**TIPI FONDAMENTALI**

L' SQL è pur sempre un linguaggio e come tale incorpora i tipi oramai standard: caratteri (e quindi stringhe), numeri (sia interi che razionali). Qualche novità di rilievo? Sì: il tipo *bit, data/ora* ed il tipo *null*.

Piccola guida di riferimento per apprendere meglio la sintassi:

[ opzionale ] = il parametro tra parentesi quadre è opzionale, cioè può essere omesso.

(necessario) = il parametro necessita delle parentesi tonde per indicarne, in genere, la dimensione.

{ ricorsività } = il parametro può essere ripetuto più volte.

le parole in **neretto** sono proprie della sintassi di SQL e vanno scritte così come sono, quelle in

***nerettocorsivo*** indicano gli argomenti dei comandi e andranno sostituite con dei valori numerici.

Iniziamo ad illustrare i tipi base:

- Caratteri/stringhe

Sono i "soliti" caratteri, che possiamo estendere alle stringhe (anche di lunghezza variabile) tramite un parametro

**character [ varying ] [ (*Lunghezza*) ]**

Esempio.

Cognome **character**(10)

Ecco qui, abbiamo definito un campo 'Cognome' di tipo stringa lungo 10 caratteri.

Se ne usassimo 12? Perdiamo informazioni, perché saranno ignorati dopo il decimo!Come risolvere?

Si utilizza il parametro "varying", che definisce unna stringa di lunghezza non definita, anche se ovviamente un limite fisico esiste, ma lo assumiamo così grande da valutarlo come infinito.

Suggerimento: si può utilizzare la sintassi alternativa **char** e **varchar** per **charater** e **varying character**

Esempio.

Cognome **char**(10)

- Numerici

**numeric [ (*Precisione* [, *Scala* ]) ]**

**decimal [ (*Precisione* [, *Scala* ]) ]**

Con "Precisione" si indica la totalità delle cifre da utilizzare, mentre con "Scala" specifichiamo quante cifre possono seguire dopo la virgola (se non specificato, sarà usato 0).

Esempi.

Valuta decimal(4, 2) ciò vuol dire che -99, 99<=Valuta<=+99.99 4 cifre di cui 2 dopo la virgola

Totale numeric(4) ciò vuol dire che -9999<=Totale<=+9999

Qual è la differenza tra **numeric** e **decimal**? Il primo rappresenta un valore esatto, l'altro un requisito minimo di base.

Suggerimento: scegliete ed utilizzate sempre uno dei due, perché sono praticamente equivalenti.

Se non siamo interessati alla precisione della rappresentazione, possiamo scegliere, invece, i tipi

**integer**

**smallint**

Se volessimo utilizzare valori numerici per rappresentare grandezze fisiche o di più alta precisione:

**float [ (*Precisione*) ]**

**double precision**

**real**

Con questi tipi possiamo rappresentare numeri approssimati tramite la notazione in virgola mobile, costituita da mantissa ed esponente. Per ottenere il numero reale associato si moltiplica la mantissa per la potenza di 10 di grado indicato dall'esponente (ES. 3, 456E4 =3456*10)

Al tipo **float** possiamo indicare la "Precisione" della mantissa, cioè il numero di cifre da utilizzare, mentre il tipo **double precision** utilizza un numero di bits che è doppio rispetto al **real**.

- Bit

Quante volte avete usato i "flag" programmando i vostri applicativi?Sono delle variabili "bandierine" che ci dicono se un particolare evento è accaduto (si alza la bandiera...) e il flusso del programma cambia a seconda di quei valori.In SQL esiste il tipo

**bit [ varying ] [ (*Lunghezza)* ]**

Anche qui possiamo definire una lunghezza variabile, come per il tipo numerico.

Esempi.

Codice Bit (3)

ShorterCodeBar Bit varying (10)

Codice può assumere uno tra le 8 configurazioni a 3 bit, ShorterCodeBar potrà usare un numero

variabile di bits, fino ad un massimo di 10.

- Data/ora

Utilissimo in caso di D.B. che utilizzino le transazioni oppure informazioni di tipo temporale, ad es. agenda, scadenze.

Una caratteristica importante è che ogni dominio può essere "spezzettato", decomposto in vari altri campi: facendo un paragone matematico, i campi sono sottoinsiemi dell'insieme Dominio.

**date**

Di questo Dominio (Tipo) possiamo controllare il campo **year**, il campo **month** e **day**

**time [ (*Precisione)* ] [ with time zone ]**

Qui possiamo interrogare, una volta riempiti, i campi **hour, minute, second.** Il parametro "Precisione" indica quante cifre decimali devono essere usate per rappresentare le frazioni di secondo (default=0, cioè precisione al secondo). Il parametro "with time zone" indica la differenza di fuso orario rispetto all'ora di Greenwich; quando abilitato, possiamo accedere ai campi **timezone_hour** e **timezone_minute.**

Esempio.

13: 45: 23+0: 00 = 14: 45: 23+1: 00

**timestamp [ (*Precisione)* ] [ whit time zone ]**

Sostanzialmente è una fotografia dell'istante di tempo in cui il campo viene riempito, contenente tutti i valori descritti, da **year** a **second**.

- Intervalli temporali

Permettono di incrementare il dettaglio di precisione sulla data da utilizzare

**interval *UnitàDiTempoPrincipale* [ to *UnitàDiTempoSecondaria* ]**

dove " UnitàDiTempoPrincipale " è più precisa di " UnitàDiTempoSecondaria "

N.B. vi sono 2 gruppi distinti di confronti: il primo è **year** con **mounth**, il secondo è **day** con **second:** questo perché non si possono confrontare i giorni con i mesi (i quali hanno da 28 a 31 giorni)

Se " UnitàDiTempoSecondaria " è impostato a **second,** si può utilizzare un range che indica le cifre significative;mentre se lo è " UnitàDiTempoPrincipale " (e quindi la sola) l'(eventuale) parametro anche qui indica il numero di decimali.Il valore di default è 2.

Esempi.

**interval year(3) to month** ha una precisione pari a 999 anni e 11 mesi

**interval day(4) to second(5)** è pari a 9999 giorni, 23 ore, 59 minuti e 59, 99999 secondi.

**Null**

Il valore **Null** è quello che si definisce un "valore polimorfico", cioè variabili di tipo diverso possono assumere lo stato **Null** e se confrontate si otterrebbe un type mysmatch.Potremmo dire che assume il valore nullo del tipo della variabile.

## CREAZIONE DI TIPI

Oltre ai tipi appena visti, possiamo crearne altri: la creazione è simile ad una ridenominazione di un tipo fondamentale visto tra quelli sopra o di un tipo creato ex-novo, ereditandone tutte le caratteristiche.

**create domain *NomeDominio* as *Tipo* [ *ValoreImpostato* ] [ *Vincolo* ] { *DefSchema* }**

Si crea un tipo di nome " NomeDominio " partendo da un precedente " Tipo ", impostando un opzionale valore di default ed un insieme di vincoli. Questa operazione permette di definire una ed una sola volta tutte le caratteristiche (vincoli) che possono essere associati ad un attributo, quando questo è utilizzato in più tabelle, evitando così ridondanze.

## DEFINIZIONE DI SCHEMA

Iniziamo ora a creare i D.B.Lo schema è una collezione di tutti gli oggetti che faranno parte della base di dati, cioè domini, tabelle, viste, privilegi, asserzioni.E' un po' come la parte dichiarativa di un programma scritto con un linguaggio imperativo: prima creiamo i tipi, le funzioni e le procedure, poi le utilizziamo nel corpo.

**create schema [ *NomeSchema* ] [ [ autorization ] *Autorizzazione* ] { *DefElementoSchema* }**

*Autorizzazione* è il nome dell' utente propietario dello schema;se mancante si assume che sia l'utente che ha lanciato il comando.Il *NomeSchema* può essere omesso: in tal caso il nome sarà quello dell'utente propietario.Per ogni utente ci sono delle *Autorizzazioni* cioè possiamo imporre certi limiti a chi interroga la nostra B.D.La loro utilità e caratteristica sarà considerata più in là, quando sarà introdotto il concetto di Vista.

## DEFINIZIONE DI TABELLE

Una tabella è un insieme ordinato di attributi e di vincoli (questi ultimi possono anche mancare). Perno fondamentale delle basi di dati, permettono di collezionare dati secondo caratteristiche comuni o di interesse comune (ad esempio, per un'agenzia sapere codice fiscale, nome, cognome ed indirizzo di un cliente è di vitale importanza; raggrupparle in una tabella è l'operazione più comoda).

**create table *NomeTab* (*NomeAttr Dominio* [ *ValoreImpostato* ] [ *Vincoli* ]**

**{,** ***NomeAttr Dominio*** **[** ***ValoreImpostato*** **] [** ***Vincoli*** **] }**

***AltriVincoli***

***)***

Esempio.

**create table** Cliente

**(**

CodFiscale **char(16) primary key,**

Nome **char(15),**

Cognome **char(20),**

Indirizzo **char(50)**

**)**

Si può notare che l'attributo CodFiscale ha un vincolo - **primary key** - il cui significato sarà visto tra breve.

**DEFAULT**

Può essere utile una struttura che assegna automaticamente ad un attributo un valore particolare da noi prescelto, senza doverlo riempire " a mano " ogni volta.Ad esempio pensiamo ad un sistema di ordini di prodotti on line e assumiamo di essere dei clienti: quando scegliamo un articolo è ovvio che la quantità deve essere maggiore o uguale ad 1.Una buona applicazione lato-server deve considerare questi piccoli - ma importanti -particolari.

Esempio.

ProdottoScelto **smallint default** 1

Inoltre possiamo definire anche un attributo che ci informa dell'utente che sta interrogando la base di dati:

Cliente **char(15) default user**

In questo modo il processo sul server potrà sapere quale utente ha avuto accesso alla base di dati.

Ultima possibilità è quella di annullare una precedente impostazione di un attributo, dando il valore null.

Dato **smallint default null**

**VINCOLI**

- INTRARELAZIONALI

Sono vincoli che interessano una sola tabella (relazione nel modello E.R. da cui il nome) e quindi hanno senso solo sugli attributi dove definiti. Essi sono:

**Primary key**

Può essere specificato una sola volta per tabella;con esso si dichiarano uno o più attributi come chiave primaria.Una chiave primaria serve a rendere univoche le righe della tabella ove essa è definita: due righe distinte non potranno avere lo stesso valore sui campi scelti come **Primary key**.

**Not null**

Quando si dichiara un attributo con valore **Not null** significa che il valore nullo non può essere accettato. Se si definisce un valore di default, si può non immettere nulla in quanto non verrà inserito **Null** bensì il valore preinpostato. Tale valore viene assegnato facendolo seguire alla definizione di un attributo.

Esempio.

Utilizzando la tabella Cliente sopra definita:

**create table** Cliente

**(**

CodFiscale **char(16) primary key,**

Nome **char(15) not null,**

Cognome **char(20) not null,**

Indirizzo **char(50)**

**)**

**Unique**

E' una caratteristica simile al **Primary Key**, con la particolarità che un attributo dichiarato **Unique** può assumere il valore nullo. Si può definire il vincolo in due modi: su di un singolo attributo oppure su una raccolta di atrributi delimitati da parentesi tonde.

Esempio.

**create table** Cliente

**(**

CodFiscale **char(16) primary key,**

Nome **char(15) unique,**

Cognome **char(20) unique,**

Indirizzo **char(50)**

**)**

oppure

**create table** Cliente

**(**

CodFiscale **char(16) primary key,**

Nome **char(15),**

Cognome **char(20),**

**unique (**Nome, Cognome)

Indirizzo **char(50)**

**)**

Attenzione: le due dichiarazioni NON sono equivalenti! Nella prima non esistono due righe nella tabella che contengono o lo stesso nome o cognome (cioè non possono esistere due Giovanni o due De Rossi), mentre nella seconda non esistono due righe con lo stesso valore su Nome e Cognome (cioè non esistono gli omonimi, ma più persone si possono chiamare Giovanni)

- INTERRELAZIONALI

Legami tra due o più tabelle.

Potremmo fare un esempio dicendo che una tabella non può essere riempita finché un attributo di un'altra tabella non è stato riempito: è inutile compilare una tabella Immatricolazione_Auto se non vi è un cliente che abbia acquistato la relativa auto!

Il vincolo interrelazionale più utilizzato è sicuramente quello di *integrità referenziale*, cioè attributi che possono assumere soltanto dei valori specificati in un'altra tabella. SQL permette di utilizzare questa regola tramite un apposito costrutto, **foreign key** (chiave esterna): si crea un collegamento tra gli attributi della tabella che state creando con un insieme di attributi di una tabella già esistente.

Chiameremo *slave* la tabella da creare, *master* l'altra. L'unica imposizione che deve essere rispettata è che nella tabella master gli attribuiti a cui si riferisce siano dichiarati come **unique**; generalmente essi sono delle chiavi primarie quindi l'unicità è assicurata.

Se dovessimo utilizzare il legame tra un solo attributo della tabella slave con un solo attributo della tabella master si utilizza il costrutto **references**.

Nota: ogni qual volta si utilizza la **foreign key** bisogna sempre specificare a quale attributo ci stiamo riferendo, facendo seguire alla chaive esterna il comando **references**.

Esempio.

**create table** Studenti

(

Matricola **char(6) primary key,**

Nome **char(10) not null,**

Cognome **char(15) not null,**

Facoltà **char(15) references** Area(Tipo_Facoltà)

)

Ogni volta che si immette un valore in "Facoltà" esso viene verificato se presente nelle righe di "Area" verificando l'attributo "Tipo_Facoltà".

**create table** Immatricolazione_Auto

(

Targa **char(7) primary key,**

Nome_Cliente **char(10) not null,**

Cognome_Cliente **char(15) not null,**

Marca **char(10),**

Modello **char(10),**

Data_Pratica **date**

**foreign key** (Nome_Cliente, Cognome_Cliente) **references** Anagrafica_Clienti(Nome, Cognome)

**foreign key** (Marca, Modello) **references** Modelli_Auto(Marca_Auto, Modello_Auto)

)

Quasi tutti i campi sono chiavi.Gli attributi "Nome_Cliente" e "Cognome_Cliente" devono già esistere nella master "Anagrafica_Clienti" così come i valori immessi in "Marca" e "Modello" devono esistere "Marca_Auto" e "Modello_Auto" nella tabella "Modelli_Auto". Se tali dati non sono presenti l'input verrà rifiutato. Se il vincolo viene violato nella tabella slave allora l'input verrà semplicemente ignorato. Diverso è il discorso se la violazione si presenta sulla master;vi sono diverse possibili soluzioni da adottare a seconda della modifica che si effettua. I casi sono cancellazione o aggiornamento:

Cancellazione:

**on delete**

- **cascade**: tutte le righe della slave verranno cancellate.
- **set null:** al posto del valore dell'attributo chiave viene posto il valore NULL.
- **set default:** viene caricato il valore di default precedentemente definito (strategia ottimale).
- **no action:** reazione passiva, cioè non viene fatto nulla.

Per simmetria anche nel caso di aggiornamento di attributi nella master si hanno le seguenti ripercussioni sulla/e slave

**on upgrade**

- **cascade:** il *nuovo* valore della master viene copiato nella/e slave.
- **set null:** il valore associato alla chiave viene sostituito con NULL.
- **set default:** associa il valore di base.
- **no action**: non modificare i dati nella tabella.

Sintatticamente devono seguire la definizione di foreign key. Le operazioni sono combinabili tra loro, nel senso che è possibile decidere al momento della definizione della tabella le strategie da utilizzare in caso di modifica delle chiavi esterne.

Esempio.

**create table** Immatricolazione_Auto

(

Targa **char(7) primary key,**

Nome_Cliente **char(10) not null,**

Cognome_Cliente **char(15) not null,**

Marca **char(10),**

Modello **char(10),**

Data_Pratica **date**

**foreign key** (Nome_Cliente, Cognome_Cliente) **references** Anagrafica_Clienti(Nome, Cognome)

**on delete no action**

**foreign key** (Marca, Modello) **references** Modelli_Auto(Marca_Auto, Modello_Auto)

**on upgrade cascade**

**on delete no action**

)

Se il cliente viene cancellato dalla tabella "Anagrafica_Clienti" non vengono modificati i dati.

Se l'automobile viene modificata nella tabella "Modelli_Auto", aggiorna tutte le tabelle ove si presenta.

Se l'auto viene eliminata dall'archivio non si deve modificare nulla.