Tecnologie di Sviluppo per il Web

Programmazione su Basi di Dati: Aspetti Metodologici Parte b

versione 3.2

G. Mecca – Università della Basilicata – mecca@unibas.it

Sommario

- Modello e DTO
  - ⇨Riferimenti e Chiavi Esterne
  - ⇨Navigazione e Join
  - ⇨Creazione degli Identificatori
- Controllo
  - ⇨Gestione dei Vincoli di Integrità
  - ⇨Autenticazione e Autorizzazione
- Un Altro Esempio

Modello e DTO

- I componenti del modello
  - ⇨duplice funzione: contengono i metodi della logica applicativa e rappresentano i dati della base di dati
  - ⇨sono utilizzati come Data Transfer Object
  - ⇨non conoscono il controllo
  - ⇨non conoscono la persistenza

Modello e DTO

- Si tratta tipicamente di JavaBeans
- JavaBean
  - ⇨classe Java con caratteristiche particolari
  - ⇨costruttore no-arg
  - ⇨proprietà tutte private
  - ⇨metodi get e/o set che rispettano le convenzioni di stile per l'accesso e la modifica delle proprietà
  - ⇨eventuali altri metodi di logica applicativa

Modello e DTO

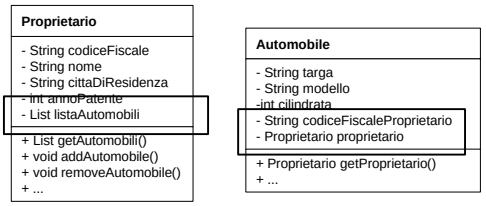
- L'aspetto più delicato
  - ⇨è il disaccoppiamento di impedenza
- Due aspetti principali
  - ⇨differenza nel modello di dati: chiavi esterne contro riferimenti
  - ⇨differenza nella filosofia delle operazioni: join contro navigazioni
  - ⇨questi aspetti rendono delicata la programmazione della persistenza

Riferimenti e Chiavi Esterne

- Nella base di dati
  - ⇨riferimenti basati sui valori
  - ⇨nella ennupla di ogni automobile è riportato il codice fiscale del proprietario
- Esempio
  - ⇨in "proprietari": ('pncpll55...', 'Pinco Pallino', 'Potenza', 1970)
  - ⇨in "automobili": ('ab123de', 'Fiat Punto', 1200, 'pncpll55...')

Riferimenti e Chiavi Esterne

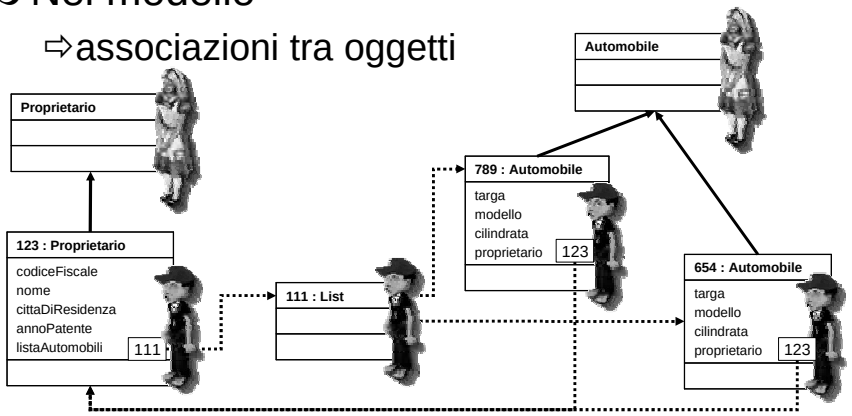
- Nell'applicazione
  - ⇨classe Proprietario e classe Automobili
  - ⇨Proprietario: coll. di riferimenti alle automobili
  - ⇨Automobile: riferimento al suo proprietario



NOTA: manteniamo sia la chiave esterna che il riferimento all' oggetto Proprietario (>>)

Riferimenti e Chiavi Esterne

- Nel modello
  - ⇨associazioni tra oggetti



Riferimenti e Chiavi Esterne

- Di conseguenza
  - ⇨gli oggetti dell'applicazione rappresentano in qualche modo alle ennuple delle tabelle
  - ⇨ma con significative differenze nella struttura
- In particolare
  - ⇨ricostruire un oggetto a partire dalla base di dati vuol dire ricostruire correttamente i riferimenti

Riferimenti e Chiavi Esterne

- Ricerca di un proprietario
  - ⇨devo produrre un oggetto "completo"
  - ⇨codice fiscale, nome, città, anno patente
  - ⇨la collezione dei riferimenti alle sue automobili
- Ricerca di un'automobile
  - ⇨targa, modello, cilindrata
  - ⇨il riferimento al suo proprietario
- Quindi, in entrambi i casi
  - ⇨è necessario prelevare dati da entrambe le tabelle
  - ⇨è necessario costruire vari oggetti

Caricamento Pigro

- Una prima possibilità
  - ⇨“caricamento immediato dei dati”
- Idea
  - ⇨ogni volta che carico i dati di un proprietario carico anche i dati (e creo gli oggetti) della sua lista di automobili
  - ⇨ogni volta che carico i dati di un’automobile dalla base di dati carico e creo anche il proprietario

Caricamento Pigro

- Vantaggi di questo approccio
  - ⇨produco sempre una “rete” di oggetti con riferimenti completamente specificati
  - ⇨dopo il caricamento iniziale posso lavorare con gli oggetti senza preoccuparmi ulteriormente di accedere alla base di dati
  - ⇨es: cerco le automobili di un proprietario nella List e non più nella base di dati

Caricamento Pigro

- Primo svantaggio di questo approccio
  - ⇨sto replicando in memoria centrale i dati della base di dati
  - ⇨rischio di produrre disallineamenti se la base di dati viene aggiornata
  - ⇨questo è possibile perchè le applicazioni client-server sono fatte per lavorare in modo distribuito e concorrente sulla base di dati

Caricamento Pigro

- Un esempio
  - ⇨il terminale n. 1 carica i dati di un proprietario e ne visualizza le automobili
  - ⇨il terminale n. 2 aggiunge una nuova automobile per il proprietario
  - ⇨da quel momento i dati visualizzati sul terminale n. 1 non riflettono più lo stato aggiornato della base di dati

Caricamento Pigro

- Secondo svantaggio di questo approccio
  - ⇨nelle applicazioni con logica applicativa complessa il caricamento di un oggetto può scatenare il caricamento di molti altri oggetti
  - ⇨a causa di associazioni complesse tra gli oggetti
- Esempio: prove d'esame
  - ⇨studente >> prenotazioni >> prove >> insegnamenti >> docenti ecc.

Caricamento Pigro

- In sostanza
  - ⇨sto producendo in memoria centrale una "cache" del contenuto della base di dati sotto forma di rete di oggetti collegati
  - ⇨devo gestire i tipici problemi di caching: ampiezza della cache (per evitare eccessivo scopo di memoria), e allineamento della cache con il deposito di dati primario
  - ⇨si tratta di problemi complessi da gestire, che preferiremmo evitare

Caricamento Pigro

- Di conseguenza
  - ⇨seguiamo un approccio diverso
- Caricamento pigro
  - ⇨alla creazione, di ciascun oggetto vengono prelevati dalla base di dati solo gli attributi della ennupla corrispondente
  - ⇨successivamente, a richiesta, vengono caricati quando è necessario, i riferimenti

Caricamento Pigro

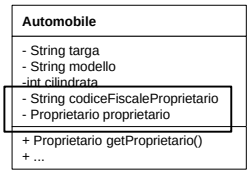
>> DAOProprietario.java
- doSelectCodiceFiscale
- caricaAutomobili

- Struttura del DAO
  - ⇨metodi CRUD per il bean corrispondente
  - ⇨metodi di caricamento pigro per le proprietà
- Esempio: DAOProprietario
  - ⇨public static void caricaAutomobili(Proprietario p)
  - ⇨accede alla base di dati a richiesta e riempie la lista di riferimenti alle automobili che appartengono al proprietario
  - ⇨viene utilizzato dal controllo

Caricamento Pigro

- Nel caso di DAOAutomobile
  - ⇨public static void caricaProprietario(Automobile a)
  - ⇨accede alla base di dati, carica il proprietario dell'automobile e inizializza il riferimento
- Nota
  - ⇨per farlo efficientemente, è opportuno conoscere il codice fiscale del proprietario
  - ⇨in modo da effettuare una selezione sulla tabella proprietari invece che un join

Caricamento Pigro



- Per questa ragione
  - ⇨nella classe Automobile bisogna "duplicare" il riferimento al Proprietario
  - ⇨private Proprietario proprietario; >> riferimento che genera l'associazione
  - ⇨private String codiceFiscaleProprietario >> chiave esterna che viene caricata dalla tabella e serve a caricare successivamente il Proprietario

Navigazione e Join

- Quindi
  - ⇨tipicamente ricostruire un oggetto richiede di prelevare dati da tabelle diverse
- Due diversi approcci sono possibili
  - ⇨navigazione
  - ⇨join
  - ⇨attenzione alle differenze di prestazioni

Navigazione e Join

- Navigazione
  - ⇨è l'approccio del caricamento pigro
  - ⇨ricostruisco il primo oggetto con una prima interrogazione (es: automobile)
  - ⇨prelevo i valori delle chiavi esterne
  - ⇨effettuo ulteriori interrogazioni per ricostruire gli oggetti collegati (es: proprietario)
- In sintesi
  - ⇨"navigo" tra tabelle, seguendo chiavi esterne

Navigazione e Join

- Esempio
  - acquisisco la targa dell'auto da caricare (*t*)
  - 1. carico l'automobile: select * from automobili where targa = *t*
  - 2. carico il proprietario utilizzando il valore della chiave esterna (p): select * from proprietari where codicefiscale = *p*
  - ho "navigato" da una tabella all'altra usando la chiave esterna

Navigazione e Join

- Join
  - utilizzo un'unica interrogazione, nella quale effettuo uno o più join
  - dal risultato del join ricostruisco tutta la struttura di oggetti necessari
  - select * from automobili join proprietari on proprietario = codicefiscale where targa = t
  - approccio naturale per il DBMS

Navigazione e Join

- Analogamente per il proprietario
  - caricamento pigro
  - sia *cf* il generico codice fiscale del proprietario da cercare
  - 1. carico il proprietario: select * from proprietari where codicefiscale = *cf*
  - 2. carico le automobili: select automobili.* automobili where automobili.proprietario = *cf*

Navigazione e Join

- In alternativa, con il join
  - caricamento immediato
  - select * from proprietari left join automobili on codicefiscale = proprietario where codicefiscale = cf
  - è necessario utilizzare il join esterno perchè i proprietari possono non avere automobili

Navigazione e Join

- Attenzione
  - ⇨possono esserci significative differenze di prestazioni tra un approccio e l'altro
  - ⇨i join sono normalmente molto più veloci
- Di conseguenza
  - ⇨bisogna valutare con attenzione vantaggi e svantaggi delle due soluzioni

Navigazione e Join

- Vantaggi del caricamento pigro
  - ⇨riduce i problemi di allineamento tra la base di dati e gli oggetti dell'applicazione
  - ⇨le istanze dei bean sono rappresentazioni OO delle ennuple della base di dati
  - ⇨non bisogna utilizzarli come "cache" in memoria centrale
  - ⇨perchè la base di dati può essere aggiornata concorrentemente

Navigazione e Join

- Svantaggi del caricamento pigro
  - ⇨il caricamento pigro è basato sulla navigazione tra le tabelle e quindi peggiora le prestazioni
  - ⇨questo fatto deve essere tenuto a mente
- Nel complesso
  - ⇨lavorando su un singolo proprietario o una singola automobile alla volta la differenza è marginale e il peggioramento è accettabile

Navigazione e Join

>> DAOAutomobile.java

- Viceversa
  - ⇨nel caso di operazioni su molti proprietari o molte automobili la differenza nei tempi di elaborazione può essere significativa
  - ⇨di conseguenza bisogna, quando possibile, adottare un approccio basato su join
- Esempio
  - ⇨stampa di tutto l'elenco dei proprietari e delle relative automobili; è opportuno usare il join

Controllo

- Lo strato di controllo
  - ⇨contiene il codice che coordina lo svolgimento dei casi d'uso
  - ⇨struttura abbastanza standard
- In particolare
  - ⇨è necessario organizzare i casi d'uso in modo da garantire l'integrità della base di dati

Gestione di Vincoli di Integrità

- Vincoli di integrità nella base di dati
  - ⇨vincoli di ennupla
  - ⇨vincoli di chiave
  - ⇨vincoli di riferimento
- In generale
  - ⇨prima di effettuare le operazioni nel controllo devo verificare che non violino vincoli
  - ⇨l'applicazione deve prevedere verifiche preventive

Gestione di Vincoli di Integrità

- Esempio
  - ⇨prima di inserire l'automobile devo verificare che la targa sia fatta di 7 caratteri (vincolo di ennupla)
  - ⇨prima di inserire il proprietario devo verificare che non ne esista un altro con lo stesso codice fiscale (vincolo di chiave)
  - ⇨prima di inserire una automobile devo verificare che esista il relativo proprietario (vincolo di riferimento)

Gestione di Vincoli di Integrità

>> controllo

- Gestione dei vincoli di ennupla
  - ⇨ordinarie convalide dei dati
- Gestione dei vincoli di chiave
  - ⇨è necessario effettuare ricerche basate sulla chiave primaria prima di effettuare gli inserimenti
  - ⇨al solito è opportuno usare una variante della programmazione difensiva
  - ⇨nel controllo effettuo le verifiche preventive
  - ⇨nel DAO bisogna essere pronti a catturare l'eventuale eccezione sollevata dal DBMS

Gestione di Vincoli di Integrità

- Gestione dei vincoli di riferimento
  - ⇨in questo caso, il caso d'uso deve essere organizzato in modo da cercare prima il valore del riferimento e poi effettuare l'aggiornamento
- Esempio
  - ⇨per inserire un'automobile, devo prima chiedere all'utente di selezionare un proprietario

Gestione di Vincoli di Integrità

- Struttura tipica del caso d'uso
  - ⇨ricerca del proprietario (con uno dei metodi forniti)
  - ⇨una volta selezionato il proprietario, acquisizione dei dati dell'automobile
  - ⇨verifiche sui dati forniti (vincoli di ennupla e vincoli di chiave)
  - ⇨inserimento della nuova automobile utilizzando il codice fiscale selezionato

Gestione di Vincoli di Integrità

- Ricerca del riferimento (“Lookup”)
  - ⇨operazione di ricerca di una ennupla durante l’inserimento di un’altra per soddisfare un vincolo di integrità referenziale
  - ⇨es: ricerca del proprietario per poter inserire un’automobile

Autenticazione e Autorizzazione

- Un aspetto importante del controllo
  - ⇨deve provvedere le funzionalità di autenticazione e autorizzazione
- Sottocaso d’uso standard
  - ⇨“Utente effettua autenticazione (login)”
  - ⇨l’utente fornisce nome utente e password
  - ⇨il sistema autentica ed autorizza l’utente all’accesso

Autenticazione e Autorizzazione

- La soluzione tipica
  - ⇨una tabella Utenti nella base di dati
  - ⇨con attributi nomeutente, password, ruolo
  - ⇨un DAOUtente con un metodo Utente doSelectNomeUtente(String nomeUtente) che cerca l'utente nella base di dati
  - ⇨un bean Utente con un metodo verifica se la password fornita è corretta

Autenticazione e Autorizzazione

- Attenzione
  - ⇨la password NON deve essere memorizzata in chiaro nella base di dati
- Una buona soluzione
  - ⇨memorizzarne un valore di hash, utilizzando per esempio l'algoritmo MD5
  - ⇨in fase di verifica calcolare l'hash della password fornita e verificare se i due hash sono uguali

Autenticazione e Autorizzazione

- Passo n. 1
  - ⇨all'atto della creazione della base di dati calcolare l'hash md5 delle password degli utenti da inserire nella tabella utenti
  - ⇨per farlo è possibile utilizzare un'utilità come HashCalc oppure la funzione md5 di Postgresql (da psql: select md5('*stringa*'); produce l'hash md5 della stringa riportata tra apici
  - ⇨es: 'pinco' >> '8ae79c7b003ff7c7199b9389fe3c6b5d'

Autenticazione e Autorizzazione

- Passo n. 2
  - ⇨inserire le ennuple nella tabella utenti specificando per le password non il valore in chiaro ma l'hash calcolato precedentemente
  - ⇨es: insert into utenti values (
    'pinco', 'Pinco Palla',
    '8ae79c7b003ff7c7199b9389fe3c6b5d',
    'utente');

Autenticazione e Autorizzazione

- Passo n. 3
  - ⇨nel DAO costruire il metodo doSelectNomeUtente() in modo ordinario
- Passo n. 4
  - ⇨nel bean Utente costruire il metodo boolean verifica(String password) in modo che confronti non le password in chiaro ma i rispettivi hash

Autenticazione e Autorizzazione

- Per farlo
  - ⇨è necessario calcolare l'hash md5 della password fornita dall'utente per l'autenticazione
  - ⇨procedura non banale che richiede di utilizzare la classe java.security.MessageDigest e poi di trasformare l'array di byte prodotto in una stringa codificandolo in esadecimale

```
public class Utente {

    private String nomeUtente;
    private String password;
    private boolean autenticato;

    ...

    public void verifica(String password){
       String hashPasswordFornita = md5hash(password);
       if (this.password.equals(hashPasswordFornita)) {
           this.autenticato = true;
       } else {
           this.autenticato = false;
       }
    }

    // continua
```



```
    // continua
    private String md5hash(String password) {
       String hashString = null;
       try {
          java.security.MessageDigest digest =
                    java.security.MessageDigest.getInstance("MD5");
          byte[] hash = digest.digest(password.getBytes());
          hashString = "";
          for (int i = 0; i < hash.length; i++) {
             hashString += Integer.toHexString(
                                  (hash[i] & 0xFF) | 0x100
                               ).toLowerCase().substring(1,3);
          }
       } catch (java.security.NoSuchAlgorithmException e) {
          System.out.println(e);
       }
       return hashString;
    }
```

Autenticazione e Autorizzazione

- Se l'utente è autenticato
  - ⇨è possibile salvare il riferimento nel Modello (es: modello.putBean("utente", utente));
  - ⇨e abilitare le varie azioni
- Successivamente
  - ⇨le azioni possono verificare che ci sia un utente autenticato nel modello prima di procedere con l'esecuzione

Autenticazione e Autorizzazione

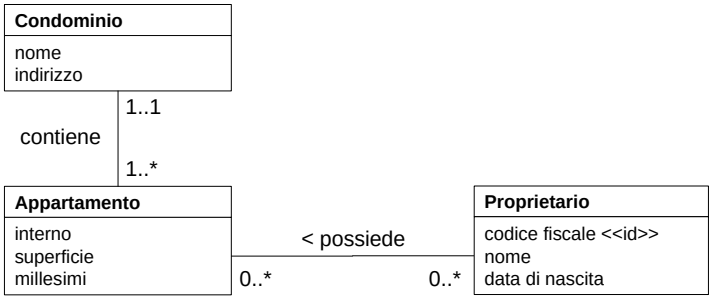
- Attenzione al processo di autorizzazione
  - ⇨possono esserci utenti appartenenti a ruoli diversi, con livelli di autorizzazione diversi
  - ⇨es: utente semplice, amministratore, ecc.
  - ⇨in generale nelle azioni bisognerebbe verificare che l'utente abbia il livello di privilegi adeguato ad eseguire l'azione richiesta

Un Altro Esempio

- Un ulteriore esempio
  - ⇨un'applicazione client-server per la gestione di condomini
  - ⇨con dati su condomini, appartamenti, proprietari
  - ⇨interessante perchè mostra un esempio di associazione molti a molti

Un Altro Esempio

- Lo schema concettuale



Un Altro Esempio

- Lo schema logico della base di dati

```
create table condomini (
  id serial primary key,
  nome varchar(20),
  indirizzo varchar(50)
)
```

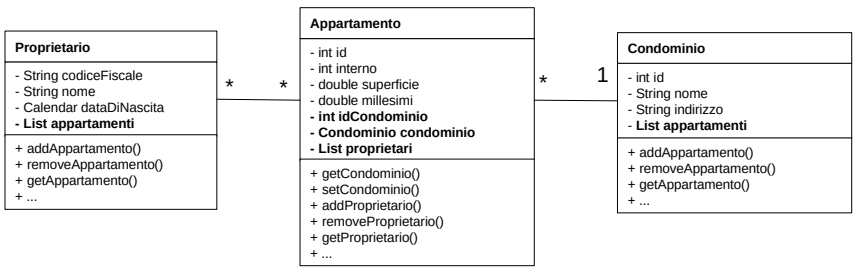
```
create table proprietari (
   codfiscale char(16) primary key,
   nome varchar(20) not null,
   datadinascita date
)
```

```
create table appartamenti (
  id serial primary key,
  interno integer not null,
  superficie real,
  millesimi real,
  condominio integer
    references condomini(id)
)
```

```
create table proprietariappartam (
   id serial primary key,
   proprietario char(16)
      references
      proprietari(codfiscale),
   appartamento integer
      references
      appartamenti(id)
)
```


Un Altro Esempio

- Il diagramma delle classi



Un Altro Esempio

- I DAO
  - ⇨in questo caso è necessario creare un DAO per ciascun bean
  - ⇨DAOCondominio
  - ⇨DAOAppartamento
  - ⇨DAOProprietario
  - ⇨viceversa, non esistono DAO per la tabella "proprietariappartamenti"

Un Altro Esempio

- DAOCondominio
  - ⇨doSelectId()
  - ⇨doSelectNome()
  - ⇨doInsert(): inserisce solo nome e indirizzo
  - ⇨doUpdate(): aggiorna nome e indirizzo (non aggiorna la chiave primaria id)
  - ⇨doDelete()
  - ⇨caricaAppartamenti()

Un Altro Esempio

- DAOAppartamento
  - doSelectId()
  - doInsert(): inserisce interno, sup., millesimi; richiede il lookup del Condominio
  - doUpdate(): aggiorna interno, sup., millesimi e id del condominio (chiave esterna)
  - doDelete()
  - caricaCondominio()
  - caricaProprietari()

Un Altro Esempio

- DAOProprietario
  - doSelectCodiceFiscale(), doSelectNome()
  - doInsert(), doUpdate(), doDelete()
  - caricaAppartamenti()
- Ma...
  - mancano i metodi CRUD relativi alla tabella proprietariautomobili

Un Altro Esempio

- Due possibili soluzioni
  - attribuire i metodi ad uno dei DAO
  - sviluppare un componente apposito
- I soluzione: in DAOProprietario
  - doInsertProprietarioAppartamento(): riceve un riferimento ad un Proprietario ed ad un Appartamento e aggiunge una ennupla alla tabella che traduce l'associazione molti a molti
  - doDeleteProprietarioAppartamento()

Un Altro Esempio

- II soluzione: DAOPropAppartamento
  - doInsertProprietarioAppartamento(): riceve un riferimento ad un Proprietario ed ad un Appartamento e aggiunge una ennupla alla tabella che traduce l'associazione molti a molti
  - doDeleteProprietarioAppartamento()

Riassumendo

- Modello e DTO
  - ⇨Riferimenti e Chiavi Esterne
  - ⇨Navigazione e Join
  - ⇨Creazione degli Identificatori
- Controllo
  - ⇨Gestione dei Vincoli di Integrità
  - ⇨Autenticazione e Autorizzazione
- Un Altro Esempio

Termini della Licenza