Appunti Basi di Dati

Luca Seggiani

23 Maggio 2024

1 Data analytics

Window function

Una window function (ovvero funzione analitica) e una funzione che associa ad ogni record r un valore ottenuto da un'operazione su insieme di record logicamente connessi ad r, come ad esempio media, rank, ecc... Vediamo un'esempio di applicazione delle window function. Vogliamo scrivere una query che indichi, per ogni cardiologo, la matricola, la parcella, e la parcella media della sua specializzazione. Senza usare correlated subquery nel select, si può invece usare la clausola OVER. La clausola OVER applica una funzione di tipo aggregato o non-aggregato a un'insieme di record associati a un record di result set, detto *partition*. L'esempio sarebbe quindi:

```
SELECT M.Matricola,
  M.Parcella,
  M.Parcella
  AVG(M.Parcella) OVER()
FROM Medico M
WHERE M.Specializzazione = 'Cardiologia';
```

Si può generalizzare questa query a tutte le specializzazioni attraverso la partizione. La partizione è l'insieme di record a cui si applica la funzione aggregata / non aggregata, e dipende dal record processato attualmente. L'esempio può essere:

```
SELECT M.Matricola,
  M.Parcella,
  M.Specializzazione,
  M.Parcella
  AVG(M.Parcella) OVER( PARTITION BY
                        M.Specializzazione)
FROM Medico M
```

Dove la clausola PARTITION BY M.Specializzazione richiede alla query di effettuare l'operazione di aggregazione AVG su sottoinsiemi formati da una partizione della tabella Medico contenente solo medici con la stessa specializzazione del medico d'interesse. Il funzionamente è simile a quello di una GROUP BY, o di una semplice aggregazione, ma mantiene l'informazione riguardante i singoli medici (da cui ricaviamo matricola e parcella). Con OVER si possono usare diverse funzioni aggregate, una cui lista non viene riportata qua. Di queste abbiamo già visto le AVG, COUNT, COUNT(DISTINCT), MAX, MIN e SUM. Notiamo poi che la definizione di WINDOW è concessa fuori dalla OVER:

```
SELECT M.Matricola,
  AVG(...) OVER w
FROM Medico M
WINDOW w AS ( ... )
```

Funzioni non aggregate

Le funzioni non aggregate si possono dividere in due sottocategorie: quelle che usano l'intera partizione, e quelle che usano un suo sottoinsieme (*frame*). Le window functions non aggregate associano a ciascun record di un result set un valore ottenuto senza l'aggregazione del result set stesso, ma calcolato su tutti i record a partire dall'inizio della partizione fino al record attuale se si specifica un ORDER BY nella partizione, e tutti i record della partizione in caso contrario. Alcuni esempi sono ad esempio la FIRST_VALUE, LAST_VALUE, ecc... Possiamo fare un'esempio: poniamo di voler associare un numero ad ogni medico nella sua specializzazione. Potremo usare la funzione ROW_NUMBER, che restituisce un'indice all'interno della partizinoe per ogni record:

```
SELECT M.Matricola,
  M.Cognome,
  M.Specializzazione,
  ROW_NUMBER() OVER( PARTITION BY
                     M.Specializzazione)
FROM Medico M;
```

Classificazione

Vediamo l'uso della funzione RANK per stilare classifiche. La funzione RANK permette di classificare record (banalmente, ordinare) associando ad ognuno un certo valore SCORE (punteggio). Poniamo per esempio di voler classificare i medici in base alla loro convenienza (quindi in base a parcelle più basse):

```
SELECT M.Matricola,
  M.Cognome,
  M.Specializzazione,
  M.Parcella
```

```
  RANK() OVER( ORDER BY M.Parcella )
FROM Medico M;
```

Questo query, per ogni medico, ottiene il sottoinsieme della partizione del resultset ordinato fino a quel medico (con ORDER BY), e restituisce la posizione (con RANK) di quel medico in tale partizione. Di default, la ORDER BY effettua un'ordinamento ascendente (quindi parcelle più basse in cima, parcelle più alte in fondo). Vediamo la gestione dei pareggi. In questa forma, la funzione rank salta, per ogni parimerito, un numero uguale al numero dei pareggi, di posizioni. Ergo, se due medici ottengono lo stesso punteggio, si avrà:

Nome	Parcella	Posizione
Fabio Rossi	150	1
Nicola Tonini	250	2
Tullio Nomefalso	250	2
Michele Poraccio	300	4

Per ottenere un comportamento diverso posso usare il DENSE_RANK. Il DENSE_RANK non scala la posizione nel caso di un parimerito, e dà quindi una classifica senza "buchi":

```
SELECT M.Matricola,
  M.Cognome,
  M.Specializzazione,
  M.Parcellam
  DENSE_RANK() OVER( ORDER BY M.Parcella )
FROM Medico M;
```

Nome	Parcella	Posizione
Fabio Rossi	150	1
Nicola Tonini	250	2
Tullio Nomefalso	250	2
Michele Poraccio	300	3

Si possono effettuare classifche anche in base su più sottinsiemi (i cosiddetti **rank multipli**). Stiliamo, ad esempio, la classifica delle visite effettuate sia riguardo alla totalità dei medici, sia riguardo alla specializzazione di appartenenza del medico:

```
WITH visite AS (
  SELECT V.Medico,
    M.Cognome,
    M.Specializzazione,
    COUNT(*) AS Visite
```

```
  FROM Visita V
    INNER JOIN Medico M ON M.Matricola = V.Medico
  GROUP BY V.Medico
)
SELECT VV.Cognome,
  VV.Specializzazione,
  VV.Visite,
  RANK() OVER(ORDER BY VV.Visite DESCS) AS GlobalRank,
  RANK() OVER(PARTITION BY VV.Specializzazione
              ORDER BY VV.Visite DESC) AS SpecRank
FROM visite VV;
```

Funzioni LEAD e LAG

Le funzioni LEAD e LAG ricavano il valore dell'attributo (che può esssere anche ricavato) di una row posta k posizioni prima o dopo un dato record. Chiaramente, per applicare le LEAD e LAG dev'essere stabilito una relazione di ordinamento sensata nella partizione considerata. La funzione LAG, ad esempio, prende due argomenti: il primo rappresenta l'attributo su cui effettuare il "salto", mentre il secondo rappresenta il numero di posizioni da saltare. Poniamo ad esempio di voler considerare, per tutte le visite otorinolaringoiatriche dal 2010 al 2019, la matricola del medico, il codice fiscale del paziente, la data e la data della visita precedente effettuata da un medico con la stessa specializzazione. Avremo:

```
SELECT V.Medico,
  V.Paziente,
  V.Data,
  LAG(V.Data, 1) OVER ( PARTITION BY V.Paziente
                        ORDER BY V.Data)
FROM Visita V
  INNER JOIN Medico M ON V.Medico = M.Matricola
WHERE M.Specializzazione = "Otorinolaringoiatria"
  AND YEAR(V.Data) BEWEEN 2010 AND 2019
```

Frame

Un frame è un sottoinsieme di una partizione. Per la precisione, a partire dal record corrente in una partizione, ci si può muovere N posizioni **preceding** (precedenti) e N posizioni **following** (seguenti) da considerare. I record al di fuori delle soglie preceding e following si chiamano preceding o following **unbounded**. Una window function su un frame processa un result set e calcola valori aggregati e non sulla base di record adiacenti alla current row, ovvero su un frame della current row. Esistono alcune funzioni aggregate che lavorano solo su frame: ad esempio FIRST_VALUE e LAST_VALUE, che restituiscono rispettivamente il primo e l'ultimo valore del frame. Se non si specifica un frame, viene impostato automaticamente un default frame (che non corrisponde necessariamente alla partizione intera). Il default frame

equivale al frame che contiene tutti valori precedenti nel caso di OVER con ORDER BY, e all'intera partizione nel caso di OVER semplici. Vediamo un'esempio dell'uso di FIRST_VALUE. Poniamo di voler ottenere le prime visite cardiologiche fatte da 3 specifici pazienti nel triennio 2012-2014 con ciascun medico:

```
SELECT V.Medico,
  V.Paziente,
  V.Data,
  FIRST_VALUE(V.Data) OVER w
FROM Visita V
WHERE V.Paziente IN ('aaa_1', 'bbc_4', 'ccc_2')
  AND YEAR(V.Data) BETWEEN 2012 AND 2014
WINDOW w AS ( PARTITION BY V.Medico, V.Paziente
              ORDER BY V.Data)
```

In questo caso il default frame è corretto, in quanto vogliamo il primo record del result set.