# **FUNZIONI**

## **FUNZIONI DI AGGREGAZIONE**

### **AVG**

La funzione AVG viene utilizzata per calcolare la media dei valori di una colonna.

**Pseudocodice**

AVG(colonna)

**Esempio**

Supponiamo di avere una tabella ordini con le seguenti colonne:

| id | cliente | prodotto | quantità |
| --- | --- | --- | --- |

La seguente query restituirà la media della quantità degli ordini:

SELECT AVG(quantità) AS media\_quantità  
FROM ordini;

### **CHECKSUM\_AGG**

La funzione CHECKSUM\_AGG viene utilizzata per calcolare un hash della colonna specificata.

**Pseudocodice**

CHECKSUM\_AGG(colonna)

**Esempio**

Supponiamo di avere una tabella ordini con le seguenti colonne:

| id | cliente | prodotto | quantità |
| --- | --- | --- | --- |

La seguente query restituirà un hash della colonna cliente:

SELECT CHECKSUM\_AGG(cliente) AS hash\_cliente  
FROM ordini;

### **COUNT**

La funzione COUNT viene utilizzata per contare il numero di righe di una tabella o di una colonna.

**Pseudocodice**

COUNT(\*)  
COUNT(colonna)

**Esempio**

Supponiamo di avere una tabella ordini con le seguenti colonne:

| id | cliente | prodotto | quantità |
| --- | --- | --- | --- |

La seguente query restituirà il numero totale di ordini:

SELECT COUNT(\*) AS numero\_ordini  
FROM ordini;

La seguente query restituirà il numero di ordini per il cliente "Mario":

SELECT COUNT(id) AS numero\_ordini\_mario  
FROM ordini  
WHERE cliente = 'Mario';

### **SUM**

La funzione SUM viene utilizzata per calcolare la somma dei valori di una colonna.

**Pseudocodice**

SUM(colonna)

**Esempio**

Supponiamo di avere una tabella ordini con le seguenti colonne:

| id | cliente | prodotto | quantità |
| --- | --- | --- | --- |

La seguente query restituirà la somma di tutte le quantità degli ordini:

SELECT SUM(quantità) AS totale\_quantità  
FROM ordini;

La seguente query restituirà la somma delle quantità degli ordini per il cliente "Mario":

SELECT SUM(quantità) AS totale\_quantità\_mario  
FROM ordini  
WHERE cliente = 'Mario';

### **MAX**

La funzione MAX viene utilizzata per restituire il valore massimo di una colonna.

**Pseudocodice**

MAX(colonna)

**Esempio**

Supponiamo di avere una tabella ordini con le seguenti colonne:

| id | cliente | prodotto | quantità |
| --- | --- | --- | --- |

La seguente query restituirà la quantità massima di tutti gli ordini:

SELECT MAX(quantità) AS quantità\_massima  
FROM ordini;

La seguente query restituirà la quantità massima degli ordini per il cliente "Mario":

SELECT MAX(quantità) AS quantità\_massima\_mario  
FROM ordini  
WHERE cliente = 'Mario';

### **MIN**

La funzione MIN viene utilizzata per restituire il valore minimo di una colonna.

**Pseudocodice**

MIN(colonna)

**Esempio**

Supponiamo di avere una tabella ordini con le seguenti colonne:

| id | cliente | prodotto | quantità |
| --- | --- | --- | --- |

La seguente query restituirà la quantità minima di tutti gli ordini:

SELECT MIN(quantità) AS quantità\_minima  
FROM ordini;

La seguente query restituirà la quantità minima degli ordini per il cliente "Mario":

SELECT MIN(quantità) AS quantità\_minima\_mario  
FROM ordini  
WHERE cliente = 'Mario';

## **WINDOW FUNCTIONS**

le window functions sono funzioni che vengono utilizzate per eseguire calcoli su un set di dati, prendendo in considerazione il contesto delle righe circostanti.

Pseudocodice

SQL

SELECT

colonna1,

colonna2,

...

funzione\_window(colonna) OVER ([PARTITION By colonna\_ordinata] ORDER BY colonna\_ordinata) AS nome\_colonna\_funzione

FROM

tabella;

La funzione funzione\_window può essere qualsiasi funzione SQL, ad esempio SUM, AVG, MAX, MIN, COUNT, ecc.

La clausola PARTITION BY viene utilizzata per suddividere il set di dati in parti, in modo che la funzione funzione\_window venga applicata a ciascuna parte.

La clausola ORDER BY viene utilizzata per ordinare le parti del set di dati, in modo che la funzione funzione\_window venga applicata in modo coerente.

### **CUM\_DIST**

La funzione **CUM\_DIST** viene utilizzata per restituire la distribuzione cumulativa dei valori di una colonna.

**Pseudocodice**

CUM\_DIST(colonna)

**Esempio**

Supponiamo di avere una tabella ordini con le seguenti colonne:

| id | cliente | prodotto | quantità |
| --- | --- | --- | --- |

La seguente query restituirà la distribuzione cumulativa delle quantità degli ordini:

SELECT  
 id,  
 cliente,  
 prodotto,  
 quantità,  
 CUM\_DIST(quantità) AS distribuzione\_cumulativa  
FROM ordini;

La prima riga della tabella restituirà un valore di 1, in quanto è il primo valore della colonna quantità. La seconda riga restituirà un valore di 2, in quanto è il secondo valore della colonna quantità e così via.

### **DENSE\_RANK**

La funzione **DENSE\_RANK** viene utilizzata per restituire il rango di un valore in una colonna, ignorando i valori duplicati.

DENSE\_RANK(colonna)

**Esempio**

Supponiamo di avere una tabella ordini con le seguenti colonne:

| id | cliente | prodotto | quantità |
| --- | --- | --- | --- |

La seguente query restituirà il rango delle quantità degli ordini, ignorando i valori duplicati:

SELECT  
 id,  
 cliente,  
 prodotto,  
 quantità,  
 DENSE\_RANK(quantità) AS rango\_densa  
FROM ordini;

La prima riga della tabella restituirà un valore di 1, in quanto è il primo valore della colonna quantità. La seconda riga restituirà un valore di 2, in quanto è il secondo valore della colonna quantità che non è duplicato.

### **FIRST\_VALUE**

La funzione **FIRST\_VALUE** viene utilizzata per restituire il primo valore di una colonna, ordinando la colonna per un'altra colonna.

**Pseudocodice**

FIRST\_VALUE(colonna) OVER (PARTITION BY colonna\_ordinata ORDER BY colonna\_ordina)

**Esempio**

Supponiamo di avere una tabella ordini con le seguenti colonne:

| id | cliente | prodotto | quantità |
| --- | --- | --- | --- |

La seguente query restituirà il primo valore della colonna quantità per ogni cliente, ordinando la colonna quantità per la colonna cliente:

SELECT  
 cliente,  
 FIRST\_VALUE(quantità) OVER (PARTITION BY cliente ORDER BY quantità) AS prima\_quantità  
FROM ordini;

La prima riga della tabella restituirà il primo valore della colonna quantità per il cliente "Mario", che è 1. La seconda riga restituirà il primo valore della colonna quantità per il cliente "Anna", che è 2.

### **LAG**

La funzione **LAG** viene utilizzata per restituire il valore precedente di una colonna, ordinando la colonna per un'altra colonna.

**Pseudocodice**

LAG(colonna) OVER (PARTITION BY colonna\_ordinata ORDER BY colonna\_ordina)

**Esempio**

Supponiamo di avere una tabella ordini con le seguenti colonne:

| id | cliente | prodotto | quantità |
| --- | --- | --- | --- |

La seguente query restituirà il valore precedente della colonna quantità per ogni cliente, ordinando la colonna quantità per la colonna cliente:

SELECT  
 cliente,  
 quantità,  
 LAG(quantità) OVER (PARTITION BY cliente ORDER BY quantità) AS quantità\_precedente  
FROM ordini;

La prima riga della tabella restituirà il valore precedente della colonna quantità per il cliente "Mario", che è NULL, in quanto è la prima riga della tabella. La seconda riga restituirà il valore precedente della colonna quantità per il cliente "Anna", che è 1.

### LAST\_VALUE

La funzione **LAST\_VALUE** viene utilizzata per restituire l'ultimo valore di una colonna, ordinando la colonna per un'altra colonna.

**Pseudocodice**

LAST\_VALUE(colonna) OVER (PARTITION BY colonna\_ordinata ORDER BY colonna\_ordina)

**Esempio**

Supponiamo di avere una tabella ordini con le seguenti colonne:

| id | cliente | prodotto | quantità |
| --- | --- | --- | --- |

La seguente query restituirà l'ultimo valore della colonna quantità per ogni cliente, ordinando la colonna quantità per la colonna cliente:

SELECT  
 cliente,  
 quantità,  
 LAST\_VALUE(quantità) OVER (PARTITION BY cliente ORDER BY quantità) AS quantità\_ultima  
FROM ordini;

La prima riga della tabella restituirà l'ultimo valore della colonna quantità per il cliente "Mario", che è 3. La seconda riga restituirà l'ultimo valore della colonna quantità per il cliente "Anna", che è 2.

### LEAD

La funzione **LEAD** viene utilizzata per restituire il valore successivo di una colonna, ordinando la colonna per un'altra colonna.

**Pseudocodice**

LEAD(colonna) OVER (PARTITION BY colonna\_ordinata ORDER BY colonna\_ordina)

**Esempio**

Supponiamo di avere una tabella ordini con le seguenti colonne:

| id | cliente | prodotto | quantità |
| --- | --- | --- | --- |

La seguente query restituirà il valore successivo della colonna quantità per ogni cliente, ordinando la colonna quantità per la colonna cliente:

SELECT  
 cliente,  
 quantità,  
 LEAD(quantità) OVER (PARTITION BY cliente ORDER BY quantità) AS quantità\_successiva  
FROM ordini;

La prima riga della tabella restituirà il valore successivo della colonna quantità per il cliente "Mario", che è NULL, in quanto è l'ultima riga della tabella. La seconda riga restituirà il valore successivo della colonna quantità per il cliente "Anna", che è 3.

### NTILE

La funzione **NTILE** viene utilizzata per dividere i dati di una colonna in un numero specificato di gruppi.

**Pseudocodice**

NTILE(numero\_gruppi) OVER (PARTITION BY colonna\_ordinata ORDER BY colonna\_ordina)

**Esempio**

Supponiamo di avere una tabella ordini con le seguenti colonne:

| id | cliente | prodotto | quantità |
| --- | --- | --- | --- |

La seguente query restituirà i gruppi in cui sono stati suddivisi i dati della colonna quantità, in base a un numero di gruppi pari a 4:

SELECT  
 cliente,  
 quantità,  
 NTILE(4) OVER (PARTITION BY cliente ORDER BY quantità) AS gruppo  
FROM ordini;

La prima riga della tabella restituirà il gruppo in cui è stata inserita la prima riga, che è il gruppo 1. La seconda riga restituirà il gruppo in cui è stata inserita la seconda riga, che è il gruppo 2.

### PERCENT\_RANK

La funzione **PERCENT\_RANK** viene utilizzata per restituire la posizione percentuale di un valore in una colonna, rispetto a tutti gli altri valori della colonna.

**Pseudocodice**

PERCENT\_RANK() OVER (PARTITION BY colonna\_ordinata ORDER BY colonna\_ordina)

**Esempio**

Supponiamo di avere una tabella ordini con le seguenti colonne:

| id | cliente | prodotto | quantità |
| --- | --- | --- | --- |

La seguente query restituirà la posizione percentuale della colonna quantità per ogni cliente, rispetto a tutti gli altri valori della colonna quantità:

SELECT  
 cliente,  
 quantità,  
 PERCENT\_RANK() OVER (PARTITION BY cliente ORDER BY quantità) AS posizione\_percentuale  
FROM ordini;

La prima riga della tabella restituirà la posizione percentuale della colonna quantità per il cliente "Mario", che è del 50%. La seconda riga restituirà la posizione percentuale della colonna quantità per il cliente "Anna", che è del 25%.

### ROW\_NUMBER

La funzione **ROW\_NUMBER** viene utilizzata per assegnare un numero progressivo a ciascuna riga di un risultato di query, a partire da 1.

**Pseudocodice**

ROW\_NUMBER() OVER (PARTITION BY colonna\_ordinata ORDER BY colonna\_ordina)

**Esempio**

Supponiamo di avere una tabella ordini con le seguenti colonne:

| id | cliente | prodotto | quantità |
| --- | --- | --- | --- |

La seguente query restituirà un numero progressivo per ciascuna riga della tabella, ordinando la colonna id per valore crescente:

SELECT  
 id,  
 cliente,  
 prodotto,  
 quantità,  
 ROW\_NUMBER() OVER (ORDER BY id) AS numero\_riga  
FROM ordini;

La prima riga della tabella restituirà il numero 1, in quanto è la prima riga della tabella. La seconda riga restituirà il numero 2, in quanto è la seconda riga della tabella e così via.

La funzione **ROW\_NUMBER** può essere utilizzata in combinazione con altre funzioni, come **PARTITION BY** e **ORDER BY**, per ottenere risultati più specifici. Ad esempio, la seguente query restituirà un numero progressivo per ciascuna riga della tabella, ordinando la colonna quantità per valore crescente e limitando i risultati ai clienti di Roma:

SELECT  
 cliente,  
 quantità,  
 ROW\_NUMBER() OVER (PARTITION BY cliente ORDER BY quantità) AS numero\_riga  
FROM ordini  
WHERE cliente = 'Roma';

In questo caso, la prima riga della tabella restituirà il numero 1, in quanto è la prima riga della tabella per il cliente "Roma". La seconda riga restituirà il numero 2, in quanto è la seconda riga della tabella per il cliente "Roma" e così via.

## FUNZIONI TEMPORALI

### GETDATE

La funzione **GETDATE** viene utilizzata per restituire la data e l'ora correnti.

**Pseudocodice**

GETDATE()

**Esempio**

Supponiamo di avere una tabella ordini con le seguenti colonne:

| id | cliente | prodotto | quantità | data\_ordine |
| --- | --- | --- | --- | --- |

La seguente query restituirà la data e l'ora correnti:

SELECT GETDATE() AS data\_corrente;

### DATENAME

La funzione **DATENAME** viene utilizzata per restituire il nome di una parte di una data.

**Pseudocodice**

DATENAME(part\_of\_date, data)

**Esempio**

Supponiamo di avere una tabella ordini con le seguenti colonne:

| id | cliente | prodotto | quantità | data\_ordine |
| --- | --- | --- | --- | --- |

La seguente query restituirà il nome del mese della data dell'ordine:

SELECT DATENAME(month, data\_ordine) AS nome\_mese;

### DATEPART

La funzione **DATEPART** viene utilizzata per restituire il valore di una parte di una data.

**Pseudocodice**

DATEPART(part\_of\_date, data)

**Esempio**

Supponiamo di avere una tabella ordini con le seguenti colonne:

| id | cliente | prodotto | quantità | data\_ordine |
| --- | --- | --- | --- | --- |

La seguente query restituirà il giorno della settimana della data dell'ordine:

SELECT DATEPART(weekday, data\_ordine) AS giorno\_settimana;

### DAY

La funzione **DAY** viene utilizzata per restituire il giorno di una data.

**Pseudocodice**

DAY(data)

**Esempio**

Supponiamo di avere una tabella ordini con le seguenti colonne:

| id | cliente | prodotto | quantità | data\_ordine |
| --- | --- | --- | --- | --- |

La seguente query restituirà il giorno della data dell'ordine:

SELECT DAY(data\_ordine) AS giorno;

### MONTH

La funzione **MONTH** viene utilizzata per restituire il mese di una data.

**Pseudocodice**

MONTH(data)

**Esempio**

Supponiamo di avere una tabella ordini con le seguenti colonne:

| id | cliente | prodotto | quantità | data\_ordine |
| --- | --- | --- | --- | --- |

La seguente query restituirà il mese della data dell'ordine:

SELECT MONTH(data\_ordine) AS mese;

### YEAR

La funzione **YEAR** viene utilizzata per restituire l'anno di una data.

**Pseudocodice**

YEAR(data)

**Esempio**

Supponiamo di avere una tabella ordini con le seguenti colonne:

| id | cliente | prodotto | quantità | data\_ordine |
| --- | --- | --- | --- | --- |

La seguente query restituirà l'anno della data dell'ordine:

SELECT YEAR(data\_ordine) AS anno;

### DATEDIFF

La funzione **DATEDIFF** viene utilizzata per calcolare la differenza tra due date.

**Pseudocodice**

DATEDIFF(part\_of\_date, data\_iniziale, data\_finale)

**Esempio**

Supponiamo di avere una tabella ordini con le seguenti colonne:

| id | cliente | prodotto | quantità | data\_ordine |
| --- | --- | --- | --- | --- |

La seguente query restituirà la differenza in giorni tra la data dell'ordine e la data odierna:

SELECT DATEDIFF(day, data\_ordine, GETDATE()) AS differenza\_giorni;

Queste funzioni possono essere utilizzate per eseguire una varietà di calcoli sui dati relativi alle date.

## Funzioni per le stringhe

Le funzioni SQL per lavorare con le stringhe sono utilizzate per eseguire una varietà di operazioni sulle stringhe, come la loro concatenazione, la loro modifica e la loro analisi.

**Tabella delle funzioni**

| Funzione | Descrizione | Pseudocodice | Esempio |
| --- | --- | --- | --- |
| LEFT | Restituisce la parte sinistra di una stringa, a partire da una posizione specificata | LEFT(stringa, lunghezza) | SELECT LEFT("Ciao mondo", 3) AS prima\_parte |
| RIGHT | Restituisce la parte destra di una stringa, a partire da una posizione specificata | RIGHT(stringa, lunghezza) | SELECT RIGHT("Ciao mondo", 3) AS seconda\_parte |
| MID | Restituisce una parte centrale di una stringa, a partire da una posizione specificata e con una lunghezza specificata | MID(stringa, posizione, lunghezza) | SELECT MID("Ciao mondo", 2, 3) AS parte\_centrale |
| LENGTH | Restituisce la lunghezza di una stringa | LENGTH(stringa) | SELECT LENGTH("Ciao mondo") AS lunghezza |
| UPPER | Converte una stringa in maiuscolo | UPPER(stringa) | SELECT UPPER("ciao mondo") AS maiuscolo |
| LOWER | Converte una stringa in minuscolo | LOWER(stringa) | SELECT LOWER("CIAO MONDO") AS minuscolo |
| LTRIM | Rimuove gli spazi bianchi iniziali da una stringa | LTRIM(stringa) | SELECT LTRIM(" Ciao mondo ") AS senza\_spazi\_iniziali |
| RTRIM | Rimuove gli spazi bianchi finali da una stringa | RTRIM(stringa) | SELECT RTRIM("Ciao mondo ") AS senza\_spazi\_finali |
| TRIM | Rimuove tutti gli spazi bianchi da una stringa | TRIM(stringa) | SELECT TRIM(" Ciao mondo ") AS senza\_spazi |
| REPLACE | Sostituisce una parte di una stringa con un'altra parte | REPLACE(stringa, vecchio, nuovo) | SELECT REPLACE("Ciao mondo", "mondo", "universo") AS nuovo\_mondo |
| SUBSTRING | Restituisce una sottostringa di una stringa, a partire da una posizione specificata | SUBSTRING(stringa, posizione, lunghezza) | SELECT SUBSTRING("Ciao mondo", 2, 3) AS sottostringa |
| CHARINDEX | Restituisce la posizione di una sottostringa in una stringa | CHARINDEX(sottostringa, stringa) | SELECT CHARINDEX("mondo", "Ciao mondo") AS posizione\_mondo |
| PATINDEX | Restituisce la posizione di una sottostringa regolare in una stringa | PATINDEX(pattern, stringa) | SELECT PATINDEX("%mondo%", "Ciao mondo") AS posizione\_mondo\_regex |

### Esempi pratici

Supponiamo di avere una tabella ordini con le seguenti colonne:

| id | cliente | prodotto | quantità | data\_ordine |
| --- | --- | --- | --- | --- |

La seguente query restituirà la data di ordine, con il giorno della settimana in maiuscolo:

SELECT UPPER(DATENAME(weekday, data\_ordine)) AS giorno\_settimana  
FROM ordini;

La seguente query restituirà il nome del cliente, con la prima lettera in maiuscolo:

SELECT UPPER(LEFT(cliente, 1)) + LOWER(SUBSTRING(cliente, 2)) AS cliente\_con\_prima\_lettera\_maiuscola  
FROM ordini;

La seguente query restituirà il prodotto, con tutti gli spazi bianchi sostituiti da un trattino:

SELECT REPLACE(prodotto, " ", "-") AS prodotto\_con\_trattini  
FROM ordini;

Queste sono solo alcune delle molte funzioni SQL che possono essere utilizzate per lavorare con le stringhe.