

# Esempi di Progetti per Basi di Dati A.A. 2007-08

Adriano Peron

Di seguito vengono riportate i temi di progetto per il corso di Basi di dati e sistemi informativi Gr. 1 assegnati nell'Anno Accademico 2007-08. Il documento contiene:

- Descrizione della documentazione richiesta
- Descrizione delle tracce proposte
- Esempi di progetti svolti.

**Si noti bene che le tracce proposte vengono riportate nella forma in cui sono state consegnate (quindi senza modifica degli eventuali problemi segnalati in fase di correzione). Ne segue che esse non vanno considerate come prive di errori e imperfezioni. Si vuole solo proporre agli studenti un esempio di svolgimento di progetto ritenuto nel complesso soddisfacente per ciascuno dei temi proposti.**

# Progetti di Basi di Dati AA 2007/08

Prof. A. Peron

## **Scelta del tema del progetto**

Per conseguire l'esame di basi di dati e sistemi informativi del Gruppo 1 (Prof. A. Peron) nell'anno accademico 2007/08 gli studenti devono svolgere un progetto da scegliere tra quelli elencati e descritti di seguito attenendosi alle seguenti norme:

- Non sono ammessi progetti su temi diversi da quelli elencati.
- Gli studenti debbono inviare una mail al docente (email peron@na.infn.it) indicando la composizione del gruppo e il tema prescelto.
- Sono ammessi gruppi di massimo due persone. Sono ammessi gruppi di una sola persona anche se si consiglia il gruppo da due persone.
- Per consulenza sulla fase di progettazione e realizzazione i gruppi si potranno rivolgere al Prof. A. Peron nell'orario di ricevimento.
- Il progetto deve essere consegnato prima della (o in occasione della) prova scritta (se la prova scritta non viene superata il progetto rimane valido per le prove successive).
- Per i progetti consegnati entro il 16 giugno viene valutato (per ogni traccia di progetto) il miglior progetto. Al miglior progetto vien riservato un bonus in termini di punteggio.

## **Contenuto del progetto**

Lo svolgimento del progetto deve comprendere le seguenti fasi:

1. Definizione dello schema concettuale utilizzando;
2. Definizione dello schema logico utilizzando il modello relazionale;
3. Definizione della base di dati utilizzando un DBMS relazionale;
4. Discussione sulla definizione dei vincoli.

## **Documentazione del progetto**

Per la valutazione del progetto va presentata al docente, entro i termini temporali fissati in questo sito (e comunque in anticipo rispetto alla valutazione del corretto funzionamento dell'applicativo) una documentazione (cartacea) del lavoro. La documentazione deve contenere i seguenti elementi:

1. Una descrizione testuale sintetica del problema;
2. Class diagram UML;
3. Dizionario dei dati del Class Diagram;
4. Eventuale ristrutturazione del Class Diagram;
5. Schema relazionale (dump dello schema della base di dati definita mediante un sistema per la gestione delle basi di dati a scelta);
6. Definizione dei vincoli nello standard SQL.

## 1 Gestione di Class Diagram UML

**LIMITAZIONE:** massimo 2 persone, massimo 10 gruppi, DISPONIBILE

Si definisca una base di dati per la memorizzazione della struttura di class diagram di UML. La base di dati dovrà essere in grado di ospitare per ciascun class diagram memorizzato, tutti i costrutti in esso ospitati (classi con attributi e metodi, associazioni con ruoli e cardinalità, association class, specializzazioni, ect. Le istanze di costrutti saranno associate ai class diagram di appartenenza. Si ponga attenzione a definire un opportuno insieme di vincoli che impedisca la composizione di class diagram scorretti.

## 2 Gestione di schemi relazionali

**LIMITAZIONE:** massimo 2 persone, massimo 10 gruppi, DISPONIBILE

Si definisca una base di dati per la memorizzazione della struttura di schemi relazionali (definizione di tabelle e di vincoli). La base di dati dovrà essere in grado di ospitare per ciascun schema relazionale memorizzato, tutti i costrutti in esso ospitati (definizione di tabelle, dei loro attributi, dei vincoli associati agli attributi e alle tabelle, vincoli di integrità etc. Si ponga attenzione a definire un opportuno insieme di vincoli che impedisca la composizione di schemi scorretti.

## 3 Gestione di Content Management System

**LIMITAZIONE:** massimo 2 persone, massimo 10 gruppi, DISPONIBILE

Si deve progettare la base di dati per un Content Management System in cui viene data la possibilità all'utente di definire la struttura e il formato di diverse tipologie di documenti (ad esempio libri, articoli). Si sottolinea il fatto che la base di dati deve essere in grado di ospitare non tanto formati di documenti a priori definiti ma piuttosto di ospitare il formato e la struttura definiti dall'utente. A questo proposito la base di dati dovrà essere in grado di ospitare la descrizione e il formato dei più comuni documenti (libri articoli etc).

**UNIVERSITA' DEGLI STUDI DI NAPOLI  
FEDERICO II**

**FACOLTA' DI SCIENZE MM.FF.NN.  
CORSO DI LAUREA IN INFORMATICA**

**PROGETTAZIONE DI UNA BASE DI DATI  
PER LA GESTIONE DI CLASS DIAGRAM UML**

Elaborato per il conseguimento  
dell'esame di Basi di Dati e Sistemi  
Informativi del gruppo 1,  
Prof. A. Peron

**STUDENTE:**

Nicola Di Leva 566/2991  
ndileva@email.it

## Contenuti:

### 1. Analisi del problema

- 1.1 Gestione di Class Diagram UML
- 1.2 Descrizione del Problema

### 2. Progettazione Concettuale

- 2.1 Diagramma UML: Classi e Relazioni
- 2.2 Diagramma UML: Tipi, Attributi e Proprieta
- 2.3 Diagramma Ristrutturato
- 2.4 Dizionario dei Dati
- 2.5 Business Rules

### 3. Progettazione Logica

- 3.1 Passaggio al Relazionale
- 3.2 Schema Relazionale
- 3.3 Implementazione delle Tabelle
- 3.4 Implementazione dei Vincoli

### 4. Test Case

- 4.1 Diagramma di esempio
- 4.2 Stampa delle tabelle Oracle

## Bibliografia

# 1. Analisi del problema

## 1.1 Gestione di Diagrammi UML

*Si definisca una base di dati per la memorizzazione della struttura di class diagram di UML. La base di dati dovr' a essere in grado di ospitare per ciascun class diagram memorizzato, tutti i costrutti in esso ospitati (classi con attributi e metodi, associazioni con ruoli e cardinalità, association class, specializzazioni, etc. Le istanze di costrutti saranno associate ai class diagram di appartenenza. Si ponga attenzione a definire un opportuno insieme di vincoli che impedisca la composizione di class diagram scorretti*

## 1.2 Descrizione del Problema

La complessità del problema dipende in ultima analisi dall'insieme di costrutti UML che si vogliono supportare e dal grado di aderenza allo standard. Non che il linguaggio preveda troppe restrizioni, anzi, UML è noto per essere estremamente flessibile - si pensi a a meccanismi di estensione come i *Profiles* o al fatto che quasi ogni costrutto di UML è *opzionale*.

Ma, se da un lato non ci sono troppe regole di well-formedness di cui preoccuparsi, dall'altro può essere difficile conciliare l'elasticità dello standard con la rigidità del modello relazionale. Un caso difficile da trattare sono per esempio le relazioni eterogenee ( es. *dependency* ) tra costrutti di tipo diverso, ciascuno realizzato da una tabella.

Nel tentativo di trovare un equilibrio tra completezza e facilità di comprensione, come linea guida generale di questo progetto, si è scelto di supportare un insieme di costrutti minimale ma sufficiente a memorizzare una ampia varietà di diagrammi.

### 1.2.1 Funzionalità

La base di dati proposta permette la memorizzazione e la gestione di un numero qualsiasi di *class diagram*. A ciascun Diagramma possono essere associati più costrutti top-level - Classi, Interfacce, Associazioni, Tipi Primitivi, Tipi Enumerati, Note, Generalization Set

Le Classi possono anche essere definite non a livello di diagramma, ma come membri di altre Classi (*inner class*).

Ogni Classe/Interfaccia può avere più Attributi e Metodi, ed ogni Metodo può avere più Parametri nella sua lista degli argomenti. Attributi, Metodi e Parametri possono essere associati a Tipi Primitivi o Enumerazioni definiti nel Diagramma a cui essi appartengono. Ogni Diagramma può avere un suo esclusivo insieme di tipi definiti dall'utente.

Classi e Interfacce possono essere collegate ad una o più Associazioni e ciascun 'collegamento' è rappresentato da una entità a se, chiamata NodoAssociazione nella base di dati, che definisce il 'ruolo' assunto dalla Classe nel contesto dell'Associazione. Una Associazione legata a due soli NodiAssociazione è *binaria*; è *n-aria* se i nodi sono più di due. Se tra una Associazione e una Classe esiste più di un collegamento, sia ha una *associazione riflessiva*.

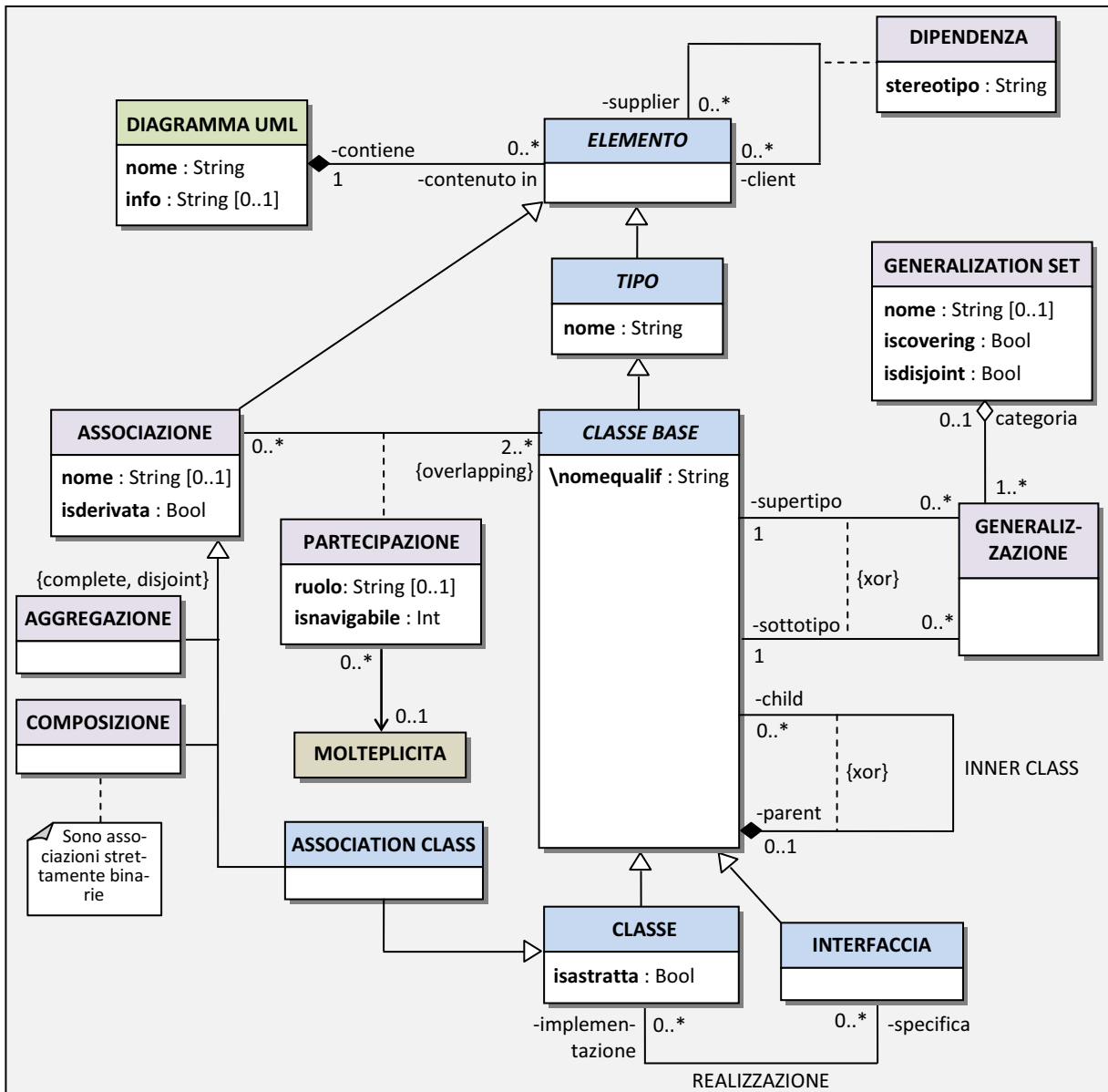
Classi e Interfacce possono essere collegate ad altre Classi e Interfacce, che le specializzano, tramite la relazione di Generalizzazione, che può essere indipendente oppure far parte di una categoria specifica di sottotipi, rappresentata dal costrutto Generalization Set.

Una Interfaccia può essere collegata ad una Classe che la implementa, tramite la relazione di Realizzazione.

Ogni elemento del Diagramma può essere decorato con proprietà definite dall'utente - Stereotipi e Constraint ( questi ultimi possono anche essere applicati a più costrutti, es. lo xor tra due Associazioni) - e può essere associato ad una o più Note che ne forniscono una descrizione testuale. Due qualsiasi elementi del Diagramma possono anche essere collegati da una relazione di Dipendenza.

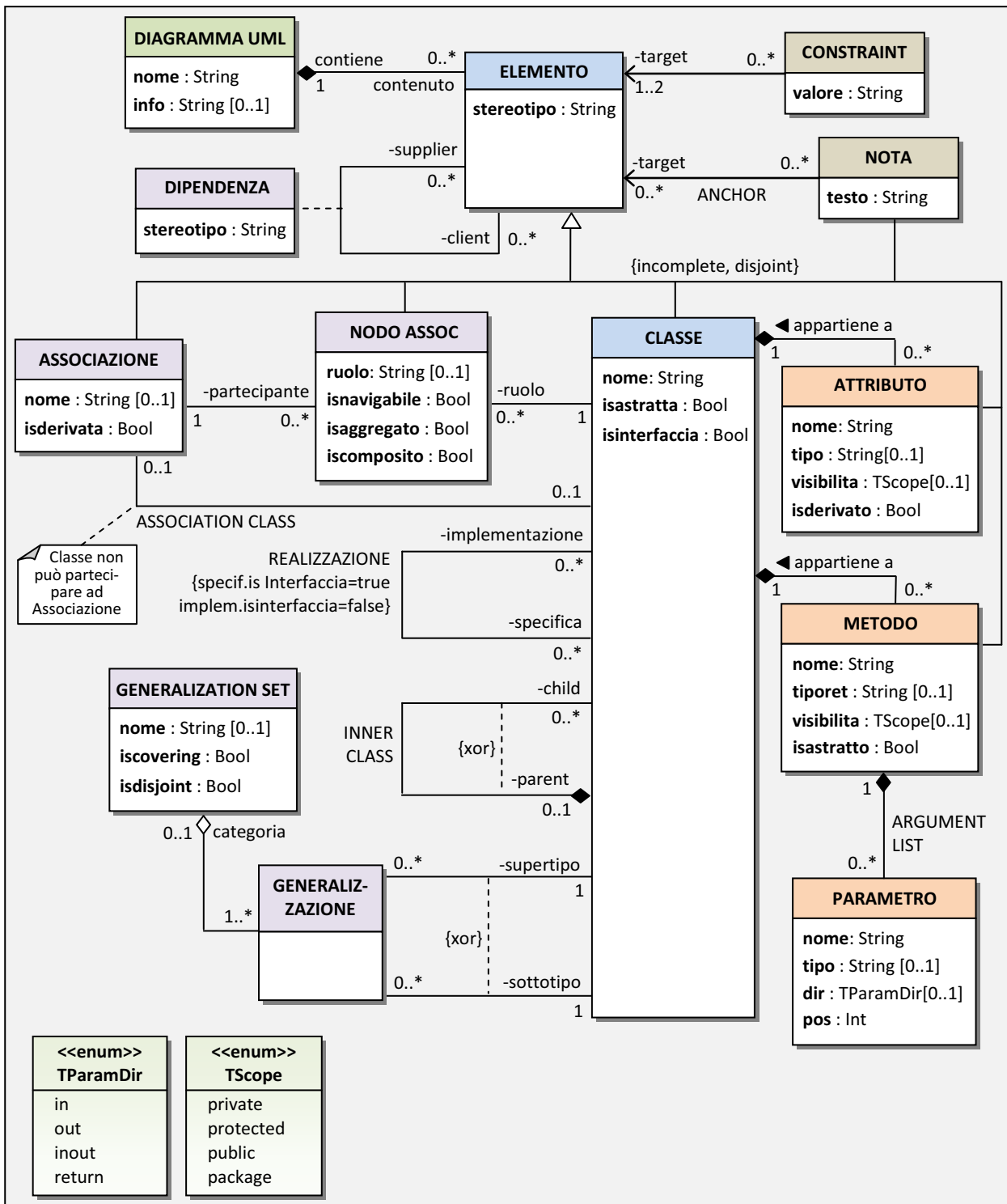
## 2. Progettazione concettuale

### 2.1 UML: Classi e Relazioni





## 2.3 Diagramma Ristrutturato



## 2.4 Dizionario dei dati

Entità	Descrizione	Attributi
<b>Diagramma UML</b>	un <i>class diagram</i> della base di dati, cioè un contenitore top-level di costrutti	<b>nome</b> (string) - nome del diagramma <b>info</b> (string, opzionale) - informazioni aggiuntive specificate dal designer
<b>Elemento</b>	generico costrutto del linguaggio UML	
<b>Classe Base</b>	costrutto generalizzabile che può avere proprietà strutturali e partecipare ad associazioni	<b>nome</b> (string) - nome della classe <b>nomequalif</b> (string,derivato) - nome qualificato della classe
<b>Classe</b>	classe UML	<b>isastratta</b> (bool) - indica se la classe è <i>astratta</i> {true}, o <i>concreta</i> {false}
<b>Interfaccia</b>	interfaccia UML	
<b>Association Class</b>	costrutto che incorpora le funzionalità di una classe e di una associazione	
<b>Associazione</b>	incapsula le proprietà di una associazione n-aria	<b>nome</b> (string,opzionale) - nome dell'associazione <b>isastratta</b> (bool) - indica se l'associazione può essere desunta da altre associazioni
<b>Aggregazione</b>	tipo di associazione binaria che modella la relazione <i>parte-di</i>	
<b>Composizione</b>	tipo di associazione binaria che modella una relazione <i>parte-di</i> esclusiva	
<b>Generalization Set</b>	'categoria' di sottotipi di un determinato tipo base (v. costr. omonimo di UML 2.0)	<b>nome</b> (string,opzionale) - nome della categoria <b>iscovering</b> (bool) – specifica se una gerarchia è <i>totale</i> {true} oppure <i>parziale</i> {false}. <b>isdisjoint</b> (bool) - specifica se una gerarchia è <i>esclusiva</i> {true} o <i>sovrapposta</i> {false}.
<b>Generalizzazione</b>	descrive le proprietà di una generalizzazione	
<b>Partecipazione &lt;associazione&gt;</b>	mette in relazione le associazioni con le classi che vi prendono parte	<b>ruolo</b> (string,opzionale) - descrive la funzione che una classe partecipante assume nell'associazione. <b>isnavigabile</b> (bool) - indica se l'estremità è <i>navigabile</i> , cioè accessibile dagli altri membri dell'associazione tramite un riferimento diretto.
<b>Inner Class &lt;associazione&gt;</b>	associa le classi membro alle classi in cui sono definite	
<b>Realizzazione &lt;associazione&gt;</b>	associa le interfacce alle classi che le implementano	

<b>Dipendenza</b> <associazione>	rappresenta una relazione di dipendenza tra due elementi qualsiasi di un diagramma	<b>stereotipo</b> (string, opzionale) - stereotipo che descrive la relazione di dipendenza fra client e supplier
<b>Costrutto Tipato</b>	costrutto a cui può essere associato un <i>tipo</i>	
<b>Class Property</b>	generica caratteristica strutturale di una classe	<b>nome</b> (string) – nome della proprietà <b>visibilita</b> (string, opzionale) - se specificato, può assumere i valori { <i>private, protected, public, package</i> }
<b>Attributo</b>	una proprietà statica di una classe	<b>isderivato</b> (bool) - indica se l'attributo sia memorizzato o calcolato in base ai valori di altri attributi ( in UML si indica col segno / ). <b>defaultval</b> (string, opzionale) - valore iniz. dell'attributo
<b>Metodo</b>	una <i>operazione</i> , nel gergo UML, di una classe	<b>isastratto</b> (bool) – specifica se il metodo è <i>astratto</i> {true}, cioè privo di implementazione, o <i>concreto</i> {false}.
<b>Parametro</b>	un' <i>argomento</i> di un metodo di classe	<b>nome</b> (string) – nome del parametro <b>dir</b> (string, opzionale) - indica se il parametro è di <i>input</i> {in}, <i>output</i> {out} o entrambi {inout}. <b>defaultval</b> (string, opzionale) - valore di default del parametro
<b>Tipo Primitivo</b>	<i>tipo</i> non strutturato	
<b>Enum</b>	tipo enumerato	
<b>Letterale</b>	elemento del dominio di un tipo enumerato	<b>valore</b> (string) - valore del letterale
<b>Constraint</b>	definisce un vincolo associato ad uno o due elementi	<b>valore</b> (string) - espressione che rappresenta il vincolo
<b>Nota</b>	informazione testuale che può essere associata ad uno o più elementi	<b>testo</b> (string) - contenuto della nota
<b>Molteplicita</b>	definisce le proprietà di un insieme di istanze di uno stesso elemento ( es. un attributo di una classe o un membro di una associazione )	<b>lower</b> (int) - numero minimo di istanze <b>upper</b> (int, opzionale) - numero massimo di istanze. Se non specificato, si assume cardinalità <i>infinita</i> <b>isordered</b> (bool) - indica se l'insieme è ordinato <b>isunique</b> (bool) - indica se le istanze sono distinte

Entità e attributi introdotti nel database ristrutturato:

Entità	Descrizione	Attributi
<b>Elemento</b>		<b>stereotipo</b> (string, opzionale) - stereotipo associato all'elemento
<b>Classe</b>	rappresenta una classe, concreta o astratta, oppure una interfaccia	<b>isinterfaccia</b> (bool) - definisce se la classe sia una interfaccia {true} oppure una classe {false}
<b>Nodo Associazione</b>	rappresenta una estremità (membro) di una associazione	<b>isaggregato</b> (bool) - specifica se l'estremità è un aggregato ( indicato in UML da un rombo vuoto ) <b>iscomposito</b> (bool) - indica se l'estremità è un composito ( indicato in UML da un rombo pieno )
<b>Parametro</b>		<b>pos</b> (int) - posizione del parametro nella lista degli argomenti. Traduce il vincolo {ordered}
<b>Association Class</b> <associazione>	collega una classe ed una associazione che formano insieme il costrutto association class	

## 2.4 Business Rules

### 2.4.1 Ownership

- [R01] se un diagramma viene cancellato ogni costruito in esso contenuto deve essere rimosso
- [R02] quando una classe viene cancellata, tutte le sue classi membro devono essere cancellate
- [R03] una classe non può essere classe membro di se stessa, a nessun livello
- [R04] una classe membro deve appartenere allo stesso diagramma della classe che la contiene

### 2.4.2 Naming

- [R05] classi definite nello stesso ambito, cioè un diagramma o un'altra classe, devono avere nomi distinti
- [R06] i tipi primitivi definiti nello stesso diagramma devono avere nomi distinti
- [R07] tutti gli attributi di una classe devono avere nomi distinti
- [R08] tutti i parametri di un metodo devono avere nomi distinti
- [R09] una classe non può avere due metodi con la stessa 'signature'

### 2.4.3 Attributi, Metodi

- [R10] il tipo di un attributo o di un metodo deve essere il nome di una classe o di un tipo primitivo definiti nel diagramma in cui è contenuta la classe a cui appartiene l'attributo
- [R11] il tipo di un parametro di un metodo deve essere una classe o un tipo primitivo definiti nel diagramma in cui è contenuta la classe a cui appartiene il metodo
- [R12] se il tipo di un elemento – attributo, parametro o metodo - è una enumerazione ed è fornito un valore di default, tale valore deve corrispondere ad un letterale della enum
- [R13] un metodo astratto può appartenere solo ad una classe astratta o ad una interfaccia

### 2.4.4 Associazione

- [R14] le classi che partecipano ad una associazione devono appartenere allo stesso diagramma
- [R15] in ogni associazione soltanto un estremo può essere aggregato o composito
- [R16] una associazione in cui un estremo è aggregato o composito può avere al più due membri
- [R17] i ruoli dei membri di una associazione, dove specificati, devono essere distinti fra loro
- [R18] una association class non può avere se stessa come estremo
- [R19] la classe e l'associazione che formano una association class devono stare nello stesso

diagramma

#### **2.4.5 Generalizzazione**

[R20] un tipo base ed ogni sua specializzazione devono appartenere allo stesso diagramma

[R21] un elemento non può derivare, direttamente o transitivamente, da se stesso. Non sono cioè ammessi cicli nel grafo (orientato) delle gerarchie.

[R22] una interfaccia può derivare solo da un'altra interfaccia ed una classe solo da un'altra classe

[R23] le generalizzazioni che sono parte di uno stesso generalization set devono avere lo stesso tipo base

#### **2.4.6 Realizzazione, Dipendenza, Constraint**

[R24] gli elementi collegati dalla relazione devono appartenere allo stesso diagramma

[R25] se uno o entrambi gli elementi collegati vengono cancellati, la relazione deve essere cancellata

[R26] 'specifica' e 'implementazione' devono essere rispettivamente una interfaccia ed una classe

#### **2.4.7 Proprietà, Decorazioni**

[R27] la molteplicità minima deve essere un intero non negativo, minore o uguale alla molteplicità massima (se quest'ultima è specificata)

[R28] una nota deve stare nello stesso diagramma degli elementi a cui è associata

[R29] una nota non può essere associata a se stessa

## 3. Progettazione Logica

### 3.1 Passaggio allo schema relazionale

Come primo passo della traduzione verso il modello relazionale, risolviamo la gerarchia che ha origine nella classe Elemento. Tra gli approcci suggeriti da [ELMA] usiamo quello definito *relazioni multiple sottoclasse-superclasse*, che prevede una relazione per ogni classe specializzata, ovvero:

<b>Elemento</b> ( <u>eid</u> , ... )	
<b>Classe</b> ( <u>eid</u> , ... )	FK: Classe.eid → Elem.eid
<b>Associazione</b> ( <u>eid</u> , ... )	FK: Associazione.eid → Elem.eid
...	

Tale rappresentazione ha diverse proprietà desiderabili, tra cui la possibilità di modellare relazioni eterogenee ( es. 'ancorare' una Nota a un Elemento qualsiasi ) con semplici tabelle di composizione che referenziano Elemento.

Affrontiamo ora le relazioni di 'ownership', cioè di appartenenza tra gli elementi del diagramma, che sono: Diagramma e {Elemento}, Classe e {Classe, Attributo, Metodo}, Metodo e {Parametro}.

Tali relazioni, tutte di tipo 1 a molti, si possono realizzare nel modo seguente:

<b>Diagramma</b> ( <u>did</u> , ... )	
<b>Elemento</b> ( <u>eid</u> , <u>did</u> , ... )	FK: Elemento.did → Diagramma.did
<b>Classe</b> ( <u>eid</u> , <u>parentid</u> , ... )	FK: Classe.parentid → Classe.eid
<b>Attributo</b> ( <u>eid</u> , <u>classid</u> , ... )	FK: Attributo.classid → Classe.eid
<b>Metodo</b> ( <u>eid</u> , <u>classid</u> , ... )	FK: Metodo.eid → Classe.eid
<b>Parametro</b> ( <u>eid</u> , <u>metid</u> , ... )	FK: Parametro.metid → Metodo.eid
...	

Con questo schema risulta però complicato verificare una condizione fondamentale della nostra base di dati, ovvero che i costrutti possono essere associati fra loro solo se appartengono allo stesso diagramma. Per trovare il diagramma di appartenenza di un costrutto specializzato è sufficiente un JOIN, ma questo implica dover aggiungere un vincolo esplicito, trigger o asserzione, a ciascuna tabella specializzata e ad ogni tabella di composizione che realizza una relazione tra elementi. Una alternativa più pratica, può essere il seguente schema:

<b>Classe</b> ( <u>eid</u> , did, parentid. . . )	FK1: (eid,did) → Elemento (eid,did)
	FK2: (parentid,did) → Classe (eid,did)
<b>Attributo</b> ( <u>eid</u> , did, classid, . . . )	FK1: (eid,did) → Elemento (eid,did)
. . .	FK1: (classid,did) → Classe (eid,did)
<b>RelazioneY</b> ( <u>did</u> , elemA, elemB )	FK1: (elemA,did) → Elemento (eid,did)
. . .	FK2: (elemB,did) → Elemento (eid,did)

La ridondanza nei dati che abbiamo introdotto è minima ( l'ID di un diagramma può essere un semplice intero ), ma in compenso i soli vincoli di chiave esterna sono sufficienti a garantire l'integrità strutturale dei diagrammi e l'isolamento' dei costrutti contenuti in essi.

Ovviamente, oltre alle chiavi esterne andranno definiti vincoli UNIQUE(eid,did) in ciascuna tabella specializzata allo scopo di designare le chiavi candidate.

Seguendo questo principio possiamo tradurre il costrutto NodoAssociazione come segue:

<b>NodoAssoc</b> ( did, <u>classid</u> , <u>associd</u> , . . . )	FK1: (classid,did) → Classe (eid,did)
	FK2: (associd,did) → Associazione (eid,did)

Come prima, i vincoli di chiave esterna sono sufficienti a garantire che gli elementi messi in relazione, in questo caso le classi che partecipano ad una stessa associazione, siano contenuti nello stesso diagramma. Resta da risolvere però ancora un problema: se {classid,associd} è chiave primaria di NodoAssoc, è chiaro che una classe non può partecipare più di una volta alla stessa associazione, cioè non sono possibili associazioni *riflessive*. Modifichiamo perciò NodoAssoc come segue:

<b>NodoAssoc</b> ( <u>nodoid</u> , did, classid, associd, . . . )
---

Per le alte relazioni possiamo usare i pattern usuali [ELMA p.239] a seconda delle molteplicità degli elementi coinvolti. Ad es. per modellare una Generalizzazione (molti a molti), usiamo la relazione:

<b>Generalizzazione</b> ( did, <u>superid</u> , <u>subid</u> , . . . )	FK1: (superid,did) → Classe (eid,did)
	FK2: (subid,did) → Associazione (eid,did)

E così di seguito per i rimanenti costrutti.

### 3.2 Schema relazionale

**DIAGRAMMA** ( DID, Nome, Info )

**ELEMENTO** ( EID, DID, Stereotipo )

**CLASSE** ( EID, DID, ParentID, Nome, IsAstratta, IsInterfaccia, NomeQualif )

**ATTRIBUTO** ( EID, DID, ClassID, Nome, Tipo, Visibilita, IsDerivato, DefaultVal, MoltID )

**METODO** ( EID, DID, ClassID, Nome, TipoRet, Visibilita, IsAstratto )

**PARAMETRO** ( MetodoID, Nome, Tipo, Pos, Dir, DefaultVal )

**ASSOCIAZIONE** ( EID, DID, Nome )

**NODOASSOC** ( EID, DID, ClassID, AssocID, Ruolo, IsNavigabile, IsAggregato,  
IsComposito, MoltID )

**CLASSEASSOC** ( DID, ClassID, AssocID )

**GENERALIZZAZIONE** ( DID, SuperID, SubID, SetID )

**GENERALIZATIONSET** ( SetID, SuperID, Nome, IsCovering, IsDisjoint )

**REALIZZAZIONE** ( DID, SpecificID, ImplemID )

**DIPENDENZA** ( DID, ClientID, SupplierID, Stereotipo )

**MOLTEPLICITA** ( MoltID, Lower, Upper, IsOrdered, IsUnique )

**TIPOPRIMITIVO** ( DID, TipoID, Nome, IsEnum )

**LETTERALE** ( TipoID, Valore )

**NOTA** ( EID, DID, Valore )

**ANCHOR** ( DID, NotaID, TargetID )

**CONSTRAINT** ( DID, Target1, Target2, Stereotipo )

### 3.3 Implementazione delle tabelle

#### 3.3.1 Tabella MOLTEPLICITA

- il vincolo mol\_c3 verifica che, dove specificato, 'upper' sia sempre maggiore o uguale a 'lower' [R27]

```
CREATE TABLE molteplicita
(
  moltid      INTEGER      PRIMARY KEY,
  lower       INTEGER      NOT NULL,
  upper       INTEGER,
  is_unique   CHAR(1)     DEFAULT 'T',
  is_ordered  CHAR(1)     DEFAULT 'F',

  CONSTRAINT mol_c1 CHECK (is_unique IN (NULL, 'T', 'F')),
  CONSTRAINT mol_c2 CHECK (is_ordered IN (NULL, 'T', 'F')),
  CONSTRAINT mol_c3 CHECK (lower >= 0 AND upper IS NULL OR upper >= lower )
);
```

#### 3.3.2 Tabella DIAGRAMMA

```
CREATE TABLE diagramma
(
  did      INTEGER      PRIMARY KEY,
  nome     VARCHAR2(255) NOT NULL,
  info     VARCHAR2(255),

  CONSTRAINT dig_c1 UNIQUE (nome)
);
```

#### 3.3.3 Tabella ELEMENTO

- il vincolo ele\_fk garantisce che tutti gli elementi contenuti in un diagramma siano cancellati quando quest'ultimo viene rimosso dalla base di dati [R01]

```
CREATE TABLE elemento
(
  eid      INTEGER      PRIMARY KEY,
  did      INTEGER      NOT NULL,

  CONSTRAINT ele_fk FOREIGN KEY (did)
    REFERENCES diagramma ON DELETE CASCADE,
  CONSTRAINT ele_c1 UNIQUE (eid, did)
);
```

#### 3.3.4 Tabella CLASSE

- il vincolo cls\_fk2 verifica che una classe membro sia associata allo stesso diagramma della classe esterna che la contiene [R04]
- il vincolo cls\_c2 garantisce che le classi che appartengono allo stesso ambito (namespace) abbiano nomi fra loro distinti [R05]

```
CREATE TABLE classe
(
  eid      INTEGER      PRIMARY KEY,
  did      INTEGER      NOT NULL,
  parentid INTEGER,
```

```

nome          VARCHAR2(255) NOT NULL,
nomequalif   VARCHAR2(255),
is_astratta  CHAR(1)      DEFAULT 'F' NOT NULL,
is_interfaccia CHAR(1)    DEFAULT 'F' NOT NULL,
temp         INTEGER      DEFAULT 0,

CONSTRAINT cls_fk1 FOREIGN KEY (eid,did)
  REFERENCES elemento (eid,did) ON DELETE CASCADE,
CONSTRAINT cls_fk2 FOREIGN KEY (parentid,did)
  REFERENCES classe (eid,did) ON DELETE CASCADE,
CONSTRAINT cls_c1 UNIQUE (eid,did),
CONSTRAINT cls_c2 UNIQUE (nome,did,parentid),
CONSTRAINT cls_c3 CHECK (is_astratta IN ('T','F') AND
  is_interfaccia IN ('T','F') AND
  NOT(is_astratta='T' AND is_interfaccia='T' ))
);

```

### 3.3.5 Tabella ATTRIBUTO

- il vincolo att\_c1 verifica che gli attributi di una classe abbiano nomi distinti [R07]

```

CREATE TABLE attributo
(
  eid          INTEGER          PRIMARY KEY,
  did          INTEGER          NOT NULL,
  classid     INTEGER          NOT NULL,
  nome        VARCHAR2(255)    NOT NULL,
  tipo        VARCHAR2(255),
  visibilita  VARCHAR2(255),
  moltid      INTEGER          REFERENCES molteplicita
  ON DELETE SET NULL,
  is_derivato CHAR(1)          DEFAULT 'F' NOT NULL,
  defaultval  VARCHAR2(255),

  CONSTRAINT att_fk1 FOREIGN KEY (eid,did)
    REFERENCES elemento (eid,did) ON DELETE CASCADE,
  CONSTRAINT att_fk2 FOREIGN KEY (classid,did)
    REFERENCES classe (eid,did) ON DELETE CASCADE,
  CONSTRAINT att_c1 UNIQUE (nome,classid),
  CONSTRAINT att_c2 CHECK (is_derivato IN ('T','F')),
  CONSTRAINT att_c3 CHECK (visibilita IN (NULL,'private',
    'package','protected','public'))
);

```

### 3.3.6 Tabella METODO

```

CREATE TABLE metodo
(
  eid          INTEGER          PRIMARY KEY,
  did          INTEGER          NOT NULL,
  classid     INTEGER          NOT NULL,
  nome        VARCHAR2(255)    NOT NULL,
  tiporet     VARCHAR2(255),
  visibilita  VARCHAR2(255),
  is_astratto CHAR(1)          DEFAULT 'F' NOT NULL,

```

```

CONSTRAINT met_fk1 FOREIGN KEY (eid,did)
  REFERENCES elemento (eid,did) ON DELETE CASCADE,
CONSTRAINT met_fk2 FOREIGN KEY (classid,did)
  REFERENCES classe (eid,did) ON DELETE CASCADE,
CONSTRAINT met_c1 CHECK (is_astratto IN ('T','F')),
CONSTRAINT met_c2 CHECK (visibilita IN (NULL,'private',
  'package','protected','public'))
);

```

### 3.3.7 Tabella PARAMETRO

```

CREATE TABLE parametro
(
  metodoid      INTEGER          NOT NULL REFERENCES metodo (eid)
                ON DELETE CASCADE,
  nome          VARCHAR2(255)    NOT NULL,
  tipo          VARCHAR2(255),
  pos           INTEGER          NOT NULL CHECK (pos>0),
  dir           VARCHAR2(255),
  default_val   VARCHAR2(255),

  CONSTRAINT parm_pk PRIMARY KEY (nome,metodoid),
  CONSTRAINT parm_c1 CHECK (dir IN (NULL,'in','out','inout','return')),
  CONSTRAINT parm_c2 UNIQUE (metodoid,pos)
);

```

### 3.3.8 Tabella ASSOCIAZIONE

```

CREATE TABLE associazione
(
  eid  INTEGER          PRIMARY KEY,
  did  INTEGER          NOT NULL,
  nome VARCHAR2(255),

  CONSTRAINT ass_fk FOREIGN KEY (eid,did)
    REFERENCES elemento (eid,did) ON DELETE CASCADE,
  CONSTRAINT ass_c1 UNIQUE (eid,did)
);

```

### 3.3.9 Tabella NODOASSOC

• i vincoli nod\_fk1 nod\_fk2 garantiscono che le classi che partecipano ad una associazione, e l'associazione stessa, appartengano allo stesso diagramma [R14]

```

CREATE TABLE nodoassoc
(
  eid          INTEGER          PRIMARY KEY,
  did          INTEGER          NOT NULL,
  classid     INTEGER          NOT NULL,
  associd     INTEGER          NOT NULL,
  ruolo       VARCHAR2(255),
  is_navigabile CHAR(1)        DEFAULT 'F' NOT NULL,
  is_aggregato CHAR(1)        DEFAULT 'F' NOT NULL,
  is_composito CHAR(1)        DEFAULT 'F' NOT NULL,
  moltid     INTEGER          REFERENCES molteplicita
                ON DELETE SET NULL,

```

```

CONSTRAINT nod_fk1 FOREIGN KEY (eid, did)
  REFERENCES elemento ON DELETE CASCADE,
CONSTRAINT nod_fk2 FOREIGN KEY (classid, did)
  REFERENCES classe (eid,did) ON DELETE CASCADE,
CONSTRAINT nod_fk3 FOREIGN KEY (associd, did)
  REFERENCES associazione (eid, did) ON DELETE CASCADE,
CONSTRAINT nod_c1 UNIQUE (classid, associd, ruolo),
CONSTRAINT nod_c2 CHECK (
  is_navigabile IN ('T','F') AND
  is_aggregato IN ('T','F') AND
  is_composito IN ('T','F') AND
  NOT(is_composito='T' AND is_aggregato='T'))
);

```

### 3.3.10 Tabella CLASSEASSOC

- i vincoli di chiave esterna cas\_fk1 e cas\_fk2 verificano che la classe e l'associazione che formano una association class siano collocati nello stesso diagramma [R19]
- il vincolo cas\_c1 si assicura che una singola classe non possa formare più association class [Rxx]

```

CREATE TABLE classeassoc
(
  did          INTEGER NOT NULL,
  classid     INTEGER NOT NULL,
  associd     INTEGER PRIMARY KEY,

  CONSTRAINT cas_fk1 FOREIGN KEY (classid, did)
    REFERENCES classe (eid,did) ON DELETE CASCADE,
  CONSTRAINT cas_fk2 FOREIGN KEY (associd, did)
    REFERENCES associazione (eid, did) ON DELETE CASCADE,
  CONSTRAINT cas_c1 UNIQUE (classid)
);

```

### 3.3.11 Tabella GENERALIZATIONSET

```

CREATE TABLE generalizationset
(
  setid          INTEGER PRIMARY KEY,
  superid       INTEGER NOT NULL
                REFERENCES classe(eid) ON DELETE CASCADE,
  nome          VARCHAR2(255),
  is_covering   CHAR(1),
  is_disjoint   CHAR(1),

  CONSTRAINT gset_fk FOREIGN KEY (superid)
    REFERENCES classe (eid) ON DELETE CASCADE,
  CONSTRAINT gset_c1 UNIQUE (setid, superid),
  CONSTRAINT gset_c2 CHECK (is_covering IN (NULL,'T','F') AND
                            is_disjoint IN (NULL,'T','F'))
);

```

### 3.3.12 Tabella GENERALIZZAZIONE

- i vincoli gen\_fk1 e gen\_fk2 garantiscono che un tipo base ed ogni sua specializzazione appartengano allo stesso diagramma [R20]
- il vincolo gen\_fk3 garantisce che le generalizzazioni che fanno parte di un generalization set condividano lo stesso tipo base [R23]

```
CREATE TABLE generalizzazione
(
    superid    INTEGER    NOT NULL,
    subid      INTEGER    NOT NULL,
    did        INTEGER    NOT NULL,
    setid      INTEGER,

    CONSTRAINT gen_pk PRIMARY KEY (superid,subid),
    CONSTRAINT gen_fk1 FOREIGN KEY (superid,did)
        REFERENCES classe (eid,did) ON DELETE CASCADE,
    CONSTRAINT gen_fk2 FOREIGN KEY (subid,did)
        REFERENCES classe (eid,did) ON DELETE CASCADE,
    CONSTRAINT gen_fk3 FOREIGN KEY (setid,superid)
        REFERENCES generalizationset (setid,superid) ON DELETE CASCADE
);
```

### 3.3.13 Tabella REALIZZAZIONE

- i vincoli rel\_fk1 e rel\_fk2 verificano che gli elementi collegati dalla relazione siano definiti nello stesso diagramma [R24]

```
CREATE TABLE realizzazione
(
    did          INTEGER    NOT NULL,
    specifid     INTEGER    NOT NULL,
    implemid     INTEGER    NOT NULL,

    CONSTRAINT rel_pk PRIMARY KEY (specifid,implemid),
    CONSTRAINT rel_fk1 FOREIGN KEY (specifid,did)
        REFERENCES classe (eid,did) ON DELETE CASCADE,
    CONSTRAINT rel_fk2 FOREIGN KEY (implemid,did)
        REFERENCES classe (eid,did) ON DELETE CASCADE
);
```

### 3.3.14 Tabella DIPENDENZA

- i vincoli dip\_fk1 e dip\_fk2 verificano che gli elementi collegati dalla relazione siano definiti nello stesso diagramma [R24]

```
CREATE TABLE dipendenza
(
    did          INTEGER    NOT NULL,
    supplierid   INTEGER    NOT NULL,
    clientid     INTEGER    NOT NULL,
    stereotipo   VARCHAR2(255),

    CONSTRAINT dip_pk PRIMARY KEY (supplierid,clientid),
    CONSTRAINT dip_fk1 FOREIGN KEY (supplierid,did)
        REFERENCES elemento (eid,did) ON DELETE CASCADE,
    CONSTRAINT dip_fk2 FOREIGN KEY (clientid,did)
        REFERENCES elemento (eid,did) ON DELETE CASCADE
);
```

### 3.3.15 Tabella TIOPRIMITIVO

- il vincolo tip\_c1 garantisce che i tipi primitivi definiti in un diagramma abbiano nomi distinti [R06]

```
CREATE TABLE tipoprimitivo
(
  did          INTEGER          NOT NULL
              REFERENCES diagramma ON DELETE CASCADE,
  tipoid       INTEGER          PRIMARY KEY,
  nome         VARCHAR2(255)    NOT NULL,
  is_enum      CHAR(1)         DEFAULT 'F' NOT NULL,

  CONSTRAINT tip_c1 UNIQUE (did,nome),
  CONSTRAINT tip_c2 CHECK (is_enum IN ('T','F'))
);
```

### 3.3.16 Tabella LETTERALE

```
CREATE TABLE letterale
(
  valore       VARCHAR2(255)    NOT NULL,
  tipoid       INTEGER          NOT NULL
              REFERENCES tipoprimitivo ON DELETE CASCADE,

  CONSTRAINT let_pk PRIMARY KEY (valore,tipoid)
);
```

### 3.3.17 Tabella NOTA

```
CREATE TABLE nota
(
  eid         INTEGER          PRIMARY KEY,
  did         INTEGER          NOT NULL,
  valore      VARCHAR2(255)    NOT NULL,

  CONSTRAINT nota_fk FOREIGN KEY (eid, did)
              REFERENCES elemento (eid, did) ON DELETE CASCADE,
  CONSTRAINT nota_c1 UNIQUE (eid, did)
);
```

### 3.3.18 Tabella ANCHOR

- i vincoli anc\_fk1 e anc\_fk2 garantiscono che una nota e tutti gli elementi a cui essa è associata appartengano allo stesso diagramma [R28]
- il vincolo anc\_c1 si assicura che una nota non sia associata a se stessa [R29]

```
CREATE TABLE anchor
(
  did         INTEGER NOT NULL,
  notaid      INTEGER NOT NULL,
  targetid    INTEGER NOT NULL,

  CONSTRAINT anc_pk PRIMARY KEY (notaid,targetid),
  CONSTRAINT anc_fk1 FOREIGN KEY (notaid,did)
              REFERENCES nota (eid,did) ON DELETE CASCADE,
  CONSTRAINT anc_fk2 FOREIGN KEY (targetid,did)
              REFERENCES elemento (eid,did) ON DELETE CASCADE,
  CONSTRAINT anc_c1 CHECK (targetid <> notaid));
```

### 3.3.19 Tabella CONSTRAINT

```
CREATE TABLE constraint
(
  did      INTEGER      NOT NULL,
  target1  INTEGER      NOT NULL,
  target2  INTEGER,
  valore   VARCHAR2(255) NOT NULL,

  CONSTRAINT con_pk PRIMARY KEY (target1, valore),
  CONSTRAINT con_fk1 FOREIGN KEY (target1, did)
    REFERENCES elemento (eid, did) ON DELETE CASCADE,
  CONSTRAINT con_fk2 FOREIGN KEY (target2, did)
    REFERENCES elemento (eid, did) ON DELETE CASCADE
);
```

### 3.3.19 Tabelle TEMP

tabella di supporto utilizzata per memorizzare i calcoli intermedi di procedure e trigger

```
CREATE TABLE temp
(
  eid INTEGER PRIMARY KEY
);
```

## 3.4 Implementazione dei vincoli

### 3.4.1 Vincolo check\_for\_class\_loops

Verifica che non vi siano classi che sono membri di se stesse [RO3], e contestualmente 'aggiorna' il nome qualificato di tutte le classi contenute nella base di dati

```
CREATE OR REPLACE TRIGGER check_for_class_loops
AFTER INSERT OR UPDATE OF nome, parentid ON classe
DECLARE
    CURSOR cur_root IS SELECT * FROM classe WHERE parentid IS NULL;
    CURSOR cur_child IS SELECT * FROM classe
                        WHERE parentid IN (SELECT eid FROM temp) AND
                        eid NOT IN (SELECT eid FROM temp);
    parent_nome classe.nome%TYPE;
    num_child INTEGER;
    num_classi INTEGER;
BEGIN
    -- inserisci nella tabella Temp tutte le classi top level (cioè non
    -- interne ad altre classi). Per tali classi nomequalif = nome
    DELETE FROM temp;
    FOR rec_root IN cur_root
    LOOP
        UPDATE classe SET nomequalif = rec_root.nome
        WHERE eid = rec_root.eid;
        INSERT INTO temp VALUES (rec_root.eid);
    END LOOP;
    -- inserisci in Temp le classi interne di ogni classe già contenuta in
    -- Temp, finchè tutte le classi figlie non sono state processate
    LOOP
        num_child := 0;
        FOR rec_child IN cur_child
        LOOP
            SELECT nomequalif INTO parent_nomequalif FROM classe
            WHERE eid = rec_child.parentid;

            UPDATE classe C SET nomequalif = parent_nomequalif||'.'||C.nome
            WHERE C.eid = rec_child.eid;

            INSERT INTO temp VALUES (rec_child.eid);
            num_child := num_child + 1;
        END LOOP;
        EXIT WHEN num_child = 0; -- non ci sono più classi figlie da inserire
    END LOOP;
    -- al termine del processo, se non vi sono cicli, Temp deve contenere ogni
    -- elemento contenuto in Classe (cioè le loro cardinalità coincidono)
    SELECT COUNT(*) INTO num_child FROM temp;
    SELECT COUNT(*) INTO num_classi FROM classe;
    IF num_child <> num_classi THEN
        Raise_Application_Error(-20003, 'una o più classi sono membro di
        se stesse');
    END IF;
END;
```

### 3.4.2 Vincolo check\_no\_same\_signature

Verifica che nella stessa classe non vi siano metodi con uguale signature [R09]. Fa uso della funzione get\_signature().

```
-- ritorna la signature del metodo di chiave 'mid'
CREATE OR REPLACE FUNCTION get_signature (mid metodo.eid%TYPE) RETURN VARCHAR2
IS
    signature VARCHAR2(255);
BEGIN
    SELECT nome INTO signature FROM metodo WHERE eid = mid;

    FOR rec_param IN (SELECT * FROM parametro
                      WHERE metodoid = mid
                      ORDER BY pos)
    LOOP
        signature := signature || '%' || rec_param.nome;
        IF rec_param.tipo IS NOT NULL THEN
            signature := signature || rec_param.tipo;
        END IF;
    END LOOP;
    RETURN signature;
END;
/

CREATE ASSERTION check_no_same_signature AS
CHECK ( NOT EXISTS (
    SELECT *
    FROM metodo M1 JOIN metodo M2 ON (M1.classid = M2.classid AND
                                       M1.eid <> M2.eid )
    WHERE get_signature(M1.eid) = get_signature(M2.eid)));

-- vista di utilità che mostra, di ogni metodo nel database, id, nome
-- numero di parametri e signature
CREATE OR REPLACE VIEW vw_info_metodi AS
    SELECT M.eid, M.nome, COUNT(P.nome) AS nparams,
           get_signature(M.eid) AS signature
    FROM metodo M LEFT OUTER JOIN parametro P ON P.metodoid = M.eid
    GROUP BY M.eid, M.nome;
```

### 3.4.3 Vincolo check\_attrib\_type

Verifica che il tipo di un attributo, dove specificato, sia il nome di una classe o di un tipo primitivo definiti nello stesso diagramma che contiene la classe in cui è definito l'attributo [R09].

Lo stesso check può essere effettuato, con formulazione analoga, per il return type di un metodo

```
CREATE ASSERTION check_attrib_type
CHECK ( NOT EXISTS (
    SELECT *
    FROM attributo A
    WHERE A.tipo IS NOT NULL AND
          A.tipo NOT IN (SELECT T.nome
                        FROM tipoprimitivo T
                        WHERE T.did = A.did) AND
          A.tipo NOT IN (SELECT C.nomequalif,
                        FROM classe C
                        WHERE C.did = A.did));
```

### 3.4.4 Vincolo check\_param\_type

Verifica che il tipo di un parametro, se specificato, sia il nome di una classe o di un tipo primitivo definiti nello stesso diagramma della classe in cui è definito il metodo a cui è associato il parametro [R11]

```
CREATE ASSERTION check_param_type
CHECK ( NOT EXISTS (
    SELECT *
    FROM parametro P JOIN metodo M on P.metid = M.eid
    WHERE P.tipo IS NOT NULL AND
        P.tipo NOT IN (SELECT T.nome
                        FROM tipoprimitivo T
                        WHERE T.did = M.did) AND
        P.tipo NOT IN (SELECT C.nomequalif,
                        FROM classe C
                        WHERE C.did = M.did));
```

### 3.4.5 Vincolo check\_attrib\_defaultval

Verifica che, se a un attributo è fornito il valore di default e il tipo dell'attributo è una enumerazione, allora il valore di default deve essere uno dei letterali della enum [R12].

```
CREATE ASSERTION check_attrib_defaultval
CHECK ( NOT EXISTS (
    SELECT *
    FROM attributo A JOIN tipoprimitivo T ON A.tipo = T.nome
    WHERE T.is_enum = 'T' AND
        A.defaultval IS NOT NULL AND
        A.defaultval NOT IN (SELECT L.valore,
                              FROM letterale L
                              WHERE L.tipoid = T.tipoid));
```

### 3.4.6 Vincolo check\_abstract\_ops

- il seguente vincolo si assicura che solo classi astratte e interfacce possano definire metodi astratti [R13]

```
CREATE ASSERTION check_abstract_ops
CHECK ( NOT EXISTS (
    SELECT *
    FROM metodo
    WHERE is_astratto = 'T' AND
        classid NOT IN (SELECT eid FROM classe
                        WHERE is_interfaccia='T' OR is_astratta='T')));
```

### 3.4.7 Vincolo validate\_nodoassoc

- il seguente vincolo verifica che il numero massimo di estremi aggregati/compositi sia =1 per le associazioni binarie [R15] e =0 per le associazioni n-arie [R16] e infine che i ruoli di una associazione, dove specificati, siano tutti diversi fra loro [R17]

```
CREATE OR REPLACE TRIGGER validate_nodoassoc
AFTER INSERT OR UPDATE ON nodoassoc
BEGIN
    FOR rec IN (SELECT associd AS eid FROM nodoassoc
                WHERE is_aggregato='T' OR is_composito='T'
                GROUP BY associd HAVING COUNT(*) > 1)
    LOOP
        Raise_Application_Error(-20001, 'l'associazione eid=' || rec.eid ||
            ' contiene due o più nodi aggregati/compositi');
    END LOOP;
```

```

FOR rec IN (SELECT associd AS eid
            FROM nodoassoc
            WHERE associd IN (SELECT N.associd FROM nodoassoc N
                              WHERE N.is_aggregato='T' OR
                                    N.is_composito='T'))
            GROUP BY associd
            HAVING COUNT(*) > 2)
LOOP
  Raise_Application_Error(-20002, 'l'associazione n-aria eid=' ||
    rec.eid || ' ha un membro aggregato/composito');
END LOOP;

FOR rec IN (SELECT associd AS eid
            FROM nodoassoc
            WHERE ruolo IS NOT NULL
            GROUP BY associd, ruolo
            HAVING COUNT(*) > 1)
LOOP
  Raise_Application_Error(-20003, 'due o più membri dell'associazione
    eid=' || rec.eid || ' hanno ruoli uguali');
END LOOP;
END;
/

```

### 3.4.8 Vincolo check\_no\_self\_assoc

Verifica che una association class non possa avere se stessa come estremo [R18]

```

CREATE ASSERTION check_no_self_assoc
CHECK ( NOT EXISTS (
  FOR rec IN (SELECT *
              FROM nodoassoc
              WHERE (associd, classid) IN (SELECT C.associd, C.classid
                                           FROM classeassoc C)))));

```

### 3.4.9 Vincolo check\_gen\_type

Si assicura che una classe possa essere specializzata solo da altre classi e una interfaccia solo da altre interfacce [R22]

```

CREATE ASSERTION check_gen_types
CHECK ( NOT EXISTS (
  SELECT *
  FROM generalizzazione JOIN classe SUP ON superid = SUP.eid
    JOIN classe SUB ON subid = SUB.eid
  WHERE SUP.is_interfaccia <> SUB.is_interfaccia ));

```

### 3.4.10 Vincolo check\_realiz\_type

Si assicura che una realizzazione possa essere stabilita solo tra una classe e una interfaccia [R26]

```

CREATE ASSERTION check_realiz_types
CHECK ( NOT EXISTS (
  SELECT *
  FROM realizzazione JOIN classe IM ON implemid = IM.eid
    JOIN classe SP ON specificad = SP.eid
  WHERE IM.is_interfaccia = 'T' OR
        SP.is_interfaccia = 'F' ));

```

### 3.4.11 Vincolo check\_gen\_loops

Verifica che non vi siano classi che sono specializzazioni di se stesse [R21]. Il vincolo fa uso della procedura DFS\_loop che esegue una dfs sul grafo orientato delle gerarchie, a partire dal nodo 'nodoid', e solleva una eccezione non appena viene incontrato un ciclo. Nota: anche se in Oracle vi è un limite di massimo 50 chiamate ricorsive, il numero delle componenti connesse di una gerarchia in un diagramma tipico è di gran lunga inferiore a tale limite.

```
CREATE OR REPLACE PROCEDURE DFS_loop ( nodoid classe.eid%TYPE )
IS
    SCOPERTO CONSTANT INTEGER := 1;
    VISITATO CONSTANT INTEGER := 2;
BEGIN
    -- marca il nodo corrente come scoperto
    UPDATE classe SET temp = SCOPERTO WHERE eid = nodoid;

    FOR child IN (SELECT G.subid, C.temp
                  FROM generalizzazione G JOIN classe C
                  ON G.subid=C.eid
                  WHERE G.superid=nodoid AND C.temp<>VISITATO)
    LOOP
        -- i 'figli' di nodoid che sono scoperti ma non visitati
        -- completamente, sono 'avi' di nodoid
        -- (sono cioè parte di un ciclo)
        IF child.temp = SCOPERTO THEN
            Raise_Application_Error(-20010, 'sono presenti cicli nella
                                         gerarchia di classi');
        ELSE
            DFS_loop( child.subid );
        END IF;
    END LOOP;

    -- marca il nodo corrente come visitato
    UPDATE classe SET temp = VISITATO WHERE eid = nodoid;
END;
/
```

```
CREATE OR REPLACE TRIGGER check_gen_loops
AFTER INSERT ON generalizzazione
BEGIN
    -- marca tutti i nodi come non visitati
    UPDATE classe SET temp = 0;

    -- esegui una DFS su ogni nodo non ancora visitato
    FOR nodo IN (SELECT * FROM classe WHERE temp = 0)
    LOOP
        DFS_loop( nodo.eid );
    END LOOP;
END;
/
```

### 3.4.12 Vincoli aggiuntivi

Di seguito sono definiti alcuni vincoli che non riguardano la semantica di UML ma servono a semplificare la gestione del sistema di tabelle 'specializzate' usato per realizzare la gerarchia di costrutti.

- il seguente vincolo si assicura che due tabelle specializzate non contengano le stesse chiavi (superfluo se non si effettuano inserimenti espliciti nella tabella Elemento)

```
CREATE ASSERTION check_no_overlap
CHECK ( NOT EXISTS (
    (SELECT eid FROM classe) INTERSECT
    (SELECT eid FROM attributo) INTERSECT
    (SELECT eid FROM metodo) INTERSECT
    (SELECT eid FROM associazione) INTERSECT
    (SELECT eid FROM nota)));
```

- quando viene inserito un nuovo elemento specializzato ( classe, metodo, ecc. ), crea una riga corrispondente nella tabella Elemento con l'id e il tipo del nuovo elemento inserito.

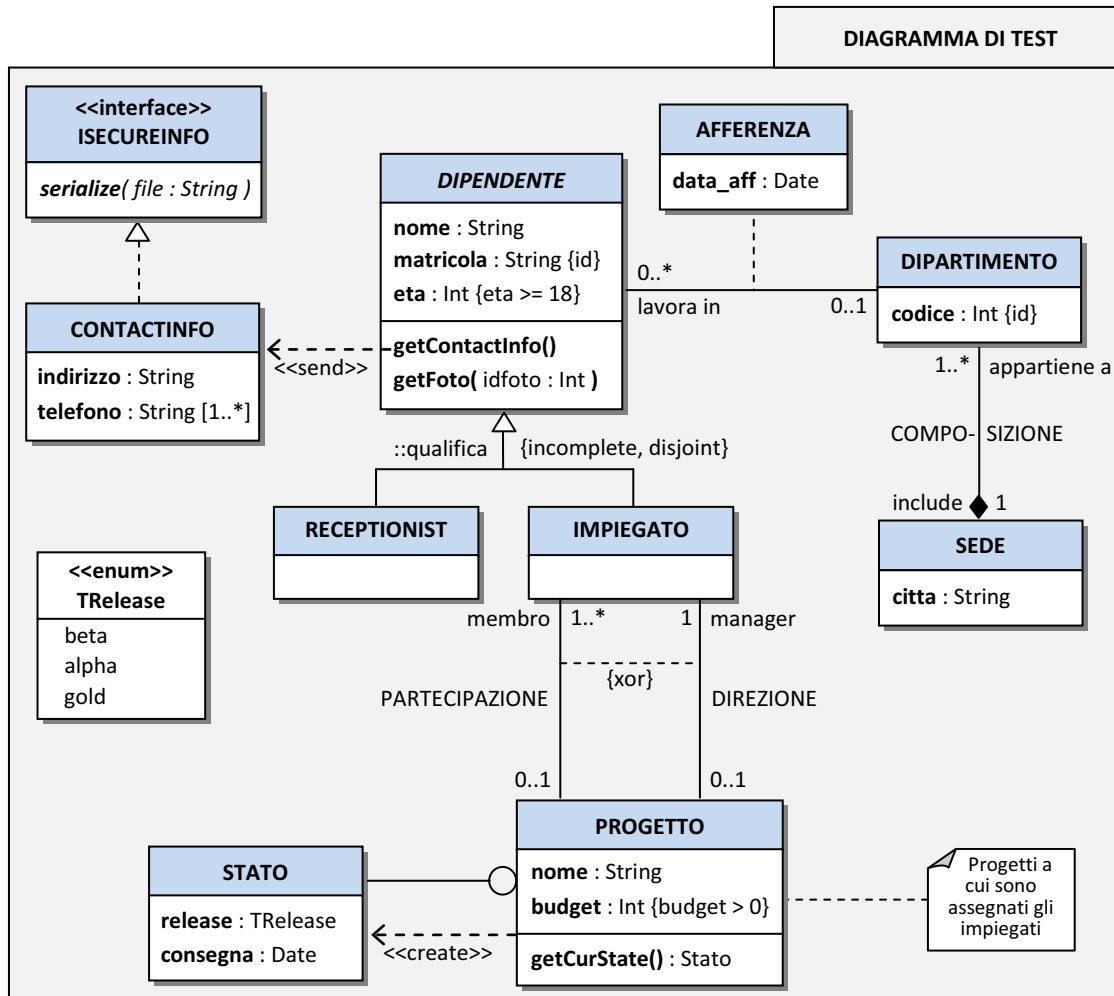
```
CREATE OR REPLACE TRIGGER classe_before_insert
BEFORE INSERT ON classe FOR EACH ROW
BEGIN
    INSERT INTO elemento VALUES (:NEW.eid, :NEW.did, 'Classe');
END;
/
CREATE OR REPLACE TRIGGER attrib_before_insert
BEFORE INSERT ON attributo FOR EACH ROW
BEGIN INSERT INTO elemento VALUES (:NEW.eid, :NEW.did, 'Attributo'); END;
/
CREATE OR REPLACE TRIGGER metodo_before_insert
BEFORE INSERT ON metodo FOR EACH ROW
BEGIN INSERT INTO elemento VALUES (:NEW.eid, :NEW.did, 'Metodo'); END;
/
CREATE OR REPLACE TRIGGER associazione_before_insert
BEFORE INSERT ON associazione FOR EACH ROW
BEGIN INSERT INTO elemento VALUES (:NEW.eid, :NEW.did, 'Associazione'); END;
/
CREATE OR REPLACE TRIGGER associazione_before_insert
BEFORE INSERT ON nota FOR EACH ROW
BEGIN INSERT INTO elemento VALUES (:NEW.eid, :NEW.did, 'Nota'); END;
/
CREATE OR REPLACE TRIGGER nodoassoc_before_insert
BEFORE INSERT ON nodoassoc FOR EACH ROW
BEGIN INSERT INTO elemento VALUES (:NEW.eid, :NEW.did, 'NodoAssoc'); END;
/
```

- quando un elemento di una tabella specializzata viene rimosso, elimina la riga corrispondente di Elemento. I trigger per le restanti tabelle specializzate, qui omessi per brevità, seguono la stessa formulazione.

```
CREATE OR REPLACE TRIGGER classe_after_delete
AFTER DELETE ON classe
BEGIN
    FOR rec_elem IN (SELECT * FROM elemento E
                    WHERE E.tipo = 'Classe' AND
                    E.eid NOT IN (SELECT eid FROM classe))
    LOOP
        DELETE FROM elemento WHERE eid = rec_elem.eid;
    END LOOP;
END;
/
```

## 4. Test Case

### 4.1 Diagramma di Esempio



### 4.2 Stampa delle tabelle Oracle

#### 4.2.1 Tabella MOLTEPLICITA

MOLTID	LOWER	UPPER	IS_UNIQUE	IS_ORDERED
50	0	1	-	-
51	1	1	-	-
52	0	-	-	-
53	1	-	-	-

riga/e 1 - 4 di 4

#### 4.2.2 Tabella DIAGRAMMA

DID	HOME	INFO
1	Diagramma di Test	by Nicola Di Leva 08-09
riga/e 1 - 1 di 1		

#### 4.2.3 Tabelle TIPOPRIMITIVO e LETTERALE

DID	TIPOID	HOME	IS_EHUM
1	10	String	F
1	11	Int	F
1	12	Date	F
1	13	TRelease	T
riga/e 1 - 4 di 4			

TIPOID	VALORE
13	alpha
13	beta
13	gold
riga/e 1 - 3 di 3	

#### 4.2.4 Tabella ELEMENTO

EID	DID	TIPO
100	1	Classe
101	1	Classe
102	1	Classe
103	1	Classe
104	1	Classe
105	1	Classe
106	1	Classe
107	1	Classe
108	1	Classe
109	1	Classe
200	1	Associazione
201	1	Associazione
202	1	Associazione
203	1	Associazione
300	1	NodoAssoc
riga/e 1 - 15 di 39		▶

EID	DID	TIPO
301	1	NodoAssoc
302	1	NodoAssoc
303	1	NodoAssoc
304	1	NodoAssoc
305	1	NodoAssoc
306	1	NodoAssoc
307	1	NodoAssoc
400	1	Attributo
401	1	Attributo
402	1	Attributo
403	1	Attributo
404	1	Attributo
405	1	Attributo
406	1	Attributo
407	1	Attributo
riga/e 16 - 30 di 39		▶

EID	DID	TIPO
408	1	Attributo
409	1	Attributo
410	1	Attributo
411	1	Attributo
500	1	Metodo
501	1	Metodo
502	1	Metodo
503	1	Metodo
600	1	Nota
riga/e 31 - 39 di 39		

## 4.2.5 Tabella CLASSE

EID	DID	PARENTID	HOME	HOMEQUALIF	IS_ASTRATTA	IS_INTERFACCIA	TEMP
100	1	-	Dipendente	Dipendente	F	F	2
101	1	-	Receptionist	Receptionist	F	F	2
102	1	-	Impiegato	Impiegato	F	F	2
103	1	-	ContactInfo	ContactInfo	F	F	2
104	1	-	ISecureInfo	ISecureInfo	F	F	2
105	1	-	Dipartimento	Dipartimento	F	F	2
106	1	-	Sede	Sede	F	F	2
107	1	-	Afferenza	Afferenza	F	F	2
108	1	-	Progetto	Progetto	F	F	2
109	1	108	Stato	Progetto.Stato	F	F	2
riga/e 1 - 10 di 10							

## 4.2.6 Tabella ATTRIBUTO

EID	DID	CLASSID	HOME	TIPO	VISIBILITA	MOLTID	IS_DERIVATO	DEFAULTVAL
400	1	100	nome	String	-	-	F	-
401	1	100	matricola	String	-	-	F	-
402	1	100	eta	Int	-	-	F	-
403	1	103	indirizzo	String	-	-	F	-
404	1	103	telefono	String	-	51	F	-
405	1	105	codice	Int	-	-	F	-
406	1	106	citta	String	-	-	F	-
407	1	107	data_aff	Date	-	-	F	-
408	1	108	nome	String	-	-	F	-
409	1	108	budget	Int	-	-	F	-
410	1	109	release	TRelease	-	-	F	-
411	1	109	consegna	TDate	-	-	F	-
riga/e 1 - 12 di 12								

#### 4.2.7 Tabella METODO

EID	DID	CLASSID	HOME	TIPORET	VISIBILITA	IS_ASTRATTO
500	1	100	getContactInfo	-	-	F
501	1	100	getFoto	-	-	F
502	1	108	getCurState	Progetto.Stato	-	F
503	1	104	serialize	Progetto.Stato	-	T
riga/e 1 - 4 di 4						

#### 4.2.8 Tabella PARAMETRO

METODOID	HOME	TIPO	POS	DIR	DEFAULT_VAL
501	idfoto	Int	1	-	-
503	file	String	1	-	-
riga/e 1 - 2 di 2					

#### 4.2.9 Tabelle ASSOCIAZIONE e CLASSEASSOC

EID	DID	HOME
200	1	Afferenza
201	1	Partecipazione
202	1	Direzione
203	1	Composizione
riga/e 1 - 4 di 4		

DID	CLASSID	ASSOCID
1	107	200
riga/e 1 - 1 di 1		

#### 4.2.10 Tabella NODOASSOC

EID	DID	CLASSID	ASSOCID	RUOLO	IS_NAVIGABILE	IS_AGGREGATO	IS_COMPOSITO	MOLTID
300	1	100	200	lavora in	F	F	F	52
301	1	105	200	-	F	F	F	50
302	1	102	201	membro	F	F	F	53
303	1	108	200	-	F	F	F	50
304	1	102	201	manager	F	F	F	51
305	1	108	200	-	F	F	F	50
306	1	106	203	include	F	F	T	51
307	1	105	203	appartiene a	F	F	F	53
riga/e 1 - 8 di 8								

#### 4.2.11 Tabelle GENERALIZATIONSET e GENERALIZZAZIONE

SETID	SUPERID	HOME	IS_COVERING	IS_DISJOINT
700	100	qualifica	F	T
riga/e 1 - 1 di 1				

SUPERID	SUBID	DID	SETID
100	101	1	700
100	102	1	700
riga/e 1 - 2 di 2			

#### 4.2.12 Tabella REALIZZAZIONE

DID	SPECIFICAID	IMPLEMENTID
1	104	103
riga/e 1 - 1 di 1		

#### 4.2.13 Tabella DIPENDENZA

DID	SUPPLIERID	CLIENTID	STEREOTIPO
1	103	500	send
1	109	108	create
riga/e 1 - 2 di 2			

#### 4.2.14 Tabelle NOTA e ANCHOR

EID	DID	VALORE
600	1	Progetti a cui sono assegnati gli impiegati
riga/e 1 - 1 di 1		

DID	HOTAID	TARGETID
1	600	108
riga/e 1 - 1 di 1		

#### 4.2.15 Tabella CONSTRAINT

DID	TARGET1	TARGET2	VALORE
1	201	202	xor
1	401	-	id
1	402	-	eta >= 18
1	409	-	budget > 0
riga/e 1 - 4 di 4			

## Bibliografia

- [ELMA] R. Elmasri, S. Navathe – Sistemi di Basi di Dati, Fondamenti 5a ed. –  
Addison Wesley
- [UMLB] T. Pender – UML Bible – Wiley & Sons

# Progettazione di una base di dati per la memorizzazione della struttura di schemi relazionali

Si definisca una base di dati per la memorizzazione della struttura di schemi relazionali (definizione di tabelle e di vincoli). La base di dati dovrà essere in grado di ospitare per ciascun schema relazionale memorizzato, tutti i costrutti in esso ospitati (definizione di tabelle, dei loro attributi, dei vincoli associati agli attributi e alle tabelle, vincoli di integrità etc...) Si ponga attenzione a definire un opportuno insieme di vincoli che impedisca la composizione di schemi scorretti.

## Analisi preliminare

Uno **Schema Relazionale** può contenere più **Tabelle**. Una **Tabella** può contenere più **Attributi**.

Uno **Schema relazionale**, una **Tabella** o un **Attributo** sono legati a dei **Vincoli**.

Un **Vincolo** si specializza in:

- **Vincolo su Database**, che sono
  - ◆ **Asserzioni** sullo **Schema Relazionale**.
  - ◆ **Trigger** lanciati in un tempo "AFTER"/"BEFORE"/"INSTEAD OF" da un'operazione di INSERT/UPDATE/DELETE su una **Tabella** o una **Vista**.
- **Vincolo sulla tabella**, che a sua può essere diviso in
  - ◆ **vincoli su un gruppo di attributi**, che sono quindi
    - **Primary key**, che riguardano un gruppo di attributi della stessa tabella
    - **Unique** su un gruppo di attributi della stessa tabella.
    - **Check** su un gruppo di attributi della stessa tabella.
  - ◆ **Foreign key**, che riguarda due gruppi compatibili di attributi tra due tabelle e che prevedono un'azione nell'atto di UPDATE/DELETE.
- **Vincolo sul singolo attributo**, ovvero
  - ◆ **Not null**
  - ◆ **Check** (su un singolo attributo della tabella).
  - ◆ **Default**

Dato che sia le **Tabelle** che le **Viste** si usano allo stesso modo possiamo generalizzarle in una classe **Tabella\_Astratta**, specializzandola poi a seconda del tipo.



## Ristrutturazione schema

**Vista** e **Tabella** vengono implementate schiacciandole verticalmente nella classe **Tabella\_Astratta**. Se **Tabella\_Astratta** è una vista, avrà il campo “query” non nullo, altrimenti sarà nullo.

**TcontieneAttr** e **Vista\_Definisce\_Attr** vengono fuse in un'unica associazione rinominata **T\_contiene\_Attributi**.

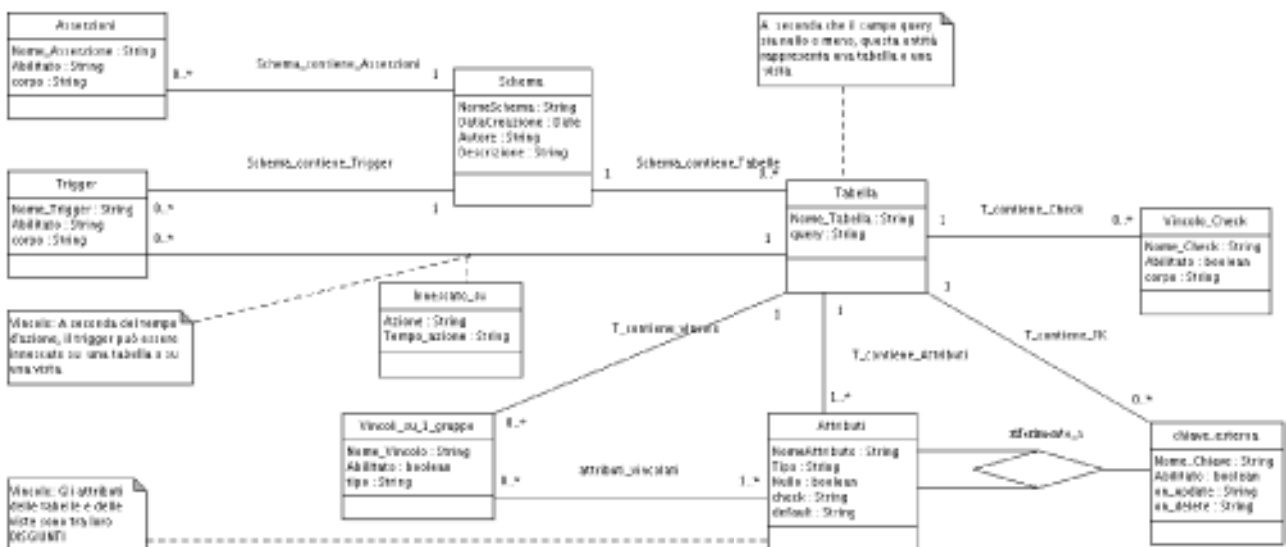
La classe astratta **Vincolo** viene implementata schiacciando orizzontalmente le proprietà e i metodi di questa classe sulle classi specializzate “**Vincolo\_su\_DB**” e “**Vincolo\_su\_Tabelle**”, quindi gli attributi di questa classe li troveremo “sdoppiati” nelle sue classi specializzate.

La classe astratta “**Vincolo\_su\_DB**” viene implementata schiacciando orizzontalmente le proprietà e i metodi di questa classe sulle classi specializzate “**Asserzioni**” e “**Trigger**”, quindi gli attributi di questa classe li troveremo “sdoppiati” nelle sue classi specializzate.

**S\_contiene\_Vincoli\_su\_Tabella** viene sdoppiata sulle classi specializzate **Trigger** e **Asserzioni** (e quindi rinominata in **Schema\_contiene\_Trigger** e **Schema\_contiene\_Asserzioni**).

La classe astratta “**Vincolo\_su\_tabella**” viene implementata schiacciando orizzontalmente le proprietà e i metodi di questa classe sulle classi specializzate “**Check**”, “**chiave\_esterna**” e “**Vincoli\_su\_1\_gruppo**”, quindi gli attributi di questa classe li troveremo “sdoppiati” nelle sue classi specializzate.

**T\_contiene\_V** viene sdoppiata sulle classi specializzate **Check**, **chiave\_esterna** e **vincoli\_su\_1\_gruppo** e quindi rinominate in **Schema\_contiene\_Trigger**, **T\_contiene\_Vincolo**, **T\_contiene\_FK**.



Le tabelle saranno implementate come entità forti, anche se in realtà tutte le entità, tranne Schema, sono deboli. A titolo d'esempio l'identificativo dell'entità chiave\_esterna dovrebbe essere id schema, id tabella, id FK, ma rendendola forte sarà solo id FK, perdendo informazioni che potrebbero essere importanti e, quindi, se necessario, saranno aggiunte.

## DESCRIZIONE MODELLO

**Schema:** Indica uno schema di un database.

**Tabelle:** Indica le tabelle o le viste definite nello schema.

**Asserzioni:** Indica le asserzioni (vincoli) sullo schema del database.

**Trigger:** Indica i trigger (vincoli) innescati da un'azione su una tabella dello schema del database.

**Schema\_contiene\_Asserzioni:** Indica il legame tra lo schema e le asserzioni. Uno schema può contenere zero o più asserzioni, mentre un'asserzione deve essere contenuta in un solo schema.

**Schema\_contiene\_Trigger:** Indica il legame tra lo schema ed i trigger. Uno schema può contenere zero o più trigger, mentre un trigger deve essere contenuto in un solo schema.

**Innescato\_su:** Indica il legame tra i trigger e le tabelle. Un trigger può essere innescato da una certa azione su una tabella o una vista, mentre ad una tabella o una vista può non essere associata ad un trigger.

**Vincolo\_Check:** Indica i vincoli di check su una tabella.

**T\_contiene\_Check:** Indica il legame tra la tabella (non una vista) ed il vincolo di check. Una tabella può contenere zero o più vincoli di check, ma un vincolo di check può essere associato ad una sola tabella.

**Attributi:** Indica l'attributo ed i vincoli specifici dell'attributo. L'insieme degli attributi delle tabelle e quelli delle viste sono disgiunti. Le viste memorizzano gli attributi dichiarati (alias), non gli attributi sui quali effettivamente si effettua l'interrogazione.

**T\_contiene\_Attributi:** è l'associazione che sussiste tra la tabella o la vista ed i suoi attributi. Una tabella o una vista possono contenere/dichiarare più attributi, ma un attributo deve appartenere/essere dichiarato in una sola tabella o vista. Nel caso si tratti:

- di una Tabella, questa è associata agli attributi reali della tabella.
- di una Vista, questa è associata agli attributi dichiarati nella vista (quelli formali), ma non è memorizzato il riferimento agli attributi (reali) delle tabelle sulle quali si basa la vista, e quindi, sui quali è effettuata l'interrogazione dalla vista.

**Vincolo\_su\_gruppo:** Indica i vincoli applicati ad un insieme di attributi.

**T\_contiene\_Vincolo:** Indica i legami tra i vincoli di gruppo e una Tabella (non una Vista). Una Tabella può contenere zero o più vincoli ed un vincolo può interessare una sola tabella.

**Attributi\_vincolati:** indica quali attributi sono vincolati da un determinato vincolo di gruppo. Un vincolo può riguardare uno o più attributi, mentre un attributo può essere vincolato da zero o più vincoli.

**Chiavi\_esterne:** Indica i vincoli di chiave esterna.

**T\_contiene\_FK:** Indica i legami tra i vincoli di chiave esterna e gli attributi di due Tabelle (non una Vista). Una Tabella può contenere zero o più chiavi esterne ed una chiave esterna deve interessare due tabelle.

**Riferimento\_a:** indica a quali coppie di attributi fa riferimento il vincolo di chiave esterna. Possono appartenere allo stesso vincolo di chiave esterna uno o più coppie di attributi ed una coppia di attributi può essere legata a zero o più chiavi esterne.

## Vincoli e precisazioni

1. Ogni entità (Schema, tabella, vincolo, attributi) deve essere identificata univocamente.
2. I riferimenti tra entità devono essere validi.
  - 2.1. Una tabella deve riferirsi obbligatoriamente ad uno schema.
  - 2.2. Un Asserzione o un Trigger devono riferirsi obbligatoriamente ad uno schema.
  - 2.3. Un Trigger deve essere innescato su una Tabella.
  - 2.4. Un attributo deve essere legato ad una tabella o una vista esistente.
  - 2.5. Un vincolo di check o di gruppo o di chiave esterna deve essere associato ad una Tabella esistente, non ad una vista.
  - 2.6. Un vincolo di gruppo deve essere associato ad attributi esistenti.
  - 2.7. Un vincolo di chiave esterna deve essere associato ad un'insieme di coppie di attributi esistenti.
3. Un attributo coinvolto in un vincolo di chiave esterna non può referenziare se stesso, nemmeno per transitività. Ad esempio, dati 3 vincoli di chiave esterna su 3 attributi, ipotizzando che il primo attributo è riferito al secondo, il secondo al terzo ed il terzo al primo, si crea un circolo di riferimenti (un autoriferimento per transitività) che non deve essere permesso.
4. Un attributo coinvolto da un vincolo di chiave esterna deve riferirsi obbligatoriamente ad un altro attributo dello stesso tipo.
5. Non devono esistere legami tra elementi appartenenti a 2 schemi diversi. Solo 2 tabelle possono avere questo problema, e sono "attributi\_vincolati" e "riferimento\_a": Nello specifico:
  - 5.1. I vincoli di gruppo devono coinvolgere attributi della stessa tabella.
  - 5.2. I vincoli di chiave esterna devono coinvolgere esclusivamente 2 tabelle (non necessariamente distinte) dello stesso schema.
6. I trigger devono essere legati ad una vista solo se il tempo d'azione è "INSTEAD OF", altrimenti deve essere associato obbligatoriamente ad una Tabella (tempo d'azione "BEFORE"/"AFTER").
  - 6.1. Non deve essere possibile impostare un valore diverso da Null al campo query di Tabella se la tabella contiene trigger attivati in un tempo d'azione AFTER/BEFORE.
  - 6.2. Non deve essere possibile impostare il valore Null al campo query di Tabella se la tabella contiene trigger attivati in un tempo d'azione "INSTEAD OF".
7. L'insieme degli attributi delle tabelle e quelli delle viste sono disgiunti. Le viste memorizzano gli attributi dichiarati (alias), non gli attributi sui quali effettivamente si effettua l'interrogazione. Il binding è effettuato dall'interprete SQL.
8. Quando cancello un attributo devo cancellare tutti i vincoli sull'attributo. Allo stesso modo, quando cancello un vincolo, devo rimuovere tutti i riferimenti agli attributi (non l'attributo).

9. Quando creo un vincolo, prima ancora di specificare gli attributi vincolati, ho un vincolo vuoto!! Posso gestire la situazione scegliendo di lasciare il vincolo vuoto oppure non devo permettere la creazione del solo vincolo, ma devo crearlo contemporaneamente ai riferimenti sugli attributi.
10. Ogni tabella deve avere una chiave primaria, alla creazione ne aggiungo una di default.
11. Un vincolo è legato ad un attributo un'unica volta. Ad esempio, non può esistere un vincolo di chiave primaria che usa un attributo più di una volta. Es: Pk(Attr1, Attr1) non consentito.

## Implementazione Logica

**Schema** (Id\_Schema, Nome\_Schema, Data\_Creazione, Autore, Descrizione)

Id_Schema	Numero identificativo dello schema (chiave primaria).	Non nullo
Nome_Schema	Nome identificativo dello schema.	Non nullo
Data_Creazione	Data indicante la creazione dello schema.	Non nullo
Autore	Stringa indicante l'autore dello schema.	Nulla
Descrizione	Stringa per descrivere lo schema.	Nulla

**Schema\_contiene\_Tabella:** viene implementato aggiungendo una chiave esterna in “Tabella”.

Tabella\_Astratta, rinominata poi **Tabella:** (Id\_Tabella, Id\_Schema, Nome\_Tabella, query).

Id_Tabella	Numero identificativo della Tabella_Astratta (chiave primaria).	Non nullo.
Id_Schema	Numero identificativo dello schema (chiave esterna).	Non nullo.
Nome_Tabella	Nome identificativo della Tabella_Astratta .	Non nullo.
query	Stringa identificante la query (solo per le viste).	Nulla.

Sia **Schema\_contiene\_Trigger** che **Schema\_contiene\_Asserzioni** vengono implementate aggiungendo una chiave esterna in Trigger e Asserzioni.

**Asserzioni:** (Id\_Asserzione, Id\_Schema, Nome\_Asserzione, Abilitato, corpo).

Id_Asserzione	Numero identificativo dell'asserzione (chiave primaria).	Non nullo.
Id_Schema	Num identificativo dello schema a cui appartiene (chiave esterna).	Non nullo.
Nome_Asserzione	Nome identificativo dell'asserzione.	Non nullo.
Abilitato	Valore booleano indicante se il vincolo è abilitato o meno.	Non nullo.
corpo	Stringa identificante il corpo dell'asserzione.	Non nullo.

**Innescato\_su:** viene implementato aggiungendo una chiave esterna in “Trigger”.

**Trigger:** (Id\_Trigger, Id\_Schema, Id\_Tabella, Nome\_Trigger, Abilitato, Azione, TempoAzione, corpo).

Id_Trigger	Numero identificativo del trigger (chiave primaria).	Non nullo.
Id_Schema	Numero identificativo dello schema a cui appartiene l'asserzione (chiave esterna).	Non nullo.
Id_Tabella	Numero identificativo della tabella sulla quale agisce il trigger (chiave esterna).	Non nullo.
Nome_Trigger	Nome identificativo del trigger.	Non nullo.
Abilitato	Valore booleano indicante se il vincolo è abilitato o meno.	Non nullo.
Azione	Stringa indicante l'azione che innesci il trigger.	Non nullo.
TempoAzione	Stringa indicante se quando innescare il trigger	Non nullo.
corpo	Stringa identificante il corpo del trigger.	Non nullo.

**T\_contiene\_Attributi** viene implementata aggiungendo una chiave esterna in Attributi.

**Attributi:** (Id\_Attributo, Id\_Tabella, Nome\_Attributo, Tipo, Nullo, Check, Default).

Id_Attributo	Numero identificativo dell'Attributo (chiave primaria).	Non nullo.
Id_Tabella	Numero identificativo della Tabella alla quale appartiene l'attributo (chiave esterna).	Non nullo.
Nome_Attributo	Nome dell'Attributo.	Non nullo.
Tipo	Tipo dell'attributo.	Non nullo.
Nullo	Valore booleano indicante se l'attributo può essere nullo.	Non nullo.
Check	Stringa indicante le condizioni che l'attributo deve rispettare.	Nulla.
Default	Stringa indicante il valore predefinito dell'attributo.	Nulla.

**T\_contiene\_check:** viene implementato aggiungendo una chiave esterna in Vincolo\_check.

**Vincolo\_check:** (Id\_Vcheck, Id\_Tabella, Nome\_Check, Abilitato).

Id_Vcheck	Numero identificativo del vincolo (chiave primaria).	Non nullo.
Id_Tabella	Numero identificativo della Tabella che memorizza il vincolo (chiave esterna).	Non nullo.
Nome_Check	Nome identificativo della vincolo.	Non nullo.
Abilitato	Valore booleano indicante se il vincolo è abilitato o meno.	Non nullo.

**T\_contiene\_Vincolo:** viene implementato aggiungendo una chiave esterna in Vincoli\_su\_1\_gruppo.

**Vincolo\_su\_gruppo:** (Id\_Vgruppo, Id\_Tabella, Nome\_Vincolo, tipo, Abilitato).

Id_Vgruppo	Numero identificativo del vincolo sul gruppo (chiave primaria).	Non nullo.
Id_Tabella	Numero identificativo della Tabella che memorizza il vincolo sul gruppo (chiave esterna).	Non nullo.
Nome_Vincolo	Nome identificativo del vincolo sul gruppo.	Non nullo.
Abilitato	Valore booleano indicante se il vincolo è abilitato o meno.	Non nullo.
tipo	Stringa identificante il tipo di vincolo.	Non nullo.

**T\_contiene\_FK:** viene implementato aggiungendo una chiave esterna in chiave\_esterna.

**chiave\_esterna:** (Id\_FK, Id\_Tabella, Id\_Tabella2, Nome\_Chiaive, Abilitato, on\_update, on\_delete).

Id_FK	Numero identificativo del vincolo (chiave primaria).	Non nullo.
Id_Tabella	Numero identificativo della Tabella che memorizza il vincolo.	Non nullo.
Id_Tabella2	Numero identificativo della Tabella alla quale si riferisce la chiave esterna. E' da spostare nella tabella "riferimento_a".	Non nullo.
Nome_Chiaive	Nome identificativo della vincolo.	Non nullo.
Abilitato	Valore booleano indicante se il vincolo è abilitato o meno.	Non nullo.
on_update	Stringa identificante l'azione da effettuare in caso di update.	Non nullo.
on_delete	Stringa identificante l'azione da effettuare in caso di delete.	Non nullo.

**Attributi\_vincolati:** (Id\_Vincolo, Id\_Attributo).

Id_Vgruppo	Numero identificativo del vincolo sull'attributo.	Non nullo.
Id_Attributo	Numero identificativo dell'Attributo vincolato (chiave esterna).	Non nullo.

**Riferimento\_a:** (Id\_Vincolo, Id\_Attributo, Id\_AttributoRef).

Id_FK	Numero identificativo del vincolo sull'attributo.	Non nullo.
Id_Attributo	Numero identificativo dell'Attributo vincolato (chiave esterna).	Non nullo.
Id_AttributoRef	Numero identificativo dell'Attributo referenziato da un attributo parte di una chiave esterna (chiave esterna).	Non nullo.

## Implementazione Logica in SQL

```

CREATE TABLE "SCHEMA" (
    ID_SCHEMA          INTEGER          NOT NULL ENABLE,
    NOME_SCHEMA        VARCHAR2(20)     NOT NULL ENABLE,
    DATACREAZIONE     DATE             NOT NULL ENABLE,
    AUTORE              VARCHAR2(20),
    DESCRIZIONE        VARCHAR2(100),

    CONSTRAINT "PK_SCHEMA" PRIMARY KEY ("ID_SCHEMA") ENABLE
)

CREATE TABLE "TABELLA" (
    ID_TABELLA         INTEGER          NOT NULL ENABLE,
    ID_SCHEMA          INTEGER          NOT NULL ENABLE,
    NOME_TABELLA       VARCHAR(20)     NOT NULL ENABLE,
    QUERY              TEXT,

    CONSTRAINT "PK_TABELLA" PRIMARY KEY ("ID_TABELLA") ENABLE
    CONSTRAINT "FK_TABLE" FOREIGN KEY ("ID_SCHEMA") REFERENCES
        SCHEMA("ID_SCHEMA") ENABLE ON UPDATE CASCADE
        ON DELETE CASCADE
)
    
```

```

CREATE TABLE "ASSERZIONI" (
  ID_ASSERZIONE      INTEGER      NOT NULL ENABLE,
  ID_SCHEMA          INTEGER      NOT NULL ENABLE,
  NOME_ASSERZIONE    VARCHAR(20)  NOT NULL ENABLE,
  ABILITATO          CHAR(2)      NOT NULL ENABLE,
  CORPO              TEXT         NOT NULL ENABLE,

  CONSTRAINT "PK_ASSERZIONI" PRIMARY KEY ("ID_ASSERZIONI")
                                     ENABLE
  CONSTRAINT "FK_ASSERZIONI"  FOREIGN KEY ("ID_SCHEMA")
  REFERENCES SCHEMA("ID_SCHEMA") ENABLE ON UPDATE CASCADE
                                     ON DELETE CASCADE
)

```

```

CREATE TABLE "TRIGGER" (
  ID_TRIGGER      INTEGER      NOT NULL ENABLE,
  ID_SCHEMA       INTEGER      NOT NULL ENABLE,
  ID_TABELLA     INTEGER      NOT NULL ENABLE,
  NOME_TRIGGER    VARCHAR(20)  NOT NULL ENABLE,
  ABILITATO      CHAR(2)      NOT NULL ENABLE,
  AZIONE         VARCHAR(40)   NOT NULL ENABLE,
  TEMPO_AZIONE   VARCHAR(40)   NOT NULL ENABLE,
  CORPO          TEXT         NOT NULL ENABLE,

  CONSTRAINT "PK_TRIGGER" PRIMARY KEY ("ID_TRIGGER") ENABLE
  CONSTRAINT "FK_TRIGGER" FOREIGN KEY ("ID_SCHEMA") REFERENCES
  SCHEMA("ID_SCHEMA") ENABLE ON UPDATE CASCADE
                                     ON DELETE CASCADE
  CONSTRAINT "FK_TRIGGER2" FOREIGN KEY ("ID_TABELLA")
  REFERENCES TABELLA("ID_TABELLA") ENABLE ON UPDATE CASCADE
                                     ON DELETE CASCADE
)

```

```

CREATE TABLE "ATTRIBUTI" (
  ID_ATTRIBUTO      INTEGER      NOT NULL ENABLE,
  ID_TABELLA       INTEGER      NOT NULL ENABLE,
  NOME_ATTRIBUTO    VARCHAR(20)  NOT NULL ENABLE,
  TIPO             VARCHAR(20)  NOT NULL ENABLE,
  NULLO            CHAR(2)      NOT NULL ENABLE,
  CHECK            TEXT,
  DEFAULT          VARCHAR(20),

  CONSTRAINT "PK_ATTRIBUTO" PRIMARY KEY ("ID_ATTRIBUTI")
                                     ENABLE,
  CONSTRAINT "FK_ATTRIBUTI"  FOREIGN KEY ("ID_TABELLA")
  REFERENCES TABELLA("ID_TABELLA") ENABLE ON UPDATE CASCADE
                                     ON DELETE CASCADE
)

```

```

CREATE TABLE "VINCOLO_SU_GRUPPO" (
  ID_VGRUPPO          INTEGER          NOT NULL ENABLE,
  ID_TABELLA          INTEGER          NOT NULL ENABLE,
  NOME_VINCOLO        VARCHAR(20)     NOT NULL ENABLE,
  ABILITATO           CHAR(2)          NOT NULL ENABLE,
  TIPO                 VARCHAR(20)     NOT NULL ENABLE,

  CONSTRAINT "PK_VINCOLO_GRUPPO" PRIMARY KEY
                        ("ID_VGRUPPO") ENABLE
  CONSTRAINT "FK_GRUPPO"   FOREIGN KEY ("ID_TABELLA")
                        REFERENCES TABELLA("ID_TABELLA") ENABLE ON UPDATE CASCADE
                                                ON DELETE CASCADE
)

```

```

CREATE TABLE VINCOLO_CHECK (
  ID_VCHECK           INTEGER          NOT NULL ENABLE,
  ID_TABELLA          INTEGER          NOT NULL ENABLE,
  NOME_CHECK          VARCHAR(20)     NOT NULL ENABLE,
  ABILITATO           CHAR(2)          NOT NULL ENABLE,

  CONSTRAINT "PK_VINCOLO_CHECK" PRIMARY KEY
                        ("ID_VCHECK") ENABLE
  CONSTRAINT "FK_CHECK"   FOREIGN KEY ("ID_TABELLA")
                        REFERENCES TABELLA("ID_TABELLA") ENABLE ON UPDATE CASCADE
                                                ON DELETE CASCADE
)

```

```

CREATE TABLE "CHIAVE_ESTERNA" (
  ID_FK               INTEGER          NOT NULL ENABLE,
  ID_TABELLA          INTEGER          NOT NULL ENABLE,
  ID_TABELLA2         INTEGER          NOT NULL ENABLE,
  NOME_FK             VARCHAR(20)     NOT NULL ENABLE,
  ABILITATO           CHAR(2)          NOT NULL ENABLE,
  ON_UPDATE           VARCHAR(10)     NOT NULL ENABLE,
  ON_DELETE           VARCHAR(10)     NOT NULL ENABLE,

  CONSTRAINT "PK_FK"   PRIMARY KEY ("ID_FK") ENABLE,
  CONSTRAINT "FK_FK"   FOREIGN KEY ("ID_TABELLA")
                        REFERENCES TABELLA("ID_TABELLA") ENABLE ON UPDATE CASCADE
                                                ON DELETE CASCADE,
  CONSTRAINT "FK_FK"   FOREIGN KEY ("ID_TABELLA")
                        REFERENCES TABELLA("ID_TABELLA2") ENABLE ON UPDATE CASCADE
                                                ON DELETE CASCADE
)

```

```

CREATE TABLE "ATTRIBUTI_VINCOLATI" (
    ID_VGRUPPO          INTEGER          NOT NULL ENABLE,
    ID_ATTRIBUTO        INTEGER          NOT NULL ENABLE,

    CONSTRAINT "PK_ATTR_VINCOLATI" PRIMARY KEY
        ("ID_VGRUPPO, ID_ATTRIBUTO") ENABLE,
    CONSTRAINT "FK_ATTR_VINCOLATI1" FOREIGN KEY ("ID_ATTRIBUTO")
        REFERENCES VINCOLO_SU_GRUPPO("ID_VGRUPPO") ENABLE,
    CONSTRAINT "FK_ATTR_VINCOLATI2" FOREIGN KEY ("ID_ATTRIBUTO")
        REFERENCES ATTRIBUTI("ID_ATTRIBUTO") ENABLE
)

```

```

CREATE TABLE "RIFERIMENTO_A" (
    ID_FK                INTEGER          NOT NULL ENABLE,
    ID_ATTRIBUTO         INTEGER          NOT NULL ENABLE,
    ID_ATTRIBUTO_REF     INTEGER          NOT NULL ENABLE,

    CONSTRAINT "PK_ATTR_VINCOLATI" PRIMARY KEY
        ("ID_FK, ID_ATTRIBUTO, ID_ATTR_REF") ENABLE
    CONSTRAINT "VINCOLO_FK" FOREIGN KEY ("ID_FK")
        REFERENCES CHIAVE_ESTERNA("ID_ATTRIBUTO") ENABLE
    CONSTRAINT "VINCOLO_FK" FOREIGN KEY ("ID_FK")
        REFERENCES CHIAVE_ESTERNA("ID_ATTRIBUTO_REF") ENABLE,
    CONSTRAINT "NO_AUTORIFERIMENTO" CHECK ID_ATTRIBUTO <>
        ID_ATTRIBUTO_REF ENABLE
)

```

//seleziona i 2 attributi referenziante e referenziato coinvolti in una chiave esterna che hanno tipo diverso, violando il vincolo.

```

CREATE ASSERTION VINCOLO_FK_TRA_2_ATTRIBUTI_STESSO_TIPO
CHECK NOT EXISTS
SELECT *
FROM     CHIAVE_ESTERNA FK
        JOIN RIFERIMENTO_A RIF
            ON RIF.ID_FK = FK.ID_FK
        JOIN ATTRIBUTI ATTR
            ON RIF.ID_ATTRIBUTO = ATTR.ID_ATTRIBUTO
        JOIN ATTRIBUTI ATTR2
            ON RIF.ID_ATTRIBUTO_RIF = ATTR2.ID_ATTRIBUTO
WHERE ATTR.TIPO <> ATTR2.TIPO

```

//dato un vincolo di gruppo contenuto in una certa tabella, seleziona gli attributi coinvolti dal vincolo che non appartengono alla stessa tabella memorizzante il vincolo.

```
CREATE ASSERTION VINCOLI_SU_ATTRIBUTI_DELLA_TABELLA
CHECK NOT EXISTS
SELECT *
FROM      VINCOLI_SU_GRUPPO VG
JOIN      ATTRIBUTI_VINCOLATI ATT_VINC
ON        VG.ID_VGRUPPO = ATT_VINC.ID_VINCOLO
JOIN      ATTRIBUTI ATTR
ON        ATT_VINC.ID_ATTRIBUTO = ATTR.ID_ATTRIBUTO
WHERE     VG.ID_TABELLA <> ATTR.ID_TABELLA
```

//dato un vincolo di chiave esterna contenuto in una certa tabella, seleziona gli attributi coinvolti dal vincolo che non appartengono alla stessa tabella memorizzante il vincolo.

```
CREATE ASSERTION FK_SOLO_DELLA_1_TABELLA
CHECK NOT EXISTS
SELECT *
FROM      CHIAVE_ESTERNA FK
JOIN      RIFERIMENTO_A RIF
ON        RIF.ID_FK = FK.ID_FK
JOIN      ATTRIBUTI ATTR
ON        RIF.ID_ATTRIBUTO = ATTR.ID_ATTRIBUTO
WHERE     ATTR.ID_ATTRIBUTO <> FK.ID_TABELLA
```

//dato un vincolo di chiave esterna che riferenzia una certa tabella, seleziona gli attributi coinvolti dal vincolo che non appartengono alla stessa tabella referenziata dal vincolo.

```
CREATE ASSERTION FK_RIFERITE_SOLO_ALLA_2_TABELLA
CHECK NOT EXISTS
SELECT *
FROM      CHIAVE_ESTERNA FK
JOIN      RIFERIMENTO_A RIF
ON        RIF.ID_FK = FK.ID_FK
JOIN      ATTRIBUTI ATTR
ON        RIF.ID_ATTRIBUTO = ATTR.ID_ATTRIBUTO
WHERE     FK.ID_TABELLA2 <> ATTR.ID_TABELLA2
```

//dato un vincolo di chiave esterna memorizzato in una prima tabella e che referenzia una seconda tabella, seleziona lo schema di appartenenza della prima e della seconda tabella solo se sono diversi.

```
CREATE ASSERTION FK_TRA_TABELLE_STESSO_SCHEMA CHECK NOT EXISTS
SELECT *
FROM      CHIAVE_ESTERNA FK
JOIN      TABELLA T1
ON        FK.ID_TABELLA1 = T1.ID_TABELLA
JOIN      TABELLA T2
ON        FK.ID_TABELLA2 = T2.ID_TABELLA
WHERE     T1.ID_SCHEMA <> T2.ID_SCHEMA
```

//dato un vincolo di check memorizzato in una tabella, seleziona il vincolo se la tabella è una vista.

```
CREATE ASSERTION VINCOLO_CHECK_NON_SU_VISTA CHECK NOT EXISTS
  SELECT *
  FROM VINCOLO_CHECK VC JOIN TABELLA T
    ON VC.ID_TABELLA = T.ID_TABELLA
  WHERE T.QUERY IS NULL.
```

//dato un vincolo di chiave esterna memorizzato in una tabella, seleziona il vincolo solo se la tabella è una vista.

```
CREATE ASSERTION VINCOLO_FK_NON_SU_VISTA CHECK NOT EXISTS
  SELECT *
  FROM CHIAVE_ESTERNA FC JOIN TABELLA T
    ON FC.ID_TABELLA = T.ID_TABELLA
  WHERE T.QUERY IS NULL.
```

//dato un vincolo di gruppo memorizzato in una tabella, seleziona il vincolo se la tabella è una vista

```
CREATE ASSERTION VINCOLO_SU_GRUPPO_NON_SU_VISTA CHECK NOT EXISTS
  SELECT *
  FROM VINCOLO_SU_GRUPPO VG JOIN TABELLA T
    ON VG.ID_TABELLA = T.ID_TABELLA
  WHERE T.QUERY IS NULL.
```

//seleziona il trigger solo se deve essere legato ad una vista (tempo d'azione "instead of") ed invece è legato ad una tabella (query nulla).

```
CREATE ASSERTION TEMPO_OBBLIGATO_TRIGGER_SU_VISTA CHECK NOT EXISTS
  SELECT *
  FROM TRIGGER TR JOIN TABELLA T
    ON TR.ID_TABELLA = T.ID_TABELLA
  WHERE TR.TEMPO_AZIONE = "INSTEAD OF" AND T.QUERY IS NULL
```

//seleziona il trigger solo se deve essere legato ad una tabella (tempo d'azione diverso da "instead of") ed invece è legato ad una vista (query non nulla).

```
CREATE ASSERTION TEMPO_OBBLIGATO_TRIGGER_SU_TABELLA CHECK NOT
EXISTS  SELECT *
  FROM TRIGGER TR JOIN TABELLA T
    ON TR.ID_TABELLA = T.ID_TABELLA
  WHERE TR.TEMPO_AZIONE <> "INSTEAD OF" AND T.QUERY IS NULL
```

//Quando cancello un attributo, devo PRIMA cancellare tutte le chiavi esterne che si riferiscono all'attributo. Seleziono ed elimino le chiavi esterne che si riferiscono all'attributo da eliminare

```
CREATE TRIGGER ELIMINA_ATTRIBUTO_E_FK_CORRETTAMENTE
    BEFORE DELETE ON ATTRIBUTO
    FOR EACH ROW
BEGIN
    DELETE * FROM CHIAVE_ESTERNA FK
    WHERE FK.ID_FK IN
        SELECT FK.ID_FK
        FROM      CHIAVE_ESTERNA FK
        JOIN RIFERIMENTO_A RIF
            ON RIF.ID_FK = FK.ID_FK
        JOIN ATTRIBUTI ATTR
            ON RIF.ID_ATTRIBUTO = ATTR.ID_ATTRIBUTO
        JOIN ATTRIBUTI ATTR2
            ON RIF.ID_ATTRIBUTO_RIF = ATTR2.ID_ATTRIBUTO
    WHERE ATTR.ID_ATTRIBUTO = OLD.ID_ATTRIBUTO
        OR RIF.ID_ATTRIBUTO_RIF = OLD.ID_ATTRIBUTO
END
```

//Quando cancello un attributo, devo PRIMA cancellare tutte i vincoli di gruppo che si riferiscono all'attributo. Seleziono ed elimino i vincoli di gruppo che si riferiscono all'attributo da eliminare

```
CREATE TRIGGER ELIMINA_ATTRIBUTO_E_VINCOLO_GRUPPO_CORRETTAMENTE
    BEFORE DELETE ON ATTRIBUTO
    FOR EACH ROW
BEGIN
    DELETE * FROM VINCOLO_SU_GRUPPO VG
    WHERE VD.ID_VGRUPPO IN
        SELECT VG.ID_VGRUPPO
        FROM      VINCOLO_SU_GRUPPO VG
        JOIN ATTRIBUTI_VINCOLATI ATTR_V
            ON VG.ID_VGRUPPO = ATTRV.ID_ID_VGRUPPO
        JOIN ATTRIBUTI ATTR
            ON ATTR_V.ID_ATTRIBUTO = ATTR.ID_ATTRIBUTO
    WHERE ATTR.ID_ATTRIBUTO = OLD.ID_ATTRIBUTO
END
```

//Quando cancello una chiave esterna, devo PRIMA rimuovere tutti i riferimenti agli attributi (non l'attributo).

```
CREATE TRIGGER ELIMINA_FK_CORRETTAMENTE
    BEFORE DELETE ON CHIAVE_ESTERNA
BEGIN
    DELETE * FROM RIFERIMENTO_A RIF
    WHERE RIF.ID_FK = OLD.ID_FK
END
```

//Quando cancello un vincolo di gruppo, devo PRIMA rimuovere tutti i riferimenti agli attributi.

```
CREATE TRIGGER ELIMINA_VINCOLO_GRUPPO_CORRETTAMENTE
    BEFORE DELETE ON VINCOLO_SU_GRUPPO
    BEGIN
        DELETE * FROM RIFERIMENTO_A_RIF
        WHERE RIF.ID_VGRUPPO = OLD.ID_VGRUPPO
    END
```

//Quando creo una tabella DOPO devo inserire automaticamente una chiave primaria. Il tutto dovrebbe avvenire in una transazione, in modo da implementare un vincolo contro i vincoli vuoti.

```
CREATE TRIGGER "CREA_PK_PER_NUOVA_TABELLA"
    AFTER INSERT ON TABELLA
    DECLARE
        ID_NEW_ATTR INTEGER;
    BEGIN
        ID_NEW_ATTR = ATTR.NEXTVALUE;

        INSERT INTO ATTRIBUTI_ATTR
        VALUE (ID_NEW_ATTR, NEW.ID_TABELLA, "PK_" +
            NEW.NOME_TABELLA, INTEGER, 'NO', NULL, NULL);

        CALL INSERISCI_ATTRIBUTO_A_VG (VINCOLO.NEXT_VALUE,
            NEW.ID_TABELLA, "ID_" + NEW.NOME_TABELLA, 'SI', "PRIMARY KEY");
    END
```

//Quando creo un vincolo, prima ancora di specificare gli attributi vincolati, ho un vincolo vuoto!!  
Non devo permettere la creazione del solo vincolo, ma devo crearlo contemporaneamente ai riferimenti sugli attributi.

```
CREATE PROCEDURE INSERISCI_ATTRIBUTO_A_VG (ID_VG IN INTEGER,  
ID_TAB IN INTEGER, NOME VARCHAR(20), TIPO VARCHAR(20), ID_ATTR IN  
INTEGER)  
  DECLARE  
    ESISTE_VINCOLO INTEGER;  
  BEGIN  
  
    //Controllo se il vincolo esiste già  
  
    SELECT ID_VG INTO ESISTE_VINCOLO  
    FROM VINCOLO_SU_GRUPPO VG  
    WHERE VG.ID_VGRUPPO = :ID_VG;  
  
    //Se esiste già, allora  
    IF (ESISTE_VINCOLO IS NOT NULL) THEN  
      BEGIN  
        //devo inserire solo i riferimenti agli attributi vincolati.  
        INSERT INTO ATTRIBUTI_VINCOLATI ATTR_V  
        VALUE (:ID_VG, :ID_ATTR)  
      END  
    ELSE  
      //in una transazione nel caso ci fosse un vincolo che impedisce vincoli vuoti  
      BEGIN TRANSACTION  
        //devo inserire prima il vincolo  
        INSERT INTO VINCOLI_SU_GRUPPO  
        VALUE (:ID_VG, :ID_TAB, :NOME, :TIPO, "ENABLE");  
  
        //Poi i riferimenti agli attributi vincolati.  
        INSERT INTO ATTRIBUTI_VINCOLATI ATTR_V  
        VALUE (:ID_VG, :ID_ATTR)  
  
      END TRANSACTION  
    END IF;  
  END
```

//Quando elimino un riferimento agli attributi vincolati da un vincolo, c'è la possibilità di ritrovarci con un vincolo vuoto!! Non devo permettere l'eliminazione di un riferimento ad un attributo vincolato, ma contemporaneamente all'eliminazione dei riferimenti sugli attributi, devo eliminare anche il vincolo.

```

CREATE PROCEDURE ELIMINA_ATTRIBUTO_DA_VG (ID_VG IN INTEGER,
                                           ID_ATTR IN INTEGER)

  DECLARE
    VINCOLO_2_ATTR INTEGER;
  BEGIN
    VINCOLO_2_ATTR = 0;

    //Controllo se il vincolo ha solo un attributo, selezionando 2 attributi
    //distinti legati al vincolo id_VG
    SELECT A_V.ID_VGRUPPO INTO VINCOLO_2_ATTR
    FROM     ATTRIBUTI_VINCOLATI A_V
           JOIN ATTRIBUTI_VINCOLATI A_V2
             ON A_V.ID_VGRUPPO = A_V2.ID_VGRUPPO
    WHERE A_V.ID_ATTRIBUTO <> A_V2.ID_ATTRIBUTO
          AND A_V.ID_VGRUPPO = :ID_VG;

    //Se il vincolo ha più di un attributo, allora
    IF (VINCOLO_2_ATTR IS NOT NULL) THEN
      BEGIN
        //devo eliminare solo il riferimento all'attributo vincolato.
        DELETE FROM ATTRIBUTI_VINCOLATI ATTR_V
        WHERE ATTR_V.ID_VGRUPPO = :ID_VG
              AND ATTR_V.ID_ATTRIBUTO = :ID_ATTR;
      END
    ELSE
      //in una transazione nel caso ci fosse un vincolo che impedisce vincoli vuoti
      BEGIN TRANSACTION
        //devo rimuovere il legame
        DELETE FROM ATTRIBUTI_VINCOLATI ATTR_V
        WHERE ATTR_V.ID_VGRUPPO = :ID_VG
              AND ATTR_V.ID_ATTRIBUTO = :ID_ATTR;

        // Poi rimuovere il vincolo
        DELETE FROM VINCOLO_SU_GRUPPO VG
        WHERE VG = :ID_VG;

      END TRANSACTION
    END IF;
  END

```

## Risoluzione vincoli

1. Ogni entità (Schema, tabella, vincolo, attributi) deve essere identificata univocamente.  
CONSTRAINT "PK\_SCHEMA" PRIMARY KEY ("ID\_SCHEMA") ENABLE  
CONSTRAINT "PK\_TABELLA" PRIMARY KEY ("ID\_TABELLA") ENABLE  
CONSTRAINT "PK\_TRIGGER" PRIMARY KEY ("ID\_TRIGGER") ENABLE  
CONSTRAINT "PK\_ASSERZIONI" PRIMARY KEY ("ID\_ASSERZIONI")  
ENABLE  
CONSTRAINT "PK\_ATTRIBUTO" PRIMARY KEY ("ID\_ATTRIBUTI")  
ENABLE  
CONSTRAINT "PK\_FK" PRIMARY KEY ("ID\_FK") ENABLE  
CONSTRAINT "PK\_VINCOLO\_GRUPPO" PRIMARY KEY  
("ID\_VGRUPPO") ENABLE  
CONSTRAINT "PK\_VINCOLO\_CHECK" PRIMARY KEY  
("ID\_VCHECK") ENABLE  
CONSTRAINT "PK\_ATTR\_VINCOLATI" PRIMARY KEY  
("ID\_FK, ID\_ATTRIBUTO") ENABLE
2. I riferimenti tra entità devono essere validi.
  - 2.1. Una tabella deve riferirsi obbligatoriamente ad uno schema.  
CONSTRAINT "FK\_TABLE" FOREIGN KEY ("ID\_SCHEMA")  
REFERENCES SCHEMA("ID\_SCHEMA") ENABLE
  - 2.2. Un Asserzione o un Trigger devono riferirsi obbligatoriamente ad uno schema.  
CONSTRAINT "FK\_TRIGGER" FOREIGN KEY ("ID\_SCHEMA")  
REFERENCES SCHEMA("ID\_SCHEMA") ENABLE  
CONSTRAINT "FK\_ASSERZIONI" FOREIGN KEY ("ID\_SCHEMA")  
REFERENCES SCHEMA("ID\_SCHEMA") ENABLE
  - 2.3. Un Trigger deve essere innescato su una Tabella.  
CONSTRAINT "FK\_TRIGGER2" FOREIGN KEY ("ID\_TABELLA")  
REFERENCES SCHEMA("ID\_SCHEMA") ENABLE
  - 2.4. Un attributo deve essere legato ad una tabella o una vista esistente.  
CONSTRAINT "FK\_ATTRIBUTI" FOREIGN KEY ("ID\_TABELLA")  
REFERENCES TABELLA("ID\_TABELLA") ENABLE
  - 2.5. Un vincolo di check o di gruppo o di chiave esterna deve essere associato ad una Tabella esistente, non ad una vista.  
CONSTRAINT "FK\_GRUPPO" FOREIGN KEY ("ID\_TABELLA")  
REFERENCES TABELLA("ID\_TABELLA") ENABLE  
CONSTRAINT "FK\_CHECK" FOREIGN KEY ("ID\_TABELLA")  
REFERENCES TABELLA("ID\_TABELLA") ENABLE  
CONSTRAINT "FK\_FK" FOREIGN KEY ("ID\_TABELLA")  
REFERENCES TABELLA("ID\_TABELLA") ENABLE  
La garanzia che i vincoli non siano associati ad una vista è garantito dalle asserzioni  
VINCOLO\_SU\_GRUPPO\_NON\_SU\_VISTA, VINCOLO\_FK\_NON\_SU\_VISTA e  
VINCOLO\_CHECK\_NON\_SU\_VISTA.
  - 2.6. Un vincolo di gruppo deve essere associato ad attributi esistenti.  
RISOLTO TRAMITE LA TABELLA ATTRIBUTI\_VINCOLATI.
  - 2.7. Un vincolo di chiave esterna deve essere associato ad un'insieme di coppie di attributi  
esistenti. RISOLTO TRAMITE LA TABELLA RIFERIMENTO\_A.

3. Un attributo coinvolto in un vincolo di chiave esterna non può referenziare se stesso, nemmeno per transitività. Ad esempio, dati 3 vincoli di chiave esterna su 3 attributi, ipotizzando che il primo attributo è riferito al secondo, il secondo al terzo ed il terzo al primo, si crea un circolo di riferimenti (un autoriferimento per transitività) che non deve essere permesso. Il caso transitivo deve essere risolto tramite l'aggiunta di una tabella temporanea, quindi non verrà considerato, mentre il primo caso (semplice) è risolto dal vincolo  
**CONSTRAINT "NO\_AUTORIFERIMENTO" CHECK ID\_ATTRIBUTO <>  
ID\_ATTRIBUTO\_REF ENABLE**
4. Un attributo coinvolto da un vincolo di chiave esterna deve riferirsi obbligatoriamente ad un altro attributo dello stesso tipo. **RISOLTO DALL'ASERZIONE VINCOLO\_FK\_TRA\_2\_ATTRIBUTI\_STESSO\_TIPO.**
5. Non devono esistere legami tra elementi appartenenti a 2 schemi diversi. Solo 2 tabelle possono avere questo problema, e sono "attributi\_vincolati" e "riferimento\_a": Nello specifico:
  - 5.1. I vincoli di gruppo devono coinvolgere attributi della stessa tabella. **RISOLTO DALL'ASERZIONE VINCOLI\_SU\_ATTRIBUTI\_DELLA\_TABELLA.**
  - 5.2. I vincoli di chiave esterna devono coinvolgere 2 gruppi di attributi appartenenti esclusivamente a 2 tabelle (non necessariamente distinte) dello stesso schema: **RISOLTI DALLE ASERZIONI FK\_SOLO\_DELLA\_1\_TABELLA, FK\_RIFERITE\_SOLO\_ALLA\_2\_TABELLA ED FK\_TRA\_TABELLE\_STESSO\_SCHEMA.**
6. I trigger devono essere legati ad una vista solo se il tempo d'azione è "INSTEAD OF", altrimenti deve essere associato obbligatoriamente ad una Tabella (tempo d'azione "BEFORE"/"AFTER").
  - 6.1. Non deve essere possibile impostare un valore diverso da Null al campo query di Tabella se la tabella contiene trigger attivati in un tempo d'azione AFTER/BEFORE. **RISOLTO DALL'ASERZIONE TEMPO\_OBBLIGATO\_TRIGGER\_SU\_VISTA**
  - 6.2. Non deve essere possibile impostare il valore Null al campo query di Tabella se la tabella contiene trigger attivati in un tempo d'azione "INSTEAD OF". **RISOLTO DALL'ASERZIONE TEMPO\_OBBLIGATO\_TRIGGER\_SU\_TABELLA**
7. L'insieme degli attributi delle tabelle e quelli delle viste sono disgiunti. Le viste memorizzano gli attributi dichiarati (alias), non gli attributi sui quali effettivamente si effettua l'interrogazione. Il binding è effettuato dall'interprete SQL.
8. Quando cancello un attributo devo cancellare tutti i vincoli sull'attributo. Allo stesso modo, quando cancello un vincolo, devo rimuovere tutti i riferimenti agli attributi (non l'attributo). Risolto dal vincolo **ELIMINA\_ATTRIBUTO\_E\_FK\_CORRETTAMENTE, ELIMINA\_ATTRIBUTO\_E\_VINCOLO\_GRUPPO\_CORRETTAMENTE, ELIMINA\_VINCOLO\_GRUPPO\_CORRETTAMENTE ED ELIMINA\_FK\_CORRETTAMENTE.**

9. Quando creo un vincolo, prima ancora di specificare gli attributi vincolati, ho un vincolo vuoto!! Posso gestire la situazione scegliendo di lasciare il vincolo vuoto oppure non devo permettere la creazione del solo vincolo, ma devo crearlo contemporaneamente ai riferimenti sugli attributi. Risolto dalle procedure `INSERISCI_ATTRIBUTO_A_VG` ED `ELIMINA_ATTRIBUTO_DA_VG`; Per chiave\_esterna le procedure sono molto simili e non saranno implementate.
10. Ogni tabella deve avere una chiave primaria, alla creazione ne aggiungo una di default. Risolto dal vincolo `CREA_PK_PER_NUOVA_TABELLA`.
11. Un vincolo è legato ad un attributo un'unica volta. Ad esempio, non può esistere un vincolo di chiave primaria che usa un attributo più di una volta. Es: `Pk(Attr1, Attr1)` non consentito. E' **GARANTITO DALLA CHIAVE PRIMARIA DELLE TABELLE**.

**Daniele Finelli**  
**Matr. 566/1094**

# Documentazione progetto

Gerardo Adelizzi, 566/2471  
Angelo Theodorou, 566/2558

15 giugno 2008

## Indice

<b>1</b>	<b>Introduzione</b>	<b>1</b>
1.1	Descrizione sintetica . . . . .	1
1.2	Note sulla progettazione . . . . .	2
1.3	Note sull'implementazione . . . . .	2
<b>2</b>	<b>Class Diagram</b>	<b>3</b>
<b>3</b>	<b>Class Diagram ristrutturato</b>	<b>4</b>
<b>4</b>	<b>Dizionario dei dati</b>	<b>5</b>
4.1	Dizionario delle classi . . . . .	5
4.2	Dizionario delle Associazioni . . . . .	6
<b>5</b>	<b>Schema relazionale</b>	<b>6</b>
<b>6</b>	<b>Definizione dei vincoli</b>	<b>9</b>
<b>7</b>	<b>Esempio</b>	<b>12</b>

## 1 Introduzione

### 1.1 Descrizione sintetica

Il problema pone in essere la memorizzazione della struttura e del formato dei più comuni documenti e delle loro istanze all'interno di una base di dati relazionale.

- **La struttura**

L'approccio risolutivo adottato descrive un documento come un insieme di elementi tra loro correlati attraverso una relazione di contenimento.

Il documento è interamente individuato da un elemento radice che potrà contenere altri elementi, che a loro volta potranno contenerne degli altri.

Al fine di una più chiara organizzazione e strutturazione, gli elementi sono distinti in due macro-categorie: gli elementi contenitore e gli elementi contenuto.

Mentre i primi possono contenere esclusivamente altri elementi (siano essi contenitori o contenuti), gli elementi contenuto memorizzeranno l'informazione vera e propria (nei fatti rappresentata da semplice testo, che potrà essere interpretato in maniera diversa a seconda del sottotipo dell'elemento).

Ogni elemento può inoltre essere arricchito con ulteriori informazioni attraverso la definizione di nuovi attributi.

- **Il formato**

Il formato è inteso come un insieme di regole, ognuna delle quali individua un tipo di elemento particolare.

Ad ogni regola vengono associati degli attributi, rappresentati da coppie nome/valore che ne descrivono le proprietà.

- **L'istanziamento:**

Successivamente alla definizione della struttura del documento, è possibile creare le istanze degli elementi che lo compongono.

## 1.2 Note sulla progettazione

La relazione di contenimento tra tipi non fa riferimento ad un particolare tipo di documento, affinché sia possibile il loro riutilizzo all'interno di diversi tipi di documento.

Per quanto riguarda la relazione di contenimento tra istanze di elementi questo non è ammesso, ovvero non è possibile che un'istanza di un elemento sia contenuta da un'altra istanza appartenente ad un documento diverso da quello associato alla prima.

## 1.3 Note sull'implementazione

L'attributo "QualsiasiContenuto" della classe "Contenitore" permette allo stesso di contenere elementi senza vincoli di tipo, di ordine o di cardinalità.

L'attributo "Ordine" della classe associazione "Contenimento" stabilisce un ordine relativo tra elementi che condividono lo stesso contenitore.

Per assicurare la conformità tra l'ordine delle istanze e quello definito per i tipi, è stata creata una vista particolare, col compito di mettere in relazione tali ordini. Essa viene poi utilizzata per semplificare la creazione dell'asserzione relativa.





## 4 Dizionario dei dati

### 4.1 Dizionario delle classi

Nome Classe	Descrizione	Attributi	Chiavi
Tipo_Elemento	Memorizza i tipi definiti dall'utente	Id: INTEGER Nome: VARCHAR Descrizione: VARCHAR	Id
Contenitore	Definisce gli elementi derivati che possono contenere altri elementi	Tipo: INTEGER QualsiasiContenuto: BOOLEAN	Tipo
Contenuto	Definisce gli elementi derivati che non possono contenere altri elementi	Tipo: INTEGER Testo: BOOLEAN Codifica: VARCHAR Lingua: VARCHAR URI: BOOLEAN Protocollo: VARCHAR Marcatore: BOOLEAN	Tipo
Contenimento	Definisce il contenuto ammissibile per un elemento di tipo contenitore	NumMax: INTEGER NumMin: INTEGER Ordine: INTEGER	
Tipo_Attributo	Memorizza i tipi di attributo definiti dall'utente	Id: INTEGER Nome: VARCHAR Descrizione: VARCHAR Obbligatorio: BOOLEAN TipoValore: VARCHAR	Id
Istanza_Elemento	Memorizza tutte le istanze di elementi presenti nella base di dati	Id: INTEGER Nome: VARCHAR Contenuto: TEXT	Id
Contenimento_Istanze	Memorizza il contenuto di specifiche istanze di tipo contenitore	Ordine: INTEGER	
Istanza_Attributo	Memorizza tutte le istanze di attributi presenti nella base di dati	Id: INTEGER Valore: VARCHAR	Id
Formato	Definisce un insieme di regole da associare ad un tipo di documento	Id: INTEGER Nome: VARCHAR Descrizione: VARCHAR	Id
Tipo_Documento	Definisce i tipi di documento ammessi, eventualmente associati ad un formato	Id: INTEGER Nome: VARCHAR Descrizione: VARCHAR Categoria: TEXT	Id
Documento	Memorizza tutte le istanze di documenti presenti nella base di dati	Radice: INTEGER Nome: VARCHAR Autore: VARCHAR DataCreazione: DATE	Radice
Regola	Definisce una regola da associare ad un formato	Id: INTEGER Nome: VARCHAR Descrizione: VARCHAR	Id
Attributo_Regola	Definisce un attributo da associare ad una regola	Id: INTEGER Nome: VARCHAR Valore: VARCHAR TipoValore: VARCHAR	Id

## 4.2 Dizionario delle Associazioni

Nome Associazione	Descrizione	Classi Coinvolte	Classi Associazione
Contenimento	Definisce il contenuto ammissibile per un elemento	Tipo_Elemento, Contiene, 1..* Tipo_Elemento, È contenuto da, 0..*	Contenimento
Contenimento_Istanze	Memorizza il contenuto di specifiche istanze	Istanza_Elemento, Contiene, 0..* Istanza_Elemento, È contenuto da, 0..*	Contenimento_Istanze

## 5 Schema relazionale

Nel seguente schema sintetico sono state evidenziate le chiavi primarie mediante sottolineatura, mentre le chiavi esterne sono state riportate in corsivo.

Tipo\_Elemento(Id, Nome, Descrizione)  
 Contenitore(Tipo, QualsiasiContenuto)  
 Contenuto(Tipo, Testo, Codifica, Lingua, URI, Protocollo, Marcatore)  
 Contenimento(*Contenitore*, *Contenuto*, NumMax, NumMin, Ordine)  
 Tipo\_Attributo(Id, Nome, Descrizione, Obbligatorio, TipoValore, *Tipo*)  
 Istanza\_Elemento(Id, Nome, Contenuto, *Tipo*)  
 Contenimento\_Istanze(*Contenitore*, *Contenuto*, Ordine)  
 Istanza\_Attributo(Id, *TipoAttr*, *Istanza*, Valore)  
 Formato(Id, Nome, Descrizione)  
 Tipo\_Documento(Id, Nome, Descrizione, Categoria, *TipoRadice*, *Formato*)  
 Documento(Radice, Nome, Autore, DataCreazione, *TipoDoc*)  
 Regola(Id, Nome, Descrizione, *SelettoreTipo*)  
 Composizione\_Regole(*Formato*, *Regola*)  
 Attributo\_Regola(Id, Nome, Valore, TipoValore)  
 Composizione\_AttrReg(*Regola*, *Attributo*)

Listing 1: Definizione della BD

```

1 CREATE TABLE Tipo_Elemento (
2   Id INTEGER PRIMARY KEY,
3   Nome VARCHAR(64) NOT NULL,
4   Descrizione VARCHAR(64) DEFAULT NULL
5 );
6
7
8 CREATE TABLE Contenitore (
9   Tipo INTEGER PRIMARY KEY,
10  QualsiasiContenuto BOOLEAN NOT NULL,
11  CONSTRAINT fk_Contentitore_Tipo FOREIGN KEY (Tipo) REFERENCES Tipo_Elemento(Id)
12     ON DELETE CASCADE ON UPDATE CASCADE
13 );
14
15 CREATE TABLE Contenuto (
16   Tipo INTEGER PRIMARY KEY,
17   Testo BOOLEAN DEFAULT FALSE NOT NULL,
18   Codifica VARCHAR(64),
19   Lingua VARCHAR(64),
20   URI BOOLEAN DEFAULT FALSE NOT NULL,

```

```

21  Protocollo VARCHAR(64),
22  Marcatore BOOLEAN DEFAULT FALSE NOT NULL,
23  CONSTRAINT fk_Contentuto_Tipo FOREIGN KEY (Tipo) REFERENCES Tipo_Elemento(Id)
      ON DELETE CASCADE ON UPDATE CASCADE
24  CONSTRAINT Contentuto_mutua_esclusione CHECK ((Testo = TRUE AND URI = FALSE AND
      Marcatore = FALSE)
25  OR (Testo = FALSE AND URI = TRUE AND Marcatore = FALSE)
26  OR (Testo = FALSE AND URI = FALSE AND Marcatore = TRUE))
27 );
28
29
30 CREATE TABLE Contenimento (
31   Contenitore INTEGER NOT NULL,
32   Contentuto INTEGER NOT NULL,
33   NumMax INTEGER DEFAULT NULL CHECK (NumMax > 0),
34   NumMin INTEGER DEFAULT NULL CHECK (NumMin > 0 AND NumMin <= NumMax),
35   Ordine INTEGER NOT NULL CHECK (Ordine >= 0),
36   CONSTRAINT fk_Contentimento_Contenitore FOREIGN KEY (Contenitore) REFERENCES
      Contenitore(Tipo) ON DELETE CASCADE ON UPDATE CASCADE
37   CONSTRAINT fk_Contentimento_Contentuto FOREIGN KEY (Contentuto) REFERENCES
      Tipo_Elemento(Id) ON DELETE CASCADE ON UPDATE CASCADE
38   CONSTRAINT Contenimento_coppia_unica UNIQUE (Contenitore, Contentuto),
39   CONSTRAINT Contenimento_ordine_unico UNIQUE (Contenitore, Ordine),
40   CONSTRAINT Contenimento_auto_contenimento CHECK (Contenitore <> Contentuto)
41 );
42
43
44 CREATE TABLE Tipo_Attributo (
45   Id INTEGER PRIMARY KEY,
46   Nome VARCHAR(64) NOT NULL,
47   Descrizione VARCHAR(64) DEFAULT NULL,
48   Obbligatorio BOOLEAN NOT NULL,
49   TipoValore VARCHAR(64) NOT NULL,
50   Tipo INTEGER NOT NULL,
51   CONSTRAINT fk_Tipo_Attributo_Tipo FOREIGN KEY (Tipo) REFERENCES Tipo_Elemento(
      Id) ON DELETE CASCADE ON UPDATE CASCADE
52 );
53
54
55 CREATE TABLE Istanza_Elemento (
56   Id INTEGER PRIMARY KEY,
57   Nome VARCHAR(64) NOT NULL,
58   Contentuto TEXT DEFAULT NULL,
59   Tipo INTEGER NOT NULL,
60   — Documento INTEGER, — aggiunta in differita
61   — CONSTRAINT fk_Istanza_Elemento_Documento FOREIGN KEY (Documento) REFERENCES
      Documento(Radice) ON DELETE SET NULL ON UPDATE CASCADE, — aggiunta in
      differita
62   CONSTRAINT fk_Istanza_Elemento_Tipo FOREIGN KEY (Tipo) REFERENCES
      Tipo_Elemento(Id) ON DELETE CASCADE ON UPDATE CASCADE
63 );
64
65
66 CREATE TABLE Contenimento_Istanze (
67   Contenitore INTEGER NOT NULL,
68   Contentuto INTEGER NOT NULL,
69   Ordine INTEGER NOT NULL CHECK (Ordine >= 0),
70   CONSTRAINT fk_Contentimento_Istanze_Contenitore FOREIGN KEY (Contenitore)
      REFERENCES Istanza_Elemento(Id) ON DELETE CASCADE ON UPDATE CASCADE
71   CONSTRAINT fk_Contentimento_Istanze_Contentuto FOREIGN KEY (Contentuto)
      REFERENCES Istanza_Elemento(Id) ON DELETE CASCADE ON UPDATE CASCADE

```

```

72  CONSTRAINT Contenimento_Istanze_coppia_unica UNIQUE (Contenitore, Contenuto),
73  CONSTRAINT Contenimento_Istanze_ordine_unico UNIQUE (Contenitore, Ordine)
74 );
75
76
77 CREATE TABLE Istanza_Attributo (
78   Id INTEGER PRIMARY KEY,
79   TipoAttr INTEGER NOT NULL,
80   Istanza INTEGER NOT NULL,
81   Valore VARCHAR(64) NOT NULL,
82   CONSTRAINT fk_Istanza_Attributo_TipoAttr FOREIGN KEY (TipoAttr) REFERENCES
      Tipo_Attributo(Id) ON DELETE CASCADE ON UPDATE CASCADE
83   CONSTRAINT fk_Istanza_Attributo_Istanza FOREIGN KEY (Istanza) REFERENCES
      Istanza_Elemento(Id) ON DELETE CASCADE ON UPDATE CASCADE
84   CONSTRAINT Istanza_Attributo_coppia_unica UNIQUE (TipoAttr, Istanza)
85 );
86
87
88 CREATE TABLE Formato (
89   Id INTEGER PRIMARY KEY,
90   Nome VARCHAR(64) NOT NULL,
91   Descrizione VARCHAR(64) DEFAULT NULL
92 );
93
94
95 CREATE TABLE Tipo_Documento (
96   Id INTEGER PRIMARY KEY,
97   Nome VARCHAR(64) NOT NULL,
98   Descrizione VARCHAR(64) DEFAULT NULL,
99   Categoria VARCHAR(64),
100  TipoRadice INTEGER NOT NULL,
101  Formato INTEGER,
102  CONSTRAINT fk_Tipo_Documento_TipoRadice FOREIGN KEY (TipoRadice) REFERENCES
      Tipo_Elemento(Id) ON DELETE CASCADE ON UPDATE CASCADE
103  CONSTRAINT fk_Tipo_Documento_Formato FOREIGN KEY (Formato) REFERENCES Formato(
      Id) ON DELETE SET NULL ON UPDATE CASCADE
104 );
105
106
107 CREATE TABLE Documento (
108   Radice INTEGER PRIMARY KEY,
109   Nome VARCHAR(64) NOT NULL,
110   Autore VARCHAR(64) NOT NULL DEFAULT 'Anonimo',
111   DataCreazione DATE DEFAULT current_date,
112   TipoDoc INTEGER NOT NULL,
113   CONSTRAINT fk_Documento_Radice FOREIGN KEY (Radice) REFERENCES
      Istanza_Elemento(Id) ON DELETE CASCADE ON UPDATE CASCADE
114   CONSTRAINT fk_Documento_TipoDoc FOREIGN KEY (TipoDoc) REFERENCES
      Tipo_Documento(Id) ON DELETE CASCADE ON UPDATE CASCADE
115 );
116
117
118 — Risolve il problema del riferimento incrociato nell'inserimento della radice
119 ALTER TABLE Istanza_Elemento ADD COLUMN Documento INTEGER;
120 ALTER TABLE Istanza_Elemento ADD CONSTRAINT fk_Istanza_Elemento_Documento
      FOREIGN KEY (Documento) REFERENCES Documento(Radice) ON DELETE SET NULL ON
      UPDATE CASCADE;
121
122
123 CREATE TABLE Regola (
124   Id INTEGER PRIMARY KEY,

```

```

125 Nome VARCHAR(64) NOT NULL,
126 Descrizione VARCHAR(64) DEFAULT NULL,
127 SelettoreTipo INTEGER NOT NULL,
128 CONSTRAINT fk_Regole_SelettoreTipo FOREIGN KEY (SelettoreTipo) REFERENCES
    Tipo_Elemento(Id) ON DELETE CASCADE ON UPDATE CASCADE
129 );
130
131
132 CREATE TABLE Composizione_Regole (
133     Formato INTEGER NOT NULL,
134     Regola INTEGER NOT NULL,
135     CONSTRAINT fk_Composizione_Regole_Formato FOREIGN KEY (Formato) REFERENCES
        Formato(Id) ON DELETE CASCADE ON UPDATE CASCADE
136     CONSTRAINT fk_Composizione_Regole_Regola FOREIGN KEY (Regola) REFERENCES
        Regola(Id) ON DELETE CASCADE ON UPDATE CASCADE
137     CONSTRAINT Composizione_Regole_coppia_unica UNIQUE(Formato, Regola)
138 );
139
140
141 CREATE TABLE Attributo_Regola (
142     Id INTEGER PRIMARY KEY,
143     Nome VARCHAR(64) NOT NULL,
144     Valore VARCHAR(64) NOT NULL,
145     TipoValore VARCHAR(64) NOT NULL
146 );
147
148
149 CREATE TABLE Composizione_AttrReg (
150     Regola INTEGER NOT NULL,
151     Attributo INTEGER NOT NULL,
152     CONSTRAINT fk_Composizione_AttrReg_Regola FOREIGN KEY (Regola) REFERENCES
        Regola(Id) ON DELETE CASCADE ON UPDATE CASCADE
153     CONSTRAINT fk_Composizione_AttrReg_Attributo FOREIGN KEY (Attributo)
        REFERENCES Attributo_Regola(Id) ON DELETE CASCADE ON UPDATE CASCADE
154     CONSTRAINT Composizione_AttrReg_coppia_unica UNIQUE(Regola, Attributo)
155 );
156
157
158 CREATE VIEW VOrd AS
159     SELECT CI1.Contenitore, CI1.Contenuto, IE1.Tipo AS TipoContenitore, IE2.Tipo
        AS TipoContenuto, CI1.Ordine AS OrdineIstanza, CO.Ordine AS OrdineTipo
160     FROM ((Contenimento_Istanze CI1 JOIN Istanza_Elemento IE1 ON CI1.Contenitore =
        IE1.Id) JOIN Istanza_Elemento IE2 ON CI1.Contenuto = IE2.Id)
161     JOIN Contenimento CO ON IE1.Tipo = CO.Contenitore AND IE2.Tipo = CO.
        Contenuto; — Evidenzia gli ordini relativi per i tipi e per le istanza,
        data ogni tupla della relazione di contenimento delle istanze

```

## 6 Definizione dei vincoli

La seguente asserzione garantisce che il contenuto di una particolare istanza rispetti il vincolo di contenimento definito per il suo tipo.

Listing 2: CheckContenutoIstanze

```

1 CREATE ASSERTION CheckContenutoIstanze
2 CHECK NOT EXISTS (
3     SELECT *
4     FROM Contenimento_Istanze CI JOIN Istanza_Elemento IE1 ON CI.Contenitore =
        IE1.Id
5     JOIN Istanza_Elemento IE2 ON CI.Contenuto = IE2.Id

```

```

6      JOIN Contenitore CO ON CO.Tipo = IE1.Tipo
7      WHERE CO.QualsiasiContenuto = FALSE AND
8      (IE1.Tipo, IE2.Tipo) NOT IN (SELECT Contenitore, Contenuto FROM
9      Contenimento);

```

La seguente asserzione garantisce che l'istanza di un attributo rispetti il vincolo sul tipo di elemento associabile come indicato nella sua definizione.

Listing 3: CheckIstanzaAttributo

```

1 CREATE ASSERTION CheckIstanzaAttributo
2 CHECK NOT EXISTS (
3     SELECT *
4     FROM Istanza_Attributo IA JOIN Istanza_Elemento IE ON IA.Istanza = IE.Id
5     JOIN Tipo_Attributo TA ON IA.TipoAttr = TA.Id
6     WHERE IE.Tipo <> TA.Tipo;
7 );

```

La seguente asserzione garantisce che l'istanza di un documento rispetti i vincoli sul tipo di radice ammissibile come definito per il tipo di documento ad esso associato.

Listing 4: CheckRadiceDocumento

```

1 CREATE ASSERTION CheckRadiceDocumento
2 CHECK NOT EXISTS (
3     SELECT *
4     FROM Documento D JOIN Istanza_Elemento IE ON D.Radice = IE.Id
5     JOIN Tipo_Documento TD ON D.TipoDoc = TD.Id
6     WHERE TD.TipoRadice <> IE.Tipo;
7 );

```

La seguente asserzione garantisce che i vincoli sul numero massimo e minimo di elementi, contenibili da un particolare tipo, siano rispettati da una sua istanza.

Listing 5: CheckNumContenuti

```

1 CREATE ASSERTION CheckNumContenuti
2 CHECK NOT EXISTS (
3     SELECT CI.Contenitore, COUNT(*), C.NumMax, C.NumMin
4     FROM Contenimento C JOIN Istanza_Elemento IE ON C.Contenitore = IE.Tipo
5     JOIN Contenimento_Istanze CI ON CI.Contenitore = IE.Id
6     GROUP BY CI.Contenitore, C.NumMax, C.NumMin
7     HAVING COUNT(*) > NumMax OR COUNT(*) < NumMin;
8 ) DEFERRED;

```

La seguente asserzione garantisce che la specializzazione di un tipo in contenuto o contenitore sia disgiunta.

Listing 6: CheckDisgiunzioneTipi

```

1 CREATE ASSERTION CheckDisgiunzioneTipi
2 CHECK NOT EXISTS (
3     SELECT ore.Tipo, uto.Tipo
4     FROM Contenitore ore JOIN Contenuto uto ON ore.Tipo = uto.Tipo;
5 );

```

La seguente asserzione garantisce che la specializzazione di un tipo in contenuto o contenitore sia totale.

Listing 7: CheckTotaleTipi

```

1 CREATE ASSERTION CheckTotaleTipi
2 CHECK NOT EXISTS (
3     (SELECT Id FROM Tipo_Elemento)
4 EXCEPT
5     (SELECT Tipo FROM Contenitore UNION SELECT Tipo FROM Contenuto);
6 );

```

La seguente asserzione garantisce che un contenitore per cui sia impostata la proprietà di contenimento arbitrario non figuri nella relazione di contenimento come contenitore.

Listing 8: CheckQualsiasiContenuto

```

1 CREATE ASSERTION CheckQualsiasiContenuto
2 CHECK NOT EXISTS (
3     SELECT *
4     FROM Contenitore ore JOIN Contenimento nto ON ore.Tipo = nto.Contenitore
5     WHERE ore.QualsiasiContenuto = TRUE;
6 );

```

La seguente asserzione garantisce che il contenuto di una istanza che sia di tipo contenitore sia nullo.

Listing 9: CheckContenutoIstanza

```

1 CREATE ASSERTION CheckContenutoIstanza
2 CHECK NOT EXISTS (
3     SELECT *
4     FROM Istanza_Elemento IE JOIN Contenitore CO ON IE.Tipo = CO.Tipo
5     WHERE IE.Contenuto IS NOT NULL;
6 );

```

La seguente asserzione garantisce che l'ordinamento tra le istanze rispetti quello definito per i rispettivi tipi.

Listing 10: CheckOrdinamento

```

1 CREATE ASSERTION CheckOrdinamento
2 CHECK NOT EXISTS (
3     SELECT *
4     FROM VOrd V1 JOIN VOrd V2 ON V1.Contenitore = V2.Contenitore
5     WHERE V1.Contenuto <> V2.Contenuto
6         AND V1.OrdineTipo < V2.OrdineTipo
7         AND V1.OrdineIstanza > V2.OrdineIstanza;
8 );

```

La seguente asserzione garantisce che l'istanza contenuta da un'istanza contenitore appartenga allo stesso documento di quest'ultima.

Listing 11: CheckDocumentoContenuto

```

1 CREATE ASSERTION CheckDocumentoContenuto
2 CHECK NOT EXISTS (
3     SELECT *
4     FROM Contenimento_Istanze CI JOIN Istanza_Elemento IE1 ON CI.Contenitore =
5         IE1.Id

```

```

5 JOIN Istanza_Elemento IE2 ON CI.Contenuto = IE2.Id
6 WHERE IE1.Documento <> IE2.Documento;
7 );

```

## 7 Esempio

Listing 12: Popolamento della BD con alcuni dati di prova

```

1 INSERT INTO Tipo_Elemento VALUES
2 (11, 'TipoLibro'),
3 (12, 'TipoCapitolo'),
4 (13, 'TipoSezione'),
5 (14, 'TipoSottosezione'),
6 (15, 'TipoTesto');
7
8 INSERT INTO Contenitore VALUES
9 (11, FALSE),
10 (12, FALSE),
11 (13, FALSE),
12 (14, FALSE);
13
14 INSERT INTO Contenuto VALUES (15, TRUE, NULL, NULL, FALSE, NULL, FALSE);
15
16 INSERT INTO Contenimento VALUES
17 (11, 12, NULL, NULL, 0),
18 (12, 13, NULL, NULL, 0),
19 (13, 14, NULL, NULL, 0),
20 (14, 15, NULL, NULL, 0);
21
22 INSERT INTO Tipo_Attributo VALUES (31, 'SaltaIndice', NULL, FALSE, 'Booleano',
23 12);
24
25 INSERT INTO Istanza_Elemento VALUES
26 (21, 'Libro', NULL, 11),
27 (22, 'Capitolo 1', NULL, 12),
28 (23, 'Sezione 1', NULL, 13),
29 (24, 'Sottosezione 1', NULL, 14),
30 (25, 'Testo', 'Testo vero e proprio', 15);
31
32 INSERT INTO Contenimento_Istanze VALUES
33 (21, 22, 0),
34 (22, 23, 0),
35 (23, 24, 0),
36 (24, 25, 0);
37
38 INSERT INTO Istanza_Attributo VALUES (41, 31, 22, 'Vero');
39
40 INSERT INTO Formato VALUES (51, 'Default');
41 INSERT INTO Regola VALUES (61, 'RegolaCapitolo', NULL, 12);
42 INSERT INTO Composizione_Regole VALUES (51, 61);
43 INSERT INTO Attributo_Regola VALUES (71, 'CapoLettera', 'Vero', 'Booleano');
44 INSERT INTO Composizione_AttrReg VALUES (61, 71);
45
46 INSERT INTO Tipo_Documento VALUES (81, 'TipoDocLibro', NULL, NULL, 11, 51);
47 INSERT INTO Documento VALUES (21, 'Libro', 'Anonimo', current_date, 81);

```

# Documentazione progetto

Gerardo Adelizzi, 566/2471  
Angelo Theodorou, 566/2558

15 giugno 2008

## Indice

<b>1</b>	<b>Introduzione</b>	<b>1</b>
1.1	Descrizione sintetica . . . . .	1
1.2	Note sulla progettazione . . . . .	2
1.3	Note sull'implementazione . . . . .	2
<b>2</b>	<b>Class Diagram</b>	<b>3</b>
<b>3</b>	<b>Class Diagram ristrutturato</b>	<b>4</b>
<b>4</b>	<b>Dizionario dei dati</b>	<b>5</b>
4.1	Dizionario delle classi . . . . .	5
4.2	Dizionario delle Associazioni . . . . .	6
<b>5</b>	<b>Schema relazionale</b>	<b>6</b>
<b>6</b>	<b>Definizione dei vincoli</b>	<b>9</b>
<b>7</b>	<b>Esempio</b>	<b>12</b>

## 1 Introduzione

### 1.1 Descrizione sintetica

Il problema pone in essere la memorizzazione della struttura e del formato dei più comuni documenti e delle loro istanze all'interno di una base di dati relazionale.

- **La struttura**

L'approccio risolutivo adottato descrive un documento come un insieme di elementi tra loro correlati attraverso una relazione di contenimento.

Il documento è interamente individuato da un elemento radice che potrà contenere altri elementi, che a loro volta potranno contenerne degli altri.

Al fine di una più chiara organizzazione e strutturazione, gli elementi sono distinti in due macro-categorie: gli elementi contenitore e gli elementi contenuto.

Mentre i primi possono contenere esclusivamente altri elementi (siano essi contenitori o contenuti), gli elementi contenuto memorizzeranno l'informazione vera e propria (nei fatti rappresentata da semplice testo, che potrà essere interpretato in maniera diversa a seconda del sottotipo dell'elemento).

Ogni elemento può inoltre essere arricchito con ulteriori informazioni attraverso la definizione di nuovi attributi.

- **Il formato**

Il formato è inteso come un insieme di regole, ognuna delle quali individua un tipo di elemento particolare.

Ad ogni regola vengono associati degli attributi, rappresentati da coppie nome/valore che ne descrivono le proprietà.

- **L'istanziamento:**

Successivamente alla definizione della struttura del documento, è possibile creare le istanze degli elementi che lo compongono.

## 1.2 Note sulla progettazione

La relazione di contenimento tra tipi non fa riferimento ad un particolare tipo di documento, affinché sia possibile il loro riutilizzo all'interno di diversi tipi di documento.

Per quanto riguarda la relazione di contenimento tra istanze di elementi questo non è ammesso, ovvero non è possibile che un'istanza di un elemento sia contenuta da un'altra istanza appartenente ad un documento diverso da quello associato alla prima.

## 1.3 Note sull'implementazione

L'attributo "QualsiasiContenuto" della classe "Contenitore" permette allo stesso di contenere elementi senza vincoli di tipo, di ordine o di cardinalità.

L'attributo "Ordine" della classe associazione "Contenimento" stabilisce un ordine relativo tra elementi che condividono lo stesso contenitore.

Per assicurare la conformità tra l'ordine delle istanze e quello definito per i tipi, è stata creata una vista particolare, col compito di mettere in relazione tali ordini. Essa viene poi utilizzata per semplificare la creazione dell'asserzione relativa.





## 4 Dizionario dei dati

### 4.1 Dizionario delle classi

Nome Classe	Descrizione	Attributi	Chiavi
Tipo_Elemento	Memorizza i tipi definiti dall'utente	Id: INTEGER Nome: VARCHAR Descrizione: VARCHAR	Id
Contenitore	Definisce gli elementi derivati che possono contenere altri elementi	Tipo: INTEGER QualsiasiContenuto: BOOLEAN	Tipo
Contenuto	Definisce gli elementi derivati che non possono contenere altri elementi	Tipo: INTEGER Testo: BOOLEAN Codifica: VARCHAR Lingua: VARCHAR URI: BOOLEAN Protocollo: VARCHAR Marcatore: BOOLEAN	Tipo
Contenimento	Definisce il contenuto ammissibile per un elemento di tipo contenitore	NumMax: INTEGER NumMin: INTEGER Ordine: INTEGER	
Tipo_Attributo	Memorizza i tipi di attributo definiti dall'utente	Id: INTEGER Nome: VARCHAR Descrizione: VARCHAR Obbligatorio: BOOLEAN TipoValore: VARCHAR	Id
Istanza_Elemento	Memorizza tutte le istanze di elementi presenti nella base di dati	Id: INTEGER Nome: VARCHAR Contenuto: TEXT	Id
Contenimento_Istanze	Memorizza il contenuto di specifiche istanze di tipo contenitore	Ordine: INTEGER	
Istanza_Attributo	Memorizza tutte le istanze di attributi presenti nella base di dati	Id: INTEGER Valore: VARCHAR	Id
Formato	Definisce un insieme di regole da associare ad un tipo di documento	Id: INTEGER Nome: VARCHAR Descrizione: VARCHAR	Id
Tipo_Documento	Definisce i tipi di documento ammessi, eventualmente associati ad un formato	Id: INTEGER Nome: VARCHAR Descrizione: VARCHAR Categoria: TEXT	Id
Documento	Memorizza tutte le istanze di documenti presenti nella base di dati	Radice: INTEGER Nome: VARCHAR Autore: VARCHAR DataCreazione: DATE	Radice
Regola	Definisce una regola da associare ad un formato	Id: INTEGER Nome: VARCHAR Descrizione: VARCHAR	Id
Attributo_Regola	Definisce un attributo da associare ad una regola	Id: INTEGER Nome: VARCHAR Valore: VARCHAR TipoValore: VARCHAR	Id

## 4.2 Dizionario delle Associazioni

Nome Associazione	Descrizione	Classi Coinvolte	Classi Associazione
Contenimento	Definisce il contenuto ammissibile per un elemento	Tipo_Elemento, Contiene, 1..* Tipo_Elemento, È contenuto da, 0..*	Contenimento
Contenimento_Istanze	Memorizza il contenuto di specifiche istanze	Istanza_Elemento, Contiene, 0..* Istanza_Elemento, È contenuto da, 0..*	Contenimento_Istanze

## 5 Schema relazionale

Nel seguente schema sintetico sono state evidenziate le chiavi primarie mediante sottolineatura, mentre le chiavi esterne sono state riportate in corsivo.

Tipo\_Elemento(Id, Nome, Descrizione)  
 Contenitore(Tipo, QualsiasiContenuto)  
 Contenuto(Tipo, Testo, Codifica, Lingua, URI, Protocollo, Marcatore)  
 Contenimento(*Contenitore*, *Contenuto*, NumMax, NumMin, Ordine)  
 Tipo\_Attributo(Id, Nome, Descrizione, Obbligatorio, TipoValore, *Tipo*)  
 Istanza\_Elemento(Id, Nome, Contenuto, *Tipo*)  
 Contenimento\_Istanze(*Contenitore*, *Contenuto*, Ordine)  
 Istanza\_Attributo(Id, *TipoAttr*, *Istanza*, Valore)  
 Formato(Id, Nome, Descrizione)  
 Tipo\_Documento(Id, Nome, Descrizione, Categoria, *TipoRadice*, *Formato*)  
 Documento(Radice, Nome, Autore, DataCreazione, *TipoDoc*)  
 Regola(Id, Nome, Descrizione, *SelettoreTipo*)  
 Composizione\_Regole(*Formato*, *Regola*)  
 Attributo\_Regola(Id, Nome, Valore, TipoValore)  
 Composizione\_AttrReg(*Regola*, *Attributo*)

Listing 1: Definizione della BD

```

1 CREATE TABLE Tipo_Elemento (
2   Id INTEGER PRIMARY KEY,
3   Nome VARCHAR(64) NOT NULL,
4   Descrizione VARCHAR(64) DEFAULT NULL
5 );
6
7
8 CREATE TABLE Contenitore (
9   Tipo INTEGER PRIMARY KEY,
10  QualsiasiContenuto BOOLEAN NOT NULL,
11  CONSTRAINT fk_Contentitore_Tipo FOREIGN KEY (Tipo) REFERENCES Tipo_Elemento(Id)
12  ON DELETE CASCADE ON UPDATE CASCADE
13 );
14
15 CREATE TABLE Contenuto (
16   Tipo INTEGER PRIMARY KEY,
17   Testo BOOLEAN DEFAULT FALSE NOT NULL,
18   Codifica VARCHAR(64),
19   Lingua VARCHAR(64),
20   URI BOOLEAN DEFAULT FALSE NOT NULL,

```

```

21  Protocollo VARCHAR(64),
22  Marcatore BOOLEAN DEFAULT FALSE NOT NULL,
23  CONSTRAINT fk_Contentuto_Tipo FOREIGN KEY (Tipo) REFERENCES Tipo_Elemento(Id)
      ON DELETE CASCADE ON UPDATE CASCADE
24  CONSTRAINT Contentuto_mutua_esclusione CHECK ((Testo = TRUE AND URI = FALSE AND
      Marcatore = FALSE)
25  OR (Testo = FALSE AND URI = TRUE AND Marcatore = FALSE)
26  OR (Testo = FALSE AND URI = FALSE AND Marcatore = TRUE))
27 );
28
29
30 CREATE TABLE Contenimento (
31   Contenitore INTEGER NOT NULL,
32   Contentuto INTEGER NOT NULL,
33   NumMax INTEGER DEFAULT NULL CHECK (NumMax > 0),
34   NumMin INTEGER DEFAULT NULL CHECK (NumMin > 0 AND NumMin <= NumMax),
35   Ordine INTEGER NOT NULL CHECK (Ordine >= 0),
36   CONSTRAINT fk_Contentimento_Contenitore FOREIGN KEY (Contenitore) REFERENCES
      Contenitore(Tipo) ON DELETE CASCADE ON UPDATE CASCADE
37   CONSTRAINT fk_Contentimento_Contentuto FOREIGN KEY (Contentuto) REFERENCES
      Tipo_Elemento(Id) ON DELETE CASCADE ON UPDATE CASCADE
38   CONSTRAINT Contenimento_coppia_unica UNIQUE (Contenitore, Contentuto),
39   CONSTRAINT Contenimento_ordine_unico UNIQUE (Contenitore, Ordine),
40   CONSTRAINT Contenimento_auto_contenimento CHECK (Contenitore <> Contentuto)
41 );
42
43
44 CREATE TABLE Tipo_Attributo (
45   Id INTEGER PRIMARY KEY,
46   Nome VARCHAR(64) NOT NULL,
47   Descrizione VARCHAR(64) DEFAULT NULL,
48   Obbligatorio BOOLEAN NOT NULL,
49   TipoValore VARCHAR(64) NOT NULL,
50   Tipo INTEGER NOT NULL,
51   CONSTRAINT fk_Tipo_Attributo_Tipo FOREIGN KEY (Tipo) REFERENCES Tipo_Elemento(
      Id) ON DELETE CASCADE ON UPDATE CASCADE
52 );
53
54
55 CREATE TABLE Istanza_Elemento (
56   Id INTEGER PRIMARY KEY,
57   Nome VARCHAR(64) NOT NULL,
58   Contentuto TEXT DEFAULT NULL,
59   Tipo INTEGER NOT NULL,
60   — Documento INTEGER, — aggiunta in differita
61   — CONSTRAINT fk_Istanza_Elemento_Documento FOREIGN KEY (Documento) REFERENCES
      Documento(Radice) ON DELETE SET NULL ON UPDATE CASCADE, — aggiunta in
      differita
62   CONSTRAINT fk_Istanza_Elemento_Tipo FOREIGN KEY (Tipo) REFERENCES
      Tipo_Elemento(Id) ON DELETE CASCADE ON UPDATE CASCADE
63 );
64
65
66 CREATE TABLE Contenimento_Istanze (
67   Contenitore INTEGER NOT NULL,
68   Contentuto INTEGER NOT NULL,
69   Ordine INTEGER NOT NULL CHECK (Ordine >= 0),
70   CONSTRAINT fk_Contentimento_Istanze_Contenitore FOREIGN KEY (Contenitore)
      REFERENCES Istanza_Elemento(Id) ON DELETE CASCADE ON UPDATE CASCADE
71   CONSTRAINT fk_Contentimento_Istanze_Contentuto FOREIGN KEY (Contentuto)
      REFERENCES Istanza_Elemento(Id) ON DELETE CASCADE ON UPDATE CASCADE

```

```

72  CONSTRAINT Contenimento_Istanze_coppia_unica UNIQUE (Contenitore, Contenuto),
73  CONSTRAINT Contenimento_Istanze_ordine_unico UNIQUE (Contenitore, Ordine)
74 );
75
76
77 CREATE TABLE Istanza_Attributo (
78   Id INTEGER PRIMARY KEY,
79   TipoAttr INTEGER NOT NULL,
80   Istanza INTEGER NOT NULL,
81   Valore VARCHAR(64) NOT NULL,
82   CONSTRAINT fk_Istanza_Attributo_TipoAttr FOREIGN KEY (TipoAttr) REFERENCES
      Tipo_Attributo(Id) ON DELETE CASCADE ON UPDATE CASCADE
83   CONSTRAINT fk_Istanza_Attributo_Istanza FOREIGN KEY (Istanza) REFERENCES
      Istanza_Elemento(Id) ON DELETE CASCADE ON UPDATE CASCADE
84   CONSTRAINT Istanza_Attributo_coppia_unica UNIQUE (TipoAttr, Istanza)
85 );
86
87
88 CREATE TABLE Formato (
89   Id INTEGER PRIMARY KEY,
90   Nome VARCHAR(64) NOT NULL,
91   Descrizione VARCHAR(64) DEFAULT NULL
92 );
93
94
95 CREATE TABLE Tipo_Documento (
96   Id INTEGER PRIMARY KEY,
97   Nome VARCHAR(64) NOT NULL,
98   Descrizione VARCHAR(64) DEFAULT NULL,
99   Categoria VARCHAR(64),
100  TipoRadice INTEGER NOT NULL,
101  Formato INTEGER,
102  CONSTRAINT fk_Tipo_Documento_TipoRadice FOREIGN KEY (TipoRadice) REFERENCES
      Tipo_Elemento(Id) ON DELETE CASCADE ON UPDATE CASCADE
103  CONSTRAINT fk_Tipo_Documento_Formato FOREIGN KEY (Formato) REFERENCES Formato(
      Id) ON DELETE SET NULL ON UPDATE CASCADE
104 );
105
106
107 CREATE TABLE Documento (
108   Radice INTEGER PRIMARY KEY,
109   Nome VARCHAR(64) NOT NULL,
110   Autore VARCHAR(64) NOT NULL DEFAULT 'Anonimo',
111   DataCreazione DATE DEFAULT current_date,
112   TipoDoc INTEGER NOT NULL,
113   CONSTRAINT fk_Documento_Radice FOREIGN KEY (Radice) REFERENCES
      Istanza_Elemento(Id) ON DELETE CASCADE ON UPDATE CASCADE
114   CONSTRAINT fk_Documento_TipoDoc FOREIGN KEY (TipoDoc) REFERENCES
      Tipo_Documento(Id) ON DELETE CASCADE ON UPDATE CASCADE
115 );
116
117
118 — Risolve il problema del riferimento incrociato nell'inserimento della radice
119 ALTER TABLE Istanza_Elemento ADD COLUMN Documento INTEGER;
120 ALTER TABLE Istanza_Elemento ADD CONSTRAINT fk_Istanza_Elemento_Documento
      FOREIGN KEY (Documento) REFERENCES Documento(Radice) ON DELETE SET NULL ON
      UPDATE CASCADE;
121
122
123 CREATE TABLE Regola (
124   Id INTEGER PRIMARY KEY,

```

```

125 Nome VARCHAR(64) NOT NULL,
126 Descrizione VARCHAR(64) DEFAULT NULL,
127 SelettoreTipo INTEGER NOT NULL,
128 CONSTRAINT fk_Regole_SelettoreTipo FOREIGN KEY (SelettoreTipo) REFERENCES
    Tipo_Elemento(Id) ON DELETE CASCADE ON UPDATE CASCADE
129 );
130
131
132 CREATE TABLE Composizione_Regole (
133     Formato INTEGER NOT NULL,
134     Regola INTEGER NOT NULL,
135     CONSTRAINT fk_Composizione_Regole_Formato FOREIGN KEY (Formato) REFERENCES
        Formato(Id) ON DELETE CASCADE ON UPDATE CASCADE
136     CONSTRAINT fk_Composizione_Regole_Regola FOREIGN KEY (Regola) REFERENCES
        Regola(Id) ON DELETE CASCADE ON UPDATE CASCADE
137     CONSTRAINT Composizione_Regole_coppia_unica UNIQUE(Formato, Regola)
138 );
139
140
141 CREATE TABLE Attributo_Regola (
142     Id INTEGER PRIMARY KEY,
143     Nome VARCHAR(64) NOT NULL,
144     Valore VARCHAR(64) NOT NULL,
145     TipoValore VARCHAR(64) NOT NULL
146 );
147
148
149 CREATE TABLE Composizione_AttrReg (
150     Regola INTEGER NOT NULL,
151     Attributo INTEGER NOT NULL,
152     CONSTRAINT fk_Composizione_AttrReg_Regola FOREIGN KEY (Regola) REFERENCES
        Regola(Id) ON DELETE CASCADE ON UPDATE CASCADE
153     CONSTRAINT fk_Composizione_AttrReg_Attributo FOREIGN KEY (Attributo)
        REFERENCES Attributo_Regola(Id) ON DELETE CASCADE ON UPDATE CASCADE
154     CONSTRAINT Composizione_AttrReg_coppia_unica UNIQUE(Regola, Attributo)
155 );
156
157
158 CREATE VIEW VOrd AS
159     SELECT CI1.Contenitore, CI1.Contenuto, IE1.Tipo AS TipoContenitore, IE2.Tipo
        AS TipoContenuto, CI1.Ordine AS OrdineIstanza, CO.Ordine AS OrdineTipo
160     FROM ((Contenimento_Istanze CI1 JOIN Istanza_Elemento IE1 ON CI1.Contenitore =
        IE1.Id) JOIN Istanza_Elemento IE2 ON CI1.Contenuto = IE2.Id)
161     JOIN Contenimento CO ON IE1.Tipo = CO.Contenitore AND IE2.Tipo = CO.
        Contenuto; — Evidenzia gli ordini relativi per i tipi e per le istanza,
        data ogni tupla della relazione di contenimento delle istanze

```

## 6 Definizione dei vincoli

La seguente asserzione garantisce che il contenuto di una particolare istanza rispetti il vincolo di contenimento definito per il suo tipo.

Listing 2: CheckContenutoIstanze

```

1 CREATE ASSERTION CheckContenutoIstanze
2 CHECK NOT EXISTS (
3     SELECT *
4     FROM Contenimento_Istanze CI JOIN Istanza_Elemento IE1 ON CI.Contenitore =
        IE1.Id
5     JOIN Istanza_Elemento IE2 ON CI.Contenuto = IE2.Id

```

```

6      JOIN Contenitore CO ON CO.Tipo = IE1.Tipo
7      WHERE CO.QualsiasiContenuto = FALSE AND
8      (IE1.Tipo, IE2.Tipo) NOT IN (SELECT Contenitore, Contenuto FROM
9      Contenimento);

```

La seguente asserzione garantisce che l'istanza di un attributo rispetti il vincolo sul tipo di elemento associabile come indicato nella sua definizione.

Listing 3: CheckIstanzaAttributo

```

1 CREATE ASSERTION CheckIstanzaAttributo
2 CHECK NOT EXISTS (
3     SELECT *
4     FROM Istanza_Attributo IA JOIN Istanza_Elemento IE ON IA.Istanza = IE.Id
5     JOIN Tipo_Attributo TA ON IA.TipoAttr = TA.Id
6     WHERE IE.Tipo <> TA.Tipo;
7 );

```

La seguente asserzione garantisce che l'istanza di un documento rispetti i vincoli sul tipo di radice ammissibile come definito per il tipo di documento ad esso associato.

Listing 4: CheckRadiceDocumento

```

1 CREATE ASSERTION CheckRadiceDocumento
2 CHECK NOT EXISTS (
3     SELECT *
4     FROM Documento D JOIN Istanza_Elemento IE ON D.Radice = IE.Id
5     JOIN Tipo_Documento TD ON D.TipoDoc = TD.Id
6     WHERE TD.TipoRadice <> IE.Tipo;
7 );

```

La seguente asserzione garantisce che i vincoli sul numero massimo e minimo di elementi, contenibili da un particolare tipo, siano rispettati da una sua istanza.

Listing 5: CheckNumContenuti

```

1 CREATE ASSERTION CheckNumContenuti
2 CHECK NOT EXISTS (
3     SELECT CI.Contenitore, COUNT(*), C.NumMax, C.NumMin
4     FROM Contenimento C JOIN Istanza_Elemento IE ON C.Contenitore = IE.Tipo
5     JOIN Contenimento_Istanze CI ON CI.Contenitore = IE.Id
6     GROUP BY CI.Contenitore, C.NumMax, C.NumMin
7     HAVING COUNT(*) > NumMax OR COUNT(*) < NumMin;
8 ) DEFERRED;

```

La seguente asserzione garantisce che la specializzazione di un tipo in contenuto o contenitore sia disgiunta.

Listing 6: CheckDisgiunzioneTipi

```

1 CREATE ASSERTION CheckDisgiunzioneTipi
2 CHECK NOT EXISTS (
3     SELECT ore.Tipo, uto.Tipo
4     FROM Contenitore ore JOIN Contenuto uto ON ore.Tipo = uto.Tipo;
5 );

```

La seguente asserzione garantisce che la specializzazione di un tipo in contenuto o contenitore sia totale.

Listing 7: CheckTotaleTipi

```

1 CREATE ASSERTION CheckTotaleTipi
2 CHECK NOT EXISTS (
3     (SELECT Id FROM Tipo_Elemento)
4 EXCEPT
5     (SELECT Tipo FROM Contenitore UNION SELECT Tipo FROM Contenuto);
6 );

```

La seguente asserzione garantisce che un contenitore per cui sia impostata la proprietà di contenimento arbitrario non figuri nella relazione di contenimento come contenitore.

Listing 8: CheckQualsiasiContenuto

```

1 CREATE ASSERTION CheckQualsiasiContenuto
2 CHECK NOT EXISTS (
3     SELECT *
4 FROM Contenitore ore JOIN Contenimento nto ON ore.Tipo = nto.Contenitore
5 WHERE ore.QualsiasiContenuto = TRUE;
6 );

```

La seguente asserzione garantisce che il contenuto di una istanza che sia di tipo contenitore sia nullo.

Listing 9: CheckContenutoIstanza

```

1 CREATE ASSERTION CheckContenutoIstanza
2 CHECK NOT EXISTS (
3     SELECT *
4 FROM Istanza_Elemento IE JOIN Contenitore CO ON IE.Tipo = CO.Tipo
5 WHERE IE.Contenuto IS NOT NULL;
6 );

```

La seguente asserzione garantisce che l'ordinamento tra le istanze rispetti quello definito per i rispettivi tipi.

Listing 10: CheckOrdinamento

```

1 CREATE ASSERTION CheckOrdinamento
2 CHECK NOT EXISTS (
3     SELECT *
4 FROM VOrd V1 JOIN VOrd V2 ON V1.Contenitore = V2.Contenitore
5 WHERE V1.Contenuto <> V2.Contenuto
6     AND V1.OrdineTipo < V2.OrdineTipo
7     AND V1.OrdineIstanza > V2.OrdineIstanza;
8 );

```

La seguente asserzione garantisce che l'istanza contenuta da un'istanza contenitore appartenga allo stesso documento di quest'ultima.

Listing 11: CheckDocumentoContenuto

```

1 CREATE ASSERTION CheckDocumentoContenuto
2 CHECK NOT EXISTS (
3     SELECT *
4 FROM Contenimento_Istanze CI JOIN Istanza_Elemento IE1 ON CI.Contenitore =
5     IE1.Id

```

```

5 JOIN Istanza_Elemento IE2 ON CI.Contenuto = IE2.Id
6 WHERE IE1.Documento <> IE2.Documento;
7 );

```

## 7 Esempio

Listing 12: Popolamento della BD con alcuni dati di prova

```

1 INSERT INTO Tipo_Elemento VALUES
2 (11, 'TipoLibro'),
3 (12, 'TipoCapitolo'),
4 (13, 'TipoSezione'),
5 (14, 'TipoSottosezione'),
6 (15, 'TipoTesto');
7
8 INSERT INTO Contenitore VALUES
9 (11, FALSE),
10 (12, FALSE),
11 (13, FALSE),
12 (14, FALSE);
13
14 INSERT INTO Contenuto VALUES (15, TRUE, NULL, NULL, FALSE, NULL, FALSE);
15
16 INSERT INTO Contenimento VALUES
17 (11, 12, NULL, NULL, 0),
18 (12, 13, NULL, NULL, 0),
19 (13, 14, NULL, NULL, 0),
20 (14, 15, NULL, NULL, 0);
21
22 INSERT INTO Tipo_Attributo VALUES (31, 'SaltaIndice', NULL, FALSE, 'Booleano',
23 12);
24
25 INSERT INTO Istanza_Elemento VALUES
26 (21, 'Libro', NULL, 11),
27 (22, 'Capitolo 1', NULL, 12),
28 (23, 'Sezione 1', NULL, 13),
29 (24, 'Sottosezione 1', NULL, 14),
30 (25, 'Testo', 'Testo vero e proprio', 15);
31
32 INSERT INTO Contenimento_Istanze VALUES
33 (21, 22, 0),
34 (22, 23, 0),
35 (23, 24, 0),
36 (24, 25, 0);
37
38 INSERT INTO Istanza_Attributo VALUES (41, 31, 22, 'Vero');
39
40 INSERT INTO Formato VALUES (51, 'Default');
41 INSERT INTO Regola VALUES (61, 'RegolaCapitolo', NULL, 12);
42 INSERT INTO Composizione_Regole VALUES (51, 61);
43 INSERT INTO Attributo_Regola VALUES (71, 'CapoLettera', 'Vero', 'Booleano');
44 INSERT INTO Composizione_AttrReg VALUES (61, 71);
45
46 INSERT INTO Tipo_Documento VALUES (81, 'TipoDocLibro', NULL, NULL, 11, 51);
47 INSERT INTO Documento VALUES (21, 'Libro', 'Anonimo', current_date, 81);

```