Linguaggio SQL

Il *modello relazionale* definisce i **concetti generali** e i **vincoli** per modellare e strutturare i dati di una certa applicazione. Tuttavia, tramite lo studio del solo modello relazionale, non sarebbe possibile costruire lo *schema* del DB e manipolare le *istanze* associate. Questa richiesta è soddisfatta tramite l'impiego di **linguaggi data-oriented**, i quali possono essere suddivisi in tre macro-categorie rispetto alla compatibilità o meno con un RDBMS, ossia:

- *Linguaggi rappresentativi tramite interfacce grafiche*
- *Linguaggi basati su proprietà algebrico/logiche*
- *Linguaggio SQL*

Tra i tre citati senza alcun dubbio, in associazione ad un modello relazionale, **SQL** rappresenta il *linguaggio di riferimento*.

Introduzione

SQL è un linguaggio per la costruzione di modelli relazionali. Caratterizzato dagli stessi concetti generali su cui si fonda un qualsiasi *modello relazionale*, ma pone alcune differenze, in riferimento:

- La denominazione data alle *relazioni* in questo contesto prende il nome di **tabelle**
- Il risultato di un'operazione sui dati può restituire una tabella con **righe duplicate**
- Il sistema dei **vincoli** è più **espressivo** o **informativo**
- Il vincolo di **integrità referenziale**, ossia in relazione alla *foreign key*, è **meno stringente**. Si ricordi la regola generale, la quale richiede che tra tabella referenziante e referenziata, le colonne che compongano la referenza siano corrispettivamente qualsiasi dominio per la prima relazione, ma che ricada esattamente sulla chiave primaria della seconda relazione; ciò rispetto ad una manipolazione di dati potrebbe aumentare di molto la complessità gestionale.

La grande diffusione di SQL è dovuta da un duro lavoro di standardizzazione che ha portato ad innumerevoli versioni e modifiche del paradigma, impegno portato avanti sin dagli anni ottanta. Tuttavia ancora oggi, differenti pubblicazioni sono ancora lontane dall'essere comunemente adottate. Per questa ragione, spesso si fa riferimento a *SQL-2*, il quale pone sottili caratteristiche legate alla sintattica del linguaggio.

SQL non è solamente adottato per esprimere interrogazioni rispetto alla base di dati di riferimento. Contiene infatti al suo interno funzionalità dedicate al **DDL**, *Data Definition Language*, il quale permette la definizione dello *schema* di una base di dati, e al **DML**, *Data Manipulation Language*, ossia un insieme di comandi per la modifica e l'interrogazione delle istanze di una base di dati.

Data Definition Language

La prima parte trattata riguarda le funzionalità adoperate per la definizione dello *schema* della base di dati. Tramite il costrutto *create database*, è possibile costruire uno **schema** di

una base di dati, da non confondere con lo *schema* di una tabella, ossia dall'insieme posto dal nome della relazione e degli attributi associati. I comandi dedicati sono:

create database [if not exists] nameDataBase

implementato per la creazione del collettore successivo di *tabelle*, *viste* e *interrogazioni* alle istanze di riferimento, e di seguito è riportato il corrispondente comando per l'eliminazione del database

drop database [if exists] nameDataBase

si osserva come in entrambe le funzioni siano descritti comportamenti *opzionali*, dovuti dall'uso della parentesi quadrate *[if not exists]* e *[if exists]*, le quali permettono di verificare, in questo contesto, dell'esistenza o meno del DB indicato nella nomenclatura.

Tramite il costrutto *create table*, è possibile costruire una **tabella** all'interno dello *schema* del DB. Il comando dedicato è:

create table nameTable (

nameAttribute1 Domain [ValDefault][Constraint]
nameAttribute2 Domain [ValDefault][Constraint]
...

);

nuovamente, tutto ciò che è circondato da parentesi quadrate indica un comportamento opzionale, ad eccezione del nome e del dominio dell'attributo specifico. In questo esempio la denominazione *[ValDefault]* stabilisce quale sia la variabile di default associata all'istanza immessa qualora non sia specificato il valore, mentre *[Constraint]* stabilisce un vincolo al quale un qualsiasi record immesso dovrà sottostare, di seguito verranno riportati i *vincoli* principali.

Domini elementari

In SQL possono essere associati differenti **domini elementari** agli attributi di uno schema; *schema* inteso a livello di tabella, per cui l'insieme dettato dal nominativo della relazione e dal nome degli attributi. *Character*
Il dominio *character* permette di rappresentare singoli caratteri oppure stringhe. La lunghezza di una stringa può essere sia fissa che variabile, nonostante in entrambi casi occorra specificare la lunghezza massima. La sintassi è:

character [varying] [(Length)] [character set Language]

quindi qualora si voglia inserire una stringa di 20 caratteri occorre specificare *character (20)*, oppure in relazione ad una stringa con un massimo di 1000 caratteri, adottando uno specifico alfabeto occorre indicare *character varying (1000) character set Greek*. Se la lunghezza non è specificata, il dominio dell'attributo indica un unico carattere. SQL ammette anche delle varianti compatte, spesso più utilizzate rispetto alle prime citate, le quali rispettivamente

sono *char* e *varchar*, per stabilire una dimensione fissa piuttosto che una dimensione variabile.

Tipi numerici esatti
Questa famiglia consente di rappresentare valori esatti, interi o con una parte decimale di lunghezza prefissata. SQL mette a disposizione differenti soluzioni, quali:

numeric [(Precision [, Scale])]

decimal [(Precision [, Scale])]

integer

smallint

I domini *numeric* e *decimal* rappresentano numeri in base *decimale*. Il parametro *Precision* indica l'accurattezza del numero, mentre *Scale* indica quante cifre sono dedicate alla parte frazionaria.

numeric (4, 2) : Intervallo[-99.99 : 99.99]

Nei casi in cui non sia zona di interesse la rappresentazione della parte frazionaria, allora diventa possibile usare i domini predefiniti *int* e *smallint*. Rispetto a questi ultimi due formati è possibile combinare il campo *auto−increment*, mediante (*integer auto−increment*), consentendo di creare dei campi numerici che si auto-incrementano ad ogni nuovo inserimento nella tabella.

Tipi numerici approssimati
I tipi **numerici approssimati** consentono di rappresentare valori reali con rappresentazione in virgola mobile. SQL fornisce i seguenti tipi:

float [(Precision)]

real

doule precision

I domini in questione permettono la rappresentazione di numeri approssimati mediante l'utilizzo di una virgola mobile, suddividendo il numero in una coppia di valori: la mantissa e l'esponente (*0.17E16 −> 1,7∗10*^{15}*, dove 1,7 indica la mantissa mentre 15 l'esponente*).

Domini Temporali
I **domini temporali** consentono di rappresentare informazioni temporali, intervalli di tempo o istanti di tempo. SQL-2 mette a disposizione tre diverse forme:

date

time [(Precision)] [with time zone]

timestamp [(Precision)] [with time zone]

Ciascuno di questi domini è strutturato e suddivisibile in un insieme di campi. Come ad esempio *date* ammette i campi *year*, *month* e *day*, il dominio *time* ammette *hour*, *minute* e *second*, infine *timestamp* ammette tutti i campi, da *year* a *second*. Sia per *time* che per *timestamp* è possibile specificare una precisione, che rappresenta il numero di cifre decimali

che si devono utilizzare nella rappresentazione delle frazioni di secondo.

Domini elementari
I **domini elementari blob** e **cblob** consentono di rappresentare oggetti di grandi dimensioni come sequenza di valori binari o di caratteri. La *dimensione massima* del file dipende dalla specifiche implementazioni del linguaggio SQL. Tuttavia tramite tale formato non è possibile specificare interrogazioni sui record accomunati da tale attributo.

Tramite il costrutto **domain** è possibile costruire il proprio **dominio di dati** a partire dai domini elementari. La sintassi risulta essere cosi descritta:

create domain NameDomain as TypeDate

[Default value]
[Constraint]

create domain NameDomain as TypeDate Default (27)

inoltre rispetto a quanto detto, è possibile specificare un **valore di default** attraverso il costrutto *default*, ossia

default [value | user | null]

Vincoli intra-relazionali

Sia nella definizione dei domini che delle tabelle, possono essere definiti alcuni vincoli, ovvero delle proprietà che devono essere rispettate da ogni record della base di dati. Il costrutto più potente per specificare vincoli generici, è il costrutto *check*, che richiede di formulare delle interrogazioni alla base di dati.

nameAttribute smallint check((attribute >= number1) and (attribute <= number2))

vincolo applicato per ogni *ennupla*. I più semplici vincoli adottabili sono di tipo *intra-relazionale*, quali *not null*, *unique* e *primary key*.

Not Null
Il valore *NULL* è un particolare valore adottato in assenza di informazioni. Tuttavia il vincolo stabilito non ammette il valore *nullo* come valore dell'attributo, quindi il dato dovrà essere sempre specificato, indipendentemente dall'istanza che si voglia inserire. Invece se l'attributo in questione è combinato rispetto ad un valore di *default* diverso dal valore nullo, allora diventa possibile inserire l'istanza anche in assenza di specifica a livello di inserimento. Il vincolo è specificato attribuendo alla definizione dell'attributo la dichiarazione *not null*, come segue:

nameAttribute varchar(20) not null

Unique
Il vincolo *unique* impone che l'attributo o l'insieme di attributi su cui si applica non possa / possano avere duplicati, ossia stessi valori in istanze differenti, pertanto gli attributi in questione segueno regole definite dal concetto di *superchiave*, per cui essi stessi sono delle superchiavi. Il vincolo può essere espresso in due sintassi, dove la prima delle due descrive:

nameAttribute Domain [ValDefault] unique

qualora la superchiave della relazione sia un unico attributo, oppure quando è composta da più attributi si adotta solitamente

unique(nameAttribute1, nameAttribute2, ...)

concludendo, quest'ultima sintassi è riportata al termine del costrutto per la creazione della tabella. Tuttavia, le due sintassi non hanno la stessa valenza, poichè specificare il vincolo per ogni attributo piuttosto che raggruppare il tutto come la seconda sintassi non persegue nello stesso obiettivo. Infatti, rispettivamente, il primo citato indica che la specifica colonna non possa avere duplicati al suo interno, mentre la definizione *unique(nameAttribute1, nameAttribute2, ...)* indica che l'insieme dei domini riportati per tutte le istanze seguenti non possano avere valori combinati duplicati.

Primary key
Il vincolo *primary key* può essere ugualmente posto come il vincolo *unique*, ma con alcune differenze a livello di implementazione. Tale proprietà ammette che gli attributi associati non possano mai assumere il valore nullo, inoltre l'uso di questo vincolo può avvennire una sola volta, al contrario di unique e not null. In relazione a quanto detto, la sintassi del costrutto è:

create table nameTable(

nameAttribute1 Domain [ValDefault]
nameAttribute2 Domain [ValDefault]
primary key (nameAttribute1, nameAttribute2)

);

Vincoli inter-relazionali

I vincoli inter-relazionali più diffusi sono i *vincoli di integrità referenziale*. In SQL spesso sono denominati come *vincoli* di *foreign key*, ossia di *chiave esterna*. Questa proprietà crea un collegamento tra una tabella *referenziante* e una tabella *referenziata*. Il vincolo impone che ogni riga che compare nella tabella referenziata di un attributo specificato, compaia anche nelle righe della tabella referenziante in un totale di attributi di simile dominio. L'unico requisito imposto richiede che l'attributo o l'insieme di attributi di riferimento, posti internamente alla tabella referenziata, siano soggetti al vincolo *unique*, ovvero sia un identificatore della relazione. Il vincolo può essere posto sia come vincolo *unique* oppure come vincolo *primary key*. Per cui seguono due costrutti di differente sintassi:

create table nameTable1(

nameAttribute1 Domain [ValDefault] references nameTable2(nameAttribute1)
nameAttribute2 Domain [ValDefault]
primary key (nameAttribute1, nameAttribute2)

);

dove si osserva la nuova semantica *references* utilizzabile solo quando un singolo attributo risulta essere coinvolto per la costruzione della referenza, mentre qualora sia composta da più di un singolo attributo si adotta

create table nameTable1(

nameAttribute1 Domain [ValDefault] primary key
nameAttribute2 Domain [ValDefault]
foreign key (nameAttribute1) references nameTable2(nameAttribute1)

);

uso del costrutto *foreign key* per la costruzione è posto sempre al termine della definizione degli attributi, inoltre qualora siano specificati tra le parentesi un numero superiore di attributi per la referenza, SQL segue lo stesso ordine imposto sia per la tabella referenziante e sia per la tabella referenziata.