

Data warehousing in Oracle

Materialized views and SQL extensions
to analyze data in Oracle data warehouses



1

SQL extensions for data warehouse analysis



2

Available OLAP functions

- Computation windows
 - window
- Ranking functions
 - rank, dense rank, ...
- Group by clause extensions
 - rollup, cube, ...



Oracle data warehousing - 3

3

Physical aggregation example

- Example table
 - SALES(**City**, **Date**, Amount)
- Select, separately for each city, for each date the amount and the average amount over the current and the previous two rows



Oracle data warehousing - 4

4

Physical aggregation example

```
SELECT Date, Amount,  
       AVG(Amount) OVER (  
         PARTITION BY City  
         ORDER BY Date  
         ROWS 2 PRECEDING  
       ) AS MovingAverage  
FROM Sales  
ORDER BY Date;
```



Oracle data warehousing - 5

5


Logical aggregation example

- Example table
 - SALES(**City**, **Date**, Amount)
- Select, separately for each city, for each date the amount and the average amount over the current row and the sales of the two previous days



Oracle data warehousing - 6

6




Logical aggregation example

```


SELECT Date, Amount,
       AVG(Amount) OVER (
         PARTITION BY City
         ORDER BY Date
         RANGE BETWEEN INTERVAL '2'
         DAY PRECEDING AND CURRENT ROW
       ) AS Last3DaysAverage
FROM Sales
ORDER BY Date;

```




Oracle data warehousing - 7

7




Example tables

- Schema
 - SUPPLIERS(**Cod S**, Name, SLocation)
 - ITEM(**Cod I**, Type, Color, Weight)
 - PROJECTS(**Cod P**, Name, PLocation)
 - FACTS(**Cod S**, **Cod I**, **Cod P**, SoldAmount)




Oracle data warehousing - 8

8




Ranking example

- Select for each item the total amount sold and the ranking according to the total amount sold



Oracle data warehousing - 9

9




Ranking example

```


SELECT COD_I, SUM(SoldAmount),
       RANK() OVER (
         ORDER BY SUM(SoldAmount)
       ) AS SalesRank
FROM Facts
GROUP BY COD_I;

```




Oracle data warehousing - 10

10




Ranking example

| COD_I | SUM(SoldAmount) | DenseSalesRank |
|-------|-----------------|----------------|
| I2 | 300 | 1 |
| I5 | 1100 | 2 |
| I4 | 1300 | 3 |
| I6 | 1300 | 3 |
| I1 | 1900 | 5 |
| I3 | 4500 | 6 |



Oracle data warehousing - 11

11




Dense ranking

```


SELECT COD_I, SUM(SoldAmount),
       DENSE_RANK() OVER (
         ORDER BY SUM(SoldAmount)
       ) AS DenseSalesRank
FROM Facts
GROUP BY COD_I;

```




Oracle data warehousing - 12

12




Ranking example

| COD_I | SUM(SoldAmount) | DenseSalesRank |
|-------|-----------------|----------------|
| I2 | 300 | 1 |
| I5 | 1100 | 2 |
| I4 | 1300 | 3 |
| I6 | 1300 | 3 |
| I1 | 1900 | 4 |
| I3 | 4500 | 5 |




Oracle data warehousing - 13

13




Double ranking

- Select for each item the code, the weight, the total amount sold, the ranking according to the weight and the ranking according to the total amount sold



Oracle data warehousing - 14

14




Double ranking

```


SELECT Item.COD_I, Item.Weight,
       RANK() OVER (ORDER BY Item.Weight
                   ) AS WeightRank
       RANK() OVER (ORDER BY SUM(SoldAmount)
                   ) AS SalesRank
FROM Facts, Item
WHERE Facts.COD_I = Item.COD_I
GROUP BY Item.COD_I, Item.Weight
ORDER BY WeightRank;

```




Oracle data warehousing - 15

15




Double ranking

| COD_I | Weigh | SUM(SoldAmount) | WeightRank | SalesRank |
|-------|-------|-----------------|------------|-----------|
| I1 | 12 | 1900 | 1 | 5 |
| I5 | 12 | 1100 | 1 | 2 |
| I4 | 14 | 1300 | 3 | 3 |
| I2 | 17 | 300 | 4 | 1 |
| I3 | 17 | 4500 | 4 | 6 |
| I6 | 19 | 1300 | 6 | 3 |




Oracle data warehousing - 16

16




Top N ranking selection

- Select
 - the **top two** most sold items
 - their code
 - their weight
 - the total amount sold
 - and their ranking according to the total amount sold




Oracle data warehousing - 17

17



Top N ranking selection

- Returning only the top two items can be performed by nesting the ranking query inside an outer query
- The outer query uses the nested ranking query as a table (after the FROM clause)
- The outer query selects the requested values of the rank field



Oracle data warehousing - 18

18

Top N ranking selection

```

SELECT * FROM
  (SELECT COD_I, SUM(SoldAmount),
    RANK() OVER (ORDER BY SUM(SoldAmount))
      AS SalesRank
  FROM Facts
  GROUP BY COD_I)
WHERE SalesRank <= 2;

SUPPLIERS(Cod_S, Name, SLocation)
ITEM(Cod_I, Type, Color, Weight)
PROJECTS(Cod_P, Name, PLocation)
FACTS(Cod_S, Cod_I, Cod_P, SoldAmount)

```

DBG Oracle data warehousing - 19

19

Top N ranking selection

```

SELECT * FROM
  (SELECT COD_I, SUM(SoldAmount),
    RANK() OVER (ORDER BY SUM(SoldAmount))
      AS SalesRank
  FROM Facts
  GROUP BY COD_I)
WHERE SalesRank <= 2;

```

↑
Temporary table created at runtime and dropped at the end of the outer query

DBG Oracle data warehousing - 20

20

ROW_NUMBER

- ROW_NUMBER
 - in each partition it assigns a progressive number to each row
- Partition the items according to their type and enumerate in progressive order the data in each partition. In each partition the rows are sorted according to the weight

DBG Oracle data warehousing - 21

21

ROW_NUMBER

```

SELECT Type, Weight, ROW_NUMBER OVER (
  PARTITION BY Type
  ORDER BY Weight
) AS RowNumberWeight
FROM Item;

```

DBG Oracle data warehousing - 22

22

ROW_NUMBER

| Type | Weight | RowNumberWeight | |
|-------|--------|-----------------|-------------|
| Bar | 12 | 1 | Partition 1 |
| Gear | 19 | 1 | Partition 2 |
| Screw | 12 | 1 | Partition 3 |
| Screw | 14 | 2 | |
| Screw | 16 | 3 | |
| Screw | 16 | 4 | |
| Screw | 16 | 5 | |
| Screw | 16 | 6 | |
| Screw | 17 | 7 | |
| Screw | 17 | 8 | |
| Screw | 18 | 9 | |
| Screw | 20 | 10 | |

DBG Oracle data warehousing - 23

23

CUME_DIST

- CUME_DIST
 - in each partition it assigns a weight between 0 and 1 to each row according to the number of values which precede the value of the attribute employed for the sorting in the partition
- Given a partition with N rows, for each row x the CUME_DIST is computed as follows:
 - CUME_DIST(x) = number of values, which precede or have the same value of the attribute employed for the sorting, divided by N

DBG Oracle data warehousing - 24

24



CUME_DIST example

- Partition the items according to the type and sort in each partition according to the weight of items. Assign to each row the corresponding value of CUME_DIST



Oracle data warehousing - 25

25



CUME_DIST example

```
SELECT Type, Weight, CUME_DIST() OVER (
  PARTITION BY Type
  ORDER BY Weight
  ) AS CumeWeight
FROM Item;
```



Oracle data warehousing - 26

26



Example CUME_DIST

| Type | Weight | RowNumber | Weight | | |
|-------|--------|-----------|--------|----------|-------------|
| Bar | 12 | 1 | | (=1/1) | Partition 1 |
| Gear | 19 | 1 | | (=1/1) | Partition 2 |
| Screw | 12 | | 0.1 | (=1/10) | Partition 3 |
| Screw | 14 | | 0.2 | (=2/10) | |
| Screw | 16 | | 0.6 | (=6/10) | |
| Screw | 16 | | 0.6 | (=6/10) | |
| Screw | 16 | | 0.6 | (=6/10) | |
| Screw | 16 | | 0.6 | (=6/10) | |
| Screw | 17 | | 0.8 | (=8/10) | |
| Screw | 17 | | 0.8 | (=8/10) | |
| Screw | 18 | | 0.9 | (=9/10) | |
| Screw | 20 | | 1 | (=10/10) | |



Oracle data warehousing - 27

27



NTILE

- NTILE(n)
 - Allows splitting each partition in n subgroups (if it is possible) containing the same number of records. An identifier is associated to each subgroup.



Oracle data warehousing - 28

28



NTILE example

- Partition the items according to the type and split each partition in 3 sub-groups with the same number of data. In each partition the rows are ordered by the weight of items



Oracle data warehousing - 29

29




NTILE example

```
SELECT Type, Weight, NTILE(3) OVER (
  PARTITION BY Type
  ORDER BY Weight
  ) AS Ntile3Weight
FROM ITEM;
```




Oracle data warehousing - 30

30



NTILE example

| Type | Weight | RowNumberWeight | |
|-------|--------|-----------------|-------------|
| Bar | 12 | 1 | Partition 1 |
| Gear | 19 | 1 | Partition 2 |
| Screw | 12 | 1 | Partition 3 |
| Screw | 14 | 1 | Subgroup 1 |
| Screw | 16 | 1 | |
| Screw | 16 | 1 | |
| Screw | 16 | 2 | Subgroup 2 |
| Screw | 16 | 2 | |
| Screw | 17 | 2 | |
| Screw | 17 | 3 | Subgroup 3 |
| Screw | 18 | 3 | |
| Screw | 20 | 3 | |



Oracle data warehousing - 31


31

Materialized views




Oracle data warehousing - 32

32




Materialized views

- The result is **precomputed** and stored on the disk
- They improve **response times**
 - Aggregations and joins are precomputed
- Usually they are associated to queries with **aggregations**
- They may be used also for non aggregating queries
- Materialized views can be used as a **table** in any query




Oracle data warehousing - 33

33




Query rewriting

- The DBMS can change the execution of a query to **optimize performance**
- Materialized views can be **automatically** used by the DBMS **without user intervention**
 - Materialized views help answering queries very similar to the query which created them



Oracle data warehousing - 34


34



Creating materialized views


```

CREATE MATERIALIZED VIEW Name
[BUILD {IMMEDIATE|DEFERRED}]
[REFRESH {COMPLETE|FAST|FORCE|NEVER}
      {ON COMMIT|ON DEMAND}]
[ENABLE QUERY REWRITE]
AS
  Query
  
```




Oracle data warehousing - 35

35



Creating materialized views

- ***Name***
 - materialized view **name**
- ***Query***
 - query associated to the materialized view (i.e., query that **creates** the materialized view)



Oracle data warehousing - 36

36



Creating materialized views

- BUILD
 - IMMEDIATE
 - **creates** the materialized view and **immediately loads** the query results into the view
 - DEFERRED
 - **creates** the materialized view but does **not** immediately load the query results into the view



Oracle data warehousing - 37

37



Creating materialized views

- REFRESH
 - COMPLETE
 - **recomputes** the query result by executing the query on **all data**
 - FAST
 - **updates** the content of the materialized view using the changes **since the last refresh**



Oracle data warehousing - 38

38



Creating materialized views

- REFRESH
 - FORCE
 - when possible, the **FAST** refresh is performed
 - otherwise the **COMPLETE** refresh is performed
 - NEVER
 - the content of the materialized view is **not updated** using Oracle standard procedures



Oracle data warehousing - 39

39



Materialized views options

- ON COMMIT
 - an **automatic refresh** is performed when SQL operations affect the materialized view content
- ON DEMAND
 - the refresh is performed only upon explicit **request** of the user issuing the command
 - DBMS_MVIEW.REFRESH



Oracle data warehousing - 40

40



Materialized views options

- ENABLE QUERY REWRITE
 - enables the DBMS to automatically use the materialized view as a basic block (i.e., a table) to improve other queries performance
 - available only in the high-end versions of DBMS (e.g., not available in Oracle Express)
 - when unavailable, the query must be rewritten by the user to access the materialized view



Oracle data warehousing - 41

41



Creation constraints

- Depending on the DBMS and the query, you can create a materialized view associated to the query if some constraints are satisfied
 - constraints on the aggregating attributes
 - constraints on the tables and the joins
 - etc.
 - you must be aware of the constraint existence!



Oracle data warehousing - 42

42



Materialized view example

- Tables
 - SUPPLIERS(**Cod_S**, Name, SLocation)
 - ITEM(**Cod_I**, Type, Color)
 - PROJECTS(**Cod_P**, Name, PLocation)
 - FACTS(**Cod_S**, **Cod_I**, **Cod_P**, Measure)



Oracle data warehousing - 43

43



Materialized view example

- The materialized view query is
 - SELECT Cod_S, Cod_I, SUM(Measure)
FROM Facts
GROUP BY Cod_S, Cod_I;
- Options
 - Immediate data loading
 - Complete refresh only upon user request
 - The DBMS can use the materialized view to optimize other queries



Oracle data warehousing - 44

44



Materialized view example

```
CREATE MATERIALIZED VIEW Sup_Item_Sum
BUILD IMMEDIATE
REFRESH COMPLETE ON DEMAND
ENABLE QUERY REWRITE
AS
SELECT Cod_S, Cod_I, SUM(Measure)
FROM Facts
GROUP BY Cod_S, Cod_I;
```



Oracle data warehousing - 45

45



Fast refresh

- Requires proper structures to log changes to the tables involved by the materialized view query
- MATERIALIZED VIEW LOG
 - there is a log for each table of a materialized view
 - each log is associated to a single table and some of its attributes
 - it stores changes to the materialized view table



Oracle data warehousing - 46

46



Fast refresh

- The REFRESH FAST option can be used only if the materialized view query satisfies some constraints
 - materialized view **logs** for the tables and attributes of the query must exist
 - when the GROUP BY clause is used, in the SELECT statement an **aggregation** function must be specified (e.g., COUNT, SUM, ...)



Oracle data warehousing - 47

47



Materialized view log example

- Create a materialized view log associated to the FACTS table, on Cod_S, Cod_I and MEASURE attributes
 - enable the options SEQUENCE and ROWID
 - enable new values handling



Oracle data warehousing - 48

48



Materialized view log example

```
CREATE MATERIALIZED VIEW LOG
  ON Facts
  WITH SEQUENCE, ROWID
  (Cod_S, Cod_I, Measure)
  INCLUDING NEW VALUES;
```



Oracle data warehousing - 49

49



Example with fast refresh option

- The materialized view query is
 - SELECT Cod_S, Cod_I, SUM(Measure)
 - FROM Facts
 - GROUP BY Cod_S, Cod_I;
- Options
 - Immediate data loading
 - Automatic fast refresh
 - The DBMS can use the materialized view to optimize other queries



Oracle data warehousing - 50

50



Example with fast refresh option

```
CREATE MATERIALIZED VIEW LOG ON Facts
  WITH SEQUENCE, ROWID (Cod_S, Cod_I, Measure)
  INCLUDING NEW VALUES;

CREATE MATERIALIZED VIEW Sup_Item_Sum2
  BUILD IMMEDIATE
  REFRESH FAST ON COMMIT
  ENABLE QUERY REWRITE
  AS
  SELECT Cod_S, Cod_I, SUM(Measure)
  FROM Facts
  GROUP BY Cod_S, Cod_I;
```



Oracle data warehousing - 51

51



Fast refreshing materialized views

- The user or a system job can request the materialized view update by issuing the command
 - DBMS_MVIEW.REFRESH('view', {'C'/'F'})
 - *view*: name of the view to update
 - 'C': COMPLETE refresh
 - 'F': FAST refresh



Oracle data warehousing - 52

52



Fast refreshing materialized views

- Example
 - COMPLETE refresh of the materialized view "Sup_Item_Sum"
- ```
EXECUTE DBMS_MVIEW.REFRESH('Sup_Item_Sum', 'C');
```



Oracle data warehousing - 53

53



## Changing and deleting views

- Changing
  - ALTER MATERIALIZED VIEW *name*
    - options*;
- Deleting
  - DROP MATERIALIZED VIEW *name*;



Oracle data warehousing - 54

54



## Analyzing materialized views

- The command `DBMS_MVIEW.EXPLAIN_MVIEW` allows the materialized view inspection
  - refresh type
  - operations on which the fast refresh is enabled
  - query rewrite status (enabled, allowed, disabled)
  - errors



Oracle data warehousing - 55

55



## Execution plan

- Analyzing the execution plan of frequent queries allows us to know whether materialized views are used
- Query execution plans can be shown
  - enabling the auto trace in `SQLPLUS> set autotrace on;`
  - clicking on the **Explain** link in the Oracle web interface

| Operation        | Options  | Object | Rows  | Time | Cost | Bytes   |
|------------------|----------|--------|-------|------|------|---------|
| SELECT STATEMENT |          |        | 5     | 1    | 14   | 85      |
| HASH             | GROUP BY |        | 5     | 1    | 14   | 85      |
| HASH JOIN        |          |        | 7,809 | 1    | 13   | 101,517 |



Oracle data warehousing - 56

56