Il Linguaggio

SQL

Teoria ed esempi

A cura del Prof. Enea Ferri

Cos'è SQL?

SQL , ovvero Structured Query Language (Linguaggio di interrogazione strutturato) è un linguaggio standard per poter accedere e manipolare i dati memorizzati in un Data Base. Consente:

- La definizione delle strutture dati e le relazioni tra le tabelle del Data base
- L'esecuzione di query su un database per reperire le informazioni richieste
- L'inserimento, aggiornamento e cancellazione dei record in un database

I DBMS, quali MS Access, DB4, Informix, MS SQL Server, Oracle, Sybase, FoxPro, etc. usano diverse versioni di SQL, anche se tutte devono supportare la maggior parte dei comandi fondamentali come SELECT, UPDATE, DELETE, INSERT, WHERE, e altri.
Distinguiamo:

DDL	Data Definition Language	Definizione struttura delle tabelle
DML	Data Manipulation Language	Modifica dati contenuti nelle tabelle
QL	Query Language	Interrogazione Data Base
DMCL	Device Media Control Language	Controllo unità di memorizzazione Data Base

Data Definition Language (DDL)

Il Data Definition Language (DDL) è una parte di SQL che permette di creare, modificare o eliminare le tabelle dalla struttura di un database. Permette anche di definire gli indici (delle chiavi), necessari per velocizzare la ricerca di un record, specificare collegamenti tra le tabelle ed imporre vincoli tra le tabelle del database.

I comandi più importanti sono:

CREATE TABLE	crea una nuova tabella
ALTER TABLE	modifica una tabella
DROP TABLE	cancella una tabella
CREATE INDEX	crea un indice (una chiave di ricerca)
DROP INDEX	elimina un indice

SQL CREATE e DROP

Per Creare un Database:

```
CREATE DATABASE Nome_Database
```

Per creare una tabella di un database:

```
CREATE TABLE Nome_Tabella
(
Nome_Colonna1 Tipo_di_Dato,
Nome_Colonna2 Tipo_di_Dato,
.......
)
```

L'esempio mostra come creare una tabella chiamata "TbPersone", con cinque colonne. Le colonne saranno: "Id_Persona","Cognome", "Nome", "Indirizzo", ed "Età":

```
CREATE TABLE TbPersone
(
Id_Persona COUNTER PRIMARY KEY,
Cognome char(20),
Nome char(15),
Indirizzo char(30),
Età Shortint
);
```

Il "tipo di dato" specifica quale tipo di dato la colonna può contenere. Per il campo ID_Persona il tipo è Contatore gestito come chiave primaria artificiale.

Creare Indici

Gli indici vengono creati in una tabella esistente per localizzare le righe in modo più efficiente.
E' possibile creare un indice su una o più colonne di una tabella, e ad ogni indice si da un nome.
L'utente non può vedere gli indici, questi ultimi sono usati per rendere più veloci le query.

Nota: Aggiornare una tabella che contiene degli indici prende più tempo rispetto ad una che non ne ha, questo perche occorre aggiornare anche gli indici.

Unique Index

Crea un indice univoco nella tabella. Un indice univoco vuol dire che due righe non possono avere la stesso valore del campo usato per creare l'indice.

CREATE UNIQUE INDEX Nome_Indice
ON Nome_Tabella (Nome_Colonna)
"Nome_Colonna" specifica la colonna che vogliamo indicizzare.

Simple Index

Crea un indice semplice, con la parola chiave UNIQUE omessa, valori duplicati sono ammessi.
CREATE INDEX Nome_Indice
ON Nome_Tabella (Nome_Colonna);
"Nome_Colonna" specifica la colonna che vogliamo indicizzare

Questo esempio crea un indice semplice, chiamato "IndicePersone", sul cognome della tabella Persone:
CREATE INDEX IndicePersone
ON TbPersone (Cognome)

Se vogliamo indicizzare i valori in ordine discendente, dobbiamo utilizzare la parola chiave DESC dopo il nome della colonna:
CREATE INDEX IndicePersone
ON TbPersone (Cognome DESC);

Se vogliamo indicizzare più di una colonna, possiamo elencare i nomi delle colonne tra parentesi, separandole con delle virgole:
CREATE INDEX IndicePersone
ON TbPersone (Cognome, Nome);

Drop Index

Possiamo eliminare un indice creato su una tabella tramite l'istruzione DROP.
DROP INDEX Nome_Indice ON NomeTabella

Delete a Database or Table

Per eliminare un database:
DROP DATABASE Nome_Database
Per eliminare una tabella (vengono eliminati anche struttura, attributi ed indici):
DROP TABLE Nome_Tabella

SQL ALTER TABLE

Modifica la struttura della tabella, consentendo aggiunte di nuovi attributi o eliminazione di attributi.
ALTER TABLE Nome_Tabella
ADD Nome_Colonna TipoDiDato

oppure
ALTER TABLE Nome_Tabella
DROP Nome_Colonna
Nota: Alcuni DBMS non permettono di fare il DROP di una colonna..

Esempio: Per aggiungere una colonna chiamata "Città" nella tabella "TbPersone":
ALTER TABLE TbPersone ADD Città Char(30);

Per eliminare la colonna "Indirizzo" nella tabella "TbPersone":
ALTER TABLE TbPersone DROP Indirizzo;

Data Manipulation Language (DML)

Data Manipulation Language include i comandi per inserire, aggiornare e cancellare un record e sono:

UPDATE – aggiorna i dati in una tabella

DELETE – elimina I dati da una tabella

INSERT INTO – inserisce dati in una tabella

INSERT INTO

Sintassi
INSERT INTO Nome_Tabella
(colonna1, colonna 2,...)
VALUES
(valore1, valore2, ...)

Inserire una nuova riga

INSERT INTO TbPersone
(Cognome, Nome, Indirizzo, Età, Città)
VALUES
 (‘De Giorgi', 'Alfredo', 'Via dei Mille 23', 45, ‘Bari’);
Nota: il campo Id_Persona non è stato considerato, in quanto è di tipo autoincremento, gestito in modo automatico dal DBMS

UPDATE

Sintassi
UPDATE Nome_Tabella
SET Nome_Colonna = Nuovo_Valore
WHERE Nome_Colonna = Un_Valore
Aggiornare una colonna in una riga

Vogliamo aggiungere il nome “Angelo” alla persona con cognome "De Ceglie":
UPDATE TbPersone
SET Nome = 'Angelo'
WHERE Cognome = 'De Ceglie';

Aggiornare più colonne in una riga

Vogliamo cambiare l'indirizzo ed il nome della città:
UPDATE TbPersone
SET Indirizzo = 'Abruzzi 12', Età=47
WHERE Cognome = 'De Ceglie';

DELETE

Sintassi
DELETE
FROM Nome_Tabella
WHERE Nome_Colonna = Un_Valore

Eliminare una riga

Per eliminare Tizio:
DELETE
FROM TbPersone
WHERE Cognome = 'Tizio';

Eliminare tutte le righe

E' possible eliminare tutte le righe di una tabella. Vengono comunque mantenuti attributi, struttura ed indici:
DELETE FROM Nome_Tabella; Oppure DELETE * FROM Nome_Tabella;

QL (Query Language)

Una query è un comando da usarsi per recuperare record all'interno di un database. Usando le query è possibile estrarre dati da uno o più campi di una o più tabelle. I dati recuperati possono essere sottoposti a vincoli, i cosiddetti criteri, che servono a limitare il numero di dati recuperati.
Select: con tale comando possiamo chiedere al RDBMS di inviarci dei dati che soddisfino dei nostri criteri.

SQL SELECT

L'istruzione Select è usata per selezionare dei dati da una tabelle. Il risultato sarà inserito in una tabella chiamata result-set, eventualmente utilizzabile per altre elaborazioni.

SELECT Nome/Nomi_Colonna
FROM Nome_Tabella;

Selezionare alcune colonne

Per selezionare le colonne chiamate "Cognome" e "Nome", utilizziamo un'istruzione SELECT come la seguente:

SELECT Cognome,Nome
FROM TbPersone;

Selezionare tutte le colonne

Per selezionare tutte le colonne della tabella "Persone", possiamo utilizzare il simbolo * al posto dei nomi delle colonne:

SELECT * FROM TbPersone;

SELECT DISTINCT

La parola chiave DISTINCT è utilizzata per restituire soltanto dei valori distinti.
L'istruzione SELECT fornisce informazioni circa il contenuto delle tabelle. Ma se volessimo selezionare soltanto elementi distinti?
In SQL, occorre aggiungere la parola chiave DISTINCT all'istruzione SELECT:

SELECT DISTINCT Nome/Nomi_Colonna
FROM Nome_Tabella

Per selezionare TUTTI I valori della colonna chiamata "PROV" nella tabella TbComuni(Id_Comune, Des_Comune, PROV, CAP, COD_CF), utilizziamo un'istruzione SELECT come questa:

```
SELECT    PROV
FROM     TbComuni;
```

Per selezionare soltanto i valori diversi dalla colonna "PROV" dobbiamo utilizzare l'istruzione SELECT DISTINCT come nell'esempio:

```
SELECT DISTINCT    PROV
FROM    TbComuni;
```

La clausola WHERE

La clausola WHERE è utilizzata per specificare un criterio di selezione.

```
SELECT  Colonna
FROM  tabella
WHERE  colonna operatore valore
```

Possiamo utilizzare i seguenti operatori:

Operatore	Descrizione
=	Uguale
<>	Diverso
>	Maggiore di
<	Minore di
>=	Maggiore o uguale
<=	Minore o uguale
BETWEEN	In un intervallo
LIKE	Ricerca un pattern

Nota: In alcune versioni di SQL, l'operatore <> è scritto come !=

Per selezionare le persone che vivono nella città di "Taranto", aggiungiamo la clausola WHERE all'istruzione SELECT:

```
SELECT  *
FROM  TbPersone
WHERE  Città = 'Taranto';
```

Notare che si è utilizzato l'apice nel valore condizionale 'Taranto'.
SQL utilizza le virgolette per il testo, mentre non occorre farlo per valori numerici.

Per il testo:

```
SELECT * FROM TbPersone
WHERE Nome = 'Giorgio';
```

Per valori numerici:

```
SELECT * FROM TbPersone
WHERE Anno > 1965;
```

La condizione LIKE

La condizione LIKE è usata per specificare la ricerca in una colonna di un pattern (modello specificato, contenente o meno caratteri jolly).
I Caratteri speciali utilizzati nelle query :
_ (Underscore) Serve per indicare la condizione di presenza caratteri in determinate posizioni
% (Percento) Serve per indicare che prima o dopo alcuni caratteri può esserci qualunque cosa

Sintassi

SELECT colonna
FROM tabella
WHERE colonna LIKE pattern

Utilizzare LIKE

La seguente istruzione SQL ritorna i dati delle persone il cui Cognome ha come 4° e 5° carattere le lettere MA. Prima di essi devono esserci 3 caratteri qualsiasi. Dopo MA può esserci nulla o qualunque carattere.
SELECT *
FROM TbPersone
WHERE Cognome LIKE '___MA%';
Dopo l'esecuzione di questo comando verranno visualizzati i dati delle persone con cognome:
AleMAnno
BelMAn
CalMA

La seguente istruzione SQL ritorna le persone con il nome che inizia con una 'O':
SELECT * FROM Persone
WHERE Nome LIKE 'O%';

La seguente istruzione SQL ritorna le persone con il nome che finisce con una 'a':
SELECT * FROM Persone
WHERE Nome LIKE '%a';

La seguente istruzione SQL ritorna le persone con il nome che contiene il pattern 'la':
SELECT * FROM Persone
WHERE Nome LIKE '%la%';

SQL ORDER BY

Serve per ordinare I risultati di una query

Per visualizzare i prodotti di magazzino con relativo prezzo di vendita dalla tabella TbProdotti in ordine alfabetico:
SELECT Des_Prodotto, Prezzo
FROM TbProdotti
ORDER BY Des_Prodotto;

Per visualizzare i prodotti in ordine alfabetico inverso:

SELECT Des_prodotto, Prezzo
FROM TbProdotti
ORDER BY Des_prodotto DESC;

SQL AND & OR

L'operatore AND visualizza una riga se TUTTE le condizioni sono vere. L'operatore OR visualizza una riga se UNA delle condizioni è vera.

Utilizziamo AND per visualizzare le persone con nome uguale a "Aldo", e cognome uguale a "Baglio":
SELECT *
FROM TbPersone
WHERE Nome = 'Aldo' AND Cognome ='Baglio';
Esempio
Utilizziamo OR per visualizzare le persone con nome uguale a "Aldo", oppure cognome uguale a "Baglio":

```
SELECT *
FROM TbPersone
WHERE Nome = 'Aldo' OR Cognome ='Baglio';
```

Possiamo anche combinare AND e OR (usando le parentesi per rendere più chiare le espressioni complesse):

```
SELECT *
FROM TbPersone
WHERE (Nome = 'Aldo' OR Nome = 'Filippo')
AND Cognome = 'Baglio';
```

SQL BETWEEN..AND

Sintassi

```
SELECT  Nome_Colonna
FROM Nome_Tabella
WHERE Nome_Colonna BETWEEN valore1 AND valore2
```

Per visualizzare le persono alfabeticamente tra "Bisio" e "De Ceglie" (inclusi):

```
SELECT * FROM TbPersone
WHERE Cognome BETWEEN 'Bisio' AND 'De Ceglie';
```

Esempio
Per visualizzare le persone al di fuori del range utilizzare l'operatore NOT:

```
SELECT * FROM TbPersone
WHERE Cognome NOT BETWEEN 'Bisio' AND 'De Ceglie';
```

SQL Alias

Gli Alias SQL sono usati per dare dei nomi alle colonne e alle tabelle.

Alias di colonne

Sintassi:

```
SELECT colonna AS alias_colonna FROM tabella
```

Istruzione SQL:

```
SELECT Cognome AS Famiglia, Nome
FROM TbPersone;
```

Alias di tabella

Sintassi:

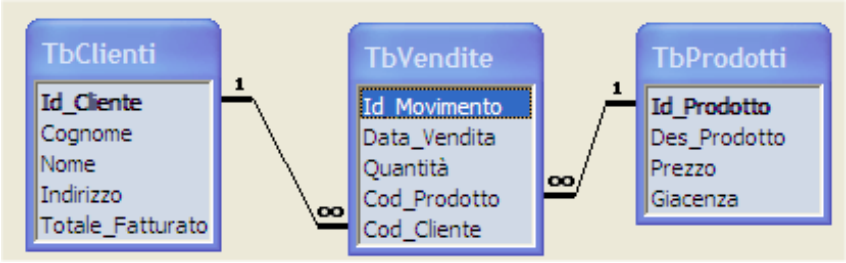
```
SELECT colonna FROM tabella AS alias_ tabella
```

Esempio: alias di tabella
Istruzione SQL:

```
SELECT Cognome, Nome
FROM TbPersone AS Impiegati;
```

SQL JOIN

A volte dobbiamo selezionare dati da due o più tabelle per avere dei risultati complete. In questi casi occorre effettuare dei Join.
Si abbia il seguente DataBase, che riguarda le vendite effettuate a Clienti



Le tabelle del database sono in relazione l'una con l'altra tramite le chiavi primarie ed esterne.
Nella tabella "TbClienti" il campo "ID_Cliente" è la chiave primaria, questo significa che non possono esistere due righe con lo stesso identificatore. ID_Cliente distingue due persone, anche se queste hanno lo stesso nome.
Nella tabella TbProdotti il campo Id_Prodotto è la chiave primaria.
Se guardiamo l'esempio, notiamo che:
Il campo "Cod_Prodotto" è la chiave esterna per la tabella "TbVendite" per fare riferimento al prodotto venduto
Il campo "Cod_Cliente" nalla tabella "TbVendite" è usata per fare riferimento al cliente nella tabella "TbClienti" senza dover usare il nome.

Fare riferimento a più tabelle

Chi ha acquistato un qualunque prodotto, e quali prodotti sono stati acquistati?

Esempio: Chi ha acquistato una stampante?

```
SELECT Cognome, Nome, Des_Prodotto
FROM TbClienti, TbProdotti, TbVendite
WHERE ID_Cliente= Cod_Cliente
AND Id_Prodotto= Cod_Prodotto
AND Des_Prodotto='Stampante%';
```

```
SELECT TbClienti.Cognome, TbProdotti.Des_Prodotto
FROM TbClienti,TbVebdite, TbProdotti
WHERE TbClienti.ID_Cliente=TbVendite.Cod_Cliente
AND TbProdotti.Id_Prodotto=TbVendite.Cod_Vendite;
```

Utilizzare i Join

Possiamo anche selezionare dei dati da due tabelle utilizzando la parola chiave JOIN

Abbiamo tre tipi di Join:
INNER JOIN
LEFT JOIN
RIGTH JOIN
Con significati leggermente diversi

INNER JOIN

L'INNER JOIN restituisce le righe delle tabelle se c'è un legame, altrimenti non le mostra.

Sintassi

```
SELECT campo1, campo2, campo3
FROM prima_tabella, seconda_tabella
WHERE prima_tabella.Chiave_P = seconda_tabella_Chiave_E;
```

Esempio: Chi ha acquistato un prodotto, e quali prodotti sono stati acquistati?

```
SELECT Cognome, Nome, Des_Prodotto
FROM TbClienti
```

INNER JOIN (TbProdotti INNER JOIN TbVendite ON TbProdotti.Id_Prodotto=TbVendite.Cod_Prodotto)
ON TbClienti.Id_Cliente=TbVendite.Cod_Cliente;

Esempio: Chi ha ordinato una stampante?

SELECT TbClienti.Cognome
FROM TbClienti, Tb Ordini
WHERE TbClienti.ID_Cliente= TbOrdini.Cod_Cliente
AND TbOrdini.Prodotto='Stampante';

LEFT JOIN

Il LEFT JOIN restituisce tutte le righe della prima tabella (nell'esempio TbClienti), anche se non ci sono
corrispondenze nella seconda tabella (nell'esempio TbVendite).

Sintassi
SELECT campo1, campo2, campo3
FROM prima_tabella
LEFT JOIN seconda_tabella
ON prima_tabella.chiave_P = seconda_tabella.chiave_E;

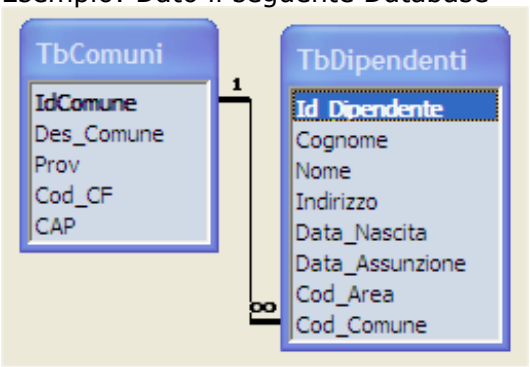
Esempio: Visualizzare tutti i Clienti ed i loro acquisti (se ne hanno)
SELECT TbClienti.Cognome, TbProdotti.Des_Prodotto
FROM TbClienti
LEFT JOIN (TbProdotti LEFT JOIN TbVendite ON Id_prodotto=Cod_Prodotto)
ON TbClienti.ID_Cliente= TbVendite.Cod_Cliente;

RIGHT JOIN

Un RIGHT JOIN restituisce tutte le righe della seconda tabella, anche se non ci sono legami con la prima

Sintassi
SELECT campo1, campo2, campo3
FROM prima_tabella
RIGHT JOIN seconda_tabella
ON prima_tabella.chiave_P = seconda_tabella.chiave_E;

Esempio: Dato il seguente Database



Mostrare tutti i i Comuni italiani, e i Dipendenti che vi risiedono (se ce ne sono)
SELECT Cognome, Nome, Des_Comune
FROM TbDipendenti
RIGHT JOIN TbComuni
ON TbComuni.ID_Comune= TbDipendenti.Cod_Comune;

SQL Funzioni

SQL ha molte funzioni built-in. La sintassi per le funzioni SQL è la seguente:

SELECT funzione(colonna) FROM tabella;

Abbiamo due tipi di funzioni in SQL:

- **FUNZIONI AGGREGATE**
- **FUNZIONI SCALARI**

Funzioni di Aggregazione

Le funzioni aggregate operano su una collezione di valori, e restituiscono un valore singolo.
Nota: se usata insieme ad altre espressioni nell'istruzione SELECT, la SELECT deve avere la clausola GROUP BY!

Funzione	Descrizione
AVG(column)	Restituisce il valore medio di una colonna
COUNT(column)	Restituisce il numero di righe (escludendo quelle con valore NULL) di una colonna
COUNT(*)	Restituisce il numero di righe selezionate
FIRST(column)	Restituisce il valore del primo record di un campo specificato
LAST(column)	Restituisce il valore dell'ultimo record di un campo specificato
MAX(column)	Il valore massimo in una colonna
MIN(column)	Il valore minimo
STDEV(column)	Calcola la deviazione standard sui dati contenuti nella colonna indicata
STDEVP(column)	Calcola la deviazione standard sui dati contenuti nella colonna indicata ignorando i valori NULLL
SUM(column)	Restituisce la somma dei valori di una colonna
VAR(column)	Calcola la Varianza tra I valori contenuti in una colonna
VARP(column)	Calcola la Varianza tra I valori contenuti in una colonna ignorando i valori NULL

Funzioni scalari

Le funzioni scalari lavorano su valori singoli, e restituiscono valori singoli in base al tipo di ingresso.

Funzione	Descrizione
UCASE(c)	Converte un campo in maiuscole
LCASE(c)	Converte un campo minuscole
MID(c,inizio [,fine])	Estrae dei caratteri da un campo di testo
LEN(c)	Restituisce la lunghezza di un campo di testo
INSTR(c)	Restituisce la posizione di un carattere in un capmpo di testo
LEFT(c,numero_di_caratteri)	Restituisce la parte sinistra di un campo di testo
RIGHT(c, numero_di_caratteri)	Restituisce la parte destra di un campo di testo
ROUND(c,decimali)	Arrotonda campo numerico al numero di decimali specificato
MOD(x,y)	Restituisce il resto di una divisione
NOW()	Restituisce la data corrente di sistema
FORMAT(c,format)	Cambia il modo in cui un campo è visualizzato
DATEDIFF(d,date1,date2)	Per fare calcoli sulle date

Funzione SUM

SUM(espr)	Somma dei valori di una colonna. La colonna deve contenere soltanto dati numerici. La funzione SUM considera i record contenenti campi di tipo Null come aventi valore 0.

Calcolare, nella TbClienti, il totale del fatturato

```
SELECT SUM(Totale_fatturato) AS [Fatturato Clienti]
FROM TbClienti;
```

Fatturato Clienti
5653

Funzione AVG

AVG(espr)	Media dei valori di una colonna. La colonna deve contenere soltanto dati numerici. La media calcolata dalla funzione AVG equivale alla media aritmetica e non include nel calcolo i valori di tipo Null presenti nella colonna.

Calcolare, nella tbClienti, la media del fatturato

```
SELECT AVG(Totale_fatturato) AS Media_fatturato
FROM TbClienti;
```

Media_fatturato
353,3125

Funzioni MIN e MAX

MAX(espr)	Valore massimo di una colonna o l'ultimo valore in ordine alfabetico per tipi di dati di testo. I valori NULL vengono ignorati.
MIN(espr)	Valore minimo di una colonna o il primo valore in ordine alfabetico per tipi di dati di testo. I valori NULL vengono ignorati.

Calcolare, nella tbClienti, l'importo minimo e massimo fatturato.

```
SELECT MIN(Totale_fatturato) AS Fatturato_MIN, MAX(Totale_fatturato) AS Fatturato_MAX
FROM TbClienti;
```

Fatturato_MIN	Fatturato_MAX
100	700

Funzione COUNT

La funzione COUNT conta il numero di righe presenti in una tabella.

COUNT(espr), COUNT(*)	Conteggio dei valori di una colonna, se per espr si specifica un nome di colonna, o conteggio di tutte le righe di una tabella o gruppo, se si specifica *. Con COUNT(espr) i valori NULL vengono ignorati, mentre con COUNT(*) vengono inclusi nel conteggio.

Calcolare il numero di righe presenti nella tabella tbClienti

```
SELECT COUNT(*) AS Numero_Clienti
FROM TbClienti;
```

Numero_Clienti
16

Calcolare il numero di clienti per i quali, nella tbClienti, è specificato l'indirizzo.

```
SELECT COUNT(Indirizzo) AS [Clienti con indirizzo]
```

FROM TbClienti;

Clienti con indirizzo
13

Calcolare il numero dei clienti, nella TbClienti, il cui fatturato è superiore a 200 Euro

```
SELECT COUNT(Totale_fatturato) AS Fatturato
FROM tbClienti
WHERE Totale_fatturato > 200;
```

Fatturato
10

Calcolare, nella tbComuni, il numero di comuni della provincia di Taranto

```
SELECT COUNT(Prov) AS Totale_Comuni
FROM TbComuni
WHERE Provincia = 'TA';
```

Totale_Comuni
29

GROUP BY…

GROUP BY... è stata inserita perchè le funzioni aggregate (come SUM) restituiscono un valore aggregato per tutte le colonne tutte le volte che vengono chiamate.

Esempio: Comunicare per ogni Cliente la quantità totale acquistata

```
SELECT Cognome, Nome, SUM(quantità) AS [Quantità totale]
FROM  TbClienti,TbProdotti,TbVendite
WHERE  Id_Cliente=Cod_Cliente
AND Id_prodotto=Cod_Prodotto
GROUP BY Cognome, Nome;
```

Nota: la clausola GROUP BY necessita che vengano ripetuti tutti I campi elencati nella SELECT (escluse le funzioni di aggregazione)

HAVING…

HAVING... è stata inserita perché la parola chiave WHERE non poteva essere usata con le funzioni aggregate.

La sintassi è:

```
SELECT colonna,SUM(colonna) FROM tabella
GROUP BY colonna
HAVING SUM(colonna) condizione valore;
```

Esempio: Elencare gli acquisti effettuati con importo superiore a 10000 euro:

```
SELECT Des_prodotto, SUM(Quantità * prezzo) AS Totale_Ricavo
FROM TbProdotti, TbVendite
WHERE Id_prodotto=Cod_Prodotto
GROUP BY Des_prodotto
HAVING SUM(Quantità * Prezzo)>10000;
```

SQL SELECT INTO

L’istruzione SELECT INTO è spesso usata per creare copie di backup di tabelle oppure per archiviare dei record.

Sintassi
SELECT Nome/Nomi_Colonna
INTO Nuova_Tabella [IN DatabaseEsterno]
FROM Sorgente

Creare una copia di backup

L'esempio seguente mostra come creare una copia di backup della tabella "Persone":
SELECT *
INTO Backup_Persone
FROM TbPersone;

La clausola IN può essere utilizzata per copiare le tabelle in un altro database:
SELECT TbDipendenti.*
INTO Persone
IN 'Backup.mdb'
FROM TbDipendenti;

Se vogliamo copiare soltanto alcuni campi, possiamo indicarli nella SELECT:

SELECT Cognome, Nome
INTO Backup_Persone
FROM TbDipendenti;

Possiamo anche aggiungere una clausola WHERE. L'esempio seguente crea una tabella "Backup_Persone " con due colonne (Cognome e Nome) estraendo le persone che vivono a Milano dalla tebella TbDipendenti:

SELECT Cognome, Nome
INTO Persone_Milano
FROM TbDipendenti, TbComuni
WHERE Id_Comune=Cod_Comune
AND Des_Comune='Milano';

E' anche possibile selezionare dati da più di una tabella. L'esempio successivo crea una nuova tabella "Backup_Clienti_Acquisti" che contiene dei dati provenienti dal join di due tabelle:

SELECT TbClienti.Cognome, TbProdotti.Des_Prodotto
INTO Backup_Clienti_Acquisti
FROM TbClienti, TbVendite, TbProdotti
WHERE TbClienti.ID_Cliente= TbVendite.Cod_Cliente
AND id_Prodotto= Cod_Prodotto;

Predicato IS NULL

Il predicato IS NULL confronta il valore in una colonna con il valore Null. L'uso di questo predicato è il solo modo per controllare la presenza del valore Null in una colonna. E' possibile inserire l'operatore di negazione NOT per controllare se un attributo non ha valore Null.

Elencare tutti i clienti in cui non risulta alcuna data ordine effettuato :

SELECT Cognome, Nome, Data_Vendita
FROM TbClienti, TbVendite
WHERE Id_Cliente= Cod_Cliente
AND (Data_Vendita IS NULL)
ORDER BY Cognome;

Cognome	Nome	Data_ordine
ANCORA	MICHELE	
BIANCHI	MARIO	
MORANTE	ELSA	

Elencare tutti i clienti per i quali è indicata una per l'ultimo ordine effettuato :

```
SELECT Cognome, Nome, Data_Vendita
FROM TbClienti,TbVendite
WHERE Id_Cliente= Cod_Cliente
AND (Data_ordine IS NOT NULL)
ORDER BY Cognome;
```

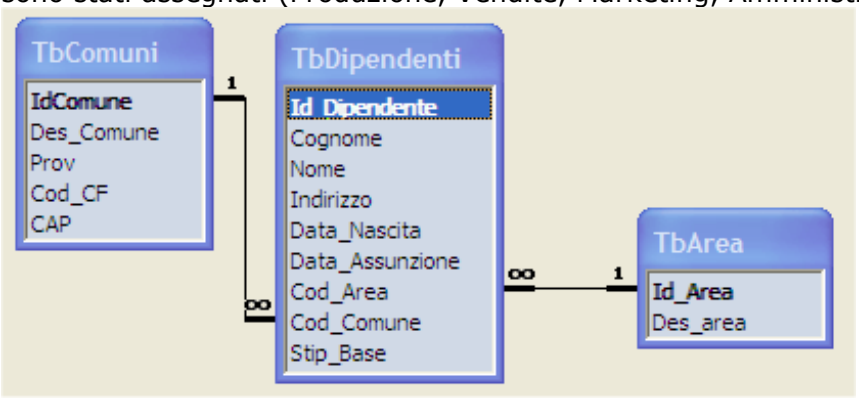
Cognome	Nome	Data_ordine
BIANCHI	GIOVANNI	12/09/2009
GATTO	ALESSANDRO	25/10/2009
PAVONE	GIACOMO	03/01/2010
ROSSI	ANTONIO	13/01/2010
ROSSI	GIUSEPPE	20/01/2010

QUERY NIDIFICATE

Le interrogazioni nidificate consistono in una o più comandi SELECT inseriti all'interno di una istruzione di SELECT.
Esempio: Elenco dei Clienti con totale fatturato inferiore alla media dei fatturati

```
SELECT Cognome, Nome, Totale_Fatturato
FROM TbClienti
WHERE Totale_Fatturato < SELECT (AVG(Totale_Fatturato)
                                 FROM TbClienti);
```

Sia dato il seguente database, contenente le informazioni dei dipendenti e delle aree funzionali a cui sono stati assegnati (Produzione, Vendite, Marketing, Amministrazione,…)



Esempio: Cognome e nome e stipendio base dei dipendenti con stipendio base uguale al valore massimo di tutti gli stipendi dei dipendenti dell'area Produzione in ordine alfabetico:

```
SELECT Cognome, Nome, Des_Area, Stip_Base AS [Stipendio]
FROM TbDipendenti, TbArea
WHERE Id_Area=Cod_area
AND Stip_Base = (SELECT MAX(Stip_Base)
                 FROM TbDipendenti, TbArea
                 WHERE Id_area=Cod_area
                 AND Des_Area='Produzione')
ORDER BY Cognome, Nome;
```

Predicato ANY

Si utilizza quando si desidera che la condizione di ricerca si verifichi almeno per uno dei valori restituiti dalla query.

Esempio: Elenco ordinato con Cognome e Nome e stipendio base dei dipendenti il cui stipendio base è maggiore di uno qualsiasi tra i dipendenti dell'area Amministrazione:

```
SELECT Cognome, Nome, stip_Base AS [Stipendio]
FROM TbDipendenti, TbArea
WHERE Id_Area=Cod_area
AND Stip_base > ANY (SELECT stip_base
                     FROM TbDipendenti, TbArea
                     WHERE Id_area =Cod_Area
                     AND Des_Area = 'Amministrazione')
ORDER BY Cognome, Nome;
```

Predicato ALL

Si utilizza quando si desidera che la condizione di ricerca si verifichi per ciascuno dei valori restituiti dalla query.
Esempio: Elenco dei dipendenti con cognome, nome, stipendio base e Area funzionale di tutti i dipendenti che non sono dell'area funzionale Produzione con stipendio base maggiore di tutti i dipendenti dell'area produzione

```
SELECT Cognome, Nome, Stip_base, Des_area
FROM TbDipendenti, Tb_Area
WHERE Id_Area= Cod_Area
AND des_area <> 'Produzione'
AND Stip_base > (SELECT Stip_base
                 FROM TbDipendenti, TbArea
                 WHERE Id_area = Cod_area
                 AND Des_Area='Produzione');
```

Predicato IN

Si utilizza quando si vuole controllare se il valore di un attributo è compreso tra quelli restituiti dalla subquery
Esempio: Cognome e nome dei dipendenti che lavorano in Aree funzionali con più di 10 dipendenti

```
SELECT Cognome, Nome, Des_Area
FROM TbDipendenti, TBArea
WHERE Id_area=Cod_Area
AND Cod_Area IN (SELECT Cod_Area
                 FROM TbPersonale
                 GROUP BY Cod_Area
                 HAVING COUNT(*) >10);
```