**CAPITOLO 7: PERSISTENZA**

*Questo capitolo è focalizzato sul data modeling. A tal proposito si intente illustrare il processo utilizzato per creare un modello concettuale del sistema di storage attraverso l’identificazione e la definizione delle entità, nonché delle loro relazioni, utili affinchè il sistema svolga i suoi compiti correttamente.*

**7.1 Problematica**

Il sistema deve garantire la persistenza dei dati ma allo stesso tempo le particolari modalità per l’accesso alla base dei dati devono essere trasparenti al resto del sistema. Ottemperando a questo obiettivo sarà quindi possibile ridurre al minimo l’accoppiamento presente tra codice applicativo e livello di persistenza.

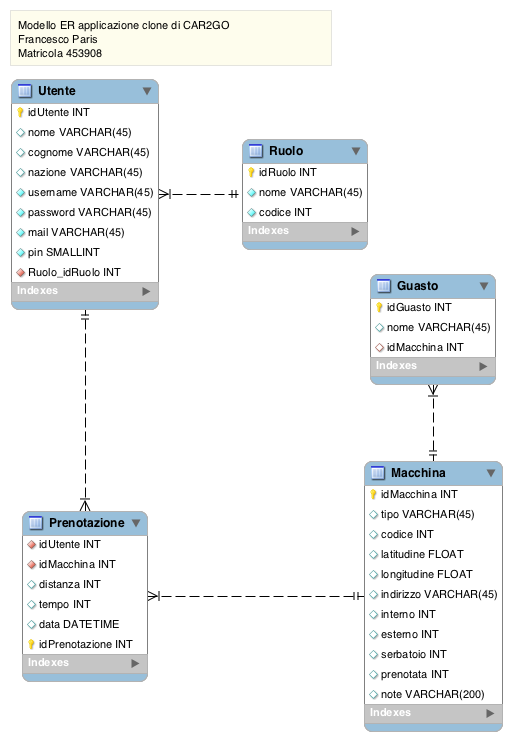
Per soddisfare tali requisiti il sistema, oltre ad una attenta progettazione, utilizza il framework Hibernate.

**7.2 Il modello**

A seguito dell’analisi dei requisiti dell’applicazione clone di Car2Go, tenendo presenti le linee guida del pattern Model Driven Design, sono state individuate le seguenti entità:

* **Ruolo**: rappresenta i possibili ruoli che un utente può ricoprire. Esistono due tipi di ruoli: Admin e User. Di un ruolo è di interesse sapere il nome ed il codice.
* **Utente**: rappresenta un utente registrato al sistema. E’ di interesse conoscere il nome, cognome, nazione, username, password, mail, pin ed il Ruolo a cui appartiene.
* **Macchina**: è una vettura che il sistema può gestire. Di una macchina è necessario tenere traccia del tipo, codice, coordinate geografiche quali latitudine, longitudine, indirizzo, situazione dello stato interno ed esterno, Infine è necessario sapere se una vettura risulti essere prenotata.
* **Prenotazione**: rappresenta l’entità che deve gestire il concetto di noleggio e prenotazione. A tal proposito è necessario che tale entità gestisca informazioni inerenti all’utente che intende effettuare una prenotazione e qual è la vettura che vorrebbe usare. Per gestire i costi di un noleggio è necessario memorizzare la distanza ed il tempo di durata del viaggio. Il sistema tiene anche traccia della data in cui il noleggio è stato effettuato.
* **Guasto**: tale entità rappresenta un possibile malfunzionamento che colpisce una vettura. Di un guasto è di interesse conoscere il nome e la vettura su cui il malfunzionamento ricade.

La figura seguente mostra il diagramma Entità Relazioni dell’applicazione clone di Car2Go realizzata nell’ambito del lavoro di tesi. Le classi che rappresentano il modello sono state codificate nel package car2go-model.



**7.3 Strutturazione dei Data Access Object**

Per poter gestire differenti tipi di accesso al database, per ogni entità, è stata creata una interfaccia che ha la funzione di gestire l’accesso ai repository. Tali interfacce sono specificate nel progetto car2go-persistance-common, all’interno del package it.car2go.persistance.common. In tali interfacce ci sono i metodi per realizzare operazioni CRUD (Create, Read, Update e Delete). Le concretizzazioni di tali interfacce sono realizzate in dei componenti DAO che implementano quindi tali interfacce. Un DAO è infatti un oggetto che permette di stratificare ed isolare l’accesso ad una tabella del database mediante query creando quindi un maggior livello di astrazione. Seguendo le linee guida dell’Architettura Esagonale, le interfacce servono per poter mettere a fattor comune le operazioni che i vari DAO dovranno implementare. In questo modo se bisogna fruire del database con una tecnologia diversa sarà sufficiente realizzare dei nuovi DAO, che implementano le interfacce presenti nel package it.car2go.persistance.common, ma questo non avrà ripercussioni sul resto del sistema grazie all’astrazione offerta dalle interfacce. Nell’applicazione che è stata usata nel lavoro di tesi è stato usato il framework Hibernate. Quando sono stati usati gli Enterprise Java Beans (EJB), per mettere in maggior evidenza il ciclo di vita di tali componenti, è stato usato JDBC con query in Sql.

**7.4 Hibernate**

Hibernate è una piattaforma middleware open source per lo sviluppo di applicazioni Java, attraverso l'appoggio al relativo framework, che fornisce un servizio di Object Relational Mapping (ORM). L’ORM è il sistema di persistenza automatico e trasparente di oggetti in tabelle di un database relazionale. Ogni oggetto viene persistito nel database tramite l'inserimento di nuovi record i cui campi contengono i valori degli attributi dell'oggetto. Di solito ad ogni oggetto corrisponde un record di una particolare tabella associata alla classe dell'oggetto. L'associazione tra la classe e la tabella viene ottenuta tramite l'utilizzo di file di descrizione (file di mapping), in cui si specificano le modalità di mapping tra gli attributi dell'oggetto e i campi della tabella. Ogni interrogazione viene effettuata utilizzando un linguaggio simil-SQL che permette di scrivere query utilizzando il nome delle classi e degli attributi. Le interrogazioni vengono convertite dal tool ORM in istruzioni SQL da eseguire sul database relazionale sottostante. I resultset delle interrogazioni vengono convertiti nei corrispondenti oggetti in maniera del tutto trasparente. Dal punto di vista dello sviluppatore uno strumento ORM permette di effettuare delle richieste di istanze di particolari classi, con filtri basati sulle proprietà delle classi, i cui risultati sono liste di oggetti. Tutti i dettagli sottostanti al livello object-oriented sono praticamente nascosti.

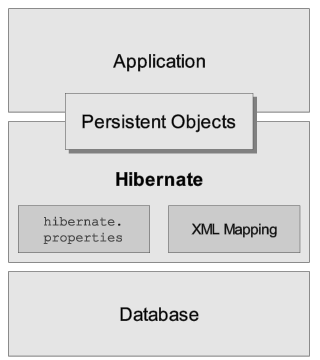
In definitiva una soluzione ORM consiste in:

* API per eseguire le operazioni di base (CRUD) sugli oggetti delle classi persistenti
* Un linguaggio per la costruzione di query sulle classi e le proprietà delle classi
* Un sistema per specificare il mapping tramite metadata
* Un sistema interno per l'interazione con oggetti transazionali, per il dirty checking, il fetching delle associazioni lazy.

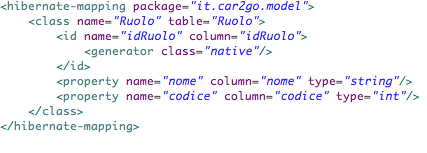
I benefici che si hanno nell'utilizzo di una soluzione ORM sono svariati:

* **Produttività**: il codice che si occupa della persistenza dei dati è la parte forse più tediosa di un'applicazione. Un tool ORM elimina molto di questo codice, semplificandolo e automatizzandolo al massimo
* **Manutenibilità**: dovendo scrivere meno codice è più facile manutenere l'applicazione. Inoltre in soluzioni fatte “a mano”, che trasformano il modello ad oggetti in query relazionali e record di database in oggetti, è molto più difficile far variare il modello di dominio insieme al modello relazionale. Con un tool ORM si mantengono separati i due modelli tramite l'uso di uno strato intermedio di mapping, che spesso minimizza anche la propagazione di variazione tra i due modelli.
* **Performance**: anche se a prima vista sembra che lo strato aggiuntivo di mapping introduca dei perggioramenti alla performace, ci sono talmente tante opzioni di ottimizzazioni (utilizzo di cache di primo e secondo livello, utilizzo di batch query ecc) che in realtà, in applicazioni con un numero elevato di accessi in lettura, è possibile ottenere delle performance estremamente più efficienti di una semplice soluzione basata su SQL.
* **Indipendenza dal tipo di database**: questa è una delle caratteristiche più interessanti di uno strumento di ORM, il quale astrae l'applicazione dal sottostante database SQL e dal suo dialetto. ORM introduce un linguaggio proprietario per le interrogazioni e le operazioni CRUD. Questo linguaggio viene poi convertito automaticamente nell'SQL del database sottostante. Cambiano un parametro nella configurazione dell'ORM, è possibile portare l'applicazione su altri database senza praticamente toccare una riga di codice. Inoltre l'ORM conosce molto bene (spesso molto meglio del programmatore) il dialetto del particolare database, e ottimizza le query utilizzando quanto più possibile funzioni SQL proprietarie.

Hibernate cerca di essere una soluzione completa al problema della gestione di dati persistenti in Java. Si pone nel mezzo tra l'applicazione e un database relazionale, lasciando lo sviluppatore libero di concentrarsi sugli aspetti di business dell'applicazione. Hibernate non è una soluzione intrusiva e si adatta bene ad applicazioni nuove ed esistenti, e non richiede modifiche distruttive al resto dell'applicazione.



Per rendere persistente una classe è necessario specificare un mapping tra la classe ed il database. Nella tesi, per ottemperare a tale scopo, sono stati usati dei file XML per definire questa associazione. Ogni classe ha il suo file di mapping associato. Di solito il file XML ha lo stesso nome della classe con estensione “.hbm.xml”. Il file di mapping per Ruolo ha quindi nome Ruolo.hbm.xml. Si seguito il file di mapping per l’entità Ruolo.



Come è possibile notare, il mapping viene definito tramite:

* Un elemento <class> in cui è specificato il nome della classe Java e la tabella corrispondente al database;
* All’interno di <class> si trova la definizione dell’id dell’oggetto tramite l’elemento <id>, in cui si definisce quale sia la politica di generazione di tale id. Nell’esempio viene definito un id con una politica che delega al database sottostante il compito di creare l’id (ad esempio un campo auto-increment) tramite l’elemento <generator class=”native”/>
* Seguono poi le definizioni delle proprietà tramite l’elemento <property>, ognuna delle quali viene mappata con una colonna della tabella.

Per poter utilizzare Hibernate deve essere costruita innanzitutto una **SessionFactory**. La SessionFactory è la factory che permette la creazione di una Session di Hibernate, la quale può essere vista in maniera semplicistica come la cache delle istanze di oggetti Java persistenti e permette l’interazione con il database. La SessionFactory deve essere creata una sola volta, è molto dispendiosa a livello di risorse e contiene tutti gli aspetti di configurazione per l’accesso ad un database. La SessionFactory viene configurata attraverso un file XML di nome hibernate.cfg.xml.



Il file specifica:

* Il driver JDBC di connessione al database;
* Il tipo di dialetto per il particolare database da usare; cambiando dialetto e driver è possibile usare un qualsiasi altro database senza modificare assolutamente l’applicazione;
* I parametri di connessione al database
* Il gestore per le transazioni
* I file di mapping (hbm.xml) da usare per definire le classi persistenti