

## Data warehouse in Oracle

Viste materializzate ed estensioni al linguaggio SQL per l'analisi dei dati presenti nei data warehouse

Estensioni al linguaggio SQL per l'analisi dei dati presenti nei data warehouse

## Funzioni OLAP disponibili

- Finestre di calcolo
  - *window*
- Funzioni di ranking
  - *rank, dense rank, ...*
- Estensione della clausola group by
  - *rollup, cube, ...*

## Tabella d'esempio

- Schema tabella
  - VENDITE(Città, Data, Importo)

## Esempio di raggruppamento a livello fisico

- Selezionare per ogni data l'importo e la media dell'importo considerando la riga corrente e le due righe che la precedono

## Esempio di raggruppamento a livello fisico

```
SELECT Data, Importo, AVG(Importo) OVER (  
    ORDER BY Data  
    ROWS 2 PRECEDING  
    ) AS MediaMobile  
FROM Vendite  
ORDER BY Data;
```

## Esempio di raggruppamento a livello logico

- Selezionare per ogni data l'importo e la media dell'importo considerando la riga corrente e le vendite avvenute nei due giorni precedenti

## Esempio di raggruppamento a livello logico

```
SELECT Data, Importo, AVG(Importo) OVER (  
    ORDER BY Data  
    RANGE BETWEEN INTERVAL '2'  
    DAY PRECEDING AND CURRENT  
    ROW  
    ) AS MediaUltimi3Giorni  
FROM Vendite  
ORDER BY Data;
```

## Tabelle d'esempio

- Schema tabelle
  - FRN(COD\_F, Nome, Sede\_F, )
  - ART(COD\_A, Tipo, Colore, Peso)
  - PRG(COD\_P, Nome, Sede\_P)
  - FAP(COD\_F, COD\_P, COD\_A, Q)

## Esempio di ranking

- Selezionare per ogni articolo la quantità totale venduta e il "ranking" in funzione delle quantità totali vendute per ogni articolo

## Esempio di ranking

```
SELECT COD_A, SUM(Q), RANK() OVER (  
    ORDER BY SUM(Q)  
    ) AS RankVendite  
FROM FAP  
GROUP BY COD_A;
```

## Esempio di ranking

COD_A	SUM(Q)	RankVendite
A2	300	1
A5	1100	2
A4	1300	3
A6	1300	3
A1	1900	5
A3	4500	6

## Esempio di dense ranking

```
SELECT COD_A, SUM(Q), DENSE_RANK()
OVER (
ORDER BY SUM(Q)
) AS DenseRankVendite
FROM FAP
GROUP BY COD_A;
```

## Esempio di dense ranking

COD_A	SUM(Q)	DenseRankVendite
A2	300	1
A5	1100	2
A4	1300	3
A6	1300	3
A1	1900	4
A3	4500	5

## Esempio di doppio ranking

- Selezionare per ogni articolo il codice, il peso, la quantità totale venduta, il ranking in funzione del peso e il ranking in funzione delle quantità totali vendute per ogni articolo

## Esempio doppio ranking

```
SELECT ART.COD_A, ART.Peso, SUM(Q),
RANK() OVER (ORDER BY ART.Peso
) AS R_Peso,
RANK() OVER (ORDER BY SUM(Q)
) AS R_Vend
FROM FAP,ART
WHERE FAP.COD_A=ART.COD_A
GROUP BY ART.COD_A, ART.Peso
ORDER BY R_Peso;
```

## Esempio di doppio ranking

COD_A	PESO	SUM(Q)	R_Peso	R_Vend
A1	12	1900	1	5
A5	12	1100	1	2
A4	14	1300	3	3
A2	17	300	4	1
A3	17	4500	4	6
A6	19	1300	6	3

## Selezione Top N nel ranking

- Se voglio solo i primi due articoli nel ranking posso usare l'interrogazione che calcola il ranking come sottointerrogazione e poi fare una selezione in base al campo di ranking
  - La sottointerrogazione è specificata tra parentesi tonde subito dopo la FROM e viene utilizzata come se fosse una tabella

## Selezione Top N nel ranking

```
SELECT * FROM
  (SELECT COD_A, SUM(Q),
    RANK() OVER (ORDER BY SUM(Q))
    AS RankVendite
  FROM FAP
  GROUP BY COD_A)
WHERE RankVendite<=2;
```

## Selezione Top N nel ranking

```
SELECT * FROM
  (SELECT COD_A, SUM(Q),
    RANK() OVER (ORDER BY SUM(Q))
    AS RankVendite
  FROM FAP
  GROUP BY COD_A)
WHERE RankVendite<=2;
```

↑  
Viene gestita come una tabella temporanea creata a runtime ed eliminata alla conclusione dell'esecuzione della query principale

## Selezione Top N nel ranking

COD_A	SUM(Q)	RankVendite
A2	300	1
A5	1100	2

## ROW\_NUMBER

- ROW\_NUMBER
  - all'interno di ogni partizione assegna un numero progressivo ad ogni riga

## Esempio ROW\_NUMBER

- Partizionare gli articoli in base alla tipologia ed enumerare in modo progressivo i dati all'interno di ogni partizione. All'interno di ogni partizione i dati sono ordinati in base al peso.

## Esempio ROW\_NUMBER

```
SELECT Tipo, Peso, ROW_NUMBER OVER (
  PARTITION BY Tipo
  ORDER BY Peso
) AS RowNumberPeso
FROM ART;
```

## Esempio ROW\_NUMBER

Tipo	Peso	RowNumberPeso	
Barra	12	1	Partizione 1
Ingranaggio	19	1	Partizione 2
Vite	12	1	Partizione 3
Vite	14	2	
Vite	16	3	
Vite	16	4	
Vite	16	5	
Vite	16	6	
Vite	17	7	
Vite	17	8	
Vite	18	9	
Vite	20	10	

## CUME\_DIST

- CUME\_DIST
  - all'interno di ogni partizione (gruppo) viene assegnato un peso tra 0 e 1 ad ogni riga in funzione del numero di valori che precedono il valore assunto dal campo usato per effettuare l'ordinamento all'interno delle partizioni

## CUME\_DIST

- Data una partizione contenente N dati, per ogni riga x calcolo CUME\_DIST come
  - $CUME\_DIST(x) = \frac{\text{numero valori che precedono } x}{N}$

## Esempio CUME\_DIST

- Partizionare gli articoli in base alla tipologia degli articoli ed effettuare un ordinamento nei gruppi in base al peso degli articoli. Associare ad ogni riga il rispettivo valore di CUME\_DIST

## Esempio CUME\_DIST

```
SELECT Tipo, Peso, CUME_DIST() OVER (
    PARTITION BY Tipo
    ORDER BY Peso
) AS CumePeso
FROM ART;
```

## Esempio CUME\_DIST

Tipo	Peso	CumePeso	
Barra	12	1 (= 1/1)	Partizione 1
Ingranaggio	19	1 (= 1/1)	Partizione 2
Vite	12	.1 (= 1/10)	Partizione 3
Vite	14	.2 (= 2/10)	
Vite	16	.6 (= 6/10)	
Vite	16	.6 (= 6/10)	
Vite	16	.6 (= 6/10)	
Vite	16	.6 (= 6/10)	
Vite	17	.8 (= 8/10)	
Vite	17	.8 (= 8/10)	
Vite	18	.9 (= 9/10)	
Vite	20	1 (= 10/10)	

## NTILE

- NTILE(*n*)
  - permette di dividere ogni partizione in *n* sottogruppi (se possibile) ognuno con lo stesso numero di dati/record. Ad ogni sottogruppo viene associato un numero identificativo

## Esempio NTILE

- Partizionare gli articoli in base alla tipologia ed effettuare un'ulteriore suddivisione in 3 sottogruppi ognuno contenente lo stesso numero di dati. All'interno di ogni partizione i dati sono ordinati in base al peso degli articoli

## Esempio NTILE

```
SELECT Tipo, Peso, NTILE(3) OVER (  
    PARTITION BY Tipo  
    ORDER BY Peso  
    ) AS Ntile3Peso  
FROM ART;
```

## Esempio NTILE

Tipo	Peso	Ntile3Peso	
Barra	12	1	Partizione 1
Ingranaggio	19	1	Partizione 2
Vite	12	1	Partizione 3
Vite	14	1	
Vite	16	1	Sottogruppo 1
Vite	16	1	
Vite	16	2	
Vite	16	2	Sottogruppo 2
Vite	17	2	
Vite	17	3	
Vite	18	3	Sottogruppo 3
Vite	20	3	

## Viste materializzate

## Viste materializzate

- Viste materializzate
  - sono viste il cui risultato viene precalcolato e memorizzato su disco
  - permettono di velocizzare i tempi di risposta
    - precalcolo degli aggregati, join, ...
  - solitamente sono associate a interrogazioni che operano aggregazioni
  - possono essere usate anche per interrogazioni che non operano aggregazioni

## Viste materializzate

- La vista materializzata può essere usata in qualunque interrogazione di selezione come se fosse una tabella

## Viste materializzate e riscrittura delle interrogazioni

- “Queries rewriting”
  - il DBMS può trasformare le interrogazione al fine di ottimizzarne l’esecuzione
  - le viste materializzate possono essere usate *automaticamente* in fase di riscrittura delle interrogazioni in modo *trasparente all’utente*
    - usate per risolvere interrogazioni simili a quella alla quale sono associate

## Creazione viste materializzate

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

## Creazione viste materializzate

- *Name*: nome della vista materializzata
- *Query*: interrogazione associata alla vista materializzata

## Creazione viste materializzate

- BUILD
  - IMMEDIATE
    - crea la vista materializzata e carica immediatamente i risultati dell’interrogazione al suo interno
  - DEFERRED
    - crea la vista materializzata ma non carica i dati associati all’interrogazione al suo interno

## Creazione viste materializzate

- REFRESH
  - COMPLETE
    - ricalcola il risultato dell’interrogazione eseguendo l’interrogazione su tutti i dati
  - FAST
    - aggiorna il contenuto della vista materializzata basandosi sulle variazioni avvenute dall’ultima operazione di refresh ad ora

## Creazione viste materializzate

- REFRESH
  - FORCE
    - se possibile viene eseguito il refresh in modalità FAST
    - altrimenti viene usata la modalità COMPLETE
  - NEVER
    - il contenuto della vista non viene aggiornata con le procedure standard di Oracle

## Creazione viste materializzate

- Opzioni
  - ON COMMIT
    - refresh effettuato automaticamente quando le operazioni sql eseguite comportano una variazione del contenuto della vista materializzata
  - ON DEMAND
    - refresh effettuato solo su richiesta esplicita dell'utente usando la procedura DBMS\_MVIEW.REFRESH

## Creazione viste materializzate

- ENABLE QUERY REWRITE
  - abilita il DBMS ad utilizzare la vista materializzata come blocco base per eseguire “più velocemente” altre interrogazioni

## Esempio di vista materializzata

- Schema tabelle
  - FRN(Cod\_F, Nome, Sede\_F )
  - ART(Cod\_A, Tipo, Colore)
  - PRG(Cod\_P, Nome, Sede\_P)
  - FAP(Cod\_F, Cod\_P, Cod\_A, Q)

## Esempio di vista materializzata

- Voglio “materializzare” l’interrogazione
  - SELECT Cod\_F, Cod\_A, SUM(Q)  
FROM FAP  
GROUP BY Cod\_F, Cod\_A;
- Opzioni
  - Caricamento dei dati immediato, refresh completo operato solo su richiesta dell’utente e abilitazione alla riscrittura delle interrogazioni

## Esempio di vista materializzata

```
CREATE MATERIALIZED VIEW Frn_Art_sumQ
BUILD IMMEDIATE
REFRESH COMPLETE ON DEMAND
ENABLE QUERY REWRITE
AS
SELECT Cod_F, Cod_A, SUM(Q)
FROM FAP
GROUP BY Cod_F, Cod_A;
```



## Procedura per il refresh delle viste materializzate

- L'utente, o un job di sistema, può richiedere il refresh del contenuto di una vista materializzata usando la procedura
  - DBMS\_MVIEW.REFRESH('vista', {'C'|'F'})
    - *vista*: nome vista da aggiornare
    - 'C': refresh di tipo COMPLETE
    - 'F': refresh di tipo FAST

## Procedura per il refresh delle viste materializzate

- Esempio
    - refresh in modalità COMPLETE della vista materializzata Frm\_Art\_sumQ
- ```
EXECUTE
DBMS_MVIEW.REFRESH('Frm_Art_sumQ', 'C');
```

## Fast refresh

- Richiede l'uso di apposite strutture di appoggio per il "log" delle variazioni relative alle tabelle usate nell'interrogazione associata alla vista materializzata
  - MATERIALIZED VIEW LOG
    - memorizza le variazioni che avvengono sulla tabella a cui è associato
    - ogni materialized view log è associato ad una sola tabella e ad alcuni dei suoi attributi

## Fast refresh

- L'opzione FAST REFRESH può essere usata solo se l'interrogazione associata alla vista soddisfa un insieme di vincoli
  - devono esistere le materialized view log per le tabelle e gli attributi utilizzati nell'interrogazione
  - quando si usa la group by deve sempre essere presente COUNT(\*), SUM(..), o una funzione di aggregazione nella clausola SELECT
  - ecc.

## Esempio di materialized view log

- Creare un materialized view log associato alla tabella FAP e in particolare agli attributi Cod\_F, Cod\_A, Q
  - abilitare le opzioni SEQUENCE e ROWID
  - abilitare la gestione di nuovi valori

## Esempio di materialized view log

```
CREATE MATERIALIZED VIEW LOG ON
FAP
WITH SEQUENCE, ROWID
(Cod_F, Cod_A, Q)
INCLUDING NEW VALUES;
```

## Esempio di vista materializzata con opzione fast refresh

- Voglio materializzare l'interrogazione
  - SELECT Cod\_F, Cod\_A, SUM(Q)  
FROM FAP  
GROUP BY Cod\_F, Cod\_A;
- Opzioni
  - Caricamento dei dati immediato, fast refresh eseguita automaticamente dopo ogni commit e abilitazione alla riscrittura delle interrogazioni

## Esempio di vista materializzata con opzione fast refresh

```
CREATE MATERIALIZED VIEW LOG ON FAP  
WITH SEQUENCE, ROWID  
(Cod_F, Cod_A, Q)  
INCLUDING NEW VALUES;
```

## Esempio di vista materializzata con opzione fast refresh

```
CREATE MATERIALIZED VIEW Fm_Art_sumQ  
BUILD IMMEDIATE  
REFRESH FAST ON COMMIT  
ENABLE QUERY REWRITE  
AS  
SELECT Cod_F, Cod_A, SUM(Q)  
FROM FAP  
GROUP BY Cod_f, Cod_a;
```

## Eliminazione e modifica delle viste materializzate

- Eliminazione
  - DROP MATERIALIZED VIEW *Nome*;
- Modifica
  - ALTER MATERIALIZED VIEW *Nome*  
*opzioni*;