Il linguaggio SQL

Introduzione

SQL (Structured Query Language) è il linguaggio ormai assunto come **standard** per la trattazione di Data Base (DB) relazionali. La sua stesura è dovuta essenzialmente a E. F. Codd, all'inizio degli anni '70. L'ANSI ha definito le specifiche (SQL ANSI-89), ma le case produttrici spesso propongono dialetti SQL non completamente conformi, sia perché non implementano alcuni costrutti, sia perché includono nuove caratteristiche.

SQL è un linguaggio **non procedurale**. I linguaggi non procedurali si distinguono dai linguaggi imperativi perché permettono di descrivere gli obiettivi da raggiungere, astraendo dagli algoritmi che svolgono il compito effettivo.

Il nome SQL può essere tradotto in italiano come *Linguaggio Strutturato di Interrogazione*; ma oltre all'interrogazione, SQL permette la creazione, manutenzione e modifica di DB.

In generale, nei linguaggi per DB si distinguono le seguenti categorie semantiche:

- Primitive per la definizione dei dati: **DDL** *(Data Definition Language)*. Compiti: creare e definire nuovi DB, campi, indici.
- Primitive per la gestione dei dati: **DML** *(Data Management Language)*. Compiti: creare query che consentono di ordinare, filtrare, estrarre dati dal DB.

Microsoft Jet è un modulo per la gestione di DB relazionali, utilizzato da Microsoft Visual Basic e da Microsoft Access. Microsoft Jet ammette due modelli di gestione dei DB:

- Modello navigazionale, basato su spostamenti tra i record del DB
- Modello relazionale, basato sul linguaggio SQL.

In questa dispensa si fa riferimento al linguaggio SQL in riferimento al modulo di gestione dei DB Microsoft Jet, salvo che non sia esplicitato altrimenti.

Componenti di SQL

Componenti di SQL sono: comandi, proposizioni, operatori, funzioni di aggregazione e predicati. Qui di seguito si elencano le principali parole chiave. Successivamente si introducono altre parole chiave.

Anche in SQL si può mantenere la classificazione dei comandi in DDL e DML.

Comandi DDL

Comando	Descrizione
CREATE	Consente di creare nuove tabelle, campi e indici
DROP	Consente di eliminare tabelle ed indici dal DB
ALTER	Consente di modificare tabelle aggiungendovi campi o modificandone la definizione

Comandi DML

Comando	Descrizione
SELECT	Consente di richiedere i record del DB rispondenti a criteri specifici
INSERT	Consente di caricare gruppi di dati nel DB con una sola operazione
UPDATE	Consente di modificare i valori di particolari record e campi
DELETE	Consente di rimuovere record da una tabella del DB

Proposizioni

Le proposizioni sono condizioni che consentono di definire i dati che si desidera selezionare o gestire.

Proposizione	Descrizione
FROM	Consente di specificare il nome delle tabelle di cui si desidera selezionare i record
WHERE	Consente di specificare i criteri a cui debbono corrispondere i record da selezionare
GROUP BY	Consente di suddividere in gruppi i record selezionati
HAVING	Consente di specificare la condizione a cui deve corrispondere ciascun gruppo
ORDER BY	Consente di ordinare i record selezionati in base all'ordine specificato

Operatori

SQL prevede due tipi di operatori: logici e di confronto.

Gli operatori logici consentono di collegare due espressioni logiche o di negarne una. Sono i già noti:

- AND
- OR
- NOT

Gli operatori di confronto consentono di confrontare il valore relativo di due espressioni per determinare l'azione che verrà eseguita.

Operatore di cfr	Descrizione
<	Minore di
<=	Minore o uguale a
>	Maggiore
>=	Maggiore o uguale
=	Uguale
<>	Diverso
BETWEEN	Consente di specificare un intervallo di valori
LIKE	Utilizzato per criteri di ricerca
IN	Consente di specificare record in DB

Tipicamente gli operatori si usano all'interno di una proposizione WHERE. Esempio:

```
SELECT * FROM Tabella1 WHERE COD=5 AND Importo < 100000
```

Funzioni di aggregazione

Le funzioni di aggregazione vengono utilizzate in una proposizione SELECT ed applicate a più gruppi di record per restituire un valore singolo riferito ad un gruppo di record.

Funz. Di Aggregazione	Descrizione
AVG	Consente di ottenere la media dei valori di un particolare campo
COUNT	Restituisce il numero dei record selezionati
SUM	Restituisce la somma di tutti i valori di un particolare campo
MAX	Restituisce il valore massimo del campo specificato
MIN	Restituisce il valore minimo del campo specificato

Gli identificatori degli oggetti possono contenere spazi. In questo caso è obbligatorio racchiudere l'identificatore tra parentesi quadre[ ]; negli altri casi è facoltativo.

Operazioni DDL

Il linguaggio di definizione dei dati permette di creare tabelle ed indici, di modificare le tabelle aggiungendovi o rimuovendo colonne ed indici.

Nella pratica si preferisce di solito usare per questo compito altri metodi più diretti (come Access, in casa Microsoft, che grazie alla modalità visuale, semplifica grandemente il compito e riduce gli errori formali). Di seguito si indicano, per ciascuna delle funzioni, esempi di uso del SQL.

Creazione di tabelle

Creare una tabella di nome *Persone*, con due colonne di testo, *Cognome*, di 30 caratteri e *Nome* di 20 caratteri:

```
CREATE TABLE Persone ([Cognome] TEXT (30), [Nome] TEXT (20));
```

Aggiunta e rimozione di colonne

Aggiungere alla tabella *Persone* una colonna *Indirizzo* di 50 caratteri:

```
ALTER TABLE Persone ADD COLUMN Indirizzo TEXT(50);
```

Eliminare dalla tabella *Persone* la colonna *Nome*:

```
ALTER TABLE Persone DROP COLUMN Nome;
```

Modificare la colonna *Indirizzo* portandola a 40 caratteri. Prima bisogna rimuoverla e successivamente aggiungere la colonna della dimensione scelta:

```
ALTER TABLE Persone DROP COLUMN Indirizzo;
ALTER TABLE Persone ADD Indirizzo TEXT(40);
```

Creazione ed eliminazione di indici

1° metodo: Creare l'indice all'atto della creazione della tabella

Si usa la parola chiave CONSTRAINT, che deve comparire all'inizio della definizione di un indice. CONSTRAINT garantisce il mantenimento dell'integrità referenziale.

Creazione di una tabella con indice su una sola colonna (*DataNascita*)

```
CREATE TABLE Persone ([Cognome] TEXT (30), [Nome] TEXT(20), _
[DataNascita] DATETIME CONSTRAINT IndicePersone PRIMARY);
```

Creazione di una tabella con indice su tre campi:

```
CREATE TABLE Persone ([Cognome] TEXT (30), [Nome] TEXT(20), _
[DataNascita] DATETIME, CONSTRAINT IndicePersone UNIQUE _
([Cognome], [Nome], [DataNascita]);
```

2° metodo: con il comando CREATE INDEX

Se si desidera creare un indice in un momento successivo alla creazione, si usa questa modalità.

Creazione di un indice sul campo *DataNascita*:

```
CREATE UNIQUE INDEX MioIndice ON Persone ([DataNascita]);
```

Con la proposizione facoltativa WITH si possono aggiungere ulteriori condizioni:

Condizione	Descrizione
PRIMARY	Identifica la colonna primaria dell'indice
DISALLOW NULL	La colonna non può essere vuota

IGNORE NULL	Record non indicizzato se la colonna è vuota

Creare un indice impedendo che sia aggiunta una riga in cui il *Cognome* sia nullo:

```
CREATE UNIQUE INDEX MioIndice ON Persone ([Cognome]) WITH
DISALLOW NULL;
```

3° metodo: con il comando ALTER TABLE

All'interno del comando di modifica di una tabella si indica l'indice con la parola CONSTRAINT.

Indice su *Cognome:*

```
ALTER TABLE Persone ADD CONSTRAINT MioIndice PRIMARY Cognome;
```

Indice su due colonne:

```
ALTER TABLE Persone ADD CONSTRAINT MioIndice UNIQUE
([Cognome], [Nome]);
```

Integrità referenziale legata alla parola CONSTRAINT

La parola CONSTRAINT, come si è visto precedentemente, è utilizzata per creare o eliminare indici.

Inoltre permette di definire chiavi primarie ed esterne, definire relazioni e mantenere l'integrità referenziale.

La sintassi per la creazione di un indice a campo singolo è:

```
CONSTRAINT Nome {PRIMARY KEY | UNIQUE | REFERENCES
TabellaEsterna [(Campoesterno1, Campoesterno2]}
```

La sintassi per creare un indice a campi multipli è:

```
CONSTRAINT Nome {PRIMARY KEY (Primaria1[,Primaria2[,…]])|
UNIQUE (Univoca1[,Univoca2[,…]])| FOREIGN KEY
(Rif1[,Rif2[,…]])REFERENCES TabellaEsterna
[(CampoEsterno1[,CampoEsterno2[,…]])]}
```

Descrizione degli argomenti:

Argomento	Descrizione
Nome	Nome dell'indice che verrà creato
PrimariaX	Nomi dei campi designati come chiave primaria
UnivocaX	Nomi dei campi designati come chiave univoca
RifX	Nomi dei campi esterni che fanno riferimento a campi contenuti in un'altra tabella
TabellaEsterna	Nome della tabella esterna contenente i campi specificati da *CampoEsternoX*
CampoEsternoX	Nomi dei campi di *TabellaEsterna* specificati da *RifX*

Tipi di indice di un campo:

Tipo Indice	Descrizione
UNIQUE	Identifica un campo (o un insieme di campi) come chiave univoca, cioè due righe della tabella non possono avere lo stesso valore su questo campo (o sull'insieme dei campi, presi nel loro complesso)
PRIMARY KEY	Identifica un campo come chiave primaria. In una tabella può esservi una sola chiave primaria. Tutti i valori della chiave primaria debbono essere univoci.
FOREIGN KEY	Identifica un campo come chiave esterna.

Esempio: Si vuol aggiungere un indice di nome *MioIndice* alla tabella *Titoli*, creando una relazione tra il campo *PubID* della tabella Titoli (chiave esterna - più titoli possono avere la stessa casa editrice) e il campo *PubID* della tabella *Editori* (chiave interna - ogni editore ha un codice primario):

```
ALTER TABLE Titoli ADD CONSTRAINT MioIndice FOREIGN KEY
(PubID) REFERENCES Editori (PubID);
```

Operazioni DML

La maggior parte delle query utilizzano il comando SELECT, che consente di recuperare un set di record (Recordset) da un DB e di memorizzarlo in un nuovo Recordset. Il nuovo Recordset può essere gestito visualizzandolo, modificandolo, e/o eliminando determinati record, o producendo e realizzando report.

Con la SELECT si realizzano le operazioni dell'algebra relazionale di: selezione, proiezione, prodotto cartesiano e join naturale.

La sintassi generale è:

```
SELECT ElencoCampi
FROM NomiTabelle IN NomeDB
WHERE CondizioniRicerca
GROUP BY ElencoCampi
HAVING CriteriRaggruppamento
ORDER BY ElencoCampi
WITH OWNERACCESS OPTION;
```

Un comando SELECT deve essere obbligatoriamente seguito da una proposizione FROM; tutte le altre parole chiave sono facoltative.

Query di base

Origine dei dati e proposizione FROM - Proiezione

La proposizione FROM indica la tabella di provenienza dei record.

Restituire tutte le colonne di una tabella:

```
SELECT *
FROM tabella;
```

L'asterisco indica l'insieme di tutte le colonne.

Restituire alcune colonne da una tabella:

```
SELECT [nomecolonna1[,nomecolonna2[,…]]]
FROM tabella;
```

DB esterni e proposizione IN

E' possibile far riferimento ad una tabella proveniente da un solo DB esterno, con la proposizione IN.

Se il DB esterno è diverso da Jet, è necessario specificare, oltre al percorso, anche il nome del DBMS (dBASE, FoxPro, Paradox, …). Esempi equivalenti:

```
SELECT *
FROM Tabella
IN "" [dBASE IV; DATABASE=C:\DBASE\DATI\VENDITE;];
```

oppure:

```
SELECT *
FROM Tabella
```

```
IN "C:\DBASE\DATI\VENDITE" "dBASE IV;";
```

Alias e proposizione AS

E' possibile assegnare ad una colonna un nome diverso tramite la proposizione AS. Esempio:

```
SELECT [nomecolonna] AS altronome
FROM tabella;
```

Proposizione WHERE - Selezione

Estrarre tutte le righe della tabella *Persone* relative a persone di sesso femminile

```
SELECT *
FROM Persone
WHERE Sesso = "F";
```

Filtri, ordinamenti e raggruppamenti

Predicato DISTINCT

Si usa per evitare che siano presentate righe uguali. Per esempio, se ci sono più persone di cognome Bianchi, per averne una sola si usa:

```
SELECT DISTINCT [Cognome]
FROM Persone;
```

Predicato ORDER BY

Ordina i record in base al campo indicato. Per default intende ordinamento ascendente (ASC); per ordinare in modo discendente si usa la parola DESC. Esempio:

```
SELECT Cognome, Nome
FROM Persone
ORDER BY Eta;
```

Si può richiedere un ordinamento primariamente su un campo e, in subordine, in altri campi. Per esempio, ordinare in modo decrescente per retribuzione e, in subordine, sul cognome in ordine crescente:

```
SELECT Cognome, Nome
FROM Persone
ORDER BY Retribuzione DESC,[Cognome];
```

Predicato TOP

Seguito da un numero positivo *n*, seleziona le prime *n* righe. Ha senso in congiunzione con ORDER BY. Esempio per selezionare le 20 persone più giovani:

```
SELECT TOP 20 Cognome, Nome
FROM Persone
ORDER BY Eta;
```

Con l'uso della parola chiave PERCENT si ottiene l'estrazione di una percentuale di righe sul totale selezionato. Esempio per ottenere il 10 % dei migliori studenti diplomati nel 99:

```
SELECT TOP 10 PERCENT Cognome, Nome
FROM Studenti
WHERE [AnnoDiploma]=1999
ORDER BY [Punteggio] DESC;
```

Proposizione GROUP BY

Consente di raggruppare in un unico record i record contenenti valori identici nell'elenco dei campi specificati. Di solito è usato in congiunzione con una funzione di aggregazione come SUM o COUNT. Esempio: Calcolo del totale delle ore di straordinario per dipendente:

```
SELECT Cognome, SUM(OreStraordinario)
FROM LavoriMensili
GROUP BY Cognome;
```

La proposizione WHERE, quando usata, seleziona i record prima del raggruppamento.

La proposizione HAVING (v. sotto) filtra i record dopo il raggruppamento.

Tutti i campi elencati nell'istruzione SELECT debbono o essere inclusi nella proposizione GROUP BY o inseriti come argomenti di una funzione di aggregazione.

Proposizione HAVING

Usato in congiunzione con GROUP BY, specifica quali raggruppamenti di record verranno visualizzati. Esempio:

```
SELECT Cognome, Mansione, SUM(OreStraordinario)
FROM LavoriMensili
GROUP BY Cognome
HAVING Mansione = "Fresatore";
```

Proposizione WITH OWNERACCESS OPTION

Posta alla fine di una query, in ambiente multiutente, permette all'utente di visualizzare i dati anche delle tabelle delle quali l'utente non ha il permesso di accesso.

Inoltre, se l'utente non ha il permesso di creare tabelle o accodare record, l'autorizzazione può essere occasionalmente concessa grazie a tale istruzione.

Manipolazione di tabelle

Query per la creazione di tabelle: proposizione INTO

Se invece che un Recordset si desidera ottenere una nuova tabella, si usa la proposizione INTO. Esempio: Creare una nuova tabella da Persone:

```
SELECT *
INTO NuovePersone
FROM Persone
```

Usato per archiviare record, eseguire copie di backup, copie da esportare in altro DB, ecc.

Nella nuova tabella vengono trasferiti solo le proprietà di tipo e di lunghezza dei campi, oltre i dati.

Query per l'eliminazione di record: comando DELETE

Si possono eliminare dalla tabella indicata nella proposizione FROM, i record selezionati dalla proposizione WHERE. Sintassi:

```
DELETE *
FROM tabella
WHERE criteri
```

Eliminando tutte le righe della tabella, rimane comunque la sua struttura.

È possibile utilizzare DELETE per rimuovere i record da tabelle che si trovano in una relazione uno-a-molti con altre tabelle, indicando l'elenco dei nomi nella proposizione FROM. Le operazioni di eliminazione a catena causano l'eliminazione dei record di tabelle che rappresentano il lato

"molti" della relazione quando viene eliminato nella query il corrispondente record che rappresenta il lato "uno" della relazione. Nella relazione tra la tabella Clienti e la tabella Ordini, ad esempio, la tabella Clienti rappresenta il lato "uno" e la tabella Ordini rappresenta il lato "molti" della relazione. Se è specificata l'opzione di eliminazione a catena, l'eliminazione di un record dalla tabella Clienti dà come risultato l'eliminazione dei record corrispondenti nella tabella Ordini.

Query per l'eliminazione di una tabella o un indice: comando DROP

Per eliminare l'intera tabella, compresa la sua struttura, si usa il comando DROP. Esempio:

```
DROP TABLE Impiegati;
```

Il comando DROP permette di cancellare anche solo un indice di una tabella. Esempio:

```
DROP INDEX NuovoIndice ON Impiegati;
```

Query di accodamento: comando INSERT INTO

Accodamento di una singola riga:

```
INSERT INTO destinazione [(campo1[,campo2[,…]])]
VALUES (valore1[,valore2[,…]])
```

inserisce in fondo alla tabella *destinazione* un record con *valore1* nel *campo1*, *valore2* nel *campo2*, ecc. Se non vengono specificati tutti i campi, nelle colonne mancanti viene inserito il valore predefinito, se c'è, oppure il valore NULL.
Se l'elenco dei campi viene omesso, è necessario assegnare tutti i valori per tutti i campi della riga; altrimenti la riga non viene aggiunta.
Accodamento di più righe prelevate da una tabella o da una query:

```
INSERT INTO destinazione [IN DBEsterno]
SELECT [origine.campo1[,campo2[,…]])]
FROM espressionetabella
```

La proposizione IN permette di accodare righe in un DB esterno.
Sia l'origine sia la destinazione possono essere una tabella o una query. Se la destinazione è una query, Microsoft Jet aggiunge un set di record a tutte le tabelle specificate nella query.
L'origine non subisce alterazioni.
Se la tabella di destinazione contiene un campo che è chiave primaria, non vengono aggiunte tutte quelle righe che hanno un valore duplicato o un valore NULL in quel campo; pertanto è necessario operare gli opportuni controlli prima di effettuare l'accodamento.

Query di aggiornamento: comando UPDATE

Modificare i valori dei campi indicati in base ai criteri specificati:

```
UPDATE tabella
SET nuovovalore
WHERE criteri
```

E' possibile modificare contemporaneamente più campi. Esempio: il campo *Importo* viene aumentato del 10% il campo *Iva* viene posto a 15 per tutte le righe di *Categoria* XYZ:

```
UPDATE Movimenti
SET Importo=Importo*1.1,
    Iva=15
WHERE Categoria = "XYZ";
```

Il comando UPDATE non genera nessuna tabella.

Operazioni su più tabelle: JOIN

La proposizione FROM può essere seguita da più nomi di tabelle. L'ordine con cui esse appaiono non è rilevante.

Se il nome di un campo è incluso in più tabelle, per evitare ambiguità, deve essere indicato con la notazione punto ".", cioè: `[nome della tabella].[nome del campo]`

Se sono specificate più tabelle, in assenza di condizioni WHERE o JOIN, la query genera il prodotto cartesiano delle tabelle.

Il comando JOIN permette di ottenere una tabella contenente informazioni provenienti da più tabelle. La tabella risultante viene costruita in base alle relazioni istituite tra le tabelle origine. Il caso tipico è quando una tabella contiene una chiave esterna in relazione con la chiave principale di un'altra tabella. Sono previsti tre tipi di JOIN:

Tipo di Join	Descrizione
INNER JOIN	Nella tabella risultante vengono inclusi i record di entrambe le tabelle solo se il valore del campo della prima tabella corrisponde al valore del campo della seconda tabella
LEFT OUTER JOIN	Nella tabella risultante vengono inclusi tutti i record della prima tabella, e solo i record della seconda tabella in cui i campi specificati si incontrano
RIGHT OUTER JOIN	Nella tabella risultante vengono inclusi tutti i record della seconda tabella, e solo i record della prima tabella in cui i campi specificati si incontrano

Gli argomenti del comando SELECT possono essere una qualunque combinazione dei campi delle due tabelle correlate.

Join interno: comando INNER JOIN

Il Join interno è detto anche Join naturale. La sintassi è:

```
FROM tabella1 INNER JOIN tabella2
ON tabella1.campo1 = tabella2.campo2
```

Il Join interno è commutativo, cioè T1 INNER JOIN T2 è equivalente a T2 INNER JOIN T1. E' possibile unire campi numerici di qualsiasi tipo, anche se contengono campi diversi. Esempio:

```
SELECT [NomeCategoria],[NomeProdotto]
FROM Categorie INNER JOIN Prodotti
ON Categorie.[CodCategoria]=Prodotti.[CodCategoria];
```

Join esterni: comandi LEFT JOIN e RIGHT JOIN

Il Join esterno non è commutativo e l'ordine con cui si indicano le tabelle è significativo. Esempio: si abbia una tabella *Dipendenti* e una tabella *Reparti*. Nella tabella *Dipendenti* è presente la chiave esterna *CodReparto.* Non tutti i dipendenti sono assegnati ad un reparto, e non tutti i reparti hanno dipendenti.

Creare una tabella che elenchi tutti i reparti; se un reparto ha dipendenti, elencare tutti i dipendenti:

```
SELECT Reparti.Nome, Dipendenti.Nome
FROM Reparti LEFT JOIN Dipendenti
ON Reparti.CodReparto = Dipendenti.CodReparto;
```

Creare una tabella che elenchi tutti i dipendenti; per i dipendenti che lavorano in un reparto, indicare il nome del reparto:

```
SELECT Dipendenti.Nome, Reparti.Nome
FROM Reparti RIGHT JOIN Dipendenti
ON Reparti.CodReparto = Dipendenti.CodReparto;
```

Esclusione di righe duplicate: predicato DISTINCTROW

Nella tabella risultante da un Join possono comparire più righe uguali. Per evitarlo usare DISTINCTROW. Esempio:

```
SELECT DISTINCTROW [Nome]
FROM Clienti INNER JOIN Ordini
ON Clienti.CodCli=Ordini.CodCli
ORDER BY Nome;
```

Omettendo DISTINCTROW sarebbero state presenti per lo stesso cliente tante righe quanti erano i suoi ordini.

Join nidificati

Il join può essere generalizzato a più tabelle. Esempio con 3 tabelle:

```
SELECT campi
FROM tabella1 INNER JOIN
(tabella2 INNER JOIN tabella3
ON tabella3.campo3 = tabella2.campo2)
ON tabella2.campo2 = tabella1.campo1;
```

Query complesse

Consentono di comporre più strutture espressive utilizzando comandi SELECT dentro comandi SELECT o con più proposizioni FROM.

Sottoquery

Le condizioni di un comando SELECT, SELECT INTO, INSERT INTO, DELETE o UPDATE possono aver bisogno di operare su una tabella risultante da una SELECT interna o nidificata, detta appunto *sottoquery*.

Una sottoquery deve necessariamente produrre una sola colonna.

E' possibile usare una sottoquery dentro l'elenco dei campi di una SELECT, o di una proposizione WHERE o HAVING.

Per creare una sottoquery si possono usare tre forme sintattiche:

- *confronto* [**ANY** | **SOME** | **ALL**] (*istruzionesql*)
- *espressione* [**NOT**] **IN** (*istruzionesql*)
- [**NOT**] **EXISTS** (*istruzionesql*)

1ª forma sintattica: predicati ANY, SOME, ALL

Si usa quando nella query principale si pone la necessità di confronto con qualcuno o tutti gli elementi della sottoquery.

Esempio 1: Elencare tutti i prodotti con prezzo unitario maggiore di quello di qualunque prodotto venduto con uno sconto non minore del 20%:

```
SELECT * FROM Prodotti
WHERE [PrezzoUnitario] > ANY
     (SELECT [Prezzo] FROM Ordini
      WHERE [Sconto] >= 20);
```

I predicati ANY e SOME sono sinonimi.

Esempio 2: Elencare tutti i prodotti con prezzo unitario maggiore di quello di tutti i prodotti venduti con uno sconto non minore del 20%:

```
SELECT * FROM Prodotti
WHERE [PrezzoUnitario] > ALL
     (SELECT [Prezzo] FROM Ordini
      WHERE [Sconto] >= 20);
```

Il predicato ALL è più restrittivo di ANY / SOME.

Siano assegnate le seguenti tabelle:

Prodotti

CodProd	PrezzoUnitario
1	100
2	150
3	90

Ordini

CodOrd	CodProd	Prezzo	Sconto
1	1	100	20
1	2	150	15
2	1	150	0
2	2	90	30
3	3	100	10

Risultati delle query e sottoquery:

Entrambe le sottoquery:

Prezzo
100
90

Esempio 1: ANY

CodProd	PrezzoUnitario
1	100
2	150

Esempio2: ALL

CodProd	PrezzoUnitario
2	150

2ª forma sintattica: proposizione IN

Elenca nella query principale solo i record con valore uguale a quello contenuto nella sottoquery.

Esempio: elencare tutti i prodotti venduti con sconto non inferiore al 20%:

```
SELECT * FROM Prodotti
WHERE [CodProd] IN
      (SELECT [CodProd] FROM Ordini
       WHERE [Sconto] >= 20);
```

L'operatore NOT evidentemente nega la condizione.

Considerando le tabelle precedenti, nell'esempio attuale si ha:

Sottoquery

CodProd
1
2

Query principale

CodProd	PrezzoUnitario
1	100
2	150

3ª forma sintattica: proposizione EXISTS

La query principale opera se la sottoquery restituisce record.

Esempio: selezionare il nome di tutti gli impiegati che hanno ricevuto almeno un ordine. Tale operazione può essere eseguita anche tramite INNER JOIN.

```
SELECT Nome, Cognome FROM Impiegati
WHERE EXISTS
      (SELECT IDOrdine FROM Ordini
       WHERE Ordini.IDImpiegato = Impiegati.IDImpiegato);
```

Query a campi incrociati: comando TRANSFORM

Una query a campi incrociati estrae valori di riepilogo, come somme, conteggi e medie, da un campo di una tabella e li raggruppa visualizzandoli in un set di dati elencati sul lato sinistro del foglio di dati e in un altro set di dati elencato nella parte superiore del foglio di dati.

Esempio: i risultati di una query normale, a sinistra sono molto più dispersivi di una query a campi incrociati, a destra:

Nome	Lavoro	Subtotale
Rossi	Normale	24

	Lavoro		
Nome	Normale	Straordinario	Notturno

	Straordinario	5
	Notturno	12
Verdi	Normale	30
	Straordinario	2
	Notturno	0

Rossi	24	5	12
Verdi	30	2	0

Le query a campi incrociati calcolano una somma, una media, un conteggio o altri tipi di totali per i dati raggruppati in base a due tipi di informazione: uno sul lato sinistro del foglio dati, l'altro sul lato superiore. Per creare query a campi incrociati si usa il comando TRANSFORM, la cui sintassi è:

```
TRANSFORM funzioneaggr
istruzioneselect
PIVOT campopivot [IN (valore1[,valore2[,…]]]
```

Argomento	Descrizione
Funzioneaggr	Funzione di aggregazione SQL eseguita sui dati selezionati
Istruzioneselect	Istruzione SELECT
Campopivot	Il campo o l'espressione che si desidera utilizzare per creare le intestazioni di colonna nel set di risultati della query
valore1,valore2	Valori fissi utilizzati per creare le intestazioni di colonna; quando i dati vengono riepilogati, è necessario selezionare i valori come intestazioni di colonna dai campi o espressioni specificati

Esempio: creare una query a campi incrociati che mostra le vendite registrate ogni mese di un anno specificato dall'utente. I mesi vengono restituiti da sinistra a destra (pivot) come colonne e i nomi dei prodotti vengono restituiti dall'alto verso il basso come righe.

```
PARAMETERS [Vendite per quale anno?] LONG;
TRANSFORM SUM([Dettagli ordini].Quantità * ([Dettagli
Ordine].PrezzoUnitario - ([Dettagli ordini].Sconto / 100) *
[Dettagli ordini].PrezzoUnitario)) AS Vendite
SELECT NomeProdotto FROM Ordini
INNER JOIN (Prodotti INNER JOIN [Dettagli ordini] ON
Prodotti.IDProdotto = [Dettagli ordini].IDProdotto)
ON Ordini.IDOrdine = [Dettagli ordini].IDOrdine
WHERE DatePart("aaaa", DataOrdine) = [Vendite per quale anno?]
GROUP BY NomeProdotto
ORDER BY NomeProdotto
PIVOT DatePart("m", DataOrdine);
```

Query di unione: comando UNION

Unisce i risultati di due o più tabelle o query indipendenti. Tutte le query in un'operazione UNION devono avere lo stesso numero di campi; questi tuttavia, non devono essere delle stesse dimensioni o dello stesso tipo di dati. Sintassi:

```
[TABLE] query1 UNION [ALL] [TABLE] query2 [UNION [ALL] [TABLE]
queryn [ ... ]]
```

query1-n sono istruzione SELECT, nome di una query memorizzata o nome di una tabella memorizzata preceduto dalla parola chiave TABLE.

Esempio 1: recuperare i nomi e gli ID di tutti i fornitori e i clienti. Tale unione presuppone che ogni tabella contenga lo stesso numero di colonne.

```
TABLE Clienti UNION TABLE Fornitori;
```

In base all'impostazione predefinita, quando si utilizza l'operazione UNION non vengono restituiti record duplicati; tuttavia, è possibile includere il predicato **ALL** per assicurare che vengano restituiti tutti i record. In questo modo l'esecuzione della query è più rapida.
È possibile unire i risultati di due o più query, tabelle e istruzioni SELECT in una combinazione qualsiasi in un'unica operazione UNION.
Esempio 2: unire una tabella esistente denominata *Nuovi account* e un'istruzione SELECT:

```
TABLE [Nuovi account] UNION ALL
SELECT *
FROM Clienti
WHERE AmmontareOrdine > 1000;
```

Esempio 3: recuperare i nomi e le città di tutti i fornitori e i clienti in Brasile:

```
SELECT NomeSocietà, Città FROM Fornitori
WHERE Paese = 'Brasile'
UNION SELECT NomeSocietà, Città FROM Clienti
WHERE Paese = 'Brasile';
```

Bibliografia:

- Microsoft Visual Basic - Funzioni professionali - Microsoft Corporation 1995
- Manuale in linea di Microsoft Access 97