

# **Vincoli di Integrità**

## Approccio dichiarativo alla loro implementazione

**Antonella Poggi**

Dipartimento di informatica e Sistemistica  
SAPIENZA Università di Roma

**Progetto di Applicazioni Software**  
**Anno accademico 2008-2009**

*Questi lucidi sono stati prodotti sulla base del materiale preparato per il corso di Progetto di Basi di Dati da A. Calì e da D. Lembo.*

## Generalità

---

- I vincoli di integrità specificano proprietà dei dati
- I vincoli **restringono** l'insieme delle istanze possibili su uno schema, allo scopo di riflettere meglio il dominio che i dati medesimi rappresentano.
- Nella definizione di uno schema, occorre specificare tutti i vincoli che devono sussistere sulle istanze dello schema.
- Occorre specificare vincoli di integrità:
  - sullo schema concettuale
  - sullo schema logico

## Nota

---

Consideriamo nel seguito i vincoli **sullo schema logico**, che sono quelli di cui dobbiamo garantire il soddisfacimento a livello implementativo. Infatti, i dati della nostra applicazione verranno memorizzati in un DB relazionale.

Ovviamente, nel progetto della base di dati, questi vincoli corrispondono a vincoli precedentemente espressi sullo schema concettuale (e sul concettuale ristrutturato).

## Classificazione dei vincoli di integrità

---

Possiamo distinguere i vincoli in:

- vincoli **di stato**: si riferiscono allo stato dei dati
- vincoli **di transizione di stato**: si riferiscono a due stati consecutivi dei dati
- vincoli **di sequenza di stati (temporali)**: si riferiscono a più di due stati (non necessariamente consecutivi) dei dati

## Classificazione dei vincoli di integrità (cont.)

---

Un'altra classificazione possibile, ortogonale dalla precedente, è la seguente:

- vincoli **di tupla (riga)**: si riferiscono alle singole righe di una tabella
- vincoli **di tabella**: si riferiscono all'insieme delle righe di una tabella
- vincoli **inter-tabella**: sono i più generali, e si riferiscono a più di una tabella

## Vincoli di stato

---

Specificano delle proprietà statiche dei dati.

```
impiegato (codice, nome, cognome, livello, salario, dipartimento)
dipartimento (codice, descrizione)
```

1. Il livello è un intero da 1 a 8 (vincolo di **tupla**)
2. La somma dei salari di tutti gli impiegati non può superare un tetto  $M$  (vincolo di **tabella**)
3. Chiave (primaria): non ci sono due impiegati con lo stesso codice (vincolo di **tabella**)
4. Integrità referenziale (foreign key) tra l'attributo dipartimento della tabella impiegato e l'attributo codice della tabella dipartimento (vincolo **inter-tabella**)

## Vincoli di transizione di stato

---

Facendo riferimento alla precedente relazione impiegato, sono ad esempio vincoli di transizione di stato i seguenti:

- Il salario di un impiegato non può essere incrementato di più del 5 per cento con una singola operazione di aggiornamento (vincolo di **tupla**)
- Il salario medio non può essere incrementato di più del 5 per cento con una singola operazione di aggiornamento (vincolo di **tabella**)
- Supponendo che esista nello schema una tabella `pensionato`, un impiegato può diventare un pensionato ma non viceversa (vincolo **inter-tabella**)

Quelli più comuni sono i vincoli a livello di tupla

## Vincoli temporali

---

Questi vincoli coinvolgono un insieme di più di due stati non necessariamente consecutivi. Ad esempio (facendo sempre riferimento allo scenario precedente):

- Una volta che un dipendente è stato licenziato da un dipartimento, non può più essere impiegato in quel dipartimento (vincolo di **tupla**)
- Il salario medio non può incrementare di oltre il 10% in meno di 10 operazioni di aggiornamento (vincolo di **tabella**)
- Prima di diventare pensionato, un impiegato deve aver assunto almeno 3 diversi livelli di impiego (vincolo **inter-tabella**)

## Nota

---

Alcuni vincoli sono impliciti nello schema relazionale, ad esempio: “I valori nelle righe devono essere conformi ai tipi specificati”

**Tali vincoli non vanno specificati**, in quanto si presume che siano automaticamente verificati.

## Specifica dei vincoli

---

- È opportuno che i vincoli di stato siano specificati in modo **dichiarativo**, e cioè facendo riferimento a **proprietà** dello stato, e non a operazioni che consentano il soddisfacimento di dette proprietà. Si può usare semplicemente una descrizione precisa in linguaggio naturale, oppure formalismi ad-hoc
  - ad es., Object Constraint Language - OCL di UML
- I vincoli devono essere **decomposti** in vincoli elementari, in modo che possano essere facilmente implementati (come vedremo più avanti)

## Consistenza dei vincoli

---

- Un insieme di vincoli si dice **consistente** se esiste almeno un database (non vuoto) che li verifica tutti.
- In molti casi non è difficile definire vincoli non consistenti.

## Ridondanza

---

- Un insieme di vincoli si dice **ridondante** se esiste in esso almeno un vincolo che implica un altro.
- È opportuno che non ci sia ridondanza nei vincoli definiti su uno schema relazionale.

## Implementazione dei vincoli

- **Implementare** i vincoli significa assicurare che le operazioni effettuate sui dati da parte dell'applicazione siano tali che i dati si trovino sempre in uno stato consistente rispetto ai vincoli.
- I vincoli possono essere implementati a **livello di gestione dei dati** (tramite gli strumenti del DBMS), o mediante il **comportamento dell'applicazione**
- Nel primo caso, possiamo definire i vincoli
  - (a) all'interno di un comando `CREATE TABLE` o `ALTER TABLE` (o anche `CREATE ASSERTION` e tramite la definizione di nuovi domini) – **approccio dichiarativo**
  - (b) Attraverso l'uso di User Defined Function (UDF) o Trigger – **approccio procedurale** (lo vedremo

piuàvanti nel corso

## Approccio dichiarativo alla implementazione

---

Per **dichiarare** vincoli di integrità possiamo usare i seguenti costrutti SQL:

- Costrutto NOT NULL
- Costrutto DEFAULT
- Costrutto UNIQUE
- Costrutto PRIMARY KEY
- Costrutto FOREIGN KEY (REFERENCES)
- Costrutto CHECK
- Costrutto CREATE DOMAIN
- Costrutto CREATE ASSERTION

Tranne CREATE DOMAIN e CREATE ASSERTION, gli altri costrutti sono usati all'interno dei comandi CREATE TABLE e ALTER TABLE.

## Approccio dichiarativo alla implementazione

---

- Ogni vincolo di integrità ha un **nome unico all'interno di uno schema**. Se il nome non è specificato esplicitamente, questo viene assegnato dal sistema secondo una modalità dipendente dall'implementazione
- Inoltre, ogni vincolo è caratterizzato da una **modalità di controllo** (checking mode), che determina il momento in cui il vincolo viene verificato: nel caso in cui la modalità sia **IMMEDIATE** il vincolo è controllato dopo ogni esecuzione di un comando SQL; se la modalità è **DEFERRED** il vincolo è verificato solo al termine della transazione corrente
- La modalità deferred può essere applicata solo ai vincoli **DEFERRABLE** (si veda documentazione SQL)

## Esempio

---

```
CREATE TABLE IMPIEGATO (  
  COD_IMP INT PRIMARY KEY,  
  NOME VARCHAR(20),  
  COGNOME VARCHAR(20),  
  LIVELLO INT CHECK (LIVELLO <= 8 AND LIVELLO >=1),  
  SALARIO INT CHECK (SALARIO >=0),  
  DIPARTIMENTO INT,  
  CONSTRAINT fk_ID FOREIGN KEY (DIPARTIMENTO)  
    REFERENCES DIPARTIMENTO(COD_DIP)  
    INITIALLY IMMEDIATE DEFERRABLE  
);  
SET CONSTRAINT fk_ID DEFERRED;
```

I valori di default sono IMMEDIATE e DEFERRABLE.

## Il costrutto CHECK

---

- La sintassi SQL99 è

`CHECK (<condizione>)`

dove `<condizione>` è una qualsiasi condizione che può comparire all'interno di una clausola `WHERE` (incluse quelle che fanno uso di sub-query)

- In linea di principio, consente di implementare vincoli a livello di tupla (ad esempio, `SALARIO >= 0`), di tabella (in genere tramite sub-query con operatori aggregati), ed inter-tabella

## Esempio

---

```
CREATE TABLE IMPIEGATO (  
  COD_IMP INT PRIMARY KEY,  
  NOME VARCHAR(20),  
  COGNOME VARCHAR(20),  
  LIVELLO INT CHECK (LIVELLO <= 8 AND LIVELLO >=1),  
  SALARIO INT CHECK (SALARIO >=0),  
  DIPARTIMENTO INT,  
  CONSTRAINT fk_ID FOREIGN KEY (DIPARTIMENTO)  
    REFERENCES DIPARTIMENTO (COD_DIP)  
);
```

```
ALTER TABLE IMPIEGATO ADD CONSTRAINT check_salario_medio  
  CHECK (50 > (SELECT AVG(SALARIO) FROM IMPIEGATO));
```

## Il costrutto ASSERTION

---

- Per i vincoli inter-tabella, nel caso in cui non sia ovvio stabilire quale sia la tabella a cui associare il vincolo, il costrutto CHECK dovrebbe essere usato all'interno di ASSERTION

```
CREATE ASSERTION <NomeAss> CHECK (<condizione>);
```

## Esempio (vincolo di inclusione)

---

**vincolo:** Ogni impiegato risiede in una città (supponiamo che esista una tabella `RISIEDE_IN(IMPIEGATO,CITTA)`, con chiave su entrambe le colonne)

### usando una check

```
ALTER TABLE IMPIEGATO ADD CONSTRAINT impiegato_in_persona
    CHECK (COD_IMP IN (SELECT IMPIEGATO FROM RISIEDE_IN));
```

### usando una assertion

```
CREATE ASSERTION impiegato_risiede
    CHECK (NOT EXISTS (SELECT * FROM IMPIEGATO
        WHERE COD_IMP NOT IN
            (SELECT IMPIEGATO FROM RISIEDE_IN)
    ));
```

## Il costrutto FOREIGN KEY

---

- Consente di esprimere un vincolo di integrità referenziale (ad es.,  $R_1[A] \subseteq R_2[B]$ , con  $B$  **chiave** di  $R_2$ )
- $B$  può essere chiave primaria (PRIMARY KEY) o secondaria (UNIQUE)
- SQL99 consente di definire una **politica di reazione alle modifiche** (cancellazioni ed aggiornamenti) dei dati contenuti nella tabella **esterna** ( $R_2$ ) che possono portare alla violazione del vincolo
- modifiche alle righe della tabella interna ( $R_1$ ) che violano il vincolo non sono consentite

## Il costrutto FOREIGN KEY (cont.)

---

- per le cancellazioni si fa uso della clausola

```
ON DELETE [CASCADE|SET NULL|SET DEFAULT|  
          NO ACTION|RESTRICT]
```

tutte le righe della tabella interna ( $R_1$ ) corrispondenti alla riga cancellata vengono eliminate (opzione CASCADE), poste a NULL (SET NULL), poste al valore di default (SET DEFAULT), oppure non viene eseguita alcuna reazione (NO ACTION). Con l'opzione RESTRICT si impedisce la cancellazione.

## Il costrutto FOREIGN KEY (cont.)

---

- per gli aggiornamenti si fa uso della clausola

```
ON UPDATE [CASCADE|SET NULL|SET DEFAULT|  
NO ACTION|RESTRICT]
```

il nuovo valore dell'attributo della tabella esterna ( $R_2$ ) viene riportato su tutte corrispondenti righe della interna ( $R_1$ ) (CASCADE), all'attributo referente viene assegnato NULL (SET NULL) oppure il valore di default (SET DEFAULT) al posto del valore modificato nella tabella esterna, oppure non viene eseguita alcuna reazione (NO ACTION). Con l'opzione RESTRICT si impedisce l'aggiornamento.

## Il costrutto FOREIGN KEY (esempio)

---

```
CREATE TABLE IMPIEGATO (  
  COD_IMP INT PRIMARY KEY,  
  NOME VARCHAR(20),  
  COGNOME VARCHAR(20),  
  LIVELLO INT CHECK (LIVELLO <= 8 AND LIVELLO >=1),  
  SALARIO INT CHECK (SALARIO >=0),  
  DIPARTIMENTO INT,  
  CONSTRAINT fk_ID FOREIGN KEY (DIPARTIMENTO)  
    REFERENCES DIPARTIMENTO (COD_DIP) ON DELETE SET NULL  
    ON UPDATE CASCADE  
);
```

## MySQL vs. l'approccio dichiarativo

---

MySQL consente di usare:

- Costrutto PRIMARY KEY
- Costrutto UNIQUE
- Costrutto FOREIGN KEY
- Costrutto NOT NULL
- Costrutto DEFAULT

**Non è possibile** utilizzare il costrutto CHECK (questo impedisce la definizione dichiarativa di molti vincoli fra cui inclusioni ed esclusioni).

## Un'alternativa al costrutto CHECK

---

Nei casi in cui si voglia forzare il valore di alcuni campi ad appartenere da un determinato insieme finito, è possibile usare un tipo enumerato ENUM.

```
CREATE TABLE PERSONA (  
NOME      VARCHAR(20),  
COGNOME  VARCHAR(20),  
ETA       INT,  
SESSO    ENUM ('M', 'F')  
);
```

In questo esempio, posso inserire tuple che abbiano nel campo `SESSO` esclusivamente i valori `'M'`, `'F'`, `'m'`, `'f'`.

## MySQL ed il costrutto foreign key

---

- È possibile specificare le clausole  
`ON {DELETE | UPDATE} CASCADE` e  
`ON {DELETE | UPDATE} SET NULL`
- In mancanza di una clausola, viene impedito ogni cancellazione o aggiornamento sulla tabella esterna (di fatto si ha un comportamento analogo a `RESTRICT`).
- L'opzione `NO ACTION` si comporta in maniera analoga (differentemente dallo standard SQL).
- L'opzione `SET DEFAULT` non è supportata (anche se viene riconosciuta dal parser).
- **Non è supportato** l'uso della clausola `DEFERRED`.

## Esempio

---

Persona (cod\_p, nome, salario, tipo)

Azienda (cod\_a, nome)

Impiego (cod\_p, cod\_a)

Gli attributi sottolineati indicano la chiave della relazione

### Vincoli sullo schema logico

1.  $\text{Persona}[\text{tipo}] \in \{P, I\}$

i.e., il tipo di ogni persona deve essere "P" o "I"  
(pensionato o impiegato)

2.  $\text{Persona}[\text{salario}] \leq 180.000$

i.e., il salario di ogni persona deve essere al più pari a  
180.000

$$3. \forall cp, n, s, t, ca \text{ Persona}(cp, n, s, t), \\ \text{Impiego}(cp, ca) \Rightarrow t = \text{'I'}$$

i.e., solo le tuple di `Persona` che corrispondono ad impiegati (cioè hanno tipo pari ad "I") possono avere il valore dell'attributo `cod_p` che compare in `Impiego[cod_p]`

$$4. \forall cp, n, s \text{ Persona}(cp, n, s, \text{'I'}) \\ \exists ca | \text{Impiego}(cp, ca)$$

i.e. tutte le tuple di `Persona` che corrispondono ad impiegati devono avere il valore dell'attributo `cod_p` che compare in `Impiego[cod_p]`

5. Impiego[cod\_p]  $\subseteq$  Persona[cod\_p] (foreign key)

6. Impiego[cod\_a]  $\subseteq$  Azienda[cod\_a] (foreign key)

## Implementazione (cont.)

---

**1. Persona[tipo]  $\in$  {P, I}** Vincolo di stato tupla

Implementato tramite un tipo enumerato (costrutto ENUM).

**5. Impiego[cod\_p]  $\subseteq$  Persona[cod\_p]** Vincolo di stato inter-tabella

**6. Impiego[cod\_a]  $\subseteq$  Azienda[cod\_a]** Vincolo di stato inter-tabella

Implementati facilmente con FOREIGN KEY con l'opzione ON DELETE CASCADE

## Implementazione (cont.)

---

2.  $\text{Persona}[\text{salario}] \leq 180.000$  Vincolo di stato di tupla

3.  $\forall cp, n, s, t, ca \text{ Persona}(cp, n, s, t), \text{ Impiego}(cp, ca)$   
 $\Rightarrow t = \text{'I'}$  Vincolo di stato inter-tabella

4.  $\forall cp, n, s \text{ Persona}(cp, n, s, \text{'I'})$   
 $\exists ca | \text{ Impiego}(cp, ca)$  Vincolo di stato inter-tabella

Non si possono implementare seguendo un approccio dichiarativo!

Questi vincoli richiedono di essere implementati in maniera procedurale, ovvero controllando il loro soddisfacimento ad ogni inserimento/cancellazione nelle tabelle coinvolte.

⇒ è necessario fare un'analisi accurata del da farsi in corrispondenza delle operazioni critiche.

Vedremo più avanti come implementarli con un approccio procedurale.

## Creazione tabelle – codice SQL

---

```
create table PERSONA(  
cod_p    varchar(10) primary key,  
nome     varchar(15) not null,  
cognome  varchar(15) not null,  
salario  int,  
tipo     enum ('I','P')  
);
```

```
create table AZIENDA(  
cod_a    varchar(10) primary key,  
nome     varchar(15) not null  
);
```

## Creazione tabelle – codice SQL (cont.)

---

```
create table IMPIEGO(  
cod_p varchar(10),  
cod_a varchar(10),  
primary key (cod_p, cod_a)  
);
```

```
alter table IMPIEGO add constraint fk_impiego_persona  
foreign key (cod_p) references PERSONA(cod_p)  
on delete cascade;
```

```
alter table IMPIEGO add constraint fk_impiego_azienda  
foreign key (cod_a) references AZIENDA(cod_a)  
on delete cascade;
```