



Progettazione e sviluppo di una Base di dati relazionale per la gestione di un software repository con funzionalità di versioning e testing

Emanuele Rocco Petrone
N86002155

Dario Leone
N86002202

July 18, 2019

Questa pagina è stata lasciata intenzionalmente bianca.

Indice

1	Descrizione del progetto	4
1.1	Descrizione sintetica e analisi del problema	4
1.2	Il Repository Software	4
2	Progettazione Concettuale	6
2.1	Introduzione	6
2.2	Class Diagram	7
2.3	Ristrutturazione del Class Diagram	8
2.4	Analisi delle ridondanze	8
2.5	Analisi degli identificativi	8
2.6	Rimozione degli attributi multipli	8
2.7	Rimozione delle classi di associazione	8
2.8	Rimozione delle gerarchie di specializzazione	8
2.9	Class Diagram Ristrutturato	9
2.10	Dizionario delle classi	10
2.11	Dizionario delle Associazioni	13
2.12	Dizionario dei Vincoli	15
3	Progettazione Logica	17
3.1	Schema logico	17
4	Progettazione fisica	19
4.1	Definizione tabelle	19
4.1.1	Definizione della Tabella DEVELOPER	20
4.1.2	Definizione della Tabella SOFTWARE_PROJECT	21
4.1.3	Definizione della Tabella RELEASE	22
4.1.4	Definizione della Tabella PACKAGE	23
4.1.5	Definizione della Tabella CLASS	24
4.1.6	Definizione della Tabella METHOD	25
4.1.7	Definizione della Tabella PARAMETER	26
4.1.8	Definizione della Tabella ATTRIBUTE	27
4.1.9	Definizione della Tabella COMMIT	28
4.1.10	Definizione della Tabella TEST	29
4.1.11	Definizione della Tabella EXECUTED_TEST	30
4.1.12	Definizione della Tabella RELEASE_TEST	31
4.1.13	Definizione della Tabella COMMIT_METHOD	32
4.1.14	Definizione della Tabella DEV_ASSIGN	33
4.1.15	Definizione della Tabella BASIC_TYPES	34

4.2	Viste	35
4.2.1	CLASSNAMES	35
4.2.2	TESTNAMES	35
4.3	Funzioni, Procedure ed altre Automazioni	36
4.3.1	Calcolo automatico del path di una classe	36
4.3.2	Calcolo automatico del path di un file sorgente	36
4.3.3	Verificare se un tipo è valido	37
4.3.4	Verificare se un FileName è valido per un Caso di Test	38
4.4	Implementazione dei Vincoli	39
4.4.1	Implementazione del vincolo Distinct Personal Mails	39
4.4.2	Implementazione del vincolo TypeValues	40
4.4.3	Implementazione del vincolo Name Consistency for Test	41
4.4.4	Implementazione del vincolo Single Public Class per File e File structure consistency for Class	42
4.4.5	Implementazione del vincolo File consistency for methods	44
4.4.6	Implementazione del vincolo Commit consistency	45
5	Manuale d'Uso	46
5.1	Popolamento con Dati di Esempio	46
5.2	Esempio d'Uso	51
5.2.1	Inserimento Record	51

Capitolo 1

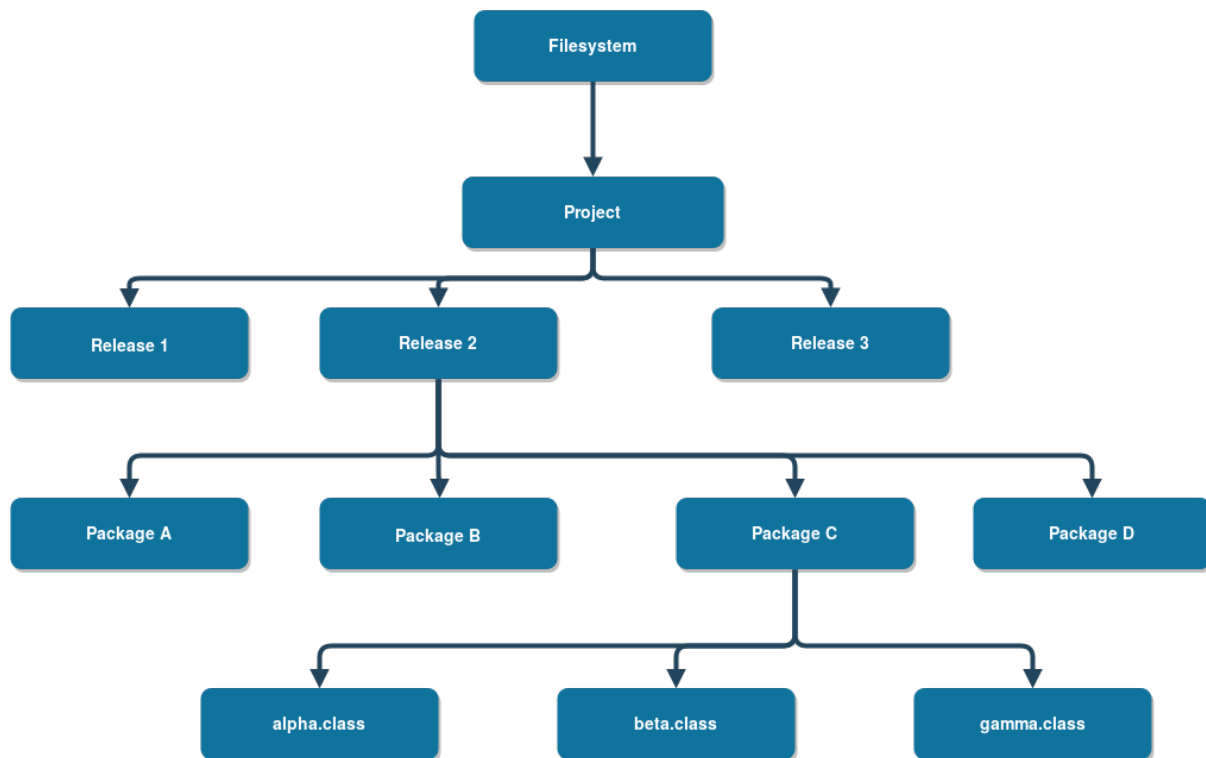
Descrizione del progetto

1.1 Descrizione sintetica e analisi del problema

Si progetterà ed implementerà una base di dati relazionale che possa essere d'aiuto alla gestione di un repository software. La base di dati conterrà i descrittori di diversi progetti software e informazione sulla loro strutturazione in packages e classi. Il sistema permetterà di associare a ciascun descrittore nei vari livelli gerarchici il file (memorizzato in un file-system) che contiene il codice sorgente, nonché informazione sugli autori del codice e sulla tempistica relativa allo sviluppo. Ciascun progetto potrà avere diverse versioni (realease) e tra elementi di release successive andrà tenuta una traccia di corrispondenza che indichi lo stato dell'elemento rispetto al suo omologo della realease precedente. In particolare tale corrispondenza indicherà se un elemento è rimasto immutato, se è stato modificato, se è stato aggiunto nella realease 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 di ospitare casi di test usati per validare il progetto, inoltre terrà traccia dell'esecuzione dei test effettuati su ogni progetto e in particolare del loro esito. Lo stesso caso di test potrà essere eseguito più volte su una stessa realease o su realease differenti di un progetto, tenendo traccia delle strutture interessate dal test.

1.2 Il Repository Software

I codici sorgente dei progetti presenti nel repository sono organizzati nel file system di appoggio in maniera gerarchica secondo la struttura del progetto stesso. Il ciclo di sviluppo e rilascio di progetti può sintetizzarsi nei seguenti passaggi:



- **Creazione di una nuova release di sviluppo.** Si procede creando una nuova cartella all'interno della cartella del progetto. In essa potranno essere copiati i sorgenti della eventuale release precedente che saranno poi modificati oppure si potrà riscrivere tutto il codice.
- **Sviluppo.** Tutti gli sviluppatori abilitati a lavorare al progetto effettuano modifiche (commit) ai sorgenti. Il sistema tiene traccia degli autori e delle tempistiche relative a dette modifiche.
- **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.
- Se lo sviluppo del progetto non viene abbandonato si torna al punto 1 e si crea una nuova release di sviluppo.

Gli sviluppatori possono istanziare test per tutte le release indipendentemente dal loro stato di completamento. Il loro esito verrà calcolato e salvato attraverso opportuni trigger in un log.

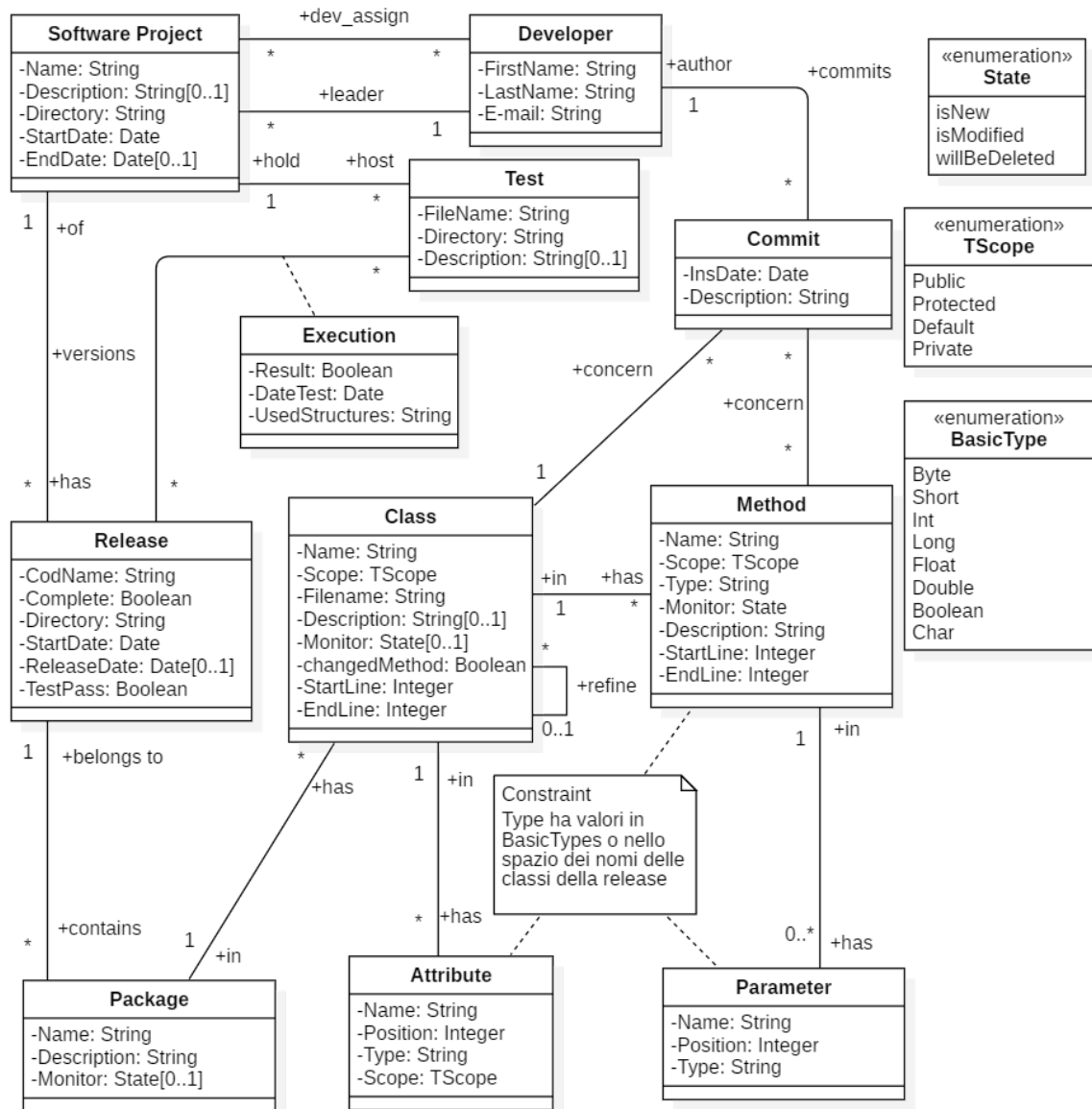
Capitolo 2

Progettazione Concettuale

2.1 Introduzione

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.

2.2 Class Diagram



2.3 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.

2.4 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.

2.5 Analisi degli identificativi

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

2.6 Rimozione degli attributi multipli

Non sono presenti attributi multipli da eliminare, anzi in fase di implementazione sarebbe utile aggiungere un ulteriore attributo **E-mail2** nella classe **DEVELOPER** in caso si voglia immettere un indirizzo e-mail secondario(facoltativo).

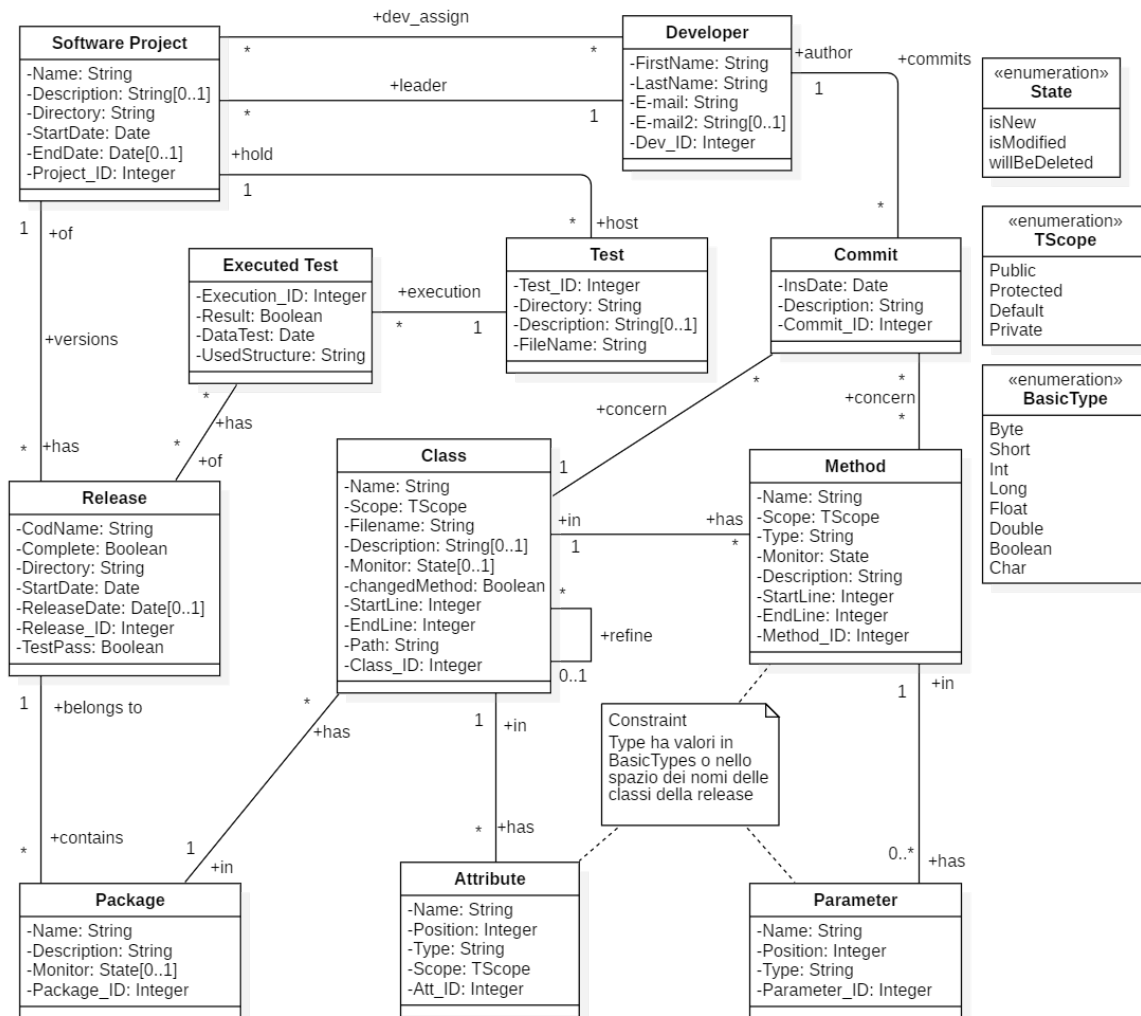
2.7 Rimozione delle classi di associazione

Si procede con l'eliminazione della classe di associazione **EXECUTION** e la reintroduzione della stessa come classe vera e propria contenente gli stessi attributi più l'aggiunta di un identificativo. Si instaureranno inoltre due associazioni, una con la classe **RELEASE** e l'altra con la classe **TEST**.

2.8 Rimozione delle gerarchie di specializzazione

Non sono presenti specializzazione nel class diagram.

2.9 Class Diagram Ristrutturato



2.10 Dizionario delle classi

Table 2.1: Dizionario delle classi

Classe	Descrizione	Attributi
Software Project	Descrittore di ciascun progetto presente nel repository software	Name (<i>string</i>): nome del progetto. Description (<i>string, opzionale</i>): breve descrizione del progetto. Directory (<i>string</i>): nome della directory radice del progetto all'interno della radice del repository. StartDate (<i>date</i>): data di creazione del progetto. EndDate (<i>date, opzionale</i>): data di fine del progetto. Project_ID (<i>integer</i>): chiave tecnica. Identifica univocamente ciascun progetto nel repository.
Release	Descrittore di ogni versione rilasciata del progetto	CodName (<i>string</i>): nome interno della release. Complete (<i>boolean</i>): indica se la release è da considerarsi completa oppure in fase di sviluppo. Directory (<i>string</i>): nome della directory radice della release all'interno della radice del progetto. StartDate (<i>date</i>): data di inizio dei lavori per la release. ReleaseDate (<i>date, opzionale</i>): indica la data (eventuale) in cui la release è stata indicata come completa. Release_ID (<i>integer</i>): chiave tecnica. identifica univocamente ciascuna release del progetto. TestPass (<i>boolean</i>): indica se la release a passato con successo tutti i casi di test del progetto.
Package	Descrittore di Package appartenenti ad una release.	Name (<i>string</i>): nome del pacchetto. Coincide con il nome della directory che contiene le classi del pacchetto. Description (<i>string</i>): breve descrizione del contenuto del Package Monitor (<i>State, opzionale</i>): indica se il pacchetto è nuovo, se è stato modificato dalla release precedente oppure verrà eliminato alla prossima release. Package_ID (<i>integer</i>): identifica univocamente ogni istanza di Package.

Classe	Descrizione	Attributi
Class	Descrittore di una classe in un progetto.	<p>Name (<i>string</i>): il nome della classe.</p> <p>Scope (<i>TScope</i>): indica lo scope della classe all'interno del programma.</p> <p>FileName (<i>string</i>): indica il nome del file sorgente all'interno del quale è definita la classe.</p> <p>Description (<i>string, opzionale</i>): breve descrizione esplicativa della classe.</p> <p>Monitor (<i>State, opzionale</i>): indica se la classe è nuova, se è stata modificata dalla release precedente oppure verrà eliminata alla prossima release.</p> <p>changedMethod (<i>boolean, derivato</i>): indica se almeno un metodo della classe ha subito modifiche all'implementazione, ha cambiato segnatura, è stato aggiunto o eliminato.</p> <p>StartLine (<i>integer</i>): indica la linea dove inizia la definizione della classe nel file .java.</p> <p>EndLine (<i>integer</i>): indica la linea dove finisce la definizione della classe.</p> <p>Path (<i>string</i>): indica il percorso completo, relativo alla directory root del repository, del file.</p> <p>Class_ID (<i>integer</i>): identifica univocamente ciascuna istanza di Class.</p>
Attribute	Descrittore di un attributo (campo) di una classe.	<p>Name (<i>string</i>): name binding dell'attributo.</p> <p>Position (<i>integer</i>): indica la posizione dell'attributo.</p> <p>Type (<i>string</i>): indica il tipo dell'attributo.</p> <p>Scope (<i>TScope</i>): indica lo scope dell'attributo.</p> <p>Att_ID (<i>integer</i>): identifica univocamente un attributo.</p>
Developer	Descrittore di sviluppatore abilitato a lavorare nel repository	<p>FirstName (<i>string</i>): nome dello sviluppatore.</p> <p>LastName (<i>string</i>): cognome dello sviluppatore.</p> <p>eMail (<i>string</i>): indirizzo email principale dello sviluppatore.</p> <p>eMail2 (<i>string</i>): indirizzo email secondario (di recupero).</p> <p>Dev_ID (<i>integer</i>).</p>

Classe	Descrizione	Attributi
Method	Descrittore del metodo di una classe.	Name (<i>string</i>): nome del metodo. Scope (<i>TScope</i>): indica lo scope del metodo. Type (<i>string</i>): indica il tipo del valore ritornato dal metodo. Monitor (<i>State</i>): breve descrizione esplicativa della classe. Monitor (<i>State, opzionale</i>): indica se il metodo è nuovo, se è stato modificato dalla release precedente oppure verrà eliminato alla prossima release. Description (<i>string</i>): breve descrizione esplicativa del metodo. StartLine (<i>integer</i>): indica la linea dove inizia la definizione del metodo nel file .java. EndLine (<i>integer</i>): indica la linea dove finisce la definizione del metodo. Method_ID (<i>integer</i>): identifica univocamente un metodo.
Parameter	Descrittore di un parametro di metodo.	Name (<i>string</i>): identificativo del parametro formale. Position (<i>integer</i>): posizione del parametro formale nella lista dei parametri formali del metodo. Type (<i>string</i>): tipo del parametro formale. Param_ID (<i>integer</i>): indentificativo del parametro.
Commit	Descrittore di un'istanza di modifica ai codici sorgente di una classe da parte di uno sviluppatore.	InsDate (<i>date</i>): marca temporale della modifica. Description (<i>string</i>): descrizione della modifica apportata. Commit_ID (<i>integer</i>): identifica univocamente ciascuna operazione di modifica ad una classe.
Test	Descrittore del caso di Test ospitato dal progetto	Test_ID (<i>integer</i>): identificativo univoco del file di test. Directory (<i>string</i>): nome directory radice dove è ospitato il caso di test. Description (<i>string, opzionale</i>): breve descrizione dei test contenuti nel file. Filename (<i>string</i>): nome del caso di test.
Executed Test	Descrittore di un caso di Test eseguito	Executio_ID (<i>integer</i>): chiave tecnica. Result (<i>boolean</i>): esito del test. DataTest (<i>date</i>): marca temporale in cui il test è eseguito. UsedStructure (<i>string</i>): informazioni sulle strutture coinvolte dal test.

2.11 Dizionario delle Associazioni

Table 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 in : indica la classe a cui appartiene un attributo Attribute [0..*] ruolo has : indica gli attributi posseduti da una classe.
Method cont.	Esprime l'appartenenza di un metodo ad una classe.	Class [1] ruolo of : 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. 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 in : 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 [0..*] ruolo assigned to : indica il progetto a cui è abilitato a lavorare uno sviluppatore. Developer [0..*] ruolo developed by : indica lo sviluppatore che ha lavorato a quel 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.

Nome	Descrizione	Classi coinvolte
Commit Class	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.
Commit Meth	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.
Commit Dev	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 lo sviluppatore che ha eseguito un commit.

2.12 Dizionario dei Vincoli

Table 2.3: Dizionario dei Vincoli

Nome Vincolo		Descrizione
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 Mail	Personal	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.
Legit Names		I nomi possono contenere solo i caratteri da A-Z o a-z oppure 0-9, il nome deve contenere almeno un carattere.
Single Public Class per file		Ogni file sorgente possiede al più una classe public . Ovvero non possono esistere due classi con scope public definite all'interno dello stesso file .java .
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 tency	Consis-	Ciascun commit può interessare una sola classe. Tutti i metodi interessati dalla modifica, quindi, devono appartenere alla stessa classe, cioè quella modificata.
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 .
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 EndDate 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 .
File Structure Consistency for Classes		In ciascun file sorgente non è possibile che esistano due classi C1 e C2 tali che valga la seguente: 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).

Nome Vincolo	Descrizione
File Consistency for Methods	Per ciascun metodo M di una classe C vale la seguente: 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.
Name Consistency for Test	Il Filename di un test deve avere valori nello spazio dei nomi delle strutture del Progetto . (Releases,Packages,Classes,Methods)
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.

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 Schema logico

SoftwareProject (Name, Description, Directory, StartDate, EndDate, Project_ID, Leader)

Leader→Developer.Dev_ID

Release (Release_ID, CodName, Complete, Directory, StartDate, ReleaseDate, TestPass, SwProject)

SwProject→SoftwareProject.Project_ID

Developer (Dev_ID, FirstName, LastName, E-mail, E-mail2)

Commit Commit(InsDate, Description, Commit_ID, Author, Class)

Author→Developer.Dev_ID, Class→Class.Class_ID

Class (Name, Scope, Filename, Description, Monitor, changed-Method, StartLine, EndLine, Path, Class_ID, Package, SuperClass)

SuperClass→Class.Class_ID, Package→Package.Package_ID

Package (Name, Description, Monitor, Package_ID, Release)

Release→Release.Release_ID

Method	(Name, Scope, Type, Monitor, Description, StartLine, End-Line, <u>Method_ID</u> , <u>Class</u>) Class→Class.Class_ID
Parameter	(Name, Position, Type, <u>Parameter_ID</u> , <u>Method</u>) Method→Method_ID
Attribute	(Name, Position, Type, Scope, <u>Att_ID</u> , <u>Class</u> Class→Class.Class_ID)
Test	(<u>Test_ID</u> , Directory, Description, FileName, <u>SwProject</u>) SwProject→SoftwareProject.Project_ID
ExecutedTest	(<u>Execution_ID</u> , Result, DataTest, UsedStructure, <u>Test</u>) Test→Test.Test_ID
ReleaseTest	(<u>ExecutedTest</u> , <u>Release</u>) ExecutedTest→ExecutedTest.Execution_ID, Release→Release.Release_ID
DevAssign	(<u>Dev</u> , <u>Software</u>) Dev→Developer.Dev_ID, Software→SoftwareProject.Project_ID
CommitMethod	(<u>Commit</u> , <u>Method</u>) Commit→Commit_ID, Method→Method.Method_ID

Capitolo 4

Progettazione fisica

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.

4.1 Definizione tabelle

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

4.1.1 Definizione della Tabella DEVELOPER

```
01 |
02 |  /**
03 |  * TABELLA: DEVELOPER
04 |  * Crea la tabella e implementa i vincoli piu' semplici.
05 |  */
06 |  -- Crea la tabella DEVELOPER
07 |  CREATE TABLE DEVELOPER_BASE
08 |  (
09 |  Dev_ID INTEGER DEFAULT '0' NOT NULL,
10 |  FirstName VARCHAR2(64) NOT NULL,
11 |  LastName VARCHAR2(64) NOT NULL,
12 |  eMail VARCHAR2(320) NOT NULL UNIQUE CHECK(eMail LIKE '_%@_%._%'),
13 |  eMail2 VARCHAR2(320) DEFAULT NULL CHECK(eMail2 LIKE '_%@_%._%' OR
14 |  eMail2 IS NULL)
15 |  -- LEGIT EMAILS
16 |  -- una email legittima puo' contare {64}@{255} caratteri.
17 |  -- In totale 320.
18 |  );
19 |  /
20 |  -- Crea il vincolo di chiave primaria
21 |  ALTER TABLE DEVELOPER_BASE
22 |  ADD CONSTRAINT Dev_pk PRIMARY KEY(Dev_ID);
23 |  /
24 |
25 |  -- Trigger per settare, se necessario, la chiave primaria
26 |  -- automaticamente
27 |  CREATE OR REPLACE TRIGGER DeveloperPK
28 |  BEFORE INSERT ON DEVELOPER_BASE
29 |  FOR EACH ROW
30 |  BEGIN
31 |  DECLARE
32 |  pk DEVELOPER_BASE.Dev_ID%TYPE;
33 |  BEGIN
34 |  IF (:NEW.Dev_ID = 0) THEN
35 |  SELECT NVL(MAX(Dev_ID),0) + 1 INTO pk FROM DEVELOPER_BASE
36 |  ;
37 |  :NEW.Dev_ID := pk;
38 |  END IF;
39 |  END;
40 |  /
41 |  -- Crea la vista utilizzata nel trigger checkEmails
42 |  CREATE OR REPLACE VIEW DEVELOPER AS SELECT * FROM DEVELOPER_BASE;
43 |  /
```

4.1.2 Definizione della Tabella SOFTWARE_PROJECT

```
44 | /**
45 |  * TABELLA: SOFTWARE_PROJECT
46 |  * Crea la tabella e implementa i vincoli più semplici.
47 |  */
48 |
49 |
50 | -- Crea la tabella SOFTWARE_PROJECT
51 | CREATE TABLE SOFTWARE_PROJECT
52 | (
53 | Project_ID INTEGER DEFAULT 0,
54 | Name VARCHAR2(64) DEFAULT 'New_Project' NOT NULL,
55 | Description VARCHAR2(256) DEFAULT NULL,
56 | Directory VARCHAR2(64) NOT NULL UNIQUE,
57 | StartDate DATE DEFAULT SYSDATE NOT NULL,
58 | EndDate DATE DEFAULT NULL,
59 | Leader INTEGER DEFAULT NULL
60 | );
61 |
62 | /
63 |
64 | -- Aggiunge i vincoli
65 | ALTER TABLE SOFTWARE_PROJECT ADD(
66 |     -- Chiave primaria
67 |     CONSTRAINT project_pk PRIMARY KEY (Project_ID), -- Project_ID NOT
        NULL
68 |     -- Chiave esterna
69 |     -- Se lo sviluppatore a capo di un progetto viene eliminato, il
        progetto
70 |     -- viene mantenuto e il leader viene settato a NULL
71 |     CONSTRAINT project_fk FOREIGN KEY (Leader)
72 |     REFERENCES DEVELOPER_BASE(Dev_ID) ON DELETE SET NULL,
73 |     -- Vincolo "Legit Names"
74 |     CONSTRAINT valid_dir CHECK (REGEXP_LIKE(Directory, '^[a-zA-Z0-9\_/_
        -]{1,}$')),
75 |     -- Vincolo Time Consistency
76 |     CONSTRAINT TimeConsistency CHECK(ENDDATE > STARTDATE)
77 | );
78 | /
79 | -- Trigger per settare, se necessario, la chiave primaria
        automaticamente
80 | CREATE OR REPLACE TRIGGER ProjectPK
81 | BEFORE INSERT ON SOFTWARE_PROJECT
82 | FOR EACH ROW
83 | BEGIN
84 |     DECLARE
85 |         pk SOFTWARE_PROJECT.PROJECT_ID%TYPE;
86 |     BEGIN
87 |         IF (:NEW.Project_ID = 0) THEN
88 |             SELECT NVL(MAX(Project_ID), 0) + 1 INTO pk FROM
        SOFTWARE_PROJECT;
89 |             :NEW.Project_ID := pk;
90 |         END IF;
91 |     END;
92 | END;
93 |
94 | /
```

4.1.3 Definizione della Tabella RELEASE

```
95 | /**
96 | * TABELLA: RELEASE
97 | * Crea la tabella e implementa i vincoli piu semplici.
98 | */
99 |
100 | -- Crea la tabella RELEASE
101 | CREATE TABLE RELEASE
102 | (
103 | Release_ID INTEGER DEFAULT 0,
104 | CodName VARCHAR2(64) NULL,
105 | Complete CHAR DEFAULT 'F' NOT NULL CHECK(Complete IN ('T','F')),
106 | Directory VARCHAR2(64) NOT NULL,
107 | StartDate DATE DEFAULT SYSDATE NOT NULL,
108 | ReleaseDate DATE DEFAULT NULL,
109 | Test_Pass CHAR DEFAULT 'F' NOT NULL CHECK(Test_Pass IN ('T','F')),
110 | SwProject INTEGER NOT NULL
111 | );
112 |
113 | /
114 | -- Aggiunge i vincoli
115 | ALTER TABLE RELEASE
116 | ADD(
117 | -- Chiave primaria
118 | CONSTRAINT rel_pk PRIMARY KEY(Release_ID),
119 | -- Chiave esterna verso SOFTWARE_PROJECT
120 | -- Se un progetto viene cancellato, vengono cancellate anche
121 | -- tutte le sue release.
122 | CONSTRAINT rel_fk FOREIGN KEY(SwProject)
123 | REFERENCES SOFTWARE_PROJECT(Project_ID) ON DELETE CASCADE,
124 | -- Non possono esistere nello stesso progetto due release con
125 | -- directory uguale.
126 | CONSTRAINT unique_RelDir UNIQUE (Directory, SwProject),
127 | -- Vincolo Time Consistency
128 | CONSTRAINT TimeConsistencyR CHECK(RELEASEDATE > STARTDATE)
129 | );
130 |
131 | /
132 |
133 | -- Trigger per settare, se necessario, la chiave primaria
   | automaticamente
134 | CREATE OR REPLACE TRIGGER ReleasePK
135 | BEFORE INSERT ON RELEASE
136 | FOR EACH ROW
137 | BEGIN
138 |     DECLARE
139 |         pk RELEASE.RELEASE_ID%TYPE;
140 |     BEGIN
141 |         IF (:NEW.Release_ID = 0) THEN
142 |             SELECT NVL(MAX(Release_ID),0) + 1 INTO pk FROM RELEASE;
143 |             :NEW.Release_ID := pk;
144 |         END IF;
145 |     END;
146 | END;
147 |
148 | /
```

4.1.4 Definizione della Tabella PACKAGE

```
149 | /**
150 | * TABELLA: PACKAGE
151 | * Crea la tabella e implementa i vincoli piu semplici.
152 | */
153 |
154 | -- Crea la tabella PACKAGE
155 | CREATE TABLE PACKAGE
156 | (
157 | Package_ID INTEGER DEFAULT 0,
158 | Name VARCHAR2(64) NOT NULL,
159 | Monitor VARCHAR(16) DEFAULT 'UNCHANGED' NOT NULL,
160 | Description VARCHAR2(512) DEFAULT 'Nessuna descrizione inserita.' NOT
      NULL,
161 | Release INTEGER NOT NULL
162 | );
163 |
164 | /
165 |
166 | -- Aggiunge i vincoli
167 | ALTER TABLE PACKAGE
168 | ADD(
169 | -- Chiave primaria
170 | CONSTRAINT pack_pk PRIMARY KEY(Package_ID),
171 | -- Chiave esterna verso Release
172 | CONSTRAINT pack_fk FOREIGN KEY(Release) REFERENCES RELEASE(Release_ID),
173 | -- Vincolo di dominio su Monitor
174 | CONSTRAINT PackMonitor CHECK (
175 | UPPER(Monitor) IN ('ISNEW','ISMODIFIED','WILLBEDELETED')
176 | ),
177 | -- RELEASE COMPOSITION
178 | -- Non possono esistere due package con lo stesso nome nella stessa
      release.
179 | CONSTRAINT unique_PackDir UNIQUE(Name, Release)
180 | );
181 |
182 | /
183 |
184 | -- Trigger per settare, se necessario, la chiave primaria
      automaticamente
185 | CREATE OR REPLACE TRIGGER PackagePK
186 | BEFORE INSERT ON PACKAGE
187 | FOR EACH ROW
188 | BEGIN
189 | DECLARE
190 | pk PACKAGE.PACKAGE_ID%TYPE;
191 | BEGIN
192 | IF(:NEW.Package_ID = 0) THEN
193 | SELECT NVL(MAX(Package_ID),0) + 1 INTO pk FROM PACKAGE;
194 | :NEW.Package_ID := pk;
195 | END IF;
196 | END;
197 | END;
198 |
199 | /
```


4.1.5 Definizione della Tabella CLASS

```
200 | /**
201 | * TABELLA: CLASS
202 | * Crea la tabella e implementa i vincoli piu' semplici.
203 | */
204 |
205 |
206 | -- Crea la tabella CLASS_BASE
207 | CREATE TABLE CLASS_BASE
208 | (
209 |   Name VARCHAR2(64) NOT NULL,
210 |   Scope VARCHAR2(16) DEFAULT 'DEFAULT' NOT NULL,
211 |   Filename VARCHAR2(64) NOT NULL,
212 |   Description VARCHAR(512) DEFAULT 'Nessuna descrizione inserita.' NOT
      NULL,
213 |   Monitor VARCHAR(16) DEFAULT 'UNCHANGED' NOT NULL,
214 |   changedMethod CHAR DEFAULT 'F' NOT NULL CHECK(changedMethod IN ('T','F'
      )),
215 |   StartLine INTEGER NOT NULL CHECK(StartLine>=0),
216 |   EndLine INTEGER NOT NULL CHECK(EndLine>0),
217 |   Path VARCHAR2(256),
218 |   Class_ID INTEGER DEFAULT 0,
219 |   Package_ID INTEGER NOT NULL,
220 |   SuperClass INTEGER DEFAULT NULL
221 | );
222 |
223 | /
224 | -- Aggiunge i vincoli
225 | ALTER TABLE CLASS_BASE
226 | ADD(
227 |   -- Chiave primaria
228 |   CONSTRAINT class_pk PRIMARY KEY(Class_ID),
229 |   -- Chiave esterna verso PACKAGE
230 |   CONSTRAINT class_fk FOREIGN KEY(Package_ID) REFERENCES PACKAGE(
      Package_ID),
231 |   -- Vincolo di dominio su Scope
232 |   CONSTRAINT ClassScope CHECK (
233 |     UPPER(Scope) IN ('PUBLIC','PRIVATE','DEFAULT','PROTECTED')
234 |   ),
235 |   -- Vincolo di dominio su Monitor
236 |   CONSTRAINT ClassMonitor CHECK (
237 |     UPPER(Monitor) IN ('ISNEW','ISMODIFIED','WILLBEDELETED')
238 |   ),
239 |   -- PACKAGE COMPOSITION
240 |   -- In un package non possono esistere due classi con lo stesso nome
241 |   CONSTRAINT unique_ClassPack UNIQUE(Name, Package_ID),
242 |   -- Vincolo No Auto-Refine
243 |   -- Una classe non pu estendere se stessa.
244 |   CONSTRAINT NO_AUTO_REFINE CHECK (CLASS_ID <> SuperClass)
245 | );
246 | /
247 | -- Per futuri Trigger INSTEAD OF
248 | CREATE OR REPLACE VIEW CLASS AS SELECT * FROM CLASS_BASE;
249 |
250 | /
```

4.1.6 Definizione della Tabella METHOD

```
251 | /**
252 | * TABELLA: METHOD
253 | * Crea la tabella e implementa i vincoli piu' semplici.
254 | */
255 |
256 |
257 | CREATE TABLE METHOD_BASE
258 | (
259 |   Name VARCHAR2(64) NOT NULL,
260 |   Scope VARCHAR2(16) DEFAULT 'DEFAULT' NOT NULL,
261 |   Type VARCHAR2(64) NOT NULL,
262 |   Monitor VARCHAR(16) DEFAULT 'UNCHANGED' NOT NULL,
263 |   Description VARCHAR(512) DEFAULT 'Nessuna descrizione inserita.' NOT
      NULL,
264 |   StartLine INTEGER NOT NULL CHECK(StartLine>0),
265 |   EndLine INTEGER NOT NULL CHECK(EndLine>0),
266 |   Method_ID INTEGER DEFAULT 0,
267 |   Class INTEGER NOT NULL
268 | );
269 |
270 | /
271 |
272 | -- Aggiunge i vincoli
273 | ALTER TABLE METHOD_BASE
274 | ADD(
275 |   -- Chiave primaria (meth_pk)
276 |   CONSTRAINT meth_pk PRIMARY KEY(Method_ID),
277 |   -- Chiave esterno verso CLASS
278 |   -- Se una classe viene cancellata vengono cancellati anche tutti i suoi
      metodi
279 |   CONSTRAINT meth_fk FOREIGN KEY(Class) REFERENCES CLASS_BASE(Class_ID)
      ON DELETE CASCADE,
280 |   -- Vincolo di dominio su Scope
281 |   CONSTRAINT methScope CHECK (
282 |     UPPER(Scope) IN ('PUBLIC','PRIVATE','DEFAULT','PROTECTED')
283 |   ),
284 |   CONSTRAINT methMonitor CHECK (
285 |     UPPER(Monitor) IN ('ISNEW','ISMODIFIED','WILLBEDELETED')
286 |   )
287 | );
288 |
289 | /
290 | -- Per futuri Trigger INSTEAD OF
291 | CREATE OR REPLACE VIEW METHOD AS SELECT * FROM METHOD_BASE;
292 |
293 | /
```

4.1.7 Definizione della Tabella PARAMETER

```
294 | /**
295 | * TABELLA: PARAMETER
296 | * Crea la tabella e implementa i vincoli piu' semplici.
297 | */
298 |
299 |
300 | -- Crea la tabella PARAMETER
301 | CREATE TABLE PARAMETER
302 | (
303 | Name VARCHAR2(64) NOT NULL,
304 | Position INTEGER NOT NULL CHECK(Position>0),
305 | Type VARCHAR2(64) NOT NULL,
306 | Param_ID INTEGER DEFAULT 0,
307 | Method INTEGER NOT NULL
308 | );
309 |
310 | /
311 |
312 | -- Aggiunge i vincoli
313 | ALTER TABLE PARAMETER
314 | ADD (
315 | -- Chiave primaria (Param_ID)
316 | CONSTRAINT param_pk PRIMARY KEY(Param_ID),
317 | -- Chiave esterna verso METHOD
318 | -- Se un metodo viene cancellato, vengono cancellati tutti i suoi
      parametri
319 | CONSTRAINT param_fk FOREIGN KEY(Method) REFERENCES METHOD_BASE(
      Method_ID) ON DELETE CASCADE,
320 | -- Non possono esservi pi parametri di un metodo nella stessa posizione
321 | CONSTRAINT unique_pos UNIQUE(METHOD,Position)
322 | );
323 |
324 | /
325 |
326 | -- Trigger per settare, se necessario, la chiave primaria
      automaticamente
327 | CREATE OR REPLACE TRIGGER ParameterPK
328 | BEFORE INSERT ON PARAMETER
329 | FOR EACH ROW
330 | BEGIN
331 |     DECLARE
332 |         pk PARAMETER.PARAM_ID%TYPE;
333 |     BEGIN
334 |         IF(:NEW.Param_ID = 0) THEN
335 |             SELECT NVL(MAX(Param_ID),0) + 1 INTO pk FROM PARAMETER;
336 |             :NEW.Param_ID := pk;
337 |         END IF;
338 |     END;
339 | END;
340 |
341 | /
```

4.1.8 Definizione della Tabella ATTRIBUTE

```
342 | /**
343 | * TABELLA: ATTRIBUTE
344 | * Crea la tabella e implementa i vincoli piu' semplici.
345 | */
346 |
347 | -- Crea la tabella ATTRIBUTE
348 | CREATE TABLE ATTRIBUTE
349 | (
350 | Name VARCHAR2(64) NOT NULL,
351 | Position INTEGER NOT NULL CHECK(Position>0),
352 | Type VARCHAR2(64) NOT NULL,
353 | Scope VARCHAR2(16) DEFAULT 'DEFAULT' NOT NULL,
354 | Attr_ID INTEGER DEFAULT 0,
355 | Class INTEGER NOT NULL
356 | );
357 |
358 | /
359 | -- Aggiunge i vincoli
360 | ALTER TABLE ATTRIBUTE
361 | ADD(
362 | -- Chiave primaria Attr_ID
363 | CONSTRAINT attr_pk PRIMARY KEY(Attr_ID),
364 | -- Chiave esterna verso CLASS
365 | -- Se una classe viene cancellata, vengono cancellati anche tutti i
    suoi attributi
366 | CONSTRAINT attr_fk FOREIGN KEY(Class) REFERENCES CLASS_BASE(Class_ID)
    ON DELETE CASCADE,
367 | -- CLASS COMPOSITION
368 | -- Non possono esistere due attributi con identificatore uguale nella
    stessa classe
369 | CONSTRAINT unique_ClassAttr UNIQUE(Name, Class),
370 | -- Vincolo di dominio su Scope
371 | CONSTRAINT AttrScope CHECK (UPPER(Scope) IN ('PUBLIC','PRIVATE','
    DEFAULT','PROTECTED'))
372 | );
373 |
374 | /
375 | -- Trigger per settare, se necessario, la chiave primaria
    automaticamente
376 | CREATE OR REPLACE TRIGGER AttributePK
377 | BEFORE INSERT ON ATTRIBUTE
378 | FOR EACH ROW
379 | BEGIN
380 |     DECLARE
381 |         pk ATTRIBUTE.ATTR_ID%TYPE;
382 |     BEGIN
383 |         IF (:NEW.Attr_ID = 0) THEN
384 |             SELECT NVL(MAX(Attr_ID),0) + 1 INTO pk FROM ATTRIBUTE;
385 |             :NEW.Attr_ID := pk;
386 |         END IF;
387 |     END;
388 | END;
389 | /
```

4.1.9 Definizione della Tabella COMMIT

```
390 |  /**
391 |  * TABELLA: COMMIT_T
392 |  * Crea la tabella e implementa i vincoli piu' semplici.
393 |  */
394 |
395 |  -- Crea la tabella COMMIT_T
396 |  CREATE TABLE COMMIT_T
397 |  (
398 |  InsDate DATE DEFAULT SYSDATE NOT NULL,
399 |  Description VARCHAR2(512) NOT NULL,
400 |  Commit_ID INTEGER DEFAULT 0,
401 |  Author INTEGER DEFAULT NULL,
402 |  Class INTEGER NOT NULL
403 |  );
404 |
405 |  /
406 |  -- Aggiunge i vincoli
407 |  ALTER TABLE COMMIT_T
408 |  ADD (
409 |  -- Chiave primaria (Commit_ID)
410 |  CONSTRAINT commit_pk PRIMARY KEY(Commit_ID),
411 |  -- Chiave esterna verso CLASS
412 |  -- Se una classe viene cancellata, vengono cancellate anche tutte le
      modifiche
413 |  -- ad essa apportate
414 |  CONSTRAINT com_fk FOREIGN KEY(Class) REFERENCES CLASS_BASE(Class_ID)
      ON DELETE CASCADE,
415 |  -- Chiave esterna verso DEVELOPER
416 |  -- Se uno sviluppatore viene cancellato, tutte le modifiche da lui
      apportate
417 |  -- vengono attribuite a NULL.
418 |  CONSTRAINT com_fk_dev FOREIGN KEY(Author) REFERENCES DEVELOPER_BASE(
      Dev_ID) ON DELETE SET NULL
419 |  );
420 |
421 |  /
422 |
423 |  -- Trigger per settare, se necessario, la chiave primaria
      automaticamente
424 |  CREATE OR REPLACE TRIGGER Commit_TPK
425 |  BEFORE INSERT ON COMMIT_T
426 |  FOR EACH ROW
427 |  BEGIN
428 |      DECLARE
429 |          pk COMMIT_T.COMMIT_ID%TYPE;
430 |      BEGIN
431 |          IF (:NEW.Commit_ID = 0) THEN
432 |              SELECT NVL(MAX(Commit_ID),0) + 1 INTO pk FROM COMMIT_T;
433 |              :NEW.Commit_ID := pk;
434 |          END IF;
435 |      END;
436 |  END;
437 |
438 |  /
```

4.1.10 Definizione della Tabella TEST

```
439 |  /**
440 |  * TABELLA: TEST
441 |  * Crea la tabella e implementa i vincoli piu' semplici.
442 |  */
443 |
444 |
445 |  -- Crea la tabella TEST
446 |  CREATE TABLE TEST
447 |  (
448 |  Test_ID INTEGER DEFAULT 0,
449 |  Directory VARCHAR2(64) NOT NULL,
450 |  Description VARCHAR(512) DEFAULT 'Nessuna descrizione inserita.' NOT
      NULL,
451 |  Filename VARCHAR2(64) NOT NULL,
452 |  SwProject INTEGER NOT NULL
453 |  );
454 |
455 |  /
456 |  -- Aggiunge i vincoli
457 |  ALTER TABLE TEST
458 |  ADD(
459 |  -- Chiave primaria
460 |  CONSTRAINT test_pk PRIMARY KEY(Test_ID),
461 |  -- Chiave esterna verso SOFTWARE_PROJECT
462 |  -- Se un progetto viene cancellato, vengono cancellate anche
463 |  -- tutte i suoi casi di test.
464 |  CONSTRAINT test_fk FOREIGN KEY(SwProject)
465 |  REFERENCES SOFTWARE_PROJECT(Project_ID) ON DELETE CASCADE
466 |  );
467 |
468 |  /
469 |
470 |  -- Trigger per settare, se necessario, la chiave primaria
      automaticamente
471 |  CREATE OR REPLACE TRIGGER TestPK
472 |  BEFORE INSERT ON TEST
473 |  FOR EACH ROW
474 |  BEGIN
475 |      DECLARE
476 |          pk TEST.Test_ID%TYPE;
477 |      BEGIN
478 |          IF (:NEW.Test_ID = 0) THEN
479 |              SELECT NVL(MAX(Test_ID),0) + 1 INTO pk FROM TEST;
480 |              :NEW.Test_ID := pk;
481 |          END IF;
482 |      END;
483 |  END;
484 |
485 |  /
```

4.1.11 Definizione della Tabella EXECUTED_TEST

```
486 |  /**
487 |  * TABELLA: EXECUTED_TEST
488 |  * Crea la tabella e implementa i vincoli piu' semplici.
489 |  */
490 |
491 |
492 |  -- Crea la tabella EXECUTED_TEST
493 |  CREATE TABLE EXECUTED_TEST
494 |  (
495 |  Execution_ID INTEGER DEFAULT 0,
496 |  Result CHAR DEFAULT 'F' NOT NULL CHECK(Result IN ('T','F')),
497 |  DataTest DATE DEFAULT SYSDATE NOT NULL,
498 |  UsedStructure VARCHAR2(512) NOT NULL,
499 |  Test INTEGER NOT NULL
500 |  );
501 |
502 |  /
503 |  -- Aggiunge i vincoli
504 |  ALTER TABLE EXECUTED_TEST
505 |  ADD(
506 |  -- Chiave primaria
507 |  CONSTRAINT execTest_pk PRIMARY KEY(Execution_ID),
508 |  -- Chiave esterna verso TEST
509 |  -- Se un caso di test viene cancellato, vengono cancellate anche
510 |  -- tutte le sue esecuzioni.
511 |  CONSTRAINT execTest_fk FOREIGN KEY(Test)
512 |  REFERENCES TEST(Test_ID) ON DELETE CASCADE
513 |  );
514 |
515 |  /
516 |  -- Trigger per settare, se necessario, la chiave primaria
   |  automaticamente
517 |  CREATE OR REPLACE TRIGGER Exec_TestPK
518 |  BEFORE INSERT ON EXECUTED_TEST
519 |  FOR EACH ROW
520 |  BEGIN
521 |      DECLARE
522 |          pk EXECUTED_TEST.Execution_ID%TYPE;
523 |      BEGIN
524 |          IF (:NEW.Execution_ID = 0) THEN
525 |              SELECT NVL(MAX(Execution_ID),0) + 1 INTO pk FROM
EXECUTED_TEST;
526 |              :NEW.Execution_ID := pk;
527 |          END IF;
528 |      END;
529 |  END;
530 |
531 |  /
```

4.1.12 Definizione della Tabella RELEASE_TEST

```
532 |  /**
533 |  * TABELLA: RELEASE_TEST
534 |  * Crea la tabella e implementa i vincoli piu' semplici.
535 |  */
536 |
537 |
538 |  -- Crea la tabella RELEASE_TEST
539 |  CREATE TABLE RELEASE_TEST
540 |  (
541 |  ExecutedTest  INTEGER NOT NULL,
542 |  Release  INTEGER NOT NULL
543 |  );
544 |
545 |  /
546 |
547 |
548 |  -- Aggiunge i vincoli
549 |  ALTER TABLE RELEASE_TEST
550 |  ADD(
551 |  -- Chiave esterna verso EXECUTED_TEST
552 |  -- Eliminando l'esecuzione di un test, vengono eliminate anche tutte
    le sue associazioni
553 |  -- alle release.
554 |  CONSTRAINT RelTestfk1 FOREIGN KEY(ExecutedTest)
555 |  REFERENCES EXECUTED_TEST(Execution_ID) ON DELETE CASCADE,
556 |  -- Chiave esterna verso RELEASE
557 |  -- Eliminando una release, vengono eliminate anche tutte le
    associazioni ai test eseguiti
558 |  -- da esso.
559 |  CONSTRAINT RelTestfk2 FOREIGN KEY(Release)
560 |  REFERENCES RELEASE(Release_ID) ON DELETE CASCADE
561 |  );
562 |
563 |  /
```


4.1.13 Definizione della Tabella COMMIT_METHOD

```
564 |  /**
565 |  * TABELLA: COMMIT_METH
566 |  * Crea la tabella e implementa i vincoli piu' semplici.
567 |  */
568 |
569 |
570 |  -- Crea la tabella COMMIT_METH
571 |  CREATE TABLE COMMIT_METH
572 |  (
573 |  Commit_ID INTEGER NOT NULL,
574 |  Method INTEGER NOT NULL
575 |  );
576 |
577 |  /
578 |
579 |  -- Aggiunge i vincoli
580 |  ALTER TABLE COMMIT_METH
581 |  ADD (
582 |  -- Chiave esterna verso COMMIT_T
583 |  -- Se viene cancellato un commit vengono cancellate anche le
    associazioni di modifica
584 |  -- a metodi che il commit cancellato apportava.
585 |  CONSTRAINT commit_meth_fk1 FOREIGN KEY(Commit_ID) REFERENCES
    COMMIT_T(Commit_ID)
586 |  ON DELETE CASCADE,
587 |  -- Chiave esterna verso METHOD
588 |  -- Se viene cancellato un metodo vengono cancellate anche tutte le
    istanze di modifica
589 |  -- a quel metodo
590 |  CONSTRAINT commit_meth_fk2 FOREIGN KEY(Method) REFERENCES
    METHOD_BASE(Method_ID)
591 |  ON DELETE CASCADE
592 |  );
593 |
594 |  /
```

4.1.14 Definizione della Tabella DEV_ASSIGN

```
595 |  /**
596 |  * TABELLA: DEV_ASSIGN
597 |  * Crea la tabella e implementa i vincoli piu' semplici.
598 |  */
599 |
600 |  -- Crea la tabella DEV_ASSIGN
601 |  CREATE TABLE DEV_ASSIGN
602 |  (
603 |  Dev INTEGER NOT NULL,
604 |  Project_ID INTEGER NOT NULL
605 |  );
606 |
607 |  /
608 |
609 |
610 |  -- Aggiunge i vincoli
611 |  ALTER TABLE DEV_ASSIGN
612 |  ADD(
613 |  -- Chiave esterna verso DEVELOPER
614 |  -- Eliminando uno sviluppatore, vengono eliminate anche tutte le sue
        associazioni
615 |  -- a progetti.
616 |  CONSTRAINT devAssfk1 FOREIGN KEY(Dev)
617 |  REFERENCES DEVELOPER_BASE(Dev_ID) ON DELETE CASCADE,
618 |  -- Chiave esterna verso SOFTWARE_PROJECT
619 |  -- Eliminando un progetto, vengono eliminate anche tutte le
        associazioni di sviluppatori
620 |  -- ad esso.
621 |  CONSTRAINT devAssfk2 FOREIGN KEY(Project_ID)
622 |  REFERENCES SOFTWARE_PROJECT(Project_ID) ON DELETE CASCADE,
623 |  -- Ciascuno sviluppatore pu essere assegnato ad un dato progetto una
        sola volta.
624 |  CONSTRAINT unique_DevAssign UNIQUE(DEV,PROJECT_ID)
625 |  );
626 |
627 |  /
```

4.1.15 Definizione della Tabella BASIC_TYPES

```
628 | /**
629 | * TABELLA: BASICTYPES
630 | * Crea la tabella e la popola
631 | */
632 |
633 | -- Creazione tabella
634 | CREATE TABLE BASICTYPE
635 | (
636 | Name VARCHAR2(16) NOT NULL
637 | );
638 |
639 | /
640 |
641 | -- Popolamento con tipi predefiniti
642 | INSERT ALL
643 | INTO BASICTYPE VALUES ('VOID')
644 | INTO BASICTYPE VALUES ('BYTE')
645 | INTO BASICTYPE VALUES ('SHORT')
646 | INTO BASICTYPE VALUES ('INT')
647 | INTO BASICTYPE VALUES ('LONG')
648 | INTO BASICTYPE VALUES ('FLOAT')
649 | INTO BASICTYPE VALUES ('DOUBLE')
650 | INTO BASICTYPE VALUES ('BOOLEAN')
651 | INTO BASICTYPE VALUES ('CHAR')
652 | INTO BASICTYPE VALUES ('STRING')
653 | SELECT * FROM DUAL;
654 |
655 | /
```

4.2 Viste

4.2.1 CLASSNAMES

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

```
656 |      -- Vista che mostra tutti i nomi di classe con la relativa release.
657 |      CREATE VIEW ClassNames AS
658 |      SELECT R.Release_ID, CL.Name
659 |      FROM (RELEASE R JOIN PACKAGE PK ON PK.Release=R.Release_ID) JOIN
        CLASS CL ON CL.Package_ID=PK.PACKAGE_ID;
660 |
661 |      /
```

4.2.2 TESTNAMES

La vista mostra per ciascun progetto , i nomi delle strutture che vi appartengono. Viene utilizzata nella stored function **isValidName()** per l'implementazione del vincolo **Name Consistency For Test**

```
662 |      -- Vista che mostra tutti i nomi delle strutture con relativo
        progetto.
663 |      CREATE VIEW TestNames AS
664 |      SELECT S.Project_ID, R.CodName AS ReleaseName , PK.Name AS
        PackageName, CL.Name AS ClassName, M.Name AS MethodName
665 |      FROM (((SOFTWARE_PROJECT S LEFT JOIN RELEASE R ON S.Project_ID=R.
        SwProject) LEFT JOIN PACKAGE PK ON PK.Release=R.Release_ID) LEFT
        JOIN
666 |              CLASS CL ON CL.Package_ID=PK.PACKAGE_ID) LEFT JOIN METHOD M
        ON M.Class=CL.Class_ID;
667 |
668 |      /
```

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:

```
669 |      -- Funzione calculate_path. Calcola il percorso completo del file
        sorgente filename nel package
670 |      -- con ID p. La funzione viene usata nel trigger CALC_PATH, per
        calcolare automaticamente il percorso
671 |      -- nel file system della classe appena inserita.
672 |      CREATE OR REPLACE FUNCTION calculate_path(filename CLASS.FILENAME%
        TYPE, p CLASS.PACKAGE_ID%TYPE)
673 |      RETURN VARCHAR2
674 |      IS
675 |      path VARCHAR2(256);
676 |      BEGIN
677 |      SELECT '/'PR.DIRECTORY '/'RE.DIRECTORY '/'src/' REPLACE(PA.NAME, '.', '/')
        ) INTO path
678 |      FROM (PACKAGE PA JOIN RELEASE RE ON PA.RELEASE = RE.RELEASE_ID)
        JOIN SOFTWARE_PROJECT PR ON RE.SWPROJECT = PR.PROJECT_ID
679 |      WHERE PA.PACKAGE_ID = p;
680 |      path := path '/' filename '.java';
681 |      return path;
682 |      END;
683 |
684 |      /
```

4.3.2 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 o aggiornamento di dati nella tabella `CLASS_BASE`.

```
685 |      -- Trigger calc_path. Calcola il valore di PATH prima dell'
        inserimento di un nuovo record.
686 |      CREATE OR REPLACE TRIGGER calc_path
687 |      BEFORE INSERT OR UPDATE ON CLASS_BASE
688 |      FOR EACH ROW
689 |      BEGIN
690 |      BEGIN
691 |      :NEW.PATH := CALCULATE_PATH(:NEW.FILENAME, :NEW.PACKAGE_ID);
692 |      END;
693 |      END;
694 |
695 |      /
```

4.3.3 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.

```
696 |  -- Dato un nome del tipo e una release, ritorna 'T' se quel tipo
      |      valido (cioè
697 |  -- se in BASICTYPES oppure se esiste una classe con lo stesso nome
      |      nella stessa
698 |  -- release) o 'F' altrimenti.
699 |  CREATE OR REPLACE FUNCTION isValidType(typename VARCHAR2, rel RELEASE.
      |      RELEASE_ID%TYPE)
700 |  RETURN CHAR
701 |  IS
702 |  found INTEGER;
703 |  BEGIN
704 |  SELECT COUNT(*) INTO found --controlla se il tipo primitivo
705 |  FROM BASICTYPE
706 |  WHERE Name=UPPER(typename);
707 |  IF FOUND <> 0 THEN
708 |  RETURN 'T';
709 |  END IF;
710 |  SELECT COUNT(*) INTO found
711 |  FROM CLASSNAMES
712 |  WHERE CLASSNAMES.NAME = typename AND RELEASE_ID = rel;
713 |  IF FOUND <> 0 THEN
714 |  RETURN 'T';
715 |  ELSE
716 |  RETURN 'F';
717 |  END IF;
718 |  END;
719 |  /
```

4.3.4 Verificare se un FileName è valido per un Caso di Test

La funzione `isValidName()` verifica se il filename di un caso di test è legale per un dato progetto.

```
720 |  
721 | CREATE OR REPLACE FUNCTION isValidName(name VARCHAR2, p  
      SOFTWARE_PROJECT.PROJECT_ID%TYPE)  
722 | RETURN CHAR  
723 | IS  
724 | found INTEGER;  
725 | BEGIN  
726 | SELECT COUNT(*) INTO found --controlla se il NOME e' presente nelle  
      strutture del progetto  
727 | FROM TESTNAMES  
728 | WHERE Project_ID = p AND (TESTNAMES.ReleaseName=name OR TESTNAMES.  
      PackageName=name OR TESTNAMES.ClassName=name OR TESTNAMES.MethodName  
      =name);  
729 | IF FOUND <> 0 THEN  
730 | RETURN 'T';  
731 | ELSE  
732 | RETURN 'F';  
733 | END IF;  
734 | END;  
735 |  
736 | /
```

4.4 Implementazione dei Vincoli

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

4.4.1 Implementazione del vincolo Distinct Personal Mails

```
737 |      -- Descrizione: Lo stesso indirizzo email non pu essere utilizzato
738 |      da pi
739 |      -- sviluppatori.
740 |      -- Note : Calcola ed imposta anche la chiave primaria, se
741 |      necessario.
742 |
743 |      CREATE OR REPLACE TRIGGER checkEmails
744 |      INSTEAD OF INSERT OR UPDATE ON DEVELOPER
745 |      FOR EACH ROW
746 |      BEGIN
747 |      DECLARE
748 |      TEMP INTEGER:=0;
749 |      pk DEVELOPER.DEV_ID%TYPE;
750 |      BEGIN
751 |      IF INSERTING THEN -- effettua l'inserimento
752 |      SELECT COUNT(*) INTO TEMP
753 |      FROM DEVELOPER_BASE D
754 |      WHERE (D.eMail=:NEW.eMail OR D.eMail2=:NEW.eMail OR
755 |      D.eMail=:NEW.eMail2 OR D.eMail2=:NEW.eMail2);
756 |      IF(TEMP <> 0) THEN -- Violazioni presenti
757 |      RAISE_APPLICATION_ERROR(-20001,'Indirizzo eMail gia presente!');
758 |      ELSE
759 |      SELECT NVL(MAX(Dev_ID),0)+1 INTO pk FROM DEVELOPER_BASE;
760 |      INSERT INTO DEVELOPER_BASE (DEV_ID,FIRSTNAME,LASTNAME,EMAIL,EMAIL2)
761 |      VALUES (pk, :NEW.FIRSTNAME, :NEW.LASTNAME, :NEW.EMAIL, :NEW.EMAIL2)
762 |      ;
763 |      END IF;
764 |      ELSE -- UPDATING
765 |      SELECT COUNT(*) INTO TEMP
766 |      FROM DEVELOPER_BASE D
767 |      WHERE D.Dev_ID <> :OLD.DEV_ID AND (D.eMail=:NEW.eMail OR D.eMail2=:
768 |      NEW.eMail OR
769 |      D.eMail=:NEW.eMail2 OR D.eMail2=:NEW.eMail2);
770 |      IF(TEMP <> 0) THEN -- Violazioni presenti
771 |      RAISE_APPLICATION_ERROR(-20001,'Indirizzo eMail gia presente!');
772 |      ELSE
773 |      UPDATE DEVELOPER_BASE SET
774 |      FIRSTNAME = :NEW.FIRSTNAME,
775 |      LASTNAME = :NEW.LASTNAME,
776 |      EMAIL = :NEW.EMAIL,
777 |      EMAIL2 = :NEW.EMAIL2
778 |      WHERE DEV_ID = :OLD.DEV_ID;
779 |      END IF;
780 |      END IF;
781 |      END;
782 |      END;
```


4.4.2 Implementazione del vincolo TypeValues

```
780 |      --
      -----
781 |      -- Vincolo : "Type Values"
782 |      -- Descrizione: I tipi degli attributi, dei parametri formali dei
metodi ed il
783 |      -- tipo del valore ritornato da un metodo devono essere valori di
784 |      -- in BasicTypes oppure nomi di classi della stessa release.
785 |      -- Note : Implementato sulle tabelle ATTRIBUTE, METHOD, PARAMETER
con i
786 |      -- seguenti tre trigger.
787 |      --
      -----

788 |
789 |      -- Implementa il vincolo per ATTRIBUTE
790 |      CREATE OR REPLACE TRIGGER checkType_attr
791 |      BEFORE INSERT OR UPDATE ON ATTRIBUTE
792 |      FOR EACH ROW
793 |      BEGIN
794 |      DECLARE
795 |      rel RELEASE.RELEASE_ID%TYPE;
796 |      BEGIN
797 |      SELECT PK.RELEASE INTO rel
798 |      FROM CLASS CL JOIN PACKAGE PK ON CL.PACKAGE_ID = PK.PACKAGE_ID
799 |      WHERE CL.CLASS_ID = :NEW.CLASS;
800 |      IF isValidType(:NEW.TYPE, rel) = 'F' THEN
801 |      RAISE_APPLICATION_ERROR(-20002, 'Tipo non valido!');
802 |      END IF;
803 |      END;
804 |      END;
805 |
806 |      /
807 |
808 |      -- Implementa il vincolo per METHOD
809 |      CREATE OR REPLACE TRIGGER checkType_meth
810 |      BEFORE INSERT OR UPDATE ON METHOD_BASE
811 |      FOR EACH ROW
812 |      BEGIN
813 |      DECLARE
814 |      rel RELEASE.RELEASE_ID%TYPE;
815 |      BEGIN
816 |      SELECT PK.RELEASE INTO rel
817 |      FROM CLASS CL JOIN PACKAGE PK ON CL.PACKAGE_ID = PK.PACKAGE_ID
818 |      WHERE CL.CLASS_ID = :NEW.CLASS;
819 |      IF isValidType(:NEW.TYPE, rel) = 'F' THEN
820 |      RAISE_APPLICATION_ERROR(-20002, 'Tipo non valido!');
821 |      END IF;
822 |      END;
823 |      END;
824 |
825 |      /
826 |
827 |      -- Implementa il vincolo per PARAMETER
828 |      CREATE OR REPLACE TRIGGER checkType_param
829 |      BEFORE INSERT OR UPDATE ON PARAMETER
```

```

830 |     FOR EACH ROW
831 |     BEGIN
832 |     DECLARE
833 |     rel RELEASE.RELEASE_ID%TYPE;
834 |     BEGIN
835 |     SELECT PK.RELEASE INTO rel
836 |     FROM (METHOD M JOIN CLASS CL ON M.CLASS = CL.CLASS_ID) JOIN PACKAGE
      PK ON CL.PACKAGE_ID =
837 |     PK.PACKAGE_ID
838 |     WHERE M.METHOD_ID = :NEW.METHOD;
839 |     IF isValidType(:NEW.TYPE, rel) = 'F' THEN
840 |     RAISE_APPLICATION_ERROR(-20002, 'Tipo non valido!');
841 |     END IF;
842 |     END;
843 |     END;
844 |
845 | /

```

4.4.3 Implementazione del vincolo Name Consistency for Test

```

847 | --
      -----
848 | -- Vincolo : "Name Consistency for TEST"
849 | -- Descrizione: Il filename dei test deve avere lo stesso valore di
      una struttura
850 | -- all'interno del progetto ospitante del test.
851 | --
      -----

852 |
853 | CREATE OR REPLACE TRIGGER checkName_Test
854 | BEFORE INSERT OR UPDATE ON TEST
855 | FOR EACH ROW
856 | BEGIN
857 | DECLARE
858 | p SOFTWARE_PROJECT.PROJECT_ID%TYPE;
859 | BEGIN
860 | p := :NEW.SwProject;
861 | IF isValidName(:NEW.FileName, p) = 'F' THEN
862 | RAISE_APPLICATION_ERROR(-20002, 'FileName non valido!');
863 | END IF;
864 | END;
865 | END;
866 | /

```

4.4.4 Implementazione del vincolo Single Public Class per File e File structure consistency for Class

```
867 |  --
      |  -----
868 |  -- Vincolo : "Single Public Class per File"
869 |  -- Descrizione: Ogni file sorgente contiene alpi una classe con scope
      |  PUBLIC.
870 |  --
      |  -----
871 |  -- Vincolo : "File structure consistency"
872 |  -- Descrizione: In ciascun file sorgente, non possono sovrapporsi
      |  definizioni di
873 |  -- classi diverse.
874 |  --
      |  -----
875 |  -- Note : Necessitano della definizione di una vista di appoggio per
      |  non
876 |  -- incorrere in errori causati da mutating tables. Il seguente
877 |  -- trigger implementa entrambi i vincoli.
878 |  --
      |  -----
879 |
880 |
881 |  CREATE OR REPLACE TRIGGER check_Valid_Class
882 |  INSTEAD OF INSERT OR UPDATE ON CLASS
883 |  FOR EACH ROW
884 |  BEGIN
885 |  DECLARE
886 |  VIOLATION INTEGER:=0;
887 |  pk CLASS.CLASS_ID%TYPE;
888 |  BEGIN
889 |  -- Verifico se sussistono violazioni
890 |  -- Conto le classi public nello stesso file sorgente e nello stesso
      |  package
891 |  IF UPDATING THEN
892 |  SELECT COUNT(*) INTO VIOLATION
893 |  FROM CLASS_BASE C
894 |  WHERE C.PACKAGE_ID = :NEW.PACKAGE_ID AND C.FILENAME = :NEW.FILENAME
      |  AND C.SCOPE = 'PUBLIC' AND C.CLASS_ID <> :OLD.CLASS_ID ;
895 |  ELSE -- Sto inserendo un nuovo record
896 |  SELECT COUNT(*) INTO VIOLATION
897 |  FROM CLASS_BASE C
898 |  WHERE C.PACKAGE_ID = :NEW.PACKAGE_ID AND C.FILENAME = :NEW.FILENAME
      |  AND
899 |  C.SCOPE = 'PUBLIC';
900 |  END IF;
901 |  IF( VIOLATION > 0 AND :NEW.SCOPE = 'PUBLIC') THEN -- c' gi una classe
      |  public in quel file in quel package
902 |  RAISE_APPLICATION_ERROR(-20003,'In un file sorgente pu esserci un''
      |  unica classe public!');
903 |  ELSE -- non c' violazione del vincolo "Single Public Class per File"
904 |  -- Verifico il vincolo "File structure consistency"
```

```

905 | SELECT COUNT (*) INTO VIOLATION
906 | FROM CLASS_BASE CL
907 | WHERE (:NEW.FILENAME = CL.FILENAME AND :NEW.PACKAGE_ID = CL.PACKAGE_ID
      AND :NEW.NAME <>
908 | CL.NAME) AND
909 | (((CL.STARTLINE BETWEEN :NEW.STARTLINE AND :NEW.ENDLINE) AND CL.
      ENDLINE > :NEW.ENDLINE) OR
910 | ((:NEW.STARTLINE BETWEEN CL.STARTLINE AND CL.ENDLINE) AND
911 | :NEW.ENDLINE > + CL.ENDLINE));
912 | IF (VIOLATION > 0) THEN
913 | RAISE_APPLICATION_ERROR(-20004, 'Violata corretta strutturazione dei
      blocchi del file. ');
914 | ELSE -- Nessuna violazione. Proseguo
915 | IF INSERTING THEN -- effettua l'inserimento
916 | SELECT NVL(MAX(CLASS_ID),0)+1 INTO pk FROM CLASS_BASE;
917 | INSERT INTO CLASS_BASE (CLASS_ID,NAME,SCOPE,FILENAME,PATH,STARTLINE,
      ENDLINE,MONITOR,CHANGEDMETHOD,DESCRIPTION,PACKAGE_ID)
918 | VALUES (pk,:NEW.NAME,:NEW.SCOPE,:NEW.FILENAME,:NEW.PATH,:NEW.STARTLINE
      ,:NEW.ENDLINE,:NEW.MONITOR,:NEW.CHANGEDMETHOD,:NEW.DESCRPTION,:NEW.
      PACKAGE_ID);
919 | ELSE -- UPDATING
920 | UPDATE CLASS_BASE SET
921 | NAME = :NEW.NAME,
922 | SCOPE = :NEW.SCOPE,
923 | FILENAME = :NEW.FILENAME,
924 | PATH = :NEW.PATH,
925 | STARTLINE = :NEW.STARTLINE,
926 | ENDLINE = :NEW.ENDLINE,
927 | MONITOR = :NEW.MONITOR,
928 | CHANGEDMETHOD = :NEW.CHANGEDMETHOD,
929 | DESCRIPTION = :NEW.DESCRPTION,
930 | PACKAGE_ID = :NEW.PACKAGE_ID
931 | WHERE CLASS_ID = :OLD.CLASS_ID;
932 | END IF;
933 | END IF;
934 | END IF;
935 | END;
936 | END;
937 |
938 | /

```

4.4.5 Implementazione del vincolo File consistency for methods

```
939 | -- Descrizione: In ciascun file sorgente, non possono sovrapporsi
    | definizioni di
940 | -- metodi diversi. Inoltre, un la definizione di un metodo non puo
941 | -- trovarsi al di fuori di quella della classe a cui appartiene.
942 |
943 | CREATE OR REPLACE TRIGGER CHECK_VALID_METHOD
944 | INSTEAD OF INSERT OR UPDATE ON METHOD
945 | FOR EACH ROW
946 | BEGIN
947 | DECLARE
948 | classInfo CLASS%ROWTYPE;
949 | pk METHOD.METHOD_ID%TYPE;
950 | VIOLATION INTEGER;
951 | BEGIN
952 | -- recupero informazioni relative alla classe cui il metodo appartiene
953 | SELECT * INTO classInfo
954 | FROM CLASS
955 | WHERE CLASS_ID = :NEW.CLASS;
956 | -- verifico se il metodo definito al di fuori della classe
957 | IF( :NEW.STARTLINE < classInfo.STARTLINE OR
958 | :NEW.ENDLINE > classInfo.ENDLINE) THEN
959 | RAISE_APPLICATION_ERROR(-20005, 'Il metodo definito al di fuori della
    | classe a cui appartiene');
960 | ELSE -- verifico se il metodo si sovrappone ad altri metodi della
    | stessa classe
961 | SELECT COUNT(*) INTO VIOLATION
962 | FROM METHOD_BASE M
963 | WHERE M.CLASS = classInfo.CLASS_ID AND (
964 | M.STARTLINE BETWEEN (:NEW.STARTLINE+1) AND :NEW.ENDLINE-1 OR
965 | :NEW.STARTLINE BETWEEN (M.STARTLINE+1) AND M.ENDLINE-1 );
966 | IF VIOLATION > 0 THEN
967 | RAISE_APPLICATION_ERROR(-20006, 'Il metodo si sovrappone ad altri
    | metodi della stessa classe');
968 | ELSE
969 | IF INSERTING THEN
970 | SELECT NVL(MAX(METHOD_ID),0)+1 INTO pk FROM METHOD_BASE;
971 | INSERT INTO METHOD_BASE (METHOD_ID,SCOPE, NAME, TYPE, STARTLINE,
    | ENDLINE, MONITOR, DESCRIPTION, CLASS)
972 | VALUES(pk,:NEW.SCOPE, :NEW.NAME, :NEW.TYPE, :NEW.STARTLINE, :NEW.
    | ENDLINE, :NEW.MONITOR, :NEW.DESCRPTION, :NEW.CLASS);
973 | ELSE --UPDATING
974 | UPDATE METHOD_BASE SET
975 | SCOPE = :NEW.SCOPE,
976 | NAME = :NEW.NAME,
977 | TYPE = :NEW.TYPE,
978 | STARTLINE = :NEW.STARTLINE,
979 | ENDLINE = :NEW.ENDLINE,
980 | MONITOR = :NEW.MONITOR,
981 | DESCRIPTION = :NEW.DESCRPTION,
982 | CLASS = :NEW.CLASS
983 | WHERE METHOD_ID = :OLD.METHOD_ID;
984 | END IF;
985 | END IF;
986 | END IF;
987 | END;
988 | END;
```

4.4.6 Implementazione del vincolo Commit consistency

```
989 |      -- Descrizione: Ciascun commit interessa una ed una sola classe. I
    |      metodi
990 |      -- associati a ciascun commit devono essere metodi di quella classe
    |      .
991 |      --
    |      -----
992 |
993 |      CREATE OR REPLACE TRIGGER CHECK_COMMIT_CONSISTENCY
994 |      BEFORE INSERT OR UPDATE ON COMMIT_METH
995 |      FOR EACH ROW
996 |      BEGIN
997 |      DECLARE
998 |      commit_class COMMIT_T.CLASS%TYPE;
999 |      method_class METHOD.CLASS%TYPE;
1000 |      BEGIN
1001 |      SELECT CLASS INTO commit_class
1002 |      FROM COMMIT_T
1003 |      WHERE COMMIT_ID = :NEW.COMMIT_ID;
1004 |      SELECT CLASS INTO method_class
1005 |      FROM METHOD
1006 |      WHERE METHOD_ID = :NEW.METHOD;
1007 |      IF commit_class <> method_class THEN
1008 |      RAISE_APPLICATION_ERROR(-20008, 'Il metodo non appartiene alla
    |      classe interessata dal commit');
1009 |      END IF;
1010 |      END;
1011 |      END;
1012 |
1013 |      /
```

Capitolo 5

Manuale d'Uso

5.1 Popolamento con Dati di Esempio

```
01 | -----
02 | -- SCRIPT PER IL POPOLAMENTO DELLA BASE DI DATI
03 | -----
04 | -- Universit degli Studi di Napoli Federico II
05 | -- Insegnamento di Basi di Dati e Sistemi Informativi
06 | --
07 | -----
08 | -- Questo script popola il database con dati fittizi.
09 | -- Si consiglia di utilizzarlo su un database vuoto,
10 | -- per evitare eventuali problemi dovuti a conflitti.
11 | -----
12 |
13 | -- SVILUPPATORI
14 | INSERT INTO DEVELOPER_BASE (DEV_ID,FIRSTNAME,LASTNAME,EMAIL,EMAIL2)
15 | VALUES (1,'Dario','Leone','dari.leone@studenti.unina.it',NULL);
16 | INSERT INTO DEVELOPER_BASE (DEV_ID,FIRSTNAME,LASTNAME,EMAIL,EMAIL2)
17 | VALUES (2,'Marco','Rossi','marco.rossi@azienda.it',NULL);
18 | INSERT INTO DEVELOPER_BASE (DEV_ID,FIRSTNAME,LASTNAME,EMAIL,EMAIL2)
19 | VALUES (3,'Ciro','Bianchi','cbianchi@azienda.it',NULL);
20 | INSERT INTO DEVELOPER_BASE (DEV_ID,FIRSTNAME,LASTNAME,EMAIL,EMAIL2)
21 | VALUES (4,'Giovanni','Verdi','giovanniv@azienda.it',NULL);
22 | INSERT INTO DEVELOPER_BASE (DEV_ID,FIRSTNAME,LASTNAME,EMAIL,EMAIL2)
23 | VALUES (5,'Natalia','Esposito','natalia.espo@azienda.it','nat.es@fmail.
    com');
24 | INSERT INTO DEVELOPER_BASE (DEV_ID,FIRSTNAME,LASTNAME,EMAIL,EMAIL2)
25 | VALUES (6,'Alessandro','Romano','aleromano@azienda.it','
    alessandr@kmail.com');
26 | INSERT INTO DEVELOPER_BASE (DEV_ID,FIRSTNAME,LASTNAME,EMAIL,EMAIL2)
27 | VALUES (7,'Anna Chiara','de Martino','acdemartino@azienda.it','
    annachiara@fmail.com');
28 | INSERT INTO DEVELOPER_BASE (DEV_ID,FIRSTNAME,LASTNAME,EMAIL,EMAIL2)
29 | VALUES (8,'Michele','Guelfi','mguelfi@azienda.it',NULL);
30 | INSERT INTO DEVELOPER_BASE (DEV_ID,FIRSTNAME,LASTNAME,EMAIL,EMAIL2)
31 | VALUES (9,'Nick','Morris','nmorris@azienda.it',NULL);
32 | INSERT INTO DEVELOPER_BASE (DEV_ID,FIRSTNAME,LASTNAME,EMAIL,EMAIL2)
33 | VALUES (10,'Emma','Brown','emmab@azienda.it','emmabrown@fmail.com');
34 | COMMIT;
35 |
36 |
37 |
```

```

38 |
39 | -- PROGETTI
40 |
41 | INSERT INTO SOFTWARE_PROJECT (PROJECT_ID,NAME,DESCRIPTION,DIRECTORY,
    STARTDATE,ENDDATE,LEADER)
42 | VALUES ('1','Zoo','Programma per la gestione di uno zoo.','zoo',TO_DATE
    ('10-LUG-19','dd-mm-yy'),NULL,'1');
43 | INSERT INTO SOFTWARE_PROJECT (PROJECT_ID,NAME,DESCRIPTION,DIRECTORY,
    STARTDATE,ENDDATE,LEADER)
44 | VALUES ('2','Hello World','Funzionalità di Hello World di livello
    enterprise.','hello',to_date('10-LUG-19','dd-mm-yy'),NULL,'5');
45 | COMMIT;
46 |
47 |
48 | -- Assegnazioni sviluppatori - progetti
49 | INSERT INTO DEV_ASSIGN (DEV,PROJECT_ID) VALUES ('1','1');
50 | INSERT INTO DEV_ASSIGN (DEV,PROJECT_ID) VALUES ('2','1');
51 | INSERT INTO DEV_ASSIGN (DEV,PROJECT_ID) VALUES ('3','1');
52 | INSERT INTO DEV_ASSIGN (DEV,PROJECT_ID) VALUES ('4','1');
53 | INSERT INTO DEV_ASSIGN (DEV,PROJECT_ID) VALUES ('5','2');
54 | INSERT INTO DEV_ASSIGN (DEV,PROJECT_ID) VALUES ('6','2');
55 | INSERT INTO DEV_ASSIGN (DEV,PROJECT_ID) VALUES ('7','2');
56 | INSERT INTO DEV_ASSIGN (DEV,PROJECT_ID) VALUES ('8','2');
57 | INSERT INTO DEV_ASSIGN (DEV,PROJECT_ID) VALUES ('9','1');
58 | INSERT INTO DEV_ASSIGN (DEV,PROJECT_ID) VALUES ('10','1');
59 | INSERT INTO DEV_ASSIGN (DEV,PROJECT_ID) VALUES ('6','1');
60 | COMMIT;
61 |
62 | -- RELEASE
63 | -- Release del progetto "Zoo"
64 | INSERT INTO RELEASE (RELEASE_ID,DIRECTORY,CODNAME,STARTDATE,COMPLETE,
    RELEASEDATE,SWPROJECT,TEST_PASS)
65 | VALUES ('1','alpha','Alpha',to_date('14-LUG-19','dd-mm-yy'),'F',NULL,'1
    ','F');
66 | -- Release del progetto "HelloWorld"
67 | INSERT INTO RELEASE (RELEASE_ID,DIRECTORY,CODNAME,STARTDATE,COMPLETE,
    RELEASEDATE,SWPROJECT,TEST_PASS)
68 | VALUES ('2','hello1','1.0 pro',to_date('14-LUG-19','dd-mm-yy'),'F',NULL
    ,'2','F');
69 | COMMIT;
70 |
71 | -- PACKAGE
72 | -- Package del progetto "Zoo", release "Alpha"
73 | INSERT INTO PACKAGE (PACKAGE_ID,NAME,MONITOR,DESCRIPTION,RELEASE)
74 | VALUES ('1','it.zoo.animali','ISNEW','Contiene tutti gli animali.','1')
    ;
75 | INSERT INTO PACKAGE (PACKAGE_ID,NAME,MONITOR,DESCRIPTION,RELEASE)
76 | VALUES ('2','it.zoo.strutture','ISNEW','Contiene tutte le strutture
    dello zoo.','1');
77 | -- Package del progetto "Hello World", release "1.0 pro"
78 | INSERT INTO PACKAGE (PACKAGE_ID,NAME,MONITOR,DESCRIPTION,RELEASE)
79 | VALUES ('3','it.azienda.hello','ISNEW','Pacchetto principale.','2');
80 | COMMIT;
81 |
82 |
83 |
84 |
85 |

```



```

86 |
87 | -- CLASSI
88 | -- PROGETTO "Zoo"
89 | -- -- Classi del pacchetto it.zoo.animali
90 | INSERT INTO CLASS_BASE (CLASS_ID,NAME,SCOPE,FILENAME,PATH,STARTLINE,
    ENDLINE,MONITOR,CHANGEDMETHOD,DESCRIPTION,PACKAGE_ID,SUPERCLASS)
91 | VALUES ('1','Mammifero','PUBLIC','Mammifero','/zoo/alpha/src/it/zoo/
    animali/Mammifero.java','0','50','ISNEW','F','Classe Generica
    Mammiferi.','1','4');
92 | INSERT INTO CLASS_BASE (CLASS_ID,NAME,SCOPE,FILENAME,PATH,STARTLINE,
    ENDLINE,MONITOR,CHANGEDMETHOD,DESCRIPTION,PACKAGE_ID,SUPERCLASS)
93 | VALUES ('2','Leone','PUBLIC','Leone','/zoo/alpha/src/it/zoo/animali/
    Leone.java','1','60','ISNEW','F','Il re della savana.','1','1');
94 | INSERT INTO CLASS_BASE (CLASS_ID,NAME,SCOPE,FILENAME,PATH,STARTLINE,
    ENDLINE,MONITOR,CHANGEDMETHOD,DESCRIPTION,PACKAGE_ID,SUPERCLASS)
95 | VALUES ('3','Gnu','PUBLIC','Gnu','/zoo/alpha/src/it/zoo/animali/Gnu.
    java','0','100','ISNEW','F','Classe Gnu.','1','1');
96 | INSERT INTO CLASS_BASE (CLASS_ID,NAME,SCOPE,FILENAME,PATH,STARTLINE,
    ENDLINE,MONITOR,CHANGEDMETHOD,DESCRIPTION,PACKAGE_ID)
97 | VALUES ('4','Animale','PUBLIC','Animale','/zoo/alpha/src/it/zoo/animali
    /Animale.java','0','100','ISNEW','F','Classe generica di tutti gli
    Animali.','1');
98 | INSERT INTO CLASS_BASE (CLASS_ID,NAME,SCOPE,FILENAME,PATH,STARTLINE,
    ENDLINE,MONITOR,CHANGEDMETHOD,DESCRIPTION,PACKAGE_ID,SUPERCLASS)
99 | VALUES ('5','Rettile','PUBLIC','Rettile','/zoo/alpha/src/it/zoo/animali
    /Rettile.java','0','80','ISNEW','F','Classe Generica Rettile','1','4
    ');
100 | INSERT INTO CLASS_BASE (CLASS_ID,NAME,SCOPE,FILENAME,PATH,STARTLINE,
    ENDLINE,MONITOR,CHANGEDMETHOD,DESCRIPTION,PACKAGE_ID,SUPERCLASS)
101 | VALUES ('6','Cobra','PUBLIC','Cobra','/zoo/alpha/src/it/zoo/animali/
    Cobra.java','0','90','ISNEW','F','Serpente velenoso.','1','5');
102 | INSERT INTO CLASS_BASE (CLASS_ID,NAME,SCOPE,FILENAME,PATH,STARTLINE,
    ENDLINE,MONITOR,CHANGEDMETHOD,DESCRIPTION,PACKAGE_ID,SUPERCLASS)
103 | VALUES ('7','Varano','PUBLIC','Varano','/zoo/alpha/src/it/zoo/animali/
    Varano.java','0','50','ISNEW','F','Lucertola molto cresciuta.','1','
    5');
104 | INSERT INTO CLASS_BASE (CLASS_ID,NAME,SCOPE,FILENAME,PATH,STARTLINE,
    ENDLINE,MONITOR,CHANGEDMETHOD,DESCRIPTION,PACKAGE_ID,SUPERCLASS)
105 | VALUES ('13','Pesce','PUBLIC','Pesce','/zoo/alpha/src/it/zoo/animali/
    Pesce.java','0','100','ISNEW','F','Classe Generica Pesci.','1','4');
106 |
107 | -- -- Classi del pacchetto it.zoo.strutture
108 | INSERT INTO CLASS_BASE (CLASS_ID,NAME,SCOPE,FILENAME,PATH,STARTLINE,
    ENDLINE,MONITOR,CHANGEDMETHOD,DESCRIPTION,PACKAGE_ID,SUPERCLASS)
109 | VALUES ('8','Recinto','PUBLIC','Recinto','/zoo/alpha/src/it/zoo/
    strutture/Recinto.java','0','50','ISNEW','F','Zona recintata adatta
    a contenere mammiferi.','2','11');
110 | INSERT INTO CLASS_BASE (CLASS_ID,NAME,SCOPE,FILENAME,PATH,STARTLINE,
    ENDLINE,MONITOR,CHANGEDMETHOD,DESCRIPTION,PACKAGE_ID,SUPERCLASS)
111 | VALUES ('9','Rettilario','PUBLIC','Rettilario','/zoo/alpha/src/it/zoo/
    strutture/Rettilario.java','0','70','ISNEW','F','Contenitore in
    vetro adatto a contenere rettili.','2','11');
112 | INSERT INTO CLASS_BASE (CLASS_ID,NAME,SCOPE,FILENAME,PATH,STARTLINE,
    ENDLINE,MONITOR,CHANGEDMETHOD,DESCRIPTION,PACKAGE_ID,SUPERCLASS)
113 | VALUES ('10','Acquario','PUBLIC','Acquario','/zoo/alpha/src/it/zoo/
    strutture/Acquario.java','0','50','ISNEW','F','Contenitore in vetro.
    ','2','11');
114 | INSERT INTO CLASS_BASE (CLASS_ID,NAME,SCOPE,FILENAME,PATH,STARTLINE,

```

```

        ENDLINE, MONITOR, CHANGEDMETHOD, DESCRIPTION, PACKAGE_ID)
115 | VALUES ('11', 'Habitat', 'PUBLIC', 'Habitat', '/zoo/alpha/src/it/zoo/
      strutture/Habitat.java', '0', '40', 'ISNEW', 'F', 'Generico Habitat.
      Classe astratta.', '2');
116 |
117 | -- PROGETTO "Hello World"
118 | -- -- Classi del pacchetto it.azienda.hello
119 | INSERT INTO CLASS_BASE (CLASS_ID, NAME, SCOPE, FILENAME, PATH, STARTLINE,
      ENDLINE, MONITOR, CHANGEDMETHOD, DESCRIPTION, PACKAGE_ID)
120 | VALUES ('12', 'Hello', 'PUBLIC', 'Hello', '/hello/hello1/src/it/azienda/
      hello/Hello.java', '0', '50', 'ISNEW', 'F', 'Fornisce funzionalita di
      saluto.', '3');
121 | COMMIT;
122 |
123 |
124 |
125 | -- ATTRIBUTI
126 | -- Attributi di it.zoo.animali.Animale
127 | INSERT INTO ATTRIBUTE (ATTR_ID, NAME, POSITION, TYPE, SCOPE, CLASS)
128 | VALUES ('1', 'Nome', '1', 'String', 'PRIVATE', '4');
129 | INSERT INTO ATTRIBUTE (ATTR_ID, NAME, POSITION, TYPE, SCOPE, CLASS)
130 | VALUES ('2', 'Sesso', '2', 'char', 'PRIVATE', '4');
131 | --Attributi di it.zoo.strutture.Habitat
132 | INSERT INTO ATTRIBUTE (ATTR_ID, NAME, POSITION, TYPE, SCOPE, CLASS)
133 | VALUES ('3', 'Nome', '1', 'String', 'PRIVATE', '11');
134 | --Attributi di it.zoo.strutture.Acquario
135 | INSERT INTO ATTRIBUTE (ATTR_ID, NAME, POSITION, TYPE, SCOPE, CLASS)
136 | VALUES ('4', 'Volume', '1', 'double', 'PRIVATE', '10');
137 | --Attributi di it.zoo.strutture.Retttilario
138 | INSERT INTO ATTRIBUTE (ATTR_ID, NAME, POSITION, TYPE, SCOPE, CLASS)
139 | VALUES ('5', 'Temperatura', '1', 'double', 'PRIVATE', '9');
140 | COMMIT;
141 |
142 | -- METODI
143 | -- metodo aggiungi(Animale anim) di habitat
144 | INSERT INTO METHOD_BASE (METHOD_ID, SCOPE, NAME, TYPE, STARTLINE, ENDLINE,
      MONITOR, DESCRIPTION, CLASS)
145 | VALUES ('1', 'PUBLIC', 'aggiungi', 'void', '20', '10', 'ISNEW', 'Aggiunge un
      animale all''habitat.', '11');
146 |
147 | -- metodo saluta(String nome, String lingua) di hello
148 | INSERT INTO METHOD_BASE (METHOD_ID, SCOPE, NAME, TYPE, STARTLINE, ENDLINE,
      MONITOR, DESCRIPTION, CLASS)
149 | VALUES ('2', 'PUBLIC', 'saluta', 'String', '10', '10', 'ISNEW', 'Ritorna una
      stringa di saluto.', '12');
150 | COMMIT;
151 |
152 |
153 | -- PARAMETRI
154 | -- metodo aggiungi()
155 | INSERT INTO PARAMETER (PARAM_ID, NAME, POSITION, TYPE, METHOD) VALUES ('1',
      'anim', '1', 'Animale', '1');
156 | --metodo saluta()
157 | INSERT INTO PARAMETER (PARAM_ID, NAME, POSITION, TYPE, METHOD) VALUES ('2',
      'persona', '1', 'String', '2');
158 | INSERT INTO PARAMETER (PARAM_ID, NAME, POSITION, TYPE, METHOD) VALUES ('3',
      'lingua', '2', 'String', '2');
159 | COMMIT;

```

```

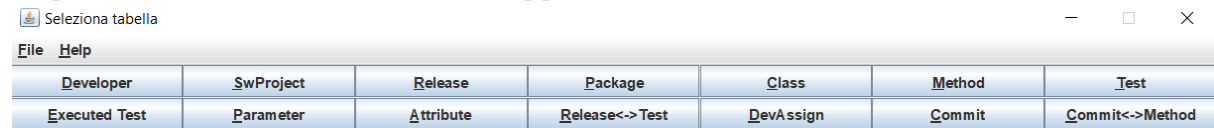
160 |
161 | --TEST
162 | --aggiungi Test di mammiferi con coda
163 | INSERT INTO TEST (TEST_ID,DIRECTORY,DESCRIPTION,FILENAME,SWPROJECT)
164 | VALUES ('1','test','I mammiferi hanno la coda?','Mammifero','1');
165 | COMMIT;
166 |
167 | --TEST ESEGUITI
168 | --aggiungi Test ESEGUITO di mammiferi con coda
169 | INSERT INTO EXECUTED_TEST (EXECUTION_ID,RESULT,DATATEST,USEDSTRUCTURE,
    TEST)
170 | VALUES ('1','T',SYSDATE,'CLASSE:Mammifero, SOTTOCLASSI:Leone,Gnu','1');
171 | COMMIT;
172 |
173 | --ASSOCIAZIONE MOLTI A MOLTI RELEASE<->TEST ESEGUITI
174 | INSERT INTO RELEASE_TEST(EXECUTEDTEST, RELEASE)
175 | VALUES ('1','1');
176 | COMMIT;
177 |
178 | --COMMIT
179 | --aggiungi i commit per classi e metodi
180 | INSERT INTO COMMIT_T (COMMIT_ID,INSDATE,DESCRIPTION,CLASS,AUTHOR)
181 | VALUES ('1',SYSDATE,'Commit classe Habitat','11','1');
182 | INSERT INTO COMMIT_T (COMMIT_ID,INSDATE,DESCRIPTION,CLASS,AUTHOR)
183 | VALUES ('2',SYSDATE,'Commit classe Hello','12','1');
184 |
185 | INSERT INTO COMMIT_METH(COMMIT_ID,METHOD)
186 | VALUES ('1','1');
187 | INSERT INTO COMMIT_METH(COMMIT_ID,METHOD)
188 | VALUES ('2','2');
189 | COMMIT;

```

5.2 Esempio d'Uso

5.2.1 Inserimento Record

Dopo l'autenticazione col DataBase apparirà la finestra di **selezione delle tabelle**.



Selezionare la tabella **desiderata** cliccando il bottone corrispondente. (es. Developer)

The screenshot shows a window titled "Developers" with a menu bar containing "File" and "Help". Below the menu is a form with several input fields and buttons.

Buttons: New, Open, Save, Search, Delete, Cancel

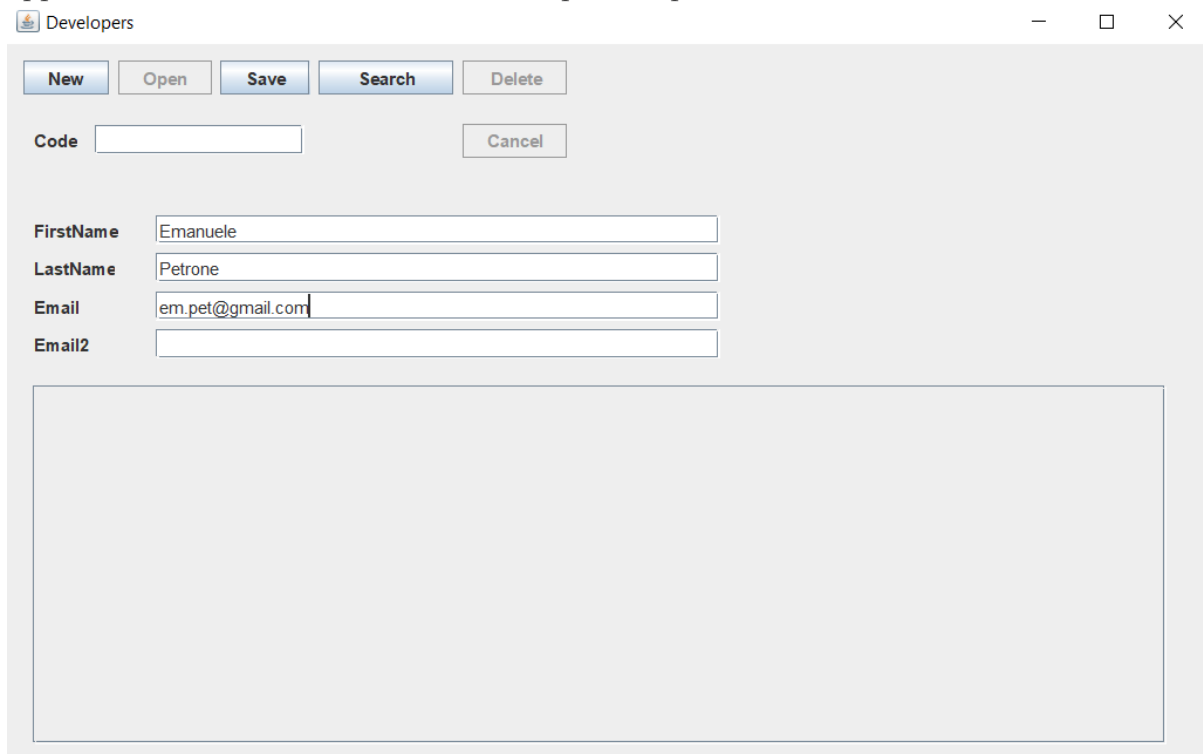
Code:

Fields:

- FirstName:
- LastName:
- Email:
- Email2:

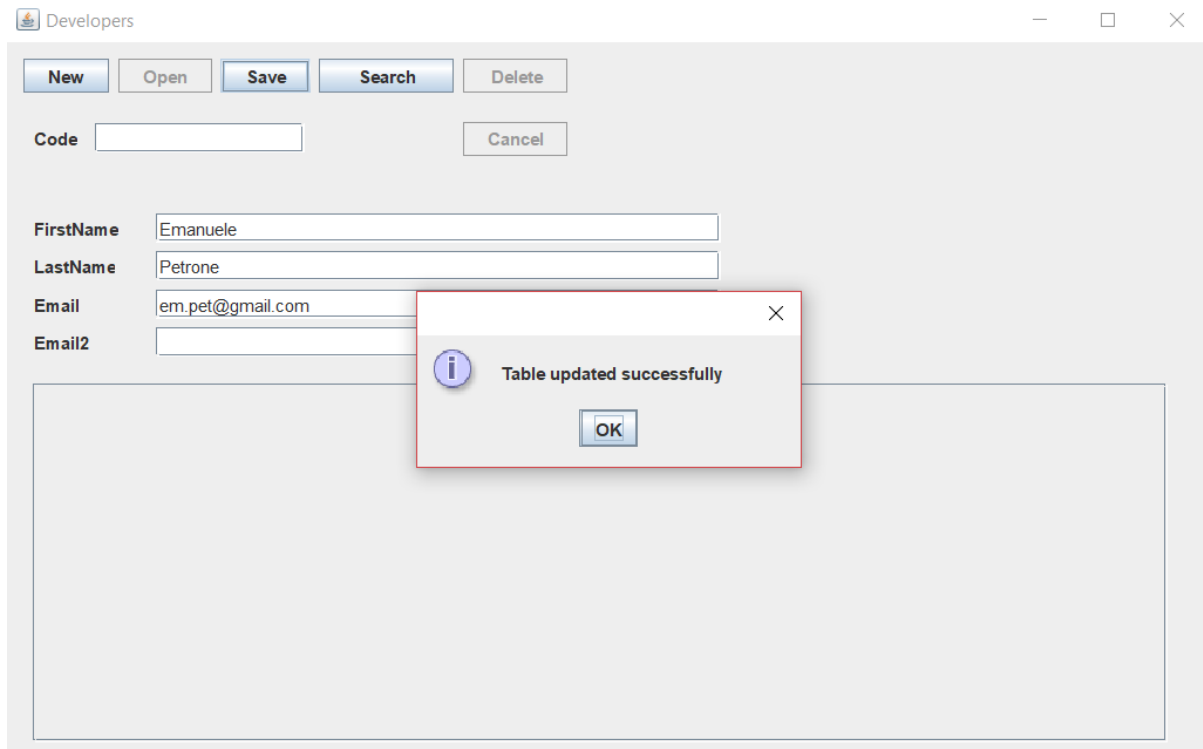
Below the fields is a large empty rectangular box.

Ora Cliccare su **NEW** per inserire il nuovo record, poi immettere i dati rispettando opportunamente i vincoli descritti nei capitoli sopra.



The screenshot shows a window titled 'Developers' with standard Windows window controls (minimize, maximize, close). The window contains a toolbar with buttons: 'New' (highlighted in blue), 'Open', 'Save', 'Search', and 'Delete'. Below the toolbar is a 'Code' input field and a 'Cancel' button. The main area contains four labeled text input fields: 'FirstName' (containing 'Emanuele'), 'LastName' (containing 'Petrone'), 'Email' (containing 'em.pet@gmail.com'), and 'Email2' (empty). Below these fields is a large, empty rectangular area, likely for a list or details.

Infine Cliccare su **Save**.



This screenshot shows the same 'Developers' window as before, but with a modal dialog box displayed in the center. The dialog has a title bar with a close button (X). It contains an information icon (i) and the text 'Table updated successfully'. At the bottom of the dialog is an 'OK' button. The background window is slightly dimmed, showing the same toolbar and input fields as in the previous screenshot.