Politecnico di Torino

Linguaggi e Traduttori

Specificazione della Tesina

SQLDoclet - Traduttore *XDoclet based*

Indice

1 Scopo

Sviluppo di un traduttore che realizzerà il *mapping* della struttura di tabelle relativa a un insieme di classi Java, generando in uscita il codice SQL per la creazione di queste tabelle. Tale *mapping* sarà effettuato per mezzo di alcuni *tag* speciali (@sql.xxxx), collocati in commenti di tipo Javadoc nel codice delle classi Java essendo analizzati.

Questo progetto è stato basato nel strumento XDoclet (http://xdoclet.sourceforge.net), più specificamente nel suo componente **hibernatedoclet**, che fornisce un meccanismo di creazione del mapping tra classi Java e il **hibernate**, attraverso generazione di documenti XML.

2 Funzionamento

2.1 Descrizione

Il traduttori agisce come un ausiliare nel sviluppo di software in Java, poiché è in grado di generare automaticamente il codice SQL per la creazione delle tabelle relativi alle classi persistenti della applicazione.

Il funzionamento è semplice: mentre si scrive i commentari *Javadoc* nelle classi (che dovrebbe essere fatto sempre!), allora si deve mettere anche i *tag* SQLDoclet. Codesti *tag* saranno utilizzati per il traduttori come sorgente di meta-dati, che saranno la base per la generazione del SQL di uscita.

Un esempio:

```
/**
*  @sql.class table="TEST"
*/
public class Test {

        private String foo;
        private Long bar;

        /**
         *  @sql.id generator-class="assigned" column="FOO"
         */
        public String getFoo() {
                return foo;
        }

        /**
         * @sql.property column="BAR"
         */
        public Long getBar() {
                return bar;
        }
}
```

Avrà in uscita il seguente codice SQL:

```
create table TEST (
   FOO varchar(255) not null,
   BAR bigint,
   primary key (FOO)
);
```

2.2 *Tag* Riconosciuti

I *tag* (e suoi parametri) che il traduttore riconoscerà sono:

@sql.class – Crea una tabella referente alla classe che contiene il *tag*.

Parametro	Tipo	Descrizione	Obbligatorio
table	testo	Il nome della tabella	Si

@sql.property – Crea una colonna nella tabella referenti alla classe che contiene il *tag.*

Parametro	Tipo	Descrizione	Obbligatorio
column	testo	Il nome della colonna nella base di dati.	Si
not-null	bool	Se a colonna è not-null	Non
unique	bool	Se a colonna è unique	Non

@sql.id – Crea la chiave primaria nella tabella referente alla classe che contiene il *tag*. Nei casi in cui la classe contiene due *tag @sql.id* ci saranno due comandi SQL per creare la chiave primaria in uscita.

Parametro	Tipo	Descrizione	Obbligatorio
column	testo	Il nome della colonna nella base di dati.	Si
generator-class	testo	Le scelte valide sono: native: Il suo valore sarà auto incrementale. assigned: Il suo valore sarà sempre fornito del programma.	Si

@sql.many-to-one – Crea una relazione *many-to-one* tra la tabella referente alla classe che contiene il *tag* e la tabella referente alla classe di ritorno del metodo associato al *tag.*

Parametro	Tipo	Descrizione	Obbligatorio
column	testo	Il nome della colonna di relazione nella base di dati.	Si
not-null	bool	Se not-null, la relazione è 1-n; altrimenti è 0…1-n	Non
unique	bool	Se unique, la relazione è 1-1; altrimenti è 1-n	Non
foreign-key	testo	Il nome della *foreign key constraint* ad essere creata	Si

@sql.many-to-many - Crea una relazione *many-to-many* tra la tabella referente alla classe che contiene il *tag* e la tabella referente alla classe descritta nel parametro *class* del *tag*. Non è possibile utilizzare la classe di ritorno, come nella relazione *many-to-one*, perché questa sarà probabilmente una collezione. Questo *tag* crea una tabella intermediaria che implementa la relazione *many-to-many.*

Parametro	Tipo	Descrizione	Obbligatorio
table	testo	Il nome della tabella di relazione tra le due classi	Si
column1	testo	Colonna nella tabella di relazione che fa riferimento alla classe attuale	Si
column2	testo	Colonna nella tabella di relazione che fa riferimento alla classe specificata per *class*	Si
class	testo	La seconda classe della relazione. Deve essere specificato soltanto il nome della classe.	Si

2.3 Relazione 1-n e n-n

Nelle relazione del tipo 1-n oppure n-n, la descrizione SQLDoclet deve essere fatta solo in una delle due classi. Mettere la descrizione in entrambi sarebbe ridondante e, quindi, non necessario.

Nei casi in cui esistono la descrizione in entrambi classi, saranno creati due comandi, uno per ogni classe.

3 Restrizioni

- Gli attributi del tipo *text* dovranno contenere solamente caratteri alfanumerici, numeri oppure il _ (*underscore*). Qualsiasi altro carattere utilizzato risulterà in errore.
- Non sarà considerato il *package* cui la classe appartiene, soltanto il nome della classe. Per questo motivo, le classi "it.polito.Utente" e "br.usp.Utente", per esempio, saranno identiche al traduttore.
- Il codice SQL generato avrà come punto di riferimento la sintassi del MySQL.
- Le primitive riconosciuti sono:

Primitive	Data Type nel MySQL
boolean	`bit`
byte	`tinyint`
char	`char(1)`
double	`double precision`
float	`float`
int	`integer`
long	`bigint`
short	`smallint`

- Le Classi riconosciuti (della API del J2SE) sono:

Classe	Data Type nel MySQL
Boolean	`bit`
Byte	`tinyint`
Character	`char(1)`
Double	`double precision`
Float	`float`
Integer	`integer`
Long	`bigint`
Short	`smallint`
String	`varchar(255)`
Blob	`blob`
Clob	`text`
GregorianCalendar	`datetime`
Date	`datetime`

- Riferimenti a classi sconosciute risulteranno in errore.
- Relazione tra 2 classi dove una non è conosciuta del traduttori (non è stata fornita nell'ingresso) risulteranno in errore.

4 Esempi

4.1 Una classe senza relazione

```
package br.usp.webcarnow.model.bean;
import java.util.Collection;
…
/**
 * @author Foo
 *
*  @sql.class
 *  table="CLIENTE"
 *
 */
public class Cliente {
	private String login;
	private String pass;

	/**
	 *  @return Ritorna il Login del Cliente. Chiave primaria.
	 *
	 *  @sql.id
	 *   generator-class="assigned"
	 *   column="LOGIN"
	 */
	public String getLogin() {
		return login;
	}

	/**
	 * @return Ritorna il Password del Cliente.
	 *
	 * @sql.property
	 *  column="PASS"
	 *  not-null="true"
	 *  unique="false"
	 */
	public String getPass() {
		return pass;
	}

}
```

Avrà in uscita il seguente codice SQL:

```
create table CLIENTE (
   LOGIN varchar(255) not null,
   PASS varchar(255) not null,
   primary key (LOGIN)
);
```

4.2 Due classi con relazione 1-n

Prima classe:

```
package br.usp.webcarnow.model.bean;
…

/**
 *  @sql.class
 *  table="FILIALE"
 */
public class Filiale {

	private Long codice;
	private String nome;
	private Collection promozioni;

	/**
	 * @return Returns the codice.
	 *
	 *  @sql.id
	 *  generator-class="assigned"
	 *  not-null="true"
	 *  column="CNPJ"
	 */
	public Long getCodice() {
		return codice;
	}


	/**
	 * @return Returns the nome
	 *
	 * @sql.property  column="NOME " not-null="true" unique="false"
	 */
	public String getNome () {
		return nome;
	}

	/**
	 * @return Returns the promozioni.
	 *
	 */
	public Collection getPromozioni() {
		return this.promozione;
	}

}
```

Seconda classe:

```
package br.usp.webcarnow.model.bean;
import java.util.Collection;
…
/**
 * @sql.class
 *  table="PROMOZIONE"
 */
public class Promozione {
	private Long codice;
	private String descrizione;
	private Filiale filiale;

	/**
	 * @sql.id generator-class="native"
	 * not-null="true"
	 * column="CODICE_PROMOZIONE"
	 */
	public Long getCodice() {
		return codice;
	}

	/**
	 * @sql.property column="DESCRIZIONE"
	 * not-null="true"
	 * unique="false"
	 */
	public String getDescrizione() {
		return descrizione;
	}

      /**
	* @return Ritorna la filiale
	*
	* @sql.many-to-one column="CODICE_FILIALE"
	* foreign-key=”PROMOZIONE_FILIALE”
	*/
	 public Filiale getFiliale() {
		return this.filiale;
	 }
```

Avrà in uscita:

```
create table FILIALE (
   CODICE bigint not null,
   NOME varchar(255) not null,
   primary key (CODICE)
);

create table PROMOZIONE (
   CODICE_PROMOZIONE bigint not null auto_increment,
   DESCRIZIONE varchar(255) not null,
   CODICE_FILIALE bigint,
   primary key (CODICE_PROMOZIONE)
);

alter table PROMOZIONE add constraint PROMOZIONE_FILIALE_00
foreign key (CODICE_FILIALE) references FILIALE (CODICE);
```

4.3 Due classi con relazione 1-1

Sarà lo stesso che la relazione 1-n, però, avrà la constraint unique nella colonna della relazione.

4.4 Due classi con relazione n-n

La prima classe:

```
package it.polito.bar;
…

/**
 *
 * @sql.class
 *  table="PROPRIETARIO"
 */
public class Proprietario {

            private Long identita;
            private String nome;
            private Collection aziende;

            /**
             * @sql.id
             * generator-class="native"
             * not-null="true"
             * column="IDENTITA"
             */
            public Long getIdentita() {
                        return identita;
            }

            /**
             * @return Ritorna il nome
             * @sql.property
             * column="NOME"
             * not-null="true"
             * unique="false"
             */
            public String getNome() {
                        return nome;
            }

      /**
       * @return Ritorna le aziende
       *
       *  Non è necessario definire di nuovo la relazione.
       *  Già è stata definita nella classe Proprietario.
       */
       public Collection getAziende () {
                  return this.aziende;
            }

}
```

E la seconda classe:

```
package it.polito.foo;
…
/**
 *
 * @sql.class
 *  table="AZIENDA"
 */
public class Azienda {

            private Long codice;
            private String nome;
            private Collection  proprietari;

            /**
             * @return Ritorna il codice
             *
             * @sql.id
             * generator-class="native"
             * not-null="true"
             * column="CODICE_AZIENDA"
             */
            public Long getCodice() {
                        return codice;
            }

            /**
             * @return Ritorna il nome della azienda
             * @sql.property
             * column="NOME"
             * not-null="true"
             * unique="false"
             */
            public String getNome() {
                        return nome;
            }

  /**
    * @return Ritorna i proprietari
    *
    * @sql.many-to-many
    *  table=”AZIENDA_PROPRIETARIO”
    *  column1="CODICE_AZIENDA"
    *  column2="CODICE_PROPRIETARIO"
    *  class=” Proprietario”
    */
     public Collection getProprietari () {
                return this.proprietari;
            }

}
```

Avrà in uscita:

```
create table AZIENDA (
   CODICE bigint not null,
   NOME varchar(255) not null,
   primary key (CODICE)
);

create table PROPRIETARIO (
   IDENTITA bigint not null,
   NOME varchar(255) not null,
   primary key (IDENTITA)
);

create table AZIENDA_PROPRIETARIO (
   CODICE_AZIENDA bigint not null,
   CODICE_PROPRIETARIO bigint not null,
   primary key (CODICE_AZIENDA, CODICE_PROPRIETARIO)
);


alter table AZIENDA_PROPRIETARIO add constraint AZIENDA_PROPRIETARIO_00
foreign key (CODICE_AZIENDA) references AZIENDA (CODICE);

alter table AZIENDA_PROPRIETARIO add constraint AZIENDA_PROPRIETARIO_01
foreign key (CODICE_PROPRIETARIO) references PROPRIETARIO (IDENTITA);
```