UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II
SCUOLA POLITECNICA E DELLE SCIENZE DI BASE
DIPARTIMENTO DI INGEGNERIA ELETTRICA E TECNOLOGIE DELL'INFORMAZIONE

CORSO DI LAUREA IN INFORMATICA
INSEGNAMENTO DI BASI DI DATI E SISTEMI INFORMATIVI I
ANNO ACCADEMICO 2018/2019

Progettazione e sviluppo di una base di dati relazionale per la descrizione e la memorizzazione di class diagram UML con supporto a tutte le relative caratteristiche

Autori:
Gennaro SORRENTINO
MATRICOLA N86/2351
gennaro.sorrentino5@studenti.unina.it
Gianluca L'ARCO
MATRICOLA N86/2799
g.larco@studenti.unina.it

Docenti:
Prof. Adriano PERON
Prof. Alessandro DE LUCA

26/06/2019

Indice

Capitolo 1

Descrizione del progetto

1.1 | Analisi del problema

Si provvederà alla progettazione ed allo sviluppo di una base di dati dedicata alla descrizione, memorizzazione e gestione di class diagram UML. La base di dati fornirà supporto a tutte le caratteristiche relative alla modellazione di class diagram UML, nello specifico:

- **Classi, Attributi e Metodi**

  Definizione 1.1: Una classe rappresenta la descrizione di un insieme di oggetti omogenei individuati da comportamenti e proprietà comuni.

  UML consente la specializzazione di una classe in:

  - Astratta: Una classe astratta possiede almeno un comportamento privo di implementazione, per tale motivo una classe con il suddetto tipo non può essere istanziata.
  - Associazione: Una classe di associazione fa parte di una relazione di associazione tra due o più classi. Una classe di questo tipo consente la specifica di ulteriori informazioni relative all'associazione stessa e che non sono quindi riconducibili univocamente alle classi coinvolte nell'associazione.
  - Parametrica: Una classe parametrica consente l'astrazione di uno o più tipi T. È possibile, all'interno di una classe parametrica, definire proprietà e comportamenti che utilizzino il/i tipo/i T definito/i dalla classe parametrica stessa.

  Le proprietà definite da una classe sono dette **attributi**. Un attributo modella una proprietà locale della classe ed è caratterizzato da un nome, un tipo e una molteplicità.

  I comportamenti invece sono detti **metodi**. Un metodo definisce le operazioni che si possono compiere sugli attributi della classe ed è caratterizzato anch'esso da un nome, una serie di parametri (se previsti) e un eventuale tipo restituito.

- **Interfacce**

Definizione 1.2: Un'interfaccia definisce una serie di comportamenti che altri elementi, quali ad esempio classi, devono implementare.

Si tenga presente che un'interfaccia definisce solo ed esclusivamente comportamenti (con visibilità public) ed è quindi sprovvista di proprietà. Infine la specifica è ristretta al tipo di funzionalità da implementare e non al come queste siano effettivamente implementate.

- **Associazioni**

Definizione 1.3: Un'associazione esprime una connessione logica tra classi.
Da un punto di vista matematico possiamo dire che un'associazione tra una classe C_1 ed una C_2 modella una relazione matematica tra l'insieme delle istanze di C_1 e l'insieme delle istanze di C_2.

Un'associazione può coinvolgere più di due classi e in tal caso si parla di associazioni N-arie (una tale associazione modella una relazione tra N insiemi). È possibile associare un nome ad un'associazione, e per quest'ultimo definire un certo verso di lettura, inoltre può essere utile aggiungere un'informazione che specifica il ruolo che una certa classe ricopre nell'associazione. Ogni classe che partecipa ad un'associazione lo fa con una certa molteplicità.

UML consente la specializzazione di un'associazione in:

- Composizione: La composizione è un tipo speciale di associazione binaria la quale sta ad indicare che un'oggetto di una determinata classe "è composto da":
  - Le componenti non possono esistere senza l'oggetto composto.
  - La proprietà delle componenti da parte del composto è esclusiva. Dunque una componente non può comporre più oggetti.
  - La molteplicità dal lato del composto deve essere 1, mentre può essere qualunque dal lato del componente.
- Aggregazione: Un'aggregazione è un tipo speciale di associazione binaria la quale sta ad indicare che un'oggetto di una determinata classe "è un insieme di":
  - A differenza della composizione gli aggreganti possono esistere senza l'aggregato.
  - La proprietà delle aggreganti da parte dell'aggregato non è più esclusiva. Un'aggregante può aggregare più oggetti.
  - La molteplicità dal lato dell'aggregato non è fissata ad 1.

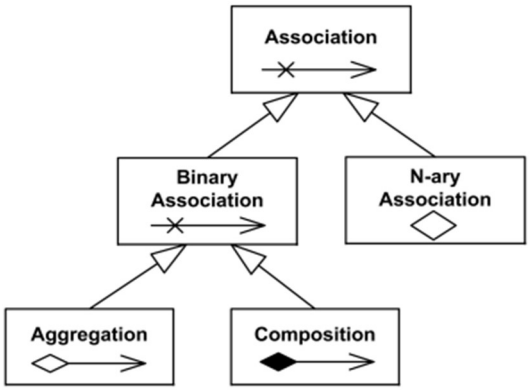

Immagine 1.1*:* - Gerarchia delle associazioni

Infine le associazioni binarie possono essere provviste di qualificatore. I qualificatori si rendono utili per identificare gli attributi coinvolti ai lati dell'associazione.

- **Generalizzazioni**

Definizione 1.4: Una relazione di generalizzazione indica che una delle due classi correlate (sottoclasse o classe specializzata) è assimilabile ad una forma specializzata dell'altra classe (superclasse o classe generale), viceversa la superclasse è considerata una generalizzazione della sottoclasse.

Dalla definizione di generalizzazione discende che ogni istanza di una sottoclasse è anche istanza della classe generale. Ogni sottoclasse eredità i comportamenti e le proprietà della superclasse e generalmente aumenta il grado di dettaglio di quest'ultima.

Una stessa superclasse può essere specializzata in più sottoclassi, in tal caso la relazione di generalizzazione può essere esclusiva oppure no. Nel primo caso ogni istanza della superclasse può essere istanza di una sola sottoclasse, mentre nel secondo ogni istanza della superclasse può essere istanza di più sottoclassi. Infine la generalizzazione può essere totale o parziale. Se totale ogni istanza della superclasse è anche istanza di una delle sottoclassi, viceversa, se parziale, ogni istanza della superclasse può essere istanza di una delle sottoclassi.

- **Realizzazioni e Dipendenze**

Definizione 1.5: Una realizzazione specifica una relazione tra due elementi in cui uno dei due elementi (talvolta detto client) realizza (implementa o esegue) il comportamento specificato dall'altro elemento del modello (il fornitore). Una relazione di realizzazione può essere definita tra un'interfaccia ed una classe, in tal caso la classe implementa il comportamento specificato dall'interfaccia.

Definizione 1.6: Una relazione di dipendenza specifica la dipendenza di un certo elemento nei confronti di un altro. Il significato di una dipendenza è espresso dallo stereotipo associato ad essa. UML fornisce la possibilità di impiegare stereotipi predefiniti oppure di definirne dei nuovi.

Alcuni degli stereotipi predefiniti di UML sono:

- <<call>>: Specifica che un certo elemento ne richiama un altro.
- <<create>>: Specifica che un certo elemento crea un'istanza di un altro.
- <<instantiate>>: Ha lo stesso significato del precedente.
- <<responsability>>: Indica che un certo elemento (dipendente) presenta un certo obbligo nei confronti di un altro elemento (superiore).
- <<send>>: Specifica che un certo elemento invia un "segnale" ad un altro.
- <<use>>: Un certo elemento richiede l'uso di un altro.

Infine la base di dati supporterà anche la gestione dei tipi e di conseguenza la definizione di tipi strutturati ed enumerazioni e la possibilità di impiegare, oltre ai due suddetti e al tipo primitivo, il tipo classe e il tipo parametrico. Si osservi che il tipo parametrico corrisponde al parametro definito in una classe parametrica, pertanto il suo "scope" è confinato in essa.

Capitolo 2

Progettazione concettuale

2.1 | Alcune convenzioni per la lettura dei Class Diagram

Al fine di semplificare la lettura dei class diagram UML che seguono si è scelto di adottare le seguenti convenzioni:

- Tutti gli attributi, ad eccezioni di quelli in cui specificata, hanno molteplicità pari ad [1].
- La molteplicità di una partecipazione è situata a destra se la linea di collegamento è verticale, viceversa in alto se la linea di collegamento è orizzontale.
- Le associazioni colorate sono state introdotte solo al fine di eliminare qualunque ambiguità in caso di intersezione tra le linee di collegamento delle associazioni.
- Le frecce ◄, ►, ▲, ▼ indicano esclusivamente il verso preferenziale di lettura, non forniscono alcuna informazione circa la navigabilità di un'associazione.
- Le enumerazioni sono identificabili dal colore verde.
- Il simbolo • in arancione, indica che verranno fornite informazioni aggiuntive, immediatamente dopo il Class Diagram, sull'elemento etichettato.
- Alcuni nomi di associazioni e ruoli sono stati omessi. La descrizione completa di questi è comunque riportata in seguito nel dizionario delle associazioni.

NOTA: Nel caso in cui i Class Diagram non risultassero essere abbastanza leggibili si riportano qui di seguito i link per reperire le versioni digitali:

- Class Diagram: https://ibb.co/G7nQL9m
- Class Diagram Ristrutturato: https://ibb.co/vXDv6C4

2.2 | Class Diagram

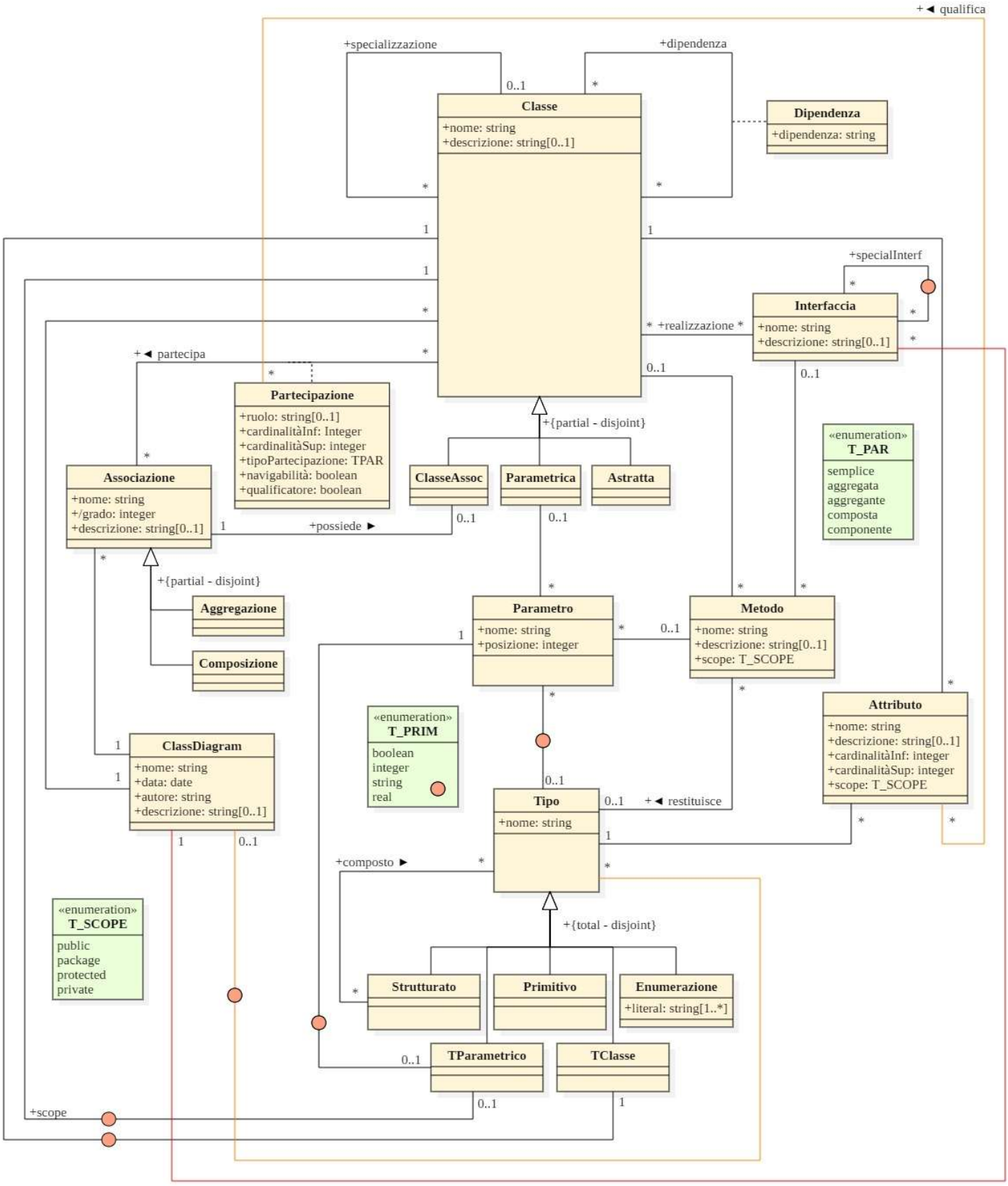

Immagine 2.1: - Class Diagram della base di dati

2.2.1 | Alcune precisazioni sul Class Diagram: [Parametro - Tipo] parziale

Pur essendo un argomento trattato in seguito nel dizionario delle associazioni, in questo paragrafo si spiegherà il perché l'associazione tra Parametro e Tipo è [*, 0..1], ovvero perché un parametro può non avere un tipo.

Consideriamo la seguente classe parametrica:

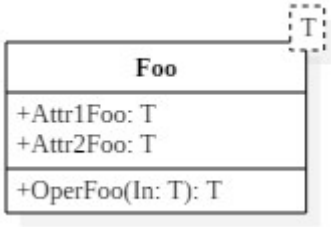

Immagine 2.2*:* - Una generica classe parametrica

Il parametro T, definito per la classe Foo, assume due diversi ruoli:

- Parametro per la classe Foo.
- Tipo per gli attributi e le operazioni in Foo.

Il primo caso è quello che rende la partecipazione di parametro, nell'associazione [Parametro - Tipo], parziale. Infatti, pur essendo un parametro, T non presenta alcun tipo, al contrario ne definisce uno nuovo (parametrico) utilizzabile solo all'interno della classe stessa Foo.

2.2.2 | Alcune precisazioni sul Class Diagram: T_PRIM e [CD - Tipo]

UML consente la definizione di tipi primitivi personali. Per tale motivo, quando un nuovo tipo primitivo viene definito, il dominio del suo nome non è limitato a TPRIM. Tale enumerazione è stata solo inserita al fine di indicare che questi tipi primitivi (*boolean*, *integer*, *string*, *real*) saranno già presenti nella base di dati. Inoltre, essendo tipi predefiniti, non appartengono a nessun Class Diagram e ciò spiega anche il perché l'associazione [ClassDiagram - Tipo] risulta essere parziale.

2.2.3 | Alcune precisazioni sul Class Diagram: SpecialInterf N a N

A differenza di quanto accade con le classi, dove una classe specializzata ha uno ed un solo padre, con le interfacce ciò non è vero. Infatti è possibile che una stessa interfaccia ne specializzi più di una.

2.2.4 | Alcune precisazioni sul Class Diagram: [TClasse - Classe]

Quando viene definito una nuova classe questa può essere utilizzata come tipo all'interno del Class Diagram di appartenenza. Per tale motivo ogni qualvolta che viene creata una classe viene anche creato il tipo "Classe" associato. L'associazione

[TClasse - Classe] sta proprio ad indicare a quale classe fa riferimento un certo tipo “Classe”.

2.2.5 | Alcune precisazioni sul Class Diagram: [TParametrico - Parametro]

Come già accennato nel paragrafo 2.2.1, un parametro di una classe parametrica non presenta tipo, ma al contrario ne definisce uno nuovo “Parametrico”. L’associazione [TParametrico - Parametro] sta proprio ad indicare a quale parametro fa riferimento un certo tipo “Parametrico”.

2.2.6 | Alcune precisazioni sul Class Diagram: Scope

Quando viene definito un nuovo tipo “Parametrico”, questo può essere impiegato solo nella classe del parametro da cui è stato definito. L’associazione scope sta proprio ad indicare la classe in cui tale tipo “Parametrico” può essere utilizzato, tale classe, per quanto detto, sarà la stessa del parametro che ha definito il tipo “Parametrico”.

2.3 | Ristrutturazione

2.3.1 | Informazioni ridondanti

L'unica informazione ridondante presente è quella relativa al grado di un'associazione. È infatti possibile calcolare il grado di un'associazione contando il numero di classi che partecipano ad essa, ovvero, nel nostro caso, calcolare il numero delle istanze di Partecipazione correlate all'associazione di cui si vuole conoscere il grado.

2.3.2 | Attributi multipli

È necessario gestire l'attributo multiplo literal della classe Enumerazione. Non essendo possibile effettuare alcuna stima sul numero di literal che un'enumerazione conterrà si provvederà a creare una classe per i literal. Tale classe avrà un'associazione con Enumerazione del tipo [Enumerazione: 1..* - literal: 1] in quanto un'enumerazione ha uno o più literal, mentre un literal appartiene ad una sola enumerazione.

2.3.3 | Generalizzazioni

Per quanto concerne le generalizzazioni si è scelto, per tutte e tre le generalizzazioni, di accorpare le classi specializzate in quella generale, introducendo quindi un attributo discriminante. Il motivo di tale scelta è riconducibile a diversi fattori:

- La maggior parte delle classi specializzate è priva di attributi, quindi l'informazione aggiunta alla classe generale è minima.
- La generalizzazione di Classe e quella di Associazione sono parziali non è dunque possibile accorpare la generale nelle specializzate.
- Le classi generale hanno un numero significativo di associazioni correlate, accorpare la generalizzazione dall'alto verso il basso causerebbe una duplicazione di quest'ultime.
- Riduzione significativa della quantità delle classi.
- Facilità nel recupero delle informazioni.

Ovviamente tale scelta non è priva di conseguenze. L'introduzione di un attributo discriminante richiederà un controllo preliminare sui dati e dunque l'introduzione di vincoli aggiuntivi.

2.3.4 | Analisi degli identificativi

Per tutte le classi si è scelto di impiegare chiavi surrogate in modo da evitare l'impiego di chiavi primarie composte, come ad esempio (ID_ClassDiagram, Nome) per Classe. Tale scelta comporterà una riduzione dello sforzo computazionale richiesto nella ricerca di un determinato record.
Per convenzione le chiavi primarie inizieranno con il prefisso ID_.

2.3.5 | Class Diagram ristrutturato

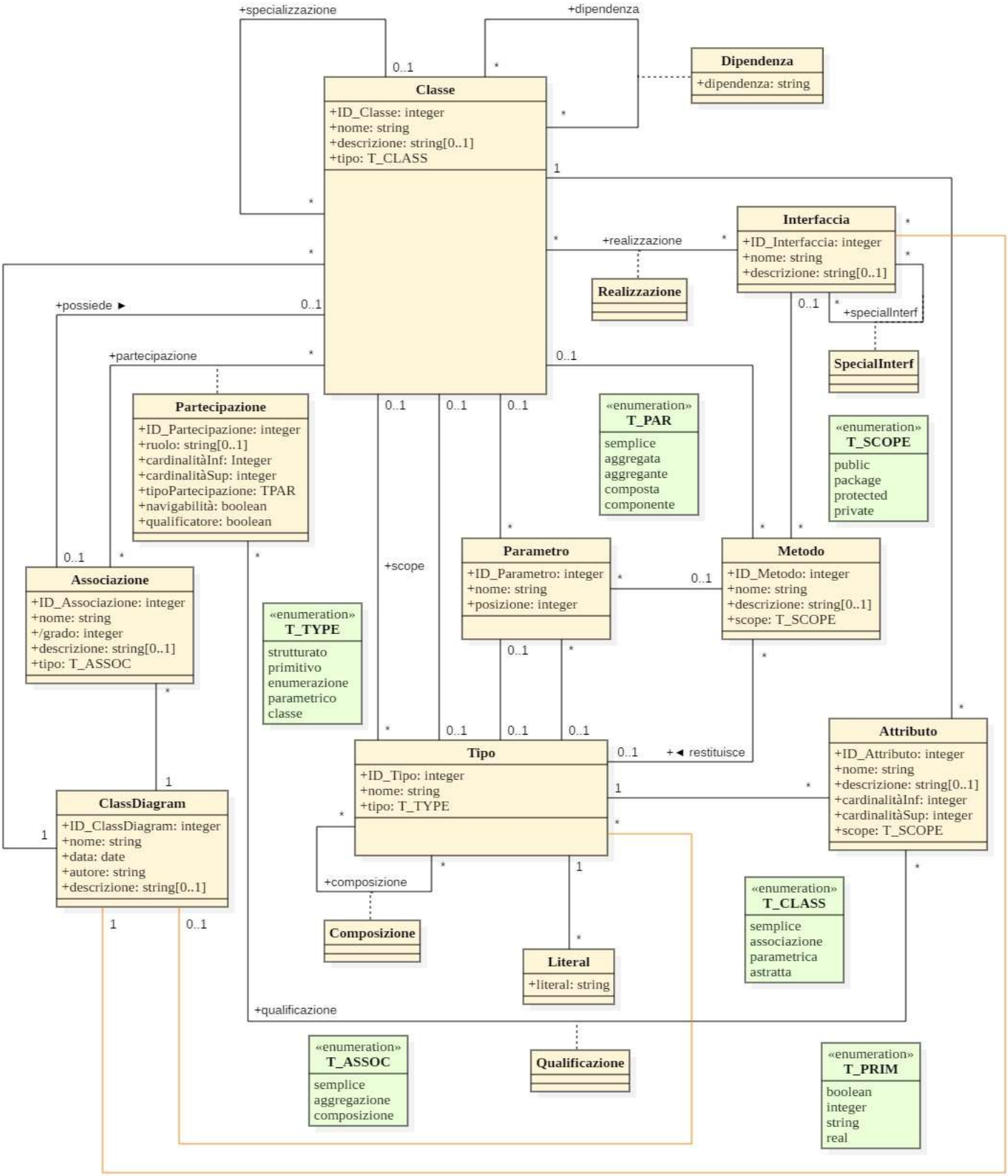

Immagine 2.1: - Class Diagram ristrutturato della base di dati

2.4 | Dizionari

2.4.1 | Dizionario delle classi

Classe	*Descrizione*	*Attributi*
ClassDiagram	Descrittore di un Class Diagram UML.	**ID_ClassDiagram** (*integer*): Chiave tecnica. Identifica univocamente ciascuna istanza di ClassDiagram. **Nome** (*string*): Nome associato al Class Diagram. **Autore** (*string*): Nome dell'autore del Class Diagram. **Data** (*date*): Data della creazione del Class Diagram. **Descrizione** (*string*): Descrizione facoltativa del Class Diagram.
Classe	Descrittore di una classe appartenente ad un Class Diagram.	**ID_Classe** (*integer*): Chiave tecnica. Identifica univocamente ciascuna istanza di Classe. **Nome** (*string*): Nome associato alla classe. **Descrizione** (*string*): Descrizione facoltativa della classe. **Tipo** (*T_CLASS*): Indica il tipo della classe.
Dipendenza	Descrittore di una dipendenza tra due classi.	**Dipendenza** (*string*): Indica il tipo di dipendenza.
Attributo	Descrittore di un attributo appartenente ad una classe.	**ID_Attributo** (*integer*): Chiave tecnica. Identifica univocamente ciascuna istanza di Attributo. **Nome** (*string*): Nome associato all'attributo. **Descrizione** (*string*): Descrizione facoltativa dell'attributo. **CardinalitàInf** (*integer*): Indica l'estremo inferiore della cardinalità dell'attributo. **CardinalitàSup** (*integer*): Indica l'estremo superiore della cardinalità dell'attributo. **Scope** (*T_SCOPE*): Indica lo scope dell'attributo.

Tabella 2.1*:* - Dizionario delle classi – cont.

Tabella 2.1: - Dizionario delle classi – prec.

Classe	*Descrizione*	*Attributi*
Metodo	Descrittore di un metodo di appartenente ad una classe.	**ID_Metodo** (*integer*): Chiave tecnica. Identifica univocamente ciascuna istanza di Metodo. **Nome** (*string*): Nome associato al metodo. **Descrizione** (*string*): Descrizione facoltativa del metodo. **Scope** (*T_SCOPE*): Indica lo scope del metodo.
Parametro	Descrittore di un parametro di un metodo oppure di una classe 'Parametrica'.	**ID_Parametro** (*integer*): Chiave tecnica. Identifica univocamente ciascuna istanza di Parametro. **Nome** (*string*): Nome associato al parametro. **Posizione** (*integer*): Indica la posizione del parametro all'interno della segnatura di un metodo oppure nella definizione di una classe 'Parametrica'.
Tipo	Descrittore di un tipo.	**ID_Tipo** (*integer*): Chiave tecnica. Identifica univocamente ciascuna istanza di Tipo. **Nome** (*string*): Nome del tipo. **Tipo** (*T_TYPE*): Identifica il genere del tipo (Primitivo, Strutturato, Enumerazione, etc.).
Composizione	Descrittore della composizione di un tipo.	
Qualificazione	Descrittore di una qualificazione di una Partecipazione che prevede qualificatore.	
Realizzazione	Descrittore di una realizzazione di un'interfaccia.	
Literal	Descrittore di un literal di un tipo 'Enumerazione'.	**Literal** (*string*): Indica un certo "valore" fisso.

Tabella 2.1: - Dizionario delle classi – cont.

Tabella 2.1: - Dizionario delle classi – prec.

Classe	*Descrizione*	*Attributi*
Partecipazione	Descrittore di una partecipazione ad un'associazione.	**ID_Partecipazione** (*integer*): Chiave tecnica Identifica univocamente ciascuna istanza di Partecipazione. **Ruolo** (*string*): Indentifica il ruolo con cui la classe coinvolta partecipa. **CardinalitàInf** (*integer*): Indica l'estremo inferiore della cardinalità di partecipazione. **CardinalitàSup** (*integer*): Indica l'estremo superiore della cardinalità di partecipazione. **TipoPartecipazione** (*T_PAR*): Indica la posizione intrapresa dalla classe nell'associazione (Aggregata, Componente, etc.). **Navigabilità**: Indica se l'associazione è navigabile verso la partecipazione. **Qualificatore** (*boolean*): Indica se la partecipazione possiede o meno un qualificatore.
Associazione	Descrittore di un'associazione appartenente ad un Class Diagram.	**ID_Associazione** (*integer*): Chiave tecnica. Identifica univocamente ciascuna istanza di Associazione. **Nome** (*string*): Nome dell'associazione. **/Grado** (*integer*): Indica il numero di classi coinvolte nell'associazione (Calcolato). **Descrizione** (*string*): Descrizione facoltativa dell'associazione. **Tipo** (*T_ASSOC*): Indica il tipo di un'associazione.
Interfaccia	Descrittore di un'interfaccia appartenente ad un Class Diagram.	**ID_Interfaccia** (*integer*): Chiave tecnica. Identifica univocamente ciascuna istanza di Interfaccia. **Nome** (*string*): Nome dell'interfaccia. **Descrizione** (*string*): Descrizione facoltativa dell'interfaccia.
SpecialInterf	Descrittore di una specializzazione tra due interfacce.	

Tabella 2.1: - Dizionario delle classi – fine.

2.4.2 | Dizionario delle associazioni

Associazione	*Descrizione*	*Classi coinvolte*
Specializzazione	Esprime una relazione gerarchica che intercorre tra due classi.	**Classe [0..1]** *ruolo* (**generale**): Indica la classe generalizzata. **Classe [0..*]** *ruolo* (**specializzata**): Indica le classi che specializzano una classe.
Dipendenza	Esprime una relazione di dipendenza che intercorre tra due classi.	**Classe [0..*]** *ruolo* (**superiore**): Indica le classi da cui ne dipendono altre. **Classe [0..*]** *ruolo* (**dipendente**): Indica le classi che dipendono da altre.
Realizzazione	Esprime l'implementazione di un'interfaccia da parte di una classe.	**Classe [0..*]** *ruolo* (**implementa**): Indica le classi che implementano le interfacce. **Interfaccia [0..*]** *ruolo* (**implementate**): Indica le interfacce implementate.
SpecialInterf	Esprime una relazione gerarchica che intercorre tra due interfacce.	**Interfaccia [0..*]** *ruolo* (**generale**): Indica le interfacce generalizzate. **Interfaccia [0..*]** *ruolo* (**specializzata**): Indica le interfacce che ne specializzano altre.
Possiede	Esprime l'appartenenza di una certa classe di associazione ad un'associazione.	**Classe [0..1]** *ruolo* (**è posseduta**): Indica la classe (di tipo "Associazione") posseduta da un'associazione. **Associazione [0..1]** *ruolo* (**possiede**): Indica l'associazione che possiede la classe di associazione.
Partecipazione	Esprime la partecipazione di una classe ad un'associazione.	**Classe [0..*]** *ruolo* (**partecipa**): Indica le classi che partecipano all'associazione. **Associazione [0..*]** *ruolo* (**partecipano**): Indica le associazioni a cui partecipano le classi.
Restituisce	Esprime la possibilità di un metodo di restituire un tipo.	**Metodo [0..1]** *ruolo* (**restituisce**): Indica il metodo che restituisce un certo tipo. **Tipo [0..*]** *ruolo* (**restituito**): Indica il tipo restituito da uno o più metodi.
Scope	Esprime l'ambiente in cui può essere utilizzato un certo tipo parametrico.	**Classe [0..*]** *ruolo* (**ambiente**): Indica la classe in cui può essere utilizzato il tipo parametrico. **Tipo [0..1]** *ruolo* (**utilizzabili**): Indica i tipi ("Parametrici") che possono essere utilizzati nella classe ambiente.

Tabella 2.2*:* - Dizionario delle associazioni – cont.

Tabella 2.2*:* - Dizionario delle associazioni – prec.

Associazione	*Descrizione*	*Classi coinvolte*
Composizione	Esprime la composizione di un tipo strutturato.	**Tipo [0..*]** *ruolo* (**composto**): Indica i tipi strutturati. **Tipo [0..*]** *ruolo* (**compone**): Indica i tipi che ne compongono altri ("Strutturati").
Qualificazione	Esprime la qualifica di un attributo nei confronti di una partecipazione che ammette qualificatore.	**Partecipazione [0..*]** *ruolo* (**qualificata**): Indica le partecipazioni qualificate. **Attributo [0..*]** *ruolo* (**qualificano**): Indica gli attributi che qualificano le partecipazioni.
Ha	Esprime l'appartenenza di un attributo ad una classe.	**Classe [0..*]** *ruolo* (**ha**): Indica la classe che possiede gli attributi. **Attributo [1]** *ruolo* (**di**): Indica gli attributi che appartengono ad una classe.
Contiene Classe	Esprime l'appartenenza di una classe ad un Class Diagram.	**Classe [1]** *ruolo* (**di**): Indica le classi che appartengono ad un Class Diagram. **Class Diagram [0..*]** *ruolo* (**ha**): Indica il Class Diagram a cui appartengono le classi.
Contiene Associazione	Esprime l'appartenenza di un'associazione ad un Class Diagram.	**Associazione [1]** *ruolo* (**di**): Indica le associazioni che appartengono ad un Class Diagram. **Class Diagram [0..*]** *ruolo* (ha**):** Indica il Class Diagram a cui appartengono le associazioni.
Contiene Tipo	Esprime l'appartenenza di un tipo ad un Class Diagram.	**Tipo [0..1]** *ruolo* (**di**): Indica i tipi che appartengono ad un Class Diagram (i tipi primitivi *T_PRIM* non appartengono ad alcun Class Diagram per questo la partecipazione risulta essere parziale). **Class Diagram [0..*]** *ruolo* (**ha**): Indica il Class Diagram a cui appartengono i tipi.
Contiene Interfaccia	Esprime l'appartenenza di un'interfaccia ad un Class Diagram.	**Interfaccia [1]** *ruolo* (**di**): Indica le interfacce che appartengono ad un Class Diagram. **Class Diagram [0..*]** *ruolo* (**ha**): Indica il Class Diagram a cui appartengono le interfacce.

Tabella 2.2*:* - Dizionario delle associazioni – cont.

Tabella 2.2*:* - Dizionario delle associazioni – prec.

Associazione	*Descrizione*	*Classi coinvolte*
Classe dispone	Esprime l'appartenenza di un metodo ad una classe.	**Classe [0..*]** *ruolo* (**ha**): Indica la classe a cui appartengono i metodi. **Metodo [0..1]** *ruolo* (**di**): Indica i metodi che appartengono ad una classe (la partecipazione è parziale dal momento che un metodo può appartenere ad un'interfaccia).
Interfaccia dispone	Esprime l'appartenenza di un metodo ad un'interfaccia.	**Interfaccia [0..*]** *ruolo* (**ha**): Indica l'interfaccia a cui appartengono i metodi. **Metodo [0..1]** *ruolo* (**di**): Indica i metodi che appartengono ad un'interfaccia (la partecipazione è parziale dal momento che un metodo può appartenere ad una classe).
Segnatura	Esprime l'appartenenza di un parametro ad un metodo.	**Metodo [0..*]** *ruolo* (**ha**): Indica il metodo a cui appartengono i parametri. **Parametro [0..1]** *ruolo* (**di**): Indica i parametri che appartengono ad un metodo (la partecipazione è parziale dal momento che un parametro può appartenere ad una classe "Parametrica").
Template	Esprime l'appartenenza di un parametro ad una classe "Parametrica".	**Classe [0..*]** *ruolo* (**ha**): Indica la classe ("Parametrica") a cui appartengono i parametri. **Parametro [0..1]** *ruolo* (**di**): Indica i parametri che appartengono ad una classe "Parametrica" (la partecipazione è parziale dal momento che un parametro può appartenere ad un metodo).
Attributo è	Esprime il tipo di un attributo.	**Attributo [1]** *ruolo* (**è**): Indica gli attributi. **Tipo [0..*]** *ruolo* (**di**): Indica il tipo assunto dagli attributi.
Enum contiene	Esprime l'appartenenza di un literal da parte di un tipo "Enumerazione".	**Tipo [0..*]** *ruolo* (**ha**): Indica il tipo ("Enumerazione") a cui appartengono i literal. **Literal [1]** *ruolo* (**di**): Indica i literal che appartengono ad un tipo ("Enumerazione").

Tabella 2.2*:* - Dizionario delle associazioni – cont.

Tabella 2.2*:* - Dizionario delle associazioni – prec.

Associazione	*Descrizione*	*Classi coinvolte*
Parametro è	Esprime il tipo di un parametro.	**Parametro [0..1]** *ruolo* (**è**): Indica i parametri (la partecipazione è parziale dal momento che un parametro di una classe "Parametrica" non ha tipo). **Tipo [0..*]** *ruolo* (**di**): Indica il tipo assunto dai parametri.
Riferisce parametro	Esprime da quale parametro (di una classe "Parametrica") è stato definito un tipo "Parametrico".	**Parametro [0..1]** *ruolo* (**definisce**): Indica il parametro che definisce il tipo "Parametrico". **Tipo [0..1]** *ruolo* (**definito**): Indica il tipo "Parametrico" definito da un parametro.
Riferisce classe	Esprime da quale classe è stato definito un tipo "Classe".	**Classe [0..1]** *ruolo* (**definisce**): Indica la classe che definisce il tipo "Classe". **Tipo [0..1]** *ruolo* (**definito**): Indica il tipo "Classe" definito da una classe.

Tabella 2.2*:* - Dizionario delle associazioni – fine.

2.4.3 | Dizionario dei vincoli

Vincolo	*Tipo*	*Descrizione*
noCycleClass	Intrarelazionale	Non possono esistere relazioni gerarchiche cicliche tra classi.
noCycleInterface	Intrarelazionale	Non possono esistere relazioni gerarchiche cicliche tra interfacce.
isValidTemplate	Interrelazionale	L'associazione **template** ([Classe - Parametro]) risulta valida solo se il tipo della classe è "*Parametrica"*.
isValidScope	Interrelazionale	L'associazione **scope** ([Classe - Tipo]) risulta valida solo se il tipo della classe è "*Parametrica"* e il tipo è *"Parametrico".*
isValidRefParam	N-upla	L'associazione **riferisce parametro** [(Parametro - Tipo)] è valida solo quando il tipo è "*Parametrico*".
ParamNoType	Interrelazionale	Un parametro non possiede tipo solo se appartiene ad una classe di tipo "*Parametrica"*.
ParamBelongsTo	N-upla	Un parametro deve appartenere o ad una classe oppure ad un metodo, non ad entrambi.
defTypeClass	Interrelazionale	La definizione di una classe implica la definizione di un tipo 'Classe' avente il suo stesso nome.
defTypeParam	Interrelazionale	La definizione di un parametro per una classe di tipo "*Parametrica"* implica la definizione di un tipo "*Parametrico"* avente il suo stesso nome e il cui scope è uguale alla classe a cui appartiene il parametro.
Scope	Interrelazionale	Un metodo, un attributo e/o un parametro possono assumere un certo tipo parametrico X solo se lo scope di X è uguale alla classe a cui questi appartengono.
notTPRIM	N-upla	Se un tipo è primitivo allora il suo nome non deve appartenere a *TPRIM*. Ovviamente nell'inserimento dei tipi predefiniti tale vincolo verrà temporaneamente disabilitato.
isValidComp	Interrelazionale (Classe di associazione)	L'associazione **composizione** ([Tipo - Tipo]) risulta valida solo se il tipo composto è "*Strutturato"* e quello componente non è *"Parametrico".*

Tabella 2.3: - Dizionario dei vincoli – cont.

Tabella 2.3: - Dizionario dei vincoli – prec.

Vincolo	*Tipo*	*Descrizione*
isValidLiteral	Interrelazionale	L'associazione **tipo contiene** ([Tipo - Literal]) risulta valida solo se il tipo è "*Enumerazione"*.
isValidRefClass	N-upla	L'associazione **riferisce classe** ([Classe - Tipo]) risulta valida solo se il tipo è "*Classe*".
uniqueClassName	Intrarelazionale	Il nome di una classe deve essere univoco nel Class Diagram di appartenenza.
uniqueAssocName	Intrarelazionale	Il nome di un'associazione deve essere univoco nel Class Diagram di appartenenza.
uniqueAttrName	Intrarelazionale	Il nome di un attributo deve essere univoco nella classe di appartenenza.
uniqueCDName	Intrarelazionale	Il nome di un Class Diagram deve essere univoco.
uniqueQualif	Intrarelazionale	Un attributo può qualificare una partecipazione una ed una sola volta.
uniqueComposition	Intrarelazionale	Ogni tipo ne compone un altro al massimo una sola volta.
uniqueDependency	Intrarelazionale	Tra due classi intercorre una ed una sola volta un certo tipo di dipendenza.
uniqueTypeName	Intrarelazionale	Il nome di un tipo è univoco nel Class Diagram di appartenenza.
uniquePos	Intrarelazionale	La posizione di un parametro deve essere univoca nella segnatura del metodo o nel template della classe di appartenenza.
uniqueLiteral	Intrarelazionale	Ogni enumerazione non contiene più volte uno stesso literal.
noSelfSpecialClass	N-upla	Una classe non può generalizzare sé stessa.
noSelfSpecialInterf	N-upla	Un'interfaccia non può generalizzare sé stessa.
uniqueAssocClass	Intrarelazionale	Un'associazione può possedere solo una classe di tipo "*Associazione"*.

Tabella 2.3: - Dizionario dei vincoli – cont.

Tabella 2.3: - Dizionario dei vincoli – prec.

Vincolo	*Tipo*	*Descrizione*
cantQualif	Interrelazionale	Un'attributo non può qualificare una partecipazione che non prevede un qualificatore e/o che non appartiene alla classe che partecipa in Partecipazione.
checkInterval	N-upla	In Partecipazione e Attributo la cardinalità inferiore (cardinalitàInf) deve essere minore o uguale a quella superiore (cardinalitàSup). Inoltre devono essere entrambe maggiori o uguali a zero.
AggrIsBin	Interrelazionale	Un'associazione di tipo "*Aggregazione*" è binaria. Tale vincolo è Interrelazionale dal momento che l'attributo grado dipende dal numero di istanza di Partecipazione associate.
CompIsBin	Interrelazionale	Un'associazione di tipo "*Composizione"* è binaria. Tale vincolo è Interrelazionale dal momento che l'attributo grado dipende dal numero di istanza di Partecipazione associate.
checkPart	Interrelazionale	Il tipo di partecipazione delle due classi che partecipano ad un'associazione di tipo 'Composizione' deve essere l'uno componente l'altra composta o viceversa. Il tipo di partecipazione di due classi che partecipano ad un'associazione di tipo "*Aggregazione*" deve essere l'uno aggregante l'altro aggregata o viceversa. Infine il tipo di partecipazione di una qualunque altra associazione deve essere semplice.
compInterval	N-upla	Se il tipo di partecipazione è componente allora cardinalitàInf è minore o uguale a cardinalitàSup che è a sua volta uguale ad 1.
excComp	Intrarelazionale	Una classe non può essere componente di più classi.
qualifAssoc	Interrelazionale	Una partecipazione può essere qualificata solo se l'associazione in cui partecipa è binaria.

Tabella 2.3: - Dizionario dei vincoli – cont.

Tabella 2.3: - Dizionario dei vincoli – prec.

Vincolo	*Tipo*	*Descrizione*
delClas	Interrelazionale	La cancellazione di una classe che partecipa ad una o più associazioni binarie implica l'eliminazione di tali associazioni.
validAssoc	Interrelazionale	Ogni associazione coinvolge almeno due classi.
EnumerationType	Interrelazionale	Solo un attributo può assumere un tipo "*Enumerazione*".
MethIsPublic	N-upla	Se un metodo appartiene ad un'interfaccia allora il suo scope deve essere public.
uniqueRealization	Intrarelazionale	Una classe implementa un'interfaccia al massimo una sola volta.
uniqueSpecial	Intrarelazionale	Un'interfaccia ne generalizza un'altra al massimo una sola volta.
immutableType	Dominio	Il tipo di una classe, un'associazione, di un tipo o di una partecipazione non può mutare.
uniqueInterfName	Intrarelazionale	Il nome di un'interfaccia è univoco nel Class Diagram di appartenenza.
ValidDate	Dominio	La data di inserimento/creazione di un Class Diagram non deve essere maggiore di quella attuale.
MethOf	N-upla	Un metodo appartiene ad una classe o ad un'interfaccia, non ad entrambi.

Tabella 2.3: - Dizionario dei vincoli – cont.

Capitolo 3

Progettazione logica

3.1 | Schema logico

3.1.1 | Traduzione

ClassDiagram (ID_ClassDiagram, Nome, Autore, Data, Descrizione)

Classe (ID_Classe, Nome, Descrizione, Tipo, FK_ClassDiagram, FK_Associazione, FK_Generale)

Chiavi esterne: FK_ClassDiagram ↪ ClassDiagram.ID_ ClassDiagram
FK_Associazione ↪ Associazione.ID_Associazione
FK_Generale ↪ Classe.ID_Classe

Dipendenza (Dipendenza, FK_Superiore, FK_Dipendente)

Chiavi esterne: FK_Superiore ↪ Classe.ID_Classe
FK_Dipendente ↪ Classe.ID_Classe

Attributo (ID_Attributo, Nome, Descrizione, CardinalitàInf, CardinalitàSup, Scope, FK_Classe, FK_Tipo)

Chiavi esterne: FK_Classe ↪ Classe.ID_Classe
FK_Tipo ↪ Tipo.ID_Tipo

Metodo (ID_Metodo, Nome, Descrizione, Scope, FK_Classe, FK_Interfaccia, FK_Tipo)

Chiavi esterne: FK_Classe ↪ Classe.ID_Classe
FK_Interfaccia ↪ Interfaccia.ID_Interfaccia
FK_Tipo ↪ Tipo.ID_Tipo

Interfaccia (ID_Interfaccia, Nome, Descrizione, FK_ClassDiagram)

Chiavi esterne: FK_ClassDiagram ↪ ClassDiagram.ID_ ClassDiagram

Schema 3.1*: -* Traduzione in schema logico – cont.

Schema 3.1*:* - Traduzione in schema logico – prec.

Realizzazione (FK_Interfaccia, FK_Classe)

Chiavi esterne: FK_Interfaccia ↪ Interfaccia.ID_Interfaccia
FK_Classe ↪ Classe.ID_Classe

Tipo (ID_Tipo, Nome, Tipo, FK_Scope FK_ClassDiagram, FK_Parametro, FK_Classe)

Chiavi esterne: FK_Scope ↪ Classe.ID_ Classe
FK_ClassDiagram ↪ ClassDiagram.ID_ ClassDiagram
FK_Parametro ↪ Parametro.ID_Parametro
FK_Classe ↪ Classe.ID_Classe

SpecialInterf (FK_Generale, FK_Specializzata)

Chiavi esterne: FK_Generale ↪ Classe.ID_Classe
FK_Specializzata ↪ Classe.ID_Classe

Composizione (FK_Strutturato, FK_Tipo)

Chiavi esterne: FK_Strutturato ↪ Tipo.ID_Tipo
FK_Tipo ↪ Tipo.ID_Tipo

Literal (Literal, FK_Enumerazione)

Chiavi esterne: FK_Enumerazione ↪ Tipo.ID_Tipo

Associazione (ID_Associazione, Nome, Descrizione, Tipo, FK_ClassDiagram)

Chiavi esterne: FK_ClassDiagram ↪ ClassDiagram.ID_ ClassDiagram

Partecipazione (ID_Partecipazione, Ruolo, CardinalitàInf, CardinalitàSup, tipoPartecipazione, Navigabilità, Qualificatore, FK_Associazione, FK_Classe)

Chiavi esterne: FK_Associazione ↪ Associazione.ID_Associazione
FK_Classe ↪ Classe.ID_Classe

Schema 3.1*:* - Traduzione in schema logico – cont.

Schema 3.1*: -* Traduzione in schema logico – prec.

Qualificazione (FK_Partecipazione, FK_Attributo)

Chiavi esterne: FK_Partecipazione ↪ Partecipazione.ID_ Partecipazione
FK_Attributo ↪ Attributo.ID_ Attributo

Parametro (ID_Parametro, Nome, Posizione, FK_Classe, FK_Metodo, FK_Tipo)

Chiavi esterne: FK_Classe ↪ Classe.ID_Classe
FK_Metodo ↪ Metodo.ID_Metodo
FK_Tipo ↪ Tipo.ID_Tipo

Schema 3.1*: -* Traduzione in schema logico – fine.

3.1.2 | Traduzione delle associazioni

Associazione	*Implementazione*
Specializzazione	Chiave esterna in Classe ↪ Classe
Dipendenza ►	Chiave esterna in Dipendenza ↪ Classe
Dipendenza ◄	Chiave esterna in Dipendenza ↪ Classe
Realizzazione ►	Chiave esterna in Realizzazione ↪ Classe
Realizzazione ◄	Chiave esterna in Realizzazione ↪ Interfaccia
SpecialInterf ►	Chiave esterna in SpecialInterf ↪ Interfaccia
SpecialInterf ◄	Chiave esterna in SpecialInterf ↪ Interfaccia
Possiede	Chiave esterna in Classe ↪ Associazione
Partecipazione ►	Chiave esterna in Partecipazione ↪ Classe
Partecipazione ◄	Chiave esterna in Partecipazione ↪ Associazione
Restituisce	Chiave esterna in Metodo ↪ Tipo
Scope	Chiave esterna in Tipo ↪ Classe
Composizione ►	Chiave esterna in Composizione ↪ Tipo
Composizione ◄	Chiave esterna in Composizione ↪ Tipo
Qualificazione ►	Chiave esterna in Qualificazione ↪ Partecipazione
Qualificazione ◄	Chiave esterna in Qualificazione ↪ Attributo
Ha	Chiave esterna in Attributo ↪ Classe
Contiene Classe	Chiave esterna in Classe ↪ Class Diagram
Contiene Assoc.	Chiave esterna in Associazione ↪ Class Diagram
Contiene Tipo	Chiave esterna in Tipo ↪ Class Diagram

Tabella 3.1*: -* Traduzione delle associazioni – cont.

Tabella 3.1*:* - Traduzione delle associazioni – prec.

Associazione	*Implementazione*
Enum contiene	Chiave esterna in Literal ↪ Tipo
Template	Chiave esterna in Parametro ↪ Classe
Contiene Interf.	Chiave esterna in Interfaccia ↪ Class Diagram
Attributo è	Chiave esterna in Attributo ↪ Tipo
Classe dispone	Chiave esterna in Metodo ↪ Classe
Interf. dispone	Chiave esterna in Metodo ↪ Interfaccia
Segnatura	Chiave esterna in Parametro ↪ Metodo
Parametro è	Chiave esterna in Parametro ↪ Tipo
Riferisce param.	Chiave esterna in Tipo ↪ Parametro
Riferisce classe	Chiave esterna in Tipo ↪ Classe

Tabella 3.1*:* - Traduzione delle associazioni – fine.

3.1.3 | Schema logico

ClassDiagram	(ID_ClassDiagram, Nome, Autore, Data, Descrizione)
Classe	(ID_Classe, Nome, Descrizione, Tipo, FK_ClassDiagram, FK_Associazione, FK_Generale)
Dipendenza	(Dipendenza, FK_Superiore, FK_Dipendente)
Attributo	(ID_Attributo, Nome, Descrizione, CardinalitàInf, CardinalitàSup, Scope, FK_Classe, FK_Tipo)
Metodo	(ID_Metodo, Nome, Descrizione, Scope, FK_Classe, FK_Interfaccia, FK_Tipo)
Parametro	(ID_Parametro, Nome, Posizione, FK_Classe, FK_Metodo, FK_Tipo)
Interfaccia	(ID_Interfaccia, Nome, Descrizione, FK_ClassDiagram)
Tipo	(ID_Tipo, Nome, Tipo, FK_Scope FK_ClassDiagram, FK_Parametro, FK_Classe)
SpecialInterf	(FK_Generale, FK_Specializzata)
Composizione	(FK_Strutturato, FK_Tipo)
Literal	(Literal, FK_Enumerazione)
Associazione	(ID_Associazione, Nome, Descrizione, Tipo, FK_ClassDiagram)
Partecipazione	(ID_Partecipazione, Ruolo, CardinalitàInf, CardinalitàSup, tipoPartecipazione, Navigabilità, Qualificatore, FK_Associazione, FK_Classe)
Qualificazione	(FK_Partecipazione, FK_Attributo)
Realizzazione	(FK_Interfaccia, FK_Classe)

Tabella 3.2: - Schema logico – fine.

Capitolo 4

Definizioni SQL

4.1 | Definizioni delle tabelle

In questo paragrafo si forniranno le definizioni SQL delle tabelle indicate dallo schema logico, inoltre verranno anche fornite le implementazioni dei vincoli più semplici.
Si osservi che ad ogni tabella prevista di chiave primaria sarà associato un trigger che provvederà a fornire un valore valido a quest'ultima nel caso in cui il suo valore risultasse essere uguale a NULL.

4.1.1 | Definizione della tabella CLASSDIAGRAM

```
-- TABLE: CLASSDIAGRAM
CREATE TABLE CLASSDIAGRAM(
    ID_ClassDiagram     INTEGER,
    Nome                VARCHAR2(50)    NOT NULL,
    Autore              VARCHAR2(50)    DEFAULT ON NULL 'Guest',
    C_Data              DATE            DEFAULT ON NULL SYSDATE,
    Descrizione         VARCHAR2(300)   DEFAULT ON NULL 'no description'
);


-- GESTIONE PRIMARY KEY [NULL]
CREATE OR REPLACE TRIGGER SET_ID_CLASSDIAGRAM
    BEFORE INSERT ON CLASSDIAGRAM
    FOR EACH ROW
    DECLARE
        var_ID_ClassDiagram CLASSDIAGRAM.ID_ClassDiagram%TYPE;
    BEGIN
        IF (:NEW.ID_ClassDiagram IS NULL) THEN
            SELECT NVL(MAX(C.ID_ClassDiagram), 0) + 1 INTO var_ID_ClassDiagram
            FROM CLASSDIAGRAM C;
            :NEW.ID_ClassDiagram := var_ID_ClassDiagram;
        END IF;
    END;
    /


-- CONSTRAINTS: CLASSDIAGRAM
ALTER TABLE CLASSDIAGRAM
    ADD(
        -- Chiave primaria
        CONSTRAINT PK_CLASSDIAGRAM PRIMARY KEY (ID_ClassDiagram),
        -- Vincolo UNIQUE_CD_NAME
        CONSTRAINT UNIQUE_CD_NAME UNIQUE (Nome)
    );
```

Codice 4.1.1: -Definizione della tabella CLASSDIAGRAM – fine.

4.1.2 | Definizione della tabella ASSOCIAZIONE

```
-- TABLE: ASSOCIAZIONE
CREATE TABLE ASSOCIAZIONE(
    ID_Associazione     INTEGER,
    Nome                VARCHAR2(50),
    Descrizione         VARCHAR2(300)   DEFAULT ON NULL 'no description',
    Tipo                VARCHAR2(50)    NOT NULL,
    FK_ClassDiagram     INTEGER         NOT NULL
);


-- GESTIONE PRIMARY KEY [NULL]
CREATE OR REPLACE TRIGGER SET_ID_ASSOCIAZIONE
    BEFORE INSERT ON ASSOCIAZIONE
    FOR EACH ROW
    DECLARE
        var_ID_Associazione ASSOCIAZIONE.ID_Associazione%TYPE;
    BEGIN
        IF (:NEW.ID_Associazione IS NULL) THEN
            SELECT NVL(MAX(A.ID_Associazione), 0) + 1 INTO var_ID_Associazione
            FROM ASSOCIAZIONE A;
            :NEW.ID_Associazione := var_ID_Associazione;
        END IF;
    END;
    /


-- CONSTRAINTS: ASSOCIAZIONE
ALTER TABLE ASSOCIAZIONE
    ADD(
        -- Chiave primaria
        CONSTRAINT PK_ASSOCIAZIONE PRIMARY KEY (ID_Associazione),
        -- Chiave esterna
        CONSTRAINT ASSOC_FK_CLASSDIAGRAM FOREIGN KEY (FK_ClassDiagram)
            REFERENCES CLASSDIAGRAM(ID_ClassDiagram) ON DELETE CASCADE,
        -- Vincolo T_ASSOC
        CONSTRAINT T_ASSOC CHECK (Tipo IN ('semplice', 'aggregazione', 'composizione'))
    );
```

Codice 4.1.2*:* -Definizione della tabella ASSOCIAZIONE – fine.

4.1.3 | Definizione della tabella CLASSE

```
-- TABLE: CLASSE
CREATE TABLE CLASSE(
   ID_Classe           INTEGER,
   Nome                VARCHAR2(50)    NOT NULL,
   Descrizione         VARCHAR2(300)   DEFAULT ON NULL 'no description',
   Tipo                VARCHAR2(20)    NOT NULL,
   FK_ClassDiagram     INTEGER         NOT NULL,
   FK_Associazione     INTEGER,
   FK_Generale         INTEGER
);



-- GESTIONE PRIMARY KEY [NULL]
CREATE OR REPLACE TRIGGER SET_ID_CLASSE
   BEFORE INSERT ON CLASSE
   FOR EACH ROW
   DECLARE
       var_ID_Classe CLASSE.ID_Classe%TYPE;
   BEGIN
       IF (:NEW.ID_Classe IS NULL) THEN
           SELECT NVL(MAX(C.ID_Classe), 0) + 1 INTO var_ID_Classe
           FROM CLASSE C;
           :NEW.ID_Classe := var_ID_Classe;
       END IF;
   END;
   /
```

Codice 4.1.3*:* -Definizione della tabella CLASSE – cont.

```
-- CONSTRAINTS: CLASSE
ALTER TABLE CLASSE
   ADD(
      -- Chiave primaria
      CONSTRAINT PK_CLASSE PRIMARY KEY (ID_Classe),
      -- Chiave esterna
      CONSTRAINT CLASSE_FK_CLASSDIAGRAM FOREIGN KEY (FK_ClassDiagram)
          REFERENCES CLASSDIAGRAM(ID_ClassDiagram) ON DELETE CASCADE,
      -- Chiave esterna
      CONSTRAINT CLASSE_FK_ASSOCIAZIONE FOREIGN KEY (FK_Associazione)
          REFERENCES ASSOCIAZIONE(ID_Associazione) ON DELETE CASCADE,
      -- Chiave esterna
      CONSTRAINT CLASSE_FK_CLASSE FOREIGN KEY (FK_Generale)
          REFERENCES CLASSE(ID_Classe) ON DELETE SET NULL,
      -- Vincolo T_CLASS
      CONSTRAINT T_CLASS CHECK (Tipo IN ('semplice', 'associazione',
                                         'parametrica', 'astratta')),
      -- Vincolo UNIQUE_CLASS_NAME
      CONSTRAINT UNIQUE_CLASS_NAME UNIQUE (FK_ClassDiagram, Nome),
      -- Vincolo NO_SELF_SPECIAL_CLASS
      CONSTRAINT NO_SELF_SPECIAL_CLASS CHECK (ID_Classe <> FK_Generale),
      -- Vincolo UNIQUE_ASSOC_CLASS
      CONSTRAINT UNIQUE_ASSOC_CLASS UNIQUE (FK_Associazione),
      -- Vincolo IS_VALID_ASSOC_CLASS
      CONSTRAINT IS_VALID_ASSOC_CLASS CHECK ((FK_Associazione IS NULL AND tipo
                                              <> 'associazione') OR
                                             (FK_Associazione IS NOT NULL
                                              AND tipo = 'associazione'))
   );
```

Codice 4.1.3: -Definizione della tabella CLASSE – fine.

4.1.4 | Definizione della tabella INTERFACCIA

```
-- TABLE: INTERFACCIA
CREATE TABLE INTERFACCIA(
   ID_Interfaccia      INTEGER,
   Nome                VARCHAR2(50)    NOT NULL,
   Descrizione         VARCHAR2(300)   DEFAULT ON NULL 'no description',
   FK_ClassDiagram     INTEGER         NOT NULL
);



-- GESTIONE PRIMARY KEY [NULL]
CREATE OR REPLACE TRIGGER SET_ID_INTERFACCIA
   BEFORE INSERT ON INTERFACCIA
   FOR EACH ROW
   DECLARE
       var_ID_Interfaccia INTERFACCIA.ID_Interfaccia%TYPE;
   BEGIN
       IF (:NEW.ID_Interfaccia IS NULL) THEN
           SELECT NVL(MAX(I.ID_Interfaccia), 0) + 1 INTO var_ID_Interfaccia
           FROM INTERFACCIA I;
           :NEW.ID_Interfaccia := var_ID_Interfaccia;
       END IF;
   END;
   /



-- CONSTRAINTS: INTERFACCIA
ALTER TABLE INTERFACCIA
   ADD(
       -- Chiave primaria
       CONSTRAINT PK_INTERFACCIA PRIMARY KEY (ID_Interfaccia),
        -- Chiave esterna
       CONSTRAINT INTERF_FK_CLASSDIAGRAM FOREIGN KEY (FK_ClassDiagram)
           REFERENCES CLASSDIAGRAM(ID_ClassDiagram) ON DELETE CASCADE,
       -- Vincolo UNIQUE_INTERF_NAME
       CONSTRAINT UNIQUE_INTERF_NAME UNIQUE (Nome, FK_ClassDiagram)
   );
```

Codice 4.1.4: -Definizione della tabella INTERFACCIA – fine.

4.1.5 | Definizione della tabella DIPENDENZA

```
-- TABLE: DIPENDENZA
CREATE TABLE DIPENDENZA(
   Dipendenza          VARCHAR2(50)     NOT NULL,
   FK_Superiore        INTEGER          NOT NULL,
   FK_Dipendente       INTEGER          NOT NULL
);


 -- CONSTRAINTS: DIPENDENZA
ALTER TABLE DIPENDENZA
   ADD(
       -- Chiave esterna
       CONSTRAINT DIPEND_FK_SUPERIORE FOREIGN KEY (FK_Superiore)
           REFERENCES CLASSE(ID_Classe) ON DELETE CASCADE,
       -- Chiave esterna
       CONSTRAINT DIPEND_FK_DIPENDENTE FOREIGN KEY (FK_Dipendente)
           REFERENCES CLASSE(ID_Classe) ON DELETE CASCADE,
       -- Vincolo UNIQUE_DEPENDENCY
       CONSTRAINT UNIQUE_DEPENDENCY UNIQUE (Dipendenza, FK_Superiore, FK_Dipendente)
   );
```

Codice 4.1.5*:* -Definizione della tabella DIPENDENZA – fine.

4.1.6 | Definizione della tabella SPECIAL_INTERF

```
-- TABLE: SPECIAL_INTERF
CREATE TABLE SPECIAL_INTERF(
   FK_Generale         INTEGER          NOT NULL,
   FK_Specializzata    INTEGER          NOT NULL
);


 -- CONSTRAINTS: SPECIAL_INTERF
ALTER TABLE SPECIAL_INTERF
   ADD(
       -- Chiave esterna
       CONSTRAINT INTERF_FK_GENERALE FOREIGN KEY (FK_Generale)
           REFERENCES INTERFACCIA(ID_Interfaccia) ON DELETE CASCADE,
       -- Chiave esterna
       CONSTRAINT INTERF_FK_SPECIALIZZATA FOREIGN KEY (FK_Specializzata)
           REFERENCES INTERFACCIA(ID_Interfaccia) ON DELETE CASCADE,
       -- Vincolo UNIQUE_SPECIAL
       CONSTRAINT UNIQUE_SPECIAL UNIQUE (FK_Generale, FK_Specializzata),
       -- Vincolo NO_SELF_SPECIAL_INTERF
       CONSTRAINT NO_SELF_SPECIAL_INTERF CHECK (FK_Generale <> FK_Specializzata)
   );
```

Codice 4.1.6*:* -Definizione della tabella SPECIAL_INTEF – fine.

4.1.7 | Definizione della tabella REALIZZAZIONE

```
-- TABLE: REALIZZAZIONE
CREATE TABLE REALIZZAZIONE(
   FK_Interfaccia      INTEGER         NOT NULL,
   FK_Classe           INTEGER         NOT NULL
);



 -- CONSTRAINTS: REALIZZAZIONE
ALTER TABLE REALIZZAZIONE
   ADD(
      -- Chiave esterna
      CONSTRAINT REALIZ_FK_INTERF FOREIGN KEY (FK_Interfaccia)
         REFERENCES INTERFACCIA(ID_Interfaccia) ON DELETE CASCADE,
      -- Chiave esterna
      CONSTRAINT REALIZ_FK_CLASSE FOREIGN KEY (FK_Classe)
         REFERENCES CLASSE(ID_Classe) ON DELETE CASCADE,
      -- Vincolo UNIQUE_REALIZATION
      CONSTRAINT UNIQUE_REALIZATION UNIQUE (FK_Interfaccia, FK_Classe)
   );
```

Codice 4.1.7*: -*Definizione della tabella REALIZZAZIONE – fine.

4.1.8 | Definizione della tabella PARTECIPAZIONE

```
-- TABLE: PARTECIPAZIONE
CREATE TABLE PARTECIPAZIONE(
    ID_Partecipazione    INTEGER          NOT NULL,
    Ruolo                VARCHAR2(50),
    cardinalita_Inf      INTEGER          NOT NULL,
    cardinalita_Sup      INTEGER          NOT NULL,
    Tipo                 VARCHAR2(50)     NOT NULL,
    Navigabilita         NUMBER(1)        DEFAULT ON NULL 1
                                             CHECK(Navigabilita IN (0,1)),
    Qualificatore        NUMBER(1)        DEFAULT ON NULL 0
                                             CHECK(Qualificatore IN (0,1)),
    FK_Associazione      INTEGER          NOT NULL,
    FK_Classe            INTEGER          NOT NULL
);



-- GESTIONE PRIMARY KEY [NULL]
 CREATE OR REPLACE TRIGGER SET_ID_PARTECIPAZIONE
    BEFORE INSERT ON PARTECIPAZIONE
    FOR EACH ROW
    DECLARE
        var_ID_Partecipazione PARTECIPAZIONE.ID_Partecipazione%TYPE;
    BEGIN
        IF (:NEW.ID_Partecipazione IS NULL) THEN
            SELECT NVL(MAX(P.ID_Partecipazione), 0) + 1 INTO var_ID_Partecipazione
            FROM PARTECIPAZIONE P;
            :NEW.ID_Partecipazione := var_ID_Partecipazione;
        END IF;
    END;
    /
```

Codice 4.1.8: -Definizione della tabella PARTECIPAZIONE – cont.

```
 -- CONSTRAINTS: PARTECIPAZIONE
ALTER TABLE PARTECIPAZIONE
   ADD(
        -- Chiave primaria
       CONSTRAINT PK_PARTECIPAZIONE PRIMARY KEY (ID_Partecipazione),
        -- Chiave esterna
       CONSTRAINT PART_FK_ASSOCIAZIONE FOREIGN KEY (FK_Associazione)
           REFERENCES ASSOCIAZIONE(ID_Associazione) ON DELETE CASCADE,
        -- Chiave esterna
       CONSTRAINT PART_FK_CLASSE FOREIGN KEY (FK_Classe)
           REFERENCES CLASSE(ID_Classe) ON DELETE CASCADE,
       -- Vincolo T_PAR
       CONSTRAINT T_PAR CHECK (Tipo IN ('semplice', 'aggregata', 'aggregante',
                                        'composta', 'componente')),
       -- Vincolo PART_CHECK_INTERVAL
       CONSTRAINT PART_CHECK_INTERVAL CHECK ((cardinalita_Inf <= cardinalita_Sup)
                                             AND (cardinalita_Sup > 0)),
       -- Vincolo COMP_INTERVAL
       CONSTRAINT COMP_INTERVAL CHECK ((Tipo <> 'componente') OR
                                       (Tipo = 'componente' AND cardinalita_Sup = 1))
   );
```

Codice 4.1.8: -Definizione della tabella PARTECIPAZIONE – fine.

4.1.9a | Definizione della tabella TIPO

```
-- TABLE: TIPO
CREATE TABLE TIPO(
    ID_Tipo             INTEGER,
    Nome                VARCHAR2(50)    NOT NULL,
    Tipo                VARCHAR2(50)    NOT NULL,
    FK_Scope            INTEGER,
    FK_ClassDiagram     INTEGER,
    FK_Parametro        INTEGER,
    FK_Classe           INTEGER
);



-- GESTIONE PRIMARY KEY [NULL]
  CREATE OR REPLACE TRIGGER SET_ID_TIPO
    BEFORE INSERT ON TIPO
    FOR EACH ROW
    DECLARE
        var_ID_Tipo TIPO.ID_Tipo%TYPE;
    BEGIN
        IF (:NEW.ID_Tipo IS NULL) THEN
            SELECT NVL(MAX(T.ID_Tipo), 0) + 1 INTO var_ID_Tipo
            FROM TIPO T;
            :NEW.ID_Tipo := var_ID_Tipo;
        END IF;
    END;
    /



-- TIPI PREDEFINITI
INSERT INTO TIPO VALUES(NULL, 'boolean', 'primitivo', NULL, NULL, NULL, NULL);
INSERT INTO TIPO VALUES(NULL, 'integer', 'primitivo', NULL, NULL, NULL, NULL);
INSERT INTO TIPO VALUES(NULL, 'string',  'primitivo', NULL, NULL, NULL, NULL);
INSERT INTO TIPO VALUES(NULL, 'real',    'primitivo', NULL, NULL, NULL, NULL);
```

Codice 4.1.9a*: -*Definizione della tabella TIPO – cont.

```
-- CONSTRAINTS: TIPO
ALTER TABLE TIPO
   ADD(
      -- Chiave primaria
      CONSTRAINT PK_TIPO PRIMARY KEY (ID_Tipo),
      -- Chiave esterna
      CONSTRAINT TIPO_FK_SCOPE FOREIGN KEY (FK_Scope)
         REFERENCES CLASSE(ID_Classe) ON DELETE CASCADE,
      -- Chiave esterna
      CONSTRAINT TIPO_FK_CLASSDIAGRAM FOREIGN KEY (FK_ClassDiagram)
         REFERENCES CLASSDIAGRAM(ID_ClassDiagram) ON DELETE CASCADE,

      -- FK_Parametro -> dopo la definizione della tabella PARAMETRO

      -- Chiave esterna
      CONSTRAINT TIPO_FK_CLASSE FOREIGN KEY (FK_Classe)
         REFERENCES CLASSE(ID_Classe) ON DELETE CASCADE,
      -- Vincolo T_TYPE
      CONSTRAINT T_TYPE CHECK (Tipo IN ('strutturato', 'classe',
                                        'parametrico', 'primitivo', 'enumerazione')),
      -- Vincolo IS_VALID_REF_PARAM
      CONSTRAINT IS_VALID_REF_PARAM CHECK ((Tipo <> 'parametrico'
                                            AND FK_Parametro IS NULL) OR
                                           (Tipo = 'parametrico'
                                            AND FK_Parametro IS NOT NULL)),
      -- Vincolo IS_VALID_REF_CLASS
      CONSTRAINT IS_VALID_REF_CLASS CHECK ((Tipo <> 'classe' AND FK_Classe IS NULL) OR
                                           (Tipo = 'classe' AND FK_Classe IS NOT NULL)),
      -- Vincolo IS_VALID_SCOPE
      CONSTRAINT IS_VALID_SCOPE CHECK ((Tipo <> 'parametrico' AND FK_Scope IS NULL) OR
                                       (Tipo = 'parametrico' AND FK_Scope IS NOT NULL)),
      -- Vincolo NOT_T_PRIM
      CONSTRAINT NOT_T_PRIM CHECK ((Tipo != 'primitivo') OR
                                   (Tipo = 'primitivo' AND Nome NOT IN
                                   ('boolean', 'integer', 'string', 'real'))) NOVALIDATE,
      -- Vincolo UNIQUE_TYPE_NAME
      CONSTRAINT UNIQUE_TYPE_NAME UNIQUE (FK_ClassDiagram, Nome, Tipo, FK_Scope)
   );
```

Codice 4.1.9a: -Definizione della tabella TIPO – fine.

4.1.10 | Definizione della tabella ATTRIBUTO

```
-- Tabella ATTRIBUTO
CREATE TABLE ATTRIBUTO(
   ID_Attributo        INTEGER,
   Nome                VARCHAR2(50)    NOT NULL,
   Descrizione         VARCHAR2(300)   DEFAULT ON NULL 'no description',
   cardinalita_Inf     INTEGER         NOT NULL,
   cardinalita_Sup     INTEGER         NOT NULL,
   h_scope             VARCHAR2(30)    DEFAULT ON NULL 'public',
   FK_Classe           INTEGER         NOT NULL,
   FK_Tipo             INTEGER         NOT NULL
);


-- GESTIONE PRIMARY KEY [NULL]
CREATE OR REPLACE TRIGGER SET_ID_ATTRIBUTO
   BEFORE INSERT ON ATTRIBUTO
   FOR EACH ROW
   DECLARE
       var_ID_Attributo ATTRIBUTO.ID_Attributo%TYPE;
   BEGIN
       IF (:NEW.ID_Attributo IS NULL) THEN
           SELECT NVL(MAX(A.ID_Attributo), 0) + 1 INTO var_ID_Attributo
           FROM ATTRIBUTO A;
           :NEW.ID_Attributo := var_ID_Attributo;
       END IF;
   END;
   /


-- CONSTRAINTS: ATTRIBUTO
ALTER TABLE ATTRIBUTO
   ADD(
       -- Chiave primaria
       CONSTRAINT PK_ATTRIBUTO PRIMARY KEY (ID_Attributo),
       -- Chiave esterna
       CONSTRAINT ATTR_FK_CLASSE FOREIGN KEY (FK_Classe)
           REFERENCES CLASSE(ID_Classe) ON DELETE CASCADE,
       -- Chiave esterna
       CONSTRAINT ATTR_FK_TIPO FOREIGN KEY (FK_Tipo)
           REFERENCES TIPO(ID_Tipo) ON DELETE CASCADE,
       -- Vincolo ATTR_SCOPE
       CONSTRAINT ATTR_SCOPE CHECK (h_scope IN ('public', 'package',
                                                'protected', 'private')),
       -- Vincolo UNIQUE_ATTR_NAME
       CONSTRAINT UNIQUE_ATTR_NAME UNIQUE (FK_Classe, Nome),
       -- Vincolo ATTR_CHECK_INTERVAL
       CONSTRAINT ATTR_CHECK_INTERVAL CHECK ((cardinalita_Inf <= cardinalita_Sup)
                                             AND (cardinalita_Sup > 0))
   );
```

Codice 4.1.10: -Definizione della tabella ATTRIBUTO – fine.

4.1.11 | Definizione della tabella COMPOSIZIONE

```
-- TABLE: COMPOSIZIONE
CREATE TABLE COMPOSIZIONE(
   FK_Strutturato      INTEGER         NOT NULL,
   FK_Tipo             INTEGER         NOT NULL
 );


-- CONSTRAINTS: COMPOSIZIONE
ALTER TABLE COMPOSIZIONE
   ADD(
       -- Chiave esterna
       CONSTRAINT COMP_FK_STRUTTURATO FOREIGN KEY (FK_Strutturato)
           REFERENCES TIPO(ID_Tipo) ON DELETE CASCADE,
       -- Chiave esterna
       CONSTRAINT COMP_FK_TIPO FOREIGN KEY (FK_Tipo)
           REFERENCES TIPO(ID_Tipo) ON DELETE CASCADE,
       -- Vincolo UNIQUE_COMPOSITION
       CONSTRAINT UNIQUE_COMPOSITION UNIQUE (FK_Strutturato, FK_Tipo)
   );
```

Codice 4.1.11: -Definizione della tabella COMPOSIZIONE – fine.

4.1.12 | Definizione della tabella LITERAL

```
-- TABLE: LITERAL
CREATE TABLE LITERAL(
   Literal             VARCHAR2(50)    NOT NULL,
   FK_Enumerazione     INTEGER         NOT NULL
);


-- CONSTRAINTS: LITERAL
ALTER TABLE LITERAL
   ADD(
       -- Chiave esterna
       CONSTRAINT LITERAL_FK_ENUMERAZIONE FOREIGN KEY (FK_Enumerazione)
           REFERENCES TIPO(ID_Tipo) ON DELETE CASCADE,
       -- Vincolo UNIQUE_LITERAL
       CONSTRAINT UNIQUE_LITERAL UNIQUE (Literal, FK_Enumerazione)
   );
```

Codice 4.1.12: -Definizione della tabella LITERAL – fine.

4.1.13 | Definizione della tabella METODO

```
-- Tabella METODO
CREATE TABLE METODO(
   ID_Metodo           INTEGER,
   Nome                VARCHAR2(50)    NOT NULL,
   Descrizione         VARCHAR2(300)   DEFAULT ON NULL 'no description',
   h_scope             VARCHAR2(30)    DEFAULT ON NULL 'public',
   FK_Classe           INTEGER,
   FK_Interfaccia      INTEGER,
   FK_Tipo             INTEGER
);


-- GESTIONE PRIMARY KEY [NULL]
CREATE OR REPLACE TRIGGER SET_ID_METODO
   BEFORE INSERT ON METODO
   FOR EACH ROW
   DECLARE
       var_ID_Metodo METODO.ID_Metodo%TYPE;
   BEGIN
       IF (:NEW.ID_Metodo IS NULL) THEN
           SELECT NVL(MAX(M.ID_Metodo), 0) + 1 INTO var_ID_Metodo
           FROM METODO M;
           :NEW.ID_Metodo := var_ID_Metodo;
       END IF;
   END;
   /


-- CONSTRAINTS: METODO
ALTER TABLE METODO
   ADD(
       -- Chiave primaria
       CONSTRAINT PK_METODO PRIMARY KEY (ID_Metodo),
       -- Chiave esterna
       CONSTRAINT METODO_FK_CLASSE FOREIGN KEY (FK_Classe)
           REFERENCES CLASSE(ID_Classe) ON DELETE CASCADE,
       -- Chiave esterna
       CONSTRAINT METODO_FK_INTERFACCIA FOREIGN KEY (FK_Interfaccia)
           REFERENCES INTERFACCIA(ID_Interfaccia) ON DELETE CASCADE,
       -- Chiave esterna
       CONSTRAINT METODO_FK_TIPO FOREIGN KEY (FK_Tipo)
           REFERENCES TIPO(ID_Tipo) ON DELETE SET NULL,
       -- Vincolo METH_SCOPE
       CONSTRAINT METH_SCOPE CHECK (h_scope IN ('public', 'package',
                                                'protected', 'private')),
       -- Vincolo METH_IS_PUBLIC
       CONSTRAINT METH_IS_PUBLIC CHECK ((FK_Interfaccia IS NULL) OR
                                        (FK_Interfaccia IS NOT NULL
                                           AND h_scope = 'public')),
       -- Vincolo METH_OF
       CONSTRAINT METH_OF CHECK((FK_Interfaccia IS NULL AND FK_Classe IS NOT NULL) OR
                                (FK_Interfaccia IS NOT NULL AND FK_Classe IS NULL))
   );
```

Codice 4.1.13: -Definizione della tabella METODO – fine.

4.1.14 | Definizione della tabella PARAMETRO

```
-- Tabella PARAMETRO
CREATE TABLE PARAMETRO(
    ID_Parametro        INTEGER,
    Nome                VARCHAR2(50)    NOT NULL,
    Posizione           INTEGER         NOT NULL,
    FK_Classe           INTEGER,
    FK_Metodo           INTEGER,
    FK_Tipo             INTEGER
);


-- GESTIONE PRIMARY KEY [NULL]
CREATE OR REPLACE TRIGGER SET_ID_PARAMETRO
    BEFORE INSERT ON PARAMETRO
    FOR EACH ROW
    DECLARE
        var_ID_Parametro PARAMETRO.ID_Parametro%TYPE;
    BEGIN
        IF (:NEW.ID_Parametro IS NULL) THEN
            SELECT NVL(MAX(P.ID_Parametro), 0) + 1 INTO var_ID_Parametro
            FROM PARAMETRO P;
            :NEW.ID_Parametro := var_ID_Parametro;
        END IF;
    END;
    /


 -- CONSTRAINTS: PARAMETRO
ALTER TABLE PARAMETRO
    ADD(
        -- Chiave primaria
        CONSTRAINT PK_Parametro PRIMARY KEY (ID_Parametro),
        -- Chiave esterna
        CONSTRAINT PARAMETRO_FK_CLASSE FOREIGN KEY (FK_Classe)
            REFERENCES CLASSE(ID_Classe) ON DELETE CASCADE,
        -- Chiave esterna
        CONSTRAINT PARAMETRO_FK_METODO FOREIGN KEY (FK_Metodo)
            REFERENCES METODO(ID_Metodo) ON DELETE CASCADE,
        -- Chiave esterna
        CONSTRAINT PARAMETRO_FK_TIPO FOREIGN KEY (FK_Tipo)
            REFERENCES TIPO(ID_Tipo) ON DELETE CASCADE,
        -- Vincolo
        CONSTRAINT PARAM_BELONGS_TO CHECK ((FK_Classe IS NOT NULL
                                            AND FK_Metodo IS NULL) OR
                                           (FK_Classe IS NULL
                                            AND FK_Metodo IS NOT NULL)),
        -- Vincolo NO_TYPE
        CONSTRAINT NO_TYPE CHECK ((FK_Classe IS NULL
                                   AND FK_Tipo IS NOT NULL) OR
                                  (FK_Classe IS NOT NULL
                                   AND FK_Tipo IS NULL)),
        -- Vincolo UNIQUE_POS
        CONSTRAINT UNIQUE_POS UNIQUE (Posizione, FK_Classe, FK_Metodo)
    );
```

Codice 4.1.14: -Definizione della tabella PARAMETRO – fine.

4.1.9b | Definizione della chiave esterna di PARAMETRO in TIPO

```
-- CONSTRAINTS: TIPO -> FK_Parametro
ALTER TABLE TIPO
    ADD(
        -- Chiave esterna
        CONSTRAINT TIPO_FK_PARAMETRO FOREIGN KEY (FK_Parametro)
            REFERENCES PARAMETRO(ID_Parametro) ON DELETE CASCADE
    );
```

Codice 4.1.9b: -Definizione della chiave esterna di PARAMETRO in TIPO – fine.

4.1.15 | Definizione della tabella QUALIFICAZIONE

```
-- TABLE: QUALIFICAZIONE
CREATE TABLE QUALIFICAZIONE(
    FK_Partecipazione    INTEGER         NOT NULL,
    FK_Attributo         INTEGER         NOT NULL
);


 -- CONSTRAINTS: QUALIFICAZIONE
ALTER TABLE QUALIFICAZIONE
    ADD(
        -- Chiave esterna
        CONSTRAINT QUALIF_FK_PARTECIPAZIONE FOREIGN KEY (FK_Partecipazione)
            REFERENCES PARTECIPAZIONE(ID_Partecipazione) ON DELETE CASCADE,
        -- Chiave esterna
        CONSTRAINT QUALIF_FK_ATTRIBUTO FOREIGN KEY (FK_Attributo)
            REFERENCES ATTRIBUTO(ID_Attributo) ON DELETE CASCADE,
        -- Vincolo UNIQUE_QUALIF
        CONSTRAINT UNIQUE_QUALIF UNIQUE (FK_Partecipazione, FK_Attributo)
    );
```

Codice 4.1.15: -Definizione della tabella QUALIFICAZIONE – fine.