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 2014/2015

Progettazione e sviluppo di una base di dati relazionale per la gestione di un software repository con focus sulle funzionalità di versioning e gestione issues

Autore:
Luigi Libero Lucio Starace
Matricola N86/1404
lui.starace@studenti.unina.it

Docenti:
Prof. Adriano Peron
Prof. Alessandro De Luca

13 agosto 2015

Questa pagina è stata lasciata intenzionalmente bianca.

Indice

Capitolo 1

Descrizione del progetto

1.1 Descrizione sintetica e requisiti

Si progetterà ed implementerà una base di dati relazionale che possa essere d'ausilio alla gestione di un repository software. La base di dati conterrà i descrittori di diversi progetti Java e informazione sulla loro strutturazione gerarchica in packages e classi. Il sistema permetterà di associare a ciascun descrittore nei vari livelli gerarchici il file (memorizzato in un file-system) che ne contiene il codice sorgente, nonché informazione sugli autori del codice e sulla tempistica relativa allo sviluppo.

Ciascun progetto potrà avere diverse release e tra gli elementi di release successive andrà tenuta una traccia di corrispondenza che indichi lo stato dell'elemento rispetto al suo omologo della release precedente. In particolare tale corrispondenza indicherà se un elemento è rimasto immutato, se è stato modificato, se è stato aggiunto nella release corrente o eliminato. Al livello di dettaglio più basso andrà tenuta per ciascuna classe una traccia di corrispondenza che indichi se ci sono stati cambiamenti negli attributi, nell'implementazione o nella segnatura dei metodi.

La base di dati permetterà inoltre di associare a ciascun livello di descrizione delle istanze di intervento (issues) che saranno prese in carico dagli sviluppatori.

1.2 Il repository software

I codici sorgente dei progetti presenti nel repository sono organizzati nel file system di appoggio in maniera rigidamente gerarchica secondo la struttura del progetto stesso, in linea con quello che è l'approccio predefinito delle IDE Java più comuni come *Eclipse* e *Netbeans*. Il seguente schema mostra la struttura generale di tale file-system:

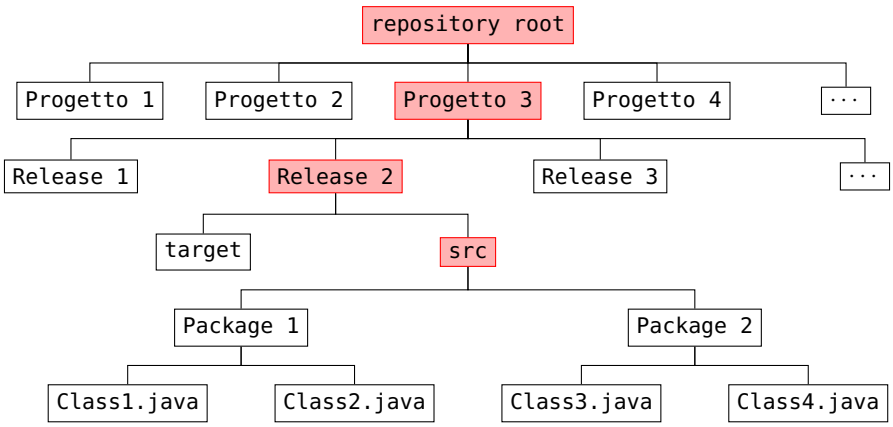


Figura 1.1: Rappresentazione del file-system del repository espanso per mostrare i file sorgente della release 2 del progetto 3.

Il ciclo di sviluppo e rilascio dei progetti può sintetizzarsi nei seguenti passaggi:

1. **Creazione di una nuova release di sviluppo**. Si procede creando una nuova directory all'interno della directory del progetto. In essa potranno essere copiati i sorgenti della (eventuale) release precedente che saranno poi modificati oppure si potrà riscrivere tutto il codice ex novo.

2. **Sviluppo**. Tutti gli sviluppatori abilitati a lavorare al progetto effettuano modifiche (commit) ai sorgenti. Il sistema tiene traccia degli autori e della tempistica relativa a dette modifiche.

3. **Rilascio**. Quando la release di sviluppo viene considerata completa e pronta al rilascio, il responsabile del progetto, settando un opportuno flag, la trasforma in una release completa. In seguito a questa operazione la base di dati, con opportuni trigger, calcolerà l'informazione relativa all'andamento del progetto rispetto alle release precedenti.

4. Se lo sviluppo del progetto non viene abbandonato si torna al punto 1 e si crea una nuova release di sviluppo.

L'instanziazione di issues è possibile per tutte le release indipendentemente dal loro stato di completamento. Una issue riguardante una release già completa potrà essere risolta all'interno della stessa release (ad esempio con il rilascio di una patch) oppure nelle release successive.

Note Sebbene non sia strettamente richiesto dal linguaggio Java, nel repository sarà obbligatorio che ogni classe appartenga ad un package. Tale imposizione non è restrittiva e favorisce una buona programmazione proibendo l'utilizzo, generalmente sconsigliabile[1], di classi in *unnamed packages*.

[1] Gli *unnamed packages* sono stati introdotti in Java principalmente per testing di piccole applicazioni. In generale infatti è auspicabile che i nomi dei package siano unici e inserire classi in *unnamed packages* viola questa convenzione. Inoltre non è possibile importare classi da *unnamed packages*.
`http://docs.oracle.com/javase/specs/jls/se8/html/jls-7.html`

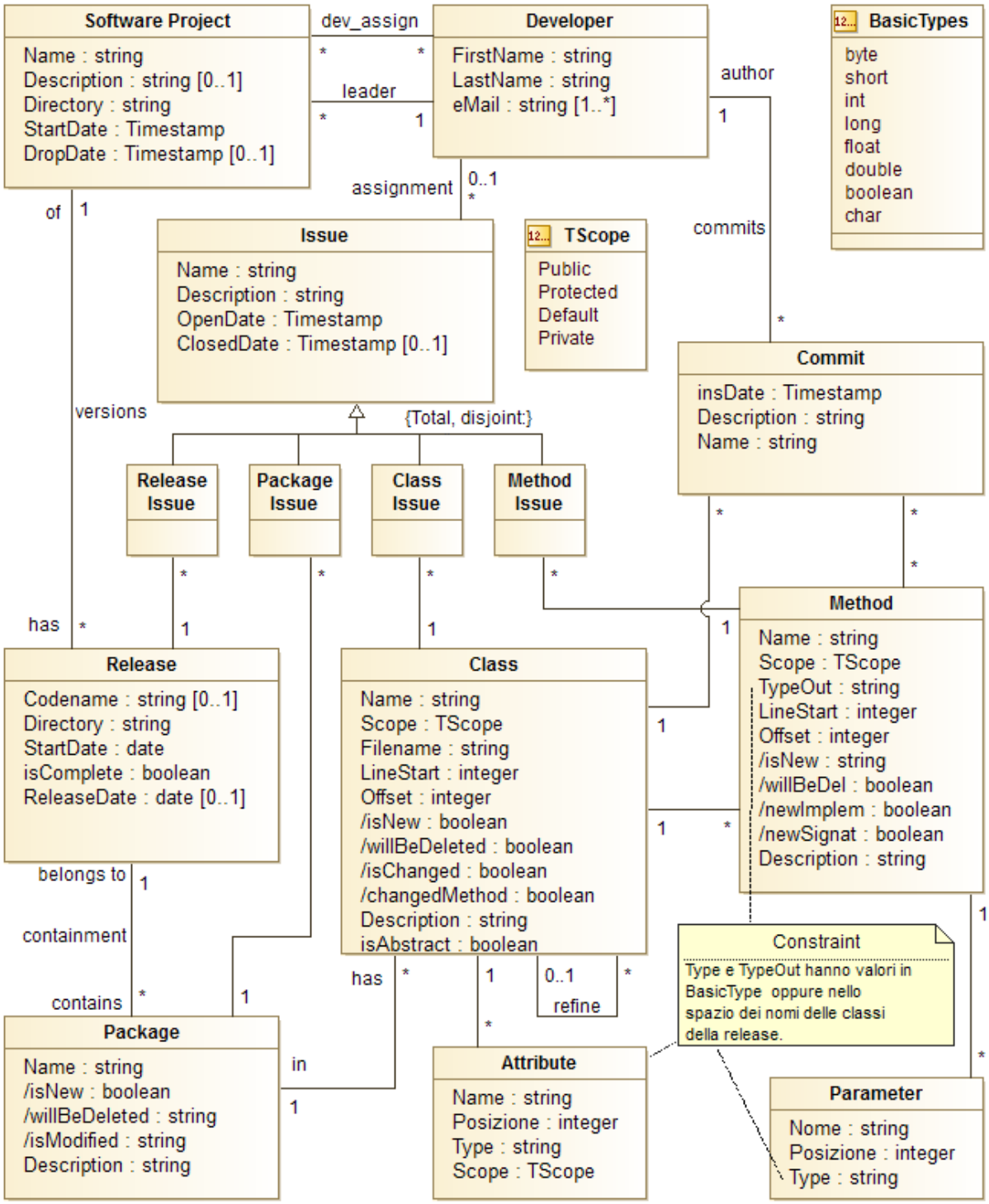
Capitolo 2

Progettazione concettuale

In questo capitolo inizia la progettazione della base di dati al livello di astrazione più alto. Dal risultato dell'analisi dei requisiti che devono essere soddisfatti si arriverà ad uno **schema concettuale** indipendente dalla struttura dei dati e dall'implementazione fisica. In tale schema concettuale, che verrà rappresentato usando un class diagram UML, si evidenzieranno le entità (concetti) rilevanti ai fini della rappresentazione dei dati e le relazioni che intercorrono tra esse. Si delineeranno anche eventuali vincoli da imporre.

Convenzioni sulla rappresentazione dei class diagram
Nelle rappresentazioni di class diagram UML che seguono tutti gli attributi, salvo ove specificato diversamente, hanno molteplicità `[1]`. Per semplicità di rappresentazione, inoltre, i nomi di alcune associazioni ed i ruoli che le classi giocano in esse sono stati omessi. La descrizione completa è comunque riportata in seguito nel dizionario delle associazioni.

2.1 Class diagram



2.2 Ristrutturazione del class diagram

Al fine di rendere il class diagram idoneo alla traduzione in schemi relazionali e di migliorare l'efficienza dell'implementazione si procede alla ristrutturazione dello stesso. Al termine del procedimento il class diagram non conterrà attributi strutturati, attributi multipli e gerarchie di specializzazione.

2.2.1 Analisi delle ridondanze

Non sono presenti significative ridondanze da eliminare. Al contrario, in fase implementativa potrebbe rivelarsi conveniente introdurre alcuni attributi ridondanti al fine di alleggerire il carico sulla macchina ospite. Ad esempio si potrebbe introdurre in `CLASS` un attributo `Path` che contenga il percorso completo al file sorgente dove è definita la classe. Calcolare tale informazione un'unica volta è vantaggioso rispetto al doverla ricavare da molteplici prodotti ogni volta, specialmente dal momento che si può presumere che gli accessi diretti ai file sorgente da parte delle applicazioni che si interfacceranno alla base di dati saranno frequenti.

2.2.2 Analisi degli identificativi

Risulta conveniente ai fini dell'efficienza l'introduzione di chiavi "tecniche" in ogni entità. Tali chiavi techiche altro non saranno che identificativi numerici che permetteranno di discriminare con maggiore facilità le istanze.

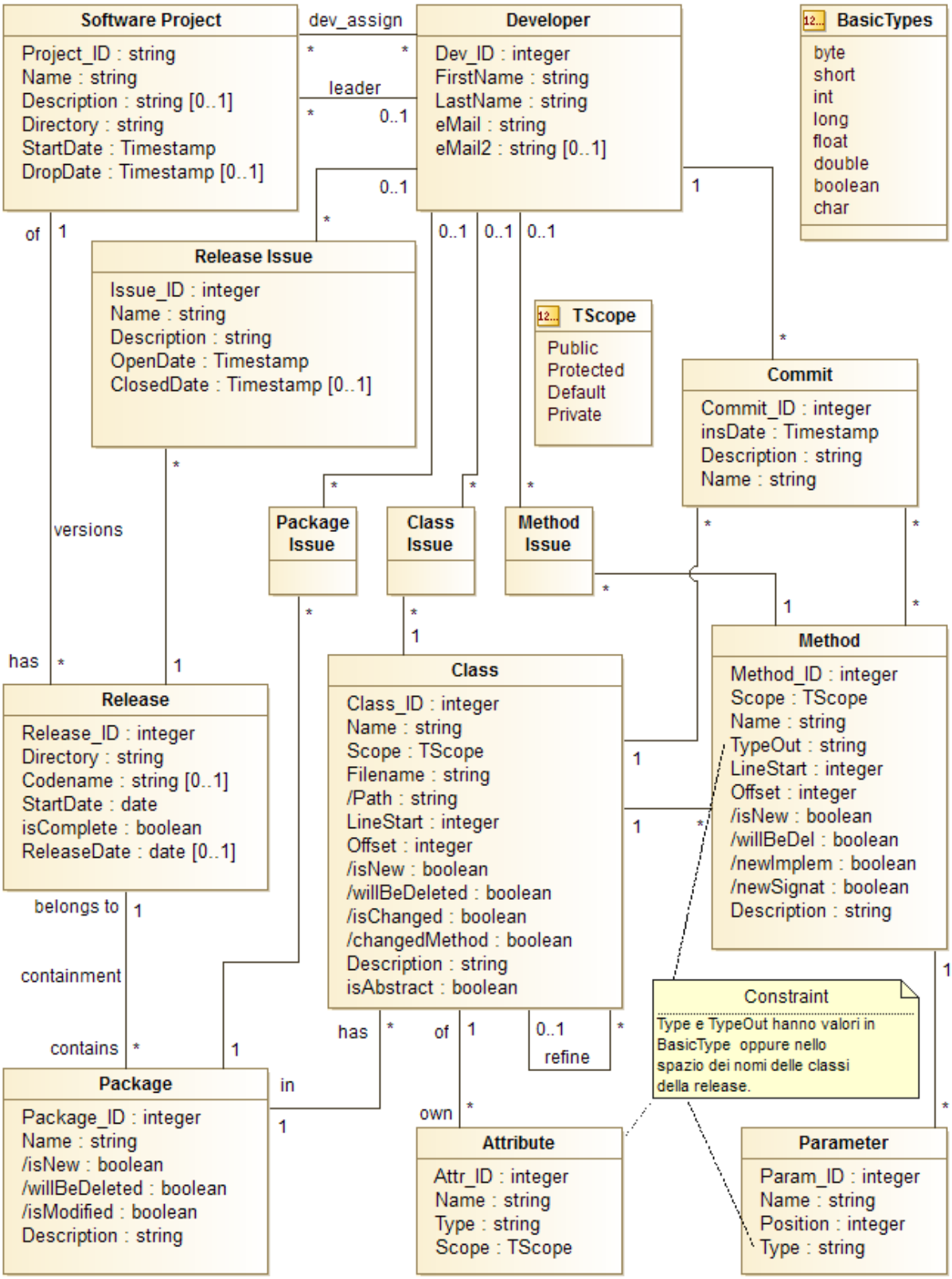
2.2.3 Rimozione degli attributi multipli

L'attributo multiplo `eMail` della classe `Developer` è da rimuovere. Una soluzione ragionevole potrebbe essere quella di limitare a due il numero di possibili indirizzi email per sviluppatore. Quindi si introdurrà al posto dell'attributo multiplo un attributo `eMail` con molteplicità `[1]` e un attributo `eMail2` con molteplicità `[0..1]` che rappresenterà l'indirizzo secondario (facoltativo).

2.2.4 Rimozione delle gerarchie di specializzazione

Si procede con l'eliminazione della specializzazione di `Issue` nelle issue specifiche per i vari livelli. Si tratta di una specializzazione totale e disgiunta e si procederà all'eliminazione "schiacciando" la superclasse nelle sottoclassi, preservando maggiormente la struttura gerarchica ed evitando l'introduzione di attributi che permettano di discriminare tra le issue dei vari livelli.

2.3 Class diagram ristrutturato



Nella rappresentazione del class diagram ristrutturato proposta si consideri che gli attributi delle classi `Package Issue`, `Class Issue` e `Method Issue` sono gli stessi di `Release Issue` e sono stati omessi per semplicità di rappresentazione.

2.4 Dizionario dei dati

2.4.1 Dizionario delle classi

Tabella 2.1: Dizionario delle classi

Classe	Descrizione	Attributi
Software Project	Descrittore di ciascun progetto presente nel repository software.	`Project_ID` (*integer*): chiave tecnica. Identifica univocamente ciascun progetto nel repository. `Name` (*string*): nome del progetto. `Details` (*string, opzionale*): breve descrizione del progetto. `Directory` (*string*): nome della directory radice del progetto all'interno della radice del repository. `StartDate` (*timestamp*): data di creazione del progetto. `DropDate` (*timestamp, opzionale*): data di abbandono del progetto.
Release	Descrittore di ciascuna versione rilasciata (o rilascianda) del progetto.	`Release_ID` (*integer*): identifica univocamente ogni istanza di `Release`. `Directory` (*string*): il nome della sottodirectory della directory del progetto che contiene i sorgenti e i binari della release. `Codename` (*string, opzionale*): nome in codice per la release. `StartDate` (*date*): data di inizio dei lavori per la release. `isComplete` (*boolean*): indica se la release è da considerarsi completa oppure in fase di sviluppo. `ClosedDate` (*date, opzionale*): indica la data (eventuale) in cui la release è stata indicata come completa.
Package	Descrittore di Package Java appartenenti ad una release.	`Package_ID` (*integer*): identifica univocamente ogni istanza di `Package`. `Name` (*string*): nome del pacchetto. Coincide con il nome della directory che contiene le classi del pacchetto. `isNew` (*boolean, derivato*): indica se il pacchetto è stato introdotto in questa release, cioè se non esisteva un pacchetto con lo stesso nome nella eventuale release precedente. `willBeDeleted` (*bool, derivato*): indica se il pacchetto non è presente nella release successiva. `isModified` (*bool, derivato*): indica se il pacchetto è stato modificato rispetto alla release precedente. `Description`(*string*): descrizione del package.

Tabella 2.1: *continua nella prossima pagina*

Tabella 2.1: *continua dalla pagina precedente*

Classe	Descrizione	Attributi
Class	Descrittore di una classe in un progetto.	`Class_ID` (*integer*): identifica univocamente ciascuna istanza di `Class`. `Name` (*string*): il nome della classe. `Scope` (*TScope*): indica lo scope della classe all'interno del programma. `FileName` (*string*): indica il nome del file sorgente all'interno del quale è definita la classe. `Path` (*string*): indica il percorso completo, relativo alla directory root del repository, del file. `LineStart` (*integer*): indica la linea dove inizia la definizione della classe nel file `.java`. `Offset` (*integer*): indica la lunghezza in linee della definizione della classe. `isNew` (*boolean, derivato*): indica se la classe non era presente nella release precedente (nello stesso package, visto che ci possono essere classi con lo stesso nome in packages diversi). `willBeDeleted` (*boolean, derivato*): indica se la classe non è presente nella release successiva (nello stesso package, visto che ci possono essere classi con lo stesso nome in packages diversi). `isChanged` (*boolean, derivato*): indica se la classe ha subito modifiche o meno rispetto alla release precedente. `changedMethod` (*boolean, derivato*): indica se almeno un metodo della classe ha subito modifiche all'implementazione, ha cambiato segnatura, è stato aggiunto o eliminato. `isAbstract` (*boolean*): indica se la classe è astratta. `Description`(*string*): descrizione della classe. Fornisce dettagli sull'implementazione e lo scopo della classe.
Attribute	Descrittore di un attributo (*campo*) di una classe.	`Attr_ID` (*integer*): identifica univocamente un attributo. `Name` (*string*): name binding dell'attributo. `Type` (*string*): indica il tipo dell'attributo. `Scope` (*TScope*): indica lo scope dell'attributo.
Developer	Descrittore di sviluppatore abilitato a lavorare nel repository	`Dev_ID` (*integer*) `FirstName` (*string*): nome dello sviluppatore. `LastName` (*string*): cognome dello sviluppatore. `eMail` (*string*): indirizzo email principale dello sviluppatore. `eMail2` (*string*): indirizzo email secondario (di recupero).

Tabella 2.1: *continua nella prossima pagina*

Tabella 2.1: *continua dalla pagina precedente*

Classe	Descrizione	Attributi
Method	Descrittore del metodo di una classe.	`Method_ID` (*integer*): identifica univocamente un metodo. `Scope` (*TScope*): indica lo scope del metodo. `Name` (*string*) `TypeOut` (*string*): indica il tipo del valore ritornato dal metodo. `LineStart` (*integer*): indica la prima riga del metodo nel file `.java`. `Offset` (*integer*): indica la lunghezza in linee del codice che implementa il metodo. `isNew` (*boolean, derivato*): indica se il metodo non era presente nella classe omologa della release precedente. `willBeDeleted` (*boolean, derivato*): indica se il metodo non è presente nella classe omologa della release successiva. `newImplem` (*boolean, derivato*): indica se la il metodo ha subito modifiche all'implementazione rispetto al suo omologo della release precedente. `newSignature` (*boolean, derivato*): indica se il metodo ha cambiato segnatura rispetto al suo omologo della release precedente. `Description`(*string*): descrizione del metodo. Fornisce dettagli sulla sua implementazione o note particolari sull'utilizzo corretto.
Parameter	Descrittore di parametro di un metodo.	`Param_ID` (*integer*) `Name` (*string*): identificativo del parametro formale. `Position` (*integer*): posizione del parametro formale nella lista dei parametri formali del metodo. `Type` (*string*): tipo del parametro formale.
Release **(Package)** **(Class)** **(Method)** **Issue**	Descrittore di un'istanza di Issue relativa ad una release/package/-classe/metodo.	`Issue_ID` (*integer*): identifica univocamente una issue di release/package/classe/metodo. `Name` (*string*): titolo dato alla segnalazione di issue. `Details` (*string*): descrizione più dettagliata della issue. `OpenDate` (*date*): data di apertura della issue. `ClosedDate` (*date, opzionale*): data di chiusura della issue
Commit	Descrittore di un'istanza di modifica ai codici sorgente di una classe da parte di uno sviluppatore.	`Commit_ID` (*integer*): identifica univocamente ciascuna operazione di modifica ad una classe. `Date` (*timestamp*): marca temporale della modifica. `Name` (*string*): titolo della modifica apportata. `Details` (*string*): descrizione della modifica apportata.

Tabella 2.1 - Dizionario delle classi

2.4.2 Dizionario delle associazioni

Tabella 2.2: Dizionario delle associazioni

Nome	Descrizione	Classi coinvolte
Versions	Esprime l'appartenenza di release ad un progetto software.	`Software Project [1]` ruolo `of`: indica il progetto a cui appartiene una release. `Release [0..*]` ruolo `has`: indica la release di un progetto
Package cont.	Esprime l'appartenenza di un package ad una release.	`Release [1]` ruolo `belongs to`: indica la release a cui appartiene un package. `Package [0..*]` ruolo `contains`: indica i pacchetti che appartengono ad una release.
Class cont.	Esprime l'appartenenza di una classe ad un package.	`Package [1]` ruolo `in`: indica il pacchetto a cui appartiene una classe. `Classe [0..*]` ruolo `has`: indica le classi che appartengono ad un package.
Fields cont.	Esprime l'appartenenza di attributi ad una classe	`Class [1]` ruolo `of`: indica la classe a cui appartiene un attributo. `Attribute [0..*]` ruolo `owns`: indica gli attributi posseduti da una classe.
Method cont.	Esprime l'appartenenza di un metodo ad una classe.	`Class [1]` ruolo `has`: indica la classe a cui appartiene un metodo. `Method [0..*]` ruolo `owns`: indica i metodi posseduti di una classe.
Refine	Esprime gerarchie di specializzazione tra classi.	`Class [0..1]` ruolo `supclass`: la classe che viene estesa da altre. La cardinalità `0..1` è dovuta al fatto che Java non supporta l'ereditarietà multipla. `Class [0..*]` ruolo `extends`: le sottoclassi che estendono una classe.
Param. Spec.	Esprime la corrispondenza tra un metodo e i suoi parametri formali.	`Method [1]` ruolo `of`: indica il metodo a cui appartengono una release. `Parameter [0..*]` ruolo `has`: indica i parametri formali di un metodo.
Dev. Assign.	Esprime la possibilità di uno sviluppatore di lavorare ad un progetto.	`Software Project [1..*]` ruolo `assigned to`: indica il progetto a cui è abilitato a lavorare uno sviluppatore. `Developer [0..*]` ruolo `developed by`: indica gli sviluppatori abilitati a lavorare ad un progetto.
Leader	Esprime informazione relativa allo sviluppatore responsabile di un progetto.	`Software Project [0..*]` ruolo `leads`: indica il progetto di cui uno sviluppatore è responsabile. `Developer [1]` ruolo `lead by`: indica lo sviluppatore a capo di un progetto.
RelIssue Assign.	Esprime informazione relativa all'assegnamento di issues di release a sviluppatori che le prendano in carico.	`Release Issue [0..*]` ruolo `has`: indica la issue di release assegnata ad uno sviluppatore. `Developer [0..1]` ruolo `assigned to`: indica lo sviluppatore che deve prendersi carico della issue.

Tabella 2.2: *continua nella prossima pagina*

Tabella 2.2: *continua dalla pagina precedente*

Nome	Descrizione	Classi coinvolte
ClassIssue Assign.	Esprime informazione relativa all'assegnamento di issues di classe a sviluppatori che le prendano in carico.	`Class Issue [0..*]` ruolo `has`: indica la issue di classe assegnata ad uno sviluppatore. `Developer [0..1]` ruolo `assigned to`: indica lo sviluppatore che deve prendersi carico della issue.
PackIssue Assign.	Esprime informazione relativa all'assegnamento di issues di package a sviluppatori che le prendano in carico.	`Package Issue [0..*]` ruolo `has`: indica la issue di package assegnata ad uno sviluppatore. `Developer [0..1]` ruolo `assigned to`: indica lo sviluppatore che deve prendersi carico della issue.
MethIssue Assign.	Esprime informazione relativa all'assegnamento di issues di metodo a sviluppatori che le prendano in carico.	`Method Issue [0..*]` ruolo `has`: indica la issue di metodo assegnata ad uno sviluppatore. `Developer [0..1]` ruolo `assigned to`: indica lo sviluppatore assegnatario della issue.
IssueToRel	Esprime informazione relativa all'instanziazione di issues a release.	`Release Issue [0..*]` ruolo `has`: indica l'issue associata ad una release. `Release [1]` ruolo `referred to`: indica la release cui è assegnata la issue.
IssueToPack	Esprime informazione relativa all'instanziazione di issues a packages.	`Package Issue [0..*]` ruolo `has`: indica l'issue associata ad una package. `Package [1]` ruolo `referred to`: indica il package a cui è appuntata una issue.
IssueToClass	Esprime informazione relativa all'instanziazione di issues a classi.	`Class Issue [0..*]` ruolo `has`: indica l'issue associata ad una classe. `Class [1]` ruolo `referred to`: indica la classe a cui è appuntata una issue.
IssueToMeth	Esprime informazione relativa all'instanziazione di issues a metodi.	`Method Issue [0..*]` ruolo `has`: indica l'issue associata ad un metodo. `Method [1]` ruolo `referred to`: indica il metodo a cui è appuntata una issue.
ModifClass	Indica la classe che riceve una modifica	`Commit [0..*]` ruolo `modified by`: indica i commit che modificano una classe. `Class [1]` ruolo `modifies`: indica la classe a cui un commit apporta modifiche.
ModifMeth	Indica quali metodi di una classe sono modificati da un commit	`Commit [0..*]` ruolo `modified by`: indica i commit che modificano l'implementazione di un metodo. `Method [0..*]` ruolo `modifies`: indica i metodi a cui un commit apporta modifiche.
CommitDev	Indica la relazione tra sviluppatori e i commit che questi effettuano.	`Commit [0..*]` ruolo `makes`: indica i commit che sono effettuati da uno sviluppatore. `Developer [1]` ruolo `by`: indica llo sviluppatore che ha eseguito un commit.

Tabella 2.2 - Dizionario delle associazioni

2.4.3 Documento dei vincoli

Tabella 2.3: Documento dei vincoli

Nome vincolo	Descrizione
Single Public Class per file	Ogni file sorgente possiede alpiù una classe `public`. Ovvero non possono esistere due classi con scope `public` definite all'interno dello stesso file `.java`.
Release Composition	All'interno della stessa release non possono esistere due Packages con nome (e quindi cartella) uguale.
Package Composition	All'interno dello stesso package non possono esistere due Classi con nome uguale.
Class Composition	All'interno della stessa classe non possono esistere due attributi con stesso nome e stesso tipo o due metodi con la stessa segnatura.
No Auto-refine	Nella relazione `refine` una istanza di classe non può essere sottoclasse/superclasse di se stessa. In altre parole, l'istanza di classe in ruolo `extends` non può coincidere con l'istanza in ruolo `supclass`.
Legit mails	Gli indirizzi email degli sviluppatori devono essere indirizzi email di forma legittima, ovvero contenere almeno un carattere prima della `@`, almeno un carattere tra essa e il punto e almeno due caratteri nella parte finale.
Distinct Personal Mail	La email principale e quella secondaria devono essere distinte per ogni sviluppatore. Non possono esistere due sviluppatori con lo stesso indirizzo email principale o secondario. Inoltre non possono esistere due sviluppatori `S1, S2` tali che `S1.eMail1 = S2.eMail2`.
Type Values	I tipi degli attributi, dei parametri formali dei metodi ed il tipo del valore ritornato da un metodo devono essere valori di `BasicTypes` oppure nomi di classi della stessa release.
Commit Consistency	Ciascun commit può interessare una sola classe. Tutti i metodi interessati dalla modifica, quindi, devono appartenere alla stessa classe, cioè quella modificata.
Time consistency of Projects	Per ciascun progetto, non esiste una release associata tale che la data di creazione della release sia antecedente alla data di creazione del progetto. Se `DropDate` è settata, allora deve essere successiva a `StartDate` e non devono esistere release successive ad essa.
Time consistency of Releases	Per ciascuna release `ReleaseDate`, se settata, dev'essere successiva a `StartDate`.
Time consistency of Issues	Per ciascun tipo di issue e per ciascuna istanza, se `isClosed=true`, allora `ClosedDate > OpenDate`. Se `isClosed = false` allora `ClosedDate = NULL`.
File Structure Consistency for Classes	In ciascun file sorgente non è possibile che esistano due classi `C1` e `C2` tali che valga la seguente (si consideri `EndLine=StartLine+Offset`): `C1.StartLine > C2.StartLine > C1.EndLine > C2.EndLine`. In altre parole, è sempre rispettata la corretta strutturazione in blocchi. Il caso in cui una classe sia dichiarata all'interno di un'altra è invece possibile (*inner classes*).

Tabella 2.3: *continua nella prossima pagina*

Tabella 2.3: *continua dalla pagina precedente*

Nome vincolo	**Descrizione**
File Consistency for Methods	Per ciascun metodo `M` di una classe `C` vale la seguente (si consideri `EndLine=StartLine+Offset`): `C.StartLine > M.StartLine > M.EndLine > C.EndLine`. Inoltre, per ogni coppia `M1`, `M2` di metodi di una classe non può mai verificarsi che sia `M1.StartLine > M2.StartLine > M1.EndLine`.
Assigned Devs Commit	Soltanto sviluppatori assegnati ad un progetto possono effettuare commit su elementi di quel progetto.
Check valid hierarchy	Non possono essere stabilite relazioni di specializzazione tra classi appartenenti a release diverse.

Tabella 2.3 - Documento dei vincoli

Capitolo 3

Progettazione logica

In questo capitolo sarà trattata la fase successiva della progettazione della base di dati scendendo ad un livello di astrazione più basso rispetto alla precedente. Si tradurrà lo **schema concettuale** (già predisposto in seguito alla ristrutturazione) in uno **schema logico**, dipendente dal tipo di struttura dei dati prescelto cioè quello **relazionale puro**. Negli schemi relazionali che seguiranno le chiavi primarie sono indicate con una singola sottolineatura mentre le chiavi esterne con una doppia sottolineatura.

3.1 Traduzione in schemi relazionali

SOFTWARE_PROJECT(**Project_ID**, Name, Details, Directory, StartDate, DropDate, Leader)

Chiavi esterne: Leader ⟶ DEVELOPER.Dev_ID.

DEVELOPER(**Dev_ID**, FirstName, LastName, eMail, eMail2)

Chiavi esterne: nessuna.

DEV_ASSIGN(Dev, Project)

Chiavi esterne: Dev ⟶ DEVELOPER.Dev_ID.
Project ⟶ DEVELOPER.Dev_ID.

RELEASE(**Release_ID**, Directory, Version, Codename, StartDate, isComplete, ReleaseDate, Project)

Chiavi esterne: Project ⟶ SOFTWARE_PROJECT.Project_ID.

PACKAGE(**Package_ID**, Name, isNew, willBeDeleted, isModified, Description, Release)

Chiavi esterne: Release ⟶ RELEASE.Release_ID.

CLASS(**Class_ID**, Name, Scope, Filename, Path, LineStart, Offset, isNew, willBeDeleted, isChanged, changedMethod, Description, isAbstract, Package)

Chiavi esterne: Package ⟶ PACKAGE.Package_ID.

ATTRIBUTE(**Attr_ID**, Name, Type, Scope, Class)

Chiavi esterne: Class ⟶ CLASS.Class_ID.

METHOD(**Method_ID**, Scope, Name, TypeOut, LineStart, Offset, isNew, willBeDeleted, newImplem, newSignat, Description, Class)

Chiavi esterne: Class ⟶ CLASS.Class_ID.

PARAMETER(**Param_ID**, Name, Position, Type, Method)

Chiavi esterne: Method ⟶ METHOD.Method_ID.

RELEASE_ISSUE(**Issue_ID**, Name, Details, OpenDate, ClosedDate, Release, Dev)

Chiavi esterne: Release ⟶ RELEASE.Release_ID.
Dev ⟶ DEVELOPER.Dev_ID.

PACKAGE_ISSUE(**Issue_ID**, Name, Details, OpenDate, ClosedDate, Package, Dev)

Chiavi esterne: Package ⟶ PACKAGE.Package_ID.
Dev ⟶ DEVELOPER.Dev_ID.

CLASS_ISSUE(**Issue_ID**, Name, Details, OpenDate, ClosedDate, Class, Dev)

Chiavi esterne: Class ⟶ CLASS.Class_ID.
Dev ⟶ DEVELOPER.Dev_ID.

METHOD_ISSUE(**Issue_ID**, Name, Details, OpenDate, ClosedDate, Method, Dev)

Chiavi esterne: Method ⟶ METHOD.Method_ID.
Dev ⟶ DEVELOPER.Dev_ID.

COMMIT(**Commit_ID**, Details, insDate, Name, Class, Dev)

Chiavi esterne: Class ⟶ CLASS.Class_ID.
Dev ⟶ DEVELOPER.Dev_ID.

COMMIT_METH(Commit, Method)

Chiavi esterne: Commit ⟶ COMMIT.Commit_ID
Method ⟶ METHOD.Method_ID.

```
REFINE(SupClass, SubClass)
```

Chiavi esterne: SupClass ⟶ CLASS.Class_ID
SubClass ⟶ CLASS.Class_ID.

3.1.1 Traduzione delle associazioni

Nella seguente tabella vengono riportate esplicitamente le traduzioni delle varie associazioni nello schema logico.

Tabella 3.1: Traduzione delle associazioni

Associazione	Implementazione
Versions	Chiave esterna in RELEASE ⟶ SOFTWARE_PROJECT.
Package Cont.	Chiave esterna in PACKAGE ⟶ RELEASE.
Class Cont.	Chiave esterna in CLASS ⟶ PACKAGE.
Field Cont.	Chiave esterna in ATTRIBUTE ⟶ CLASS.
Method Cont.	Chiave esterna in RELEASE ⟶ CLASS.
Refine	Tradotta nello schema relazionale REFINE.
Param. Spec.	Chiave esterna in PARAMETER ⟶ METHOD.
Dev. Assign.	Tradotta nello schema relazionale DEV_ASSIGN.
Leader	Chiave esterna in SOFTWARE_PROJECT ⟶ DEVELOPER.
RelIssue Assign.	Chiave esterna in RELEASE_ISSUE ⟶ DEVELOPER.
ClassIssue Assign.	Chiave esterna in CLASS_ISSUE ⟶ DEVELOPER.
PackIssue Assign.	Chiave esterna in PACKAGE_ISSUE ⟶ DEVELOPER.
MethIssue Assign.	Chiave esterna in METHOD_ISSUE ⟶ DEVELOPER.
IssueToRel	Chiave esterna in RELEASE_ISSUE ⟶ RELEASE.
IssueToPack	Chiave esterna in PACKAGE_ISSUE ⟶ PACKAGE.
IssueToClass	Chiave esterna in CLASS_ISSUE ⟶ CLASS.
IssueToMeth	Chiave esterna in METHOD_ISSUE ⟶ METHOD.
ModifClass	Chiave esterna in COMMIT ⟶ CLASS.
ModifMeth	Tradotta nello schema relazionale COMMIT_METH.
CommitDev	Chiave esterna in COMMIT ⟶ DEVELOPER.

Tabella 3.1 - Traduzione delle associazioni

3.2 Schema Logico

Riassumendo quanto già detto nella sezione precedente, perveniamo al seguente schema logico:

SOFTWARE_PROJECT	(**Project_ID**, Name, Details, Directory, StartDate, DropDate, Leader)
DEVELOPER	(**Dev_ID**, FirstName, LastName, eMail, eMail2)
DEV_ASSIGN	(Dev, Project)
RELEASE	(**Release_ID**, Directory, Version, Codename, StartDate, isComplete, ReleaseDate, Project)
PACKAGE	(**Package_ID**, Name, isNew, willBeDeleted, isModified, Description, Release)
CLASS	(**Class_ID**, Name, Scope, Filename, Path, LineStart, Offset, isNew, willBeDeleted, isChanged, changedMethod, Description, isAbstract, Package)
ATTRIBUTE	(**Attr_ID**, Name, Type, Scope, Class)
METHOD	(**Method_ID**, Scope, Name, TypeOut, LineStart, Offset, isNew, willBeDeleted, newImplem, newSignat, Description, Class)
PARAMETER	(**Param_ID**, Name, Position, Type, Method)
RELEASE_ISSUE	(**Issue_ID**, Name, Details, OpenDate, ClosedDate, Release, Dev)
PACKAGE_ISSUE	(**Issue_ID**, Name, Details, OpenDate, ClosedDate, Package, Dev)
CLASS_ISSUE	(**Issue_ID**, Name, Details, OpenDate, ClosedDate, Class, Dev)
METHOD_ISSUE	(**Issue_ID**, Name, Details, OpenDate, ClosedDate, Method, Dev)
COMMIT	(**Commit_ID**, Details, Name, insDate, Class, Dev)
COMMIT_METH	(Commit, Method)
REFINE	(SupClass, SubClass)

Tabella 3.2: Schema logico

Capitolo 4

Progettazione fisica

La base di dati verrà implementata nel sistema per la gestione di basi di dati `Oracle XE 11g`. Per motivi di leggibilità i caratteri di whitespace nelle stringhe dei codici che seguono sono sostituiti dal carattere ␣.

4.1 Note sull'implementazione

Alcuni dettagli implementativi del progetto saranno modificati al fine di sfruttare al meglio le funzionalità del DBMS `Oracle XE 11g`. Poiché `Oracle` non implementa il tipo `boolean`, questo è stato simulato con un carattere (`CHAR`) con valori in {`'T'` (*true*), `'F'` (*false*)}. I nomi di alcune tabelle e attributi, inoltre, sono stati leggermente modificati per evitare possibili conflitti con parole chiave. Per implementare alcuni vincoli più complessi con trigger `INSTEAD OF`, infine, sono state rinominate alcune tabelle aggiungendo al nome il suffisso `_BASE` e, quindi, sono state definite delle viste semplici sulle tabelle rinominate.

4.2 Definizione delle tabelle

Seguono le definizioni delle tabelle estratte dallo script di creazione del database.

4.2.1 Definizione della tabella **DEVELOPER**

```
20
21 /**
22  * TABELLA: DEVELOPER
23  * Crea  la tabella e implementa i vincoli pi semplici.
24  */
25
26 -- Crea la tabella DEVELOPER
27 CREATE TABLE DEVELOPER_BASE
28 (
29   Dev_ID      INTEGER       DEFAULT '0' NOT NULL,
30   FirstName   VARCHAR2(64)  NOT NULL,
31   LastName    VARCHAR2(64)  NOT NULL,
32   eMail       VARCHAR2(320) NOT NULL UNIQUE CHECK(eMail LIKE '_%@_%.__%'),
33   eMail2      VARCHAR2(320) DEFAULT NULL
34                             CHECK(eMail2 LIKE '_%@_%.__%' OR eMail2 IS NULL)
35   -- una email legittima puo' contare {64}@{255} caratteri.
36   -- In totale 320.
37 );
38
39 /
40
41 -- Crea il vincolo di chiave primaria
42 ALTER TABLE DEVELOPER_BASE
43   ADD CONSTRAINT Dev_pk PRIMARY KEY(Dev_ID);
44
45 /
46
47 -- Crea la vista utilizzata nel trigger checkEmails
48 CREATE OR REPLACE VIEW DEVELOPER AS SELECT * FROM DEVELOPER_BASE;
49
50 /
51
52 -- Il calcolo della chiave primaria avviene nel trigger checkEmail alla riga 1115
```

Codice 4.1: Estratto da `createDatabase.sql` - Definizione della tabella `DEVELOPER`

4.2.2 Definizione della tabella SOFTWARE_PROJECT

```
54
55 /**
56  * TABELLA: SOFTWARE_PROJECT
57  * Crea  la tabella e implementa i vincoli pi semplici.
58  */
59
60
61 -- Crea la tabella SOFTWARE_PROJECT
62 CREATE TABLE SOFTWARE_PROJECT
63 (
64   Project_ID  INTEGER       DEFAULT 0,
65   Name        VARCHAR2(64)  DEFAULT 'New Project' NOT NULL,
66   Description VARCHAR2(256) NULL,
67   Directory   VARCHAR2(64)  NOT NULL UNIQUE,
68   StartDate   TIMESTAMP     DEFAULT SYSTIMESTAMP NOT NULL,
69   DropDate    TIMESTAMP     NULL,
70   Leader      INTEGER       NULL
71 );
72
73 /
74
75 -- Aggiunge i vincoli
76 ALTER TABLE SOFTWARE_PROJECT
77 ADD(
78   -- Chiave primaria
79   CONSTRAINT project_pk PRIMARY KEY (Project_ID), -- Project_ID NOT NULL
80   -- Chiave esterna
81   -- Se lo sviluppatore a capo di un progetto viene eliminato, il progetto
82   -- viene mantenuto e il leader viene settato a NULL
83   CONSTRAINT project_fk FOREIGN KEY (Leader)
84     REFERENCES DEVELOPER_BASE(Dev_ID) ON DELETE SET NULL,
85   -- Vincolo "Legit Names" (si veda documentazione)
86   CONSTRAINT valid_dir  CHECK (REGEXP_LIKE(Directory,'^[a-zA-Z0-9\/_-]{1,}$'))
87 );
88
89 /
90
91 -- Trigger per settare, se necessario, la chiave primaria automaticamente
92 CREATE OR REPLACE TRIGGER ProjectPK
93 BEFORE INSERT ON SOFTWARE_PROJECT
94 FOR EACH ROW
95 BEGIN
96   DECLARE
97     pk SOFTWARE_PROJECT.PROJECT_ID%TYPE;
98   BEGIN
99     IF(:NEW.Project_ID = 0) THEN
100      SELECT NVL(MAX(Project_ID),0) + 1 INTO pk FROM SOFTWARE_PROJECT;
101      :NEW.Project_ID := pk;
102    END IF;
103  END;
104 END;
105
106 /
```

Codice 4.2: Estratto da `createDatabase.sql` - Definizione della tabella `SOFTWARE_PROJECT`

4.2.3 Definizione della tabella **DEV_ASSIGN**

```
108
109 /**
110  * TABELLA: DEV_ASSIGN
111  * Crea  la tabella e implementa i vincoli pi semplici.
112  */
113
114
115 -- Crea la tabella DEV_ASSIGN
116 CREATE TABLE DEV_ASSIGN
117 (
118   Dev         INTEGER NOT NULL,
119   Project_ID  INTEGER NOT NULL
120 );
121
122 /
123
124 -- Aggiunge i vincoli
125 ALTER TABLE DEV_ASSIGN
126 ADD(
127   -- Chiave esterna verso DEVELOPER
128   -- Eliminando uno sviluppatore, vengono eliminate anche tutte le sue associazioni
129   -- a progetti.
130   CONSTRAINT devAssfk1 FOREIGN KEY(Dev)
131     REFERENCES DEVELOPER_BASE(Dev_ID) ON DELETE CASCADE,
132   -- Chiave esterna verso SOFTWARE_PROJECT
133   -- Eliminando un progetto, vengono eliminate anche tutte le associazioni di sviluppatori
134   -- ad esso.
135   CONSTRAINT devAssfk2 FOREIGN KEY(Project_ID)
136     REFERENCES SOFTWARE_PROJECT(Project_ID) ON DELETE CASCADE,
137   -- Ciascuno sviluppatore pu essere assegnato ad un dato progetto una sola volta.
138   CONSTRAINT unique_DevAssign UNIQUE(DEV,PROJECT_ID)
139 );
140
141 /
```

Codice 4.3: Estratto da `createDatabase.sql` - Definizione della tabella `DEV_ASSIGN`

4.2.4 Definizione della tabella **RELEASE**

```
143
144 /**
145  * TABELLA: RELEASE
146  * Crea  la tabella e implementa i vincoli pi semplici.
147  */
148
149
150 -- Crea la tabella RELEASE
151 CREATE TABLE RELEASE
152 (
153   Release_ID    INTEGER       DEFAULT 0,
154   Directory     VARCHAR2(64)  NOT NULL,
155   Codename      VARCHAR2(64)  NULL,
156   StartDate     TIMESTAMP     DEFAULT SYSTIMESTAMP NOT NULL,
157   isComplete    CHAR          DEFAULT 'F' NOT NULL CHECK(isComplete IN ('T','F')),
158   ReleaseDate   TIMESTAMP     DEFAULT NULL,
159   Project_ID    INTEGER       NOT NULL
160 );
161
162 /
163
164 -- Aggiunge i vincoli
165 ALTER TABLE RELEASE
166 ADD(
167   -- Chiave primaria
168   CONSTRAINT rel_pk PRIMARY KEY(Release_ID),
169   -- Chiave esterna verso SOFTWARE_PROJECT
170   -- Se un progetto viene cancellato, vengono cancellate anche
171   -- tutte le sue release.
172   CONSTRAINT rel_fk FOREIGN KEY(Project_ID)
173     REFERENCES SOFTWARE_PROJECT(Project_ID) ON DELETE CASCADE,
174   -- Non possono esistere nello stesso progetto due release con
175   -- directory uguale.
176   CONSTRAINT unique_RelDir UNIQUE (Directory, Project_ID)
177 );
178
179 /
180
181 -- Trigger per settare, se necessario, la chiave primaria automaticamente
182 CREATE OR REPLACE TRIGGER ReleasePK
183 BEFORE INSERT ON RELEASE
184 FOR EACH ROW
185 BEGIN
186   DECLARE
187     pk      RELEASE.RELEASE_ID%TYPE;
188   BEGIN
189     IF(:NEW.Release_ID = 0) THEN
190       SELECT NVL(MAX(Release_ID),0) + 1 INTO pk FROM RELEASE;
191       :NEW.Release_ID := pk;
192     END IF;
193   END;
194 END;
195
196 /
```

Codice 4.4: Estratto da `createDatabase.sql` - Definizione della tabella `RELEASE`

4.2.5 Definizione della tabella **PACKAGE**

```
198
199 /**
200  * TABELLA: PACKAGE
201  * Crea  la tabella e implementa i vincoli pi semplici.
202  */
203
204
205 -- Crea la tabella PACKAGE
206 CREATE TABLE PACKAGE
207 (
208   Package_ID    INTEGER       DEFAULT 0,
209   Name          VARCHAR2(64)  NOT NULL,
210   isNew         CHAR          DEFAULT 'F' NOT NULL CHECK(isNew IN ('T','F')),
211   willBeDeleted CHAR          DEFAULT 'F' NOT NULL CHECK(willBeDeleted IN ('T','F')),
212   isModified    CHAR          DEFAULT 'F' NOT NULL CHECK(isModified IN ('T','F')),
213   Description   VARCHAR(512)  DEFAULT 'Nessuna descrizione inserita.' NOT NULL,
214   Release       INTEGER       NOT NULL
215 );
216
217 /
218
219 -- Aggiunge i vincoli
220 ALTER TABLE PACKAGE
221 ADD(
222   -- Chiave primaria
223   CONSTRAINT pack_pk PRIMARY KEY(Package_ID),
224   -- Chiave esterna verso Release
225   CONSTRAINT pack_fk FOREIGN KEY(Release) REFERENCES RELEASE(Release_ID),
226   -- Non possono esistere due package con lo stesso nome nella stessa release.
227   CONSTRAINT unique_PackDir UNIQUE(Name, Release)
228 );
229
230 /
231
232 -- Trigger per settare, se necessario, la chiave primaria automaticamente
233 CREATE OR REPLACE TRIGGER PackagePK
234 BEFORE INSERT ON PACKAGE
235 FOR EACH ROW
236 BEGIN
237   DECLARE
238     pk PACKAGE.PACKAGE_ID%TYPE;
239   BEGIN
240     IF(:NEW.Package_ID = 0) THEN
241       SELECT NVL(MAX(Package_ID),0) + 1 INTO pk FROM PACKAGE;
242       :NEW.Package_ID := pk;
243     END IF;
244   END;
245 END;
246
247 /
```

Codice 4.5: Estratto da `createDatabase.sql` - Definizione della tabella `PACKAGE`

4.2.6 Definizione della tabella **CLASS_BASE**

```
249
250 /**
251  * TABELLA: CLASS
252  * Crea  la tabella e implementa i vincoli pi semplici.
253  */
254
255
256 -- Crea la tabella CLASS_BASE
257 CREATE TABLE CLASS_BASE
258 (
259   Class_ID      INTEGER       DEFAULT 0,
260   Name          VARCHAR2(64)  NOT NULL,
261   Scope         VARCHAR2(16)  DEFAULT 'DEFAULT' NOT NULL,
262   Filename      VARCHAR2(64)  NOT NULL,
263   Path          VARCHAR2(256),
264   LineStart     INTEGER       NOT NULL CHECK(LineStart>=0),
265   Offset        INTEGER       NOT NULL CHECK(Offset>0),
266   isNew         CHAR          DEFAULT 'F' NOT NULL CHECK(isNew IN ('T','F')),
267   willBeDeleted CHAR          DEFAULT 'F' NOT NULL CHECK(willBeDeleted IN ('T','F')),
268   isChanged     CHAR          DEFAULT 'F' NOT NULL CHECK(isChanged IN ('T','F')),
269   changedMethod CHAR          DEFAULT 'F' NOT NULL CHECK(changedMethod IN ('T','F')),
270   isAbstract    CHAR          DEFAULT 'F' NOT NULL CHECK(isAbstract IN ('T','F')),
271   Description   VARCHAR(512)  DEFAULT 'Nessuna descrizione inserita.' NOT NULL,
272   Package_ID    INTEGER       NOT NULL
273 );
274
275 /
276
277 -- Aggiunge i vincoli
278 ALTER TABLE CLASS_BASE
279 ADD(
280   -- Chiave primaria
281   CONSTRAINT class_pk PRIMARY KEY(Class_ID),
282   -- Chiave esterna verso PACKAGE
283   CONSTRAINT class_fk FOREIGN KEY(Package_ID) REFERENCES PACKAGE(Package_ID),
284   -- Vincolo di dominio su Scope
285   CONSTRAINT ClassScope CHECK (
286                         UPPER(Scope) IN ('PUBLIC','PRIVATE','DEFAULT','PROTECTED')
287                         ),
288   -- In un package non possono esistere due classi con lo stesso nome
289   CONSTRAINT unique_ClassPack UNIQUE(Name, Package_ID)
290 );
291
292 /
293
294 -- Crea la vista utilizzata nel trigger Check_Valid_Class
295 CREATE OR REPLACE VIEW CLASS AS SELECT * FROM CLASS_BASE;
296
297 /
298
299 -- La chiave primaria viene settata nel trigger "Check_Valid_Class"
```

Codice 4.6: Estratto da `createDatabase.sql` - Definizione della tabella `CLASS_BASE`

4.2.7 Definizione della tabella **ATTRIBUTE**

```
302
303 /**
304  * TABELLA: ATTRIBUTE
305  * Crea  la tabella e implementa i vincoli pi semplici.
306  */
307
308
309 -- Crea la tabella ATTRIBUTE
310 CREATE TABLE ATTRIBUTE
311 (
312   Attr_ID     INTEGER       DEFAULT 0,
313   Name        VARCHAR2(64)  NOT NULL,
314   Type_T      VARCHAR2(64)  NOT NULL,
315   Scope_T     VARCHAR2(16)  DEFAULT 'DEFAULT' NOT NULL,
316   Class       INTEGER       NOT NULL
317 );
318
319 /
320
321 -- Aggiunge i vincoli
322 ALTER TABLE ATTRIBUTE
323 ADD(
324   -- Chiave primaria Attr_ID
325   CONSTRAINT attr_pk PRIMARY KEY(Attr_ID),
326   -- Chiave esterna verso CLASS
327   -- Se una classe viene cancellata, vengono cancellati anche tutti i suoi attributi
328   CONSTRAINT attr_fk FOREIGN KEY(Class) REFERENCES CLASS_BASE(Class_ID) ON DELETE CASCADE,
329   -- Non possono esistere due attributi con identificatore uguale nella stessa classe
330   CONSTRAINT unique_ClassAttr UNIQUE(Name, Class),
331   -- Vincolo di dominio su Scope_T
332   CONSTRAINT AttrScope CHECK (UPPER(Scope_T) IN ('PUBLIC','PRIVATE','DEFAULT','PROTECTED'))
333 );
334
335 /
336
337 -- Trigger per settare, se necessario, la chiave primaria automaticamente
338 CREATE OR REPLACE TRIGGER AttributePK
339 BEFORE INSERT ON ATTRIBUTE
340 FOR EACH ROW
341 BEGIN
342   DECLARE
343     pk ATTRIBUTE.ATTR_ID%TYPE;
344   BEGIN
345     IF(:NEW.Attr_ID = 0) THEN
346       SELECT NVL(MAX(Attr_ID),0) + 1 INTO pk FROM ATTRIBUTE;
347       :NEW.Attr_ID := pk;
348     END IF;
349   END;
350 END;
351
352 /
```

Codice 4.7: Estratto da `createDatabase.sql` - Definizione della tabella `ATTRIBUTE`

4.2.8 Definizione della tabella **METHOD**

```
354
355 /**
356  * TABELLA: METHOD
357  * Crea  la tabella e implementa i vincoli pi semplici.
358  */
359
360
361 CREATE TABLE METHOD_BASE
362 (
363   Method_ID     INTEGER       DEFAULT 0,
364   Scope_T       VARCHAR2(16)  DEFAULT 'DEFAULT' NOT NULL,
365   Name          VARCHAR2(64)  NOT NULL,
366   TypeOut       VARCHAR2(64)  NOT NULL,
367   LineStart     INTEGER       NOT NULL CHECK(LineStart>0),
368   Offset        INTEGER       NOT NULL CHECK(Offset>0),
369   isNew         CHAR          DEFAULT 'F' NOT NULL CHECK(isNew IN ('T','F')),
370   willBeDeleted CHAR          DEFAULT 'F' NOT NULL CHECK(willBeDeleted IN ('T','F')),
371   newImplem     CHAR          DEFAULT 'F' NOT NULL CHECK(newImplem IN ('T','F')),
372   newSignature  CHAR          DEFAULT 'F' NOT NULL CHECK(newSignature IN ('T','F')),
373   Description   VARCHAR(512)  DEFAULT 'Nessuna descrizione inserita.' NOT NULL,
374   Class         INTEGER       NOT NULL
375 );
376
377 /
378
379 -- Aggiunge i vincoli
380 ALTER TABLE METHOD_BASE
381 ADD(
382   -- Chiave primaria (meth_pk)
383   CONSTRAINT meth_pk PRIMARY KEY(Method_ID),
384   -- Chiave esterno verso CLASS
385   -- Se una classe viene cancellata vengono cancellati anche tutti i suoi metodi
386   CONSTRAINT meth_fk FOREIGN KEY(Class) REFERENCES CLASS_BASE(Class_ID) ON DELETE CASCADE,
387   -- Vincolo di dominio su Scope_T
388   CONSTRAINT methScope CHECK (
389                               UPPER(Scope_T) IN ('PUBLIC','PRIVATE','DEFAULT','PROTECTED')
390                               )
391 );
392
393 /
394
395 CREATE OR REPLACE VIEW METHOD AS SELECT * FROM METHOD_BASE;
396
397 /
398
399 --La chiave primaria viene eventualmente calcolata nel trigger check_valid_method
```

Codice 4.8: Estratto da `createDatabase.sql` - Definizione della tabella `METHOD`

4.2.9 Definizione della tabella **PARAMETER**

```
402
403 /**
404  * TABELLA: PARAMETER
405  * Crea  la tabella e implementa i vincoli pi semplici.
406  */
407
408
409 -- Crea la tabella PARAMETER
410 CREATE TABLE PARAMETER
411 (
412   Param_ID    INTEGER       DEFAULT 0,
413   Name        VARCHAR2(64)  NOT NULL,
414   POSITION    INTEGER       NOT NULL CHECK(POSITION>0),
415   Type_T      VARCHAR2(64)  NOT NULL,
416   Method      INTEGER       NOT NULL
417 );
418
419 /
420
421 -- Aggiunge i vincoli
422 ALTER TABLE PARAMETER
423 ADD (
424   -- Chiave primaria (Param_ID)
425   CONSTRAINT param_pk PRIMARY KEY(Param_ID),
426   -- Chiave esterna verso METHOD
427   -- Se un metodo viene cancellato, vengono cancellati tutti i suoi parametri
428   CONSTRAINT param_fk FOREIGN KEY(Method) REFERENCES METHOD_BASE(Method_ID) ON DELETE CASCADE
      ,
429   -- Non possono esservi pi parametri di un metodo nella stessa posizione
430   CONSTRAINT unique_pos     UNIQUE(METHOD,POSITION)
431 );
432
433 /
434
435 -- Trigger per settare, se necessario, la chiave primaria automaticamente
436 CREATE OR REPLACE TRIGGER ParameterPK
437 BEFORE INSERT ON PARAMETER
438 FOR EACH ROW
439 BEGIN
440   DECLARE
441     pk PARAMETER.PARAM_ID%TYPE;
442   BEGIN
443     IF(:NEW.Param_ID = 0) THEN
444       SELECT NVL(MAX(Param_ID),0) + 1 INTO pk FROM PARAMETER;
445       :NEW.Param_ID := pk;
446     END IF;
447   END;
448 END;
449
450 /
```

Codice 4.9: Estratto da `createDatabase.sql` - Definizione della tabella `PARAMETER`

4.2.10 Definizione della tabella **RELEASE_ISSUE**

```
452
453 /**
454  * TABELLA: RELEASE_ISSUE
455  * Crea  la tabella e implementa i vincoli pi semplici.
456  */
457
458
459 -- Crea la tabella RELEASE_ISSUE
460 CREATE TABLE RELEASE_ISSUE
461 (
462   Issue_ID      INTEGER       DEFAULT 0,
463   Name          VARCHAR2(64)  NOT NULL,
464   Details       VARCHAR2(512) NOT NULL,
465   OpenDate      TIMESTAMP     DEFAULT SYSDATE NOT NULL,
466   ClosedDate    TIMESTAMP     NULL,
467   Release       INTEGER       NOT NULL,
468   Dev           INTEGER       DEFAULT NULL
469 );
470
471 /
472
473 -- Aggiunge i vincoli
474 ALTER TABLE RELEASE_ISSUE
475 ADD (
476   -- Chiave primaria (Issue_ID)
477   CONSTRAINT RelIssue_pk PRIMARY KEY(Issue_ID),
478   -- Chiave esterna verso RELEASE
479   -- Se una release viene cancellata vengono cancellate anche tutte le issues relative
480   CONSTRAINT RelIssue_fk FOREIGN KEY(Release) REFERENCES RELEASE(Release_ID)
481                          ON DELETE CASCADE,
482   -- Chiave esterna verso DEVELOPER
483   -- Se uno sviluppatore viene cancellato, tutte le issues a lui assegnate vengono
484   -- assegnate a null.
485   CONSTRAINT RelIssue_fk_dev FOREIGN KEY(Dev) REFERENCES DEVELOPER_BASE(Dev_ID)
486                              ON DELETE SET NULL,
487   -- Vincolo "Time Consistency of Issues" (si veda documentazione)
488   CONSTRAINT RelI_Time CHECK (ClosedDate IS NULL OR ClosedDate>OpenDate)
489 );
490
491 /
492
493 -- Trigger per settare, se necessario, la chiave primaria automaticamente
494 CREATE OR REPLACE TRIGGER Release_IssuePK
495 BEFORE INSERT ON RELEASE_ISSUE
496 FOR EACH ROW
497 BEGIN
498   DECLARE
499     pk RELEASE_ISSUE.ISSUE_ID%TYPE;
500   BEGIN
501     IF(:NEW.Issue_ID = 0) THEN
502       SELECT NVL(MAX(Issue_ID),0) + 1 INTO pk FROM RELEASE_ISSUE;
503       :NEW.Issue_ID := pk;
504     END IF;
505   END;
506 END;
507
508 /
```

Codice 4.10: Estratto da `createDatabase.sql` - Definizione della tabella `RELEASE_ISSUE`

4.2.11 Definizione della tabella **PACKAGE_ISSUE**

```
510
511 /**
512  * TABELLA: PACKAGE_ISSUE
513  * Crea  la tabella e implementa i vincoli pi semplici.
514  */
515
516 -- Crea la tabella PACKAGE_ISSUE
517 CREATE TABLE PACKAGE_ISSUE
518 (
519   Issue_ID      INTEGER       DEFAULT 0,
520   Name          VARCHAR2(64)  NOT NULL,
521   Details       VARCHAR2(512) NOT NULL,
522   OpenDate      TIMESTAMP     DEFAULT SYSDATE NOT NULL,
523   ClosedDate    TIMESTAMP     NULL,
524   Package_ID    INTEGER       NOT NULL,
525   Dev           INTEGER       DEFAULT NULL
526 );
527
528 /
529
530 -- Aggiunge i vincoli
531 ALTER TABLE PACKAGE_ISSUE
532 ADD (
533   -- Chiave primaria (Issue_ID)
534   CONSTRAINT PackIssue_pk PRIMARY KEY(Issue_ID),
535   -- Chiave esterna verso PACKAGE
536   -- Se un package viene cancellato vengono cancellate anche tutte le issues relative
537   CONSTRAINT PackIssue_fk FOREIGN KEY(Package_ID) REFERENCES PACKAGE(Package_ID)
538                           ON DELETE CASCADE,
539   -- Chiave esterna verso DEVELOPER
540   -- Se uno sviluppatore viene cancellato, tutte le issues a lui assegnate vengono
541   -- assegnate a null.
542   CONSTRAINT PackIssue_fk_dev FOREIGN KEY(Dev) REFERENCES DEVELOPER_BASE(Dev_ID)
543                               ON DELETE SET NULL,
544   -- Vincolo "Time Consistency of Issues" (si veda documentazione)
545   CONSTRAINT PackI_Time CHECK (ClosedDate IS NULL OR ClosedDate>OpenDate)
546 );
547
548 /
549
550 -- Trigger per settare, se necessario, la chiave primaria automaticamente
551 CREATE OR REPLACE TRIGGER Package_IssuePK
552 BEFORE INSERT ON PACKAGE_ISSUE
553 FOR EACH ROW
554 BEGIN
555   DECLARE
556     pk PACKAGE_ISSUE.ISSUE_ID%TYPE;
557   BEGIN
558     IF(:NEW.Issue_ID = 0) THEN
559       SELECT NVL(MAX(Issue_ID),0) + 1 INTO pk FROM PACKAGE_ISSUE;
560       :NEW.Issue_ID := pk;
561     END IF;
562   END;
563 END;
564
565 /
```

Codice 4.11: Estratto da `createDatabase.sql` - Definizione della tabella `PACKAGE_ISSUE`

4.2.12 Definizione della tabella **CLASS_ISSUE**

```
567
568 /**
569  * TABELLA: CLASS_ISSUE
570  * Crea  la tabella e implementa i vincoli pi semplici.
571  */
572
573
574 -- Crea la tabella CLASS_ISSUE
575 CREATE TABLE CLASS_ISSUE
576 (
577   Issue_ID      INTEGER       DEFAULT 0,
578   Name          VARCHAR2(64)  NOT NULL,
579   Details       VARCHAR2(512) NOT NULL,
580   OpenDate      TIMESTAMP     DEFAULT SYSDATE NOT NULL,
581   ClosedDate    TIMESTAMP     NULL,
582   Class         INTEGER       NOT NULL,
583   Dev           INTEGER       DEFAULT NULL
584 );
585
586 /
587
588 -- Aggiunge i vincoli
589 ALTER TABLE CLASS_ISSUE
590 ADD (
591   -- Chiave primaria (Issue_ID)
592   CONSTRAINT ClassIssue_pk PRIMARY KEY(Issue_ID),
593   -- Chiave esterna verso CLASS
594   -- Se una classe viene cancellata vengono cancellate anche tutte le issues relative
595   CONSTRAINT ClassIssue_fk FOREIGN KEY(Class) REFERENCES CLASS_BASE(Class_ID)
596                           ON DELETE CASCADE,
597   -- Chiave esterna verso DEVELOPER
598   -- Se uno sviluppatore viene cancellato, tutte le issues a lui assegnate vengono
599   -- assegnate a null.
600   CONSTRAINT ClassIssue_fk_dev FOREIGN KEY(Dev) REFERENCES DEVELOPER_BASE(Dev_ID)
601                               ON DELETE SET NULL,
602   -- Vincolo "Time Consistency of Issues" (si veda documentazione)
603   CONSTRAINT ClassI_Time CHECK (ClosedDate IS NULL OR ClosedDate>OpenDate)
604 );
605
606 /
607
608 -- Trigger per settare, se necessario, la chiave primaria automaticamente
609 CREATE OR REPLACE TRIGGER Class_IssuePK
610 BEFORE INSERT ON CLASS_ISSUE
611 FOR EACH ROW
612 BEGIN
613   DECLARE
614     pk CLASS_ISSUE.ISSUE_ID%TYPE;
615   BEGIN
616     IF(:NEW.Issue_ID = 0) THEN
617       SELECT NVL(MAX(Issue_ID),0) + 1 INTO pk FROM CLASS_ISSUE;
618       :NEW.Issue_ID := pk;
619     END IF;
620   END;
621 END;
622
623 /
```

Codice 4.12: Estratto da `createDatabase.sql` - Definizione della tabella `CLASS_ISSUE`

4.2.13 Definizione della tabella **METHOD_ISSUE**

```
625
626 /**
627 * TABELLA: METHOD_ISSUE
628 * Crea  la tabella e implementa i vincoli pi semplici.
629 */
630
631
632 -- Crea la tabella METHOD_ISSUE
633 CREATE TABLE METHOD_ISSUE
634 (
635   Issue_ID      INTEGER       DEFAULT 0,
636   Name          VARCHAR2(64)  NOT NULL,
637   Details       VARCHAR2(512) NOT NULL,
638   OpenDate      TIMESTAMP     DEFAULT SYSDATE NOT NULL,
639   ClosedDate    TIMESTAMP     NULL,
640   Method        INTEGER       NOT NULL,
641   Dev           INTEGER       DEFAULT NULL
642 );
643
644 /
645
646 -- Aggiunge i vincoli
647 ALTER TABLE METHOD_ISSUE
648 ADD (
649   -- Chiave primaria (Issue_ID)
650   CONSTRAINT MethIssue_pk PRIMARY KEY(Issue_ID),
651   -- Chiave esterna verso METHOD
652   -- Se un metodo viene cancellato vengono cancellate anche tutte le issues relative
653   CONSTRAINT MethIssue_fk FOREIGN KEY(Method) REFERENCES METHOD_BASE(Method_ID)
654                           ON DELETE CASCADE,
655   -- Chiave esterna verso DEVELOPER
656   -- Se uno sviluppatore viene cancellato, tutte le issues a lui assegnate vengono
657   -- assegnate a null.
658   CONSTRAINT MethIssue_fk_dev FOREIGN KEY(Dev) REFERENCES DEVELOPER_BASE(Dev_ID)
659                               ON DELETE SET NULL,
660   -- Vincolo "Time Consistency of Issues" (si veda documentazione)
661   CONSTRAINT MethI_Time CHECK (ClosedDate IS NULL OR ClosedDate>OpenDate)
662 );
663
664 /
665
666 -- Trigger per settare, se necessario, la chiave primaria automaticamente
667 CREATE OR REPLACE TRIGGER Method_IssuePK
668 BEFORE INSERT ON METHOD_ISSUE
669 FOR EACH ROW
670 BEGIN
671   DECLARE
672     pk METHOD_ISSUE.ISSUE_ID%TYPE;
673   BEGIN
674     IF(:NEW.Issue_ID = 0) THEN
675       SELECT NVL(MAX(Issue_ID),0) + 1 INTO pk FROM METHOD_ISSUE;
676       :NEW.Issue_ID := pk;
677     END IF;
678   END;
679 END;
680
681 /
```

Codice 4.13: Estratto da `createDatabase.sql` - Definizione della tabella `METHOD_ISSUE`

4.2.14 Definizione della tabella **COMMIT_T**

```
683
684 /**
685 * TABELLA: COMMIT_T
686 * Crea  la tabella e implementa i vincoli pi semplici.
687 */
688
689
690 -- Crea la tabella COMMIT_T
691 CREATE TABLE COMMIT_T
692 (
693   Commit_ID   INTEGER       DEFAULT 0,
694   Name        VARCHAR2(64)  NOT NULL,
695   Details     VARCHAR2(512) NOT NULL,
696   Class       INTEGER       NOT NULL,
697   Dev         INTEGER,
698   insDate     TIMESTAMP     DEFAULT SYSDATE NOT NULL
699 );
700
701 /
702
703 -- Aggiunge i vincoli
704 ALTER TABLE COMMIT_T
705 ADD (
706   -- Chiave primaria (Commit_ID)
707   CONSTRAINT com_pk PRIMARY KEY(Commit_ID),
708   -- Chiave esterna verso CLASS
709   -- Se una classe viene cancellata, vengono cancellate anche tutte le modifiche
710   -- ad essa apportate
711   CONSTRAINT com_fk FOREIGN KEY(Class) REFERENCES CLASS_BASE(Class_ID) ON DELETE CASCADE,
712   -- Chiave esterna verso DEVELOPER
713   -- Se uno sviluppatore viene cancellato, tutte le modifiche da lui apportate
714   -- vengono attribuite a NULL.
715   CONSTRAINT com_fk_dev FOREIGN KEY(Dev) REFERENCES DEVELOPER_BASE(Dev_ID) ON DELETE SET NULL
716 );
717
718 /
719
720 -- Trigger per settare, se necessario, la chiave primaria automaticamente
721 CREATE OR REPLACE TRIGGER Commit_TPK
722 BEFORE INSERT ON COMMIT_T
723 FOR EACH ROW
724 BEGIN
725   DECLARE
726     pk COMMIT_T.COMMIT_ID%TYPE;
727   BEGIN
728     IF(:NEW.Commit_ID = 0) THEN
729       SELECT NVL(MAX(Commit_ID),0) + 1 INTO pk FROM COMMIT_T;
730       :NEW.Commit_ID := pk;
731     END IF;
732   END;
733 END;
734
735 /
```

Codice 4.14: Estratto da `createDatabase.sql` - Definizione della tabella `COMMIT_T`

4.2.15 Definizione della tabella **COMMIT_METH**

```
737
738 /**
739 * TABELLA: COMMIT_METH
740 * Crea  la tabella e implementa i vincoli pi semplici.
741 */
742
743
744 -- Crea la tabella COMMIT_METH
745 CREATE TABLE COMMIT_METH
746 (
747   Commit_ID   INTEGER   NOT NULL,
748   Method      INTEGER   NOT NULL
749 );
750
751 /
752
753 -- Aggiunge i vincoli
754 ALTER TABLE COMMIT_METH
755 ADD (
756   -- Chiave esterna verso COMMIT_T
757   -- Se viene cancellato un commit vengono cancellate anche le associazioni di modifica
758   -- a metodi che il commit cancellato apportava.
759   CONSTRAINT commit_meth_fk1 FOREIGN KEY(Commit_ID) REFERENCES COMMIT_T(Commit_ID)
760                             ON DELETE CASCADE,
761   -- Chiave esterna verso METHOD
762   -- Se viene cancellato un metodo vengono cancellate anche tutte le istanze di modifica
763   -- a quel metodo
764   CONSTRAINT commit_meth_fk2 FOREIGN KEY(Method) REFERENCES METHOD_BASE(Method_ID)
765                             ON DELETE CASCADE
766 );
767
768 /
```

Codice 4.15: Estratto da `createDatabase.sql` - Definizione della tabella `COMMIT_METH`

4.2.16 Definizione della tabella **REFINE**

```
769
770 /**
771 * TABELLA: REFINE
772 * Crea  la tabella e implementa i vincoli pi semplici.
773 */
774
775
776 -- Crea la tabella REFINE
777 CREATE TABLE REFINE
778 (
779   SubClass    INTEGER   NOT NULL UNIQUE, -- no erediterietà multipla in JAVA!
780   SupClass    INTEGER   NOT NULL
781 );
782
783 /
784
785 -- Aggiunge i vincoli
786 ALTER TABLE REFINE
787 ADD (
788   -- Chiave esterna verso CLASS
789   -- Se viene cancellata una classe vengono cancellate anche tutte le istanze
790   -- di REFINE in cui essa  sottoclasse
791   CONSTRAINT refine_fk1 FOREIGN KEY(SubClass) REFERENCES CLASS_BASE(Class_ID) ON DELETE
      CASCADE,
792   -- Chiave esterna verso CLASS
793   -- Se viene cancellata una classe vengono cancellate anche tutte le istanze
794   -- di REFINE in cui essa  superclasse
795   CONSTRAINT refine_fk2 FOREIGN KEY(SupClass) REFERENCES CLASS_BASE(Class_ID) ON DELETE
      CASCADE,
796   -- Vincolo No Auto-Refine
797   -- Una classe non pu estendere se stessa.
798   CONSTRAINT "NO_AUTO_REFINE" CHECK (SUPCLASS <> SUBCLASS)
799 );
800
801 /
```

Codice 4.16: Estratto da `createDatabase.sql` - Definizione della tabella `REFINE`

4.2.17 Definizione della tabella `BASICTYPE`

La seguente tabella rappresenterà l'omonima enumerazione dello schema concettuale. Contestualmente alla dichiarazione è mostrato anche il popolamento.

```
803
804 /**
805  * TABELLA: BASICTYPES
806  * Crea  la tabella e la popola
807  */
808
809 -- Creazione tabella
810 CREATE TABLE BASICTYPE
811 (
812   Name      VARCHAR2(16)   NOT NULL
813 );
814
815 /
816
817 -- Popolamento con tipi predefiniti
818 INSERT ALL
819   INTO BASICTYPE VALUES ('VOID')
820   INTO BASICTYPE VALUES ('BYTE')
821   INTO BASICTYPE VALUES ('SHORT')
822   INTO BASICTYPE VALUES ('INT')
823   INTO BASICTYPE VALUES ('LONG')
824   INTO BASICTYPE VALUES ('FLOAT')
825   INTO BASICTYPE VALUES ('DOUBLE')
826   INTO BASICTYPE VALUES ('BOOLEAN')
827   INTO BASICTYPE VALUES ('CHAR')
828   INTO BASICTYPE VALUES ('STRING')
829 SELECT * FROM DUAL;
830
831 /
```

Codice 4.17: Estratto da `createDatabase.sql` - Definizione della tabella `BASICTYPE`

4.3 Funzioni, procedure ed altre automazioni

4.3.1 Calcolo automatico del path di una classe

La stored function `calculate_path()` calcola, dati in input un filename ed un id di un pacchetto, il percorso completo al file sorgente. Di seguito è riportata la definizione:

```
882
883 -- Funzione calculate_path. Calcola il percorso completo del file sorgente filename nel
        package
884 -- con ID p. La funzione viene usata nel trigger CALC_PATH, per calcolare automaticamente il
        percorso
885 -- nel file system della classe appena inserita.
886 CREATE OR REPLACE FUNCTION calculate_path(filename CLASS.FILENAME%TYPE, p CLASS.PACKAGE_ID%
        TYPE)
887 RETURN VARCHAR2
888 IS
889   path VARCHAR2(256);
890 BEGIN
891   SELECT '/'||PR.DIRECTORY||'/'||RE.DIRECTORY||'/src/'||REPLACE(PA.NAME,'.','/') INTO path
892   FROM (PACKAGE PA JOIN RELEASE RE ON PA.RELEASE = RE.RELEASE_ID) JOIN SOFTWARE_PROJECT PR ON
         RE.PROJECT_ID = PR.PROJECT_ID
893   WHERE PA.PACKAGE_ID = p;
894   path := path || '/' || filename || '.java';
895   return path;
896 END;
897
898 /
```

Codice 4.18: Estratto da `createDatabase.sql` - Definizione della funzione `calculate_path()`

4.3.2 Individuare lo sviluppatore cui assegnare una issue

La stored function `find_available_dev()`, dato in input un id di un progetto, ritorna l'id dello sviluppatore abilitato a lavorare a quel progetto che abbia il minor numero di issue ancora irrisolte. Può essere utilizzata per individuare automaticamente a quale sviluppatore assegnare una issue.

```
899
900 -- Funzione find_available_dev. Individua lo sviluppatore abilitato al progetto proj con meno
        issues irrisolte.
901 CREATE OR REPLACE FUNCTION find_available_dev(proj SOFTWARE_PROJECT.PROJECT_ID%TYPE)
902 RETURN DEVELOPER.DEV_ID%TYPE
903 IS
904   dev   DEVELOPER.DEV_ID%TYPE;
905 BEGIN
906   FOR dev_cur IN(
907       SELECT D.Dev_ID AS DEVEL
908       FROM DEVELOPER D FULL OUTER JOIN ALL_ISSUES A ON D.Dev_ID = A.Dev
909       WHERE D.Dev_ID IN (SELECT DEV FROM DEV_ASSIGN WHERE PROJECT_ID = proj) AND (A.Issue_ID
      IS NULL OR A.ClosedDate IS NULL)
910       GROUP BY D.Dev_ID
911       ORDER BY COUNT(A.ISSUE_ID) ASC
912     ) LOOP
913       dev := dev_cur.DEVEL; -- assegno lo sviluppatore abilitato con meno issue irrisolte
```

Codice 4.19: Estratto da `createDatabase.sql` - Definizione della funzione `find_available_dev()`

4.3.3 Individuare il progetto a cui appartiene un certo elemento

Le funzioni `get_project_method()`, `get_project_class()`, `get_project_package()`, `get_project_release()` restituiscono, preso in input l'id di un oggetto appropriato, l'id del progetto a cui questo appartiene. Di seguito si riporta la loro definizione.

```
920
921 -- Dato un id di metodo, ritorna l'id del progetto a cui il metodo appartiene
922 CREATE OR REPLACE FUNCTION get_project_method(method METHOD.METHOD_ID%TYPE)
923 RETURN SOFTWARE_PROJECT.PROJECT_ID%TYPE
924 IS
925   project   SOFTWARE_PROJECT.PROJECT_ID%TYPE;
926 BEGIN
927   SELECT R.PROJECT_ID INTO project
928   FROM ((METHOD M JOIN CLASS C ON M.CLASS = C.CLASS_ID AND M.METHOD_ID = method) JOIN PACKAGE
        P ON P.PACKAGE_ID = C.PACKAGE_ID ) JOIN RELEASE R ON P.RELEASE = R.RELEASE_ID;
929   RETURN project;
930 END;
931
932 /
933
934 -- Dato un id di classe, ritorna l'id del progetto a cui la classe appartiene
935 CREATE OR REPLACE FUNCTION get_project_class(class CLASS.CLASS_ID%TYPE)
936 RETURN SOFTWARE_PROJECT.PROJECT_ID%TYPE
937 IS
938   project   SOFTWARE_PROJECT.PROJECT_ID%TYPE;
939 BEGIN
940   SELECT R.PROJECT_ID INTO project
941   FROM (CLASS C JOIN PACKAGE P ON P.PACKAGE_ID = C.PACKAGE_ID AND C.CLASS_ID = class) JOIN
       RELEASE R ON P.RELEASE = R.RELEASE_ID;
942   RETURN project;
943 END;
944
945 /
946
947 -- Dato un id di package, ritorna l'id del package a cui il metodo appartiene
948 CREATE OR REPLACE FUNCTION get_project_package(pack PACKAGE.PACKAGE_ID%TYPE)
949 RETURN SOFTWARE_PROJECT.PROJECT_ID%TYPE
950 IS
951   project   SOFTWARE_PROJECT.PROJECT_ID%TYPE;
952 BEGIN
953   SELECT R.PROJECT_ID INTO project
954   FROM PACKAGE P JOIN RELEASE R ON P.RELEASE = R.RELEASE_ID AND P.PACKAGE_ID = pack;
955   RETURN project;
956 END;
957
958 /
959
960 -- Dato un id di release, ritorna l'id della release a cui il package appartiene
961 CREATE OR REPLACE FUNCTION get_project_release(rel RELEASE.RELEASE_ID%TYPE)
962 RETURN SOFTWARE_PROJECT.PROJECT_ID%TYPE
963 IS
964   project   SOFTWARE_PROJECT.PROJECT_ID%TYPE;
965 BEGIN
966   SELECT R.PROJECT_ID INTO project
967   FROM RELEASE R
968   WHERE R.RELEASE_ID = rel;
969   RETURN project;
970 END;
971
972 /
```

Codice 4.20: Estratto da `createDatabase.sql`

4.3.4 Verificare se un tipo è valido

La funzione `isValidType()` permette di verificare, dato un nome di tipo `T` e una release `R`, se il `T` è un tipo valido in `R`. La funzione controlla se `T` è tra i tipi primitivi oppure se esiste una classe in `R` con lo stesso nome. La funzione ritorna il carattere `'T'` in caso di esito del controllo positivo e `'F'` altrimenti. Di seguito si riporta la definizione.

```
973
974 -- Dato un nome del tipo e una release, ritorna 'T' se quel tipo  valido (cio
975 -- se  in BASICTYPES oppure se c' una classe con lo stesso nome nella stessa
976 -- release) o 'F' altrimenti.
977 CREATE OR REPLACE FUNCTION isValidType(typename VARCHAR2, rel RELEASE.RELEASE_ID%TYPE)
978 RETURN CHAR
979 IS
980   found INTEGER;
981 BEGIN
982   SELECT COUNT(*) INTO found --controlla se il tipo  primitivo
983   FROM  BASICTYPE
984   WHERE Name=UPPER(typename);
985   IF FOUND <> 0 THEN
986     RETURN 'T';
987   END IF;
988   SELECT COUNT(*) INTO found
989   FROM CLASSNAMES
990   WHERE CLASSNAMES.NAME = typename AND RELEASE_ID = rel;
991   IF FOUND <> 0 THEN
992     RETURN 'T';
993   ELSE
994     RETURN 'F';
995   END IF;
996 END;
997
998 /
```

Codice 4.21: Estratto da `createDatabase.sql` - Definizione della funzione `isValidType()`

4.3.5 Calcolare la segnatura di un metodo

La funzione `get_method_signature()`, dato un id di un metodo, ritorna una stringa che ne rappresenta la segnatura completa nel classico formato `typeOut metodo(typeP1 P1,..., typePn Pn)`.

```
 999
1000 -- Dato l'id di un metodo, la funzione ne ritorna la segnatura completa come stringa
1001 -- nel formato intuitivo typeout methodName( type1 par1, type2 par2, ...).
1002 CREATE OR REPLACE FUNCTION get_method_signature(meth METHOD.METHOD_ID%TYPE)
1003 RETURN VARCHAR2
1004 IS
1005   signature VARCHAR2(4096);
1006   methInfo  METHOD%ROWTYPE;
1007 BEGIN
1008   SELECT * INTO methInfo
1009   FROM METHOD M
1010   WHERE M.METHOD_ID = meth;
1011   signature := methInfo.TYPEOUT || ' ' || methInfo.NAME || '(';
1012   FOR cur IN (SELECT * FROM PARAMETER WHERE METHOD = meth ORDER BY POSITION)
1013   LOOP
1014     signature := signature || cur.TYPE_T || ' ' || cur.NAME || ', ';
1015   END LOOP;
1016   signature := TRIM(TRAILING ' ' FROM signature);
1017   signature := TRIM(TRAILING ',' FROM signature);
1018   signature := signature || ')';
1019   return signature;
1020 END;
1021
1022 /
```

Codice 4.22: Estratto da `createDatabase.sql` - Definizione della funzione `get_method_signature()`

4.3.6 Calcolo automatico del path di un file sorgente

Il trigger `calc_path`, usando la funzione `calculate_path()`, calcola automaticamente il valore del campo `Path` al momento dell'inserimento/aggiornamento di dati nella tabella `CLASS_BASE`.

```
1043
1044 -- Trigger calc_path. Calcola il valore di PATH prima dell'inserimento di un nuovo record.
1045 CREATE OR REPLACE TRIGGER calc_path
1046 BEFORE INSERT OR UPDATE ON CLASS_BASE
1047 FOR EACH ROW
1048 BEGIN
1049   BEGIN
1050     :NEW.PATH := CALCULATE_PATH(:NEW.FILENAME, :NEW.PACKAGE_ID);
1051   END;
1052 END;
1053
1054 /
```

Codice 4.23: Estratto da `createDatabase.sql` - Definizione del trigger `calc_path`

4.4 Viste

4.4.1 Vista `ALL_ISSUES`

Questa vista, oltre a fornire un comodo quadro riepilogativo delle issue, viene utilizzata nella stored function `find_available_dev()`.

```
836
837 -- Vista riepilogativa di tutte le issues presenti.
838 CREATE VIEW ALL_ISSUES(ISSUE_TYPE, ISSUE_ID, NAME, OPENDATE, CLOSEDDATE,DEV) AS
839 SELECT * FROM(
840   SELECT 'Package Issue', P.Issue_ID, P.Name, P.OpenDate, P.ClosedDate, P.Dev
841   FROM PACKAGE_ISSUE P
842     UNION
843   SELECT 'Release Issue', R.Issue_ID, R.Name, R.OpenDate, R.ClosedDate, R.Dev
844   FROM RELEASE_ISSUE R
845     UNION
846   SELECT 'Class Issue', C.Issue_ID, C.Name, C.OpenDate, C.ClosedDate, C.Dev
847   FROM CLASS_ISSUE C
848     UNION
849   SELECT 'Method Issue', M.Issue_ID, M.Name, M.OpenDate, M.ClosedDate, M.Dev
850   FROM METHOD_ISSUE M
851   )
852 ORDER BY OPENDATE ASC;
853
854 /
```

Codice 4.24: Estratto da `createDatabase.sql` - Definizione della vista `ALL_ISSUES`

4.4.2 Vista `CLASSNAMES`

La vista mostra, per ciascuna release, i nomi delle classi che vi appertengono. Viene utilizzata nella stored function `isValidType()` per l'implementazione del vincolo `Type Values`.

```
855
856 -- Vista che mostra tutti i nomi di classe con la relativa release.
857 CREATE VIEW ClassNames AS
858 SELECT R.Release_ID, CL.Name
859 FROM (RELEASE R JOIN PACKAGE PK ON PK.Release=R.Release_ID) JOIN CLASS CL ON CL.Package_ID=PK
        .PACKAGE_ID;
860
861 /
```

Codice 4.25: Estratto da `createDatabase.sql` - Definizione della vista `CLASSNAMES`

4.4.3 Vista ISSUE_SUMMARY

La seguente vista mostra informazioni riepilogative sulle issue instanziate: per ciascuna release riporta il nome del progetto a cui appartiene ed il conteggio delle issue ai livelli di release, package, classe, metodo.

```
862
863 -- Vista riepilogativa del numero delle issue aperte per ciascuna release
864 -- di ciascun progetto
865 CREATE OR REPLACE VIEW ISSUE_SUMMARY AS
866 SELECT S.Name AS PROJ, R.Directory AS RELEASE, COUNT(DISTINCT RI.ISSUE_ID) AS REL_ISSUE,
867        COUNT( DISTINCT PI.ISSUE_ID) AS PACK_ISSUE, COUNT(DISTINCT CI.ISSUE_ID) AS CL_ISSUE,
868        COUNT(DISTINCT MI.ISSUE_ID) AS METH_ISSUE
869 FROM (((((((SOFTWARE_PROJECT S JOIN RELEASE R ON R.Project_ID=S.Project_ID)
870       JOIN PACKAGE P ON P.Release=R.Release_ID) JOIN CLASS C ON C.Package_ID=P.Package_ID)
871       JOIN METHOD M ON M.Class=C.Class_ID) FULL OUTER JOIN CLASS_ISSUE CI ON CI.Class=C.
      Class_ID ))
872       LEFT OUTER JOIN RELEASE_ISSUE RI ON RI.Release=R.Release_ID )
873       LEFT OUTER JOIN PACKAGE_ISSUE PI ON PI.Package_ID=P.Package_ID)
874       LEFT OUTER JOIN METHOD_ISSUE MI ON MI.Method=M.Method_ID
875 GROUP BY S.Name, R.Directory;
876
877 /
```

Codice 4.26: Estratto da `createDatabase.sql` - Definizione della vista `ISSUE_SUMMARY`

4.4.4 Vista METHOD_SUMMARY

Mostra informazioni riepilogative sui metodi nel repository. In particolare mostra scope e segnatura completa, nonchè informazioni relative a classe, package, release e progetto corrispondenti. Utilizza la stored function `get_method_signature()`.

```
1027
1028 -- Fornisce una visione riepilogativa delle segnature dei metodi di ciascuna classe
1029 -- di ciascun package di ciascuna release di ciasun progetto.
1030 CREATE OR REPLACE VIEW METHOD_SUMMARY AS
1031 SELECT S.NAME AS PROJECT, R.DIRECTORY AS RELEASE, P.NAME AS PACKAGE, C.NAME AS CLASS, M.
      SCOPE_T || ' ' || GET_METHOD_SIGNATURE(M.METHOD_ID) AS METHOD
1032 FROM (((SOFTWARE_PROJECT S JOIN RELEASE R ON R.PROJECT_ID = S.PROJECT_ID) JOIN
1033       PACKAGE P ON P.RELEASE = R.RELEASE_ID) JOIN
1034       CLASS C ON C.PACKAGE_ID = P.PACKAGE_ID) JOIN
1035       METHOD M ON M.CLASS = C.CLASS_ID;
1036
1037 /
```

Codice 4.27: Estratto da `createDatabase.sql` - Definizione della vista `METHOD_SUMMARY`

4.5 Implementazione dei vincoli

Di seguito sono riportate le implementazioni dei vincoli che non sono già stati mostrati nella definizione delle tabelle.

4.5.1 Implementazione del vincolo `Distinct Personal Mails`

```
1059
1060 -------------------------------------------------------------------------------
1061 -- Vincolo    : "Distinct Personal Mail"
1062 -- Descrizione: Lo stesso indirizzo email non pu essere utilizzato da pi
1063 --              sviluppatori.
1064 -- Note       : Calcola ed imposta anche la chiave primaria, se necessario.
1065 -------------------------------------------------------------------------------
1066
1067
1068 CREATE OR REPLACE TRIGGER checkEmails
1069 INSTEAD OF INSERT OR UPDATE ON DEVELOPER
1070 FOR EACH ROW
1071 BEGIN
1072   DECLARE
1073     TEMP INTEGER:=0;
1074     pk   DEVELOPER.DEV_ID%TYPE;
1075   BEGIN
1076     IF INSERTING THEN -- effettua l'inserimento
1077       SELECT COUNT(*) INTO TEMP
1078       FROM DEVELOPER_BASE D
1079       WHERE (D.eMail=:NEW.eMail OR D.eMail2=:NEW.eMail OR
1080             D.eMail=:NEW.eMail2 OR D.eMail2=:NEW.eMail2);
1081       IF(TEMP <> 0) THEN -- Violazioni presenti
1082         RAISE_APPLICATION_ERROR(-20001,'Indirizzo eMail gi presente!');
1083       ELSE
1084         SELECT NVL(MAX(Dev_ID),0)+1 INTO pk FROM DEVELOPER_BASE;
1085         INSERT INTO DEVELOPER_BASE (DEV_ID,FIRSTNAME,LASTNAME,EMAIL,EMAIL2)
1086         VALUES (pk, :NEW.FIRSTNAME, :NEW.LASTNAME, :NEW.EMAIL, :NEW.EMAIL2);
1087       END IF;
1088     ELSE -- UPDATING
1089       SELECT COUNT(*) INTO TEMP
1090       FROM DEVELOPER_BASE D
1091       WHERE D.Dev_ID <> :OLD.DEV_ID AND (D.eMail=:NEW.eMail OR D.eMail2=:NEW.eMail OR
1092             D.eMail=:NEW.eMail2 OR D.eMail2=:NEW.eMail2);
1093       IF(TEMP <> 0) THEN -- Violazioni presenti
1094         RAISE_APPLICATION_ERROR(-20001,'Indirizzo eMail gi presente!');
1095       ELSE
1096         UPDATE DEVELOPER_BASE SET
1097         FIRSTNAME = :NEW.FIRSTNAME,
1098         LASTNAME  = :NEW.LASTNAME,
1099         EMAIL     = :NEW.EMAIL,
1100         EMAIL2    = :NEW.EMAIL2
1101         WHERE DEV_ID = :OLD.DEV_ID;
1102       END IF;
1103     END IF;
1104   END;
1105 END;
1106
1107 /
```

Codice 4.28: Estratto da `createDatabase.sql` - Definizione del trigger `checkEmails`

4.5.2 Implementazione del vincolo **Type Values**

Il vincolo è implementato con diversi trigger

```
1108
1109 ------------------------------------------------------------------------------
1110 -- Vincolo    : "Type Values"
1111 -- Descrizione: I tipi degli attributi, dei parametri formali dei metodi ed il
1112 --              tipo del valore ritornato da un metodo devono essere valori di
1113 --              in BasicTypes oppure nomi di classi della stessa release.
1114 -- Note       : Implementato sulle tabelle ATTRIBUTE, METHOD, PARAMETER con i
1115 --              seguenti tre trigger.
1116 ------------------------------------------------------------------------------
1117
1118 -- Implementa il vincolo per TYPE_T di ATTRIBUTE
1119 CREATE OR REPLACE TRIGGER checkType_attr
1120 BEFORE INSERT OR UPDATE ON ATTRIBUTE
1121 FOR EACH ROW
1122 BEGIN
1123   DECLARE
1124     rel RELEASE.RELEASE_ID%TYPE;
1125   BEGIN
1126     SELECT PK.RELEASE INTO rel
1127     FROM CLASS CL JOIN PACKAGE PK ON CL.PACKAGE_ID = PK.PACKAGE_ID
1128     WHERE CL.CLASS_ID = :NEW.CLASS;
1129     IF isValidType(:NEW.TYPE_T, rel) = 'F' THEN
1130       RAISE_APPLICATION_ERROR(-20002,'Tipo non valido!');
1131     END IF;
1132   END;
1133 END;
1134
1135 /
1136
1137 -- Implementa il vincolo per TYPEOUT di METHOD
1138 CREATE OR REPLACE TRIGGER checkType_meth
1139 BEFORE INSERT OR UPDATE ON METHOD_BASE
1140 FOR EACH ROW
1141 BEGIN
1142   DECLARE
1143     rel RELEASE.RELEASE_ID%TYPE;
1144   BEGIN
1145     SELECT PK.RELEASE INTO rel
1146     FROM CLASS CL JOIN PACKAGE PK ON CL.PACKAGE_ID = PK.PACKAGE_ID
1147     WHERE CL.CLASS_ID = :NEW.CLASS;
1148     IF isValidType(:NEW.TYPEOUT, rel) = 'F' THEN
1149       RAISE_APPLICATION_ERROR(-20002,'Tipo non valido!');
1150     END IF;
1151   END;
1152 END;
1153
1154 /
1155
1156 -- Implementa il vincolo per TYPE_T di PARAMETER
1157 CREATE OR REPLACE TRIGGER checkType_param
1158 BEFORE INSERT OR UPDATE ON PARAMETER
1159 FOR EACH ROW
1160 BEGIN
1161   DECLARE
1162     rel RELEASE.RELEASE_ID%TYPE;
1163   BEGIN
1164     SELECT PK.RELEASE INTO rel
1165     FROM (METHOD M JOIN CLASS CL ON M.CLASS = CL.CLASS_ID) JOIN PACKAGE PK ON CL.PACKAGE_ID =
        PK.PACKAGE_ID
1166     WHERE M.METHOD_ID = :NEW.METHOD;
1167     IF isValidType(:NEW.TYPE_T, rel) = 'F' THEN
1168       RAISE_APPLICATION_ERROR(-20002,'Tipo non valido!');
1169     END IF;
```

```
1170   END;
1171 END;
1172
1173 /
```

Codice 4.29: Estratto da `createDatabase.sql`

4.5.3 Implementazione del vincolo **Assigned Devs Commit**

```
1311
1312 -------------------------------------------------------------------------------
1313 -- Vincolo    : "Assigned Devs Commit"
1314 -- Descrizione: Soltanto sviluppatori assegnati ad un progetto possono effettuare
1315 --              commit sugli elementi di quel progetto.
1316 -------------------------------------------------------------------------------
1317
1318 CREATE OR REPLACE TRIGGER check_dev_commit
1319 BEFORE INSERT ON COMMIT_T
1320 FOR EACH ROW
1321 BEGIN
1322   DECLARE
1323     VIOLATION INTEGER:=0;
1324     PROJECT_KEY SOFTWARE_PROJECT.PROJECT_ID%TYPE;
1325   BEGIN
1326     SELECT R.PROJECT_ID INTO PROJECT_KEY
1327     FROM (CLASS CL JOIN PACKAGE PK ON CL.Package_ID=PK.Package_ID)
1328          JOIN RELEASE R ON PK.Release=R.Release_ID
1329     WHERE CL.Class_ID=:NEW.Class;
1330     SELECT COUNT(*) INTO VIOLATION
1331     FROM DEV_ASSIGN
1332     WHERE DEV=:NEW.Dev AND PROJECT_ID=PROJECT_KEY;
1333     IF (VIOLATION = 0) THEN
1334       RAISE_APPLICATION_ERROR(-20007,'Sviluppatore non abilitato ad eseguire commit sul
       progetto.');
1335     END IF;
1336   END;
1337 END;
1338
1339 /
```

Codice 4.30: Estratto da `createDatabase.sql` - Definizione del trigger `CHECK_VALID_METHOD`

4.5.4 Implementazione dei vincoli **Single Public Class per File** e **File structure consistency**

```
1174
1175 -------------------------------------------------------------------------
1176 -- Vincolo    : "Single Public Class per File"
1177 -- Descrizione: Ogni file sorgente contiene alpi una classe con scope PUBLIC.
1178 -------------------------------------------------------------------------
1179 -- Vincolo    : "File structure consistency"
1180 -- Descrizione: In ciascun file sorgente, non possono sovrapporsi definizioni di
1181 --              classi diverse.
1182 -------------------------------------------------------------------------
1183 -- Note       : Necessitano della definizione di una vista di appoggio per non
1184 --              incorrere in errori causati da mutating tables. Il seguente
1185 --              trigger implementa entrambi i vincoli.
1186 -------------------------------------------------------------------------
1187
1188
1189 CREATE OR REPLACE TRIGGER check_Valid_Class
1190 INSTEAD OF INSERT OR UPDATE ON CLASS
1191 FOR EACH ROW
1192 BEGIN
1193   DECLARE
1194     VIOLATION INTEGER:=0;
1195     pk        CLASS.CLASS_ID%TYPE;
1196   BEGIN
1197     -- Verifico se sussistono violazioni
1198     -- Conto le classi public nello stesso file sorgente e nello stesso package
1199     IF UPDATING THEN
1200       SELECT COUNT(*) INTO VIOLATION
1201       FROM CLASS_BASE C
1202       WHERE C.PACKAGE_ID = :NEW.PACKAGE_ID AND C.FILENAME = :NEW.FILENAME AND
1203             C.SCOPE = 'PUBLIC' AND C.CLASS_ID <> :OLD.CLASS_ID ;
1204     ELSE -- Sto inserendo un nuovo record
1205       SELECT COUNT(*) INTO VIOLATION
1206       FROM CLASS_BASE C
1207       WHERE C.PACKAGE_ID = :NEW.PACKAGE_ID AND C.FILENAME = :NEW.FILENAME AND
1208             C.SCOPE = 'PUBLIC';
1209     END IF;
1210     IF( VIOLATION > 0 AND :NEW.SCOPE = 'PUBLIC') THEN -- c' gi una classe public in quel file
       in quel package
1211       RAISE_APPLICATION_ERROR(-20003,'In_un_file_sorgente_pu_esserci_un''unica_classe_public!
      ');
1212     ELSE -- non c' violazione del vincolo "Single Public Class per File"
1213       -- Verifico il vincolo "File structure consistency"
1214       SELECT COUNT (*) INTO VIOLATION
1215       FROM CLASS_BASE CL
1216       WHERE (:NEW.FILENAME = CL.FILENAME AND :NEW.PACKAGE_ID = CL.PACKAGE_ID AND :NEW.NAME <>
       CL.NAME) AND
1217             (((CL.LINESTART BETWEEN :NEW.LINESTART AND (:NEW.LINESTART + :NEW.OFFSET)) AND
1218             (CL.LINESTART + CL.OFFSET) > (:NEW.LINESTART + :NEW.OFFSET)) OR
1219             ((:NEW.LINESTART BETWEEN CL.LINESTART AND (CL.LINESTART + CL.OFFSET)) AND
1220             (:NEW.LINESTART + :NEW.OFFSET) > (CL.LINESTART + CL.OFFSET)));
1221       IF (VIOLATION > 0) THEN
1222         RAISE_APPLICATION_ERROR(-20004,'Violata_corretta_strutturazione_dei_blocchi_del_file.
      ');
1223       ELSE -- Nessuna violazione. Proseguo
1224         IF INSERTING THEN -- effettua l'inserimento
1225           SELECT NVL(MAX(CLASS_ID),0)+1 INTO pk FROM CLASS_BASE;
1226           INSERT INTO CLASS_BASE (CLASS_ID,NAME,SCOPE,FILENAME,PATH,LINESTART,OFFSET,ISNEW,
      WILLBEDELETED,ISCHANGED,CHANGEDMETHOD,ISABSTRACT,DESCRIPTION,PACKAGE_ID)
1227           VALUES (pk,:NEW.NAME,:NEW.SCOPE,:NEW.FILENAME,:NEW.PATH,:NEW.LINESTART,:NEW.OFFSET,
      :NEW.ISNEW,:NEW.WILLBEDELETED,:NEW.ISCHANGED,:NEW.CHANGEDMETHOD,:NEW.ISABSTRACT,:NEW.
      DESCRIPTION,:NEW.PACKAGE_ID);
1228         ELSE -- UPDATING
1229           UPDATE CLASS_BASE SET
```

```
1230          NAME = :NEW.NAME,
1231          SCOPE = :NEW.SCOPE,
1232          FILENAME = :NEW.FILENAME,
1233          PATH = :NEW.PATH,
1234          LINESTART = :NEW.LINESTART,
1235          OFFSET = :NEW.OFFSET,
1236          ISNEW = :NEW.ISNEW,
1237          WILLBEDELETED = :NEW.WILLBEDELETED,
1238          ISCHANGED = :NEW.ISCHANGED,
1239          CHANGEDMETHOD = :NEW.CHANGEDMETHOD,
1240          ISABSTRACT = :NEW.ISABSTRACT,
1241          DESCRIPTION = :NEW.DESCRIPTION,
1242          PACKAGE_ID = :NEW.PACKAGE_ID
1243          WHERE CLASS_ID = :OLD.CLASS_ID;
1244        END IF;
1245      END IF;
1246    END IF;
1247  END;
1248 END;
1249
1250 /
```

Codice 4.31: Estratto da `createDatabase.sql` - Definizione del trigger `CHECK_VALID_CLASS`

4.5.5 Implementazione del vincolo **File consistency for methods**

```
1251
1252 -- Vincolo    : "File consistency for methods"
1253 -- Descrizione: In ciascun file sorgente, non possono sovrapporsi definizioni di
1254 --               metodi diversi. Inoltre, un la definizione di un metodo non pu
1255 --               trovarsi al di fuori di quella della classe a cui appartiene.
1256 -- Note       : Necessita di definire una vista di appoggio.
1257 --------------------------------------------------------------------------------
1258
1259 CREATE OR REPLACE TRIGGER CHECK_VALID_METHOD
1260 INSTEAD OF INSERT OR UPDATE ON METHOD
1261 FOR EACH ROW
1262 BEGIN
1263   DECLARE
1264     classInfo CLASS%ROWTYPE;
1265     pk        METHOD.METHOD_ID%TYPE;
1266     VIOLATION INTEGER;
1267   BEGIN
1268     -- recupero informazioni relative alla classe cui il metodo appartiene
1269     SELECT * INTO classInfo
1270     FROM CLASS
1271     WHERE CLASS_ID = :NEW.CLASS;
1272     -- verifico se il metodo  definito al di fuori della classe
1273     IF( :NEW.LINESTART < classInfo.LINESTART OR
1274       (:NEW.LINESTART + :NEW.OFFSET) > (classInfo.LINESTART + classInfo.OFFSET)) THEN
1275       RAISE_APPLICATION_ERROR(-20005,'Il metodo  definito al di fuori della classe a cui
      appartiene');
1276     ELSE -- verifico se il metodo si sovrappone ad altri metodi della stessa classe
1277       SELECT COUNT(*) INTO VIOLATION
1278       FROM METHOD_BASE M
1279       WHERE M.CLASS = classInfo.CLASS_ID AND (
1280             M.LINESTART BETWEEN (:NEW.LINESTART+1) AND (:NEW.LINESTART + :NEW.OFFSET)-1 OR
1281             :NEW.LINESTART BETWEEN (M.LINESTART+1) AND (M.LINESTART + M.OFFSET)-1 );
1282       IF VIOLATION > 0 THEN
1283         RAISE_APPLICATION_ERROR(-20006, 'Il metodo si sovrappone ad altri metodi della stessa
      classe');
1284       ELSE
1285         IF INSERTING THEN
1286           SELECT NVL(MAX(METHOD_ID),0)+1 INTO pk FROM METHOD_BASE;
1287           INSERT INTO METHOD_BASE (METHOD_ID,SCOPE_T, NAME, TYPEOUT, LINESTART, OFFSET, ISNEW
      , WILLBEDELETED, NEWIMPLEM, NEWSIGNATURE, DESCRIPTION, CLASS)
1288           VALUES(pk,:NEW.SCOPE_T, :NEW.NAME, :NEW.TYPEOUT, :NEW.LINESTART, :NEW.OFFSET, :NEW.
      ISNEW, :NEW.WILLBEDELETED, :NEW.NEWIMPLEM, :NEW.NEWSIGNATURE, :NEW.DESCRIPTION, :NEW.
      CLASS);
1289         ELSE --UPDATING
1290           UPDATE METHOD_BASE SET
1291           SCOPE_T = :NEW.SCOPE_T,
1292           NAME = :NEW.NAME,
1293           TYPEOUT = :NEW.TYPEOUT,
1294           LINESTART = :NEW.LINESTART,
1295           OFFSET = :NEW.OFFSET,
1296           ISNEW = :NEW.ISNEW,
1297           WILLBEDELETED = :NEW.WILLBEDELETED,
1298           NEWIMPLEM = :NEW.NEWIMPLEM,
1299           NEWSIGNATURE = :NEW.NEWSIGNATURE,
1300           DESCRIPTION = :NEW.DESCRIPTION,
1301           CLASS = :NEW.CLASS
1302           WHERE METHOD_ID = :OLD.METHOD_ID;
1303         END IF;
1304       END IF;
1305     END IF;
1306   END;
1307 END;
```

Codice 4.32: Estratto da `createDatabase.sql` - Definizione del trigger `CHECK_VALID_METHOD`

4.5.6 Implementazione del vincolo **Commit consistency**

```
1340
1341 --------------------------------------------------------------------------------
1342 -- Vincolo    : "Commit consistency"
1343 -- Descrizione: Ciascun commit interessa una ed una sola classe. I metodi
1344 --              associati a ciascun commit devono essere metodi di quella classe.
1345 --------------------------------------------------------------------------------
1346
1347 CREATE OR REPLACE TRIGGER CHECK_COMMIT_CONSISTENCY
1348 BEFORE INSERT OR UPDATE ON COMMIT_METH
1349 FOR EACH ROW
1350 BEGIN
1351   DECLARE
1352     commit_class COMMIT_T.CLASS%TYPE;
1353     method_class METHOD.CLASS%TYPE;
1354   BEGIN
1355     SELECT CLASS INTO commit_class
1356     FROM COMMIT_T
1357     WHERE COMMIT_ID = :NEW.COMMIT_ID;
1358     SELECT CLASS INTO method_class
1359     FROM METHOD
1360     WHERE METHOD_ID = :NEW.METHOD;
1361     IF commit_class <> method_class THEN
1362       RAISE_APPLICATION_ERROR(-20008, 'Il metodo non appartiene alla classe interessata dal
      commit');
1363     END IF;
1364   END;
1365 END;
1366
1367 /
```

Codice 4.33: Estratto da `createDatabase.sql` - Definizione del trigger `CHECK_COMMIT_CONSISTENCY`

Capitolo 5

Esempio d'uso

5.1 Popolamento con dati fittizi

Codice 5.1: Script per il popolamento della base di dati

```
1  ------------------------------------------------------
2  -- SCRIPT PER IL POPOLAMENTO DELLA BASE DI DATI
3  ------------------------------------------------------
4  -- Universit degli Studi di Napoli Federico II
5  -- Insegnamento di Basi di Dati e Sistemi Informativi
6  --
7  -- Progetto : TRACCIA 1 - Software Repository
8  --            con Versioning e Gestione Issues
9  --
10 -- Candidato: Luigi Libero Lucio Starace - N86/1404
11 --            lui.starace@studenti.unina.it
12 --
13 ------------------------------------------------------
14 -- Questo script popola il database con dati fittizi.
15 -- Si consiglia di utilizzarlo su un database vuoto,
16 -- per evitare eventuali problemi dovuti a conflitti.
17 ------------------------------------------------------
18
19 -- SVILUPPATORI
20 INSERT INTO DEVELOPER_BASE (DEV_ID,FIRSTNAME,LASTNAME,EMAIL,EMAIL2)
21 VALUES (1,'Luigi','Starace','lui.starace@studenti.unina.it','lui.starace@azienza.it');
22 INSERT INTO DEVELOPER_BASE (DEV_ID,FIRSTNAME,LASTNAME,EMAIL,EMAIL2)
23 VALUES (2,'Marco','Rossi','marco.rossi@azienda.it',NULL);
24 INSERT INTO DEVELOPER_BASE (DEV_ID,FIRSTNAME,LASTNAME,EMAIL,EMAIL2)
25 VALUES (3,'Ciro','Bianchi','cbianchi@azienda.it',NULL);
26 INSERT INTO DEVELOPER_BASE (DEV_ID,FIRSTNAME,LASTNAME,EMAIL,EMAIL2)
27 VALUES (4,'Giovanni','Verdi','giovanniv@azienda.it',NULL);
28 INSERT INTO DEVELOPER_BASE (DEV_ID,FIRSTNAME,LASTNAME,EMAIL,EMAIL2)
29 VALUES (5,'Natalia','Esposito','natalia.espo@azienda.it','nat.es@fmail.com');
30 INSERT INTO DEVELOPER_BASE (DEV_ID,FIRSTNAME,LASTNAME,EMAIL,EMAIL2)
31 VALUES (6,'Alessandro','Romano','aleromano@azienda.it','alessandror@kmail.com');
32 INSERT INTO DEVELOPER_BASE (DEV_ID,FIRSTNAME,LASTNAME,EMAIL,EMAIL2)
33 VALUES (7,'Anna Chiara','de Martino','acdemartino@azienda.it','annachiara@fmail.com');
34 INSERT INTO DEVELOPER_BASE (DEV_ID,FIRSTNAME,LASTNAME,EMAIL,EMAIL2)
35 VALUES (8,'Michele','Guelfi','mguelfi@azienda.it',NULL);
36 INSERT INTO DEVELOPER_BASE (DEV_ID,FIRSTNAME,LASTNAME,EMAIL,EMAIL2)
37 VALUES (9,'Nick','Morris','nmorris@azienda.it',NULL);
38 INSERT INTO DEVELOPER_BASE (DEV_ID,FIRSTNAME,LASTNAME,EMAIL,EMAIL2)
39 VALUES (10,'Emma','Brown','emmab@azienda.it','emmabrown@fmail.com');
40 COMMIT;
41
42
43 -- PROGETTI
```

```
44 INSERT INTO SOFTWARE_PROJECT (PROJECT_ID,NAME,DESCRIPTION,DIRECTORY,STARTDATE,DROPDATE,LEADER
       )
45 VALUES ('1','Zoo','Programma per la gestione di
46 uno zoo.','zoo',to_timestamp('10-AGO-15 17:09:12,432000000','DD-MON-RR HH24:MI:SSXFF'),NULL,'
       1');
47 INSERT INTO SOFTWARE_PROJECT (PROJECT_ID,NAME,DESCRIPTION,DIRECTORY,STARTDATE,DROPDATE,LEADER
       )
48 VALUES ('2','Hello World','Funzionalità di Hello World di
49 livello enterprise.','hello',to_timestamp('10-AGO-15 17:10:03,741000000','DD-MON-RR HH24:MI:
       SSXFF'),NULL,'5');
50 COMMIT;
51
52 -- Assegnazioni sviluppatori - progetti
53 INSERT INTO DEV_ASSIGN (DEV,PROJECT_ID) VALUES ('1','1');
54 INSERT INTO DEV_ASSIGN (DEV,PROJECT_ID) VALUES ('2','1');
55 INSERT INTO DEV_ASSIGN (DEV,PROJECT_ID) VALUES ('3','1');
56 INSERT INTO DEV_ASSIGN (DEV,PROJECT_ID) VALUES ('4','1');
57 INSERT INTO DEV_ASSIGN (DEV,PROJECT_ID) VALUES ('5','2');
58 INSERT INTO DEV_ASSIGN (DEV,PROJECT_ID) VALUES ('6','2');
59 INSERT INTO DEV_ASSIGN (DEV,PROJECT_ID) VALUES ('7','2');
60 INSERT INTO DEV_ASSIGN (DEV,PROJECT_ID) VALUES ('8','2');
61 INSERT INTO DEV_ASSIGN (DEV,PROJECT_ID) VALUES ('9','1');
62 INSERT INTO DEV_ASSIGN (DEV,PROJECT_ID) VALUES ('10','1');
63 INSERT INTO DEV_ASSIGN (DEV,PROJECT_ID) VALUES ('6','1');
64 COMMIT;
65
66 -- RELEASE
67 -- Release del progetto "Zoo"
68 INSERT INTO RELEASE (RELEASE_ID,DIRECTORY,CODENAME,STARTDATE,ISCOMPLETE,RELEASEDATE,
       PROJECT_ID)
69 VALUES ('1','alpha','Alpha',to_timestamp('10-AGO-15 18:31:31,117000000','DD-MON-RR HH24:MI:
       SSXFF'),'F',NULL,'1');
70 -- Release del progetto "HelloWorld"
71 INSERT INTO RELEASE (RELEASE_ID,DIRECTORY,CODENAME,STARTDATE,ISCOMPLETE,RELEASEDATE,
       PROJECT_ID)
72 VALUES ('2','hello1','1.0 pro',to_timestamp('10-AGO-15 18:32:09,740000000','DD-MON-RR HH24:MI
       :SSXFF'),'F',NULL,'2');
73 COMMIT;
74
75 -- PACKAGE
76 -- Package del progetto "Zoo", release "Alpha"
77 INSERT INTO PACKAGE (PACKAGE_ID,NAME,ISNEW,WILLBEDELETED,ISMODIFIED,DESCRIPTION,RELEASE)
78 VALUES ('1','it.zoo.animali','T','F','F','Contiene tutti gli animali.','1');
79 INSERT INTO PACKAGE (PACKAGE_ID,NAME,ISNEW,WILLBEDELETED,ISMODIFIED,DESCRIPTION,RELEASE)
80 VALUES ('2','it.zoo.strutture','T','F','F','Contiene tutte le strutture dello zoo.','1');
81 -- Package del progetto "Hello World", release "1.0 pro"
82 INSERT INTO PACKAGE (PACKAGE_ID,NAME,ISNEW,WILLBEDELETED,ISMODIFIED,DESCRIPTION,RELEASE)
83 VALUES ('3','it.azienda.hello','T','F','F','Pacchetto principale.','2');
84 COMMIT;
85
86 -- CLASSI
87 -- PROGETTO "Zoo"
88 -- -- Classi del pacchetto it.zoo.animali
89 INSERT INTO CLASS_BASE (CLASS_ID,NAME,SCOPE,FILENAME,PATH,LINESTART,OFFSET,ISNEW,
       WILLBEDELETED,ISCHANGED,CHANGEDMETHOD,ISABSTRACT,DESCRIPTION,PACKAGE_ID)
90 VALUES ('1','Mammifero','PUBLIC','Mammifero','/zoo/alpha/src/it/zoo/animali/Mammifero.java','
       0','50','T','F','F','F','T','Classe astratta.','1');
91 INSERT INTO CLASS_BASE (CLASS_ID,NAME,SCOPE,FILENAME,PATH,LINESTART,OFFSET,ISNEW,
       WILLBEDELETED,ISCHANGED,CHANGEDMETHOD,ISABSTRACT,DESCRIPTION,PACKAGE_ID)
92 VALUES ('2','Leone','PUBLIC','Leone','/zoo/alpha/src/it/zoo/animali/Leone.java','1','60','T',
       'F','F','F','F','Il re della savana.','1');
93 INSERT INTO CLASS_BASE (CLASS_ID,NAME,SCOPE,FILENAME,PATH,LINESTART,OFFSET,ISNEW,
       WILLBEDELETED,ISCHANGED,CHANGEDMETHOD,ISABSTRACT,DESCRIPTION,PACKAGE_ID)
94 VALUES ('3','Gnu','PUBLIC','Gnu','/zoo/alpha/src/it/zoo/animali/Gnu.java','0','100','T','F','
       F','F','F','Mascotte della FSF.
95 Nome convenientemente breve.','1');
```

```
96 INSERT INTO CLASS_BASE (CLASS_ID,NAME,SCOPE,FILENAME,PATH,LINESTART,OFFSET,ISNEW,
       WILLBEDELETED,ISCHANGED,CHANGEDMETHOD,ISABSTRACT,DESCRIPTION,PACKAGE_ID)
97 VALUES ('4','Animale','PUBLIC','Animale','/zoo/alpha/src/it/zoo/animali/Animale.java','0','
       100','T','F','F','F','T','Classe␣astratta␣generica.','1');
98 INSERT INTO CLASS_BASE (CLASS_ID,NAME,SCOPE,FILENAME,PATH,LINESTART,OFFSET,ISNEW,
       WILLBEDELETED,ISCHANGED,CHANGEDMETHOD,ISABSTRACT,DESCRIPTION,PACKAGE_ID)
99 VALUES ('5','Rettile','PUBLIC','Rettile','/zoo/alpha/src/it/zoo/animali/Rettile.java','0','80
       ','T','F','F','F','T','Classe␣astratta.','1');
100 INSERT INTO CLASS_BASE (CLASS_ID,NAME,SCOPE,FILENAME,PATH,LINESTART,OFFSET,ISNEW,
       WILLBEDELETED,ISCHANGED,CHANGEDMETHOD,ISABSTRACT,DESCRIPTION,PACKAGE_ID)
101 VALUES ('6','Cobra','PUBLIC','Cobra','/zoo/alpha/src/it/zoo/animali/Cobra.java','0','90','T',
       'F','F','F','T','Serpente␣velenoso.','1');
102 INSERT INTO CLASS_BASE (CLASS_ID,NAME,SCOPE,FILENAME,PATH,LINESTART,OFFSET,ISNEW,
       WILLBEDELETED,ISCHANGED,CHANGEDMETHOD,ISABSTRACT,DESCRIPTION,PACKAGE_ID)
103 VALUES ('7','Varano','PUBLIC','Varano','/zoo/alpha/src/it/zoo/animali/Varano.java','0','50','
       T','F','F','F','F','Lucertola␣molto␣cresciuta.','1');
104 INSERT INTO CLASS_BASE (CLASS_ID,NAME,SCOPE,FILENAME,PATH,LINESTART,OFFSET,ISNEW,
       WILLBEDELETED,ISCHANGED,CHANGEDMETHOD,ISABSTRACT,DESCRIPTION,PACKAGE_ID)
105 VALUES ('13','Pesce','PUBLIC','Pesce','/zoo/alpha/src/it/zoo/animali/Pesce.java','0','100','T
       ','F','F','F','T','Classe␣astratta.','1');
106 -- -- Classi del pacchetto it.zoo.strutture
107 INSERT INTO CLASS_BASE (CLASS_ID,NAME,SCOPE,FILENAME,PATH,LINESTART,OFFSET,ISNEW,
       WILLBEDELETED,ISCHANGED,CHANGEDMETHOD,ISABSTRACT,DESCRIPTION,PACKAGE_ID)
108 VALUES ('8','Recinto','PUBLIC','Recinto','/zoo/alpha/src/it/zoo/strutture/Recinto.java','0','
       50','T','F','F','F','F','Zona␣recintata␣adatta␣a␣contenere
109 mammiferi.','2');
110 INSERT INTO CLASS_BASE (CLASS_ID,NAME,SCOPE,FILENAME,PATH,LINESTART,OFFSET,ISNEW,
       WILLBEDELETED,ISCHANGED,CHANGEDMETHOD,ISABSTRACT,DESCRIPTION,PACKAGE_ID)
111 VALUES ('9','Retillario','PUBLIC','Rettilario','/zoo/alpha/src/it/zoo/strutture/Rettilario.
       java','0','70','T','F','F','F','F','Contenitore␣in␣vetro␣adatto␣a␣contenere
112 rettili.','2');
113 INSERT INTO CLASS_BASE (CLASS_ID,NAME,SCOPE,FILENAME,PATH,LINESTART,OFFSET,ISNEW,
       WILLBEDELETED,ISCHANGED,CHANGEDMETHOD,ISABSTRACT,DESCRIPTION,PACKAGE_ID)
114 VALUES ('10','Acquario','PUBLIC','Acquario','/zoo/alpha/src/it/zoo/strutture/Acquario.java','
       0','50','T','F','F','F','F','Contenitore␣in␣vetro.','2');
115 INSERT INTO CLASS_BASE (CLASS_ID,NAME,SCOPE,FILENAME,PATH,LINESTART,OFFSET,ISNEW,
       WILLBEDELETED,ISCHANGED,CHANGEDMETHOD,ISABSTRACT,DESCRIPTION,PACKAGE_ID)
116 VALUES ('11','Habitat','PUBLIC','Habitat','/zoo/alpha/src/it/zoo/strutture/Habitat.java','0',
       '40','T','F','F','F','T','Generico␣Habitat.
117 Classe␣astratta.','2');
118 -- PROGETTO "Hello World"
119 -- -- Classi del pacchetto it.azienda.hello
120 INSERT INTO CLASS_BASE (CLASS_ID,NAME,SCOPE,FILENAME,PATH,LINESTART,OFFSET,ISNEW,
       WILLBEDELETED,ISCHANGED,CHANGEDMETHOD,ISABSTRACT,DESCRIPTION,PACKAGE_ID)
121 VALUES ('12','Hello','PUBLIC','Hello','/hello/hello1/src/it/azienda/hello/Hello.java','0','50
       ','T','F','F','F','F','Fornisce␣funzionalità␣di␣saluto.','3');
122 COMMIT;
123
124 -- Relazioni gerarchiche tra Classi
125 -- Progetto "Zoo", release "Alpha"
126 -- Animale
127 --     |----- Mammifero
128 --     |           |------ Leone
129 --     |           |------ Gnu
130 --     |
131 --     |----- Rettile
132 --     |           |------ Cobra
133 --     |           |------ Varano
134 --     |
135 --     |----- Pesce
136 --
137 -- Habitat
138 --     |----- Recinto
139 --     |----- Rettilario
140 --     |----- Acquario
141
```

```
142 INSERT INTO REFINE (SUBCLASS,SUPCLASS) VALUES ('1','4');
143 INSERT INTO REFINE (SUBCLASS,SUPCLASS) VALUES ('5','4');
144 INSERT INTO REFINE (SUBCLASS,SUPCLASS) VALUES ('2','1');
145 INSERT INTO REFINE (SUBCLASS,SUPCLASS) VALUES ('3','1');
146 INSERT INTO REFINE (SUBCLASS,SUPCLASS) VALUES ('6','5');
147 INSERT INTO REFINE (SUBCLASS,SUPCLASS) VALUES ('7','5');
148 INSERT INTO REFINE (SUBCLASS,SUPCLASS) VALUES ('10','11');
149 INSERT INTO REFINE (SUBCLASS,SUPCLASS) VALUES ('9','11');
150 INSERT INTO REFINE (SUBCLASS,SUPCLASS) VALUES ('8','11');
151 INSERT INTO REFINE (SUBCLASS,SUPCLASS) VALUES ('13','4');
152 COMMIT;
153
154 -- ATTRIBUTI
155 -- Attributi di it.zoo.animali.Animale
156 INSERT INTO ATTRIBUTE (ATTR_ID,NAME,TYPE_T,SCOPE_T,CLASS)
157 VALUES ('1','Nome','String','PRIVATE','4');
158 INSERT INTO ATTRIBUTE (ATTR_ID,NAME,TYPE_T,SCOPE_T,CLASS)
159 VALUES ('2','Sesso','char','PRIVATE','4');
160 --Attributi di it.zoo.strutture.Habitat
161 INSERT INTO ATTRIBUTE (ATTR_ID,NAME,TYPE_T,SCOPE_T,CLASS)
162 VALUES ('3','Nome','String','PRIVATE','11');
163 --Attributi di it.zoo.strutture.Acquario
164 INSERT INTO ATTRIBUTE (ATTR_ID,NAME,TYPE_T,SCOPE_T,CLASS)
165 VALUES ('4','Volume','double','PRIVATE','10');
166 --Attributi di it.zoo.strutture.Rettilario
167 INSERT INTO ATTRIBUTE (ATTR_ID,NAME,TYPE_T,SCOPE_T,CLASS)
168 VALUES ('5','Temperatura','double','PRIVATE','9');
169 COMMIT;
170
171 -- METODI
172 -- metodo aggiungi(Animale anim) di habitat
173 INSERT INTO METHOD_BASE (METHOD_ID,SCOPE_T,NAME,TYPEOUT,LINESTART,OFFSET,ISNEW,WILLBEDELETED,
        NEWIMPLEM,NEWSIGNATURE,DESCRIPTION,CLASS)
174 VALUES ('1','PUBLIC','aggiungi','void','20','10','T','F','F','F','Aggiunge un animale all''
        habitat.','11');
175 -- metodo saluta(String nome, String lingua) di hello
176 INSERT INTO METHOD_BASE (METHOD_ID,SCOPE_T,NAME,TYPEOUT,LINESTART,OFFSET,ISNEW,WILLBEDELETED,
        NEWIMPLEM,NEWSIGNATURE,DESCRIPTION,CLASS)
177 VALUES ('2','PUBLIC','saluta','String','10','10','T','F','F','F','Ritrorna una stringa di
        saluto.','12');
178 COMMIT;
179
180 -- PARAMETRI
181 -- metodo aggiungi()
182 INSERT INTO PARAMETER (PARAM_ID,NAME,POSITION,TYPE_T,METHOD) VALUES ('1','anim','1','Animale'
        ,'1');
183 --metodo saluta()
184 INSERT INTO PARAMETER (PARAM_ID,NAME,POSITION,TYPE_T,METHOD) VALUES ('2','persona','1','
        String','2');
185 INSERT INTO PARAMETER (PARAM_ID,NAME,POSITION,TYPE_T,METHOD) VALUES ('3','lingua','2','String
        ','2');
186
187 -- ISSUES
188 -- ISSUE DI RELEASE
189 INSERT INTO RELEASE_ISSUE (ISSUE_ID,NAME,DETAILS,OPENDATE,CLOSEDDATE,RELEASE,DEV)
190 VALUES ('1','Mancano gli insetti','Il committente vuole che vengano aggiunti
191 gli insetti.',to_timestamp('11-AGO-15 11:30:56,080000000','DD-MON-RR HH24:MI:SSXFF'),NULL,'1'
        ,'2');
192 -- ISSUE DI PACKAGE
193 INSERT INTO PACKAGE_ISSUE (ISSUE_ID,NAME,DETAILS,OPENDATE,CLOSEDDATE,PACKAGE_ID,DEV)
194 VALUES ('1','Aggiungere gabbia',' necessario aggiungere una classe "Gabbia".',to_timestamp('
        11-AGO-15 11:36:06,761000000','DD-MON-RR HH24:MI:SSXFF'),NULL,'2','4');
195 -- ISSUE DI CLASSE -- questa issue sar gestita in seguito dallo sviluppatore assegnatario con
        un commit.
196 INSERT INTO CLASS_ISSUE (ISSUE_ID,NAME,DETAILS,OPENDATE,CLOSEDDATE,CLASS,DEV)
197 VALUES ('1','Aggiungere il metodo mangia()','Il committente richiede che sia implementato
```

```
il metodo mangia(double kgDiErba).',to_timestamp('11-AGO-15 11:39:37,845000000','DD-MON-RR
    HH24:MI:SSXFF'),to_timestamp('11-AGO-15 12:00:49,291000000','DD-MON-RR HH24:MI:SSXFF'),'3
    ','10');
-- ISSUE DI METODO
INSERT INTO METHOD_ISSUE (ISSUE_ID,NAME,DETAILS,OPENDATE,CLOSEDDATE,METHOD,DEV)
VALUES ('1','Problema NullPointerException','Durante l''invocazione del metodo con parametri
"Giuseppe","Tedesco"  stata lanciata una
NullPointerException.
Problema nel codice.',to_timestamp('11-AGO-15 11:37:15,401000000','DD-MON-RR HH24:MI:SSXFF'),
    NULL,'2','8');
COMMIT;

-- Lo sviluppatore 10 a cui  stata assenata la issue di classe Gnu: "Aggiungere il metodo
    mangia()" effettua un commit per risolvere il problema

-- Innanzitutto inserisce il metodo mangia()

-- metodo mangia(double kgDiErba) di animali.Gnu -- aggiunto per risolvere issue
INSERT INTO METHOD_BASE (METHOD_ID,SCOPE_T,NAME,TYPEOUT,LINESTART,OFFSET,ISNEW,WILLBEDELETED,
    NEWIMPLEM,NEWSIGNATURE,DESCRIPTION,CLASS)
VALUES ('3','PUBLIC','mangia','void','80','10','T','F','F','F','Mangia un certo valore di kg
    di erba.','3');
-- parametro del metodo mangia
INSERT INTO PARAMETER (PARAM_ID,NAME,POSITION,TYPE_T,METHOD) VALUES ('4','kdDiErba','1','
    double','3');

-- Quindi inserisce il commit per tenere traccia della sua modifica
INSERT INTO COMMIT_T (COMMIT_ID,NAME,DETAILS,CLASS,DEV,INSDATE)
VALUES ('1','Aggiunto metodo mangia.','Questo commit chiude anche la issue di classe con ID:
    1 aperta in data 11/08/2015 11:39:37.845000.','3','10',to_timestamp('11-AGO-15
    11:48:44,560000000','DD-MON-RR HH24:MI:SSXFF'));
-- e associa al commit appena inserito il metodo aggiunto
INSERT INTO COMMIT_METH (COMMIT_ID,METHOD)
VALUES ('1','3');
COMMIT;
```

Codice 5.1: Script per il popolamento della base di dati

5.2 Esempi di query

Di seguito sono riportate alcune tra le possibili interrogazioni alla base di dati.

5.2.1 Elencare le issue di metodo ancora aperte in ordine di apertura

La sequente query mostra le issue di metodo correntemente irrisolte nel sistema elencandole in ordine ascendente di apertura. Inoltre, per ciascuna issue, vengono mostrati anche anche il nome del metodo, della classe, del pacchetto, della release e del progetto a cui è relativa.

```
SELECT  I.Name AS ISSUE, TO_CHAR(I.OpenDate, 'DD-MM-YYYY HH24:MI') AS TIMEOPEN,
        M.Name AS METODO, C.Name AS CLASSE, P.NAME AS PACKAGE,
        R.CodeName AS RELEASE, S.Name AS PROJECT
FROM ((((METHOD_ISSUE I JOIN METHOD M ON M.Method_ID=I.Method)
      JOIN CLASS C ON C.Class_ID=M.Class) JOIN PACKAGE P ON P.Package_ID=C.Package_ID)
      JOIN RELEASE R ON R.Release_ID=P.Release) JOIN SOFTWARE_PROJECT S ON S.Project_ID=R.
    Project_ID
WHERE I.isClosed='F'
ORDER BY I.openDate ASC;
```

Codice 5.2: Esempio di query

5.2.2 Selezionare lo sviluppatore che ha risolto una issue nel minor tempo

La seguente query seleziona lo sviluppatore che ha risolto una issue di qualsiasi tipo più rapidamente. Oltre all'identificativo e ai dati dello sviluppatore, l'interrogazione calcola anche il tempo impiegato per risolvere la issue (`TIME_NEEDED`) e informazioni sulla issue risolta (identificativo, nome, tipo). Il tempo impiegato per risolvere la issue è in formato `INTERVAL DAY TO SECOND`, quindi un valore di `+00 01:16:49.335000` significa che sono stati impiegati 0 giorni, 1 ora, 16 minuti, 49 secondi.

```
1 SELECT D.Dev_ID, D.FirstName, D.LastName, (AI.CLOSEDDATE-AI.OPENDATE) AS TIME_NEEDED,
2        AI.ISSUE_ID AS ID_CLOSED_ISSUE, AI.Issue_Type AS CLOSED_ISSUE_TYPE,
3        AI.Name AS CLOSED_ISSUE_NAME
4 FROM ALL_ISSUES AI JOIN DEVELOPER D ON D.Dev_ID=AI.Dev
5 WHERE AI.isClosed='T' AND
6      (AI.CLOSEDDATE-AI.OPENDATE)=( SELECT MIN(CLOSEDDATE-OPENDATE)
7                                    FROM ALL_ISSUES
8                                    WHERE isClosed='T'
9                                  );
```

Codice 5.3: Esempio di query