

1. INTRODUZIONE

1.1 Introduzione

Il linguaggio SQL (*Structured Query Language*) è uno strumento che consente di interagire con i database.

1.2 I database

Col termine *Database* si identifica in genere un archivio organizzato di dati in formato digitale. Normalmente i dati sono suddivisi in *Tabelle*, ossia entità che contengono dati omogenei come ad esempio i dati dei clienti di un'azienda, l'elenco dei cittadini di un comune o degli alunni di una classe. Ogni tabella è suddivisa a sua volta in *Campi*. Il campo è l'elemento più elementare all'interno di un database. Tornando all'esempio dei clienti di un'azienda, possiamo identificare tutta una serie di informazioni relative ad essi (ragione sociale, partita I.V.A., indirizzo, ecc.). Ognuna di queste informazioni è definita campo.

2. DDL

Il *Data Definition Language* (DDL) è un linguaggio che permette di creare, modificare o eliminare gli oggetti in un database. Il linguaggio DDL compone una parte del linguaggio SQL.

Sono i comandi DDL a definire la struttura del database e quindi dei dati ivi contenuti. Ma non fornisce gli strumenti per modificare i dati stessi: per tale scopo si usa il *Data Manipulation Language* (DML). L'utente deve avere i permessi necessari per agire sulla struttura del database e questi permessi vengono assegnati tramite il *Data Control Language* (DCL).

2.1 I tipi di dati

In questo testo sono illustrati i tipi di dati utilizzati in Access. Bisogna tenere presente che ci sono alcune differenze tra i vari DBMS, per cui consigliamo di verificare di volta in volta le peculiarità di ognuno di essi. Di seguito sono evidenziati i vari tipi con i quali possiamo definire i campi:

Tipo di dato	Dimensione	Descrizione
BINARY	1 byte per carattere	Può contenere qualsiasi tipo di dato. L'informazione viene inserita come sequenza di byte, esattamente come viene rappresentata in memoria.
BIT o BOOLEAN	1 byte	Questo tipo può contenere i valori 0 o -1, equivalenti del vero o falso.
TINYINT o BYTE	1 byte	Rappresenta un valore intero tra 0 e 255.
MONEY o CURRENCY	8 byte	Può contenere un valore compreso tra - 922.337.203.685.477,5808 e 922.337.203.685.477,5807.
DATETIME, DATE o TIME	8 byte	Serve per memorizzare date o orari.
UNIQUEIDENTIFIER	128 bit	Numero di identificazione univoca usato per chiamate a procedure remote.
REAL o SINGLE	4 byte	Numeri a precisione singola.
FLOAT o DOUBLE	8 byte	Numeri a precisione doppia.
SMALLINT o SHORT	2 byte	Un numero intero compreso tra - 32.768 e 32.767.
INTEGER, INT o LONG	4 byte	Un numero intero compreso tra - 2.147.483.648 e 2.147.483.647.
DECIMAL o NUMERIC	17 byte	Un tipo numerico che può contenere valori da 1028 -1 a - 1028 -1. E' possibile definire sia la precisione (1-28) che la scala (0 - precisione definita). I valori di default per precisione e scala sono rispettivamente 18 e 0.
TEXT o MEMO	2 byte per carattere	Possono contenere da zero a fino ad un massimo di 2.14 gigabyte.
IMAGE	Quanto necessario	Possono contenere da zero fino ad un massimo di 2.14 gigabyte. Usato per gli oggetti OLE.
CHARACTER, CHAR o VARCHAR	2 byte per carattere	Possono contenere da zero fino ad un massimo di 255 caratteri.

2.2 Convenzione e sintassi dei nomi dei campi

I nomi dei campi non possono contenere spazi, simboli (a parte l'underscore o trattino basso “_”) e, di regola, lettere accentate (anche se alcuni DBMS lo consentono). Normalmente è convenzione definire i campi con le lettere minuscole o solo con la prima lettera maiuscola per differenziarli dalle parole chiave del linguaggio che normalmente di scrivono in maiuscolo.

2.3 La creazione delle tabelle

Per creare una nuova tabella all'interno di un database, si utilizza l'istruzione CREATE TABLE che è strutturata con la seguente sintassi:

```
CREATE TABLE Nome Tabella (  
    Campo1 Tipo [(lunghezza campo)], Campo2 Tipo [(lunghezza campo)] ...  
)
```

Questa è la forma più semplice di definizione di una tabella. Passiamo ad un esempio pratico. Supponiamo di avere la seguente tabella da voler inserire nel nostro archivio:

Tabella **CLIENTE**

ID_Cliente è il campo chiave (o Primary Key) ed è di tipo intero. Tutti gli altri campi sono di tipo stringa.

<i>Campo</i>	<i>Tipo(Dimensione)</i>	<i>Chiave</i>
ID_Cliente	Integer	PK
Nominativo	Char(40)	
Indirizzo	Char(50)	
Citta	Char(30)	
Telefono	Char(30)	
Partiva IVA	Char(11)	

Vediamo l'istruzione:

```
CREATE TABLE Cliente (  
    ID_Cliente INTEGER,  
    Nominativo CHAR(40),  
    Indirizzo CHAR(50),  
    Citta CHAR(30),  
    Telefono CHAR(30),  
    PartitaIVA CHAR(11)  
)
```

ATTENZIONE: il campo Citta è scritto senza la lettera accentata perché potrebbe dare dei problemi in alcuni DBMS. Tutte le righe di definizione dei campi sono separate da virgole. Dopo l'ultima riga non c'è la virgola ma direttamente la parentesi chiusa dell'istruzione CREATE

Se vogliamo introdurre anche la definizione della chiave primaria abbiamo due possibilità: dichiararla insieme alla definizione del campo stesso oppure dichiararla alla fine dell'elenco dei campi. Vediamo i due casi:

```
CREATE TABLE Cliente (  
    ID_Cliente INTEGER PRIMARY KEY,  
    Nominativo CHAR(40),  
    Indirizzo CHAR(50),  
    Citta CHAR(30),  
    Telefono CHAR(30),  
    PartitaIVA CHAR(11)  
)
```

```
CREATE TABLE Cliente (  
    ID_Cliente INTEGER,  
    Nominativo CHAR(40),  
    Indirizzo CHAR(50),  
    Citta CHAR(30),  
    Telefono CHAR(30),  
    PartitaIVA CHAR(11),  
    PRIMARY KEY (ID_Cliente)  
)
```

Queste due istruzioni sono equivalenti. Si tende in genere ad utilizzare la seconda perché è più semplice da scrivere quando abbiamo una chiave primaria composta da più campi. Vediamo subito un esempio concreto:

```
CREATE TABLE Magazzino (  
    ID_Centro INTEGER,  
    ID_Magazzino INTEGER,  
    Indirizzo CHAR(50),  
    PRIMARY KEY (ID_Centro, ID_Magazzino)  
)
```

Oltre alla definizione del tipo, possiamo associare ai campi anche un valore di default e l'indicazione che il campo non deve avere il valore NULL quando viene inserito. Il primo serve se vogliamo che un determinato campo sia sempre valorizzato e non sia lasciato vuoto in fase di inserimento di un record. Il secondo serve per evitare di avere un valore indefinito nei campi non valorizzati in fase di inserimento (questo perché il valore NULL provoca una serie di inconvenienti in fase di ricerca e manipolazione dei dati).

Ipotizziamo di volere creare una tabella che contenga i dati degli iscritti ad un sito e vogliamo che per default il flag relativo all'invio della newsletter sia valorizzato a "N" perché sia l'utente stesso ad esplicitare la richiesta di ricezione. Inoltre si vuole che il campo Password non sia mai NULL. Vediamo in concreto.

```
CREATE TABLE Utente (  
    UserName CHAR(15),  
    Password CHAR(15) NOT NULL,  
    Newsletter CHAR(1) DEFAULT 'N',  
    PRIMARY KEY (UserName)  
)
```

2.4 La cancellazione delle tabelle

Si fa presente che l'istruzione CREATE TABLE viene eseguita solo se la tabella non esiste nel database, nel caso contrario il DBMS restituisce un errore a meno che non viene specificata la clausola IF NOT EXISTS. Appliciamola all'esempio precedente.

```
CREATE TABLE IF NOT EXISTS Utente (  
    UserName CHAR(15),  
    Password CHAR(15) NOT NULL,  
    Newsletter CHAR(1) DEFAULT 'N',  
    PRIMARY KEY (UserName)  
)
```

In questo caso se la tabella esiste già il DBMS ignora l'istruzione di creazione della tabella. In caso contrario inserirà la nuova tabella nel database come specificata. Supponiamo invece che la tabella sia già esistente ma abbiamo commesso un errore nella definizione della stessa (per esempio abbiamo dimenticato un campo o lo abbiamo definito con una lunghezza errata). In questo caso abbiamo la possibilità, se non abbiamo ancora inserito dati, di cancellarla con la seguente istruzione:

```
DROP TABLE Utente
```

ATTENZIONE: L'istruzione DROP cancella la tabella senza preservarne il contenuto. Quindi va usata con molta cautela.

Se il comando DROP viene eseguito associato ad un nome di tabella che non esiste nel database il DBMS restituisce un errore. Per evitare questo si usa aggiungere al comando IF EXISTS.

```
DROP TABLE Utente IF EXISTS
```

E' anche possibile unire le due operazioni (di drop e create) nello stesso blocco separando le due operazioni con un punto e virgola, come nel seguente esempio:

```
DROP TABLE Utente IF EXISTS;  
  
CREATE TABLE IF NOT EXISTS Utente (  
    UserName CHAR(15),  
    Password CHAR(15) NOT NULL,  
    Newsletter CHAR(1) DEFAULT 'N',  
    PRIMARY KEY (UserName)  
)
```

ATTENZIONE: Alcune versioni di Access non consentono di eseguire le istruzioni di DROP con il parametro IF EXISTS. E' quindi possibile che le tabelle debbano essere rimosse manualmente verificandone la presenza.

2.5 Tabelle con chiavi esterne

Se una tabella ha una relazione attraverso una chiave esterna con un'altra tabella bisogna inserire la dichiarazione della relazione all'interno dell'istruzione CREATE TABLE. Vediamo un esempio

Tabella ARTICOLO			Tabella FORNITORE		
Campo	Tipo(Dimensione)	Chiave	Campo	Tipo(Dimensione)	Chiave
Codice_Articolo	Char(12)	PK	ID_Fornitore	Integer	PK
Descrizione	Char(50)		Nominativo	Char(40)	
Prezzo	Money		Indirizzo	Char(50)	
Scorta	Integer		Citta	Char(30)	
ID_Fornitore	Integer	FK	Telefono	Char(30)	
			Partiva IVA	Char(11)	

In questi casi bisogna creare prima la tabella che non ha chiavi esterne, nel caso specifico la tabella *Fornitore*. Questo perché se la tabella con cui si crea la relazione non esiste ancora, il DBMS restituisce un errore e quindi non crea la tabella. Vediamo le istruzioni di creazione.

```
CREATE TABLE Fornitore (  
    ID_Fornitore INTEGER,  
    Nominativo CHAR(40),  
    Indirizzo CHAR(50),  
    Citta CHAR(30),  
    Telefono CHAR (30),  
    PartitaIVA CHAR (11),  
    PRIMARY KEY (ID_Fornitore)  
)
```

Adesso che abbiamo creato la tabella *Fornitore* possiamo creare anche la tabella *Articolo* che ha una chiave esterna sulla prima.

```
CREATE TABLE Articolo (  
    Cod_Articolo CHAR(12),  
    Descrizione CHAR(50),  
    Prezzo CURRENCY,  
    Scorta INTEGER,  
    ID_Fornitore INTEGER,  
    PRIMARY KEY (Cod_Articolo),  
    FOREIGN KEY (ID_Fornitore)REFERENCES Fornitore (ID_Fornitore)  
)
```

ATTENZIONE: Si ricorda tutte le definizioni sono separate da virgole, tranne l'ultima che precede la chiusura della parentesi dell'istruzione CREATE. Quindi avendo inserito la definizione della chiave esterna, dopo l'istruzione PRIMARY KEY dobbiamo mettere una virgola.

2.6 Tabelle con chiavi primarie e/o esterne composte da più campi

Se una tabella ha una relazione attraverso una chiave esterna con un'altra tabella bisogna inserire la dichiarazione della relazione all'interno dell'istruzione CREATE TABLE. Vediamo un esempio

Tabella **ALUNNO**

Campo	Tipo(Dimensione)	Chiave
Matricola	Char(12)	PK
Cognome	Char(30)	
Nome	Char(30)	
ID_Classe	Integer	FK
Sezione	Char(2)	FK

Tabella **CLASSE**

Campo	Tipo(Dimensione)	Chiave
ID_Classe	Integer	PK
Sezione	Char(2)	PK
Descrizione	Char(50)	

Tabella **DOCENTE**

Campo	Tipo(Dimensione)	Chiave
ID_Docente	Integer	PK
Cognome	Char(30)	
Nome	Char(30)	
Indirizzo	Char(50)	
Data_Nascita	Date	

Tabella **DOCENTE_CLASSE**

Campo	Tipo(Dimensione)	Chiave
ID_Classe	Integer	PK (FK)
Sezione	Char(2)	PK (FK)
ID_Docente	Integer	PK (FK)
Materia	Char(15)	PK

Bisogna fare molta attenzione in questi casi nei quali ci sono relazioni che interessano diversi campi. Ma andiamo con ordine nella creazione delle tabelle. La prima da creare è quella *Classe* perché non ha chiavi esterne.

```
CREATE TABLE Classe (  
    ID_Classe INTEGER,  
    Sezione CHAR(2),  
    Descrizione CHAR(50),  
    PRIMARY KEY (ID_Classe, Sezione)  
)
```

ATTENZIONE: La PRIMARY KEY l'abbiamo definita in fondo alla CREATE TABLE, unendo i vari campi che la compongono, sia per comodità, sia perché su alcune versioni di ACCESS definire singolarmente i campi che compongono la chiave primaria dà luogo ad un errore.

Adesso possiamo procedere alla creazione della tabella *Alunno* che ha delle chiavi esterne che fanno riferimento alla tabella *Classe*.

```
CREATE TABLE Alunno (  
    Matricola CHAR(12),  
    Cognome CHAR(30),  
    Nome CHAR(30),  
    ID_Classe INTEGER,  
    Sezione Char(2),  
    PRIMARY KEY (Matricola),  
    FOREIGN KEY (ID_Classe, Sezione) REFERENCES Classe (ID_Classe, Sezione)  
)
```

A questo punto creiamo la tabella *Docente* che non ha riferimenti esterni.

```
CREATE TABLE Docente (  
    ID_Docente INTEGER,  
    Cognome CHAR(30),  
    Nome CHAR(30),  
    Indirizzo CHAR(50),  
    Data_Nascita DATE,  
    PRIMARY KEY (ID_Docente)  
)
```

Infine possiamo creare la tabella *Docente_Classe* che fa riferimento a due tabelle *Docente* e *Classe*.

```
CREATE TABLE Docente_Classe (  
    ID_Classe INTEGER,  
    Sezione CHAR(2),  
    ID_Docente INTEGER,  
    Materia CHAR(15),  
    PRIMARY KEY (ID_Classe, Sezione, ID_Docente, Materia),  
    FOREIGN KEY (ID_Classe, Sezione) REFERENCES Classe (ID_Classe, Sezione),  
    FOREIGN KEY (ID_Docente) REFERENCES Docente (ID_Docente)  
)
```