Contents

[Analytic Functions windowing function 2](#_Toc79600130)

[Aggregate and analytic 2](#_Toc79600131)

[Partition By 2](#_Toc79600132)

[Q. WRITE a query TO RETURN the count AND average weight OF bricks FOR EACH shape 2](#_Toc79600133)

[Order By 3](#_Toc79600134)

[Partition by + order by 3](#_Toc79600135)

[Windowing clause 3](#_Toc79600136)

[Sliding Windows 4](#_Toc79600137)

[Filtering Analytic Function 5](#_Toc79600138)

[Rank, Dense\_rank and Row\_number 5](#_Toc79600139)

[Lead and Lag Function 6](#_Toc79600140)

[First and Last value 6](#_Toc79600141)

[Creating, Altering, and Updating Objects Using SQL 7](#_Toc79600142)

[Creating a Table 7](#_Toc79600143)

[Inserting data into table 7](#_Toc79600144)

[Inserting large amount of data at once 8](#_Toc79600145)

[Defining Primary Key constraint 8](#_Toc79600146)

[Insert all for Multi-tables and conditional insert 8](#_Toc79600147)

[Conditional Insert 9](#_Toc79600148)

[ALTER 9](#_Toc79600149)

[Creating new table and pulling data from existing table. 9](#_Toc79600150)

[Create table with select + update data 9](#_Toc79600151)

[MERGE Statement 10](#_Toc79600152)

[SEQUENCE Statement 12](#_Toc79600153)

[creating index 13](#_Toc79600154)

[---creating index in multiple columns 13](#_Toc79600155)

[--- creating Unique Index 13](#_Toc79600156)

[--- Dropping Index 13](#_Toc79600157)

[----creating indexes with information. this gives certain information to database 13](#_Toc79600158)

[---REbuilding indexes with conpute statistics 13](#_Toc79600159)

[How to display all indexes details in database 14](#_Toc79600160)

[---how to get rid of duplicate data 14](#_Toc79600161)

[---rowid Pseudo Column --gives unique rowid to all columns 14](#_Toc79600162)

[---MIN(rowid) --- it gives one rowid to the group of same rows in the table 14](#_Toc79600163)

[--Now let's delete the repeated data 15](#_Toc79600164)

# Analytic Functions windowing function

**create** **table** bricks (

brick\_id **integer**,

colour **varchar2**(10),

shape **varchar2**(10),

weight **integer**

);

**INSERT** **ALL** **into** bricks **values** ( 1, 'blue', 'cube', 1 )

**into** bricks **values** ( 2, 'blue', 'pyramid', 2 )

**into** bricks **values** ( 3, 'red', 'cube', 1 )

**into** bricks **values** ( 4, 'red', 'cube', 2 )

**into** bricks **values** ( 5, 'red', 'pyramid', 3 )

**into** bricks **values** ( 6, 'green', 'pyramid', 1 )

**SELECT** \* **FROM** dual;

Aggregate and analytic function both enables to do calculation over many rows.

**SELECT** **count**(\*) **FROM** bricks; -- this will squash OUTPUT TO one ROW per group

**SELECT** **count**(\*) **over**() **FROM** bricks; --OVER clause IS analytic AND preserves OUTPUT ROWS

**SELECT** b.\*, --this allows us TO see VALUES FROM ALL other columns.

**count**(\*) **OVER**() total\_count

**FROM** bricks b;

## Partition By

--The group by clause splits rows into groups of the same value

--following code get number of rows and total weight for each colour

**SELECT** colour, **count**(\*), **sum** (weight)

**FROM** BRICKS

**GROUP** **BY** colour;

--we can carve up the input to an analytic function like this with partition by clause

--the FOLLOWING code returns the total weight and count of rows of each colour

**SELECT** b.\*,

**count**(\*) **OVER** (**PARTITION** **BY** colour) bricks\_per\_colour,

**sum** (weight) **OVER** (**PARTITION** **BY** colour) weight\_per\_colour

**FROM** bricks b;

### Q. WRITE a query TO RETURN the count AND average weight OF bricks FOR EACH shape

**SELECT** b.\*,

**count**(\*) **OVER** (**PARTITION** **BY** shape) bricks\_per\_shape,

**MEDIAN**(weight) **OVER** (**PARTITION** **BY** shape) median\_wiight\_per\_shape

**FROM** bricks b

**ORDER** **BY** shape, weight, brick\_id;

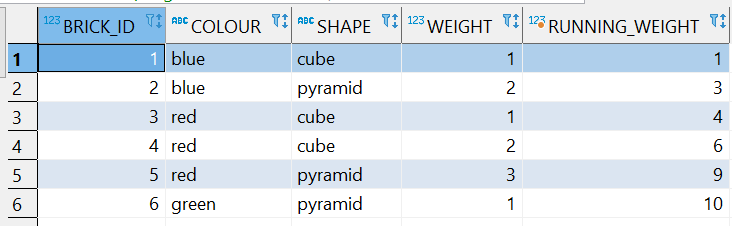
**SELECT** \* **FROM** bricks

## Order By

**SELECT** b.\*,

**sum** (weight) **OVER** (**ORDER** **BY** brick\_id) running\_weight

**FROM** BRICKS b ;



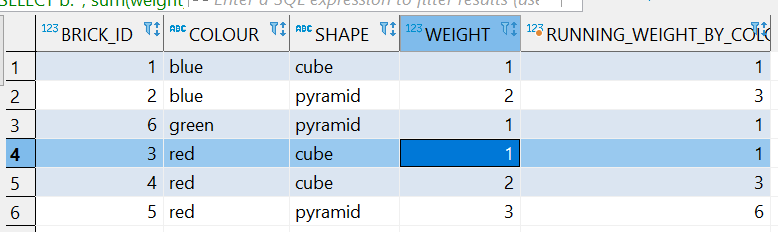
## Partition by + order by

**SELECT** b.\*,

**sum**(weight) **OVER** (**PARTITION** **BY** colour **ORDER** **BY** brick\_id) running\_weight\_by\_colour

**FROM** BRICKS b

**ORDER** **BY** colour;



## Windowing clause

When using order by clause, database by default adds windowing clause of:

**RANGE** **BETWEEN** **UNBOUNDED** **PRECEDING** **AND** **CURRENT** **ROW**

**SELECT** b.\*,

**sum** (weight) over (**ORDER** **BY** brick\_id)

**FROM** bricks B

**ORDER** **BY** brick\_id;

-- which is same as the following code. in the code above the range between... is by default there in the code by DB

**SELECT** b.\*,

**sum**(weight) **OVER** (**ORDER** **BY** brick\_id **RANGE** **BETWEEN** **UNBOUNDED** **PRECEDING** **AND** **CURRENT** **ROW**) running\_weight

**FROM** BRICKS b

**ORDER** **BY** BRICK\_ID;

-- now to further understand windowing clause let's take an example

**SELECT** b.\*,

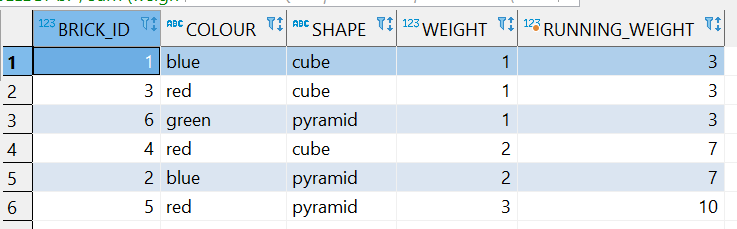
**sum** (weight) **OVER** (**ORDER** **BY** weight) running\_weight

**FROM** BRICKS b

**ORDER** **BY** WEIGHT ;

--this will give the following output. which gives three repeated 3 in columns which we are not expecting

--this is occuring due to default range between... with order by clause.



-- now to solve this problem we can use ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW

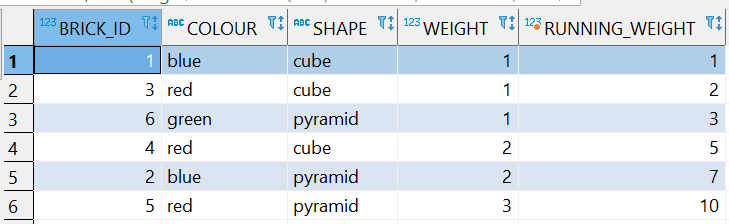
**SELECT** b.\*

, **sum**(weight) **OVER** (**ORDER** **BY** weight **ROWS** **BETWEEN** **UNBOUNDED** **PRECEDING** **AND** **CURRENT** **ROW**) running\_weight

**FROM** bricks B

**ORDER** **BY** weight;

--which gives the exact required output



## Sliding Windows

--slinding windows is replacement of unbounded by bounded value such as 1/2/3 (number of rows to work with)

--which means current row and number of preceding rows.

--in the example below it’s using current row and 1 PRECEDING row.

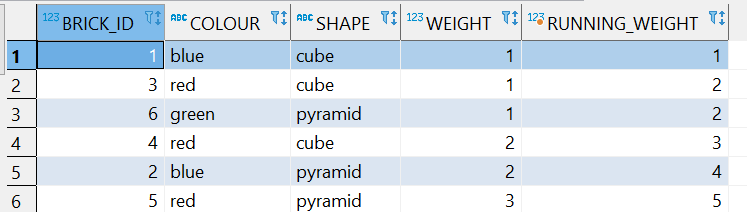
**SELECT** b.\*,

**sum** (weight) **OVER** (**ORDER** **BY** weight **ROWS** **BETWEEN** 1 **PRECEDING** **AND** **CURRENT** **ROW**) running\_weight

**FROM** BRICKS b

**ORDER** **BY** weight;

This gives the output of:



## Filtering Analytic Function

To filter rows we use the result of an aggregate. For eg

**SELECT** \*

**FROM** (

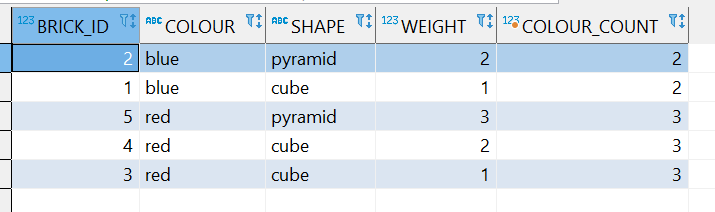
**SELECT** b.\*

, **count**(\*) **OVER** (**PARTITION** **BY** colour) colour\_count

**FROM** bricks B

)

**WHERE** colour\_count >=2;



## Rank, Dense\_rank and Row\_number

Row number gives the number of rows in the given table in a sequential order.

Rank gives rank to the given row looking at all the rows in the given column.

Dense rank gives the rank but doesn’t have gap.

Let’s try to understand more using the following example:

**select** brick\_id, weight,

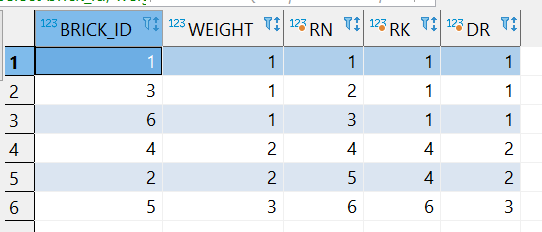
**row\_number**() **over** ( **order** **by** weight ) rn,

**rank**() **over** ( **order** **by** weight ) rk,

**dense\_rank**() **over** ( **order** **by** weight ) dr

**from** bricks;

the output is as follows:



There is difference between rank and dense rank. The difference is that rank gives rank based on total rows and places gap in the rank, which we can see in the example above RK column shows 3; 1st rank and 2 fourth ranks because there are already 3 first ranks and following second rank comes after 3 rows. Where dense rank gives rank without gap.

## Lead and Lag Function

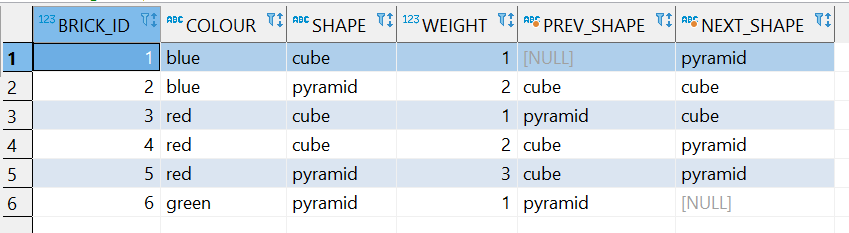
**select** b.\*,

**lag** ( shape ) **over** ( **order** **by** brick\_id ) prev\_shape,

**lead** ( shape ) **over** ( **order** **by** brick\_id ) next\_shape

**from** bricks b;

output



## First and Last value

**select** b.\*,

**first\_value** ( weight ) **over** (

**order** **by** brick\_id

) first\_weight\_by\_id,

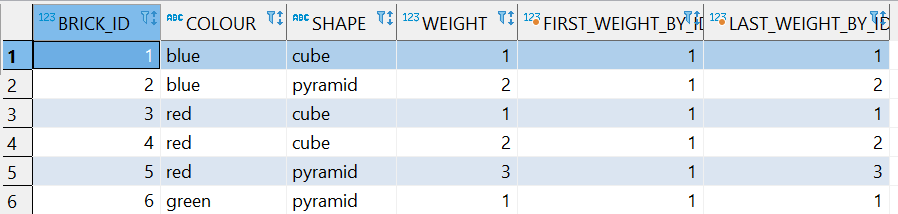
**last\_value** ( weight ) **over** (

**order** **by** brick\_id

) last\_weight\_by\_id

**from** bricks b;

gives:



In this output first value function is giving the desired output but the last\_value function is not. To solve this issue we can change the function as follows:

**select** b.\*,

**first\_value** ( weight ) **over** (

**order** **by** brick\_id

) first\_weight\_by\_id,

**last\_value** ( weight ) **over** (

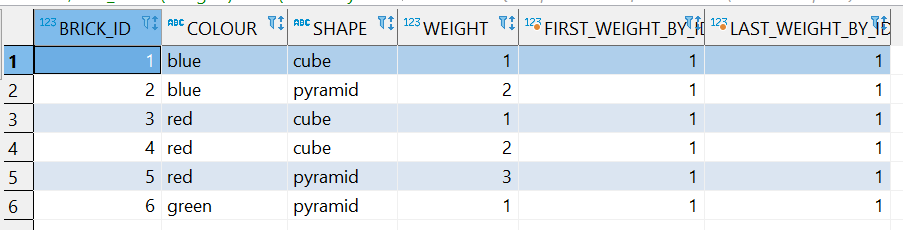
**order** **by** brick\_id

**range** **between** **current** **row** **and** **unbounded** **following**

) last\_weight\_by\_id

**from** bricks b;

which will give output of:



# Creating, Altering, and Updating Objects Using SQL

Primary Key column – cannot have duplicates, cannot be null

## Creating a Table

CREATE TABLE dest\_tbl\_3 (

    id number not null,

    name varchar2 (50),

    date\_of date

)

## Inserting data into table

INSERT INTO store (store\_id, city)

VALUES (1, 'New York City');

## Inserting large amount of data at once

INSERT \*

INTO store (store\_id, city) VALUES (1, 'New York City')

INTO store (store\_id, city) VALUES (2, 'New York City')

INTO store (store\_id, city) VALUES (3, 'New York City')

INTO store (store\_id, city) VALUES (4, 'New York City')

INTO store (store\_id, city) VALUES (5, 'New York City')

SELECT \* FROM dual;

## Defining Primary Key constraint

CREATE TABLE table\_name

(

column1 datatype null/not null,

column2 datatype null/not null,

...

CONSTRAINT constraint\_name PRIMARY KEY (column1, column2, ... column\_n)

);

CREATE TABLE supplier

(

  supplier\_id numeric(10) not null,

  supplier\_name varchar2(50) not null,

  contact\_name varchar2(50),

  CONSTRAINT supplier\_pk PRIMARY KEY (supplier\_id)

);

## Insert all for Multi-tables and conditional insert

INSERT ALL

    INTO dest\_tbl\_1 (id, name, date\_of) values (empno, ename, hiredate)

    INTO dest\_tbl\_2 (id, name, date\_of) values (empno, ename, hiredate)

    into dest\_tbl\_3 (id, name, date\_of) values (empno, ename, hiredate)

SELECT empno, ename, hiredate

FROM EMP

## Conditional Insert

INSERT ALL

WHEN sal <= 1500 THEN

    INTO dest\_tbl\_1 (id, name, date\_of) values (empno, ename, hiredate)

WHEN sal BETWEEN 1501 AND 2500 THEN

    INTO dest\_tbl\_2 (id, name, date\_of) values (empno, ename, hiredate)

WHEN sal > 2500 THEN

    into dest\_tbl\_3 (id, name, date\_of) values (empno, ename, hiredate)

SELECT empno, ename, hiredate,sal

FROM EMP

## ALTER

ALTER TABLE product

MODIFY name varchar2(50) not null;

INSERT INTO product VALUES (1011, '', 4.00, 8, 'clothing', 3) --- this won't work

---altering two or more columns same time

ALTER TABLE product

MODIFY (product\_cost number(5,2) not null,

        product\_retail number(5,2) not null);

-RENAME

        ALTER TABLE product

        RENAME column name TO product\_name;

ALRER TABLE <table\_name>

RENAME COLUMN <old\_name> TO <new\_name>

## Creating new table and pulling data from existing table.

create table employee as

select empno, ename, job, hiredate, sal, comm

from emp;

This creates new table ‘employee’ and pulls data from existing table ‘emp’.

## Create table with select + update data

Pulling data from existing table and creating new table and populating the

new table with the data from the existing table.

create table employee as

select empno, ename, job, hiredate, sal, comm

from emp;

--- this creates new table ‘employee’ and pulls data from ‘emp’.

--- note: while adding column to an existing table with data we cannot specify it to be not null.

For eg:

ALTER TABLE employee

ADD store\_id number not null; --- this won’t work because employee table already have data.

--- In order to add column first we should create a table without specifying not null constraint and populate with data and then only we should specify not null constraint.

ALTER TABLE employee

ADD store\_id number;

---Updating data in the new column

UPDATE employee

SET store\_id = 3

WHERE ename in ('KING', 'BLAKE', 'CLARK')

UPDATE employee

SET store\_id = 1

where ENAME = 'JONES'

## MERGE Statement

We will create tables and insert data into the tables to understand this

CREATE TABLE existing\_customers (

  customer\_id NUMBER(5),

  first\_name VARCHAR2(100),

  last\_name VARCHAR2(100),

  address\_state VARCHAR2(10),

  email\_address VARCHAR2(350),

  CONSTRAINT old\_pk\_cust PRIMARY KEY (customer\_id)

);

BEGIN

    INSERT INTO existing\_customers (customer\_id, first\_name, last\_name, address\_state, email\_address)

    VALUES (1, 'Teresa', 'Hudson', 'NY', 'thudson@abc.com');

    INSERT INTO existing\_customers (customer\_id, first\_name, last\_name, address\_state, email\_address)

    VALUES (2, 'Fred', 'Montgomery', 'CA', 'fmont@gmail.com');

    INSERT INTO existing\_customers (customer\_id, first\_name, last\_name, address\_state, email\_address)

    VALUES (3, 'Lois', 'Lawson', 'OR', 'lois\_law@outlook.com');

    INSERT INTO existing\_customers (customer\_id, first\_name, last\_name, address\_state, email\_address)

    VALUES (4, 'Alice', 'Perry', 'SC', 'aliceperry');

    INSERT INTO existing\_customers (customer\_id, first\_name, last\_name, address\_state, email\_address)

    VALUES (5, 'Ralph', 'Montgomery', 'TX', 'ralph\_mont25@gmail.com');

    INSERT INTO existing\_customers (customer\_id, first\_name, last\_name, address\_state, email\_address)

    VALUES (6, 'Dorothy', 'Armstrong', 'OR', 'abc123@abc.com');

    INSERT INTO existing\_customers (customer\_id, first\_name, last\_name, address\_state, email\_address)

    VALUES (7, 'Fred', 'Wallace', 'NY', 'wallacef@testwebsite.co.uk');

    INSERT INTO existing\_customers (customer\_id, first\_name, last\_name, address\_state, email\_address)

    VALUES (8, 'Joseph', 'Bell', 'FL', 'jbell@comm.edu');

    INSERT INTO existing\_customers (customer\_id, first\_name, last\_name, address\_state, email\_address)

    VALUES (9, 'Lois', 'Martinez', 'CALIF', 'loismar@awe.com');

    INSERT INTO existing\_customers (customer\_id, first\_name, last\_name, address\_state, email\_address)

    VALUES (10, 'Robert', 'Rice', 'IN', 'robrice123');

END;

CREATE TABLE  new\_customers(

  customer\_id NUMBER(5),

  first\_name VARCHAR2(100),

  last\_name VARCHAR2(100),

  address\_state VARCHAR2(10),

  email\_address VARCHAR2(350)

);

BEGIN

    INSERT INTO new\_customers (customer\_id, first\_name, last\_name, address\_state, email\_address)

    VALUES (15, 'John', 'Stock', 'ND', 'js1980@outlook.com');

    INSERT INTO new\_customers (customer\_id, first\_name, last\_name, address\_state, email\_address)

    VALUES (12, 'Joseph', 'Bell', 'OR', 'thebell@yahoo.com');

    INSERT INTO new\_customers (customer\_id, first\_name, last\_name, address\_state, email\_address)

    VALUES (3, 'Lois', 'Lawson', 'WA', 'lois\_law@outlook.com');

    INSERT INTO new\_customers (customer\_id, first\_name, last\_name, address\_state, email\_address)

    VALUES (8, 'Joseph', 'Bell', 'FL', 'jbell@gmail.com');

    INSERT INTO new\_customers (customer\_id, first\_name, last\_name, address\_state, email\_address)

    VALUES (1, 'Teresa', 'Hudson', 'NY', 'thudson@abc.com');

    INSERT INTO new\_customers (customer\_id, first\_name, last\_name, address\_state, email\_address)

    VALUES (4, 'Alice', 'Howard', 'SC', 'aliceperry@gmail.com');

END;

By using merge statement we will try to bring new updated details of costumers which are changed in new table. This merge statement will update old table with new data from new table.

For an example we have one costumer name Alice whose old last name and email address are different in the two table.

MERGE INTO existing\_customers c

USING new\_customers n

ON (c.customer\_id = n.customer\_id)

WHEN MATCHED THEN

    UPDATE SET

    c.first\_name = n.first\_name,

    c.last\_name = n.last\_name,

    c.address\_state = n.address\_state,

    c.email\_address = n.email\_address

WHEN NOT MATCHED THEN

    INSERT (c.customer\_id, c.first\_name, c.last\_name, c.address\_state, c.email\_address)

    VALUES (n.customer\_id, n.first\_name, n.last\_name, n.address\_state, n.email\_address);

## SEQUENCE Statement

CREATE SEQUENCE product\_seq

    MINVALUE 1

    MAXVALUE 999999999999999999

    START WITH 1

    INCREMENT BY 1

    CACHE 20;

--- HOW TO use sequence statement

SELECT product\_seq.NEXTVAL FROM DUAL;

---ALTER sequence

ALTER SEQUENCE product\_seq

    NOCACHE;

---Using of Sequence Statement

INSERT INTO product

(product\_id, product\_name)

VALUES

(product\_seq.NEXTVAL, 'PRODUCTS\_ANY')

---using this we can add values by creating different sequence statements for different columns.

## creating index

CREATE INDEX emp\_name\_idx

    ON EMPLOYEE (ename) ---this creates index on ename column

### ---creating index in multiple columns

CREATE INDEX emp\_job\_hdate\_idx

    ON EMPLOYEE (ename, job, hiredate)

---indexing is used to make it faster to search for certain data which are

--- used often, it will be easier for database to find such data

---eg:

SELECT \* FROM employee

WHERE ename = 'john'

AND hiredate = ''

AND job = ''

### --- creating Unique Index

--- we cannot create unique index in such column which data are not unique

---for eg:

CREATE UNIQUE INDEX emp\_job\_index

    ON employee (job)  --- this won't work

### --- Dropping Index

DROP INDEX emp\_job\_idx

### ----creating indexes with information. this gives certain information to database

CREATE INDEX emp\_name\_job\_date\_idx

    ON employee (ename, job, hiredate)

    COMPUTE STATISTICS;

### ---REbuilding indexes with conpute statistics

ALTER INDEX emp\_name\_idx

REBUILD COMPUTE STATISTICS;

## How to display all indexes details in database

select ind.index\_name,

       ind\_col.column\_name,

       ind.index\_type,

       ind.uniqueness,

       ind.table\_owner as schema\_name,

       ind.table\_name as object\_name,

       ind.table\_type as object\_type

from sys.all\_indexes ind

inner join sys.all\_ind\_columns ind\_col on ind.owner = ind\_col.index\_owner

                                    and ind.index\_name = ind\_col.index\_name

-- excluding some Oracle maintained schemas

where ind.owner not in ('ANONYMOUS','CTXSYS','DBSNMP','EXFSYS', 'LBACSYS',

   'MDSYS', 'MGMT\_VIEW','OLAPSYS','OWBSYS','ORDPLUGINS', 'ORDSYS','OUTLN',

   'SI\_INFORMTN\_SCHEMA','SYS','SYSMAN','SYSTEM', 'TSMSYS','WK\_TEST',

   'WKPROXY','WMSYS','XDB','APEX\_040000', 'APEX\_PUBLIC\_USER','DIP', 'WKSYS',

   'FLOWS\_30000','FLOWS\_FILES','MDDATA', 'ORACLE\_OCM', 'XS$NULL',

   'SPATIAL\_CSW\_ADMIN\_USR', 'SPATIAL\_WFS\_ADMIN\_USR', 'PUBLIC')

order by ind.table\_owner,

         ind.table\_name,

         ind.index\_name,

         ind\_col.column\_position;

## ---how to get rid of duplicate data

---occurances

SELECT store\_id, city, count(\*)

FROM store

GROUP BY store\_id, city

ORDER BY count(\*)

### ---rowid Pseudo Column --gives unique rowid to all columns

SELECT rowid, store\_id, city from store

### ---MIN(rowid) --- it gives one rowid to the group of same rows in the table

-- to view min rowid

SELECT MIN(rowid)

FROM store

GROUP BY store\_id, city

### --Now let's delete the repeated data

DELETE FROM store

WHERE rowid NOT IN (

    SELECT MIN(rowid)

    FROM STORE

    GROUP BY store\_id,city

)

## --System Tables

select \* from all\_tab\_columns

select \* from all\_tables

select \* from all\_objects

--- go to techonthenet.com for more system tables

## --creating synonym for a table

CREATE SYNONYM SAG\_TABLE

FOR employee

## --VIEWs, UNION/ALL

### --creating views

CREATE VIEW <give\_name\_to\_the\_view>

AS <give\_a\_leanthy\_code\_which\_worth\_saving\_for\_future\_use>

--eg:

CREATE VIEW manager\_v

AS

SELECT \* FROM employees

WHERE JOB = 'MANAGER'

## --Using Cascade Constraints and on Delete Cascade

CREATE TABLE supplier

( supplier\_id numeric(10) not null,

  supplier\_name varchar2(50) not null,

  contact\_name varchar2(50),

  CONSTRAINT supplier\_pk PRIMARY KEY (supplier\_id)

);

CREATE TABLE products

( product\_id numeric(10) not null,

  supplier\_id numeric(10) not null,

  CONSTRAINT fk\_supplier

    FOREIGN KEY (supplier\_id)

    REFERENCES supplier(supplier\_id)

);

BEGIN

INSERT INTO supplier values(10, '1st Supplier', 'John Dollan');

INSERT INTO supplier values(20, '2nd Supplier', 'Steve Reeves');

INSERT INTO supplier values(30, '3rd Supplier', 'Peter Marcello');

INSERT INTO products values(1, 10);

INSERT INTO products values(2, 10);

INSERT INTO products values(3, 10);

INSERT INTO products values(4, 20);

INSERT INTO products values(5, 20);

INSERT INTO products values(6, 30);

END;

## -- Cascade Constraints

-- To drop any parent table we need to remove constraints first.

--Cascade constraints helps to remove constraints from table.

ALTER TABLE supplier

DROP COLUMN supplier\_id

CASCADE CONSTRAINTS;

## Cascade Delete

A foreign key with cascade delete means that if a record in the parent table is deleted, then the corresponding records in the child table will automatically be deleted. This is called a cascade delete in oracle.

-- cascade delete

--for the cascade delete to work we need to create table specifying cascade delete while creating constraints as below:

CREATE TABLE products

    product\_id numeric(10) not null,

    supplier\_id numeric(10) not null,

CONSTRAINT fk\_supplier FOREIGN KEY (supplier\_id)

REFERENCES supplier(supplier\_id) ON DELETE CASCADE;

## -- dropping two rows at a same time

ALTER TABLE supplier

DROP (supplier\_id, supplier\_name, contact\_name)

CASCADE CONSTRAINT

## --Unused Columns

--Are the columns which exists in table but cannot be used ever

ALTER TABLE supplier

SET UNUSED COLUMN supplier\_name;

--the supplier\_name column can never be used ever again

### --Deleting/Dropping Unused column

ALTER TABLE supplier

DROP UNUSED COLUMNS;

## Creating External Table

First we create a directory like 'C:\datasg' in the machine we are using

Then we create a file with some data .dox or .txt

Then we query code as below:

**CREATE** DIRECTORY data\_dir **AS** 'C:\datasg';

**CREATE** **TABLE** emp\_load

(employee\_number **CHAR**(5),

employee\_dob **CHAR**(20),

employee\_last\_name **CHAR**(20),

employee\_first\_name **CHAR**(15),

employee\_middle\_name **CHAR**(15),

employee\_hire\_date **DATE**)

ORGANIZATION **EXTERNAL** --we ARE refering EXTERNAL file

(**TYPE** ORACLE\_LOADER

**DEFAULT** DIRECTORY data\_dir

**ACCESS** PARAMETERS

(RECORDS DELIMITED **BY** NEWLINE

FIELDS (employee\_number **CHAR**(2),

employee\_dob **CHAR**(20),

employee\_last\_name **CHAR**(18),

employee\_first\_name **CHAR**(11),

employee\_middle\_name **CHAR**(11),

employee\_hire\_date **CHAR**(10) date\_format **DATE** mask "mm/dd/yyyy"

)

)

LOCATION ('my\_data.txt')

);

**SELECT** \* **FROM** EMP\_LOAD

We cannot create constraints in external tables.

We cannot mark any column unused…

* <https://livesql.oracle.com/apex/f?p=590:1:14393925195020:CLEAR:NO:1:TUTORIAL_ID,P1_SHOW_LEARN_SIDEBAR:143127379594936522093884822782011566106,Y>
* <https://apex.oracle.com/pls/apex/f?p=4500:1003:116274824053627>:::::
* <https://www.udemy.com/course/the-complete-oracle-sql-certification-course/learn/lecture/20328991#content>
* <https://www.techonthenet.com/>