - Returns the first value

LAST() - Returns the last value

MAX() - Returns the largest value

MIN() - Returns the smallest value

SUM() - Returns the sum

CONVERT(char(7),DATEADD(m,-1,GETDATE()),120)

SELECT SUBSTRING(dn,1,convert(int,(CHARINDEX('/',dn) -1))) AS LEFTHALF,

SUBSTRING(dn,CHARINDEX('/',dn),100) AS RIGHTHALF from CHASSIS\_D

select SUBSTRING(dn,1,CHARINDEX('/',dn,3)) AS LEFTHALF from CHASSIS\_D

select CHARINDEX('/',dn,5),dn from CHASSIS\_D

SELECT count(\*)

FROM INFORMATION\_SCHEMA.COLUMNS

WHERE TABLE\_NAME = 'chat\_output1'

execute sp\_helpindex test

**Learn case statement**

**Datepart**

**Dateadd**

**Cast**

**Convert**

(DATEADD(month, 0, Convert(DATETIME, max(LEFT(date\_key, 8)))),DATEADD(month, -1, Convert(DATETIME, max(LEFT(date\_key, 8)))))

ANALYZE -- collect statistics about a database

extract(month from a.as\_of\_dt)

**Grant select on all tables in schema ads\_rpt to adsdb\_ownr;**

**Grant select on all tables in schema ads\_rpt to ads\_ownr\_role;**

**Grant select on all tables in schema ads\_rpt to ads\_dml\_role;**

**Grant select on all tables in schema ads\_rpt to ads\_select\_role;**

**grant all on all tables in schema ads\_rpt to rds\_superuser WITH GRANT OPTION;**

**grant all on all tables in schema ads\_rpt to adsdb\_ownr;**

**GRANT SELECT, UPDATE, INSERT, TRUNCATE, DELETE on all tables in schema ads\_rpt to ads\_dml\_role;**

**Grant select on all tables in schema ads\_rpt to ads\_select\_role;**

**Grant select on all tables in schema ads\_rpt to dev\_role;**

**Grant all on all tables in schema ads\_rpt to ads\_ownr\_role;**

**GRANT ALL ON SCHEMA ads\_rpt TO adsdb\_ownr;**

**GRANT ALL ON SCHEMA ads\_rpt TO ads\_ownr\_role;**

**GRANT USAGE ON SCHEMA ads\_rpt TO ads\_dml\_role;**

**GRANT USAGE ON SCHEMA ads\_rpt TO ads\_select\_role;**

**GRANT USAGE ON SCHEMA ads\_rpt TO dev\_role;**

**GRANT ALL ON SCHEMA ads\_rpt TO rds\_superuser WITH GRANT OPTION;**

**ALTER DEFAULT PRIVILEGES IN SCHEMA ads\_rpt**

**GRANT INSERT, SELECT, UPDATE, DELETE, TRUNCATE, REFERENCES, TRIGGER ON TABLES**

**TO rds\_superuser WITH GRANT OPTION;**

**ALTER DEFAULT PRIVILEGES IN SCHEMA ads\_rpt**

**GRANT INSERT, SELECT, UPDATE, DELETE, TRUNCATE ON TABLES**

**TO ads\_dml\_role;**

**ALTER DEFAULT PRIVILEGES IN SCHEMA ads\_rpt**

**GRANT SELECT ON TABLES**

**TO ads\_select\_role;**

**ALTER DEFAULT PRIVILEGES IN SCHEMA ads\_rpt**

**GRANT SELECT ON TABLES**

**TO dev\_role;**

**ALTER DEFAULT PRIVILEGES IN SCHEMA ads\_rpt**

**GRANT SELECT, UPDATE, USAGE ON SEQUENCES**

**TO rds\_superuser WITH GRANT OPTION;**

**ALTER DEFAULT PRIVILEGES IN SCHEMA ads\_rpt**

**GRANT SELECT, USAGE ON SEQUENCES**

**TO ads\_dml\_role;**

**ALTER DEFAULT PRIVILEGES IN SCHEMA ads\_rpt**

**GRANT SELECT, USAGE ON SEQUENCES**

**TO ads\_select\_role;**

**ALTER DEFAULT PRIVILEGES IN SCHEMA ads\_rpt**

**GRANT SELECT, USAGE ON SEQUENCES**

**TO dev\_role;**

**ALTER DEFAULT PRIVILEGES IN SCHEMA ads\_rpt**

**GRANT EXECUTE ON FUNCTIONS**

**TO rds\_superuser WITH GRANT OPTION;**

**ALTER DEFAULT PRIVILEGES IN SCHEMA ads\_rpt**

**GRANT EXECUTE ON FUNCTIONS**

**TO ads\_dml\_role;**

Types of constraints: NOT NULL, CHECK, UNIQUE, PRIMARY KEY, FOREIGN KEY. default

**INSERT INTO ads\_rpt.rlatnp\_dim(**

**name, enty\_stat\_id, wcis\_id, fundr\_enty\_stat\_dt, src\_sys,**

**src\_sys\_id, etl\_proc\_id, etl\_insrt\_dt, etl\_updt\_dt)**

**select "Name",null, cast ("IntegrationCode" as bigint) ,null,'HUB',"RelationshipID",-1,current\_timestamp,current\_timestamp**

**from "Relationship" where "IntegrationCode" ~ '^[0-9]\*.?[0-9]\*$'-----15630**

DO $$

declare

r record;

BEGIN

for r in select tablename from pg\_tables where schemaname='pt\_intg'

loop

execute 'DROP table "'||r.tablename||'" cascade';

end loop;

end$$;

**with cte as (select afl.\* from pt\_intg.acbs\_facility\_loan afl**

**join pt\_intg.acbs\_facility af on afl.acbs\_facility\_num = af.acbs\_facility\_num and fidr\_id in (513717,**

**291163,**

**291182))**

**delete from pt\_intg.acbs\_facility\_loan afl**

**using cte where**

**afl.acbs\_facility\_num = cte.acbs\_facility\_num and afl.obligation\_num = cte.obligation\_num and afl.limit\_type\_cd = cte.limit\_type\_cd and afl.limit\_key\_val = cte.limit\_key\_val**

**and afl.as\_of\_dt = cte.as\_of\_dt;**

**show databases**

A cursor is a set of rows together with a pointer that identifies a current row.

Trigger

Savepoint

**update facility**

**set facility\_extended\_fields = json\_object('{facility\_type\_description, "Equipment Lease"**

**, facility\_status\_description, "Active"**

**, facility\_status\_date, "2016-09-12"**

**,facility\_workflow\_status\_id, "563"**

**,facility\_workflow\_status\_description, "Approved"**

**, portfolio\_manager\_id, "0"**

**, portfolio\_manager\_name, ""**

**, underwriter\_id, "0"**

**, underwriter\_name, ""**

**,relationship\_manager\_id, "718"**

**, relationship\_manager\_name, "Welch**

**, Peter"**

**,credit\_analyst, "0"**

**,facility\_credit\_action\_type\_description, "New Facility"**

**,leveraged\_facility\_indicator, ""**

**,asset\_based\_lending\_lnd, "No"**

**,primary\_source\_repayment, "asfdasdf"**

**,secondary\_source\_repayment, ""**

**,tertiary\_source\_repayment, ""**

**,scorecrad, "Collateral"**

**,type\_of\_rating, "CRRP Rating Engine"**

**,pd\_rating\_id, "210174"**

**,pd\_rating\_code, "27"**

**,proposed\_rating\_value, "S"**

**,proposed\_rating\_date, "2016-09-12 14:49:26.003"**

**,crrp\_rating\_value, "S (90%)"**

**,rating\_source\_id, "1"**

**,is\_ovverride\_applied, "0"**

**,override\_note, ""**

**,is\_exception\_applied, "0"**

**,exception\_note, ""**

**,col\_id\_used\_in\_rating, "60044;77640**

**,60045;77641**

**,60046;77642"**

**,unused\_col\_id\_in\_rating, ""**

**,ext\_facyl\_note, ""**

**,userid\_facilityrated, "0"**

**,pd\_wcis\_id, "900135023507"**

**,IsThisFacilityPartOfASyndication, "Yes"**

**,ConaRoleSyndication, ""**

**,facility\_approval\_dt, "2016-09-12 14:49:32.08"}')**

**where fidr\_id = 506163**

**SELECT /\*+ PARALLEL(SIEBEL.S\_UCM\_CONTACT,5) \*/ \* FROM SIEBEL.S\_UCM\_CONTACT**

**WHERE EXTRACT(YEAR FROM LAST\_UPD) >= 2017**

1. Prefer to use views and stored procedures in spite of writing long queries. It’ll also help in minimizing network load.

2. It’s better to introduce constraints instead of triggers. They are more efficient than triggers and can increase performance.

3. Make use of table-level variables instead of temporary tables.

4. The UNION ALL clause responds faster than UNION. It doesn’t look for duplicate rows whereas the UNION statement does that regardless of whether they exist or not.

5. Prevent the usage of DISTINCT and HAVING clauses.

6. Avoid excessive use of SQL cursors.

7. Make use of SET NOCOUNT ON clause while building stored procedures. It represents the rows affected by a T-SQL statement. It would lead to reduced network traffic.

8. It’s a good practice to return the required column instead of all the columns of a table.

9. Prefer not to use complex joins and avoid disproportionate use of triggers.

10. Create indexes for tables and adhere to the standards.

**2)**HAVING clause is used to filter the rows after all the rows are selected. It is just like a filter. Do not use HAVING clause for any other purposes.  
**For Example:**Write the query as

SELECT subject, count(subject)  
FROM student\_details  
WHERE subject != 'Science'  
AND subject != 'Maths'  
GROUP BY subject;

Instead of:

SELECT subject, count(subject)  
FROM student\_details  
GROUP BY subject  
HAVING subject!= 'Vancouver' AND subject!= 'Toronto';

Sometimes you may have more than one subqueries in your main query. Try to minimize the number of subquery block in your query.  
**For Example:**Write the query as

SELECT name  
FROM employee  
WHERE (salary, age ) = (SELECT MAX (salary), MAX (age)  
FROM employee\_details)  
AND dept = 'Electronics';

Instead of:

SELECT name  
FROM employee  
WHERE salary = (SELECT MAX(salary) FROM employee\_details)  
AND age = (SELECT MAX(age) FROM employee\_details)  
AND emp\_dept = 'Electronics';

**4)**Use operator EXISTS, IN and table joins appropriately in your query.  
**a)**Usually IN has the slowest performance.  
**b)**IN is efficient when most of the filter criteria is in the sub-query.  
**c)**EXISTS is efficient when most of the filter criteria is in the main query.

**For Example:**Write the query as

Select \* from product p  
where EXISTS (select \* from order\_items o  
where o.product\_id = p.product\_id)

Instead of:

Select \* from product p  
where product\_id IN  
(select product\_id from order\_items

**5)**Use EXISTS instead of DISTINCT when using joins which involves tables having one-to-many relationship.  
**For Example:**Write the query as

SELECT d.dept\_id, d.dept  
FROM dept d  
WHERE EXISTS ( SELECT 'X' FROM employee e WHERE e.dept = d.dept);

Instead of:

SELECT DISTINCT d.dept\_id, d.dept  
FROM dept d,employee e  
WHERE e.dept = e.dept;

**6)**Try to use UNION ALL in place of UNION.  
**For Example:**Write the query as

SELECT id, first\_name  
FROM student\_details\_class10  
UNION ALL  
SELECT id, first\_name  
FROM sports\_team;

Instead of:

SELECT id, first\_name, subject  
FROM student\_details\_class10  
UNION  
SELECT id, first\_name  
FROM sports\_team;

**7)**Be careful while using conditions in WHERE clause.  
**For Example:**Write the query as

SELECT id, first\_name, age FROM student\_details WHERE age > 10;

Instead of:

SELECT id, first\_name, age FROM student\_details WHERE age != 10;

Write the query as

SELECT id, first\_name, age  
FROM student\_details  
WHERE first\_name LIKE 'Chan%';

Instead of:

SELECT id, first\_name, age  
FROM student\_details  
WHERE SUBSTR(first\_name,1,3) = 'Cha';

Write the query as

SELECT id, first\_name, age  
FROM student\_details  
WHERE first\_name LIKE NVL ( :name, '%');

Instead of:

SELECT id, first\_name, age  
FROM student\_details  
WHERE first\_name = NVL ( :name, first\_name);

Write the query as

SELECT product\_id, product\_name  
FROM product  
WHERE unit\_price BETWEEN MAX(unit\_price) and MIN(unit\_price)

Instead of:

SELECT product\_id, product\_name  
FROM product  
WHERE unit\_price >= MAX(unit\_price)  
and unit\_price <= MIN(unit\_price)

Write the query as

SELECT id, name, salary  
FROM employee  
WHERE dept = 'Electronics'  
AND location = 'Bangalore';

Instead of:

SELECT id, name, salary  
FROM employee  
WHERE dept || location= 'ElectronicsBangalore';

Use non-column expression on one side of the query because it will be processed earlier.

Write the query as

SELECT id, name, salary  
FROM employee  
WHERE salary < 25000;

Instead of:

SELECT id, name, salary  
FROM employee  
WHERE salary + 10000 < 35000;

Write the query as

SELECT id, first\_name, age  
FROM student\_details  
WHERE age > 10;

Instead of:

SELECT id, first\_name, age  
FROM student\_details  
WHERE age NOT = 10;

The **DESCRIBE** and **EXPLAIN** statements are synonyms. In practice, the **DESCRIBE** keyword is more often used to obtain information about table structure, whereas **EXPLAIN** is used to obtain a query execution plan (that is, an **explanation** of how MySQL would execute a query).

The explain plan is what the optimizer thinks will happen when you run, the execution plan is actually happened when you ran the query.

Non clustered index without clustered index will do a table scan and second case did clustered index scan.

If a clustered index is like a phone book, a nonclustered index is like the index in the back of a chemistry text book.

Nth Highest Salary

SELECT Salary FROM (

select b.salary, count(\*) as rank

from employee a, employee b

where b.salary <= a.salary

group by b.salary

) salary\_ranks

WHERE rank = N

Date Handling

**MySQL**

SELECT DATE\_SUB( '2016-12-25', INTERVAL 7 DAY )

SELECT DATE\_SUB( '2016-12-25', INTERVAL 7 DAY )

select \* from orders where dateadd(week,-1,order\_date)

**Oracle**

**SELECT** **SYSDATE** - **interval** '6' **day** **FROM** dual

ADD\_MONTHS(SYSDATE, -6)

**SQL Server**

DATEADD(MONTH,-6, GETDATE())

**Note**

if you want to do the operations from start of the current month always, TRUNC(SYSDATE,'MONTH') would give that. And it expects a Date datatype as input.

**Note**

Index scan, seek,fullscan

sys.dm\_db\_index\_physical\_stats

**Note**

The Skewness of the table means that table data is not equally distributed across the segments and workload is not divided properly between the segments.

**Data skew** means the **data** is not distributed evenly across the distributions.

A quick way to check for data skew is to use [DBCC PDW\_SHOWSPACEUSED](https://docs.microsoft.com/en-us/sql/t-sql/database-console-commands/dbcc-pdw-showspaceused-transact-sql). The following SQL code returns the number of table rows that are stored in each of the 60 distributions. For balanced performance, the rows in your distributed table should be spread evenly across all the distributions.

SQLCopy

-- Find data skew for a distributed table

DBCC PDW\_SHOWSPACEUSED('dbo.FactInternetSales');

### Re-create the table with a new distribution column

This example uses [CREATE TABLE AS SELECT](https://docs.microsoft.com/sql/t-sql/statements/create-table-as-select-azure-sql-data-warehouse?view=aps-pdw-2016-au7) to re-create a table with a different hash distribution column.

Partition, day, month or year