OLAP Functions in SQL iSeries per un'Analisi Dati Avanzata: Un Focus sulle Ranking Functions

12 Febbraio 2024

Nel campo dell'analisi e gestione dei database sui sistemi IBM iSeries, le funzioni OLAP (Online analytical processing) si distinguono come strumenti potenti per l'interrogazione e la trasformazione dei dati.

Tra queste, le funzioni di ranking forniscono capacità indispensabili per eseguire analisi comparative all'interno dei dataset, consentendo alle aziende di derivare significativi spunti dai loro dati.

Questo post si addentra nelle principali funzioni di ranking disponibili in SQL su iSeries, illustrandone l'importanza e le applicazioni pratiche in scenari reali.

1. **ROW_NUMBER()**: Questa funzione è fondamentale per assegnare un intero sequenziale unico a righe all'interno di una partizione di un set di risultati, iniziando da 1. È strumentale in scenari dove è richiesta una chiara e inequivocabile ordinazione delle righe, assicurando che ogni riga possa essere distintamente identificata e accessibile.

2. **RANK()**: La funzione RANK() va un passo oltre allocando un rango a ogni riga all'interno di una partizione di un set di risultati. A differenza di ROW_NUMBER(), RANK() permette legami; quando due o più righe si qualificano per lo stesso rango, esse vengono assegnate allo stesso numero di rango, creando intervalli nella sequenza per i ranghi successivi. Questa funzione è particolarmente preziosa nelle analisi competitive, dove è cruciale comprendere il posizionamento senza artificialmente gonfiare o sgonfiare l'importanza delle entità legate.

3. **DENSE_RANK()**: Operando su una premessa simile a RANK(), DENSE_RANK() elimina gli intervalli nella sequenza di ranking, garantendo un'allocazione di rango densa e continua anche in presenza di legami. Questa funzione è essenziale quando la continuità dei valori di rango è critica per l'analisi, permettendo un sistema di ranking più compatto e intuitivo.

4. **NTILE(n)**: Per un'analisi più granulare, la funzione NTILE(n) divide una partizione ordinata in un numero specificato di gruppi, **n**, rendendola uno strumento inestimabile per segmentare i dataset in quantili o altri raggruppamenti specificati. Che si tratti di dividere i clienti in quartili di vendite o di categorizzare metriche di performance, NTILE(n) offre un approccio diretto alla stratificazione dei dati.

Queste funzioni di ranking sono integrali per sfruttare pienamente il potere analitico di SQL su iSeries, fornendo le basi per analisi dei dati sofisticate e processi decisionali.

Dominando queste funzioni, i professionisti possono sbloccare intuizioni più profonde nei loro dati, facilitando un miglioramento dell'efficienza operativa e della previsione strategica.

Questa introduzione prepara il terreno per un'esplorazione dettagliata di ogni funzione, inclusi esempi di sintassi, casi d'uso e migliori pratiche per integrare queste funzioni OLAP nelle vostre analisi dei dati e flussi di lavoro di reporting sulla piattaforma iSeries.

Nota sugli esempi: escluso il primo, gli altri esempi **utilizzano volutamente tutti la stessa tabella temporanea** in memoria creata con CTE **per dare visione dei diversi approcci e possibilità** che si possono m,ettere in campo con le Ranking Function in esame in questo articolo.

Esempio 1: OLAP function ROW_NUMBER() per il rilevamento dei numeri doppi

Introduzione alla Query

Questa query SQL è progettata per dimostrare le capacità avanzate di iSeries SQL per l'analisi dei dati, concentrandosi in particolare sull'uso delle **funzioni finestra** e delle CTE (Common Table Expressions) per elaborare e analizzare le serie di dati.

La query mostra come partizionare i dati, applicare la classificazione all'interno delle partizioni e filtrare e aggregare i risultati in base a criteri specifici.

Utilizzando un set di dati di esempio creato al volo all'interno della query, illustra un esempio pratico di identificazione e aggregazione di righe in base ai ranghi calcolati all'interno di partizioni di dati, sottolineando la potenza di SQL nell'estrazione di informazioni significative dai dati strutturati.

```
WITH
  Sample(ID, Value) AS (
      VALUES (1,10),  (2, 20), (3, 30), (4, 30)
           , (5, 30), (6, 40), (7,20), (8,50)
           , (9,60), (10,70), (11,80), (12,90)
) -- SELECT * FROM Sample;
, Sample2 (ID, Value, ROW_NUM) AS (
```

```
    SELECT ID, Value,
           ROW_NUMBER()
              OVER (PARTITION BY Value
              ORDER BY ID)
           AS row_number
      FROM Sample
) -- SELECT * FROM Sample2;
SELECT Value, MAX(ROW_NUM)
  from Sample2
 WHERE ROW_NUM > 1
group by Value
ORDER BY 1,2;
```

Risultato:

Sample		Sample2			Result	
ID	VALUE	ID	VALUE	ROW_NUM	VALUE	ROW_NUM
1	10	1	10	1	20	2
2	20	2	20	1	30	3
3	30	7	20	2	Final Result	
4	30	3	30	1		
5	30	4	30	2		
6	40	5	30	3		
7	20	6	40	1		
8	50	8	50	1		
9	60	9	60	1		
10	70	10	70	1		
11	80	11	80	1		
12	90	12	90	1		
Initial CTE		Intermediate CTE				

Descrizione dettagliata della query

La query inizia con la dichiarazione di una CTE denominata **Sample**, che costruisce una tabella temporanea in memoria composta da coppie **ID** e **valore**.

Dopo la CTE iniziale, viene definita una seconda CTE denominata **Sample2**. Questa CTE si basa sui dati di **Sample** utilizzando la **funzione finestra ROW_NUMBER()**.

La funzione viene applicata alle partizioni del set di dati create in base alla colonna **Valore**, ordinando le righe all'interno di ciascuna partizione in base all'**ID**.

Questo passaggio assegna un numero intero sequenziale univoco (**numero di riga**) a ciascuna riga all'interno della sua partizione, classificando effettivamente le righe in base al relativo **ID** nel contesto del relativo **Valore** condiviso.

Il nucleo della query risiede nella sua istruzione **SELECT** finale, che opera sui dati preparati in **Sample2**. Si rivolge specificamente alle righe in cui il numero di riga calcolato (**ROW_NUM**) è maggiore di 1, implicando un focus sulle righe all'interno di ciascuna partizione **Valore** che hanno un **valore non unico**.

La query procede quindi con l'aggregazione di queste righe mirate in base al relativo valore, selezionando il massimo ROW_**NUM** trovato all'interno di ciascun gruppo.

Questo passaggio di aggregazione evidenzia il rango più alto raggiunto dalle righe all'interno di ciascun gruppo di valori distinto che soddisfa il criterio di filtro e, in definitiva indica **quanti "doppioini" sono presenti per quel valore**.

Infine, i risultati vengono ordinati in base al Valore e al numero massimo di righe, fornendo un riepilogo chiaro e ordinato dei dati in base alla logica analitica specificata.

In sostanza, questa query non solo esemplifica la struttura sintattica e l'uso delle CTE e delle funzioni finestra in iSeries SQL, ma offre anche uno sguardo alle applicazioni pratiche di queste funzionalità per attività complesse di analisi dei dati.

Serve come un esempio di come sezionare e riassemblare i dati **per svelare modelli o intuizioni che non sono immediatamente evidenti dal set di dati grezzi**.

Esempio 2: OLAP Function ROW_NUMBER() per classificare le vendite

Introduzione alla Query

In questo esempio, approfondiamo l'applicazione pratica delle funzioni **OLAP** all'interno di un ambiente SQL iSeries, concentrandoci su uno scenario che analizza i dati di vendita in diverse regioni e prodotti.

Nello specifico, utilizzeremo la funzione **ROW_NUMBER()** per eseguire una classifica dettagliata dei dati di vendita all'interno di ciascuna regione.

Questo approccio non solo evidenzia la versatilità delle funzioni finestra in SQL per attività analitiche, ma mostra anche come possono essere sfruttate per ottenere informazioni dettagliate sulle disparità e sui successi delle prestazioni di vendita in varie località geografiche.

Suddividendo i dati per regione e ordinandoli per vendite in modo decrescente, saremo in grado di assegnare una classificazione univoca a ciascun prodotto in base alla sua performance di vendita, fornendo una panoramica chiara e utilizzabile delle dinamiche di vendita regionali.

```
WITH SalesData(Product, Region, Sales) AS (
  VALUES
    ('Product A', 'East', 100),
    ('Product B', 'East', 150),
    ('Product A', 'West', 200),
    ('Product B', 'West', 200),
    ('Product C', 'East', 50),
    ('Product C', 'West', 100),
    ('Product D', 'East', 75),
    ('Product D', 'West', 75)
)
SELECT Product, Region, Sales,
       ROW_NUMBER()
           OVER (PARTITION BY Region
                 ORDER BY Sales DESC)
       AS RowNum
FROM SalesData;
```

Risultato:

PRODUCT	REGION	SALES	ROWNUM
Product B	East	150	1
Product A	East	100	2
Product D	East	75	3
Product C	East	50	4
Product A	West	200	1
Product B	West	200	2
Product C	West	100	3
Product D	West	75	4

Descrizione dettagliata della query

La query inizia con la creazione di una Common Table Expression (CTE) denominata **SalesData**, che funge da tabella temporanea in memoria per la nostra analisi.

Questa CTE è popolata con dati di vendita di esempio, incluse le colonne per **Product**, **Region** e **Sales**, simulando così un set di dati del mondo reale che potrebbe essere riscontrato in uno scenario di analisi aziendale.

I dati includono dati di vendita per quattro diversi prodotti in due regioni (Est e Ovest), con importi di vendita variabili.

Dopo l'impostazione del CTE **SalesData**, la query procede selezionando tutte le colonne da questo set di dati e applica la funzione OLAP **ROW_NUMBER().**

Questa funzione viene utilizzata per generare una classifica intera sequenziale per ogni riga all'interno di una partizione del set di dati.

Il partizionamento viene eseguito in base alla colonna **Region**, garantendo che la classifica sia specifica per ciascuna regione e il criterio di ordinamento è la colonna **Sales** in ordine decrescente.

Ciò significa che all'interno di ciascuna regione, i prodotti verranno classificati in base ai dati di vendita, con il prodotto più venduto che riceverà un **RowNum** pari a 1 e così via.

Il risultato finale di questa query sarà un elenco di tutti i prodotti, insieme alle rispettive regioni e ai dati di vendita, ciascuno accompagnato da un numero di riga che indica la classifica di vendita all'interno della regione.

Questa classifica fornisce visibilità immediata su quali prodotti stanno ottenendo i risultati migliori in ciascuna regione, offrendo informazioni preziose per la pianificazione strategica, la gestione dell'inventario e le attività di marketing.

Utilizzando la funzione **ROW_NUMBER()** in questo modo, le aziende possono identificare con precisione i prodotti e le regioni con le migliori prestazioni, facilitando azioni mirate per migliorare le prestazioni di vendita e affrontare le aree con minori performance.

Esempio 3: OLAP function RANK() per classificare le vendite (stessa classifica per i pari merito, con lacune)

Introduzione alla Query

In questa esplorazione, approfondiremo l'utilizzo sofisticato delle funzioni OLAP all'interno di SQL su un sistema iSeries, concentrandoci in particolare sulla funzione **RANK()** per analizzare i dati di vendita.

Questa query esemplifica come suddividere in modo efficace i dati per regione e quindi classificare i prodotti all'interno di tali partizioni in base ai dati di vendita. Utilizzando queste tecniche, miriamo a scoprire informazioni dettagliate sulle prestazioni dei prodotti in diverse regioni, consentendo decisioni strategiche mirate.

Questo approccio non solo dimostra la potenza delle funzioni OLAP per l'analisi dei dati, ma ne illustra anche l'applicazione pratica nell'identificazione delle principali tendenze di vendita e dei parametri di performance in diverse aree geografiche.

Nota: il risultato nella classifica ottenuto con questo approccio è simile a quello precedente utilizzando ROW_NUMBER(), ma fornisce lo **stesso rango per i pari merito, con lacune (Gaps):** ciò classificherà i record di vendita all'interno di ciascuna regione, consentendo classifiche identiche in caso di parità di importi di vendita, con lacune per le classifiche successive (aspetto immediatamente visibile se si confrontano i risultati delle due versioni: con **ROW_NUMBER()** e con **RANK()**.

```
WITH SalesData(Product, Region, Sales) AS (
  VALUES
    ('Product A', 'East', 100),
    ('Product B', 'East', 150),
    ('Product A', 'West', 200),
    ('Product B', 'West', 200),
    ('Product C', 'East', 50),
    ('Product C', 'West', 100),
    ('Product D', 'East', 75),
    ('Product D', 'West', 75)
) -- select * from SalesData;
 SELECT Product, Region, Sales,
        RANK()
          OVER (PARTITION BY Region
                ORDER BY Sales DESC)
        AS Rank
FROM  SalesData;
```

Risultato:

PRODUCT	REGION	SALES	RANK
Product B	East	150	1
Product A	East	100	2
Product D	East	75	3
Product C	East	50	4
Product A	West	200	1
Product B	West	200	1
Product C	West	100	3
Product D	West	75	4

Descrizione dettagliata della query

Questa query SQL inizia con la costruzione e utilizzo di una Common Table Expression (CTE) denominata **SalesData**.

Questa CTE è progettata per simulare un set di dati del mondo reale creando una tabella in memoria popolata con record di vendita per vari prodotti in diverse regioni.

Ogni record è costituito dal nome del prodotto, dalla regione in cui è stata effettuata la vendita e dall'importo totale delle vendite per quel prodotto nella rispettiva regione.

Dopo aver creato la tabella **SalesData**, la query sfrutta la funzione **RANK()**, una delle funzionalità OLAP di SQL, per eseguire un'analisi avanzata dei dati di vendita.

La funzione **RANK()** viene applicata all'interno di una finestra definita dalla clausola **OVER**, che suddivide i dati per **Region**.

All'interno di ciascuna partizione, la funzione ordina i prodotti in base alle vendite in ordine decrescente.

La funzione **RANK()** assegna quindi una classificazione a ciascun prodotto in base a questo ordine, con il prodotto più venduto in ciascuna regione che riceve un rango pari a 1.

Nei casi in cui più prodotti hanno le stesse cifre di vendita, ricevono lo stesso rango e il rango successivo viene incrementato di conseguenza.

Questo processo selettivo dà come risultato un set di dati che non solo include le informazioni originali sul prodotto, sulla regione e sulle vendite, ma lo arricchisce anche con una classifica dei prodotti all'interno di ciascuna regione in base alle prestazioni di vendita.

Questa classifica fornisce informazioni critiche su quali prodotti sono leader nelle vendite in ciascuna regione e su quanto i prodotti competitivi si comportano l'uno rispetto all'altro.

Eseguendo questa query, le aziende e gli analisti possono acquisire una comprensione più chiara delle dinamiche di mercato e delle prestazioni dei prodotti nelle diverse aree geografiche.

Le informazioni derivate da questa analisi possono informare una serie di decisioni strategiche, dallo sviluppo del prodotto e dalle strategie di marketing alla gestione dell'inventario e all'attenzione alle vendite regionali, consentendo in definitiva un approccio più basato sui dati per l'ottimizzazione e la crescita del business.

Esempio 4: OLAP function DENSE_RANK() per classificare le vendite (senza spazi vuoti)

Introduzione alla Query

In questo esempio, approfondiremo l'applicazione avanzata delle funzioni OLAP all'interno di SQL su un sistema iSeries, concentrandoci sulla funzione **DENSE_RANK()** per un'analisi dei dati di vendita in varie regioni.

Questa query è progettata per dimostrare come i dati possono essere segmentati e analizzati all'interno di partizioni specifiche, in questo caso per regione, per assegnare classifiche ai prodotti in base alle prestazioni di vendita.

A differenza della funzione **RANK(), DENSE_RANK()** fornisce una prospettiva unica garantendo una sequenza continua di classifiche, anche in presenza di parità, offrendo così una visione più granulare delle posizioni competitive all'interno di ciascuna regione.

Questo approccio è particolarmente utile per identificare quanto i prodotti competono all'interno del mercato, facilitando i processi decisionali strategici senza l'inclinazione dei divari di rango.

```
WITH SalesData(Product, Region, Sales) AS (
  VALUES
    ('Product A', 'East', 100),
    ('Product B', 'East', 150),
    ('Product A', 'West', 200),
    ('Product B', 'West', 200),
    ('Product C', 'East', 50),
    ('Product C', 'West', 100),
    ('Product D', 'East', 75),
    ('Product D', 'West', 75)
) -- select * from SalesData;
SELECT Product, Region, Sales,
       DENSE_RANK()
           OVER (PARTITION BY Region
                 ORDER BY Sales DESC)
       AS DenseRank
FROM SalesData;
```

Risultato:

PRODUCT	REGION	SALES	DENSERANK
Product B	East	150	1
Product A	East	100	2
Product D	East	75	3
Product C	East	50	4
Product A	West	200	1
Product B	West	200	1
Product C	West	100	2
Product D	West	75	3

Descrizione dettagliata della query

La query inizia con la creazione di una Common Table Expression (CTE), **SalesData**, che funge da origine dati temporanea per l'analisi.

Questa CTE è popolata con ipotetiche informazioni sulle vendite, dettagliando i nomi dei prodotti, le regioni corrispondenti e le cifre di vendita.

Questo set di dati fornisce un modello semplificato ma rappresentativo dei dati sulle prestazioni di vendita che le aziende potrebbero analizzare per comprendere le prestazioni dei prodotti nei diversi mercati.

Dopo l'impostazione CTE, la query utilizza la funzione **DENSE_RANK()** per analizzare i dati di vendita.

Questa funzione viene applicata all'interno di una finestra specifica, definita dalla clausola **OVER**, che organizza i dati in partizioni per **Region**.

All'interno di ciascuna partizione, i prodotti vengono ordinati in base alle Vendite in ordine decrescente.

La funzione **DENSE_RANK()** assegna quindi una classificazione a ciascun prodotto in base alle cifre di vendita, garantendo che i prodotti con cifre di vendita identiche ricevano la stessa classificazione.

Fondamentalmente, **DENSE_RANK()** non lascia spazi vuoti nella sequenza di classificazione per ranghi pari, il che significa che la sequenza di classificazione è continua e densa, da cui il nome.
Ad esempio, se due prodotti nella regione Est hanno entrambi i dati di vendita più alti, riceveranno entrambi un rango denso pari a 1, e il prodotto successivo più alto riceverà un rango denso pari a 2, indipendentemente dal numero di prodotti che condividono il rango più alto.

Questo metodo di classificazione è particolarmente utile per analizzare da vicino la posizione competitiva, in quanto consente alle parti interessate di vedere con precisione quanti prodotti ottengono risultati simili in termini di vendite, senza la distorsione dei divari di classifica che potrebbe implicare una disparità di prestazioni maggiore di quella effettivamente esistente.

L'output di questa query fornisce un elenco chiaro e ordinato di prodotti all'interno di ciascuna regione, classificati in base alle prestazioni di vendita con una classifica densa.

Questa intuizione è preziosa per le aziende che desiderano comprendere la posizione dei propri prodotti in varie regioni, identificare i principali concorrenti e elaborare strategie su come migliorare o sfruttare le proprie posizioni di mercato.

Sfruttando la funzione **DENSE_RANK()** in questo modo, le organizzazioni possono affinare le proprie analisi di mercato e gli sforzi di pianificazione strategica, assicurandosi che siano basati su una comprensione sfumata del proprio ambiente competitivo.

Esempio 5: OLAP Function NTILE(n) per dividere i record di vendita in gruppi

Introduzione alla Query

In questa esplorazione, approfondiremo l'applicazione avanzata delle funzioni OLAP all'interno di SQL su un sistema iSeries, concentrandoci sull'utilizzo della funzione **NTILE()** per eseguire una segmentazione "nuanced" (vale a dire, finemente sintonizzata e mirata con precisione) dei dati di vendita.

Questo approccio analitico è orientato a dividere i record di vendita in gruppi distinti all'interno di ciascuna regione, in base alla loro performance di vendita.

Implementando la funzione **NTILE(2)**, miriamo a classificare i dati in due gruppi, creando essenzialmente quartili (in questo contesto semplificato) che ci consentono di differenziare tra prestazioni di vendita superiori e inferiori all'interno di ciascun segmento geografico.

Questo metodo di analisi dei dati è particolarmente utile per le organizzazioni che desiderano ottenere informazioni più approfondite sulla distribuzione delle vendite e identificare potenziali aree di miglioramento strategico o investimento.

```
WITH SalesData(Product, Region, Sales) AS (
  VALUES
    ('Product A', 'East', 100),
    ('Product B', 'East', 150),
    ('Product A', 'West', 200),
    ('Product B', 'West', 200),
    ('Product C', 'East', 50),
    ('Product C', 'West', 100),
```

```
    ('Product D', 'East', 75),
    ('Product D', 'West', 75)
) -- select * from SalesData;
SELECT Product, Region, Sales,
       NTILE(2)
           OVER (PARTITION BY Region
                 ORDER BY Sales DESC)
       AS SalesQuartile
FROM SalesData;
```

Risultato:

PRODUCT	REGION	SALES	SALESQUARTILE
Product B	East	150	1
Product A	East	100	1
Product D	East	75	2
Product C	East	50	2
Product A	West	200	1
Product B	West	200	1
Product C	West	100	2
Product D	West	75	2

Descrizione dettagliata della query

La query in questione inizia con la costruzione di una Common Table Expression (CTE), **SalesData**, progettata per simulare un tipico set di dati di vendita.

Questa CTE comprende i record delle vendite, incluso il nome del prodotto, la regione di vendita e l'importo totale delle vendite per quel prodotto all'interno della regione specificata.

Questa configurazione fornisce un set di dati fondamentale che riflette gli scenari comuni affrontati dalle aziende nell'analisi delle prestazioni di vendita nei diversi mercati.

Dopo la creazione del **SalesData** CTE, la query procede applicando la funzione **NTILE(2)** all'interno di una finestra specificata, delineata dalla clausola **OVER**.

Questa finestra organizza i dati in partizioni per **Region** e quindi ordina i record all'interno di ciascuna partizione in base alle vendite in ordine decrescente.

La funzione **NTILE(2)** divide quindi questi record ordinati in due gruppi, o "quartili", all'interno di ciascuna regione.

La divisione mira a distribuire equamente i record in gruppi in base ai relativi dati di vendita, facilitando così una chiara distinzione tra prodotti con prestazioni più elevate e prodotti con prestazioni inferiori all'interno di ciascuna regione.

Il risultato di questa query è una visualizzazione perfezionata dei dati di vendita, in cui ogni record non è solo associato ai suoi attributi originali (Prodotto, Regione, Vendite) ma anche annotato con un valore **SalesQuartile**.

Questo valore indica se il prodotto rientra nella metà superiore (**quartile 1**) o nella metà inferiore (**quartile 2**) dell'andamento delle vendite all'interno della sua regione, in base alla categorizzazione **NTILE**.

Ovviamente la suddivsione in **NTILE** può variare, a seconda dei valori in campo e delle politiche seguite dall'azienda e, ad esempio, mantenendo l'esempio di campionatura dei dati, se si esegue la stessa query con **NTILE(3)** si ottengono tre categorie:

PRODUCT	REGION	SALES	SALESQUARTILE
Product B	East	150	1
Product A	East	100	1
Product D	East	75	2
Product C	East	50	3
Product A	West	200	1
Product B	West	200	1
Product C	West	100	2
Product D	West	75	3

Questa segmentazione fornisce approfondimenti critici sul panorama delle vendite di ciascuna regione, evidenziando quali prodotti sono leader del mercato e quali potrebbero richiedere ulteriore supporto o riallineamento strategico.

Sfruttando la funzione **NTILE()** in questo modo, le aziende possono sfruttare una comprensione più granulare della distribuzione delle vendite nelle diverse regioni.

Questa conoscenza è preziosa per la pianificazione strategica, poiché consente alle aziende di personalizzare le proprie strategie di marketing, produzione e inventario per allinearle meglio alle tendenze delle prestazioni di vendita regionali.

Inoltre, questo approccio analitico sottolinea la versatilità e la potenza delle funzioni OLAP in SQL per iSeries per condurre sofisticate analisi dei dati, in particolare in scenari che richiedono una classificazione dettagliata e la segmentazione dei dati, come la valutazione delle prestazioni di vendita su più dimensioni.

Conclusioni

In conclusione, l'esplorazione di iSeries SQL e delle **ranking functions OLAP** – **ROW_NUMBER(), RANK(), DENSE_RANK() e NTILE(n)** – rivela la profonda capacità di questi strumenti nel migliorare l'analisi dei dati e i processi decisionali all'interno di un organizzazione.

Utilizzando in modo efficace queste funzioni, le aziende possono ottenere una comprensione più articolata dei propri dati, consentendo loro di identificare tendenze, segmentare i dati in categorie significative e classificare gli elementi all'interno di contesti specifici.

Che si tratti di assegnare un rango univoco a ciascuna riga con **ROW_NUMBER()**, di comprendere la posizione relativa con **RANK()** e **DENSE_RANK()** o di classificare i dati in quantili con **NTILE(n)**, queste funzioni offrono un set versatile di strumenti per gestire set di dati complessi.

Questa abilità tecnica non solo semplifica il processo analitico, ma fornisce anche informazioni utili che sono fondamentali per la pianificazione strategica e l'analisi competitiva.

Mentre continuiamo ad approfondire le profondità di SQL sui sistemi iSeries, diventa evidente che padroneggiare queste funzioni OLAP è essenziale per qualsiasi professionista che desideri sfruttare i dati per un processo decisionale informato, evidenziando il valore indispensabile di SQL nel moderno panorama basato sui dati.

Tenere presente, come sempre, che l'implementazione e le funzionalità effettive potrebbero variare a seconda della versione del sistema operativo iSeries e del database DB2.

È sempre buona norma fare riferimento alla documentazione ufficiale IBM per avere informazioni più precise e dettagliate.

Riferimenti utili

- Data Science: opportunità e sfide da affrontare
- Funzioni OLAP in Data Science: un approccio moderno alla Business Intelligence
- Sfruttare il Potere della Data Science e SQL con OLAP Functions su IBM iSeries
- OLAP Functions in SQL iSeries per un'Analisi Dati Avanzata: Un Focus sulle Windows Functions

#SQL #Data Science #Business Intelligence #OLAP Online Analytical Processing #Ranking Functions

← PRECEDENTE
OLAP Functions in SQL iSeries per un'Analisi Dati Avanzata: Un Focus sulle Windows Functions

SEGUENTE →
OLAP Functions in SQL iSeries per un'Analisi Dati Avanzata: Un Focus sulle Grouping Operation

Articoli simili

Data Science

AI Artificial Intelligence

SkillUP

As400 Architettura

Gestire AS400 con SQL

Gestione approfondita Risorse Sistema

Spiccioli di tecnica

Windows

CERCA TESTO IN POST / TAGS

Cerca ...

SkillUp Via Marco Polo, 139 55049 Viareggio (LU)
Via Lucchese, 84 50019 Sesto Fiorentino (FI)
Tel: 055319212 - info@SkillUp.cloud

LinkedIn

Marco Del Corto
Founder/CEO Genesi srl
Visualizza profilo