

Progettazione e sviluppo di una Base di Dati relazionale per la gestione di un’Agenzia di Procuratori Sportivi

Vincenzo Noviello Francesco Rogo

N8600 N86003393

luglio 18, 2021

*Questa pagina `e stata lasciata intenzionalmente bianca.*

**Indice**

|  |  |
| --- | --- |
| **1 Descrizione del progetto** | **4** |
| 1.1 Descrizione sintetica e analisi del problema ( ok ). . . . . . . . . . . . . . . . . | 4 |
| 1.2 Il Repository Software ( no ). . . . . . . . . . . . . . . . . . . . . . . . . . . . | 4 |
| **2 Progettazione Concettuale** | **6** |
| 2.1 Introduzione ( ok ). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 6 |
| 2.2 Class Diagram ( no ). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 7 |
| 2.3 Ristrutturazione del Class Diagram ( no ) . . . . . . . . . . . . . . . . . . . . . | 8 |
| 2.4 Analisi delle ridondanze ( no ). . . . . . . . . . . . . . . . . . . . . . . . . . . | 8 |
| 2.5 Analisi degli identificativi ( no ). . . . . . . . . . . . . . . . . . . . . . . . . . | 8 |
| 2.6 Rimozione degli attributi multipli( no ) . . . . . . . . . . . . . . . . . . . . . . | 8 |
| 2.7 Rimozione delle classi di associazione( no ) . . . . . . . . . . . . . . . . . . . . | 8 |
| 2.8 Rimozione delle gerarchie di specializzazione ( no ). . . . . . . . . . . . . . . . | 8 |
| 2.9 Class Diagram Ristrutturato ( no ). . . . . . . . . . . . . . . . . . . . . . . . . | 9 |
| 2.10 Dizionario delle classi ( ok ). . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 10 |
| 2.11 Dizionario delle Associazioni ( no ) . . . . . . . . . . . . . . . . . . . . . . . . . | 13 |
| 2.12 Dizionario dei Vincoli ( no ). . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 15 |
| **3 Progettazione Logica** | **17** |
| 3.1 Schema logico ( no ). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 17 |
| **4 Progettazione fisica** ( no ) | **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](#_Toc135573)

[4.2.1 CLASSNAMES 35](#_Toc135574)

[4.2.2 TESTNAMES 35](#_Toc135575)

[4.3 Funzioni, Procedure ed altre Automazioni 36](#_Toc135576)

[4.3.1 Calcolo automatico del path di una classe 36](#_Toc135577)

[4.3.2 Calcolo automatico del path di un file sorgente 36](#_Toc135578)

[4.3.3 Verificare se un tipo `e valido 37](#_Toc135579)

[4.3.4 Verificare se un FileName `e valido per un Caso di Test 38](#_Toc135580)

[4.4 Implementazione dei Vincoli 39](#_Toc135581)

[4.4.1 Implementazione del vincolo **Distinct Personal Mails** 39](#_Toc135582)

[4.4.2 Implementazione del vincolo **TypeValues** 40](#_Toc135583)

[4.4.3 Implementazione del vincolo **Name Consistency for Test** 41](#_Toc135584)

[4.4.4 Implementazione del vincolo **Single Public Class per File e**](#_Toc135585)

[**File structure consistency for Class** 42](#_Toc135586)

[4.4.5 Implementazione del vincolo **File consistency for methods** 44](#_Toc135587)

[4.4.6 Implementazione del vincolo **Commit consistency** 45](#_Toc135588)

[**5 Manuale d’Uso** ( no ) **46**](#_Toc135589)

[5.1 Popolamento con Dati di Esempio 46](#_Toc135590)

[5.2 Esempio d’Uso 51](#_Toc135591)

[5.2.1 Inserimento Record 51](#_Toc135592)

**Capitolo 1**

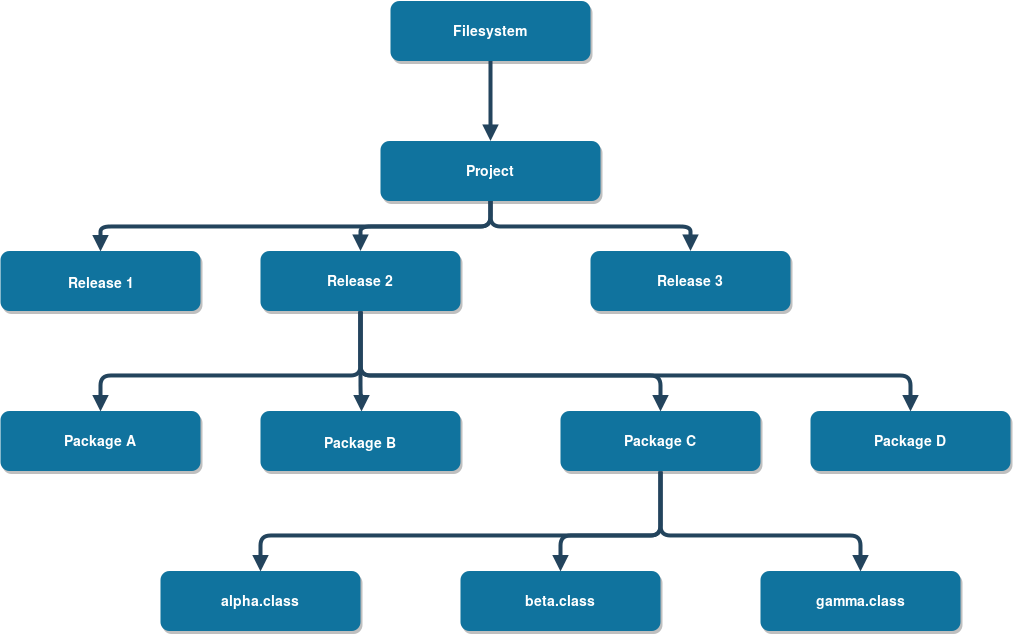
**Descrizione del progetto**

**1.1 Descrizione sintetica e analisi del problema**

Si deve progettare e implementare una base di dati relazionale che possa essere d’aiuto alla gestione di un’agenzia di procuratori sportivi. La base di dati conterrà le informazioni riguardanti i di diversi procuratori sportivi e le relazioni contrattuali tra gli atleti e le aziende, quali Club e Sponsor. Il sistema permetterà di creare account per le aziende agli amministratori del sistema, creare account per gli Atleti-Tesserati e Manager ai semplici utenti nonché associare a ciascun Manager un Atleta-Tesserato e vice versa, creare contratti tra le aziende e gli Atleti-Tesserati e fornisce informazioni sui dati in possesso dagli utenti. Ciascun utente avrà diverse possibilità: un Manager potrà avere un recap della situazione al momento ( quale numero atleti seguiti e i loro contratti associati ordinati per maggior guadagno per il Manager, associare un Atleta-Tesserato e dissociarlo ); l’Atleta-Tesserato invece può: chiedere un riepilogo dei dati ( quale numero dei contratti e le loro specifiche ), creare un contratto con un’azienda, aggiungere un Manager se non lo avesse ancora o lo volesse cambiare ed eliminarlo per dissociarlo dalla persona; in oltre i super utenti, quali amministratori del sistema possono: creare ed eliminare utenti ( Manager e Atleti-Tesserati ) e aziende ( Club e Sponsor ), hanno la possibilità di visualizzare la lista di utenti e le loro informazioni.

Le successive richieste da soddisfare dalla traccia sono: poter effettuare operazioni complesse all’interno della base dati, come a esempio calcolare il contratto che faceva fruttare di più a un determinato Manager o la maggior fonte d’introito per l’Atleta-Tesserato.

**1.2 non so che mettere qui**



**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 potr`a riscrivere tutto il codice.

**Sviluppo.** Tutti gli sviluppatori abilitati a lavorare al progetto effetturano modifiche (commit) ai sorgenti. Il sistema tiene traccia degli autori e delle tempistica 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, calcoler`a 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 verra` calcolato e salvato attraverso opportuni trigger in un log.

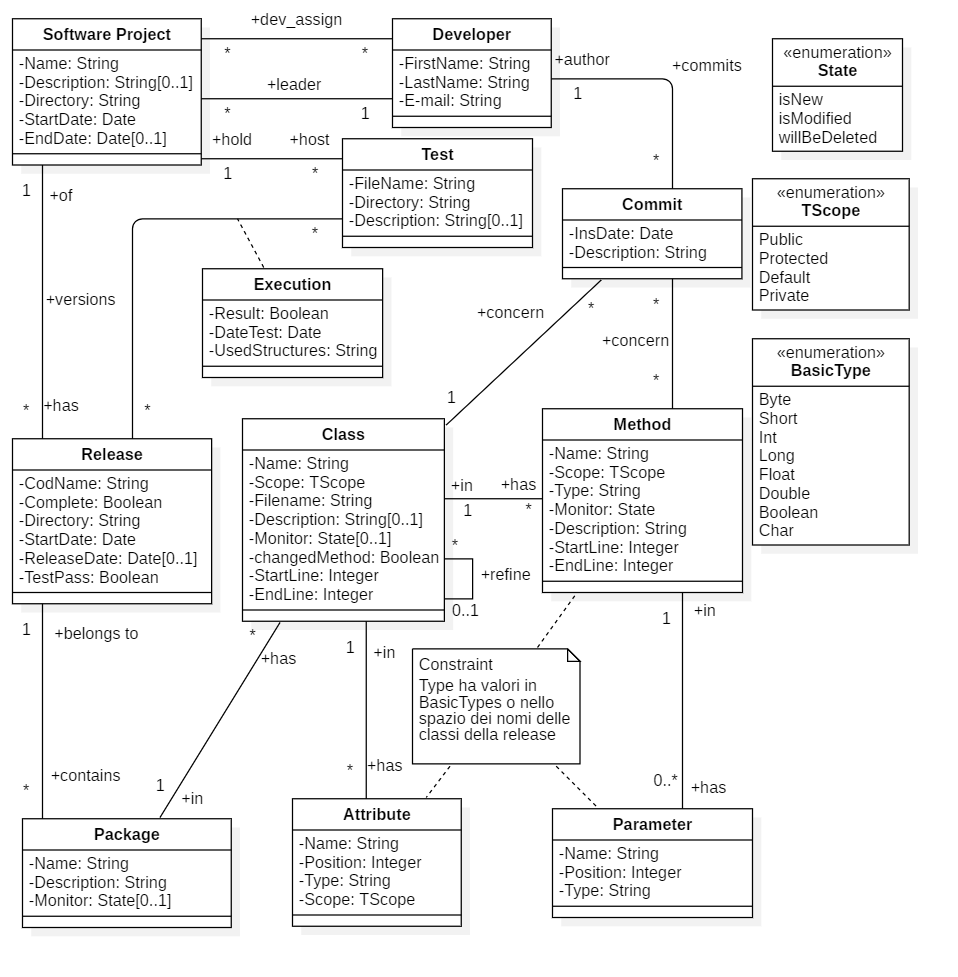
**Capitolo 2**

**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 ( work in progress )**



**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 conterra` 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 `e definita la classe. Calcolare tale informazione un’unica volta `e vantaggioso rispetto al doverla ricavare da molteplici prodotti ogni volta, specialmente dal momento che si pu`o 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 entita`. Tali chiavi tecniche altro non saranno che identificativi numerici che permetteranno di discriminare con maggiore facilita` 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).

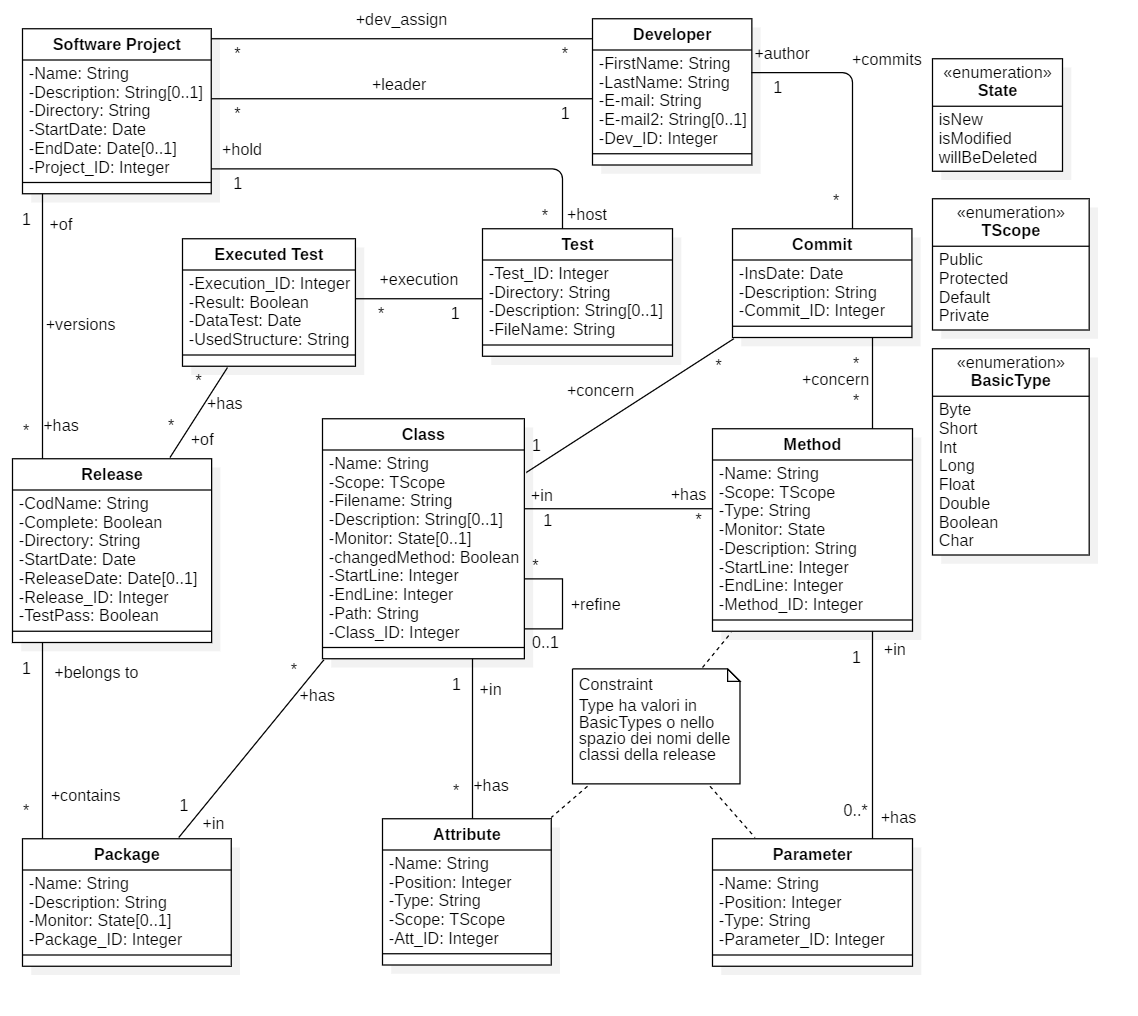
**2.7 Rimozione delle classi di associazione**

Si procede con l’eliminazione della classe di associazione **EXECUTION** e la reintroduzione della stessa come classe vera e propria contenente gli stessi attributi piu` 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** (*string*): nome del progetto.  **Description** (*string, opzionale*): breve descrizione del progetto.  **Directory** (*string*): nome della directory radice del progetto all’interno della radice del repository. **StartDate** (*date*): data di creazione del progetto.  **EndDate** (*date, opzionale*): data di fine del progetto.  **Project ID** (*integer*): chiave tecnica. Identifica univocamente ciascun progetto nel repository. |
| **Release** | Descrittore di ogni versione rilasciata del progetto | **CodName** (*string*): nome interno della release. **Complete** (*boolean*): indica se la release `e da considerarsi completa oppure in fase di sviluppo. **Directory** (*string*): nome della directory radice della release all’interno della radice del progetto. **StartDate** (*date*): data di inizio dei lavori per la release.  **ReleaseDate** *(date, opzionale)*: indica la data (eventuale) in cui la release `e stata indicata come completa.  **Release ID** (*integer*): chiave tecnica. identifica univocamente ciascuna release del progetto. **TestPass** (*boolean*): indica se la release a passato con succeso tutti i casi di test del progetto. |
| **Package** | Descrittore di Package appartenenti ad una release. | **Name** (*string*): nome del pacchetto. Coincide con il nome della directory che contiene le classi del pacchetto.  **Description** (*string*): breve descrizione del contenuto del Package  **Monitor** (*State, opzionale*): indica se il pacchetto `e nuovo, se `e stato modificato dalla release precedente oppure verr`a eliminato alla prossima release.  **Package ID** (*integer*): identifica univocamente ogni istanza di Package. |

|  |  |  |
| --- | --- | --- |
| **Classe** | **Descrizione** | **Attributi** |
| **Class** | Descrittore di una classe in un progetto. | **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 `e definita la classe.  **Description** (*string, opzionale*): breve descrizione esplicativa della classe.  **Monitor** (*State, opzionale*): indica se la classe  `e nuova, se `e stata modificata dalla release precedente oppure verra` eliminata alla prossima release.  **changedMethod** (*boolean, derivato*): indica se almeno un metodo della classe ha subito modifiche all’implementazione, ha cambiato segnatura, `e stato aggiunto o eliminato.  **StartLine** (integer): indica la linea dove inizia la definizione della classe nel file .java.  **EndLine** (*integer*): indica la linea dove finisce la definizione della classe.  **Path** (*string*): indica il percorso completo, relativo alla directory root del repository, del file.  **Class ID** (*integer*): identifica univocamente ciascuna istanza di Class. |

**Attribute** Descrittore di un attributo (campo) di una classe.

**Name** (*string*): name binding dell’attributo. **Position** (*integer*): indica la posizione dell’attributo.

**Developer** Descrittore di sviluppatore abilitato a lavorare nel repository

**Type** (*string*): indica il tipo dell’attributo.

**Scope** (*TScope*): indica lo scope dell’attributo. **Att ID** (*integer*): identifica univocamente un attributo.

**FirstName** (*string*): nome dello sviluppatore. **LastName** (*string*): cognome dello sviluppatore. **eMail** (*string*): indirizzo email principale dello sviluppatore. **eMail2** (*string*): indirizzo email secondario (di recupero).

**Dev ID** (*integer*).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Classe** | **Descrizione** | | | **Attributi** |
| **Method** Descrittore del metodo di una classe. | |  |  | **Name** (*string*): nome del metodo.  **Scope** (*TScope*): indica lo scope del metodo. **Type** (*string*): indica il tipo del valore ritornato dal metodo.  **Monitor** (*State)*: breve descrizione esplicativa della classe.  **Monitor** (*State, opzionale*): indica se il metodo  `e nuovo, se `e stato modificato dalla release precedente oppure verra` eliminato alla prossima release.  **Description** (*string*): breve descrizione esplicativa del metodo.  **StartLine** (integer): indica la linea dove inizia la definizione del metodo nel file .java.  **EndLine** (*integer*): indica la linea dove finisce la definizione del metodo.  **Method ID** (*integer*): identifica univocamente un metodo. |
| **Parameter** | Descrittore parametro metodo. | di | di un | **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.  **Param ID** (*integer*): indentificativo del parametro. |
| **Commit** | Descrittore di un’istanza di modifica ai codici sorgente di una classe da parte di uno sviluppatore. | | | **InsDate** (*date*): marca temporale della modifica. **Description** (*string*): descrizione della modifica apportata.  **Commit ID** (*integer*): identifica univocamente ciascuna operazione di modifica ad una classe. |
| **Test** | Descrittore del caso di Test ospitato dal progetto | | | **Test ID** (*integer*): identificativo univoco del file di test.  **Directory** (*string*): nome directory radice dove `e ospitato il caso di test.  **Description** (*string, opzionale*): breve descrizione dei test contenuti nel file.  **Filename** (*string*): nome del caso di test. |

**Executed** Descrittore di un caso **Executio ID** (*integer*): chiave tecnica.

**Test** di Test eseguito **Result** (*boolean*): esito del test.

**DataTest** (*date*): marca temporale in cui il test

`e eseguito.

**UsedStructure** (*string*): 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 appertiene 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 As-**  **sign** | Esprime la possibilit`a di uno sviluppatore di lavorare ad un progetto. | **Software Project [0..\*]** ruolo **assigned to**: indica il progetto a cui `e 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 `e 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 llo 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 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. |
| **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 alpiu` 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 Consis-**  **tency** | Ciascun commit puo` interessare una sola classe. Tutti i metodi interessati dalla modifica, quindi, devono appartenere alla stessa classe, cio`e quella modificata. |
| **No Auto-refine** | Nella relazione **refine** una istanza di classe non pu`o essere sottoclasse/superclasse di se stessa. In altre parole, l’istanza di classe in ruolo **extends** non pu`o 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** `e 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 Con-** In ciascun file sorgente non `e possibile che esistano due **sistency for Classes** classi **C1** e **C2** tali che valga la seguente:

**C1.StartLine***>***C2.StartLine***>***C1.EndLine***>***C2.EndLine.** In altre parole, `e sempre rispettata la corretta strutturazione in blocchi. Il caso in cui una classe sia dichiarata all’interno di un’altra `e 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 pu`o 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 Composi-**  **tion** | All’interno della stessa release non possono esistere due Packages con nome (e quindi cartella) uguale. |
| **Package Composi-**  **tion** | 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 sara` trattata la fase successiva della progettazione della base di dati scendendo ad un livello di astrazione piu` basso rispetto alla precedente.

Si tradurr`a lo **schema concettuale** (gia` predisposto in seguito alla ristrutturazione) in uno **schema logico**, dipendente dal tipo di struttura dei dati prescelto cio`e 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, changedMethod, 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, **Method ID**, Class)  Class→Class.Class ID |
| **Parameter** | (Name, Position, Type, **Parameter ID**, Method)  Method→Method ID |
| **Attribute** | (Name, Position, Type, Scope, **Att ID**, Class  Class→Class.Class ID) |
| **Test** | (**Test ID**, Directory, Description, FileName, SwProject) |

|  |  |
| --- | --- |
|  | SwProject→SoftwareProject.Project ID |
| **ExecutedTest** | (**Execution ID**, Result, DataTest, UsedStructure, Test)  Test→Test.Test ID |
| **ReleaseTest** | (ExecutedTest, Release)  ExecutedTest→ExecutedTest.Execution ID,  Release→Release.Release ID |
| **DevAssign** | (Dev, Software)  Dev→Developer.Dev ID, Software→SoftwareProject.Project ID |
| **CommitMethod** | (Commit, Method) |

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 funzionalita` del DBMS **Oracle XE 11g**. Poich´e Oracle non implementa il tipo **boolean**, questo `e 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 piu` 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  eMail2 IS NULL) | | |
| 14 | | -- LEGIT MAILS | | |
| 15 | | -- una email legittima puo’ contare {64}@{255} caratteri. | | |
| 16 | | -- In totale 320. |  | |
| 1. | 2. | | ); |  | |
| 19 | | / |  | |
| 20 | | -- Crea il vincolo di chiave primaria |  | |
| 21 | | ALTER TABLE DEVELOPER\_BASE |  | |
| 22 | | ADD CONSTRAINT Dev\_pk PRIMARY KEY(Dev\_ID); |  | |
| 1. | 2. | | / |  | |
| 25 | | -- Trigger per settare, se necessario, la chiave automaticamente | primaria | |
| 26 | | CREATE OR REPLACE TRIGGER DeveloperPK |  |  |
| 27 | | BEFORE INSERT ON DEVELOPER\_BASE |  |  |
| 28 | | FOR EACH ROW |  |  |
| 29 | | BEGIN |  |  |
| 30 | | DECLARE |  |  |
| 31 | | pk DEVELOPER\_BASE.Dev\_ID%TYPE; |  |  |
| 32 | | BEGIN |  |  |
| 33 | | IF(:NEW.Dev\_ID = 0) THEN |  |  |
| 34 | | SELECT NVL(MAX(Dev\_ID),0) + 1 INTO pk FROM  ; | | DEVELOPER\_BASE |
| 35 | | :NEW.Dev\_ID := pk; | |  |
| 36 | | END IF; | |  |
| 37 | | END; | |  |
| 1. | 2. | | 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 pi´u semplici. |  |
| 1. | 2. | 3. | | \*/ |  |
| 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 |  |
| 1. | 2. | | ); |  |
| 1. | 2. | | / |  |
| 64 | | -- Aggiunge i vincoli |  |
| 65 | | ALTER TABLE SOFTWARE\_PROJECT ADD( |  |
| 66 | | -- Chiave primaria |  |
| 67 | | CONSTRAINT project\_pk PRIMARY KEY (Project\_ID), -- Project\_ID  NULL | NOT |
| 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; | |
| 1. | 2. | | END; | |
| 94 | | / | |

##### 4.1.3 Definizione della Tabella RELEASE

|  |  |  |
| --- | --- | --- |
| 95 | | /\*\* | |
| 96 | | \* TABELLA: RELEASE | |
| 97 | | \* Crea la tabella e implementa i vincoli piu semplici. | |
| 1. | 2. | | \*/ | |
| 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 | |
| 1. | 2. | | ); | |
| 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) | |
| 1. | 2. | | ); |  |
| 1. | 2. | | / |  |
| 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; |  |
| 1. | 2. | | END; |  |
| 148 | | / |  |

##### 4.1.4 Definizione della Tabella PACKAGE

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 149 | | /\*\* | | | |
| 150 | | \* TABELLA: PACKAGE | | | |
| 151 | | \* Crea la tabella e implementa i vincoli piu semplici. | | | |
| 1. | 2. | | \*/ | |  | |
| 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  NULL, | | inserita.’ NOT | |
| 161 | | Release INTEGER NOT NULL | |  | |
| 1. | 2. | | ); | |  | |
| 1. | 2. | | / | |  | |
| 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 release. | nome nella | | stessa |
| 179 | | CONSTRAINT unique\_PackDir UNIQUE(Name, Release) |  | |  |
| 1. | 2. | | ); |  | |  |
| 1. | 2. | | / |  | |  |
| 184 | | -- Trigger per settare, se necessario, la chiave automaticamente | primaria | |  |
| 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; | | |  |
| 1. | 2. | | END; | | |  |
| 199 | | / | | |  |

##### 4.1.5 Definizione della Tabella CLASS

|  |  |  |  |
| --- | --- | --- | --- |
| 200 | | /\*\* | | |
| 201 | | \* TABELLA: CLASS | | |
| 202 | | \* Crea la tabella e implementa i vincoli piu’ semplici. | | |
| 1. | 2. | 3. | | \*/ | | |
| 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 |  | |
| 1. | 2. | | ); |  | |
| 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 | |  |
| 1. | 2. | | CREATE OR REPLACE VIEW CLASS AS SELECT \* FROM CLASS\_BASE; | |  |
| 250 | | / | |  |

##### 4.1.6 Definizione della Tabella METHOD

|  |  |  |  |
| --- | --- | --- | --- |
| 251 | | /\*\* | | |
| 252 | | \* TABELLA: METHOD | | |
| 253 | | \* Crea la tabella e implementa i vincoli piu’ semplici. | | |
| 1. | 2. | 3. | | \*/ |  | |
| 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  NULL, | inserita.’ NOT | |
| 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 |  |  |
| 1. | 2. | | ); |  |  |
| 1. | 2. | | / |  |  |
| 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 metodi | anche | tutti i suoi |
| 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 | | ) | | |
| 1. | 2. | | ); | | |
| 289 | | / | | |
| 290 | | -- Per futuri Trigger INSTEAD OF | | |
| 1. | 2. | | CREATE OR REPLACE VIEW METHOD AS SELECT \* FROM METHOD\_BASE; | | |
| 293 | | / | | |

##### 4.1.7 Definizione della Tabella PARAMETER

|  |  |  |  |
| --- | --- | --- | --- |
| 294 | | /\*\* | | |
| 295 | | \* TABELLA: PARAMETER | | |
| 296 | | \* Crea la tabella e implementa i vincoli piu’ semplici. | | |
| 1. | 2. | 3. | | \*/ |  | |
| 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 |  | |
| 1. | 2. | | ); |  | |
| 1. | 2. | | / |  | |
| 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 parametri | tutti i suoi | |
| 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) | |  |
| 1. | 2. | | ); | |  |
| 1. | 2. | | / | |  |
| 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; | | |
| 1. | 2. | | END; | | |
| 341 | | / | | |

##### 4.1.8 Definizione della Tabella ATTRIBUTE

|  |  |  |  |
| --- | --- | --- | --- |
| 342 | | /\*\* | |  |
| 343 | | \* TABELLA: ATTRIBUTE | |  |
| 344 | | \* Crea la tabella e implementa i vincoli piu’ semplici. | |  |
| 1. | 2. | | \*/ |  |  |
| 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 |  |  |
| 1. | 2. | | ); |  |  |
| 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 suoi attributi | anche | tutti i |
| 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’)) | | |
| 1. | 2. | | ); | | |
| 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. | |  |
| 1. | 2. | | \*/ |  |  |
| 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 |  |  |
| 1. | 2. | | ); |  |  |
| 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 modifiche | anche | tutte le |
| 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 | | |
| 1. | 2. | | ); | | |
| 1. | 2. | | / | | |
| 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; | | |
| 1. | 2. | | END; | | |
| 438 | | / | | |

##### 4.1.10 Definizione della Tabella TEST

|  |  |  |  |
| --- | --- | --- | --- |
| 439 | | /\*\* | | |
| 440 | | \* TABELLA: TEST | | |
| 441 | | \* Crea la tabella e implementa i vincoli piu’ semplici. | | |
| 1. | 2. | 3. | | \*/ |  |  |
| 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  NULL, | descrizione | inserita.’ NOT |
| 451 | | Filename VARCHAR2(64) NOT NULL, |  |  |
| 452 | | SwProject INTEGER NOT NULL |  |  |
| 1. | 2. | | ); |  |  |
| 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 | | |
| 1. | 2. | | ); | | |
| 1. | 2. | | / | | |
| 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; | | |
| 1. | 2. | | END; | | |
| 485 | | / | | |

##### 4.1.11 Definizione della Tabella EXECUTED TEST

|  |  |  |  |
| --- | --- | --- | --- |
| 486 | | /\*\* | | |
| 487 | | \* TABELLA: EXECUTED\_TEST | | |
| 488 | | \* Crea la tabella e implementa i vincoli piu’ semplici. | | |
| 1. | 2. | 3. | | \*/ | | |
| 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 | |  |
| 1. | 2. | | ); | |  |
| 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 |  |  |
| 1. | 2. | | ); |  |  |
| 515 | | / |  |  |
| 516 | | -- Trigger per settare, se necessario, la chiave automaticamente | primaria |  |
| 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; | | |
| 1. | 2. | | END; | | |
| 531 | | / | | |

##### 4.1.12 Definizione della Tabella RELEASE TEST

|  |  |  |  |
| --- | --- | --- | --- |
| 532 | | /\*\* | |  |
| 533 | | \* TABELLA: RELEASE\_TEST | |  |
| 534 | | \* Crea la tabella e implementa i vincoli piu’ semplici. | |  |
| 1. | 2. | 3. | | \*/ |  |  |
| 538 | | -- Crea la tabella RELEASE\_TEST |  |  |
| 539 | | CREATE TABLE RELEASE\_TEST |  |  |
| 540 | | ( |  |  |
| 541 | | ExecutedTest INTEGER NOT NULL, |  |  |
| 542 | | Release INTEGER NOT NULL |  |  |
| 1. | 2. | | ); |  |  |
| 1. | 2. | 3. | | / |  |  |
| 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 le sue associazioni | eliminate | anche tutte |
| 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 | | |
| 1. | 2. | | ); | | |
| 563 | | / | | |

##### 4.1.13 Definizione della Tabella COMMIT METHOD

|  |  |  |
| --- | --- | --- |
| 564 | | /\*\* | |
| 565 | | \* TABELLA: COMMIT\_METH | |
| 566 | | \* Crea la tabella e implementa i vincoli piu’ semplici. | |
| 1. | 2. | 3. | | \*/ |  |
| 570 | | -- Crea la tabella COMMIT\_METH |  |
| 571 | | CREATE TABLE COMMIT\_METH |  |
| 572 | | ( |  |
| 573 | | Commit\_ID INTEGER NOT NULL, |  |
| 574 | | Method INTEGER NOT NULL |  |
| 1. | 2. | | ); |  |
| 1. | 2. | | / |  |
| 579 | | -- Aggiunge i vincoli |  |
| 580 | | ALTER TABLE COMMIT\_METH |  |
| 581 | | ADD ( |  |
| 582 | | -- Chiave esterna verso COMMIT\_T |  |
| 583 | | -- Se viene cancellato un commit vengono associazioni di modifica | cancellate anche le |
| 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 | |
| 1. | 2. | | ); | |
| 594 | | / | |

##### 4.1.14 Definizione della Tabella DEV ASSIGN

|  |  |  |  |
| --- | --- | --- | --- |
| 595 | | /\*\* | | |
| 596 | | \* TABELLA: DEV\_ASSIGN | | |
| 597 | | \* Crea la tabella e implementa i vincoli piu’ semplici. | | |
| 1. | 2. | | \*/ |  |  |
| 600 | | -- Crea la tabella DEV\_ASSIGN |  |  |
| 601 | | CREATE TABLE DEV\_ASSIGN |  |  |
| 602 | | ( |  |  |
| 603 | | Dev INTEGER NOT NULL, |  |  |
| 604 | | Project\_ID INTEGER NOT NULL |  |  |
| 1. | 2. | | ); |  |  |
| 1. | 2. | 3. | | / |  |  |
| 610 | | -- Aggiunge i vincoli |  |  |
| 611 | | ALTER TABLE DEV\_ASSIGN |  |  |
| 612 | | ADD( |  |  |
| 613 | | -- Chiave esterna verso DEVELOPER |  |  |
| 614 | | -- Eliminando uno sviluppatore, vengono associazioni | eliminate | anche tutte le sue |
| 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) | | |
| 1. | 2. | | ); | | |
| 627 | | / | | |

**4.1.15 Definizione della Tabella BASIC TYPES**

1. | /\*\*
2. | \* TABELLA: BASICTYPES
3. | \* Crea la tabella e la popola
4. | \*/
5. |
6. | -- Creazione tabella
7. | CREATE TABLE BASICTYPE
8. | (

###### 636 | Name VARCHAR2(16) NOT NULL

1. | );
2. |
3. | /
4. |
5. | -- 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;

1. |
2. | /

## 4.2 Viste

### 4.2.1 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**.

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;

1. |
2. | /

### 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;

1. |
2. | /

## 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 `e riportata la definizione:

1. | -- Funzione calculate\_path. Calcola il percorso completo del file sorgente filename nel package
2. | -- con ID p. La funzione viene usata nel trigger CALC\_PATH, per calcolare automaticamente il percorso
3. | -- nel file system della classe appena inserita.
4. | CREATE OR REPLACE FUNCTION calculate\_path(filename CLASS.FILENAME%

###### TYPE, p CLASS.PACKAGE\_ID%TYPE)

673 | RETURN VARCHAR2

1. | IS
2. | 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;

1. | END;
2. |
3. | /

### 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**.

1. | -- Trigger calc\_path. Calcola il valore di PATH prima dell’ inserimento di un nuovo record.
2. | 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);

1. | END;
2. | END;
3. |
4. | /

### 4.3.3 Verificare se un tipo `e valido

La funzione **isValidType()** permette di verificare, dato un nome di tipo T e una release R, se il T `e un tipo valido in R. La funzione controlla se T `e 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 (cioe’ | | |
| 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 `e valido per un Caso di Test

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

1. |
2. | CREATE OR REPLACE FUNCTION isValidName(name VARCHAR2, p

###### SOFTWARE\_PROJECT.PROJECT\_ID%TYPE)

1. | RETURN CHAR
2. | IS

724 | found INTEGER; 725 | BEGIN

1. | SELECT COUNT(\*) INTO found --controlla se il NOME e’ presente nelle strutture del progetto
2. | FROM TESTNAMES
3. | 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’;

1. | END IF;
2. | END;
3. |
4. | /

## 4.4 Implementazione dei Vincoli

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

### 4.4.1 Implementazione del vincolo Distinct Personal Mails

1. | -- Descrizione: Lo stesso indirizzo email non pu essere utilizzato da pi
2. | -- sviluppatori.
3. | -- Note : Calcola ed imposta anche la chiave primaria, se

necessario.

1. |
2. | CREATE OR REPLACE TRIGGER checkEmails 742 | INSTEAD OF INSERT OR UPDATE ON DEVELOPER 743 | FOR EACH ROW

###### 744 | BEGIN

745 | DECLARE 746 | TEMP INTEGER:=0; 747 | pk DEVELOPER.DEV\_ID%TYPE;

1. | BEGIN
2. | IF INSERTING THEN -- effettua l’inserimento

###### 750 | SELECT COUNT(\*) INTO TEMP

751 | FROM DEVELOPER\_BASE D 752 | WHERE (D.eMail=:NEW.eMail OR D.eMail2=:NEW.eMail OR 753 | D.eMail=:NEW.eMail2 OR D.eMail2=:NEW.eMail2);

1. | IF(TEMP <> 0) THEN -- Violazioni presenti
2. | RAISE\_APPLICATION\_ERROR(-20001,’Indirizzo eMail gia presente!’);

###### 756 | ELSE

757 | SELECT NVL(MAX(Dev\_ID),0)+1 INTO pk FROM DEVELOPER\_BASE; 758 | INSERT INTO DEVELOPER\_BASE (DEV\_ID,FIRSTNAME,LASTNAME,EMAIL,EMAIL2) 759 | VALUES (pk, :NEW.FIRSTNAME, :NEW.LASTNAME, :NEW.EMAIL, :NEW.EMAIL2)

;

1. | END IF;
2. | ELSE -- UPDATING

###### 762 | SELECT COUNT(\*) INTO TEMP

763 | FROM DEVELOPER\_BASE D 764 | WHERE D.Dev\_ID <> :OLD.DEV\_ID AND (D.eMail=:NEW.eMail OR D.eMail2=: NEW.eMail OR 765 | D.eMail=:NEW.eMail2 OR D.eMail2=:NEW.eMail2);

1. | IF(TEMP <> 0) THEN -- Violazioni presenti
2. | RAISE\_APPLICATION\_ERROR(-20001,’Indirizzo eMail gia presente!’);

###### 768 | ELSE

769 | UPDATE DEVELOPER\_BASE SET 770 | FIRSTNAME = :NEW.FIRSTNAME, 771 | LASTNAME = :NEW.LASTNAME, 772 | EMAIL = :NEW.EMAIL, 773 | EMAIL2 = :NEW.EMAIL2 774 | WHERE DEV\_ID = :OLD.DEV\_ID;

1. | END IF;
2. | END IF;
3. | END;
4. | END;
5. | /

### 4.4.2 Implementazione del vincolo TypeValues

1. | --

------------------------------------------------------------------------------

1. | -- Vincolo : "Type Values"
2. | -- Descrizione: I tipi degli attributi, dei parametri formali dei metodi ed il
3. | -- tipo del valore ritornato da un metodo devono essere valori di 784 | -- in BasicTypes oppure nomi di classi della stessa release.
4. | -- Note : Implementato sulle tabelle ATTRIBUTE, METHOD, PARAMETER con i
5. | -- seguenti tre trigger.
6. | --

------------------------------------------------------------------------------

1. |
2. | -- Implementa il vincolo per ATTRIBUTE
3. | 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!’);

1. | END IF;
2. | END;
3. | END;
4. |
5. | /
6. |
7. | -- Implementa il vincolo per METHOD
8. | 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!’);

1. | END IF;
2. | END;
3. | END;
4. |
5. | /
6. |
7. | -- Implementa il vincolo per PARAMETER
8. | 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!’);

1. | END IF;
2. | END;
3. | END;
4. |
5. | /

### 4.4.3 Implementazione del vincolo Name Consistency for Test

1. | --

--------------------------------------------------------------------------------

1. | -- Vincolo : "Name Consistency for TEST"
2. | -- Descrizione: Il filename dei test deve avere lo stesso valore di una struttura
3. | -- all’interno del progetto ospitante del test. 851 | --

-----------------------------------------------------------------------------

1. |
2. | 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!’);

1. | END IF;
2. | END;
3. | END;
4. | /

### 4.4.4 Implementazione del vincolo Single Public Class per File

#### e File structure consistency for Class

1. | --

------------------------------------------------------------------------------

1. | -- Vincolo : "Single Public Class per File"
2. | -- Descrizione: Ogni file sorgente contiene alpi una classe con scope PUBLIC.
3. | --

------------------------------------------------------------------------------

1. | -- Vincolo : "File structure consistency"
2. | -- Descrizione: In ciascun file sorgente, non possono sovrapporsi definizioni di
3. | -- classi diverse. 874 | --

------------------------------------------------------------------------------

1. | -- Note : Necessitano della definizione di una vista di appoggio per

non

1. | -- incorrere in errori causati da mutating tables. Il seguente 877 | -- trigger implementa entrambi i vincoli. 878 | --

------------------------------------------------------------------------------

1. |
2. |
3. | 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

1. | -- Verifico se sussistono violazioni
2. | -- Conto le classi public nello stesso file sorgente e nello stesso package
3. | 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’;

1. | END IF;
2. | IF( VIOLATION > 0 AND :NEW.SCOPE = ’PUBLIC’) THEN -- c’ gi una classe public in quel file in quel package
3. | RAISE\_APPLICATION\_ERROR(-20003,’In un file sorgente pu esserci un’’ unica classe public!’);
4. | ELSE -- non c’ violazione del vincolo "Single Public Class per File"
5. | -- 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

1. | RAISE\_APPLICATION\_ERROR(-20004,’Violata corretta strutturazione dei blocchi del file.’);
2. | ELSE -- Nessuna violazione. Proseguo
3. | IF INSERTING THEN -- effettua l’inserimento
4. | 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.DESCRIPTION,: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.DESCRIPTION, 930 | PACKAGE\_ID = :NEW.PACKAGE\_ID 931 | WHERE CLASS\_ID = :OLD.CLASS\_ID;

1. | END IF;
2. | END IF;
3. | END IF;
4. | END;
5. | END;
6. |
7. | /

### 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 | | |
| 1. | 2. | | -- trovarsi al di fuori di quella della classe a cui appartiene. | | |
| 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.DESCRIPTION, :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.DESCRIPTION, | | |
| 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

1. | -- Descrizione: Ciascun commit interessa una ed una sola classe. I

metodi

1. | -- associati a ciascun commit devono essere metodi di quella classe

.

1. | --

------------------------------------------------------------------------------

1. |
2. | 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

1. | RAISE\_APPLICATION\_ERROR(-20008, ’Il metodo non appartiene alla classe interessata dal commit’);
2. | END IF;
3. | END;
4. | END;
5. |
6. | /

**Capitolo 5**

# Manuale d’Uso

## 5.1 Popolamento con Dati di Esempio

01 | ------------------------------------------------------02 | -- SCRIPT PER IL POPOLAMENTO DELLA BASE DI DATI

1. | -------------------------------------------------------
2. | -- Universit degli Studi di Napoli Federico II
3. | -- Insegnamento di Basi di Dati e Sistemi Informativi
4. | --
5. | ------------------------------------------------------08 | -- Questo script popola il database con dati fittizi. 09 | -- Si consiglia di utilizzarlo su un database vuoto,
6. | -- per evitare eventuali problemi dovuti a conflitti.
7. | -------------------------------------------------------
8. |
9. | -- 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’,’ alessandror@kmail.com’);

##### 26 | INSERT INTO DEVELOPER\_BASE (DEV\_ID,FIRSTNAME,LASTNAME,EMAIL,EMAIL2)

1. | VALUES (7,’Anna Chiara’,’de Martino’,’acdemartino@azienda.it’,’ annachiara@fmail.com’);
2. | 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;

1. |
2. |
3. |
4. |
5. | -- PROGETTI
6. |

###### 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’,’Funzionalita di Hello World di livello enterprise.’,’hello’,to\_Date(’10-LUG-19’,’dd-mm-yy’),NULL,’5’);

##### 45 | COMMIT;

1. |
2. |
3. | -- Assegnazioni sviluppatori - progetti
4. | 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;

1. |
2. | -- RELEASE
3. | -- Release del progetto "Zoo"
4. | INSERT INTO RELEASE (RELEASE\_ID,DIRECTORY,CODNAME,STARTDATE,COMPLETE,

##### RELEASEDATE,SWPROJECT,TEST\_PASS)

1. | VALUES (’1’,’alpha’,’Alpha’,to\_date(’14-LUG-19’,’dd-mm-yy’),’F’,NULL,’1 ’,’F’);
2. | -- Release del progetto "HelloWorld"
3. | 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;

1. |
2. | -- PACKAGE
3. | -- 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)

1. | VALUES (’2’,’it.zoo.strutture’,’ISNEW’,’Contiene tutte le strutture dello zoo.’,’1’);
2. | -- Package del progetto "Hello World", release "1.0 pro"
3. | INSERT INTO PACKAGE (PACKAGE\_ID,NAME,MONITOR,DESCRIPTION,RELEASE)
4. | VALUES (’3’,’it.azienda.hello’,’ISNEW’,’Pacchetto principale.’,’2’);

###### 80 | COMMIT;

1. |
2. |
3. |
4. |
5. |
6. |
7. | -- CLASSI
8. | -- PROGETTO "Zoo"
9. | -- -- 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)

1. | VALUES (’3’,’Gnu’,’PUBLIC’,’Gnu’,’/zoo/alpha/src/it/zoo/animali/Gnu. java’,’0’,’100’,’ISNEW’,’F’,’Classe Gnu.’,’1’,’1’);
2. | INSERT INTO CLASS\_BASE (CLASS\_ID,NAME,SCOPE,FILENAME,PATH,STARTLINE,

##### ENDLINE,MONITOR,CHANGEDMETHOD,DESCRIPTION,PACKAGE\_ID)

1. | VALUES (’4’,’Animale’,’PUBLIC’,’Animale’,’/zoo/alpha/src/it/zoo/animali

/Animale.java’,’0’,’100’,’ISNEW’,’F’,’Classe generica di tutti gli Animali.’,’1’);

1. | 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)

1. | VALUES (’6’,’Cobra’,’PUBLIC’,’Cobra’,’/zoo/alpha/src/it/zoo/animali/ Cobra.java’,’0’,’90’,’ISNEW’,’F’,’Serpente velenoso.’,’1’,’5’);
2. | INSERT INTO CLASS\_BASE (CLASS\_ID,NAME,SCOPE,FILENAME,PATH,STARTLINE,

##### ENDLINE,MONITOR,CHANGEDMETHOD,DESCRIPTION,PACKAGE\_ID,SUPERCLASS)

1. | VALUES (’7’,’Varano’,’PUBLIC’,’Varano’,’/zoo/alpha/src/it/zoo/animali/

Varano.java’,’0’,’50’,’ISNEW’,’F’,’Lucertola molto cresciuta.’,’1’,’ 5’);

1. | INSERT INTO CLASS\_BASE (CLASS\_ID,NAME,SCOPE,FILENAME,PATH,STARTLINE,

##### ENDLINE,MONITOR,CHANGEDMETHOD,DESCRIPTION,PACKAGE\_ID,SUPERCLASS)

1. | VALUES (’13’,’Pesce’,’PUBLIC’,’Pesce’,’/zoo/alpha/src/it/zoo/animali/

Pesce.java’,’0’,’100’,’ISNEW’,’F’,’Classe Generica Pesci.’,’1’,’4’);

1. |
2. | -- -- Classi del pacchetto it.zoo.strutture
3. | INSERT INTO CLASS\_BASE (CLASS\_ID,NAME,SCOPE,FILENAME,PATH,STARTLINE,

##### ENDLINE,MONITOR,CHANGEDMETHOD,DESCRIPTION,PACKAGE\_ID,SUPERCLASS)

1. | 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’);
2. | INSERT INTO CLASS\_BASE (CLASS\_ID,NAME,SCOPE,FILENAME,PATH,STARTLINE,

##### ENDLINE,MONITOR,CHANGEDMETHOD,DESCRIPTION,PACKAGE\_ID,SUPERCLASS)

1. | VALUES (’9’,’Retillario’,’PUBLIC’,’Rettilario’,’/zoo/alpha/src/it/zoo/ strutture/Rettilario.java’,’0’,’70’,’ISNEW’,’F’,’Contenitore in vetro adatto a contenere rettili.’,’2’,’11’);
2. | 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)   1. | VALUES (’11’,’Habitat’,’PUBLIC’,’Habitat’,’/zoo/alpha/src/it/zoo/ strutture/Habitat.java’,’0’,’40’,’ISNEW’,’F’,’Generico Habitat. Classe astratta.’,’2’); 2. | 3. | -- PROGETTO "Hello World" 4. | -- -- Classi del pacchetto it.azienda.hello 5. | INSERT INTO CLASS\_BASE (CLASS\_ID,NAME,SCOPE,FILENAME,PATH,STARTLINE, ENDLINE,MONITOR,CHANGEDMETHOD,DESCRIPTION,PACKAGE\_ID) 6. | VALUES (’12’,’Hello’,’PUBLIC’,’Hello’,’/hello/hello1/src/it/azienda/ | |
| hello/Hello.java’,’0’,’50’,’ISNEW’,’F’,’Fornisce funzionalita saluto.’,’3’);   1. | COMMIT; 2. | 3. | 4. | 5. | -- ATTRIBUTI 6. | -- Attributi di it.zoo.animali.Animale 7. | INSERT INTO ATTRIBUTE (ATTR\_ID,NAME,POSITION,TYPE,SCOPE,CLASS) 128 | VALUES (’1’,’Nome’,’1’,’String’,’PRIVATE’,’4’); 8. | INSERT INTO ATTRIBUTE (ATTR\_ID,NAME,POSITION,TYPE,SCOPE,CLASS) 9. | VALUES (’2’,’Sesso’,’2’,’char’,’PRIVATE’,’4’); 10. | --Attributi di it.zoo.strutture.Habitat 11. | INSERT INTO ATTRIBUTE (ATTR\_ID,NAME,POSITION,TYPE,SCOPE,CLASS) 12. | VALUES (’3’,’Nome’,’1’,’String’,’PRIVATE’,’11’); 13. | --Attributi di it.zoo.strutture.Acquario 14. | INSERT INTO ATTRIBUTE (ATTR\_ID,NAME,POSITION,TYPE,SCOPE,CLASS) 15. | VALUES (’4’,’Volume’,’1’,’double’,’PRIVATE’,’10’); 16. | --Attributi di it.zoo.strutture.Rettilario 17. | INSERT INTO ATTRIBUTE (ATTR\_ID,NAME,POSITION,TYPE,SCOPE,CLASS) 18. | VALUES (’5’,’Temperatura’,’1’,’double’,’PRIVATE’,’9’); 19. | COMMIT; 20. | 21. | -- METODI 22. | -- metodo aggiungi(Animale anim) di habitat | di |
| 1. | INSERT INTO METHOD\_BASE (METHOD\_ID,SCOPE,NAME,TYPE,STARTLINE,ENDLINE, MONITOR,DESCRIPTION,CLASS) 2. | VALUES (’1’,’PUBLIC’,’aggiungi’,’void’,’20’,’10’,’ISNEW’,’Aggiunge un animale all’’habitat.’,’11’); 3. | 4. | -- metodo saluta(String nome, String lingua) di hello 5. | INSERT INTO METHOD\_BASE (METHOD\_ID,SCOPE,NAME,TYPE,STARTLINE,ENDLINE, MONITOR,DESCRIPTION,CLASS) 6. | VALUES (’2’,’PUBLIC’,’saluta’,’String’,’10’,’10’,’ISNEW’,’Ritorna una stringa di saluto.’,’12’); 7. | COMMIT; 8. | 9. | 10. | -- PARAMETRI 11. | -- metodo aggiungi() 12. | INSERT INTO PARAMETER (PARAM\_ID,NAME,POSITION,TYPE,METHOD) VALUES (’1’,   ’anim’,’1’,’Animale’,’1’);   1. | --metodo saluta() 2. | INSERT INTO PARAMETER (PARAM\_ID,NAME,POSITION,TYPE,METHOD) VALUES (’2’, ’persona’,’1’,’String’,’2’); 3. | INSERT INTO PARAMETER (PARAM\_ID,NAME,POSITION,TYPE,METHOD) VALUES (’3’,   ’lingua’,’2’,’String’,’2’);   1. | COMMIT; | |

1. |
2. | --TEST
3. | --aggiungi Test di mammiferi con coda
4. | INSERT INTO TEST (TEST\_ID,DIRECTORY,DESCRIPTION,FILENAME,SWPROJECT)
5. | VALUES (’1’,’test’,’I mammiferi hanno la coda?’,’Mammifero’,’1’);

###### 165 | COMMIT;

1. |
2. | --TEST ESEGUITI
3. | --aggiungi Test ESEGUITO di mammiferi con coda
4. | INSERT INTO EXECUTED\_TEST (EXECUTION\_ID,RESULT,DATATEST,USEDSTRUCTURE, TEST)
5. | VALUES (’1’,’T’,SYSDATE,’CLASSE:Mammifero, SOTTOCLASSI:Leone,Gnu’,’1’);

###### 171 | COMMIT;

1. |
2. | --ASSOCIAZIONE MOLTI A MOLTI RELEASE<->TEST ESEGUITI
3. | INSERT INTO RELEASE\_TEST(EXECUTEDTEST, RELEASE)

###### 175 | VALUES (’1’,’1’); 176 | COMMIT;

1. |
2. | --COMMIT
3. | --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)

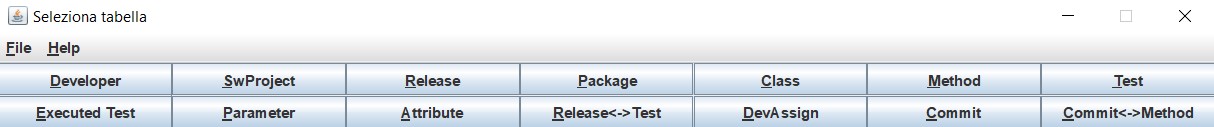
1. | VALUES (’2’,SYSDATE,’Commit classe Hello’,’12’,’1’);
2. |
3. | INSERT INTO COMMIT\_METH(COMMIT\_ID,METHOD) 186 | VALUES (’1’,’1’); 187 | INSERT INTO COMMIT\_METH(COMMIT\_ID,METHOD)

188 | VALUES (’2’,’2’); 189 | COMMIT;

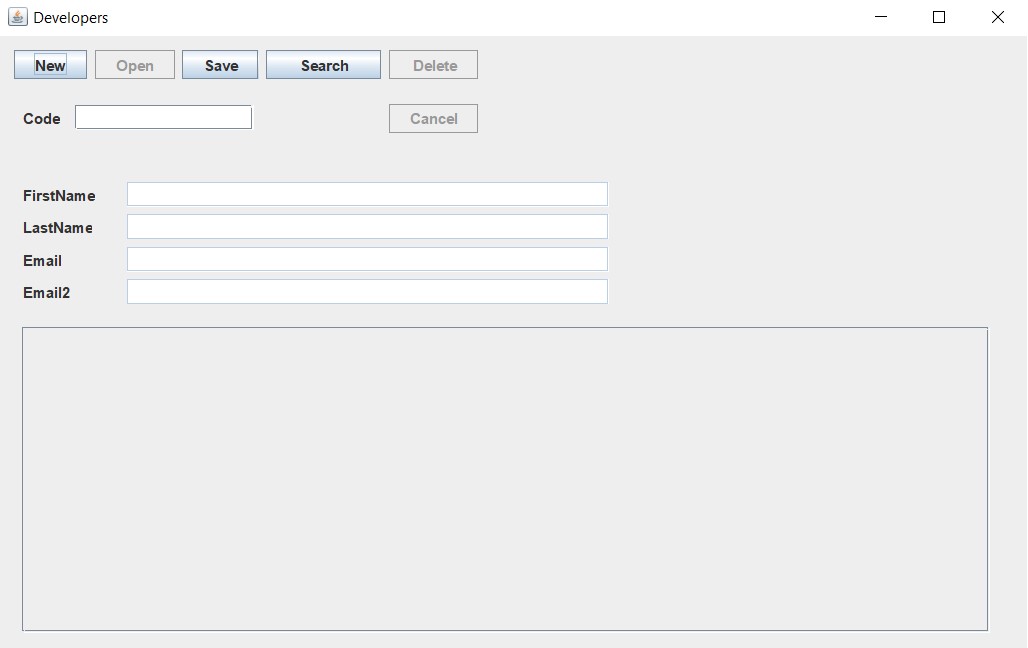
## 5.2 Esempio d’Uso

### 5.2.1 Inserimento Record

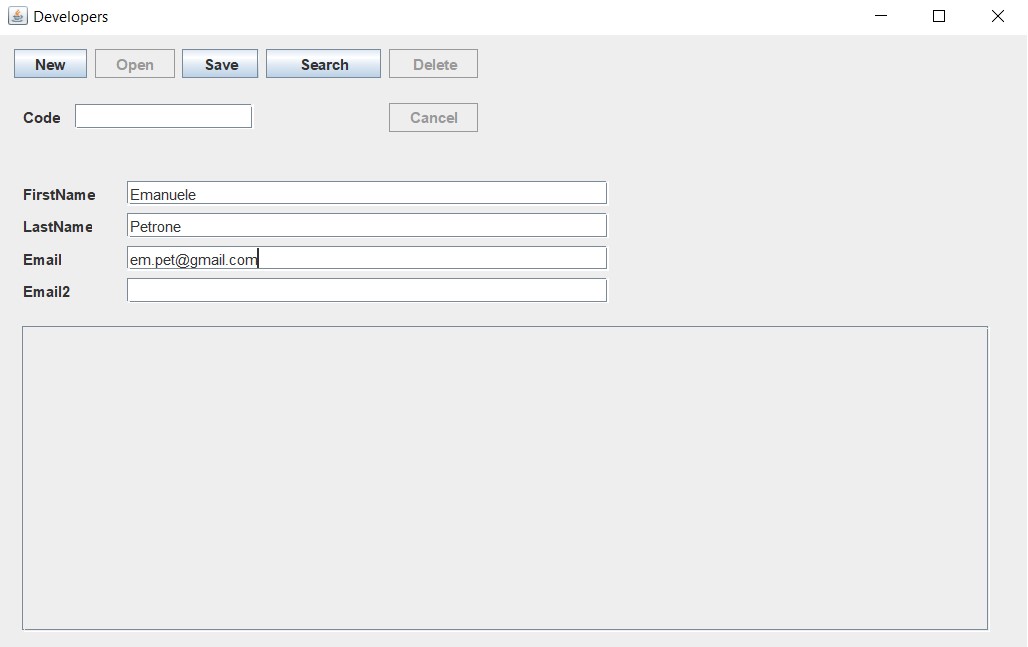
Dopo l’autenticazione col DataBase apparira` la finestra di **selezione delle tabelle**.



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



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



Infine Cliccare su **Save**.

