

SQL Data Manipulation Language

Funzioni

DML – Funzioni di SQL Server

In precedenza abbiamo già incontrato un tipo particolare di funzioni, le funzioni di aggregazione.

Sono disponibili molte altre funzioni, che lavorano principalmente su un singolo campo, tramite le quali è possibile costruire query ancora più complesse e articolate.

Per esempio:

```
SELECT ROUND( AVG(Age) , 2 ) FROM AthletesFull WHERE NOC = 'ITA'
```

Il risultato della funzione di aggregazione AVG è passato in input alla funzione ROUND assieme ad un secondo parametro fisso di valore 2.

La funzione ROUND prende il numero passato come primo parametro e lo arrotonda generando un numero con tanti decimali quanti sono specificati nel secondo parametro.

DML – Funzioni di SQL Server

Una funzione accetta uno o più parametri di input e restituisce un solo valore di output.

Gli input possono essere, indifferentemente, forniti tramite:

- Un campo di una query
- Un valore fisso
- Il risultato di altra funzione

Gli output possono invece essere utilizzati:

- Per generare una colonna di output di una query
- Per valorizzare campi in una query di UPDATE o INSERT
- Per definire clausole di JOIN, WHERE, GROUP BY...

DML – Funzioni di SQL Server

Tutti i dialetti SQL prevedono funzioni ma spesso i nomi e i parametri sono molto differenti tra una versione di SQL e l'altra.

Vedremo alcune funzioni di SQL Server. L'elenco completo è riportato nella documentazione ufficiale:

<https://docs.microsoft.com/it-it/sql/t-sql/functions/functions>

Le funzioni più utilizzate possono essere raggruppate in 4 categorie:

1. Funzioni che lavorano su stringhe
2. Funzioni che lavorano su numeri
3. Funzioni che lavorano su date
4. Funzioni generali e trasversali

DML – Funzioni di che lavorano su stringhe

Sono funzioni che permettono di manipolare stringhe di testo modificandole secondo necessità.

Funzione	Descrizione
CHARINDEX	Restituisce la posizione della prima occorrenza di una stringa all'interno di un'altra stringa
CONCAT	Concatena due o più stringhe generandone una unica
FORMAT	Formatta un numero come stringa
LEFT / RIGHT	Estrae caratteri da una stringa, partendo da sinistra / da destra
LEN	Restituisce la lunghezza di una stringa
LOWER / UPPER	Trasforma una stringa in lettere minuscole / maiuscole
LTRIM / RTRIM / TRIM	Eliminano gli spazi bianchi da una stringa rispettivamente all'inizio, alla fine e in entrambi i lati
REPLACE	Sostituisce una porzione di una stringa con un'altra
SUBSTRING	Estrae una parte di una stringa

DML – Funzioni di che lavorano su stringhe / Esempi

```
SELECT UPPER(Medal) FROM Partecipations;
```

```
SELECT A.* FROM Athletes A ORDER BY TRIM(A.Name)
```

```
SELECT A.*, LEN(A.Name) FROM Athletes A
```

```
SELECT A.*, LEN(A.Name) FROM Athletes A ORDER BY LEN(A.Name) DESC
```

DML – Funzioni di che lavorano su numeri

Sono funzioni che permettono di effettuare operazioni matematiche su numeri interi e/o decimali.

Funzione	Descrizione
ABS	Calcola il valore assoluto di un numero
CEILING	Restituisce l'intero più piccolo che è \geq al numero passato come parametro
POWER	Calcola la potenza fornendo base e esponente
FLOOR	Restituisce l'intero più grande che è \leq al numero passato come parametro
RAND	Restituisce un numero casuale
ROUND	Arrotonda un numero specificandone i decimali
SQRT	Calcola la radice quadrata

DML – Funzioni di che lavorano su numeri

Funzione	Descrizione
SIN	Calcola il seno di un numero in radianti
COS	Calcola il coseno di un numero in radianti
TAN	Calcola la tangente di un numero in radianti
ASIN	Calcola l'arcoseno di un numero nel range -1 1
ACOS	Calcola l'arcocoseno di un numero nel range -1 1
ATAN	Calcola l'arcotangente di un numero
DEGREES	Converte un numero da radianti a gradi
RADIANS	Converte un numero da gradi a radianti
PI	Restituisce la costante pi greco

DML – Funzioni di che lavorano su numeri / Esempi

```
SELECT
```

```
AVG( CAST(Age AS FLOAT) ) As Avg,
```

```
ABS(AVG( CAST(Age AS FLOAT) )) As Abs,
```

```
CEILING(AVG( CAST(Age AS FLOAT) )) As Ceil,
```

```
FLOOR(AVG( CAST(Age AS FLOAT) )) As Floor,
```

```
ROUND(AVG( CAST(Age AS FLOAT) ),2) As Round
```

```
FROM AthletesFull WHERE NOC = 'ITA'
```

DML – Funzioni di che lavorano su date

Sono funzioni che permettono di effettuare operazioni sui tipi di date DATE, TIME e DATETIME.

Funzione	Descrizione
DATEADD	Aggiunge una data o un intervallo temporale e restituisce la nuova data
DATEDIFF	Restituisce il numero di giorni che separa due date
GETDATE	Restituisce la data attuale
DAY	Estrae il giorno, numerico, da una data
MONTH	Estrae il mese, numerico, da una data
YEAR	Estrae l'anno, numerico, da una data

DML – Funzioni di che lavorano su date / Esempi

```
SELECT GETDATE ( ) ,  
       DAY ( GETDATE ( ) ) ,  
       MONTH ( GETDATE ( ) ) ,  
       YEAR ( GETDATE ( ) )
```

DML – Funzioni generali e trasversali

Sono funzioni generali di supporto alla creazione di query complesse.

Funzione	Descrizione
CASE	Permette di creare una serie di condizioni mutuamente esclusive
CAST	Permette di convertire un tipo di dato in un altro tipo di dato
COALESCE	Restituisce il primo valore NOT NULL
IIF	Permette di realizzare condizioni di tipo IF THEN ELSE

DML – Funzioni generali e trasversali - CASE

Il costrutto CASE permette di specificare una serie di condizioni e altrettanti valori da restituire quando la relativa condizione è verificata.

Nel caso di più condizioni verificate CASE restituisce il valore relativo alla prima.

```
SELECT DISTINCT Medal,  
  
CASE  
  
    WHEN Medal = 'Gold' THEN 1  
  
    WHEN Medal = 'Silver' THEN 2  
  
    WHEN Medal = 'Bronze' THEN 3  
  
    ELSE 4  
  
END AS MedalOrder  
  
FROM Participations ORDER BY MedalOrder
```

DML – Funzioni generali e trasversali - CAST

Talvolta è necessario convertire i dati da un tipo ad un altro. Può essere necessario per esigenze di output o per esigenze della query stessa.

La funzione CAST vuole due parametri, separati dalla parola chiave AS: l'espressione da convertire e il tipo di dato di destinazione.

```
SELECT CAST(COUNT(DISTINCT Sport) AS FLOAT) FROM AthletesFull
```

```
SELECT CAST(Year AS DATE) FROM Games
```

```
SELECT CAST( AVG(AGE) AS DOUBLE) FROM Partecipations
```

DML – Funzioni generali e trasversali - COALESCE

In alcune situazioni c'è la necessità di evitare che il risultato di una query contenga valori NULL, senza però cancellare le righe che li contengono.

La funzione COALESCE permette di specificare più espressioni, separate da virgola, e restituisce la prima espressione che contiene un valore NOT NULL

```
SELECT A.*, COALESCE(P.Medal, '')  
FROM Athletes A LEFT JOIN Partecipations P ON A.IdAthlete =  
P.IdAthlete
```

DML – Funzioni generali e trasversali - IIF

La funzione IF permette di specificare una condizione e due valori di ritorno, uno da restituire in caso di condizione vera e uno in caso di condizione falsa.

IF è una versione più compatta di CASE con un solo WHERE e ELSE.

```
SELECT *, IIF(Age <18, 'Atleta Minorenne', '')  
FROM Athletes A INNER JOIN Partecipations P ON A.IdAthlete =  
P.IdAthlete
```


Esercizi: Funzioni

Con riferimento alla tabelle normalizzate dei dati relativi alle olimpiadi:

1. Per ogni atleta di cui sono noti peso e altezza calcolare l'indice di massa corporeo (BMI) arrotondandolo a 2 decimali. Ordinare per IdAthlete. Il BMI è calcolato come:
$$\text{BMI} = \text{Peso} / \text{Altezza}^2 \quad (\text{dove peso è in Kg e Altezza in metri})$$
2. Creare una query di aggiornamento per modificare i campi Games e Season della tabella Games affinché siano tutti in lettere maiuscole.
3. Creare una query di selezione su Athletes che restituisca Male / Female al posto di M o F

Esercizi: Funzioni - Soluzioni

Con riferimento alla tabelle normalizzate dei dati relativi alle olimpiadi:

1. Per ogni atleta di cui sono noti peso e altezza calcolare l'indice di massa corporeo (BMI) arrotondandolo a 2 decimali. Ordinare per IdAthlete. Il BMI è calcolato come:

$$\text{BMI} = \text{Peso} / \text{Altezza}^2 \quad (\text{dove peso è in Kg e Altezza in metri})$$

```
SELECT A.IdAthlete, A.Name,  
ROUND(A.Weight / POWER( CAST(A.Height AS FLOAT) /100,2) ,2 ) AS BMI  
FROM Athletes A  
WHERE A.Height IS NOT NULL AND A.Weight IS NOT NULL  
ORDER BY IdAthlete
```

Esercizi: Funzioni - Soluzioni

Con riferimento alla tabelle normalizzate dei dati relativi alle olimpiadi:

2. Creare una query di aggiornamento per modificare i campi Games e Season della tabella Games affinché siano tutti in lettere maiuscole.

```
UPDATE Games SET Season = UPPER(Season), Games = UPPER(Games)
```

3. Creare una query di selezione su Athletes che restituisca Male / Female al posto di M o F

```
SELECT A.IdAthlete, A.Name, IIF(Sex='M', 'Male', 'Female') AS Sex, A.Height, A.Weight  
FROM Athletes A
```

SQL Data Manipulation Language

Viste

DML – CREATE VIEW

Tramite l'istruzione CREATE VIEW è possibile creare una "tabella virtuale", detta vista, le cui righe e colonne sono il risultato di una query di selezione.

Alla vista è associato un nome e può essere utilizzata come se fosse una "tabella reale" in query di SELECT. Ci sono invece forti limitazioni nell'effettuare inserimenti, cancellazioni e aggiornamenti su una vista. La query associata alla vista viene eseguita ogni volta che fa riferimento alla vista.

```
CREATE VIEW AtletiItaliani AS
```

```
SELECT DISTINCT IdAthlete, Name FROM AthletesFull WHERE NOC = 'ITA'
```

La query utilizzata per creare una vista può utilizzare tutte le istruzioni viste in precedenza (WHERE, GROUP BY, JOIN...) permettendo quindi di costruire logiche anche molto complesse.

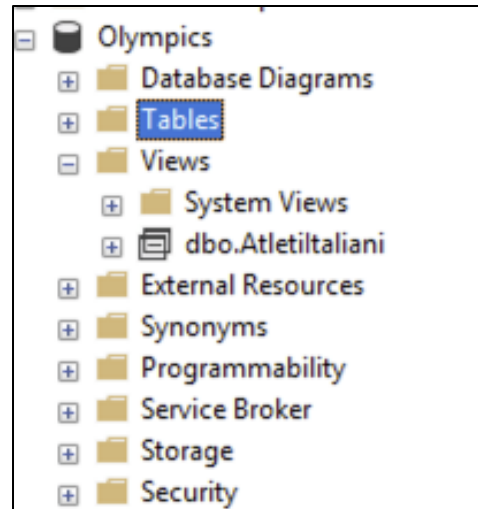
E' anche possibile costruire viste che si basano su altre viste.

DML – CREATE VIEW

L'utilizzo di una vista avviene come se fosse una tabella normale. Esistono però forti limitazioni nell'effettuare operazioni di INSERT, DELETE e UPDATE (lo vedremo tra poco)

```
SELECT * FROM AtletiItaliani
```

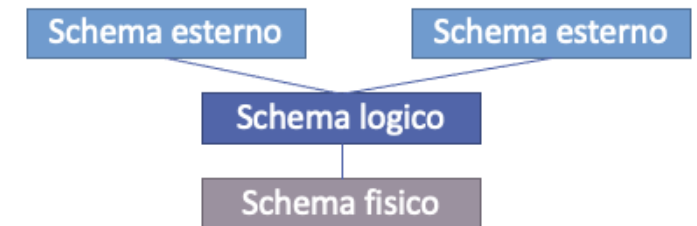
Le viste vanno create una volta sola, analogamente alle tabelle. Una volta create fanno parte degli oggetti presenti nel database e rimangono persistenti fino ad una eventuale cancellazione



DML – Utilizzo delle viste

Le viste possono essere utilizzate per vari scopi. I più frequenti sono:

- Realizzare uno o più schemi esterni per esporre versioni personalizzate del database agli applicativi esterni
- Gestire con maggior facilità le modifiche allo schema logico
- Semplificare la scrittura di query complesse
- Ottimizzare la scrittura delle query riusando il codice senza duplicarlo
- Gestire il controllo degli accessi, fornendo i diritti di esecuzione solamente ad alcune viste (lo vedremo più avanti)



DML – Utilizzo delle viste

- Realizzare uno o più schemi esterni per esporre versioni personalizzate del database agli applicativi esterni

```
CREATE VIEW AthletesView AS

SELECT A.IdAthlete, A.Name, A.Sex, P.Age, A.Height, A.Weight, P.Noc, G.Games, G.Year, G.Season,
G.City, E.Sport, E.Event, P.Medal

FROM Partecipations P

INNER JOIN Athletes A ON P.IdAthlete = A.IdAthlete

INNER JOIN Events E ON P.IdEvent = E.Id

INNER JOIN Games G ON P.IdGame = G.Id
```


DML – Utilizzo delle viste

- Gestire con maggior facilità le modifiche allo schema logico

Supponiamo che il campo Participations.NOC venga rinominato in Participations.Country e che gli applicativi utilizzino la vista AthletesView. E' sufficiente cambiare la vista senza ulteriori modifiche agli applicativi

```
ALTER VIEW AthletesView AS
```

```
SELECT A.IdAthlete, A.Name, A.Sex, P.Age, A.Height, A.Weight, P.Country AS NOC, G.Games, G.Year,  
G.Season, G.City, E.Sport, E.Event, P.Medal
```

```
FROM Participations P
```

```
INNER JOIN Athletes A ON P.IdAthlete = A.IdAthlete
```

```
INNER JOIN Events E ON P.IdEvent = E.Id
```

```
INNER JOIN Games G ON P.IdGame = G.Id
```

DML – Utilizzo delle viste

- Semplificare la scrittura di query complesse

```
CREATE VIEW GoldsView AS  
  
SELECT P.NOC, Count(distinct E.Event) AS Golds  
  
FROM Partecipations P INNER JOIN Events E ON E.Id = P.IdEvent  
  
WHERE P.IdGame = (SELECT Id FROM Games WHERE Year = 2016) AND MMedal = 'Gold' GROUP BY P.NOC
```

```
SELECT G.NOC, G.Golds, S.Silvers, B.Bronzes, G.Golds + S.Silvers + B.Bronzes AS Total FROM  
  
GoldsView G INNER JOIN SilversView S ON G.NOC = S.NOC  
  
INNER JOIN BronzesView B ON G.NOC = B.NOC  
  
ORDER BY G.Golds DESC, S.Silvers DESC, B.Bronzes DESC, G.NOC
```

DML – Utilizzo delle viste

- Ottimizzare la scrittura delle query riusando il codice senza duplicarlo

```
CREATE VIEW Medals2016View AS

SELECT G.NOC, G.Golds, S.Silvers, B.Bronzes, G.Golds + S.Silvers + B.Bronzes AS Total

FROM GoldsView G INNER JOIN SilversView S ON G.NOC = S.NOC

INNER JOIN BronzesView B ON G.NOC = B.NOC

ORDER BY G.Golds DESC, S.Silvers DESC, B.Bronzes DESC, G.NOC


SELECT * FROM Medals2016View WHERE NOC = 'ITA';

SELECT COUNT(*) FROM Medals2016View WHERE GOLDS > 0;

SELECT * FROM Medals2016View ORDER BY Total;
```

DML – VISTE e query di aggiornamento

Ci sono dei limiti all'utilizzo delle viste nelle query di aggiornamento, inserimento e modifica.

Si consideri per esempio la vista:

```
CREATE VIEW AtletiItaliani AS  
  
SELECT IdAthlete, Name, COUNT(DISTINCT Games) AS Partecipazioni  
  
FROM AthletesFull WHERE NOC = 'ITA' GROUP BY IdAthlete, Name
```

E la query di aggiornamento:

```
UPDATE AtletiItaliani SET Name = 'Mario Rossi' WHERE Id = 200
```

In una situazione del genere, vista la presenza di DISTINCT, di GROUP BY e di COUNT, il DBMS non saprebbe come effettuare l'aggiornamento.

Le più comuni restrizioni che portano le viste a non essere aggiornabili riguardano la presenza di GROUP BY, DISTINCT, Funzioni di aggregazione e JOIN. Negli altri casi le viste possono essere utilizzate anche con le funzioni di UPDATE.

DML – ALTER e DROP VIEW

La modifica e la cancellazione di una vista avvengono tramite le istruzioni ALTER VIEW e DROP VIEW.

La modifica di una vista è analoga alla creazione in quanto deve essere specificata, in modo completo, la nuova query da utilizzare.

```
ALTER VIEW AtletiItaliani AS  
  
SELECT IdAthlete, Name, COUNT(DISTINCT Games) AS Partecipazioni  
FROM AthletesFull WHERE NOC = 'ITA' GROUP BY IdAthlete, Name
```

```
DROP VIEW AtletiItaliani
```

Esercizi: Viste

Con riferimento alla tabella normalizzata dei dati relativi alle olimpiadi:

1. Creare una vista sulla tabella Athletes che restituisca solo gli atleti italiani. Fare in modo che sia aggiornabile.
Testare l'aggiornabilità portando in minuscolo il campo Sex
2. Creare una vista sulla tabella Athletes che restituisca solo i medagliati d'oro. Fare in modo che sia aggiornabile.
Testare l'aggiornabilità portando in minuscolo il campo Sex

Esercizi: Viste - Soluzioni

Con riferimento alla tabelle normalizzate dei dati relativi alle olimpiadi:

1. Creare una vista sulla tabella Athletes che restituisca solo gli atleti italiani. Fare in modo che sia aggiornabile.

Testare l'aggiornabilità portando in minuscolo il campo Sex

```
CREATE VIEW AtletiItaliani AS  
  
SELECT * FROM Athletes A  
  
WHERE A.IdAthlete IN (  
  
    SELECT DISTINCT IdAthlete FROM Partecipations WHERE NOC='ITA'  
  
)  
  
  
UPDATE AtletiItaliani SET Sex = LOWER(Sex);
```

Esercizi: Viste - Soluzioni

Con riferimento alla tabelle normalizzate dei dati relativi alle olimpiadi:

2. Creare una vista sulla tabella *Athletes* che restituisca solo i medagliati d'oro. Fare in modo che sia aggiornabile.

Testare l'aggiornabilità portando in minuscolo il campo Sex

```
CREATE VIEW Medagliati AS

SELECT * FROM Athletes A

WHERE A.IdAthlete IN (

        SELECT DISTINCT IdAthlete FROM Partecipations WHERE Medal='Gold'

)
```

```
UPDATE Medagliati SET Sex = LOWER(Sex);
```


Stored Procedures

Stored procedures

Le viste introdotte in precedenza non permettono ancora di coprire tutti i casi. In particolare:

1. Le viste non sono parametriche. La query che crea una vista è fissa.
2. Le viste non permettono di eseguire operazioni complesse con vari flussi di esecuzione e con la gestione degli errori.

Le stored procedures, presenti in SQL Server e in molti altri DBMS, permettono di superare queste limitazioni.

Una stored procedure è formata da un insieme di istruzioni SQL memorizzate con un nome e dotate di eventuali parametri. L'esecuzione dell'intero insieme di istruzioni avviene facendo riferimento al nome della stored procedure.

Oltre ad istruzioni SQL le stored procedure possono fare uso di istruzioni per il controllo del flusso, per la gestione di variabili e di errori.

Stored procedures

Le stored procedure diventano oggetti persistenti del database, analogamente a tabelle e viste.

Si creano con l'istruzione CREATE PROCEDURE, seguita dagli eventuali parametri e dalle istruzioni SQL che la compongono.

Se le istruzioni SQL sono più di una vanno racchiuse tra le parole chiave BEGIN e END.

```
CREATE PROCEDURE sp
AS
BEGIN
    SELECT * FROM AthletesFull
END
```

L'esecuzione della stored procedure, ovvero del codice che essa contiene, avviene tramite EXECUTE

```
EXECUTE sp
```

Stored procedures – Cancellazione e modifica

La cancellazione di una stored procedure avviene tramite l'istruzione **DROP PROCEDURE**

```
DROP PROCEDURE sp
```

La modifica di una store procedure avviene tramite il comando **ALTER** e specificando il nuovo codice della procedura:

```
ALTER PROCEDURE sp
```

```
AS
```

```
BEGIN
```

```
    SELECT * FROM AthletesFull
```

```
END
```

Stored procedures – Parametri di input

E' possibile passare dei parametri ad una stored procedure per renderle parametriche e per ampliare il grado di riuso.

Ogni parametro ha un nome, che deve necessariamente iniziare con @, e un tipo di dato (gli stessi previsti per i campi delle tabelle).

I parametri vanno indicati all'interno di parentesi tonde poste dopo il nome della procedura. In caso di più parametri vanno separati con la virgola ,

```
CREATE PROCEDURE spAtleti (@pNOC CHAR(3)) AS  
  
BEGIN  
  
    SELECT * FROM AthletesFull A WHERE A.NOC = @pNOC  
  
END  
  
EXECUTE spAtleti 'ITA'
```

Stored procedures – Variabili

SQL Server permette di definire variabili per memorizzare valori di qualsiasi tipo, in modo indipendente dalle tabelle del database. Sono valori provvisori, memorizzati per il solo tempo di esecuzione delle query in cui sono utilizzati. Servono pertanto per mantenere valori temporanei e per scambiare dati con query e/o con stored procedure.

Le variabili in SQL Server hanno un nome e vanno identificate antepoendo il carattere @ prima del nome. Devono essere prima dichiarate con la parola chiave DECLARE, specificandone anche il tipo, e poi assegnate con la parola chiave SET.

I tipi di dato utilizzabili sono gli stessi previsti per le colonne delle tabelle.

```
DECLARE @variabile1 INT
```

```
DECLARE @variabile2 INT, @variabile3 VARCHAR(10)
```

```
SET @variabile1 = 10
```

Stored procedures – Variabili

Le variabili in SQL Server possono essere inizializzate tramite l'istruzione di SELECT, prelevando il valore da uno dei parametri di output della query:

```
SELECT @variabile1 = COUNT(*) FROM Table
```

```
SELECT @variabile2 = Campo FROM Table WHERE Id = 123
```

Oppure prelevando il risultato scalare di una query:

```
SET @variabile1 = ( SELECT COUNT(*) FROM Table)
```

Stored procedures – Parametri di output

Una stored procedure può valorizzare dei parametri che verranno restituiti al chiamante sotto forma di variabili. I parametri di output di "sommano" ai risultati restituiti dall'ultima istruzione SQL presente dentro la procedura.

I parametri di output vanno specificati insieme agli altri parametri con l'aggiunta della parola chiave OUTPUT.

```
CREATE PROCEDURE spAtleti(@NOC CHAR(3), @conteggio INT OUTPUT) AS  
  
BEGIN  
  
    SELECT * FROM AthletesFull A WHERE A.NOC = @NOC  
  
    SELECT @conteggio = @@ROWCOUNT  
  
END
```

La prima select valorizza in parametro di output conteggio mentre la seconda select restituisce la lista degli atleti

Stored procedures – Parametri di output

Il valore di ritorno restituito da una store procedure può essere "intercettato" creando una variabile e passandola alla sp (utilizzando la parola chiave OUTPUT).

Al termine dell'esecuzione la variabile conterrà il valore di ritorno.

```
DECLARE @count INT
```

```
EXEC spAtleti 'ITA', @count OUTPUT
```

```
SELECT @count AS 'Conteggio Record'
```

Esercizi: Stored Procedure / 1

Con riferimento al database Azienda di moda:

1. Creare una stored procedure che crea un ordine di campionario, ovvero un ordine che contiene tutti gli articoli in quantità 1 e taglia M. Realizzare la sp con la seguente firma:
`spCreaOrdineCampionario(pIdCliente INT)` dove `pIdCliente` è il codice di un cliente esistente.
2. Realizzare una stored procedure che dato il codice di un filato restituisce l'elenco degli articoli che lo utilizzano con la relativa quantità. Realizzare la sp con la seguente firma:
`spUtilizzoFilato(pIdFilato INT)` dove `pIdFilato` è il codice di un filato esistente.

Esercizi: Stored Procedure / 1 - Soluzioni

Con riferimento al database Azienda di moda:

1. Creare una stored procedure che crea un ordine di campionario, ovvero un ordine che contiene tutti gli articoli in quantità 1 e taglia M. Realizzare la sp con la seguente firma: `spCreaOrdineCampionario(pIdCliente INT)` dove `pIdCliente` è il codice di un cliente esistente.

```
CREATE PROCEDURE spCreaOrdineCampionario(@pIdCliente INT) AS
BEGIN
    DECLARE @nOrd INT

    INSERT INTO ORDINI(Data,IdCliente,IdStato) VALUES(getdate(),@pIdCliente,1);

    SET @nOrd = (SELECT MAX(Id) FROM Ordini);

    INSERT INTO ArticoliOrdini(IdOrdine,IdArticolo,Idtaglia,Qta)
        SELECT @nOrd AS IdOrdine,Id AS IdArticolo,2 AS IdTaglia,1 AS Qta FROM ARTICOLI;

END
```

Esercizi: Stored Procedure / 1 - Soluzioni

Con riferimento al database Azienda di moda:

1. Realizzare una stored procedure che dato il codice di un filato restituisce l'elenco degli articoli che lo utilizzano con la relativa quantità. Realizzare la sp con la seguente firma:
`spUtilizzoFilato(pIdFilato INT)` dove `pIdFilato` è il codice di un filato esistente.

```
CREATE PROCEDURE spUtilizzoFilato(@pIdFilato INT) AS  
  
BEGIN  
  
    SELECT A.Id,A.Descrizione, SUM(FA.Qta)  
  
    FROM FilatiArticoli FA INNER JOIN Articoli A ON A.Id = FA.IdArticolo  
  
    WHERE FA.IdFilato = @pIdFilato  
  
    GROUP BY A.Id,A.Descrizione;  
  
END$
```

Stored procedures – Costrutti avanzati

Oltre alle variabili appena viste, le stored procedure possono fare uso anche di altri costrutti avanzati. In particolare:

- Istruzioni di controllo del flusso
- Tabelle temporanee
- Istruzioni di controllo degli errori

Con questi strumenti a disposizione le stored procedure possono essere viste come un linguaggio di programmazione a se stante, nel quale le istruzioni SQL sono solamente una parte.

Stored procedures – Controllo del flusso - IF

Il costrutto IF permette di creare delle ramificazioni all'interno di una stored procedure. Funziona in maniera completamente analoga a qualsiasi linguaggio di programmazione:

```
IF condizione
BEGIN
    istruzioni
END
ELSE
BEGIN
    istruzioni
END
```

Non esiste il costrutto ELSEIF e le parole chiave BEGIN e END possono essere omesse nel caso di una sola istruzione da eseguire.

Stored procedures – Controllo del flusso – WHILE

Il costrutto WHILE permette di creare un ciclo eseguendo più volte le istruzioni inserite all'interno dello stesso. Prevede una condizione di "esecuzione", ovvero il ciclo verrà eseguito fintanto che la condizione è verificata

```
WHILE condizione  
  
BEGIN  
  
    istruzioni  
  
END
```

All'interno del while è possibile utilizzare le istruzioni BREAK e CONTINUE con lo stesso significato visto in C#

Stored procedures – Tabelle temporanee

In SQL Server è possibile creare tabelle temporanee "visibili" solamente dalla connessione corrente e pertanto completamente invisibili ad altre connessioni e ad altri utenti che stanno operando sullo stesso database.

Sono molto utili per memorizzare dataset temporanei, da usarsi durante l'esecuzione di una query complessa o di una stored procedure.

Possono essere create in maniera esplicita, specificando ogni singolo campo, o in maniera implicita, come risultato di una query di select INTO. In ogni caso il nome della tabella deve essere preceduto da #

```
CREATE TABLE #TabellaTemp(  
    colonna1 DATATYPE,  
    colonna2 DATATYPE,  
)
```

```
SELECT * INTO #TabellaTemp FROM Table
```


Stored procedures – Tabelle temporanee

Le tabelle temporanee vengono automaticamente cancellate alla chiusura della connessione che le ha generate.

Tuttavia possono anche essere cancellate manualmente tramite `DROP TABLE`

Indici

Query e prestazioni

Quando eseguiamo una query il DBMS determina la modalità di esecuzione più performante, determinando il cosiddetto execution plan.

Prendiamo per esempio una semplice query:

```
SELECT * FROM AthletesFull A WHERE A.Id = 200
```

Il piano di esecuzione prevede di leggere tutte le righe della tabella AthletesFull fermandosi qualora se ne trovasse una con Id=200 (Id è chiave primaria per cui non ne possono esistere altre).

Nel caso migliore verrà letta una sola riga (sto cercando l'Id relativo alla prima riga memorizzata su disco), nel caso peggiore vengono lette tutte le righe (sto cercando un Id che non esiste)

Query e prestazioni

Prendiamo per esempio una semplice query:

```
SELECT * FROM AthletesFull A WHERE A.Id = 200
```

Un modo per velocizzare la query sarebbe quello di memorizzare i dati su disco (schema fisico) in modo che questi siano ordinati in base all'Id. In questo modo è possibile usare algoritmi di ricerca più sofisticati che, nel caso peggiore, richiedono di leggere $\log_2 N$ (dove N è il numero di righe della tabella).

Tuttavia mantenere i dati su disco ordinati NON è una cosa fattibile. Nel caso di inserimento di un record "intermedio" si renderebbe necessaria una riorganizzazione dell'intero archivio con necessità di spostamento di una quantità potenzialmente molto alta di dati.

Query e prestazioni

Prendiamo per un'altra query:

```
SELECT * FROM AthletesFull A WHERE A.NOC = 'ITA'
```

Vista l'impossibilità di mantenere un ordinamento sullo schema fisico, il piano di esecuzione di questa query non può che prevedere di controllare sempre e comunque TUTTE le righe della tabella. Il campo NOC infatti non è chiave e può pertanto essere duplicato.

Query e prestazioni

Prendiamo per una query leggermente più complessa:

```
SELECT * FROM Athletes A  
INNER JOIN Participations M ON A.IdAthlete = M.IdAthlete
```

L'operatore di JOIN prevede che per ogni riga della tabella Athletes vengano lette TUTTE le righe della tabella Participations alla ricerca di quelle che soddisfano il predicato di Join

Indici

L'unico modo per migliorare le prestazioni di ricerca è quello di ordinare i dati utilizzando strutture esterne dette INDICI.

In questo modo è possibile mantenere la tabella ordinata, contemporaneamente, su più campi. Basta creare un indice per ogni campo.

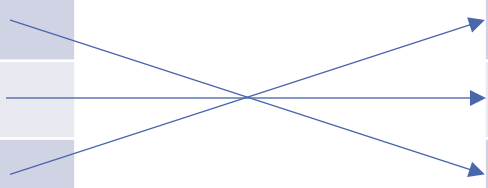
Ogni indice è una struttura dati, memorizzata su disco e gestita dal DBMS, molto più piccola rispetto al database e pertanto è più facile tenerla sempre ordinata.

Un indice contiene principalmente due informazioni:

- Il valore del campo su cui è costruito
- Un riferimento alla posizione del disco nella quale si trova il record completo

Indici

Id	Riferimento		Id	IdAthlete	Nome	Sex	
2			200	93	Jol Marc Abati	M	...
90			90	28	Jan-Erik Aarberg	M	...
200			2	2	A Lamusi	M	...



La ricerca può così avvenire sull'indice ordinato e, solo alla fine si accede alla tabella completa.

In caso di modifica dei dati solamente l'indice deve essere riorganizzato, con risparmio di tempo rispetto alla riorganizzazione dell'intera tabella.

Gli indici possono anche essere realizzati su più campi, per soddisfare esigenze di ricerca che coinvolgono più colonne.

Indici

A fronte di tabelle di grandi dimensioni anche gli indici possono essere molto grandi con conseguenze negative per le performance.

Per limitare il problema gli indici sono realizzati con strutture ad albero (dette B-Tree).

In ogni caso, l'utilizzo degli indici va limitato ai casi in cui è strettamente necessario, nei quali il miglioramento delle performance di ricerca supera il costo di gestione dell'indice.

Triggers

Triggers

I trigger sono speciali store procedure la cui esecuzione avviene in automatico al verificarsi di determinati eventi.

In SQL Server sono possibili tre tipi di trigger:

- DML trigger – eseguiti in risposta ad istruzioni di INSERT, UPDATE e DELETE
- DDL trigger – eseguiti in risposta ad istruzioni di CREATE, DROP e ALTER
- Login triggers – eseguiti in risposta ad eventi di login

Vedremo solamente i primi due

DML triggers

I trigger legati ad operazioni di INSERT, UPDATE e DELETE si creano nel seguente modo:

```
CREATE TRIGGER nomeTrigger  
ON nomeTabella  
AFTER INSERT, UPDATE, DELETE  
AS  
  
BEGIN  
    istruzioni SQL  
  
END
```

DML triggers

Durante l'esecuzione di un DML trigger è possibile far riferimento a due tabelle virtuali create e popolate direttamente da sql server: INSERTED e DELETED.

I valori contenuti dalle due tabelle dipendono dall'istruzione che ha attivato il trigger.

ISTRUZIONE	Tabella INSERTED	Tabella DELETED
INSERT	nuove righe da inserire	vuota
UPDATE	righe modificate DOPO la modifica	righe modificate PRIMA della modifica
DELETE	vuota	righe da cancellare

DML triggers - esempio

```
CREATE TRIGGER backup  
ON AthletesFull  
AFTER UPDATE, DELETE  
AS  
  
BEGIN  
    INSERT INTO AthletesBackup  
    SELECT * FROM deleted  
  
END
```

DML triggers

I trigger legati ad operazioni di CREATE, ALTER e UPDATE si creano nel seguente modo:

```
CREATE TRIGGER nomeTrigger  
ON DATABASE | ALL SERVER  
FOR azione  
AS  
  
BEGIN  
    istruzioni SQL  
END
```

Le azioni possibili sono, per esempio, CREATE_TABLE, ALTER_TABLE...