

# 12-OLAP In Oracle

#### **OLAP Features In Oracle**

- Some Features For Query Processing in ORACLE Include The Use OF ONLINE ANALYTICAL PROCESSING(OLAP) Upon The Data Base.
- OLAP Features Are Useful For Data Warehousing And Data Mart Applications.
- The OLAP Operations Are Performance Enhancements.
  - TOP\_N QUERIES
  - GROUP BY.
  - CUBE.
  - ROLLUP.

#### **ROLLUP Option:**

- It is A GROUP BY Operation And is Used To Produce Subtotals At Any Level Of The Aggregation.
- The Generated Sub Totals "Rolled Up" To Produce Grand Total.
- The Totaling is Based On A One Dimensional Data Hierarchy of Grouped Information.

**Syntax:** GROUP BY ROLLUP (Column 1, Column 2,...)

#### **Illustrations:**

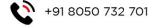
```
SQL> SELECT Deptno, SUM(SAL) FROM Emp GROUP BY ROLLUP(Deptno);
SQL> SELECT
           Job, SUM(Sal) FROM Emp GROUP BY ROLLUP(Job);
           Job, AVG(Sal) FROM Emp GROUP
SQL> SELECT
                                          BY ROLLUP (Job);
```

#### Passing Multiple Columns To ROLLUP:

 When Multiple Columns Are Passed To ROLLUP, The ROLLUP, Groups The Rows Into Blocks With The Same Column Values.

```
SQL> SELECT Deptno, Job, SUM(Sal) Salary FROM Emp
                                                    GROUP
                                                           BY ROLLUP (
Deptno, Job);
SQL> SELECT Job, Deptno, SUM(Sal) Salary FROM Emp
                                                    GROUP BY
ROLLUP (Job, Deptno);
SQL> SELECT Job, Deptno, AVG(Sal) Average FROM Emp GROUP BY
ROLLUP (Job, Deptno);
```

- NULL Values in The Output Of ROLLUP Operations Typically Mean That The Row Contains Subtotal Or Grant Total Information.
- Use NVL() Function For Proper Meaning.













#### **CUBE Option:**

- It is An Extension Similar To ROLLUP.
- Cube Allows To Take A Specified Set of Grouping Columns And Create Sub Totals For All Possible Combinations of Them.
- The Result of Cube is A Summary That Shows Subtotals For Every Combination of Columns OR Expressions in The Group By Clause.
- The Implementation of Cube is Also Called As N-Dimensional Cross Tabulation.

```
SQL>SELECT Deptno, Job, SUM(Sal) Salary FROM Emp GROUP BY
CUBE (Deptno, Job);
SQL> SELECT Job, Deptno, SUM(Sal) Salary FROM Emp GROUP BY CUBE(Job,
Deptno);
```

### **Applying GROUPING() Function:**

- The GROUPING() Function Accepts A Column And Returns 0 or 1.
- GROUPING() Function Returns 1 When The Column Value is NULL, And Returns 0 When the Column Value is NOT NULL.
- GROUPING() Function is Used Only Upon Queries That Use ROLLUP OR CUBE.
- GROUPING() Function is Useful When We Want To Display A Value When A NULL Would Otherwise Be Returned in OLAP Queries.

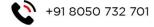
```
SQL> SELECT GROUPING (Deptno), Deptno, SUM(Sal)
                                                FROM
                                                     Emp
                                                          GROUP
ROLLUP (Deptno);
SQL> SELECT GROUPING(Job), Job, SUM(Sal) FROM Emp GROUP BY
ROLLUP (Job);
```

#### **DECODE Function:**

- It Is A Single Row Function.
- The Function Works On The Same Principle As The If-Then-Else.
- We Can Pass A Variable Number Of Values Into The Call Of The DECODE() Function.
- The First Item is Always The Name Of The Column That Need To Be Decoded.
- Once All Value-Substitute Pairs Have Been Defined, We Can Optionally Specify A Default Value.

SQL> SELECT DECODE (ColumnName, Value 1, Substitute 1, Value 2, Substitute 2,....ReturnDefault) FROM TableName;

- The Function Has No Restriction on The INPUT And OUTPUT Data Type.
- It is The Most Power full Function in Oracle.
- The Function Can Work For Only an Analysis That Considers an Equality Operator in The Logical Comparision.











# Dream. Create. Achieve.

SQL> SELECT Ename, Job, Sal, DECODE ( Deptno, 10, 'ACCOUNTING', 20, 'RESEARCH', 30, 'SALES', 40, 'OPERATIONS', 'OTHER') Departments FROM Emp ORDER BY Departments;

#### **GROUPING() With DECODE():**

The DECODE() Function Can Be Used To Convert 1 And 0 Returned Through GROUPING() into A Meaningful Output.

```
SQL> SELECT DECODE (GROUPING (Deptno), 1, 'All Departments', Deptno)
Departments, SUM(Sal) FROM Emp GROUP BY ROLLUP (Deptno);
SQL> SELECT DECODE (GROUPING (Job), 1, 'All Designations',
Designations, SUM(Sal) FROM Emp GROUP BY ROLLUP (Job);
```

### DECODE() and GROUPING() To Converting Multiple Column Values:

```
SQL> SELECT DECODE (GROUPING (Deptno), 1, 'All Departments', Deptno)
Departments, DECODE (GROUPING (Job), 1, 'All Designations', Job)
Designations, SUM(Sal) FROM Emp GROUP BY ROLLUP (Deptno, Job);
```

#### GROUPING() With DECODE() and CUBE:

SQL> SELECT DECODE (GROUPING (Deptno), 1, 'All Departments', Deptno) Departments, DECODE (GROUPING (Job), 1, 'All Designations', Job) Designations, SUM(Sal) FROM Emp GROUP BY CUBE(Deptno, Job);

#### Applying GROUPING SETS Clause:

The GROUPING SETS Clause is Used To Get The SUBTOTAL Rows.

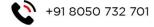
SQL> SELECT Deptno, Job, SUM(Sal) FROM Emp GROUP BY GROUPING SETS (Deptno, Job);

#### **Working With CASE Expressions:**

- The CASE Expression Can Be Used To Perform If-Then-Else Logic in SQL.
- CASE is Similar To DECODE But It is ANSI-Compliant.
- It Can be Used Even For Executing Conditions on range Based Comparision.
- Case Expressions Are of Two Types
  - SIMPLE CASE Expressions
  - SEARCHED CASE Expressions.

#### Simple CASE Expressions:

- These Expressions Are Used To Determine The Returned Value.
- They Work With Equality Comparision Only. Almost All Similar To DECODE.
- It Has A Selector Which Associate To The Compared Value Either From The Column or Constant.











# Dream, Create, Achieve,

The Value in The Selector is Used For Comparision With The Expressions Used in The WHEN Clause.

#### Syntax:

SQL> CASE Search Expr WHEN Expr 1 THEN Result 1 WHEN Expr 2 THEN Result 2 ELSE Default Result END;

#### **Illustration:**

SQL> SELECT Ename, Deptno, CASE Deptno WHEN 10 THEN 'ACCOUNTS' WHEN 20 THEN 'RESEARCH' WHEN 30 THEN 'SALES' WHEN 40 THEN 'OPERATIONS' 'NOT FOUND' END FROM Emp;

# **Searched CASE Expressions:**

- The Statement Uses Conditions To Determine The Returned Value.
- It Helps in Writing Multiple Conditions For Evaluation.
- Helps in Range Analysis of Values Also.

#### Syntax:

SQL> CASE WHEN Condition 1 THEN Result 1 WHEN Condition 2 THEN Result 2 WHEN Condition n THEN Resultn ELSE DefaultResult END;

#### **Illustration:**

SQL> SELECT Ename, Deptno, CASE WHEN 10 THEN 'ACCOUNTING' WHEN 20 THEN 'RESEARCH' WHEN 30 THEN 'SALES' WHEN 40 THEN 'OPERATIONS' 'Not Specified' END FROM Emp; SQL> SELECT Ename, Sal, CASE WHEN Sal>=800 AND Sal <= 2000 THEN 'LOWEST PAY' WHEN Sal>= 2001 AND Sal<=4000 THEN 'MODERATE PAY' ELSE 'HIGH PAY' END FROM Emp;

#### **Materialized Views:**

- Materialized Views Are Used in DATA WAREHOUSES AND DATA MARTS.
- They Are Used To Increase The Speed of Queries on Very Large Scale Databases.
- Queries Making Use of Materialized Views are...
  - Aggregations on A Single Table.
  - Joins Between Tables.
  - Aggregations And Joins Together.
- Materialized Views Can Be Used To Replicate Data.
- Prior To Materialized Views, The Concept of snapshot Was Implemented.

#### **Query Rewrite:**

Materialized Views Improve Query Performance By Pre Calculating Expensive Join And Aggregation Operations on The Database Prior To Execution Time And Stores The Results in The Database.







# Dream, Create, Achieve,

- The Query Optimizer Can Make Use of Materialized Views By Automatically Recognizing When An Existing Materialized View Can And Should Be Used To Satisfy A Request.
- After Above Process is Completed Then The Query Optimizer Transparently Rewrites The Request To Use The Materialized View.
- Queries Are Then Directed To The Materialized View And Not To The Underlying Detail Tables OR Views.
- Rewriting Queries To Use Materialized Views Rather Than Detail Relations, Results in A Significant Performance Gain.

### **Prerequisites For Materialized Views:**

#### **Privileges:**

```
SQL> GRANT QUERY REWRITE TO SCOTT;
SQL> ALTER SESSION SET QUERY REWRITE_ENABLED=TRUE;
Set the InitSid.ORA File:
OPTIMIZER MODE=CHOOSE
JOB QUEUE INTERVAL=3600
JOB QUEUE PROCESSES=1
QUERY REWRITE ENABLED=TRUE
QUERY REWRITE INTEGRITY=ENFORCED
```

#### **Materialized View With Aggregation:**

SQL> CREATE MATERIALIZED VIEW EMP SUM ENABLE QUERY REWRITE AS SELECT Deptno, Job, SUM(Sal) FROM Emp GROUP BY Deptno, Job;

#### Creating Optimizer Statistics and Refresing Materialized Views:

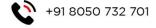
```
SQL> EXECUTE DBMS UTILITY.ANALYZE SCHEMA('SCOTT','ESTIMATE');
SQL>EXECUTE DBMS MVIEW.REFRESH('Emp Dept Sum');
```

# **Testing Materialized View:**

```
SQL> SET AUTOTRACE ON EXPLAIN;
SQL> SELECT Dname, Job, SUM(Sal) FROM Emp E, Dept D WHERE
E.Deptno=D.Deptno GROUP BY Dname, Job;
```

# **Putting the Things With ROLLUP:**

SQL> CREATE MATERIALIZED VIEW Emp Dept Agg ENABLE QUERY REWRITE AS SELECT Deptno, Job, COUNT(\*), SUM(Sal) FROM Emp GROUP BY ROLLUP (Deptno, Job);













#### **GROUPING ID() Function:**

- The Function is Used To FILTER ROWS Using A HAVING Clause To Exclude Rows That Do Not Contain A Subtotal OR Grand Total.
- The Function Accepts One OR More Columns And Returns The Decimal Equivalent of The GROUPING BIT VECTOR.
- The GROUPING BIT VECTOR is Computed By Combining The Results Of A Call To The GROUPING() Function For Each Column in Order.

# Computing The GROUPING BIT VECTOR:

- GROUPING() Function Returns 1 When The Column Value is NULL, Else Return 0, Based On This Concept.
- GROUPING ID() Returns 0, When Deptno And Job Are NOT NULL'S.
- GROUPING ID() Returns 1, If Deptno is NOT NULL And Job is NULL.
- GROUPING\_ID() Returns 2, If Deptno is NULL And Job is NULL.
- GROUPING\_ID() Returns 3, If Deptno is NULL And Job is NULL.

```
SQL> SELECT Deptno, Job, GROUPING(Deptno) GDPT, GROUPING(Job) GJOB,
GROUPING ID(Deptno, Job) GRPID, SUM(Sal) FROM Emp GROUP BY
ROLLUP(Deptno, Job);
SQL> SELECT Deptno, Job, GROUPING(Deptno) GDPT, GROUPING(Job) GJOB,
GROUPING ID (Deptno, Job) GRPID, SUM (Sal) FROM Emp GROUP BY CUBE (Deptno,
Job);
```

### **GROUPING ID() and HAVING Clause:**

SQL> SELECT Deptno, Job, GROUPING ID(Deptno, Job) GRPID, SUM(Sal) FROM Emp GROUP BY CUBE(Deptno, Job) HAVING GROUPING ID(Deptno, Job) >0;

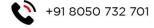
# Representing Column Multiple Times in a GROUP BY Clause:

A Column Can Be Represented Multiple Times in A GROUP BY Clause.

```
SQL> SELECT Deptno, Job, SUM(Sal) FROM Emp GROUP BY Deptno,
ROLLUP (Deptno, Job);
SQL> SELECT Deptno, Job, SUM(Sal) FROM Emp GROUP BY Deptno,
CUBE (Deptno, Job);
```

# Applying GROUP ID Function:

- The GROUP\_ID() Function is Used To Remove The duplicate Rows Returned By GROUP BY Clause.
- The GROUP\_ID() Does Not Accept Any Parameters.











# Dream. Create. Achieve.

If 'N' Duplicates Exist For A Particular Grouping, GROUP ID() Returns Numbers in The Range 0 To N-1.

SQL> SELECT Deptno, Job, GROUP ID(), SUM(Sal) FROM Emp GROUP BY ROLLUP (Deptno, Job); SQL> SELECCT Deptno, Job, GROUP ID(), SUM(Sal) FROM Emp GROUP Deptno, CUBE (Deptno, Job); SQL> SELECT Deptno, Job, GROUP ID(), SUM(Sal) FROM Emp GROUP BY Deptno, ROLLUP (Deptno, Job) HAVING GROUP ID()=0;

- Enhancing The Power Of OLAP Using Analytic Functions
- Analytic Functions Are Designed To Address Problems Such As
  - Calculate A Running Total.
  - Find Percentages With in A Group.
  - Top "N" Queries.
  - Compare "Moving Averages".
- Analytic Functions Add Grater performance To The Standard Query Processing.

#### **How Analytic Functions Work?**

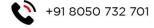
- Analytic Functions Compare An Aggregate Value Based on A Group of Rows.
- Analytic Functions Compute An Aggregate Value Based on A Group of Rows.
- The Differences From Ordinary Aggregate Functions To Analytic Functions Are, They Return Multiple Rows For Each Group.
- The Group of Rows Are Called As "WINDOW" And is Defined By The Analytic
- For Each Row, A Sliding Window of Rows Are Defined.
- The Window Define The Range of Rows Used To Perform The Calculation For The Current Row.
- Window Sizes Can Be Based Upon Either A Physical Number of Rows OR A Logical Interval Such As Time.
- Analytic Functions Are The Last Set Of Operations Performed in A Query, Except For The Final ORDER BY Clause.
- All JOINS, WHERE Clause, GROUP BY, And HAVING Clauses Are Completed Before The Analytic Functions Are Processed.
- Analytic Functions Can Appear Only in The SELECT List OR ORDER BY Clause.

#### Syntax:

AnalyticFunction (Arg 1, Arg 2, Ar3) OVER(Partition Clause

> ORDER BY Clause Windowing Clause

Analytic Function Takes 0 To 3 Arguments.













- The PARTITION BY Clause Logically Breaks A Single Result Set Into 'N' Groups.
- The Analytic Functions Are Applied For Each Group Independently, And They Are Reset For Each Group.
- The ORDER BY Clause Specifies How Data is Stored Within Each GROUP (Partition).
- The Output of Analytic Function is Affected By ORDER BY Clause.
- The Windowing Clause Gives Us A Way To Define A Sliding (OR) Analytic Function Will Operate, Within A Group.

### **Analytic Functions Categories:**

#### **Ranking Functions:**

They Enable US To Calculate Ranks, Percentiles and N-Tiles.

#### **Inverse Percentile Functions:**

• Enable To Calculate The Value Corresponding To A Percentile.

#### **Window Functions:**

Enable To Calculate Cumulative And Moving Aggregates.

#### **Reporting Functions:**

Enable To Calculate Area Like Market Shares.

#### LAG and LEAD Functions:

Enable To Get A Value in A Row Where That Row is A Certain Number of Rows Away From The Current Row.

#### First and Last Functions:

• Enable To Get The First And Last Values in An Ordered Group.

### **Linear Regression Functions:**

Enable To Fit An Ordinary-Least-Squares Regression Line To A Set of Number Pairs.

#### **Hypothetical Rank and Distribution Functions:**

Enable To Calculate The Rank And Percentile That A New Row Would Have If A Value is Inserted Into A Table.

# Normal Ranking:

SQL> SELECT Ename, Deptno, Sal, RANK() OVER (ORDER BY Sal) EmpRank FROM Emp GROUP BY Deptno, Ename, Sal ORDER BY Emprank; SQL> SELECT Ename, Deptno, Sal, DENSE RANK() OVER (ORDER BY Sal DESC) EmpRank FROM Emp GROUP BY Deptno, Ename, Sal ORDER BY EmpRank;











#### **Ranking With Partition:**

SQL> SELECT Ename, Deptno, Sal, RANK() OVER (PARTITION BY Deptno ORDER BY Sal DESC ) "TOP Sal" FROM Emp ORDER BY Deptno, Sal DESC; SQL> SELECT Ename, Deptno, Sal, DENSE\_RANK() OVER (PARTITION BY Deptno ORDER BY Sal DESC) "TOP Sal" FROM Emp) WHERE "TOP Sal" <= 3 ORDER BY Deptno, Sal DESC;

#### **Ranking With Partition And Filters:**

SQL> SELECT \* FROM (SELECT Ename, Deptno, Sal, RANK() OVER(PARTITION BY Deptno ORDER BY Sal DESC) "TOP Sal" FROM Emp ) WHERE "TOP Sal" <=3 ORDER BY Deptno, Sal DESC;

### **Applying Windows:**

- The Windowing Clause Gives A Way To Define A SLIDING OR ANCHORED WINDOW of Data, Upon Which The ANALYTIC FUNCTION Will Operate, Within A GROUP.
- The Default Window is An ANCHORED WINDOW That Simply Starts At The FIRST ROW of A GROUP And Continues To The CURRENT ROW.
- Window Can Be Used on
  - RANGES of Data Values.
  - ROWS OFFSET From The Current Row.
- Existence of An ORDER BY in An Analytic Function Will Add A DEFAULT WINDOW Clause of "RANGE UNBOUNDED PRECEDING", Which Gets All Rows in The Partition That Came Before The ORDER BY Clause.

#### **Row WINDOW:**

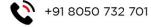
SQL> SELECT Deptno, Ename, Sal, SUM(SAL) OVER(PARTITION BY Deptno) ORDER BY Ename ROWS 2 PRECEDING) "Sliding Total" FROM Emp ORDER BY Deptno, Ename;

#### Range WINDOW:

- RANGE WINDOWS Collect ROWS Together Based on A WHERE Clause.
- The RANGE UNITS Can Either Be Numeric Comparisions OR Date Comparisions.
- Range Units Are Not Valid if Data Type is Other Than Number OR Dates.

SQL> SELECT Ename, Hiredate, HireDate-100, Count(\*) OVER (ORDER BY Hiredate ASC RANGE 100 PRECEDING) HireCnt FROM Emp ORDER BY HireDate ASC;

SQL> SELECT Ename, HireDate, Sal, AVG(Sal) OVER (ORDER BY HireDate ASC RANGE 100 PRECEDING) AvgSal FROM Emp ORDER BY HireDate ASC;











# **Accessing ROWS Around Current Row:**

#### LAG Function:

- Lag Provides Access To Move Than One Row of A Table At The Same Time Without The Use of SELF JOIN.
- Given A Series of Rows Returned From A Query And A Position of The Cursor, LAG Function Provides Access To A Row At A Given Offset Prior To That Position.
- If Offset is Not Provided Than Default Offset is Considered As 1.
- The Optional Default Value is Returned if The Offset Goes Beyoned The Scope of The Window.
- If Default is Not Specified, Than The Default is Considered As NULL.

#### Syntax:

```
LAG(ValueExpr(, Offset)(, DEFAULT))
OVER ((Query-Partition-Clause)
ORDER BY CLAUSE)
```

#### **LEAD Function:**

- LEAD Provides Access To More Than One Row of A Table At The Same Time Without The Use of A SELF JOIN.
- Given A Series of Rows Returned From A Query And A Position of The Cursor, LEAD Function Provides Access To A Given Physical Offset Beyond That Position.

#### Syntax:

```
LEAD(ValueExpr(, Offset)(, DEFAULT))
OVER ((Query-Partition-Clause)
ORDER BY CLAUSE)
```

#### Illustration:

```
SQL> SELECT Ename, Hiredate, Sal, LAG(Sal,1,0) OVER(ORDER BY Hiredate)
Presal FROM Emp;
SQL> SELECT Ename, Hiredate, Sal, LEAD(Sal,1,0) OVER(ORDER BY Hiredate)
NextSal FROM Emp;
```

#### **FIRST VALUE Function:**

- FIRST VALUE Returns The First Value in An Ordered Set of Values.
- If The First Value in The Set is NULL, Then The Function Returns NULL Unless IGNORE NULLS is Specified.
- IGNORE NULLS Setting is Useful For Data Densification, If Specified Then FIRST\_VALUE Returns The First Non-Null Value in The Set, Else NULL is Returned.

Syntax: FIRST VALUE (EXPR IGNORE NULLS) OVER (Analytic Clause)









# Dream, Create, Achieve.

#### **LAST VALUE Function:**

- LAST VALUE Returns The LAST Value in An Ordered Set of Values.
- If The Last Value in The Set is NULL, Then The Function Returns NULL Unless IGNORE NULLS is Specified.
- IGNORE NULLS Setting is Useful For Data Densification, If Specified Then LAST\_VALUE Returns The First Non-Null Value in the Set, Else NULL is Returned.

Syntax: LAST VALUE (EXPR IGNORE NULLS) OVER (Analytic Clause)

# **Illustration:**

SQL> SELECT Ename, Deptno, Sal, FIRST VALUE(Ename) OVER (PARTITION BY Deptno ORDER BY Sal DESC) Max Sal Name FROM Emp ORDER BY Deptno, SQL> SELECT Ename, Deptno, Sal, FIRST VALUE(Ename) OVER (ORDER BY Sal ASC) Min Sal Name FROM (SELECT \* FROM Emp WHERE Deptno=30);

#### **ROW\_NUMBER Function:**

 The ROW\_NUMBER Function Assigns a Unique Number Sequentially, Starting From 1, As Defined By ORDER BY To Each Row Within The partition.

#### Syntax:

ROW NUMBER() OVER([Query Partition Clause] ORDER BY Clause) SQL> SELECT ROW NUMBER() OVER (PARTITION BY Deptno ORDER BY Sal DESC NULLS LAST) RowNo, Ename, Sal, Deptno, FROM Emp;

#### **CROSSTAB OR PIVOT Queries:**

A CROSSTAB Query Can Be Used To Get A Result With Rows And Columns in A Matrix Form.

SQL> SELECT Deptno, MAX(DECODE(Seqno, 1, Ename, NULL)) First, MAX(DECODE(Seqno, 2, Ename, NULL)) Second, MAX(DECODE(Seqno, 3, Ename, NULL) Third FROM ( SELECT Deptno, Ename, ROW NUMBER () OVER (PARTITION BY Deptno ORDER BY DESC NULLS LAST) SeqNo FROM Emp) WHERE SeqNo <= 3 GROUP BY Deptno;

#### Centered SUM And Centered AVERAGE:

SQL> SELECT Ename, SUM(Sal) SalAmt, ROUND(SUM(SUM(Sal)) OVER (ORDER BY Deptno ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING), 2) CSum, ROUND (AVG (SUM (Sal)) OVER (ORDER BY Deptno ROWS BETWEEN 1 PRECEDINGAND 1 FOLLOWING),2) Cavg FROM Emp GROUP BY Deptho, Ename ORDER BY Deptno;

