Sfruttare il Potere della Data Science e SQL con OLAP Functions su IBM iSeries

7 Febbraio 2024

Nel panorama tecnologico in continua evoluzione, la Data Science si è affermata come un pilastro dell'intelligenza aziendale, della presa di decisioni strategiche e dell'efficienza operativa.

Al cuore della Data Science si trova la potente combinazione di SQL e funzioni Online Analytical Processing (OLAP), specialmente all'interno dell'ambiente robusto di IBM iSeries (AS/400).

Questo post esplora come l'utilizzo di SQL e funzioni OLAP sulla piattaforma IBM iSeries possa trasformare l'analisi dei dati, offrendo approfondimenti dettagliati e facilitando complesse query analitiche direttamente sul database.

Sebbene queste possibilità siano comuni agli engine sql che forniscono funzioni OLAP, in questo post l'attenzione è focalizzata sulla piattaforma IBM iSeries.

Introduzione alla Data Science e SQL

La Data Science combina varie competenze statistiche, analitiche e di programmazione per raccogliere, analizzare e interpretare grandi set di dati, al fine di ottenere informazioni azionabili.

SQL (Structured Query Language) è uno strumento fondamentale nell'arsenale dello scienziato dei dati, consentendo la gestione e la manipolazione di database relazionali. La precisione, l'efficienza e la versatilità di SQL lo rendono indispensabile per interrogare e analizzare i dati.

Il Ruolo delle Funzioni OLAP nella Data Science

Le funzioni OLAP estendono le capacità di SQL, abilitando avanzate funzionalità di manipolazione e analisi dei dati.

Queste funzioni sono fondamentali per eseguire complesse query analitiche, come la sommarizzazione dei dati, l'aggregazione e l'analisi multidimensionale.

Integrando le funzioni OLAP all'interno delle query SQL, gli scienziati dei dati possono potenziare le loro capacità di esplorazione e analisi dei dati, scoprendo approfondimenti più dettagliati in modo più efficiente.

Funzioni OLAP su IBM iSeries (DB2 for i)

La piattaforma IBM iSeries (AS/400), con il suo database DB2 for i, supporta un'ampia suite di funzioni OLAP progettate per facilitare sofisticate analisi dei dati.

Queste funzioni possono essere categorizzate come segue:

1 – Funzioni Aggregate: Essenziali per eseguire aggregazioni da basiche ad avanzate, queste funzioni aiutano a riassumere i dati, fornendo una base per ulteriori analisi:

- **SUM, AVG, MAX, MIN, COUNT**: Eseguono aggregazioni di base come totale, media, massimo, minimo e conteggio.
- **STDDEV, VAR**: Calcolano la deviazione standard e la varianza per comprendere la dispersione dei dati.

2 – Funzioni di Ranking: offrono avanzate capacità di classificazione, essenziali per l'analisi comparativa all'interno dei set di dati:

- **ROW_NUMBER()**: Assegna un intero sequenziale unico alle righe all'interno di una partizione di un set di risultati, a partire da 1.
- **RANK()**: Assegna un rango a ogni riga all'interno di una partizione di un set di risultati, con gap nei valori di rango in caso di parità.
- **DENSE_RANK()**: Simile a **RANK()**, ma senza gap nella sequenza di classificazione per ranghi pari.
- **NTILE(n)**: Distribuisce le righe in una partizione ordinata in un numero specificato di gruppi, n. È utile per dividere un dataset in quantili.

3 – Funzioni di Finestra: Consentono l'accesso a punti dati specifici all'interno di un set di dati, facilitando complesse comparazioni e analisi senza la necessità di self-join:

- **LEAD, LAG**: Accedono ai dati da una riga successiva (LEAD) o precedente (LAG) senza richiedere un self-join.
- **FIRST_VALUE, LAST_VALUE**: Ottengono il primo o l'ultimo valore in una finestra specificata.
- **NTH_VALUE()**: Recupera il valore di una colonna specificata alla posizione n all'interno della finestra.

4 – Funzioni Aggregate Analitiche: Consentono l'esecuzione di aggregazioni su insiemi di righe correlati, offrendo una prospettiva analitica più profonda:

- **SUM() OVER, AVG() OVER**: Eseguono aggregazioni su un insieme di righe in qualche modo correlate alla riga corrente.
- **CUME_DIST()**: Calcola la distribuzione cumulativa di un valore all'interno di un insieme di valori.
- **PERCENT_RANK()**: Calcola il rango percentuale di un valore in un insieme di valori.

5 – Operazioni di Raggruppamento: Consentono un'aggregazione e un'analisi dei dati complete, supportando complesse esigenze di reporting con efficienza e flessibilità:

- **GROUPING SETS**: Consente livelli multipli di aggregazioni in una singola query, utile per generare report con subtotali e totali generali.
- **ROLLUP**: Produce un set di risultati che mostra aggregati per una gerarchia di valori, inclusi subtotali e un totale generale.
- **CUBE**: Genera tutte le possibili combinazioni di aggregazioni per un gruppo di colonne selezionate.

6 – Funzioni Varie: Migliorano le capacità di manipolazione dei dati, consentendo presentazioni e approfondimenti innovativi sui dati:

- **LISTAGG()**: Aggrega dati stringa concatenando valori da più righe.
- **GROUPING()**: Identifica se una riga in un set di risultati aggregato è una riga di super-aggregazione.

Considerazioni sull'Uso

- **Prestazioni**: Le funzioni OLAP, specialmente quando applicate a grandi set di dati, possono essere intensive in termini di risorse. L'indicizzazione appropriata, il partizionamento e l'ottimizzazione delle query sono essenziali.
- **Compatibilità**: Assicurati che il tuo sistema sia su una versione recente di IBM i (OS/400) per utilizzare la gamma completa di capacità OLAP offerte da DB2 for i.
- **Curva di Apprendimento**: Seppur potenti, le funzioni OLAP possono essere complesse da padroneggiare. Investi tempo nella comprensione delle loro sfumature per sfruttare appieno le loro capacità analitiche.

L'integrazione di SQL e funzioni OLAP sulla piattaforma IBM iSeries offre opportunità senza precedenti per gli scienziati dei dati di eseguire sofisticate analisi e reportistica dei dati.

Padroneggiando questi strumenti, i professionisti possono sbloccare approfondimenti più dettagliati, migliorare le prestazioni e razionalizzare i flussi di lavoro di elaborazione dei dati, guidando in avanti la frontiera della Data Science all'interno delle loro organizzazioni.

Bene, cominciamo a vedere all'opera alcune OLAP Functions!

Esempi di funzioni OLAP: funzioni aggregate SUM, AVG, MAX, MIN, COUNT, STDDEV, VAR

Di seguito sono riportati esempi di comandi SQL per IBM iSeries che dimostrano l'uso delle funzioni OLAP (Online Analytical Processing), incluse funzioni aggregate come:

- **SUM, AVG, MAX, MIN, COUNT**: Eseguono aggregazioni di base come totale, media, massimo, minimo e conteggio.
- **STDDEV, VAR**: Calcolano la deviazione standard e la varianza per comprendere la dispersione dei dati.

Questi esempi incorporano anche le Common Table Expressions (CTE) e la clausola **VALUES** per preparare i dati di test.

```
WITH sales_data (sale_id, category, product_id, quantity_sold, sale_amount)
  VALUES
    (1,  'C1', 'Pa', 10, 1000),
    (2,  'C1', 'Pb', 20, 1500),
    (3,  'C1', 'Pc', 15, 1200),
    (4,  'C1', 'Pd', 25, 2000),
    (5,  'C2', 'Pe', 30, 2600),
    (6,  'C2', 'Pf', 10, 1200),
    (7,  'C2', 'Pg', 20, 1700),
    (8,  'C2', 'Ph', 15, 1400),
    (9,  'C3', 'Pi', 10, 2000),
    (10, 'C3', 'Pj', 20, 1900),
    (11, 'C3', 'Pk', 15, 1500),
    (12, 'C3', 'Pl', 25, 2300),
    (13, 'C3', 'Pm', 30, 2800),
    (14, 'C3', 'Pn', 30, 2200)
)
SELECT
  category,
  SUM(sale_amount) AS total,
  AVG(sale_amount) AS average,
```

```
    MAX(sale_amount) AS "max",
    MIN(sale_amount) AS "min",
    Round(STDDEV(sale_amount)) AS stddev,
    Round(VAR(sale_amount)) AS var,  COUNT(*) AS "Count"
  FROM sales_data
  GROUP BY category
  ORDER BY 1;
```

Risultato:

CATEGORY	TOTAL	AVERAGE	max	min	STDI
C1	5.700	1.425	2.000	1.000	377
C2	6.900	1.725	2.600	1.200	536
C3	12.700	2.116	2.800	1.500	398

Due Parole sulla Deviazione Standard e Varianza.

Le OLAP (Online Analytical Processing) functions per il calcolo della deviazione standard e della varianza sono strumenti potenti nel contesto dell'analisi dei dati, specialmente quando si lavora con grandi set di dati in database o data warehouse per supportare decisioni aziendali.

Queste funzioni consentono agli analisti di comprendere meglio la distribuzione e la variabilità dei dati, elementi fondamentali per l'analisi statistica e la presa di decisioni basata sui dati.

Concetto Statistico

- **Varianza**: La varianza è una misura della dispersione che indica quanto i valori di un insieme di dati si discostano dalla media (valore medio) dell'insieme. In termini semplici, ci dice quanto siano variabili i dati. La varianza si calcola come la media dei quadrati delle differenze tra ciascun valore e la media dell'insieme di dati. Nel contesto OLAP, la varianza aiuta a comprendere la distribuzione dei valori attraverso diverse dimensioni, come il tempo, la geografia, o categorie di prodotto.
- **Deviazione Standard**: La deviazione standard è la radice quadrata della varianza e fornisce una misura della dispersione dei dati in unità originali. Mentre la varianza dà un'idea della variabilità dei dati, la deviazione standard è più intuitivamente comprensibile perché è espressa nella stessa unità di misura dei dati originali. Questo rende più facile valutare la dispersione dei valori attorno alla media.

Significato Pratico

- **Analisi della Variabilità**: Comprendere la varianza e la deviazione standard dei dati può aiutare le aziende a identificare la variabilità nei processi di business, nelle vendite, nella produzione e in altre metriche chiave. Ad esempio, un'alta variabilità nelle vendite potrebbe indicare la necessità di esaminare più da vicino i fattori che influenzano tali variazioni, come stagionalità, promozioni o concorrenza.

- **Supporto alle Decisioni**: Le funzioni OLAP che calcolano queste statistiche possono supportare decisioni strategiche fornendo insight su come i dati si distribuiscono attorno a medie settoriali o aziendali. Per esempio, capire la deviazione standard nel tempo di consegna può aiutare a migliorare la logistica e la soddisfazione del cliente.
- **Segmentazione e Targeting**: Analizzando la varianza e la deviazione standard di comportamenti o preferenze dei clienti, le aziende possono identificare segmenti di mercato o gruppi di clienti con esigenze o comportamenti simili, permettendo strategie di marketing più mirate.
- **Rilevamento di Anomalie**: Valori che si discostano significativamente dalla media (outliers) possono essere facilmente identificati quando si analizza la deviazione standard. Questo può essere particolarmente utile per rilevare errori nei dati, frodi, o altre anomalie operative.

In sintesi, le funzioni OLAP per il calcolo della varianza e della deviazione standard sono essenziali per qualsiasi analisi avanzata dei dati, permettendo agli analisti e ai decisori di capire meglio la natura e la distribuzione dei dati all'interno dell'organizzazione. Questa comprensione può guidare l'ottimizzazione dei processi, l'innovazione dei prodotti, e strategie di marketing più efficaci, tra gli altri vantaggi competitivi.

Esempi di funzioni OLAP: Funzioni Analitiche Aggregate SUM() OVER, AVG() OVER, CUME_DIST(), PERCENT_RANK()

Di seguito sono riportati esempi di comandi SQL per IBM iSeries (noto anche come AS/400 o IBM i) che dimostrano l'uso delle funzioni OLAP (Online Analytical Processing), incluse le funzioni di aggregazione analitica, conosciute anche come **funzioni Finestra o funzioni di Partizionamento**, come **SUM() OVER**, **AVG() OVER**, **CUME_DIST()** e **PERCENT_RANK()**.

Questi esempi incorporano anche Espressioni di Tabelle Comuni (CTE) e la clausola **VALUES** per preparare i dati di test.

Esempio 1 OLAP Function: SUM() OVER e AVG() OVER

Queste funzioni calcolano la somma e la media di un insieme di righe definito dalla clausola OVER (spiegata in dettaglio sotto).

```
WITH SalesData (PersonId, Year, Amount) AS (
  VALUES (1, 2019, 1000), (1, 2020, 1500), (1, 2021, 800)
       , (1, 2022, 1400), (1, 2023, 1600),
         (2, 2020, 1500), (2, 2021, 800), (2, 2022, 1400)
       , (2, 2023, 1600)
       , (3, 2021, 800), (3, 2022, 1400), (3, 2023, 1600)
)
SELECT distinct
  PersonId
, SUM(Amount) OVER (PARTITION BY PersonId) AS Total
, AVG(Amount) OVER (PARTITION BY PersonId) AS Avg
, SUM(Amount) OVER (PARTITION BY 'ALL') AS total
, DECIMAL(100 * (SUM(Amount) OVER (PARTITION BY PersonId)) /
    DECIMAL(SUM(Amount) OVER (PARTITION BY 'ALL'), 10, 3), 5, 3) AS Perc
, COUNT(Year) OVER (PARTITION BY PersonId) AS CountYear
FROM SalesData
ORDER BY 1;
```

Risultato:

PERSONID	TOTAL	AVG	TOTAL	PERC	COUNT
1	6.300	1.260	15.400	40,909	5
2	5.300	1.325	15.400	34,415	4
3	3.800	1.266	15.400	24,675	3

In questo esempio, la CTE **SalesData** prepara dati di test con **PersonId, Year** e **Amount**. La clausola **SELECT** calcola quindi le vendite totali, la media e la Perc delle vendite per ogni venditore usando le funzioni **SUM() OVER** e **AVG() OVER**, partizionando i dati per **PersonId**.

Due parole sulle Windows Functions o Partitioning Functions

In IBM iSeries SQL e più in generale nei database SQL, le funzioni OLAP come **SUM(...) OVER (...), AVG(...) OVER (...)** e **COUNT(...) OVER (...)** fanno parte delle cosiddette **funzioni Finestra** o **funzioni di Partizionamento**.

Queste funzioni eseguono calcoli su una serie di righe che sono in qualche modo correlate alla riga corrente, consentendo analisi complesse come totali parziali, medie mobili o statistiche cumulative.

Ecco come funziona ciascuna di queste funzioni nel contesto della nostra query:

SUM(...) OVER (PARTITION BY...)

La funzione **SUM(...) OVER (PARTITION BY ...)** calcola la somma totale di una colonna per ciascuna partizione dei dati. Una **partizione** è essenzialmente un sottoinsieme di dati in base alle colonne specificate nella clausola **PARTITION BY**. Nella nostra query, **SUM(Amount) OVER (PARTITION BY PersonId)** calcola l'**importo** totale per ogni **PersonId** nelle rispettive righe. Ciò significa che raggruppa i dati per **PersonId** e quindi somma l'**importo** per ciascun gruppo.

AVG(...) OVER (PARTITION BY ...)

Similmente alla funzione **SUM, AVG(...) OVER (PARTITION BY ...)** calcola la media di una colonna per ciascuna partizione. Nel nostro esempio, **AVG(Amount) OVER (PARTITION BY PersonId)** calcola l'importo medio per ogni **PersonId**. Questa operazione viene eseguita partizionando i dati in base a **PersonId** e calcolando l'**importo** medio per ciascuna partizione.

COUNT(...) OVER (PARTITION BY...)

La funzione **COUNT(...) OVER (PARTITION BY ...)** conta il numero di righe in ciascuna partizione. Nella query fornita, **COUNT(Year) OVER (PARTITION BY PersonId)** conta il numero di anni registrati per ogni **PersonId**. Questo ti dà effettivamente il numero di voci (o record) per persona.

Note speciali sulla nostra query: **SUM(Amount) OVER (PARTITION BY 'ALL')** è un utilizzo interessante che calcola l'importo totale su tutti i dati **senza partizionamento**. Il valore 'ALL' è una costante che ha lo stesso valore per ogni riga, quindi non è stata eseguita alcuna partizione (al posto di 'ALL' possiamo utilizzare **qualsiasi valore constante**).

Nel complesso, le funzioni finestra come **SUM OVER, AVG OVER e COUNT OVER** forniscono strumenti potenti per eseguire analisi dei dati dettagliate e sofisticate all'interno di query SQL, consentendo agli analisti di dati e agli sviluppatori di ricavare informazioni significative da set di dati complessi.

Esempio 2 OLAP Function: CUME_DIST() OVER

Questa funzione calcola la distribuzione cumulativa (spiegata in dettaglio sotto) di un valore all'interno di un insieme di valori.

```
WITH ExamScores (StudentId, Score) AS (
  VALUES
    (1, 75),
    (2, 88),
    (3, 92),
    (4, 67),
    (5, 81)
)
SELECT
  StudentId,
  Score,
```

```
  CUME_DIST() OVER (ORDER BY Score) AS CumulativeDistribution
FROM ExamScores
ORDER BY Score;
```

Risultato:

STUDENTID	SCORE	CUMULATIVEDISTRIBUTION
4	67	0,2
1	75	0,4
5	81	0,6
2	88	0,8
3	92	1

Qui, la CTE **ExamScores** prepara dati di test con **StudentId** e **Score**. La clausola **SELECT** calcola la distribuzione cumulativa dei punteggi, ordinando per il valore del punteggio.

Due parole sulla Distribuzione Cumulativa

La funzione **CUME_DIST()** in SQL calcola la distribuzione cumulativa di un valore all'interno di una sequenza di valori. È una funzione OLAP (Online Analytical Processing) che fornisce un modo per calcolare la posizione relativa di un valore specifico all'interno di un gruppo di valori, ordinati per una certa colonna.

Il risultato è un valore compreso tra 0 e 1, rappresentante la proporzione di righe che hanno valori minori o uguali al valore nella riga corrente.

Ecco cosa succede passo dopo passo:

1. **Preparazione dei Dati**: L'Espressione di Tabella Comune (CTE) **ExamScores** prepara un semplice set di dati di **StudentId** e **Score**.
2. **Ordinamento dei Punteggi**: La funzione **CUME_DIST()** viene applicata sui punteggi ordinati per la colonna **Score**. Ciò significa che per ogni riga, SQL calcola la distribuzione cumulativa del punteggio corrente rispetto a tutti i punteggi nel dataset.
3. **Calcolo della Distribuzione Cumulativa**: Per ogni punteggio, **CUME_DIST()** calcola la proporzione di tutti i punteggi nel dataset che sono minori o uguali al punteggio corrente. Questo viene fatto dividendo il numero di valori di punteggi che sono minori o uguali al punteggio corrente per il numero totale di punteggi nel dataset.

Per esempio, se guardiamo un punteggio di 75:

- Ci sono 5 punteggi in totale.
- Il punteggio di 75 è maggiore o uguale a 2 punteggi (incluso sé stesso, cioè 67 e

75) e minore di 3 altri punteggi (81, 88, 92).
- La distribuzione cumulativa per il punteggio di 75 viene calcolata come il numero di punteggi fino a e inclusi 75 (che in questo caso è 2) diviso per il numero totale di punteggi (che è 5), risultando in un valore di distribuzione cumulativa di 0,4.

Questo calcolo viene ripetuto per ogni punteggio nel dataset, fornendo un modo per comprendere non solo il valore assoluto dei punteggi, ma come ogni punteggio si confronta con la distribuzione di tutti i punteggi.

Il risultato di questa query fornisce una tabella con **StudentId**, il loro **Score** e la **CumulativeDistribution** di ogni punteggio, mostrando la proporzione di studenti che hanno ottenuto un punteggio fino a e inclusi quello punteggio.

Questo può essere particolarmente utile nell'analisi educativa, nei sistemi di valutazione, o in qualsiasi scenario dove è importante comprendere la distribuzione dei valori all'interno di un dataset.

Esempio 3 OLAP Function: PERCENT_RANK() OVER

Questa funzione calcola il rango percentuale di ogni riga all'interno di una partizione di un set di risultati.

```
WITH EmployeeSales (EmployeeId, SalesAmount) AS (
  VALUES
    (1, 5000),
    (2, 7600),
    (3, 9400),
    (4, 3000),
    (5, 6200)
)
SELECT
  EmployeeId,
  SalesAmount,
  PERCENT_RANK() OVER (ORDER BY SalesAmount) AS PercentileRank
FROM EmployeeSales
ORDER BY SalesAmount;
```

Risultato:

EMPLOYEEID	SALESAMOUNT	PERCENTILERANK
4	3.000	0
1	5.000	0,25
5	6.200	0,5
2	7.600	0,75
3	9.400	1

In questo esempio, la CTE **EmployeeSales** prepara dati di test con **EmployeeId** e **SalesAmount**. La clausola **SELECT** calcola il rango percentuale di ogni dipendente in base al loro importo di vendite.

Due Parole sulla Funzione PERCENT_RANK()

La funzione **PERCENT_RANK()** in SQL calcola il rango relativo di una riga all'interno di una partizione di un insieme di risultati, escludendo il rango più alto. Il valore restituito da **PERCENT_RANK()** è la percentuale di valori che sono inferiori al valore della riga corrente.

Questa funzione fornisce un modo per comprendere come il valore di ogni riga si confronta con gli altri nel dataset, su una scala da 0 a 1, dove 0 rappresenta il primo valore nel set ordinato (dopo l'ordinamento basato sulla clausola **ORDER BY**) e i valori più vicini a 1 rappresentano ranghi più alti.

Ecco come funziona la funzione **PERCENT_RANK()** in questo scenario specifico:

1. **Preparazione dei Dati**: L'Espressione di Tabella Comune (CTE) **EmployeeSales** crea un dataset di **EmployeeId** e **SalesAmount**.
2. **Ordinamento**: La funzione **PERCENT_RANK()** viene applicata al dataset ordinato per **SalesAmount**. Questo significa che calcola il rango percentile basato sull'importo delle vendite di ogni dipendente.
3. **Calcolo del Rango Percentile**: Per ogni riga (cioè, l'importo delle vendite di ogni dipendente), la funzione **PERCENT_RANK()** calcola il rango della riga diviso per il numero di righe meno uno. La formula utilizzata è: **(rango – 1) / (totale righe – 1)**. La prima riga dopo l'ordinamento ottiene un rango percentile di 0, e il valore più alto ottiene un rango percentile 1 (poiché viene calcolato come **(N-1)/(N-1)**, dove **N** è il numero totale delle righe).

Per esempio, considerando il nostro dataset:

- Il dipendente con **SalesAmount** di 3000 (EmployeeId 4) avrebbe l'importo di vendite più basso, quindi il rango più basso, e riceverebbe un **PercentileRank** di 0 perché è il primo valore.
- Il dipendente con **SalesAmount** di 9400 (EmployeeId 3) avrebbe l'importo di vendite più alto. Se ci sono 5 dipendenti, il suo rango percentile sarebbe calcolato sulla base della sua posizione nella lista ordinata, che sarebbe **(4)/(5-1)** = 1.0, indicando che ha un importo di vendite superiore rispetto al resto del dataset.

Questa funzione è particolarmente utile per analizzare e confrontare la distribuzione dei valori all'interno di un dataset, consentendo di comprendere come ogni valore si posizioni in relazione agli altri, specialmente in termini di classifiche e percentili.

Conclusioni

In conclusione, l'integrazione delle funzioni SQL e OLAP all'interno dell'ambiente IBM iSeries preannuncia un'era di trasformazione per la scienza dei dati e la business intelligence.

Sfruttando le capacità avanzate delle funzioni OLAP, le organizzazioni possono ottenere informazioni più approfondite dai propri dati, facilitando un processo decisionale strategico più informato e migliorando l'efficienza operativa. La piattaforma IBM iSeries, con il suo solido supporto per le operazioni SQL e OLAP, rappresenta uno strumento significativo nell'arsenale di data scientist e analisti.

Attraverso funzioni aggregate, classificazione, operazioni di finestra e aggregati analitici, iSeries consente un approccio analitico completo che sfrutta l'intero spettro di tecniche della scienza dei dati.

Come abbiamo esplorato attraverso esempi pratici, la potenza delle funzioni OLAP per eseguire query e analisi complesse direttamente sul database non è solo un miglioramento tecnico; è una risorsa strategica che può far avanzare le aziende in un panorama competitivo.

L'adozione di queste funzionalità sulla piattaforma IBM iSeries può portare le organizzazioni a sfruttare tutto il potenziale dei propri dati, rendendoli non solo una risorsa ma un catalizzatore di innovazione e crescita.

Se si ha a che fare con vasti set di dati o complesse query analitiche, la sinergia di data science e SQL con le funzioni OLAP su IBM iSeries è una testimonianza della rilevanza duratura della piattaforma e del suo ruolo fondamentale nel futuro del processo decisionale basato sui dati.

Gli esempi illustrati dovrebbero fornire un buon punto di partenza per l'uso delle funzioni OLAP in SQL IBM iSeries. Ricordare sempre che la sintassi effettiva e le capacità possono variare leggermente a seconda della versione di iSeries e delle impostazioni del database, quindi è sempre una buona idea consultare la documentazione specifica di IBM i per SQL.

Riferimenti utili

- Data Science: opportunità e sfide da affrontare
- Funzioni OLAP in Data Science: un approccio moderno alla Business Intelligence

← PRECEDENTE

Recursive CTE come concatenare i risultati su un'unica riga

SEGUENTE →

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

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