Basi di dati

SQL: Caratteristiche evolute

Vincoli di integrità generici: check

- Specifica di vincoli di ennupla (e anche vincoli più complessi, non sempre supportati)

  check (Condizione)

Check, esempio

```
create table Impiegato
(
Matricola integer primary key,
Cognome character(20),
Nome character(20),
Sesso character not null check (sesso in ('M','F')) ,
Stipendio integer check (Stipendio > 0) ,
Superiore integer,
check (Stipendio <= (select Stipendio
                     from Impiegato J
                     where Superiore = J.Matricola) )
)
```

La nidificazione nel check non è supportata da tutti i sistemi

Check, esempio 2

```
create table Impiegato
(
Matricola character(6),
Cognome character(20),
Nome character(20),
Sesso character not null check (sesso in ( ‘M’ , ‘F’ ))
Stipendio integer,
Ritenute integer,
Netto integer,
Superiore character(6),
check (Netto = Stipendio - Ritenute )
)
```

ok

Check, esempio 3

```
insert into Impiegato values
(1 , 'Rossi', 'Mario', '', 100, 20, 80);
```

Non soddisfa il check sull'attiributo sesso

```
insert into Impiegato values
(2 , 'Neri', 'Mario', 'M', 100, 10, 80);
```

Non soddisfa il check sull'attributo netto

```
insert into Impiegato values
(3 , 'Rossini', 'Luca', 'M', 70, 20, 50);
```

Soddisfa tutti i requisiti del check

Vincoli di integrità generici: asserzioni

- Specifica vincoli a livello di schema

```
create assertion NomeAss check ( Condizione )

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

La nidificazione nel check non è supportata da tutti i sistemi

Viste

```
create view NomeVista [ ( ListaAttributi ) ] as SelectSQL
[ with [ local | cascaded ] check option ]

create view ImpiegatiAmmin
        (Nome, Cognome, Stipendio) as
      select Nome, Cognome, Stipendio
      from Impiegato
      where Dipart = 'Amministrazione' and
           Stipendio > 10
```

Interrogazioni sulle viste

- Possono fare riferimento alle viste come se fossero relazioni di base

```
select * from ImpiegatiAmmin
```

equivale a (e viene eseguita come)

```
select Nome, Cognome, Stipendio
from Impiegato
where Dipart = 'Amministrazione' and
      Stipendio > 10
```

Aggiornamenti sulle viste

- Ammessi (di solito) solo su viste definite a partire da una sola relazione
- Alcune verifiche possono essere imposte

Visite con modifiche: check option

```
create view ImpiegatiAmminPoveri as
select *
from ImpiegatiAmmin
where Stipendio < 50
with check option
```

- check option permette modifiche sulle viste, ma solo a condizione che la ennupla continui ad appartenere alla vista (non posso modificare lo stipendio portandolo a 60)

Esempio: operazione non consentita

```
create view ImpiegatiAmminPoveri as
select *
from ImpiegatiAmmin
where Stipendio < 50
with check option

update ImpiegatiAmminPoveri
    set stipendio = 60
    where nome = 'Paola'
```

Opzioni su viste: local e cascaded

- local (nel caso di viste su viste) specifica se il controllo sul fatto che le righe vengono aggiornate dalla vista debba essere effettuato solo all'ultimo livello della vista

- cascaded (nel caso di viste su viste) specifica che l'aggiornamento della vista debba essere propagato

Un'interrogazione scorretta per lo standard SQL

- Estrarre il numero medio di uffici per ogni dipartimento

- Interrogazione scorretta

```
select avg(count(distinct Ufficio))
from Impiegato
group by Dipart
```

  - L' interrogazione è scorretta in quanto la sintassi SQL non permette di combinare in cascata la valutazione di diversi operatori aggregati.

Un'interrogazione corretta per lo standard SQL

- Estrarre il numero medio di uffici per ogni dipartimento

- Con vista

```
create view DipartUffici(NomeDip,NroUffici) as
select Dipart, count(distinct Ufficio)
from Impiegato
group by Dipart;

select avg(NroUffici)
from DipartUffici
```

Ancora sulle viste, soluzione scorretta

- Mostrare il Dipartimento che spende la somma massima in stipendi
- Soluzione scorretta su alcuni sistemi

```
select Dipart
from Impiegato
group by Dipart
having sum(Stipendio) >= all
                        (select sum(Stipendio)
                         from Impiegato
                         group by Dipart)
```

- La nidificazione nell'having non è ammessa in alcuni sistemi

Soluzione corretta con le viste

```
create view BudgetStipendi(Dip,TotaleStipendi) as
select Dipart, sum(Stipendio)
from Impiegato
group by Dipart

select Dip
from BudgetStipendi
where TotaleStipendi =(select max(TotaleStipendi)
                       from BudgetStipendi)
```

Viste ricorsive (1)

- Per ogni persona, trovare tutti gli antenati, avendo
  Paternita (Padre, Figlio)

Paternita

Padre	Figlio
Sergio	Franco
Luigi	Olga
Luigi	Filippo
Franco	Andrea
Franco	Aldo

Viste ricorsive (2)

- Serve la ricorsione; in Datalog:

Discendenza (Antenato: p, Discendente: f) ←
    Paternita (Padre: p, Figlio: f)

Discendenza (Antenato: a, Discendente: d) ←
    Paternita (Padre: a, Figlio: f) ,
    Discendenza (Antenato: f, Discendente: d)

Viste ricorsive in SQL:1999

```
with recursive Discendenza(Antenato,Discendente) AS
(       select Padre, Figlio
        from Paternita
   union all
        select Antenato, Figlio
        from Discendenza, Paternita
        where Discendente = Padre)

select *
from Discendenza
```

- La clausola with definisce la vista Discendenza che viene costruita ricorsivamente a partire dalla tabella Paternita.

Esempio di interrogazione ricorsiva

Estrarre i superiori diretti o indiretti dell'impiegato Mario Rossi

```
with recursive Responsabile (Matr, Superiore) AS
(           select Matr, Superiore
            from Impiegato
union
            select Impiegato.Matr, Responsabile.Superiore
            from Impiegato, Responsabile
            where Impiegato.Superiore = Responsabile.Matr)

select Nome, Cognome, Responsabile.Superiore
from Impiegato join Responsabile
            on (Impiegato.Matr = Responsabile.Matr)
where Nome = 'Mario' and Cognome = 'Rossi'
```

Funzioni scalari (1)

- Funzioni a livello di ennupla che restituiscono singoli valori

- Temporali
  - current_date, ti estrae la data corrente
  - extract(year from expression), estrae una porzione di data/ora da un'espressione (month, day, hour, ecc)

Funzioni scalari (2)

- Manipolazione stringhe
  - char_length, restituisce la lunghezza della stringa
  - lower , converte la stringa in caratteri minuscoli

- Conversione
  - Cast permette di convertire un valore in un dominio nella sua rappresentazione in un altro dominio;

- Condizionali
  - …

Funzioni condizionali: coalesce (1)

La funzione coalesce ammette come argomento una sequenza di espressioni e restituisce il primo valore non nullo tra questi.

- Esempio: dato il seguente schema relazionale:
  Impiegato (matricola, dipartimento, cellulare, telefono fisso)

  Trovare un recapito telefonico valido per ogni impiegato dove, se presente usare cellulare, altrimenti il telefono fisso.

```
Select matricola, coalesce (cellulare, telefono fisso)
from impiegato
```

Funzioni condizionali: coalesce (2)

La funzione coalesce può essere usata per convertire valori nulli in valori espliciti.

- Esempio: Estrarre i nomi, i cognomi ed i dipartimenti cui afferiscono gli impiegati, usando la stringa "Ignoto" nel caso in cui non si conosca il dipartimento.

```
select Nome, Cognome, coalesce(Dipart, 'Ignoto')
from Impiegato
```

Funzioni condizionali: nullif

La funzione nullif richiede due argomenti. Esamina il primo argomento (solitamente una colonna di una relazione) e lo confronta con il secondo (solitamente un valore costante). Se i due valori sono uguali restituisce il valore nullo, altrimenti restituisce il valore del primo argomento.

- Esempio: Estrarre i cognomi e i dipartimenti cui afferiscono gli Impiegati, restituendo il valore nullo per il dipartimento quando l'attributo Dipart possiede il valore "Ignoto"

```
select Cognome, nullif(Dipart, 'Ignoto')
from Impiegato
```

Funzioni condizionali: case(1)

La funzione case permette di specificare strutture condizionali, il cui risultato dipende dalla valutazione del contenuto delle tabelle.

Viene utilizzata per fornire il tipo di logica if-then-else (se allora-altrimenti) al linguaggio SQL.

Funzioni condizionali: case(2)

- Esempio: dato il seguente schema relazionale:
  VEICOLO(Targa, Tipo, Anno, KWatt, Lunghezza, NAssi)

  Calcolare l'ammontare delle tasse di circolazione, in base al tipo di veicolo e con immatricolazione dopo il 1975.

```
select Targa,
       case Tipo
                when 'Auto' then 2.58 * KWatt
                when 'Moto' then (22.00 + 1.00 * KWatt)
                else null
       end as Tassa
  from Veicolo
  where Anno > 1975
```

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

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

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)

Concedere privilegi: grant

- 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

Revocare privilegi: revoke

- Revoca di privilegi

  revoke *Privileges* on *Resource* from *Users*
  [restrict | cascade]

La revoca deve essere fatta dall'utente che aveva concesso

i previlegi

- restrict (di default) specifica che il comando non deve essere eseguito qualora la revoca dei privilegi all'utente comporti qualche altra revoca (dovuta ad un precedente grant option)
- cascade invece forza l'esecuzione del comando

Attenzione alle reazioni a catena

Autorizzazioni, commenti

- La gestione delle autorizzazioni deve "nascondere" gli elementi cui un utente non può accedere, senza sospetti
- Esempio:
    - La tabella Impiegati non esiste
    - La tabella Impiegati esiste, ma l'utente non è autorizzato
  - L'utente deve ricevere lo stesso messaggio

Autorizzazioni, commenti, 2

- 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

Autorizzazioni: RBAC(1)

SQL-3 ha introdotto una novità nell'ambito del controllo dell'accesso proponendo un modello di controllo basato sui ruoli (Role-Based Access Control, RBAC).

- Il ruolo si comporta come una sorta di contenitore di privilegi, che vengono attribuiti ad esso tramite il comando grant.

- In ogni istante un utente dispone quindi dei privilegi che gli sono stati attribuiti direttamente e dei privilegi associati al ruolo che è stato esplicitamente attivato.

Autorizzazioni: RBAC(2)

- E’ possibile creare un ruolo tramite un opportuno comando

create role *NomeRuolo*

- Per usufruire però dei privilegi è necessario che l'utente invochi un esplicito comando

set role *nomeRuolo*

Esempio RBAC

- concedere il privilegio CREATE TABLE ad un utente creando il ruolo di impiegato:

  Per prima cosa creo un ruolo “impiegato”:

  create role *impiegato;*

  Poi concedo i privilegi di create table al ruolo:

  grant create table to *impiegato*;

  Infine concedo il ruolo a un utente:

  grant *impiegato* to *user1*;

  Invece per revocare il privilegio a un determinato ruolo:

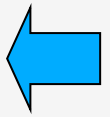
  revoke create table from *impiegato*;

Transazioni

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

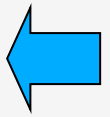
Le transazioni sono … atomiche

- La sequenza di operazioni sulla base di dati viene eseguita per intero o per niente:
  - trasferimento di fondi da un conto A ad un conto B: o si fanno il prelevamento da A e il versamento su B o nessuno dei due



Le transazioni sono … consistenti

- Al termine dell'esecuzione di una transazione, i vincoli di integrità debbono essere soddisfatti
- "Durante" l'esecuzione ci possono essere violazioni, ma se restano alla fine allora la transazione deve essere annullata per intero ("abortita")



Le transazioni sono … isolate

- L'effetto di transazioni concorrenti deve essere coerente (ad esempio "equivalente" all'esecuzione separata)
  - se due assegni emessi sullo stesso conto corrente vengono incassati contemporaneamente si deve evitare di trascurarne uno

I risultati delle transazioni sono durevoli

- La conclusione positiva di una transazione corrisponde ad un impegno (in inglese commit) a mantenere traccia del risultato in modo definitivo, anche in presenza di guasti (hardware e software) e di esecuzione concorrente

Transazioni in SQL

- Una transazione inizia al primo comando SQL dopo la "connessione" alla base di dati oppure alla conclusione di una precedente transazione (lo standard indica anche un comando start transaction, non obbligatorio, e quindi non previsto in molti sistemi)
- Conclusione di una transazione
  - commit [work]: le operazioni specificate a partire dall'inizio della transazione vengono eseguite sulla base di dati
  - rollback [work]: l'utente della base di dati può annullare gli effetti del lavoro svolto dalla transazione, indipendentemente dalla sua complessità
- Molti sistemi prevedono una modalità autocommit, in cui ogni operazione forma una transazione

Una transazione in SQL

- Trasferire dal conto 42177 al conto 12202 l'ammontare 10.

```
start transaction                          (opzionale)
  update ContoCorrente
       set Saldo = Saldo – 10
       where NumeroConto = 12202 ;
  update ContoCorrente
       set Saldo = Saldo + 10
       where NumeroConto = 42177;
  commit work;
```