BD 2 - Vincoli di Integrità

Luca Cosmo

Università Ca' Foscari Venezia

Università
Ca'Foscari
Venezia

Introduzione

Molto spesso i dati salvati all'interno di un database devono soddisfare determinati vincoli di integrità, dipendenti dalla semantica dei dati.

1. Garantire che certi attributi abbiano sempre un valore (NOT NULL)
2. Garantire che un certo insieme di attributi sia una chiave (PRIMARY KEY, UNIQUE)
3. Garantire l'integrità referenziale (vincoli su FOREIGN KEY)
4. Garantire determinati vincoli sui valori degli attributi, anche in relazione tra loro

Example

- Garantire che l'età di una persona sia sempre un numero positivo
- Garantire che il primario di un ospedale sia anche un dottore

Che strumenti ci vengono messi a disposizione dai DBMS?

NOT NULL

Il più semplice vincolo che possiamo esprimere è che un certo attributo non deve essere mai impostato a NULL.

Example

```
CREATE TABLE Movies (
    title       CHAR(100) NOT NULL,
    year        INT,
    length      INT,
    genre       CHAR(10)
)
```

Chiavi - UNIQUE

Data una tabella $R(T)$ ed un insieme di attributi $X \subseteq T$, possiamo specificare che nessuna coppia di tuple in $R(T)$ coincida su tutti gli attributi in X, a meno che almeno uno di essi non sia NULL.

Example

```
CREATE TABLE Movies (
    title       CHAR(100) NOT NULL,
    year        INT,
    length      INT,
    genre       CHAR(10),
    UNIQUE (title, year)
)
```

PRIMARY KEY

Il vincolo PRIMARY KEY si comporta come UNIQUE, ma impone in aggiunta il vincolo NOT NULL per tutti gli attributi specificati.

Example

```
CREATE TABLE Movies (
    title       CHAR(100),
    year        INT,
    length      INT,
    genre       CHAR(10),
    PRIMARY KEY (title, year)
)
```

FOREIGN KEY

Dati una tabella $R(T)$ ed $X \subseteq T$, possiamo specificare un vincolo di integrità referenziale secondo cui X è una chiave esterna di $R(T)$:

- il vincolo deve riferire una tabella $S(U)$ ed un insieme di attributi $Y \subseteq U$, dichiarati PRIMARY KEY o UNIQUE
- per ogni tupla $t \in R(T)$ tale che tutti gli attributi in X sono diversi da NULL deve esistere una tupla $t' \in S(U)$ tale che $t[X] = t'[Y]$

Si noti che implicitamente richiediamo $|X| = |Y|$.

Possiamo indicare che un attributo è una chiave esterna subito dopo la dichiarazione di un attributo:

```
REFERENCES <table> (<attribute>)
```

oppure alla fine di tutte le dichiarazioni:

```
FOREIGN KEY (<attributes>) REFERENCES <table> (<attributes>)
```

FOREIGN KEY: Esempio

```
CREATE TABLE Movies (
    title       CHAR(100),
    year        INT,
    length      INT,
    genre       CHAR(10),
    studio      CHAR(30),
    PRIMARY KEY (title, year),
    FOREIGN KEY (studio)
    REFERENCES Studio(name)
)
```

```
CREATE TABLE Studio (
    name        CHAR(30)
                PRIMARY KEY,
    address     VARCHAR(255),
    year        INT
)
```

E' possibile avere uno film indipendente senza uno studio associato (NULL). Ma non è possibile avere un film prodotto da uno studio "inesistente".

Mantenimento dell'Integrità Referenziale

```
CREATE TABLE Movies (
    title       CHAR(100),
    year        INT,
    length      INT,
    genre       CHAR(10),
    studio      CHAR(30),
    PRIMARY KEY (title, year),
    FOREIGN KEY (studio)
    REFERENCES Studio(name)
)

CREATE TABLE Studio (
    name        CHAR(30)
                PRIMARY KEY,
    address     VARCHAR(255),
    year        INT
)
```

Queste operazioni su `Movies` sono impedite:

1. inserimento di una tupla il cui attributo `studio` non è NULL e non coincide con l'attributo `name` di una tupla in `Studio`;
2. aggiornamento di una tupla per cambiare il suo attributo `studio` ad un valore non NULL che non coincide con l'attributo `name` di una tupla in `Studio`.

Attenzione! Sono gli unici casi problematici?

Mantenimento dell'Integrità Referenziale

```
CREATE TABLE Movies (
    title       CHAR(100),
    year        INT,
    length      INT,
    genre       CHAR(10),
    studio      CHAR(30),
    PRIMARY KEY (title, year),
    FOREIGN KEY (studio)
    REFERENCES Studio(name)
)

CREATE TABLE Studio (
    name        CHAR(30)
                PRIMARY KEY,
    address     VARCHAR(255),
    year        INT
)
```

Altre operazioni pericolose su `Studio`:

1. cancellazione di una tupla il cui attributo `name` coincide con l'attributo `studio` di qualche tupla in `Movies`;
2. aggiornamento di una tupla per cambiare il suo attributo `name` in modo tale che non coincida più con l'attributo `studio` di qualche tupla in `Movies`.

E' possibile gestire questi casi tramite diverse politiche di integrità.

Politiche di Integrità Referenziale

SQL mette a disposizione tre politiche per gestire i due casi descritti:

1. Default: rifiuta la modifica;
2. CASCADE: applica la stessa modifica (DELETE o UPDATE) sulle tuple che fanno uso della chiave esterna;
3. SET NULL: imposta la chiave esterna a NULL sulle tuple che fanno uso della stessa.

Possiamo specificare una politica diversa per DELETE ed UPDATE, utilizzando la sintassi ON DELETE oppure ON UPDATE seguito da CASCADE oppure SET NULL.

Politiche di Integrità Referenziale

Example

```
CREATE TABLE Movies (
    title       CHAR(100),
    year        INT,
    length      INT,
    genre       CHAR(10),
    studio      CHAR(30),
    PRIMARY KEY (title, year),
    FOREIGN KEY (studio)
    REFERENCES Studio(name)
        ON DELETE SET NULL
        ON UPDATE CASCADE
)
```

CHECK su Attributi

E' possibile specificare vincoli complessi sul valore di un attributo, usando la sintassi CHECK seguita da un'espressione booleana fra parentesi:

- si può usare qualsiasi espressione ammessa da WHERE
- standard SQL: si possono riferire altre relazioni tramite sotto-query, ma questo **non è supportato** nei DBMS commerciali (Postgres)
- il vincolo è controllato ogni volta che una tupla assume un nuovo valore per quell'attributo (INSERT o UPDATE)

Attenzione!

Questo non è necessariamente sufficiente a garantire che il vincolo non sia mai violato! Vedremo vari esempi nel resto della lezione...

CHECK su Attributi: Esempio 1

```
CREATE TABLE Movies (
    title       CHAR(100),
    year        INT CHECK(year >= 1895),
    length      INT CHECK(length >= 10),
    genre       CHAR(10),
    studio      CHAR(30),
    PRIMARY KEY (title, year),
    FOREIGN KEY (studio) REFERENCES Studio(name)
)

CREATE TABLE Studio (
    name        CHAR(30)
                PRIMARY KEY,
    address     VARCHAR(255),
    year        INT
)
```

CHECK su Attributi: Esempio 2

Questi due comandi sono simili, ma la loro semantica è diversa: perchè?

```
CREATE TABLE Movies (
    title    CHAR(100),
    year     INT CHECK(year >= 1895),
    . . .
    FOREIGN KEY (studio) REFERENCES Studio(name)
)
```

```
CREATE TABLE Movies (
    title   CHAR(100),
    year    INT CHECK(year >= 1895),
    . . .
    studio  INT CHECK (studio in (SELECT name FROM Studio))
)
```

CHECK su Attributi: Esempio 2

```
CREATE TABLE Movies (
    title    CHAR(100),
    year     INT CHECK(year >= 1895),
    . . .
    FOREIGN KEY (studio)
    REFERENCES Studio(name)
)

CREATE TABLE Movies (
    title   CHAR(100),
    year    INT CHECK(year >= 1895),
    . . .
    studio  INT CHECK (studio in
                       (SELECT name
                        FROM Studio))
)
```

La seconda forma offre meno garanzie della prima! Se un entry di `Studio` cambia il valore di `name`:

- nel primo caso possiamo usare una politica di integrità referenziale
- nel secondo caso non è possibile garantire integrità referenziale, perchè `Movies` non viene toccata

CHECK su Tuple

SQL permette di specificare anche vincoli sull'intera tupla piuttosto che sul singolo attributo. Le considerazioni sulla sintassi e sulla semantica sono sostanzialmente le stesse del caso precedente.

```
CREATE TABLE Movies (
    title       CHAR(100),
    year        INT,
    length      INT,
    genre       CHAR(10),
    studio      CHAR(30),
    PRIMARY KEY (title, year),
    FOREIGN KEY (studio) REFERENCES Studio(name),
    CHECK (year>1895 AND length > 10)
)
```

CHECK su Attributi o su Tuple?

Come scegliere se usare CHECK su attributi o su tuple?

- se un vincolo coinvolge più di un attributo e non è una congiunzione di vincoli su attributi indipendenti, è necessario ricorrere a CHECK su tuple per motivi di espressività
- se un vincolo coinvolge un solo attributo, possiamo scegliere fra i due tipi, ma i CHECK su attributi sono più efficienti dei CHECK su tuple, dato che devono essere controllati meno di frequente

Reminder!

- $A \Rightarrow B$ equivale a $\neg A \vee B$
- $\neg(A \wedge B)$ equivale a $\neg A \vee \neg B$
- $\neg(A \vee B)$ equivale a $\neg A \wedge \neg B$

Equivalenze Logiche

Come garantire che tutti coloro che soddisfano la proprietà A devono soddisfare la proprietà B?

- Logicamente equivalente a $A \Rightarrow B$
- Esprimibile quindi equivalentemente con $\neg A \vee B$

Esempio: tutti i film horror devono avere l'anno di produzione successivo al 1922.

Come garantire che solo coloro che soddisfano la proprietà A possono soddisfare la proprietà B?

- Logicamente equivalente a $B \Rightarrow A$
- Esprimibile quindi equivalentemente con $\neg B \vee A$

Esempio: solo i film di fantascienza possono durare più di 180.

Aggiornare i Vincoli

Possiamo dare un nome ai vincoli anteponendo alla loro dichiarazione la dicitura, questo ci permette di eliminarli in seguito.

```
CONSTRAINT NomeV [FOREIGN KEY, UNIQUE, CHECK, etc.] ...
```

E' possibile cancellare un vincolo esistente a partire dal suo nome:

```
ALTER TABLE NomeT DROP CONSTRAINT NomeV
```

E' possibile inserire un nuovo vincolo, ricorrendo alla sintassi:

```
ALTER TABLE NomeT ADD [CONSTRAINT NomeV] DefV
```

Il vincolo **deve già valere** sulla tabella al momento del suo inserimento! Questa caratteristica è molto desiderabile nella pratica.

La modifica di un vincolo non è supportata, ma può essere effettuata tramite una cancellazione seguita da un inserimento.

Esempio

```
CREATE TABLE Movies (
    title       CHAR(100),
    year        INT,
    length      INT,
    genre       CHAR(10),
    studio      CHAR(30),
    PRIMARY KEY (title, year),
    FOREIGN KEY (studio) REFERENCES Studio(name),
    CONSTRAINT scifi CHECK
    (genre LIKE "fantascienza" OR length<180)
)
```

Esercizio

Si consideri questo schema, con le relative chiavi primarie ed esterne:

```
Product(maker, model, type)
PC(model*, speed, ram, hd, price)
Laptop(model*, speed, ram, hd, screen, price)
Printer(model*, color, type, price)
```

Specificare i seguenti vincoli di integrità:

1. I soli tipi di prodotto sono PC, laptop oppure stampanti;
2. Un PC con velocità inferiore a 2.0 deve costare al massimo 500;
3. Un PC con velocità inferiore a 2.0 e un hard disk inferiore a 80 GB deve costare al massimo 300;
4. Un laptop con uno schermo più piccolo di 15 pollici deve avere un hard disk da almeno 40 GB o deve costare meno di 500.

Soluzione (1/4)

I soli tipi di prodotto sono PC, laptop oppure stampanti:

```
CREATE TABLE Product(
        maker VARCHAR(20),
        model VARCHAR(50) PRIMARY KEY,
        type  VARCHAR(20) CHECK (type = 'PC' OR
                                 type = 'Laptop' OR
                                 type = 'Printer')
);
```

Soluzione (2/4)

Un PC con velocità inferiore a 2.0 deve costare al massimo 500:

```
CREATE TABLE PC(
        model VARCHAR(50) PRIMARY KEY,
        speed FLOAT,
        ram INT,
        hd INT,
        price FLOAT,
        FOREIGN KEY (model) REFERENCES Product(model)
          ON DELETE CASCADE,
        CHECK (speed >= 2.0 OR price <= 500)
);
```

Soluzione (3/4)

Un PC con velocità inferiore a 2.0 e un hard disk inferiore a 80 GB deve costare al massimo 300:

```
CREATE TABLE PC(
        model VARCHAR(50) PRIMARY KEY,
        speed FLOAT,
        ram INT,
        hd INT,
        price FLOAT,
        FOREIGN KEY (model) REFERENCES Product(model)
          ON DELETE CASCADE,
        CHECK (speed >= 2.0 OR hd >= 80 OR price <= 300)
);
```

Soluzione (4/4)

Un laptop con uno schermo più piccolo di 15 pollici deve avere un hard disk da almeno 40 GB o deve costare meno di 500:

```
CREATE TABLE Laptop(
        model VARCHAR(50) PRIMARY KEY,
        speed FLOAT,
        ram INT,
        hd INT,
        screen FLOAT,
        price FLOAT,
        FOREIGN KEY (model) REFERENCES Product(model)
          ON DELETE CASCADE,
        CHECK (screen >= 15.0 OR hd >= 40 OR price < 500.0)
);
```

Limitazioni dei Vincoli: Esempio 1

Si considerino questi schemi, dove `studio` è chiave esterna per `name`:

```
Movies(tilte, year, length, genre, studio)
Studio(name, address, year)
```

Example

Il vincolo "Nessun film può essere prodotto da uno studio fondato dopo la sua uscita" non può essere espresso tramite un CHECK, a meno che il DBMS non supporti vincoli con sotto-query.

Soluzione con sotto-query (a scopo didattico):

Limitazioni dei Vincoli: Esempio 1

Si considerino questi schemi, dove `studio` è chiave esterna per `name`:

```
Movies(tilte, year, length, genre, studio)
Studio(name, address, year)
```

Example

Il vincolo "Nessun film può essere prodotto da uno studio fondato dopo la sua uscita" non può essere espresso tramite un CHECK, a meno che il DBMS non supporti vincoli con sotto-query.

Soluzione con sotto-query (a scopo didattico):

```
CHECK (100000 <= ALL(SELECT netWorth
                     FROM Studio, MovieExec
                     WHERE president = code)
```

Limitazioni dei Vincoli: Esempio 2

Si consideri lo schema:

```
Movies(title, year, length, genre, studio)
```

Example

Il vincolo "La durata complessiva dei film prodotti da ciascuno studio non deve superare i 10000 minuti" non può essere espresso con un CHECK, a meno che il DBMS non supporti vincoli con sotto-query.

Soluzione con sotto-query (a scopo didattico):

Limitazioni dei Vincoli: Esempio 2

Si consideri lo schema:

```
Movies(title, year, length, genre, studio)
```

Example

Il vincolo "La durata complessiva dei film prodotti da ciascuno studio non deve superare i 10000 minuti" non può essere espresso con un CHECK, a meno che il DBMS non supporti vincoli con sotto-query.

Soluzione con sotto-query (a scopo didattico):

```
CHECK (10000 >= ALL(SELECT SUM(length)
                    FROM Movies
                    GROUP BY studio))
```

Limitazioni dei Vincoli: Esempio 2 (cont.)

Si consideri lo schema:

```
Movies(title, year, length, genre, studio)
```

Example

Il vincolo "La durata complessiva dei film prodotti da ciascun studio deve essere di almeno 500 minuti" non può essere espresso con un CHECK, **neppure utilizzando sotto-query!**

In particolare, questo CHECK sembra corretto, ma non lo è... perchè?

```
CHECK (500 <= ALL(SELECT SUM(length)
                  FROM Movies
                  GROUP BY studio))
```

Asserzioni

Le asserzioni esprimono invarianti globali sull'intero schema relazionale:

```
CREATE ASSERTION <name> CHECK (<condition>)
```

La condizione deve essere vera quando l'asserzione è creata e continuare a rimanere tale dopo ogni modifica del database.

Attenzione!

Le asserzioni sono strettamente più potenti dei CHECK, ma molto più complicate da implementare efficientemente.

Nessuno dei principali DBMS le implementa, perchè troppo inefficienti!

Asserzioni: Esempio 1

Nessun film può essere prodotto da uno studio fondato dopola sua uscita.

```
CREATE ASSERTION RichPresident CHECK(
    NOT EXISTS(
        SELECT Movies.name
        FROM Studio, Movies
        WHERE Studio.name = Movies.studio AND
              Studio.year < Movies.year
    )
);
```

Asserzioni: Esempio 2

La durata complessiva dei film prodotti da ogni studio deve essere di almeno 500 minuti:

```
CREATE ASSERTION SumLength CHECK(
    500 <= ALL(
        SELECT SUM(length)
        FROM Movies
        GROUP BY studio
    )
);
```

Vincoli o Asserzioni?

Soggetto	**Dichiarazione**	**Controllo**	**Validità**
CHECK su attributo	Attributo di una tabella	Inserimento nella tabella o update dell'attributo	In assenza di sotto-query
CHECK su tupla	Tabella	Inserimento nella tabella o update della tupla	In assenza di sotto-query
Asserzione	Schema relazionale	Ogni modifica di ogni tabella menzionata	Sempre

Attenzione: Postgres ed anche gli altri DBMS commerciali non supportano le **asserzioni** e non ammettono **sotto-query nei CHECK**!

Checkpoint

Concetti Chiave

- Chiavi, chiavi esterne e politiche di integrità referenziale
- Vincoli di integrità su attributi e su tuple tramite CHECK
- Limitazioni dei vincoli e le sottigliezze delle sotto-query
- Asserzioni: una buona idea, ma di difficile implementazione

Materiale Didattico

Database Systems: Capitolo 7 fino alla Sezione 7.4 compresa