Laurea Specialistica in Ingegneria Informatica
Facoltà di Ingegneria
Dipartimento di Informatica e Sistemistica
Anno Accademico 2007/2008

MusicOn

Sviluppo di un'applicazione per l'interrogazione di una base di dati riguardante il mondo della musica, con accesso ai dati basato su ontologie, utilizzando il sistema QuOnto.

Corso di "Seminari di Ingegneria del Software"
Docente:
G. De Giacomo
Autori:
Casciaro Mario Palleschi Andrea

Descrizione Della Base Di Dati

Come sorgente dati per il progetto, è stata utilizzata una base di dati open-source, disponibile sul sito web http://www.musicbrainz.org/.

La base di dati contiene all'incirca 20 milioni di tuple, tra le quali:

- circa 450.000 artisti
- circa 650.000 album
- circa 7.500.000 canzoni e
- circa 1.500.000 indirizzi web

per una dimensione totale vicina ai 3 gigabyte.

Come DBMS è stato utilizzato Oracle XE

Entità Presenti Nella Base Di Dati

- La base di dati contiene informazioni riguardanti:
  - Artisti
  - Album
  - Case Discografiche
  - Canzoni
  - Release
  - Nazione(artista, release)
  - Url(artista, album, canzone)
  - Tag

Diagramma ER (1/2)

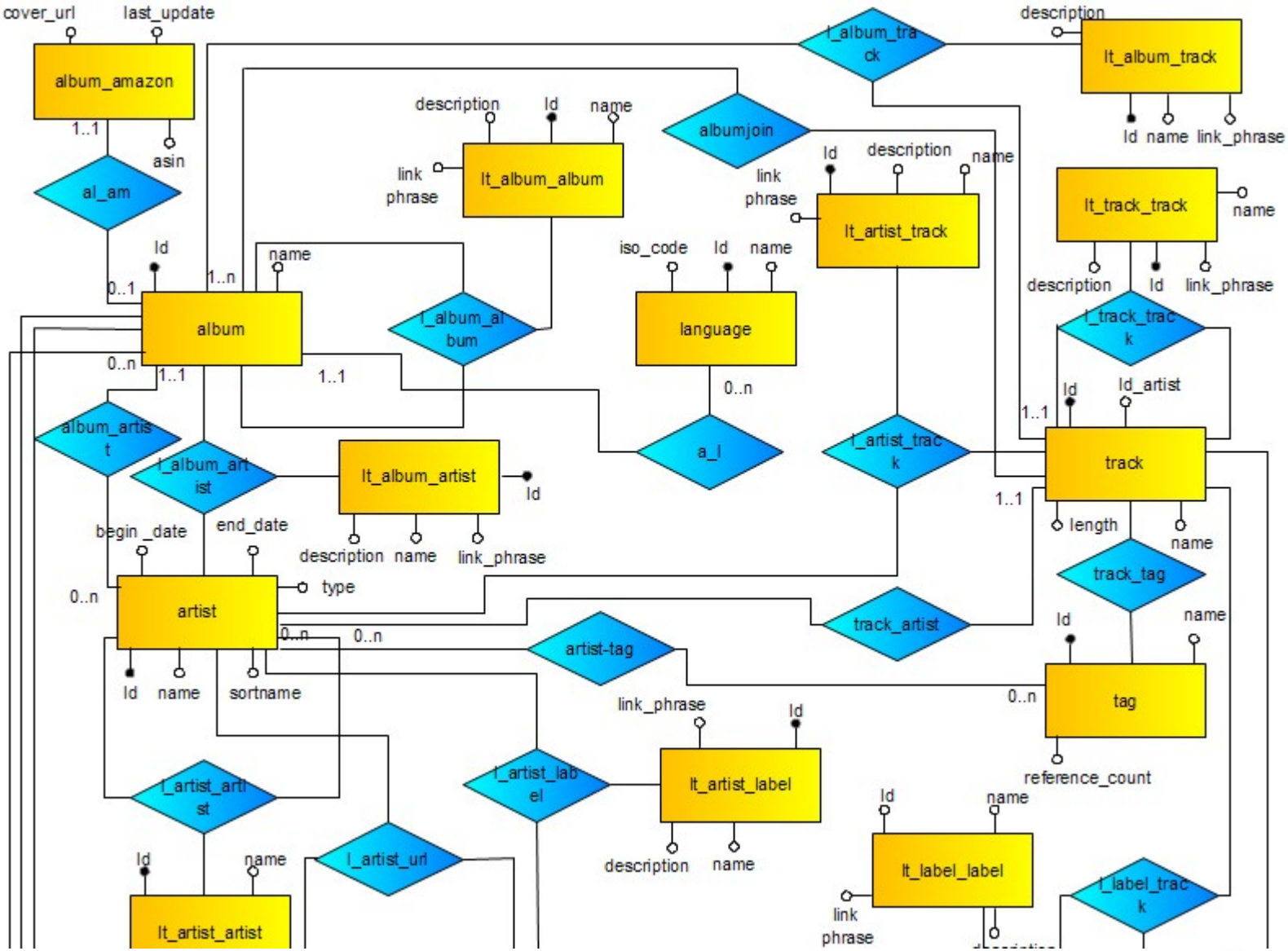
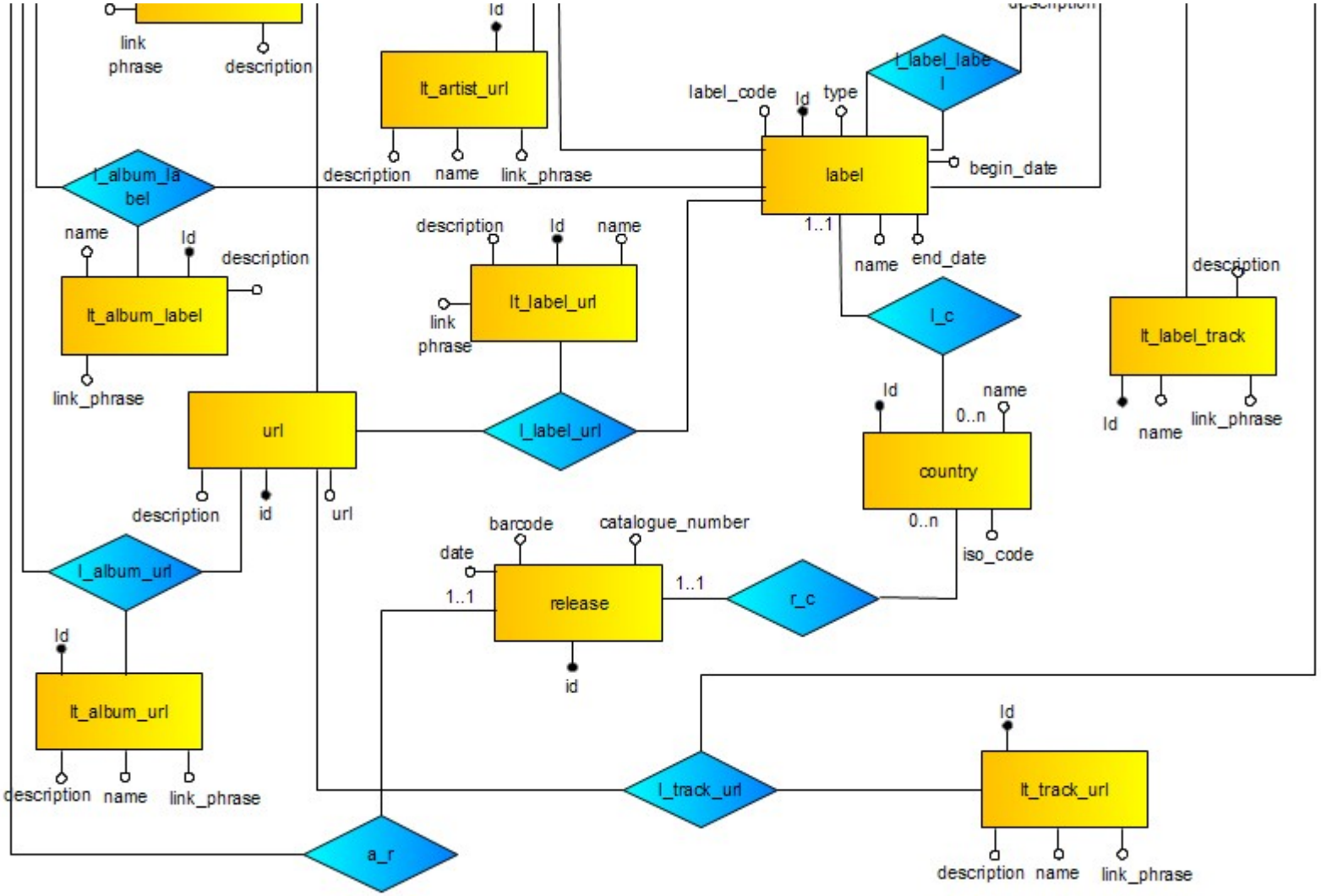


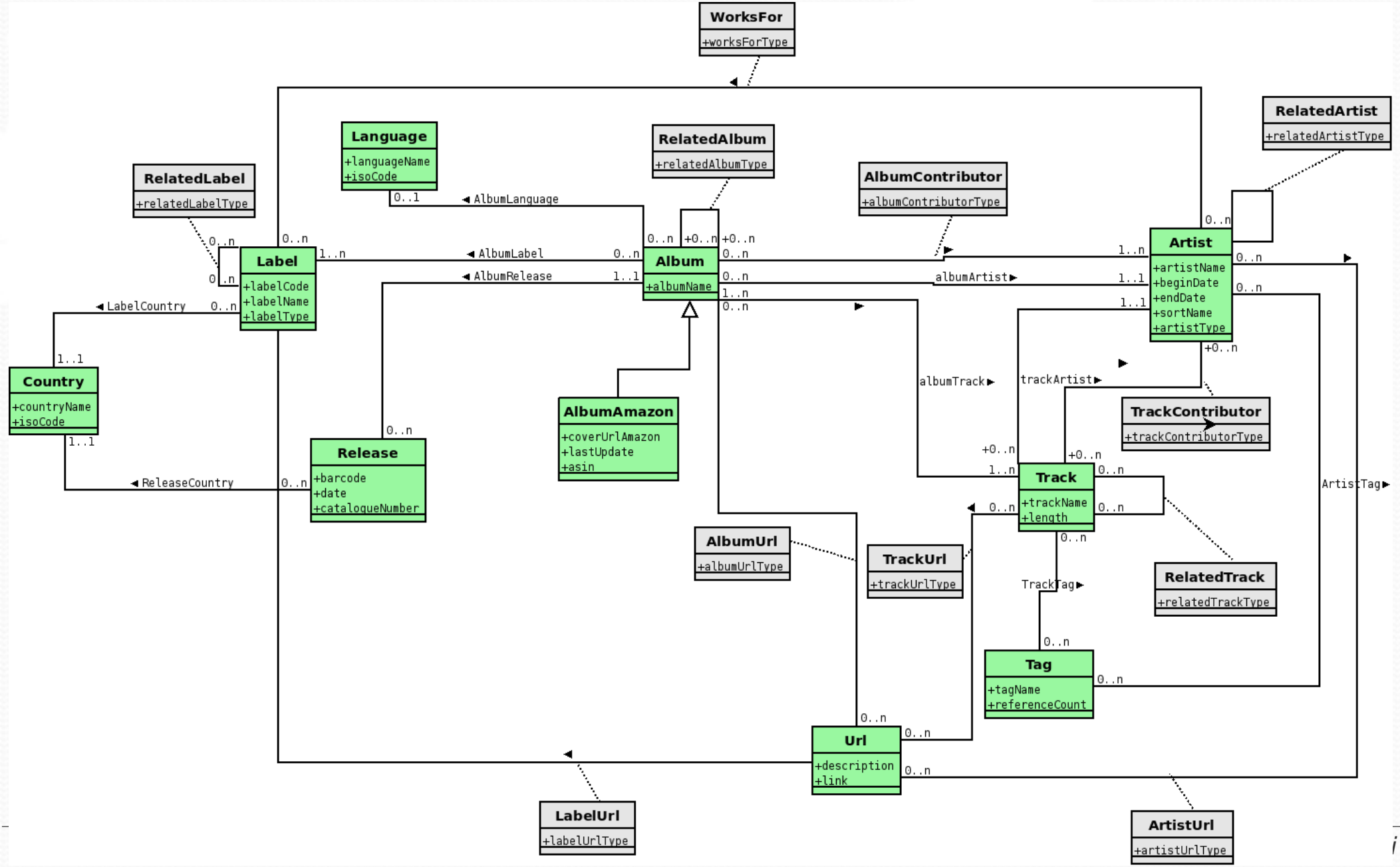
Diagramma ER (2/2)



Ontologia

L'ontologia è stata modellata attraverso un diagramma UML delle classi che rispecchia la modellazione concettuale delle specifiche.
E' possibile notare una differenza sostanziale tra il diagramma ER e l'ontologia in quanto tutte le relazioni ternarie presenti nella base di dati, sono state convertite in associazioni binarie con attributi, tale divergenza, mette in risalto il problema dell'impedance mismatch, che viene poi risolto con l'operazione di mapping.

Diagramma UML Delle Classi



DL-LiteA

- DL-LiteA é la logica descrittiva che rappresenta il compromesso migliore tra espressività e proprietà computazionali.
- Permette di esprimere sia vincoli di fuzionalità e inclusioni tra ruoli (con il vincolo che la funzionalità non sia super-ruolo)
- Prende in considerazione la distinzione tra oggetti e valori
  - Gli oggetti sono elementi di un dominio d'interpretazione
  - I valori sono elementi di tipi di dato concreti (stringhe, interi, ecc.)
- I valori sono connessi agli oggetti attraverso gli attibuti

Sintassi DL-LiteA

- Concept expressions:
  $B \rightarrow A \mid \exists Q \mid \delta(U)$
  $C \rightarrow \top \mid B \mid \neg B \mid \exists Q.C$
- Role espressions
  $Q \rightarrow P \mid P^-$
  $R \rightarrow Q \mid \neg Q$
- Value-Domain espressions
  $E \rightarrow \rho(U)$
  $F \rightarrow \top \mid D \mid T1 \mid \cdots \mid Tn$
- Attribute espressions
  $V \rightarrow U \mid \neg U$

TBox assertions:

- Concept inclusion assertion
  $B \sqsubseteq C$
- Role inclusion assertion
  $Q \sqsubseteq R$
- Value-domain inclusion assertion
  $E \sqsubseteq F$
- Attribute inclusion assertion
  $U \sqsubseteq V$
- Role functionality assertion
  $(\text{funct } Q)$
- Attribute functionality assertion
  $(\text{funct } U)$

Complessità computazionale in DL-LiteA

DL-LiteA mantiene le proprietà computazionali di DL-LiteR e DL-LiteF

- Verifica di soddisfacibilità dell' ontologia è:
  PTime nella dimesione dell'ontologia (complessità combinata)
  LogSpace nella dimensione dell'ABox (complessità risp. dati)
- TBox reasoning è:
  PTime nella dimensione della TBox
- Query answering è:
  NP-complete nella dimensione della query e dell'ontologia (complessità combinata).
  PTime nella dimensione dell'ontologia
  LogSpace nella dimensione dell' ABox (complessità risp. dati).

L'Ontologia In DL-LiteA (1/13)

Di seguito vengono riportati degli estratti di codice che rappresentano la traduzione del diagramma UML in sintassi DL -Lite A

Alphabet (Concepts):

```
Album
AlbumAmazon
Language
Label
Country
Release
Url
Tag
Track
Artist
```

L'Ontologia In DL-LiteA (2/13)

Alphabet (Roles):

```
WorksFor
RelatedAlbum
RelatedLabel
AlbumContributor
AlbumArtist
RelatedArtist
TrackContributor
TrackArtist
AlbumUrl
TrackUrl
LabelUrl
ArtistUrl
AlbumTrack
ArtistTag
TrackTag
ReleaseCountry
AlbumRelease
AlbumLabel
LabelCountry
AlbumLanguage
```

L'Ontologia In DL-LiteA (3/13)

Alphabet (Concept attributes):

```
coverUrlAmazon
lastUpdate
asin
languageName
labelName
albumName
artistName
countryName
trackName
tagName
languageIsoCode
labelCode
labelType
artistType
barcode
date
catalogueNumber
```

```
link
referenceCount
length
beginDate
endDate
sortName
```

L'Ontologia In DL-LiteA (4/13)

Alphabet (Role attributes):

```
worksForType
relatedLabelType
relatedAlbumType
AlbumContributorType
relatedArtistType
trackContributorType
albumUrlType
trackUrlType
relatedTrackType
labelUrlType
artistUrlType
```

L'Ontologia In DL-LiteA (5/13)

Concept inclusion assertions:

$$\begin{aligned}
&\text{Album} \sqsubseteq \neg\text{Language}\\
&\text{Album} \sqsubseteq \neg\text{Label}\\
&\text{Album} \sqsubseteq \neg\text{Country}\\
&\text{Album} \sqsubseteq \neg\text{Release}\\
&\text{Album} \sqsubseteq \neg\text{Url}\\
&\text{Album} \sqsubseteq \neg\text{Tag}\\
&\text{Album} \sqsubseteq \neg\text{Track}\\
&\text{Album} \sqsubseteq \neg\text{Artist}\\
&\text{Language} \sqsubseteq \neg\text{Label}\\
&\text{Language} \sqsubseteq \neg\text{Country}\\
&\text{Language} \sqsubseteq \neg\text{Release}\\
&\text{Language} \sqsubseteq \neg\text{Url}\\
&\text{Language} \sqsubseteq \neg\text{Tag}\\
&\text{Language} \sqsubseteq \neg\text{Track}\\
&\text{Language} \sqsubseteq \neg\text{Artist}\\
&\text{Country} \sqsubseteq \neg\text{Release}\\
&\text{Country} \sqsubseteq \neg\text{Url}
\end{aligned}$$

$$\begin{aligned}
&\text{Country} \sqsubseteq \neg\text{Tag}\\
&\text{Country} \sqsubseteq \neg\text{Track}\\
&\text{Country} \sqsubseteq \neg\text{Artist}\\
&\text{Release} \sqsubseteq \neg\text{Url}\\
&\text{Release} \sqsubseteq \neg\text{Tag}\\
&\text{Release} \sqsubseteq \neg\text{Track}\\
&\text{Release} \sqsubseteq \neg\text{Artist}\\
&\text{Url} \sqsubseteq \neg\text{Tag}\\
&\text{Url} \sqsubseteq \neg\text{Track}\\
&\text{Url} \sqsubseteq \neg\text{Artist}\\
&\text{Tag} \sqsubseteq \neg\text{Track}\\
&\text{Tag} \sqsubseteq \neg\text{Artist}\\
&\text{Track} \sqsubseteq \neg\text{Artist}\\
&\text{AlbumAmazon} \sqsubseteq \text{Album}
\end{aligned}$$

L'Ontologia In DL-LiteA (6/13)

Concept attribute functionality assertions:

```
(funct coverUrlAmazon)
(funct lastUpdate)
(funct asin)
(funct languageName)
(funct labelName)
(funct albumName)
(funct artistName)
(funct countryName)
(funct trackName)
(funct tagName)
(funct languageIsoCode)
(funct countryIsoCode)
(funct labelCode)
(funct barCode)
(funct labelType)
(funct artistType)
```

```
(funct date)
(funct catalogueNumber)
(funct link)
(funct referenceCount)
(funct length)
(funct beginDate)
(funct endDate)
(funct sortName)
```

L'Ontologia In DL-LiteA (7/13)

Role attribute functionality assertions:

```
(funct worksForType)
(funct relatedLabelType)
(funct relatedAlbumType)
(funct albumContributorType)
(funct relatedArtistType)
(funct trackContributorType)
(funct albumUrlType)
(funct trackUrlType)
(funct relatedTrackType)
(funct labelUrlType)
(funct artistUrlType)
```

L'Ontologia In DL-LiteA (8/13)

Concept attribute domain:

$\delta(\text{albumName}) \sqsubseteq \text{Album}$
$\text{Album} \sqsubseteq \delta(\text{albumName})$
$\delta(\text{coverUrlAmazon}) \sqsubseteq \text{AlbumAmazon}$
$\text{AlbumAmazon} \sqsubseteq \delta(\text{coverUrlAmazon})$
$\delta(\text{lastUpdate}) \sqsubseteq \text{AlbumAmazon}$
$\text{AlbumAmazon} \sqsubseteq \delta(\text{lastUpdate})$
$\delta(\text{asin}) \sqsubseteq \text{AlbumAmazon}$
$\text{AlbumAmazon} \sqsubseteq \delta(\text{asin})$
$\delta(\text{languageName}) \sqsubseteq \text{Language}$
$\text{Language} \sqsubseteq \delta(\text{languageName})$
$\delta(\text{languageIsoCode}) \sqsubseteq \text{Language}$
$\text{Language} \sqsubseteq \delta(\text{languageIsoCode})$
$\delta(\text{labelName}) \sqsubseteq \text{Label}$
$\text{Label} \sqsubseteq \delta(\text{labelName})$
$\delta(\text{labelCode}) \sqsubseteq \text{Label}$
$\text{Label} \sqsubseteq \delta(\text{labelCode})$
$\delta(\text{labelType}) \sqsubseteq \text{Label}$

$\text{Label} \sqsubseteq \delta(\text{labelType})$
$\delta(\text{artistName}) \sqsubseteq \text{Artist}$
$\text{Artist} \sqsubseteq \delta(\text{artistName})$
$\delta(\text{beginDate}) \sqsubseteq \text{Artist}$
$\text{Artist} \sqsubseteq \delta(\text{beginDate})$
$\delta(\text{endDate}) \sqsubseteq \text{Artist}$
$\text{Artist} \sqsubseteq \delta(\text{endDate})$
$\delta(\text{sortName}) \sqsubseteq \text{Artist}$
$\text{Artist} \sqsubseteq \delta(\text{sortName})$
$\delta(\text{artistType}) \sqsubseteq \text{Artist}$
$\text{Artist} \sqsubseteq \delta(\text{artistType})$
$\delta(\text{countryName}) \sqsubseteq \text{Country}$
$\text{Country} \sqsubseteq \delta(\text{countryName})$
$\delta(\text{countryIsoCode}) \sqsubseteq \text{Country}$
...

L'Ontologia In DL-LiteA (9/13)

Roles :

$\exists AlbumTrack \sqsubseteq Album$
$\exists AlbumTrack^{-} \sqsubseteq Track$
$\exists ArtistTag \sqsubseteq Artist$
$\exists ArtistTag^{-} \sqsubseteq Tag$
$\exists TrackTag \sqsubseteq Track$
$\exists TrackTag^{-} \sqsubseteq Tag$
$\exists ReleaseCountry \sqsubseteq Release$
$\exists ReleaseCountry^{-} \sqsubseteq Country$
$\exists AlbumRelease \sqsubseteq Album$
$\exists AlbumRelease^{-} \sqsubseteq Release$
$\exists AlbumArtist \sqsubseteq Album$
$\exists AlbumArtist^{-} \sqsubseteq Artist$
$\exists TrackArtist \sqsubseteq Artist$
$\exists TrackArtist^{-} \sqsubseteq Track$
$\exists AlbumLabel \sqsubseteq Album$
$\exists AlbumLabel^{-} \sqsubseteq Label$
$\exists LabelCountry \sqsubseteq Label$
$\exists LabelCountry^{-} \sqsubseteq Country$

$\exists AlbumLanguage \sqsubseteq Album$
$\exists AlbumLanguage^{-} \sqsubseteq Language$
$\exists WorksFor \sqsubseteq Artist$
$\exists WorksFor^{-} \sqsubseteq Label$
$\exists RelatedAlbum \sqsubseteq Album$
$\exists RelatedAlbum^{-} \sqsubseteq Album$
$\exists RelatedLabel \sqsubseteq Label$
$\exists RelatedLabel^{-} \sqsubseteq Label$
$\exists AlbumContributor \sqsubseteq Album$
$\exists AlbumContributor^{-} \sqsubseteq Artist$
$\exists RelatedArtist \sqsubseteq Artist$
$\exists RelatedArtist^{-} \sqsubseteq Artist$
$\exists TrackContributor \sqsubseteq Track$
$\exists TrackContributor^{-} \sqsubseteq Artist$
$\exists AlbumUrl \sqsubseteq Album$
$\exists AlbumUrl^{-} \sqsubseteq Url$
$\exists TrackUrl \sqsubseteq Track$
.....

L'Ontologia In DL-LiteA (10/13)

Role attributes:

$\exists \delta(\text{albumContributorType}) \sqsubseteq \text{Album}$
$\exists \delta(\text{albumContributorType})^- \sqsubseteq \text{Artist}$
$\exists \delta(\text{worksForType}) \sqsubseteq \text{Artist}$
$\exists \delta(\text{worksForType})^- \sqsubseteq \text{Label}$
$\exists \delta(\text{relatedAlbumType}) \sqsubseteq \text{Album}$
$\exists \delta(\text{relatedAlbumType})^- \sqsubseteq \text{Album}$
$\exists \delta(\text{relatedArtistType}) \sqsubseteq \text{Artist}$
$\exists \delta(\text{relatedArtistType})^- \sqsubseteq \text{Artist}$
$\exists \delta(\text{trackContributorType}) \sqsubseteq \text{Track}$
$\exists \delta(\text{trackContributorType})^- \sqsubseteq \text{Artist}$
$\exists \delta(\text{albumUrlType}) \sqsubseteq \text{Album}$
$\exists \delta(\text{albumUrlType})^- \sqsubseteq \text{Url}$
$\exists \delta(\text{trackUrlType}) \sqsubseteq \text{Track}$
$\exists \delta(\text{trackUrlType})^- \sqsubseteq \text{Url}$
$\exists \delta(\text{labelUrlType}) \sqsubseteq \text{Label}$
$\exists \delta(\text{labelUrlType})^- \sqsubseteq \text{Url}$
$\exists \delta(\text{artistUrlType}) \sqsubseteq \text{Artist}$
$\exists \delta(\text{artistUrlType})^- \sqsubseteq \text{Url}$

L'Ontologia In DL-LiteA (11/13)

Role functionality assertions:

$(funct\ AlbumRelease^{-})$
$(funct\ AlbumArtist)$
$(funct\ TrackArtist)$
$(funct\ ReleaseCountry)$
$(funct\ LabelCountry)$

Role attribute domain:

$\delta(relatedLabelType) \sqsubseteq RelatedLabel$
$\delta(worksForType) \sqsubseteq WorksFor$
$\delta(relatedAlbumType) \sqsubseteq RelatedAlbum$
$\delta(albumContributorType) \sqsubseteq AlbumContributor$
$\delta(relatedArtistType) \sqsubseteq RelatedArtist$
$\delta(labelUrlType) \sqsubseteq LabelUrl$
$\delta(albumUrlType) \sqsubseteq AlbumUrl$
$\delta(trackUrlType) \sqsubseteq TrackUrl$
$\delta(artistUrlType) \sqsubseteq ArtistUrl$
$\delta(relatedTrackType) \sqsubseteq RelatedTrack$
$\delta(trackContributorType) \sqsubseteq TrackContributor$

L'Ontologia In DL-LiteA (12/13)

Concept partecipation in role:

$\text{Album} \sqsubseteq \exists \text{AlbumTrack}$
$\text{Track} \sqsubseteq \exists \text{AlbumTrack}^-$

Concept and role attribute value range:

$\rho(\text{languageName}) \sqsubseteq \text{xsd:string}$
$\rho(\text{isoCode}) \sqsubseteq \text{xsd:string}$
$\rho(\text{albumName}) \sqsubseteq \text{xsd:string}$
$\rho(\text{trackCount}) \sqsubseteq \text{xsd:int}$
$\rho(\text{labelName}) \sqsubseteq \text{xsd:string}$
$\rho(\text{labelCode}) \sqsubseteq \text{xsd:string}$
$\rho(\text{labelType}) \sqsubseteq \text{xsd:int}$
$\rho(\text{countryName}) \sqsubseteq \text{xsd:string}$
$\rho(\text{isoCode}) \sqsubseteq \text{xsd:string}$
$\rho(\text{barcode}) \sqsubseteq \text{xsd:string}$
$\rho(\text{date}) \sqsubseteq \text{xsd:date}$

$\rho(\text{catalogueNumber}) \sqsubseteq \text{xsd:string}$
$\rho(\text{coverUrlAmazon}) \sqsubseteq \text{xsd:string}$
$\rho(\text{lastUpdate}) \sqsubseteq \text{xsd:date}$
$\rho(\text{asin}) \sqsubseteq \text{xsd:string}$
$\rho(\text{description}) \sqsubseteq \text{xsd:string}$
$\rho(\text{link}) \sqsubseteq \text{xsd:string}$
$\rho(\text{tagName}) \sqsubseteq \text{xsd:string}$
$\rho(\text{referenceCount}) \sqsubseteq \text{xsd:int}$
$\rho(\text{trackName}) \sqsubseteq \text{xsd:string}$
$\rho(\text{length}) \sqsubseteq \text{xsd:int}$
$\rho(\text{artistName}) \sqsubseteq \text{xsd:string}$
.....

L'Ontologia In DL-LiteA (13/13)

Role inclusion assertion:

$$\begin{aligned}
&\mathit{WorksFor} \sqsubseteq \neg \mathit{RelatedLabel} \\
&\mathit{WorksFor} \sqsubseteq \neg \mathit{RelatedAlbum} \\
&\mathit{WorksFor} \sqsubseteq \neg \mathit{AlbumContributor} \\
&\mathit{WorksFor} \sqsubseteq \neg \mathit{RelatedArtist} \\
&\mathit{WorksFor} \sqsubseteq \neg \mathit{TrackContributor} \\
&\mathit{WorksFor} \sqsubseteq \neg \mathit{AlbumUrl} \\
&\mathit{WorksFor} \sqsubseteq \neg \mathit{TrackUrl} \\
&\mathit{WorksFor} \sqsubseteq \neg \mathit{RelatedTrack} \\
&\mathit{WorksFor} \sqsubseteq \neg \mathit{LabelUrl} \\
&\mathit{WorksFor} \sqsubseteq \neg \mathit{ArtistlUrl} \\
&\mathit{RelatedLabel} \sqsubseteq \neg \mathit{RelatedAlbum} \\
&\mathit{RelatedLabel} \sqsubseteq \neg \mathit{AlbumContributor} \\
&\mathit{RelatedLabel} \sqsubseteq \neg \mathit{RelatedArtist} \\
&\mathit{RelatedLabel} \sqsubseteq \neg \mathit{TrackContributor} \\
&\mathit{RelatedLabel} \sqsubseteq \neg \mathit{AlbumUrl}
\end{aligned}$$

$$\begin{aligned}
&\mathit{RelatedLabel} \sqsubseteq \neg \mathit{TrackUrl} \\
&\mathit{RelatedLabel} \sqsubseteq \neg \mathit{RelatedTrack} \\
&\mathit{RelatedLabel} \sqsubseteq \neg \mathit{LabelUrl} \\
&\mathit{RelatedLabel} \sqsubseteq \neg \mathit{ArtistUrl} \\
&\mathit{RelatedAlbum} \sqsubseteq \neg \mathit{AlbumContributor} \\
&\mathit{RelatedAlbum} \sqsubseteq \neg \mathit{RelatedArtist} \\
&\mathit{RelatedAlbum} \sqsubseteq \neg \mathit{TrackContributor} \\
&\mathit{RelatedAlbum} \sqsubseteq \neg \mathit{AlbumUrl} \\
&\mathit{RelatedAlbum} \sqsubseteq \neg \mathit{TrackUrl} \\
&\mathit{RelatedAlbum} \sqsubseteq \neg \mathit{RelatedTrack} \\
&\mathit{RelatedAlbum} \sqsubseteq \neg \mathit{LabelUrl} \\
&\mathit{RelatedAlbum} \sqsubseteq \neg \mathit{ArtistUrl} \\
&\mathit{AlbumContributor} \sqsubseteq \neg \mathit{RelatedArtist} \\
&\mathit{AlbumContributor} \sqsubseteq \neg \mathit{TrackContributor} \\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots
\end{aligned}$$

OBDA & Mapping (1/3)

- Un ”Ontology-Based Data Access System” è una tripla O = <T,M,D> in cui
  - T è una TBox.
  - D è un database relazionale.
  - M è un insieme di **mapping assertions** tra T e D.
- L' ABox è quindi gestita da un sistema ”esterno” (DBMS)
- Problema dell”**impedence mismatch**”:
  - Sistema relazionale contiene valori
  - L'ABox, invece, dovrebbe contenere oggetti (virtuali)

OBDA & Mapping (2/3)

- LAV (Local-as-View)
  - Esprime le tabelle relazionali delle sorgenti in termini di viste sull'ontologia.
  - Il processamento delle query richiede un complesso processo di **ragionamento**.
- GAV (Global-as-View)
  - Esprime concetti, ruoli, attributi dell'ontologia in termini di viste sulle sorgenti.
  - Il processamento delle query richiede un più semplice meccanismo di **unfolding**.

OBDA & Mapping (3/3)

Una ”mapping assertion” tra un database D e una TBox T ha la forma

$$\Phi \rightsquigarrow \Psi$$

Dove:

- Φ è una query SQL su D, di arità n > 0.
- Ψ è una conjunctive query su T di arità n' > 0. In cui nei termini:
  - Si fa riferimento alle variabili di Φ per denotare semplici valori.
  - Si usa una opportuna **funzione di skolem** f(v1,..,vn) di arità n, con v1..vn variabili di Φ, per denotare oggetti astratti (risolve il problema dell'impedence mismatch).

A-Box GAV Mapping

Di seguito vengono riportati alcuni GAV mapping tra l'ontologia e la base di dati:

`SELECT artist.id as artist_id,`
`artist.name as artist_name,`
`artist.begindate as artist_begindate,`
`artist.enddate as artist_enddate,`
`artist.sortname as artist_sortname, artist.type`
`as artist_type`
`FROM artist`
	~>	`Artist(artist(artist_id)),`
		`artistName(artist(artist_id), artist_name:`
		`xsd:string),`
		`beginDate(artist(artist_id), artist_begindate:`
		`xsd:date),`
		`endDate(artist(artist_id), artist_enddate:`
		`xsd:date),`
		`sortName(artist(artist_id), artist_sortname:`
		`xsd:string),`
		`artistType(artist(artist_id), artist_type:`
		`xsd:int)`

`SELECT l_album_url.link0 as link0,`
`l_album_url.link1 as link1,`
`lt_album_url.name as name`
`FROM l_album_url, lt_album_url WHERE`
`l_album_url.link_type = lt_album_url.id`
	~>	`AlbumUrl(album(link0), url(link1)),`
		`albumUrlType(album(link0), url(link1), name:`
		`xsd:string)`

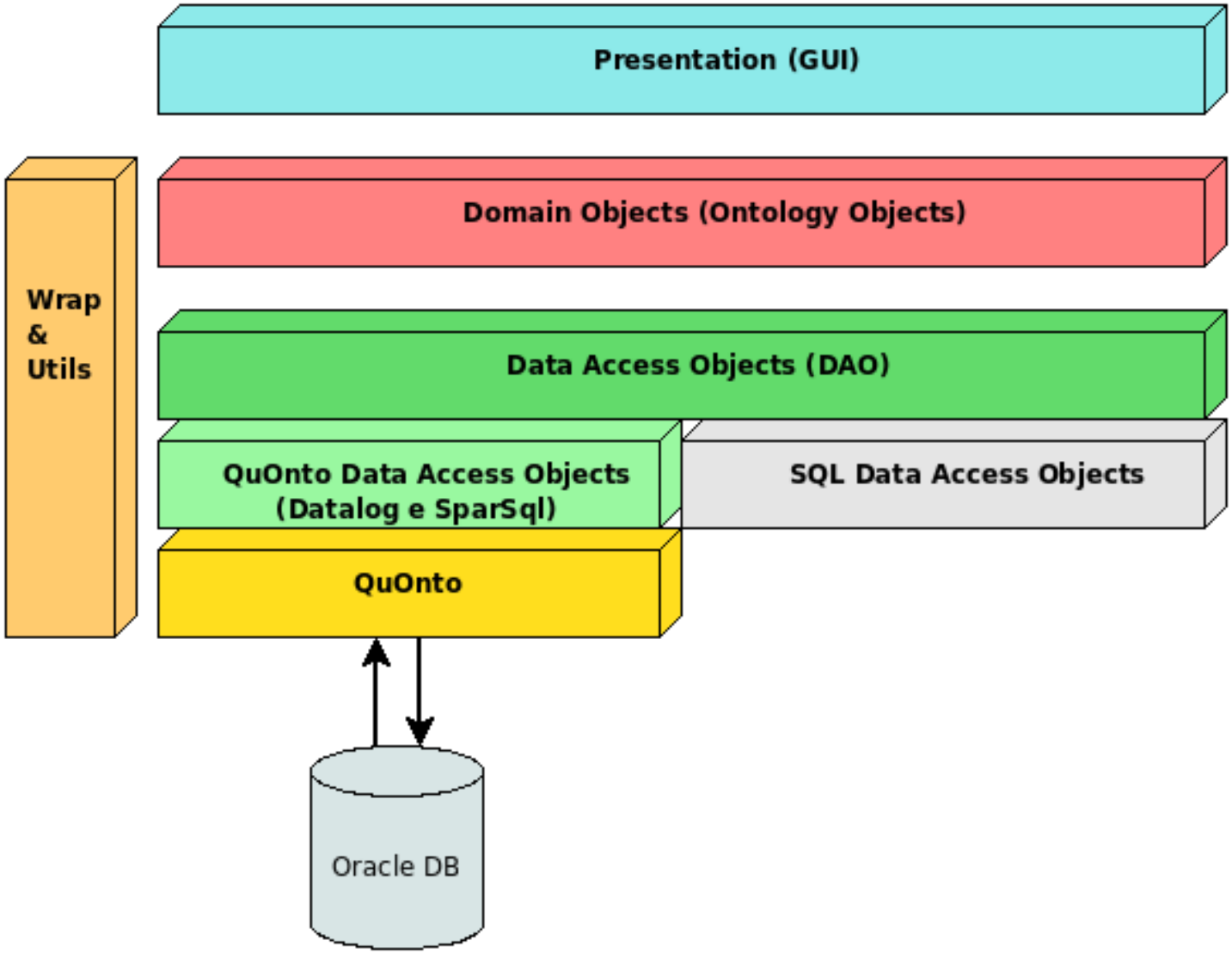
MusicOn

Progettazione e sviluppo

Struttura del Sistema (1/2)

- Separazione basilare nei **3 livelli** standard
  - Livello di presentazione
  - Livello della logica applicativa (Domain Objects)
  - Livello di Accesso ai Dati (DAOs)
- Utilizzo di **design patterns** per aumentare modularizzazione
  - Singleton (Classi wrapper e di utilità)
  - Abstract factory (per i DAOs)

Struttura del Sistema (2/2)



Livello di Presentazione

- Java Swing GUI
  - Accede direttamente allo strato della logica applicativa
  - Multithreaded (query simultanee)
    - Un thread per gestire eventi dell'interfaccia
    - Un thread per ogni query in esecuzione
    - Un thread per gestire feedback di ogni esecuzione

Livello della logica applicativa

- Domain Objects
  - Proprietà: riflettono la struttura della Tbox dell'ontologia
  - Sarebbe possibile una generazione automatizzata TBox

    → Domain Objects

TBox → Domain Objects (1/5)

- Concetti e Ruoli con attributi
  - Ad ogni concetto/ruolo(con attributi) nella TBox corrisponde un DO
  - Il nome del DO sarà uguale al nome del corrispondente concetto/ruolo(con attributi) nella TBox

→

```
public class Album {}
```

TBox → Domain Objects (2/5)

- Object id
  - E' l'id assegnato da QuOnto ad un oggetto attraverso la funzione di skolem definita nel mapping (es. 'album(12345)')
  - Apposita variabile di instanza nel DO

```
public class Album {
   protected String objId;
}
```

TBox → Domain Objects (3/5)

- Attributi (di concetto o di ruolo)
  - Ad ogni attributo corrisponde una **variabile di istanza** nel DO

→

```
public class Album {
    protected String albumName;
}
```

TBox → Domain Objects (4/5)

- Partecipazione nei ruoli
  - Ogni partecipazione in un ruolo corrisponde ad una **variabile di istanza** nel DO
  - Ad ognuna di tali variabili viene associata una **variabile booleana** per gestire il meccanismo del lazy read

TBox → Domain Objects (5/5)

```
public class Album {
   //Ruolo senza attributi, molteplicità max 1)
    protected Artist artist;
    protected boolean artistLoaded;

    //Ruolo senza attributi, molteplicità max N)
    protected List<Label> labels;
    protected boolean labelsLoaded;

    //Ruolo con attributi, molteplicità max N)
    protected List<RelatedAlbum> relatedAlbums;
    protected boolean relatedAlbumsLoaded;
}
```

Domain Objects - Lazy Read

- Caricamento on-demand degli oggetti collegati ad un DO già istanziato
- Motivazione principale: **prestazioni**

```
public  List<Release> getReleases()
    throws MusicOnException
{
    if(!releasesLoaded)
    {
            releases = MusicOnManager.getInstance().
               getDAOFactory().getReleaseDAO().
               findByAlbum(this);
            releasesLoaded = true;
    }
    return releases;
}
```

Domain Objects – Operazioni

- Ogni DO è provvisto di metodi per effettuare le operazioni richieste nelle specifiche

```
public static List<Album> findAlbums(String albumName,
              String albumArtist, String albumLabel,
              String startReleaseDate, String endReleaseDate)
              throws MusicOnException
{
    [...]
}
```

Data Access Objects (1/2)

- Solitamente un DAO per ogni entità logica a livello dati (es. Tabella relazionale in db)
- Nel nostro caso ”**entità logica livello dati**” = ”**entità a livello concettuale**” (stiamo utilizzando un OBDA)
- Corrispondenza tra DAOs, DOs e TBox
- Quindi, nessun ”impedence mismatch” tra Domain Objects e Data Access Objects!

Data Access Objects (2/2)

- Utilizzo del design pattern **Abstract Factory** (quindi possibilità di supportare diversi metodi di accesso ai dati)
- Ogni DAO contiene metodi per gestire le informazioni del relativo concetto/ruolo.

DAO – Flusso di Esecuzione (1/4)

1) Costruzione delle query
2) Esecuzione della query
3) Processamento del risultato

DAO – Flusso di Esecuzione (2/4)

- Costruzione della query
  - Query in formato **Datalog** o **SparSql**
  - Query statiche o **dinamiche** (es. Unica funzione di ricerca di un album ma possibilità di cercare per solo nome, oppure per nome e data di rilascio, oppure per nome e casa discografica, ecc.)

DAO – Flusso di Esecuzione (3/4)

- Esecuzione della query
  - Utilizzo di appositi strumenti di wrapping di MusicOn per far eseguire la valutazione della query a QuOnto

```
IEvaluationResult result = QuontoManager.getInstance().
        evaluateDatalogQuery(...);
```

```
IEvaluationResult result = QuontoManager.getInstance().
        evaluateSparSqlQuery(...);
```

DAO – Flusso di Esecuzione (4/4)

- Processamento del risultato
  - Trasformazione dei dati restituiti in Domain Object(s).
  - Trasformazione automatizzata tramite la classe *ResultsetToObject*

```
Track track = ResultsetToObject.getInstance().convert(result,
          ResultsetToObject.getInstance().track,
          1, false, new Track());
```

Interfacciamento con QuOnto

- **Classe *QuontoManager*** incapsula tutto l'interfacciamento con le API QuOnto. Contiene metodi per:
  - **Inizializzare** il sistema (ontologia, mapping, ecc.)
  - **Eseguire le query** in vari formati (Raw API, Datalog, SparSql) fornendo meccanismi integrati di logging, gestione errori, workaround a un bug di QuOnto.

La classe *ResultsetToObject* (1/3)

- Automatizza la conversione di un resultset in un Domain Object.
- Prende in input:
  - Il result set
  - Un mapping ”nome variabile del DO” → ”numero colonna del result set”
  - Lo spiazzamento della colonna del result set
  - L'oggetto in cui copiare i dati

```
public  <T> T convert(IEvaluationResult result,
          HashMap<String, Integer> mapping,
          int startColIndex, boolean closeResultSet, T obj)
```

La classe *ResultsetToObject* (2/3)

- Il mapping non è altro che un oggetto della classe Map di java.utils :

```
HashMap<String, Integer> artist = new HashMap<String, Integer>();

artist.put("objId", new Integer(1));
artist.put("name", new Integer(2));
artist.put("beginDate", new Integer(3));
artist.put("endDate", new Integer(4));
artist.put("sortName", new Integer(5));
artist.put("type", new Integer(6));
```

La classe *ResultsetToObject* (3/3)

- Internamente il metodo di conversione utilizza:
  - Vari metodi per tradurre un oggetto ***Constant*** che rappresenta i dati nel result object di QuOnto in un dato Java (es. Int, String, Calendar)
  - I meccanismi di **reflection** di Java per settare i valori dell'oggetto

MusicOn

Note sull'utilizzo del sistema QuOnto

Analisi prestazionale (1/3)

- ”Reasoner” di QuOnto molto veloce per l'ontologia considerata
- Prestazioni generali del sistema OBDA fortemente **influenzate dal DBMS utilizzato**
  - Query molto complesse a livello di annidamento, prestazioni **legate alla capacità di ottimizzazione delle query**.

Analisi prestazionale (2/3)

```
q(albumNameVar) :-  albumName(albumVar, albumNameVar),
                        artistName(artistVar, 'Pink Floyd'),
                        AlbumArtist(albumVar, artistVar)
```

```
SELECT DISTINCT alias_2.term2 FROM (SELECT DISTINCT CONCAT('album(',CONCAT(album_id,')')) AS
     term1,CONCAT('artist(',CONCAT(album_artist,')')) AS term2 FROM (
     SELECT album.id as album_id, album.name as album_name, album.language as album_language,
          album.artist as album_artist
          FROM album
          ) DummyTable) alias_0 ,
     (SELECT DISTINCT CONCAT('artist(',CONCAT(artist_id,')')) AS term1,ARTIST_NAME AS term2 FROM (
     SELECT artist.id as artist_id, artist.name as artist_name,
          artist.begindate as artist_begindate, artist.enddate as artist_enddate,
          artist.sortname as artist_sortname, artist.type as artist_type FROM artist
          ) DummyTable) alias_1 ,
     (SELECT DISTINCT CONCAT('album(',CONCAT(album_id,')')) AS term1,ALBUM_NAME AS term2 FROM (
     SELECT album.id as album_id, album.name as album_name, album.language as album_language,
          album.artist as album_artist
          FROM album
          ) DummyTable) alias_2
WHERE alias_0.term2=alias_1.term1  AND alias_1.term2='Pink Floyd'
```

Analisi prestazionale (3/3)

- Prestazioni con **MySql 5** scarse
  - Tempo di esecuzione 15 minuti con clausola DISTINCT
  - Tempo di esecuzione 10 minuti senza clausola DISTINCT
  - MySql sembra concretizzare le tabelle risultanti dalle query annidate!
- Prestazioni con **Oracle XE 10g** molto buone
  - Tempo di esecuzione 1 secondo con clausola DISTINCT!

Note positive

- **Query direttamente sull'ontologia**!
- **Impedence Mismatch** risolto in modo trasparente all'applicazione durante il mapping
- **API** semplici e intuitive, in particolare
  - Semplice inizializzazione, esecuzione delle query, configurazione.
  - Abbastanza intuitiva la scrittura della TBox e del mapping in formato XML
  - Semplice e veloce utilizzare Datalog
  - Semplice e potente l'utilizzo di SparSql

Note negative (1/3)

- **File XML** per Tbox e mapping molto prolissi (>1500 righe)
- **Query congiuntive** espresse in formato XML o direttamente tramite API poco usabili.
- QuontoEql non supporta gli **operatori SQL LOWER/ UPPER** all'interno delle query SparSql (implica impossibilità di eseguire query con stringhe case-insensitive)

Note negative (2/3)

- Poco naturale la **manipolazione del result set**:
  - Estrazione delle colonne solo tramite indice numerico
  - Utilizzo della classe *Constant* per rappresentare tutti i dati
- **Poca leggibilità delle query SQL** generate da QuOnto (quindi, difficoltà nell'eseguire il debug del mapping)

Note negative (3/3)

- **Bug** nella gestione della chiusura degli statements SQL.
  - Errore "*maximum open cursors exceeded*" lanciato dal database alla query consecutiva numero 300
  - Workaround: alla query numero 299 chiamata della funzione

```
abox.getDataSourceManager().closePendingJDBCObjects();
```

SparSQL (1/3)

- Nelle ontologie si presuppone la proprietà **OWA**
  - Tutto quello di cui si ha conoscenza è vero, il resto può essere vero o falso (si presuppone di avere una informazione incompleta)
  - Nel query answering non si può andare oltre le UCQ
  - **FOL risulta indecidibile**
- Le basi di dati relazionali seguono la regola **CWA**
  - Tutto ciò che è conosciuto è vero, il resto è falso.
  - **FOL risulta decidibile** (infatti query in SQL)

SparSQL (2/3)

- Soluzione: EQL (Epistemic Query Language)
  - FOL + operatore epistemico **K** (conoscenza minimale)
  - Proprietà fondamentale di EQL: **su tutto ciò che si conosce si ha informazione completa** (CWA)

$$KB \models \varphi \quad \Leftrightarrow \quad KB \models \mathbf{K}\,\varphi$$

$$KB \not\models \varphi \quad \Leftrightarrow \quad KB \models \neg\mathbf{K}\,\varphi$$

SparSQL (3/3)

- Risultato: **SparSQL** = SparQL + SQL

SELECT ListaAttributiOEspressioni
FROM (sparqltable(< QuerySparql >) alias) +
[where CondizioniSemplici]
[group by ListaAttributiDiRaggruppamento]
[having CondizioniAggregate]
[order by ListaAttributiDiOrdinamento]

- La sparqltable è vista come una tabella relazionale, risultato di una query SparQL sull'ontologia, a cui è applicato l'operatore **K**.
- Su tale tabella si può eseguire qualsiasi operazione SQL

Query di esempio (1/3)

Si vuole conoscere tutti i nomi degli artisti taggati come “rock”, che hanno rilasciato almeno un album in Italia.

```
q(artistNameVar) : artistName(artistVar, artistNameVar),
   AlbumArtist(albumVar,artistVar),
   ArtistTag(artistVar,tagVar), tagName(tagVar, 'rock'),
   AlbumRelease(albumVar, releaseVar),
   ReleaseCountry(releaseVar, countryVar),
   countryName(countryVar, 'Italy')
```

Query eseguita in 2 secondi, con un risultato di 94 righe.

Query di esempio (2/3)

Si vogliono conoscere i nomi degli artisti che hanno contribuito (con il tipo di tale contributo) alla realizzazione di almeno un album dei Pink Floyd nel periodo 1970-1980.

```
SELECT DISTINCT t.contrName, t.contrType
FROM
sparqltable(
    SELECT ?contrName ?contrType ?releaseDate
    WHERE {
        ?artist :artistName 'Pink Floyd'.
        ?album :AlbumArtist ?artist.
        (?album :AlbumContributor ?contr) :albumContributorType ?contrType.
        ?contr :artistName ?contrName.
        ?album :AlbumRelease ?release.
        ?release :date ?releaseDate
    }
) t
WHERE t.releaseDate BETWEEN '1970-01-01' AND '1980-12-31'
```

La query viene eseguita in 3 secondi e restituisce 47 righe.

Query di esempio (3/3)

Si vuole conoscere il nome dell'ingegnere del suono che ha collaborato alla realizzazione nel maggior numero di album.

```
SELECT DISTINCT t.engName, COUNT(t.album)
FROM
sparqltable(
    SELECT ?engName ?album
    WHERE {
        (?album :AlbumContributor ?eng) :albumContributorType 'engineer'.
        ?eng :artistName ?engName.
    }
) t
GROUP BY t.engName HAVING COUNT(t.album) >= ALL (
        SELECT COUNT(t2.album)
        FROM
        sparqltable(SELECT ?eng ?album
          WHERE {(?album :AlbumContributor ?eng) :albumContributorType 'engineer'}
        ) t2 GROUP BY t2.eng)
```

La query viene eseguita in 10 secondi e restituisce 1 riga.