Corso di Laurea in Ingegneria Informatica "*Basi di dati*" a.a. 2019-2020

Docente: Gigliola Vaglini

Docente laboratorio SQL: Francesco Pistolesi

1

Lezione 5

Vincoli

2

2

CREATE TABLE (1)

```
CREATE TABLE Impiegato(
   Matricola CHAR(6) PRIMARY KEY,
   Nome CHAR(20) NOT NULL,
   Cognome CHAR(20) NOT NULL,
   Dipart CHAR(15),
   Stipendio NUMERIC(9) DEFAULT 0,
   FOREIGN KEY(Dipart) REFERENCES
              Dipartimento(NomeDip),
   UNIQUE (Cognome,Nome)
 )
```

3

CREATE TABLE (2)

```
CREATE TABLE Infrazioni(
   Codice CHAR(6) PRIMARY KEY,
   Data DATE NOT NULL,
   Vigile INTEGER NOT NULL
              REFERENCES Vigili(Matricola),
   Provincia CHAR(2),
   Numero CHAR(6) ,
   FOREIGN KEY(Provincia, Numero)
              REFERENCES Auto(Provincia, Numero)
              ON DELETE SET NULL
              ON UPDATE CASCADE )
```

4

Vincoli di integrità generici: check

- La clausola check permette di restringere i domini e specificare predicati che devono essere soddisfatti ogni volta che un valore viene assegnato ad una variabile in quel dominio.

5

5

Vincoli di integrità generici: check

- Specifica di vincoli su attributi a livello di tabella
- check (Condizione)
- La condizione è del tipo che può comparire in una clausola where

6

6

Esempi

```
create table Impiegato
   (
   Matricola character(6),
   Cognome character(20),
   Nome character(20),
   Dipart character(6),
   Ufficio character(6),
   Sesso character not null check (sesso in ('M','F'))
   Superiore character(6),
   Stipendio integer,
   check (Stipendio <= (select J.Stipendio
   from Impiegato as J
   where J.Matricola=Superiore)
   )
```

Non sempre supportata

7

7

Sintassi, dettagli

- Sesso character not null check (sesso in ('M','F'))
  - Nella condizione è coinvolto un solo attributo
- Stipendio integer,
  check (Stipendio <= (select J.Stipendio
  from Impiegato as J
  where J.Matricola= Superiore))
  - La condizione coinvolge più attributi

8

8

CREATE DOMAIN, esempio

```
CREATE DOMAIN Voto
  AS SMALLINT DEFAULT NULL
  CHECK ( value >=18 AND value <= 30 )
```

9

Vincoli di integrità generici: asserzioni

- Specifica vincoli a livello di schema della base di dati (cioè tra più tabelle)
- create assertion NomeAss
  check (Condizione)
- Invece di mettere il vincolo in una sola delle tabelle coinvolte (poco chiaro) o in tutte (pericoloso), si mette allo stesso livello della definizione delle tabelle

10

10

Esempi

- create assertion AlmenoUnImpiegato check (1 <= (select count(*) from Impiegato))

Anche questa non sempre supportata

11

Asserzioni, commenti

- Con le asserzioni è possibile stabilire anche un qualunque vincolo predefinito
- Quando un'asserzione è stabilita, ogni variazione del database è consentita solo se non la viola
- Ad ogni asserzione è associata una politica di controllo che può essere di due tipi:
  - Immediato (da verificare dopo ogni modifica)
  - Differito (da verificare dopo una sequenza di operazioni = transazione)

12

Controllo dei vincoli

- Il costrutto per determinare (o per cambiare) il tipo di controllo associato ad un vincolo è il seguente
- set constraints [NomeAss] (immediate | deferred)

13

13

Asserzioni, commenti

- Un vincolo immediato non soddisfatto causa l'annullamento dell'operazione di modifica che ha causato la violazione (rollback parziale)
- Un vincolo differito non soddisfatto causa l'annullamento della transazione che ha prodotto la violazione (rollback)

14

14

Asserzioni, commenti

- I vincoli predefiniti sono di tipo immediato e possono essere quindi rappresentati da asserzioni con associata una politica di controllo immediato
- N.B. ad una asserzione non può però essere associata una politica di reazione alle violazioni come succede per i vincoli predefiniti

15

15

Asserzioni

- Le asserzione hanno un nome e quindi possono comparire all'interno di un'istruzione, ad es.
- drop NomeAss

16

16

Basi di dati attive

- Una base di dati può contenere regole attive (chiamate *trigger*)
- Paradigma: Evento-Condizione-Azione
  - Quando un evento si verifica
  - Se la condizione è vera
  - Allora l'azione è eseguita
- Questo modello consente computazioni reattive
- Problema: è difficile realizzare applicazioni complesse

17

Evento-Condizione-Azione

- Evento
  - Normalmente una modifica dello stato del database: insert, delete, update
  - Quando accade l'evento, il trigger è *attivato*
- Condizione
  - Un predicato che identifica se l'azione del trigger deve essere eseguita
  - Quando la condizione viene valutata, il trigger è *considerato*
- Azione
  - Una sequenza di update SQL o una procedura
  - Quando l'azione è eseguita anche il trigger è *eseguito*

18

SQL:1999, Sintassi

- Lo standard SQL:1999 (SQL-3) sui trigger è stato fortemente influenzato da DB2 (IBM); gli altri sistemi non seguono lo standard (esistono dagli anni 80')
- Ogni trigger è caratterizzato da:
  - nome
  - target (tabella controllata)
  - modalità (**before** o **after**)
  - evento (**insert**, **delete** o **update**)
  - granularità (statement-level o row-level)
  - alias dei valori o tabelle di transizione
  - Azione
  - Timestamp di creazione

19

SQL:1999, Sintassi

create trigger *NomeTrigger*
{ **before** | **after** }
{ **insert** | **delete** | **update** [**of** *Column*] } **on** *TabellaTarget*
[**referencing**
{[**old table** [**as**] *VarTuplaOld*]
[**new table** [**as**] *VarTuplaNew*] } |
{[**old** [**row**] [**as**] *VarTabellaOld*]
[**new** [**row**] [**as**] *VarTabellaNew*] }]
[**for each** { **row** | **statement** }]
[**when** *Condizione*]

20

Tipi di eventi

- BEFORE
  - Il trigger è considerato e possibilmente eseguito prima dell'evento (i.e., la modifica del database)
  - I trigger before non possono modificare lo stato del database; possono al più condizionare i valori "new" in modalità row-level (set t.new=expr)
  - Normalmente questa modalità è usata quando si vuole verificare una modifica prima che essa avvenga e "modificare la modifica"
- AFTER
  - Il trigger è considerato e eseguito dopo l'evento
  - E' la modalità più comune, adatta alla maggior parte delle applicazioni

21

Esempio "before" e "after"

- 1. "Conditioner" (agisce prima dell'update e della verifica di integrità)

```
create trigger LimitaAumenti
before update of Salario on
  Impiegato
for each row
when (New.Salario > Old.Salario *
  1.2)
set New.Salario = Old.Salario *
  1.2
```

22

Esempio “before” e “after”

- 2. “Re-installer” (agisce dopo l’update)

```
create trigger LimitaAumenti
after update of Salario on Impiegato
for each row
when (New.Salario > Old.Salario * 1.2)
set New.Salario = Old.Salario * 1.2
```

23

23

Granularità degli eventi

- Modalità statement-level (di default, opzione **`for each statement`**)
  - Il trigger viene considerato e possibilmente eseguito solo una volta per ogni statement (comando) che lo ha attivato, independentemente dal numero di tuple modificate
  - In linea con SQL (set-oriented)

24

Granularità degli eventi

- Modalità row-level (opzione **`for each row`**)
  - Il trigger viene considerato e possibilmente eseguito una volta per ogni tupla modificata
  - Scrivere trigger row-level è più semplice

25

Clausola `referencing`

- Dipende dalla granularità
  - Se la modalità è row-level, ci sono due *variabili di transizione* (**`old`** and **`new`**) che rappresentano il valore precedente o successivo alla modifica di una tupla
  - Se la modalità è statement-level, ci sono due *tabelle di transizione* (**`old table`** and **`new table`**) che contengono i valori precedenti e successivi delle tuple modificate dallo statement
- **`old`** e **`old table`** non sono presenti con l'evento **`insert`**
- **`new`** e **`new table`** non sono presenti con l'evento **`delete`**

26

Esempio di trigger row-level

```
create trigger AccountMonitor
  after update on Account
  for each row
  when new.Totale > old.Totale
  insert values
      (new.NumeroConto,
       new.Totale-old.Totale)
      into Pagamenti
```

27

Esempio di trigger statement-level

```
create trigger ArchiviaFattureCancellate
  after delete on Fattura
  referencing old_table as VecchieFatture
  insert into FattureCancellate
  (select *
   from VecchieFatture)
```

28

Esecuzione di Trigger in conflitto

- Quando vi sono più trigger associati allo stesso evento (in conflitto) vengono eseguiti come segue:
  - Per primi i **`before`** triggers (*statement-level* e *row-level*)
  - Poi viene eseguita la modifica e verificati i vincoli di integrità
  - Infine sono eseguiti gli **`after`** triggers (*row-level* e *statement level*)
- Quando vari trigger appartengono alla stessa categoria, l'ordine di esecuzione è definito in base al loro timestamp di creazione (i trigger più vecchi hanno priorità più alta)

29

Controllo dell'accesso

- In SQL è possibile specificare chi (utente) e come (lettura, scrittura, …) può utilizzare la base di dati (o parte di essa)
- Oggetto dei privilegi (diritti di accesso) sono di solito le tabelle, ma anche altri tipi di risorse, quali singoli attributi, viste o domini
- Un utente predefinito _system (amministratore della base di dati) ha tutti i privilegi
- Il creatore di una risorsa ha tutti i privilegi su di essa

30

Privilegi

- Un privilegio è caratterizzato da:
  - la risorsa cui si riferisce
  - l'utente che concede il privilegio
  - l'utente che riceve il privilegio
  - l'azione che viene permessa
  - la trasmissibilità del privilegio

31

Tipi di privilegi offerti da SQL

- insert: permette di inserire nuovi oggetti (ennuple)
- update: permette di modificare il contenuto
- delete: permette di eliminare oggetti
- select: permette di leggere la risorsa
- references: permette la definizione di vincoli di integrità referenziale verso la risorsa (può limitare la possibilità di modificare la risorsa)
- usage: permette l'utilizzo in una definizione (per esempio, di un dominio)

32

grant e revoke

- Concessione di privilegi:

  grant < Privileges | all privileges > on Resource
  to Users [with grant option]

  – grant option specifica se il privilegio può essere trasmesso ad altri utenti

  grant select on Department to Stefano

- Revoca di privilegi

  revoke Privileges on Resource from Users
  [restrict | cascade]

33

Autorizzazioni, commenti

- La gestione delle autorizzazioni deve "nascondere" gli elementi cui un utente non può accedere
- Come autorizzare un utente a vedere solo alcune ennuple di una relazione?
  - Attraverso una vista:
    - Definiamo la vista con una condizione di selezione
    - Attribuiamo le autorizzazioni sulla vista, anziché sulla relazione di base

34

Autorizzazioni, ancora

- (Estensioni di SQL:1999)
- Concetto di ruolo, cui si associano privilegi (anche articolati), poi concessi agli utenti attribuendo loro il ruolo

35

Transazione

- Insieme di operazioni da considerare indivisibile ("atomico"), corretto anche in presenza di concorrenza e con effetti definitivi
- Proprietà ("acide"):
  - Atomicità
  - Consistenza
  - Isolamento
  - Durabilità (persistenza)

36

Proprietà

- La sequenza di operazioni sulla base di dati viene eseguita per intero o per niente
- Al termine di una transazione, i vincoli di integrità debbono essere soddisfatti. "Durante" l'esecuzione ci possono essere violazioni, ma se ci sono alla terminazione allora la transazione deve essere annullata per intero ("abortita")

37

Proprietà (2)

- L'effetto di transazioni concorrenti deve essere coerente (ad esempio "equivalente" alla loro esecuzione separata)
- La conclusione positiva di una transazione corrisponde ad un impegno (commit) a mantenere traccia del risultato anche in presenza di guasti e di esecuzione concorrente

38

Transazioni in SQL

- Istruzioni fondamentali
  - begin transaction: specifica l'inizio della transazione (le operazioni non vengono eseguite sulla base di dati)
  - commit work: le operazioni specificate a partire dal begin transaction vengono eseguite sulla base di dati
  - rollback work: si rinuncia all'esecuzione delle operazioni specificate dopo l'ultimo begin transaction

39

Procedure

- SQL:1999 (come già SQL-2) permette la definizione di procedure e funzioni (chiamate genericamente "stored procedures")
- Le stored procedures sono parte dello schema

```
procedure AssignCity(:Dep char(20), :City char(20))
   update Department
   set City = :City
   where Name = :Dep
```

- Lo standard prevede funzionalità limitate, le procedure sono composte da un singolo comando SQL, e non è molto recepito
- Molti sistemi offrono estensioni ricche (ad esempio Oracle PL/SQL e Sybase-Microsoft Transact SQL)

40

SQL nei linguaggi di programmazione

41

SQL e applicazioni

- In applicazioni complesse, l'utente non vuole eseguire solo comandi SQL, ma programmi
- SQL non basta, sono necessarie altre funzionalità, per gestire:
  - input (scelte dell'utente e parametri)
  - output (con dati che non sono relazioni o se si vuole una presentazione complessa)
  - il controllo

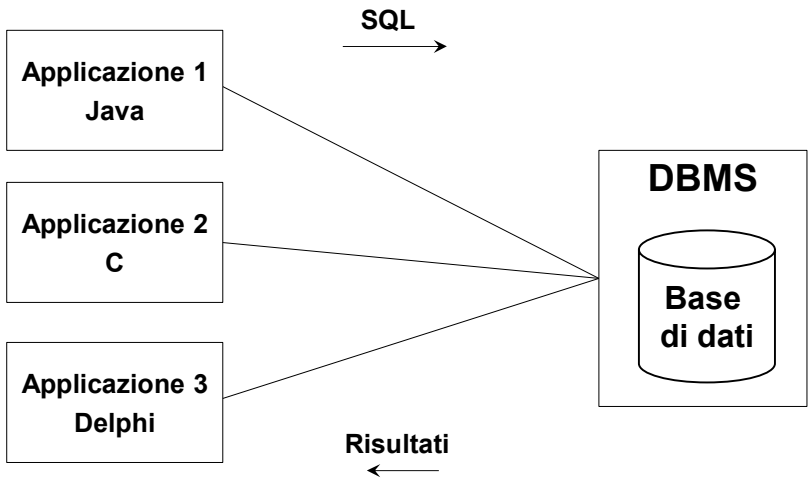
42

SQL e linguaggi di programmazione

- Le applicazioni sono scritte in
  - linguaggi di programmazione tradizionali:
    - Cobol, C, Java, Fortran
  - linguaggi "ad hoc", proprietari e non:
    - PL/SQL, Informix4GL, Delphi

43

Applicazioni ed SQL: architettura



44

Differenze, 1

- Accesso ai dati e correlazione:
  - linguaggio: dipende dal paradigma e dai tipi disponibili; ad esempio scansione di liste
  - SQL: join

45

Differenze, 2

- tipi di base:
  - linguaggi: numeri, stringhe, booleani
  - SQL: CHAR, VARCHAR, DATE, ...
- costruttori di tipo:
  - linguaggio: dipende dal paradigma
  - SQL: relazioni e ennuple

46

Tecniche principali

- SQL immerso ("Embedded SQL")
  - sviluppata sin dagli anni '70
  - "SQL statico"
- SQL dinamico
- Call Level Interface (CLI)
  - più recente
  - SQL/CLI, ODBC, JDBC

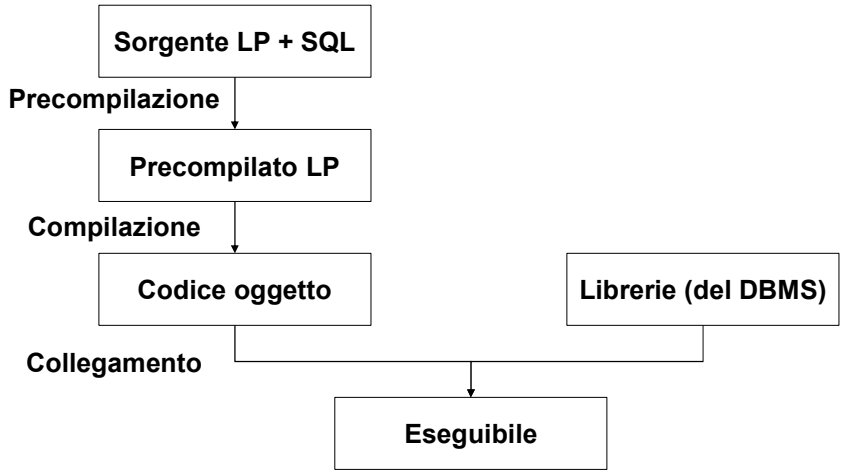
47

SQL immerso

- le istruzioni SQL sono "immerse" nel programma redatto nel linguaggio "ospite"
- un precompilatore (legato al DBMS) viene usato per analizzare il programma e tradurlo in un programma nel linguaggio ospite (sostituendo le istruzioni SQL con chiamate alle funzioni di una API del DBMS)

48

SQL immerso, fasi



49

Note

- Il precompilatore è specifico della combinazione

  linguaggio-DBMS-sistema operativo

50

SQL immerso, un esempio

```
#include<stdlib.h>
main(){
  exec sql begin declare section;
    char *NomeDip = "Manutenzione";
    char *CittaDip = "Pisa";
    int NumeroDip = 20;
  exec sql end declare section;
  exec sql connect to utente@librobd;
  if (sqlca.sqlcode != 0) {
    printf("Connessione al DB non riuscita\n"); }
  else {
    exec sql insert into Dipartimento
           values(:NomeDip,:CittaDip,:NumeroDip);
    exec sql disconnect all;
  }
}
```

51

SQL immerso, commenti

- EXEC SQL denota le porzioni di interesse del precompilatore:
  - definizioni dei dati
  - istruzioni SQL
- le variabili del programma possono essere usate come "parametri" nelle istruzioni SQL (precedute da ":") dove sintatticamente sono ammesse costanti

52

SQL immerso, commenti 2

- **`sqlca`** è una struttura dati per la comunicazione fra programma e DBMS
- **`sqlcode`** è un campo di **`sqlca`** che mantiene il codice di errore dell'ultimo comando SQL eseguito:
  - zero: successo
  - altro valore: errore o anomalia

53

Una difficoltà importante

- Conflitto di impedenza ("impedance mismatch")
  - Linguaggi di programmazione: operazioni su singole variabili o oggetti, si può accedere agli elementi di una tabella uno alla volta
  - SQL: operazioni su tabelle che restituiscono tabelle

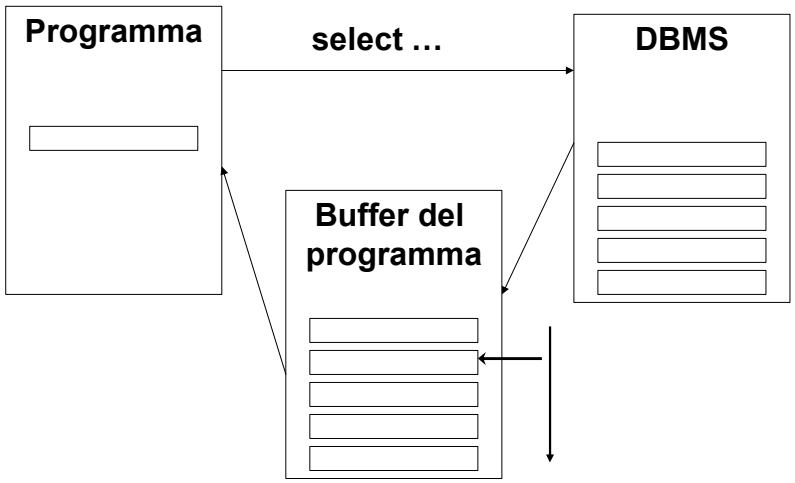
54

Interrogazioni in SQL immerso: conflitto di impedenza

- Il risultato di una select è costituito da zero o piú ennuple:
  - zero o una: ok -- l'eventuale risultato puó essere gestito in un record
  - piú ennuple: come facciamo?
    - l'insieme (in effetti, la lista) non è gestibile facilmente in molti linguaggi
- Cursore: tecnica per trasmettere al programma una ennupla alla volta

55

Cursore



56

Nota

- Il cursore
  - accede a tutte le ennuple di una interrogazione in modo globale (tutte insieme o a blocchi – è il DBMS che sceglie la strategia efficiente)
  - trasmette le ennuple al programma una alla volta

57

Operazioni sui cursori

Definizione del cursore
  declare NomeCursore [scroll] cursor for Select …
Esecuzione dell'interrogazione
  open NomeCursore
Utilizzo dei risultati (una ennupla alla volta)
  fetch NomeCursore into ListaVariabili
Disabilitazione del cursore
  close cursor NomeCursore
Accesso alla ennupla corrente (di un cursore su singola relazione a fini di aggiornamento)
  current of NomeCursore
  nella clausola where

58

Cursori, commenti

- Per aggiornamenti e interrogazioni "scalari" (cioè che restituiscano una sola ennupla) il cursore non serve
- I cursori possono far scendere la programmazione ad un livello troppo basso, pregiudicando la capacità dei DBMS di ottimizzare le interrogazioni:
  - se "nidifichiamo" due o più cursori, rischiamo di reimplementare il join!

59

SQL dinamico

- Non sempre le istruzioni SQL sono note quando si scrive il programma
- Allo scopo, è stata definita una tecnica completamente diversa, chiamata *Dynamic SQL* che permette di eseguire istruzioni SQL costruite dal programma (o addirittura ricevute dal programma attraverso parametri o da input)
- Non è banale gestire i parametri e la struttura dei risultati (non noti a priori)

60

SQL dinamico

- Le operazioni SQL possono essere:
  - eseguite immediatamente

    `execute immediate` ***SQLStatement***
  - prima "preparate":

    `prepare` ***CommandName*** **`from`** ***SQLStatement***

    e poi eseguite (anche più volte):

    `execute` ***CommandName*** **`[ into`** ***TargetList*** **`] [ using`** ***ParameterList*** **`]`**

61

Call Level Interface

- Indica genericamente interfacce che permettono di inviare richieste a DBMS per mezzo di parametri trasmessi a funzioni
- standard SQL/CLI ('95 e poi parte di SQL:1999)
- ODBC: implementazione proprietaria di SQL/CLI
- JDBC: una CLI per il mondo Java

62

SQL immerso vs CLI

- SQL immerso permette
  - precompilazione (e quindi efficienza)
  - uso di SQL completo
- CLI
  - indipendente dal DBMS
  - permette di accedere a più basi di dati, anche eterogenee

63