Università di Roma “Sapienza”

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*

Studenti:

Casciaro Mario *Palleschi Andrea*

Indice

[Capitolo 1] Analisi

[1.1] La base di dati

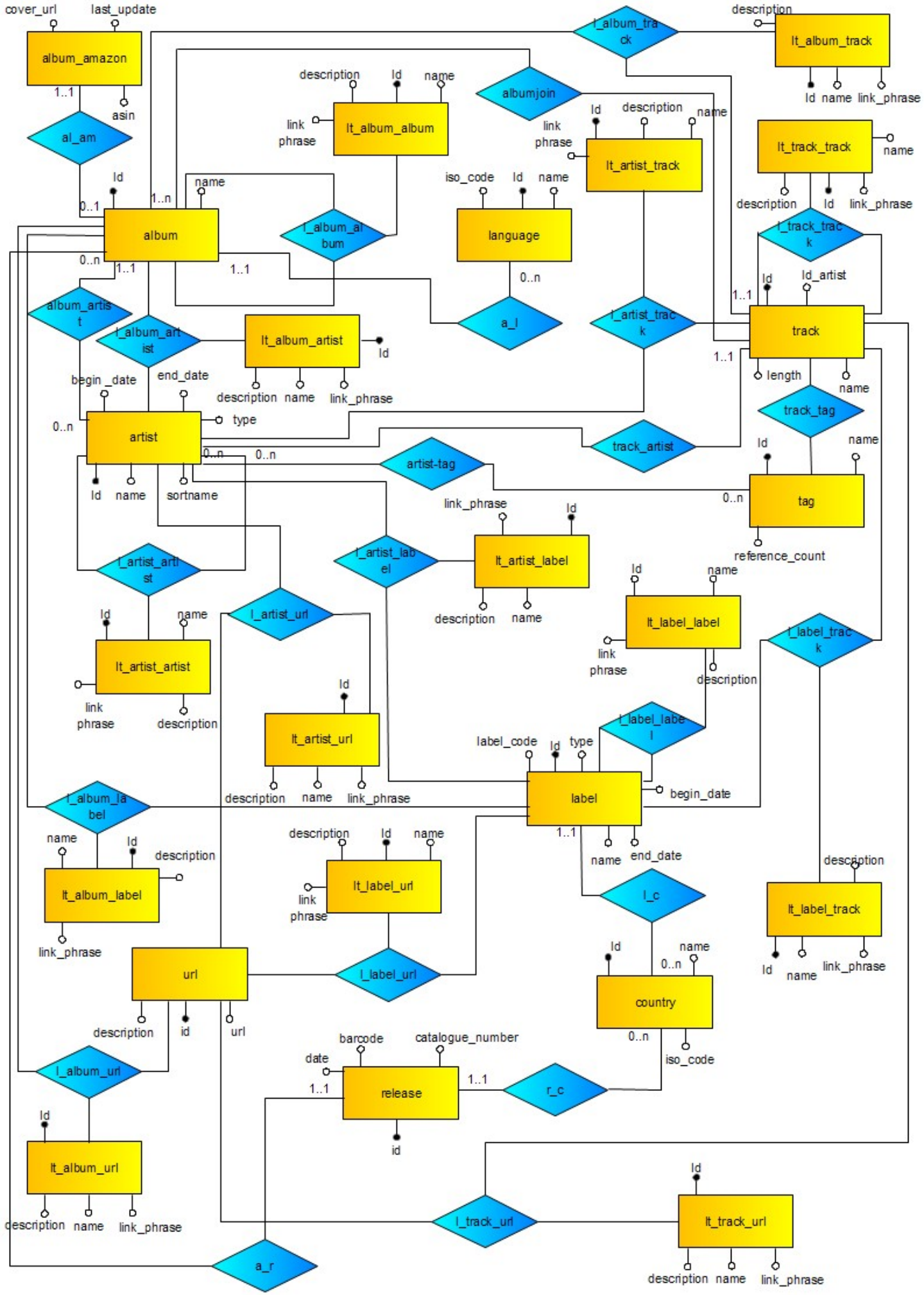
[1.1.1] Descrizione della base di dati

Come sorgente dati per il progetto, è stata utilizzata una base di dati open-source, disponibile sul sito web www.musicbrainz.org , contenente un' esauriente raccolta di informazioni riguardanti il mondo della musica. La base di dati contiene all'incirca 20 milioni di tuple, nella fattispecie, circa 450000 artisti, 650000 album, 7500000 canzoni e circa 1500000 indirizzi web, per una dimensione totale vicina ai 3 gigabyte. La base di dati originariamente in formato PostgreSQL è stata convertita prima in formato MySql e in seguito per il DBMS Oracle XE(per i dettagli riguardanti le motivazioni dell'utilizzo di differenti DBMS, si rimanda al capitolo 4 paragrafo 2, riguardante l'analisi delle prestazioni). Tale base di dati, considerate le dimensioni, è risultata molto adatta per analizzare le prestazioni del sistema Mastro/QuOnto.

[1.1.2] Reverse-Engineering della base di dati

La base di dati contiene informazioni riguardanti: i vari artisti con i relativi album , le case discografiche con le quali hanno un contratto e responsabili della pubblicazione dei loro lavori, la data di pubblicazione degli album con la relativa nazione, e le tracce in essi presenti. Sono presenti inoltre, informazioni sui link dove è possibile trovare ulteriori informazioni relative ad artisti, album e case discografiche e i tag attraverso cui sono catalogati.

Di seguito viene riportato il diagramma ER della base di dati ottenuto a partire dalla base di dati stessa, attraverso un processo di reverse-engineering.


cover_url
last_update
album_amazon
1..1
asin
al_am
album
0..1
1..n
0..n
1..1
1..1
id
name
description
id
name
link phrase
lt_album_album
l_album_album
albumjoin
l_album_track
description
lt_album_track
id
name
link_phrase
id
description
name
link phrase
lt_artist_track
lt_track_track
name
description
id
link_phrase
l_track_track
iso_code
id
name
language
0..n
a_l
album_artist
l_album_artist
lt_album_artist
id
description
name
link_phrase
begin_date
end_date
0..n
artist
type
0..n
0..n
id
name
sortname
l_artist_track
track
1..1
1..1
id
id_artist
length
name
track_tag
track_artist
artist-tag
id
name
0..n
tag
reference_count
link_phrase
id
l_artist_label
lt_artist_label
description
name
l_artist_artist
l_artist_url
id
name
lt_artist_artist
link phrase
description
id
lt_artist_url
description
name
link_phrase
id
name
lt_label_label
link phrase
description
l_label_track
l_label_label
label_code
id
type
label
begin_date
1..1
name
end_date
l_album_label
name
id
description
lt_album_label
link_phrase
description
id
name
lt_label_url
link phrase
l_label_url
l_c
description
lt_label_track
id
name
link_phrase
id
name
0..n
country
0..n
iso_code
url
description
id
url
l_album_url
barcode
catalogue_number
date
1..1
release
1..1
r_c
id
id
lt_album_url
description
name
link_phrase
l_track_url
id
lt_track_url
description
name
link_phrase
a_r

[1.1.3] Specifiche dell'applicazione

Si vuole realizzare un sistema che permetta all'utente di ottenere informazioni sul mondo della musica.

In particolare, occorre gestire informazioni su:

- Gli **album** con il *nome,*il *numero di tracce,* le **canzoni**, la **lingua**, i vari **artisti** che hanno contribuito alla sua realizzazione (con il **tipo di contributo** prestato), le **case discografiche** coinvolte nella pubblicazione dell'album stesso, una serie di **url** che rimandano a varie informazioni, riguardanti l'album (con la relativa **tipologia di informazione**), le varie **release** con cui l'album è stato pubblicato, e gli eventuali **album correlati** (con il **tipo di relazione**). Interessano, inoltre, i cosidetti **album amazon**, che rappresentano gli album distribuiti su amazon, con l'*url della cover* presente su amazon, a *l'asin* ovvero l'identificativo dell'album su amazon.
- Di ogni **lingua** interessa, il *nome* e il *codice iso*.
- Di ogni **canzone** interessa, *il nome*, la *durata*, una serie di **url** che rimandano a varie risorse correlate alla traccia (con relativo **tipo di risorsa**), un insieme di **tag**, l'**artista** che ha contribuito alla realizzazione della canzone (con il **tipo di contributo** dato), le altre **canzoni correlate** (con il **tipo di correlazione**).
- Di ogni **artista** interessa, il *nome*, la *data di inizio* della carriera, la *data di fine,* la *tipologia* (se persona o gruppo), il *nome di indicizzazione,* la **casa discografica** per cui lavora (con il **tipo di lavoro**), un insieme di **url** che rimandano a varie risorse sull'artista (con il relativo **tipo di risorsa**), un insieme di **tag**, e le varie **relazioni con altri artisti** (con il **tipo di relazione**).
- Di ogni **casa discografica** interessa, il *codice*, il *nome*, il *tipo*, la **nazione** di appartenenza, un insieme di **url** che rimandano a varie risorse sulla casa discografica (con il relativo **tipo di risorsa**), e le altre **case discografiche correlate** (con il **tipo di correlazione**).
- Di ogni **url** interessa, l'*indirizzo web*.
- Di ogni **release** interessa, il *barcode*, la *data*, il *numero di catalogo*, la **nazione** in cui ha avuto luogo.
- Di ogni **tag** interessa il *nome*, e il *conteggio dei riferimenti*.
- Di ogni **nazione** interessa il *nome*, e *il codice iso*.

Il sistema, inoltre, dovrebbe rendere possibili le seguenti operazioni:

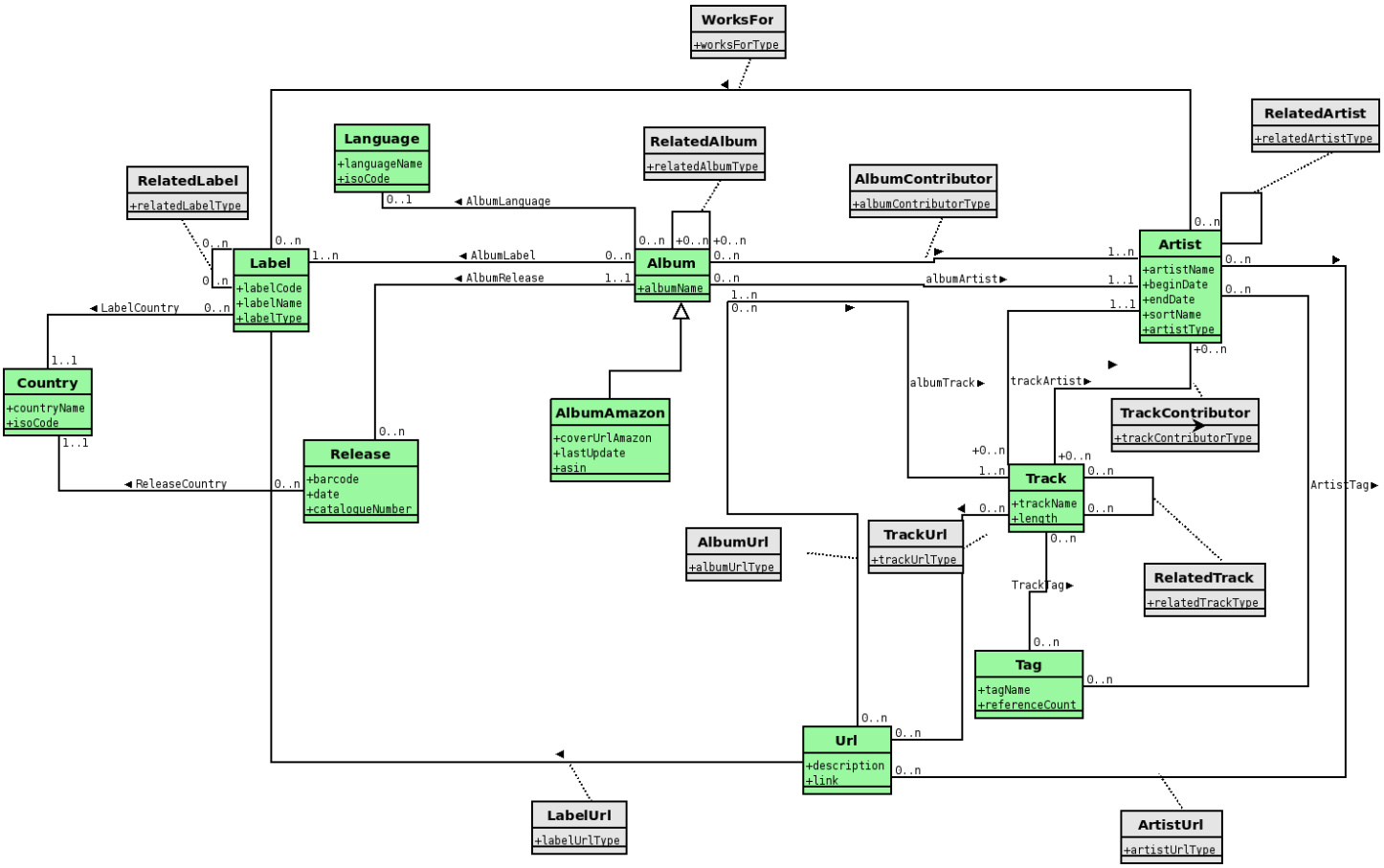
- Ricerca di album per nome, artista, casa discografica, data di pubblicazione
- Ricerca di artisti per nome, tag, casa discografica, data di inizio della carriera
- Ricerca di canzoni per nome, artista, tag, album
- Visualizzazione di tutte le informazioni di un album

- Visualizzazione di tutte le informazioni di un artista
- Visualizzazione di tutte le informazioni di una canzone

[1.1.4] Diagramma UML

Viene qui presentato il diagramma UML delle classi che corrisponde alla modellazione concettuale delle specifiche. Tale diagramma, è di fatto la rappresentazione grafica dell'ontologia che descrive l'applicazione.

E' possibile notare una differenza sostanziale tra il diagramma ER e l'ontologia qui descritta, in quanto tutte le relazioni ternarie presenti nella base di dati, sono state convertite in associazioni binarie con attributi, tale divergenza, che viene risolta attraverso l' operazione di mapping dell'ontologia sul database, fa si che l'ontologia non sia una copia uno a uno della base di dati.



[Capitolo 2] Progettazione e Sviluppo

[2.1] Implementazione dell'ontologia

[2.1.1] TBox in DL-LiteA

Di seguito viene riportata la descrizione dell'ontologia in sintassi DL-Lite A.

Alphabet (Concepts):

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

Alphabet (Roles):

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

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
```

Alphabet (Role attributes):

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

Concept inclusion assertions:

```
 Album ⊆ ¬Language
 Album ⊆ ¬Label
 Album ⊆ ¬Country
 Album ⊆ ¬Release
 Album ⊆ ¬Url
 Album ⊆ ¬Tag
 Album ⊆ ¬Track
Album  ⊆ ¬Artist
Language  ⊆ ¬Label
Language ⊆ ¬Country
Language ⊆ ¬Release
Language ⊆ ¬Url
Language ⊆ ¬Tag
Language ⊆ ¬Track
Language ⊆ ¬Artist
Country ⊆ ¬Release
```

```
Country ⊆ ¬Url
Country ⊆ ¬Tag
Country ⊆ ¬Track
Country ⊆ ¬Artist
Release ⊆ ¬Url
Release ⊆ ¬Tag
Release ⊆ ¬Track
Release ⊆ ¬Artist
Url ⊆ ¬Tag
Url ⊆ ¬Track
Url ⊆ ¬Artist
Tag ⊆ ¬Track
Tag ⊆ ¬Artist
Track  ⊆ ¬Artist
AlbumAmazon ⊆  Album
```

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)
```

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)
```

Concept attribute domain:

```
δ(albumName) ⊆ Album
Album ⊆ δ(albumName)
δ(coverUrlAmazon) ⊆ AlbumAmazon
AlbumAmazon ⊆ δ(coverUrlAmazon)
δ(lastUpdate) ⊆ AlbumAmazon
AlbumAmazon ⊆ δ(lastUpdate)
δ(asin) ⊆ AlbumAmazon
AlbumAmazon ⊆ δ(asin)
δ(languageName) ⊆ Language
Language ⊆ δ(languageName)
δ(languageIsoCode) ⊆ Language
Language ⊆ δ(languageIsoCode)
δ(labelName) ⊆ Label
Label ⊆ δ(labelName)
δ(labelCode) ⊆ Label
Label ⊆ δ(labelCode)
δ(labelType) ⊆ Label
Label ⊆ δ(labelType)
δ(artistName) ⊆ Artist
Artist ⊆ δ(artistName)
δ(beginDate) ⊆ Artist
Artist ⊆ δ(beginDate)
δ(endDate) ⊆ Artist
Artist ⊆ δ(endDate)
δ(sortName) ⊆ Artist
Artist ⊆ δ(sortName)
δ(artistType) ⊆ Artist
Artist ⊆ δ(artistType)
δ(countryName) ⊆ Country
Country ⊆ δ(countryName)
δ(countryIsoCode) ⊆ Country
Country ⊆ δ(countryIsoCode)
δ(barcode) ⊆ Release
Release ⊆ δ(barcode)
δ(date) ⊆ Release
Release ⊆ δ(date)
δ(catalogueNumber) ⊆ Release
Release ⊆ δ(catalogueNumber)
δ(trackName) ⊆ Track
Track ⊆ δ(trackName)
δ(length) ⊆ Track
Track ⊆ δ(length)
δ(link) ⊆ Url
Url ⊆ δ(link)
δ(tagName) ⊆ Tag
Tag ⊆ δ(tagName)
δ(referenceCount) ⊆ Tag
Tag ⊆ δ(referenceCount)
```

Roles :

$$\exists\ \text{AlbumTrack} \subseteq \text{Album}$$
$$\exists\ \text{AlbumTrack}^- \subseteq \text{Track}$$
$$\exists\ \text{ArtistTag} \subseteq \text{Artist}$$
$$\exists\ \text{ArtistTag}^- \subseteq \text{Tag}$$
$$\exists\ \text{TrackTag} \subseteq \text{Track}$$
$$\exists\ \text{TrackTag}^- \subseteq \text{Tag}$$
$$\exists\ \text{ReleaseCountry} \subseteq \text{Release}$$
$$\exists\ \text{ReleaseCountry}^- \subseteq \text{Country}$$
$$\exists\ \text{AlbumRelease} \subseteq \text{Album}$$
$$\exists\ \text{AlbumRelease}^- \subseteq \text{Release}$$
$$\exists\ \text{AlbumArtist} \subseteq \text{Album}$$
$$\exists\ \text{AlbumArtist}^- \subseteq \text{Artist}$$
$$\exists\ \text{TrackArtist} \subseteq \text{Artist}$$
$$\exists\ \text{TrackArtist}^- \subseteq \text{Track}$$
$$\exists\ \text{AlbumLabel} \subseteq \text{Album}$$
$$\exists\ \text{AlbumLabel}^- \subseteq \text{Label}$$
$$\exists\ \text{LabelCountry} \subseteq \text{Label}$$
$$\exists\ \text{LabelCountry}^- \subseteq \text{Country}$$
$$\exists\ \text{AlbumLanguage} \subseteq \text{Album}$$
$$\exists\ \text{AlbumLanguage}^- \subseteq \text{Language}$$
$$\exists\ \text{WorksFor} \subseteq \text{Artist}$$
$$\exists\ \text{WorksFor}^- \subseteq \text{Label}$$
$$\exists\ \text{RelatedAlbum} \subseteq \text{Album}$$
$$\exists\ \text{RelatedAlbum}^- \subseteq \text{Album}$$
$$\exists\ \text{RelatedLabel} \subseteq \text{Label}$$
$$\exists\ \text{RelatedLabel}^- \subseteq \text{Label}$$
$$\exists\ \text{AlbumContributor} \subseteq \text{Album}$$
$$\exists\ \text{AlbumContributor}^- \subseteq \text{Artist}$$
$$\exists\ \text{RelatedArtist} \subseteq \text{Artist}$$
$$\exists\ \text{RelatedArtist}^- \subseteq \text{Artist}$$
$$\exists\ \text{TrackContributor} \subseteq \text{Track}$$
$$\exists\ \text{TrackContributor}^- \subseteq \text{Artist}$$
$$\exists\ \text{AlbumUrl} \subseteq \text{Album}$$
$$\exists\ \text{AlbumUrl}^- \subseteq \text{Url}$$
$$\exists\ \text{TrackUrl} \subseteq \text{Track}$$
$$\exists\ \text{TrackUrl}^- \subseteq \text{Url}$$
$$\exists\ \text{LabelUrl} \subseteq \text{Label}$$
$$\exists\ \text{LabelUrl}^- \subseteq \text{Url}$$
$$\exists\ \text{ArtistUrl} \subseteq \text{Artist}$$
$$\exists\ \text{ArtistUrl}^- \subseteq \text{Url}$$

Role attributes Domain:

$$\exists\ \delta(\text{albumContributorType}) \subseteq \text{Album}$$
$$\exists\ \delta(\text{albumContributorType})^- \subseteq \text{Artist}$$

```
∃ δ(worksForType) ⊆ Artist
∃ δ(worksForType)⁻ ⊆ Label
∃ δ(relatedAlbumType) ⊆ Album
∃ δ(relatedAlbumType)⁻ ⊆ Album
∃ δ(relatedArtistType) ⊆ Artist
∃ δ(relatedArtistType)⁻ ⊆ Artist
∃ δ(trackContributorType) ⊆ Track
∃ δ(trackContributorType)⁻ ⊆ Artist
∃ δ(albumUrlType) ⊆ Album
∃ δ(albumUrlType)⁻ ⊆ Url
∃ δ(trackUrlType) ⊆ Track
∃ δ(trackUrlType)⁻ ⊆ Url
∃ δ(labelUrlType) ⊆ Label
∃ δ(labelUrlType)⁻ ⊆ Url
∃ δ(artistUrlType) ⊆ Artist
∃ δ(artistUrlType)⁻ ⊆ Url
```

Role functionality assertions:

```
(funct AlbumRelease⁻)
(funct AlbumArtist)
(funct TrackArtist)
(funct ReleaseCountry)
(funct LabelCountry)
```

Role attribute domain:

```
δ(relatedLabelType) ⊆ RelatedLabel
δ(worksForType) ⊆ WorksFor
δ(relatedAlbumType) ⊆ RelatedAlbum
δ(albumContributorType) ⊆ AlbumContributor
δ(relatedArtistType) ⊆ RelatedArtist
δ(labelUrlType) ⊆ LabelUrl
δ(albumUrlType) ⊆ AlbumUrl
δ(trackUrlType) ⊆ TrackUrl
δ(artistUrlType) ⊆ ArtistUrl
δ(relatedTrackType) ⊆ RelatedTrack
δ(trackContributorType) ⊆ TrackContributor
```

Concept partecipation in role:

```
Album ⊆ ∃ AlbumTrack
Track ⊆ ∃ AlbumTrack⁻
```

Concept and role attribute value range:

```
ρ(languageName) ⊆  xsd:string
ρ(isoCode) ⊆ xsd:string
ρ(albumName) ⊆ xsd:string
```

```
ρ(trackCount) ⊆ xsd:int
ρ(labelName) ⊆ xsd:string
ρ(labelCode) ⊆ xsd:string
ρ(labelType) ⊆ xsd:int
ρ(countryName) ⊆ xsd:string
ρ(isoCode) ⊆ xsd:string
ρ(barcode) ⊆ xsd:string
ρ(date) ⊆  xsd:date
ρ(catalogueNumber) ⊆ xsd:string
ρ(coverUrlAmazon) ⊆ xsd:string
ρ(lastUpdate) ⊆  xsd:date
ρ(asin) ⊆ xsd:string
ρ(description) ⊆ xsd:string
ρ(link) ⊆ xsd:string
ρ(tagName) ⊆ xsd:string
ρ(referenceCount) ⊆ xsd:int
ρ(trackName) ⊆ xsd:string
ρ(length) ⊆ xsd:int
ρ(artistName) ⊆ xsd:string
ρ(beginDate) ⊆  xsd:date
ρ(endDate) ⊆ xsd:date
ρ(sortName) ⊆ xsd:string
ρ(artistType) ⊆ xsd:int
ρ(relatedLabelType) ⊆ xsd:string
ρ(worksForType) ⊆ xsd:string
ρ(relatedAlbumType) ⊆ xsd:string
ρ(albumContributorType) ⊆ xsd:string
ρ(relatedArtistType) ⊆ xsd:string
ρ(trackContributorType) ⊆ xsd:string
ρ(trackUrlType) ⊆ xsd:string
ρ(albumUrlType) ⊆ xsd:string
ρ(relatedTrackType) ⊆ xsd:string
ρ(labelUrlType) ⊆ xsd:string
ρ(artistUrlType) ⊆ xsd:string
```

Role inclusion assertion:

```
WorksFor ⊆ ¬RelatedLabel
WorksFor ⊆ ¬RelatedAlbum
WorksFor ⊆ ¬AlbumContributor
WorksFor ⊆ ¬RelatedArtist
WorksFor ⊆ ¬TrackContributor
WorksFor ⊆ ¬AlbumUrl
WorksFor ⊆ ¬TrackUrl
WorksFor ⊆ ¬RelatedTrack
WorksFor ⊆ ¬LabelUrl
WorksFor ⊆ ¬ArtistlUrl
RelatedLabel ⊆ ¬RelatedAlbum
RelatedLabel ⊆ ¬AlbumContributor
RelatedLabel ⊆ ¬RelatedArtist
RelatedLabel ⊆ ¬TrackContributor
RelatedLabel ⊆ ¬AlbumUrl
```

```
RelatedLabel ⊆ ¬TrackUrl
RelatedLabel ⊆ ¬RelatedTrack
RelatedLabel ⊆ ¬LabelUrl
RelatedLabel ⊆ ¬ArtistUrl
RelatedAlbum ⊆ ¬AlbumContributor
RelatedAlbum ⊆ ¬RelatedArtist
RelatedAlbum ⊆ ¬TrackContributor
RelatedAlbum ⊆ ¬AlbumUrl
RelatedAlbum ⊆ ¬TrackUrl
RelatedAlbum ⊆ ¬RelatedTrack
RelatedAlbum ⊆ ¬LabelUrl
RelatedAlbum ⊆ ¬ArtistUrl
AlbumContributor ⊆ ¬RelatedArtist
AlbumContributor ⊆ ¬TrackContributor
AlbumContributor ⊆ ¬AlbumUrl
AlbumContributor ⊆ ¬TrackUrl
AlbumContributor ⊆ ¬RelatedTrack
AlbumContributor ⊆ ¬LabelUrl
AlbumContributor ⊆ ¬ArtistUrl
RelatedArtist ⊆ ¬TrackContributor
RelatedArtist ⊆ ¬AlbumUrl
RelatedArtist ⊆ ¬TrackUrl
RelatedArtist ⊆ ¬RelatedTrack
RelatedArtist ⊆ ¬LabelUrl
RelatedArtist ⊆ ¬ArtistUrl
TrackContributor ⊆ ¬AlbumUrl
TrackContributor ⊆ ¬TrackUrl
TrackContributor ⊆ ¬AlbumUrl
TrackContributor ⊆ ¬RelatedTrack
TrackContributor ⊆ ¬LabelUrl
TrackContributor ⊆ ¬ArtistUrl
AlbumUrl ⊆ ¬TrackUrl
AlbumUrl ⊆ ¬RelatedTrack
AlbumUrl ⊆ ¬LabelUrl
AlbumUrl ⊆ ¬ ArtistUrl
TrackUrl ⊆ ¬RelatedTrack
TrackUrl ⊆ ¬LabelUrl
TrackUrl  ⊆ ¬ArtistUrl
RelatedTrack ⊆ ¬LabelUrl
RelatedTrack ⊆ ¬ArtistUrl
LabelUrl ⊆ ¬ArtistUrl
```

[2.1.2] ABox GAV Mapping

Di seguito viene riportato il mapping di tipo GAV tra l'ontologia e la base di dati.

```
SELECT album.id as album_id, album.name as          Album(album(album_id)),
album_name, album.language as album_language,  ~>   albumName(album(album_id),album_name:xsd:strin
```

```
album.artist as album_artist
FROM album
```

```
g),
albumLanguage(album(album_id),language(album_l
anguage)),
albumArtist(album(album_id),artist(album_artis
t))
```

```
SELECT album_amazon_asin.album as
album_amazon_asin_album,
album_amazon_asin.coverarturl as
album_amazon_asin_coverarturl,
album_amazon_asin.lastupdate as
album_amazon_asin_lastupdate,
album_amazon_asin.asin as
album_amazon_asin_asin
FROM album_amazon_asin
```

```
~>
```

```
AlbumAmazon(album(album_amazon_asin_album)),
asin(album(album_amazon_asin_album),album_amaz
on_asin_asin:xsd:string),
coverUrlAmazon(album(album_amazon_asin_album),
album_amazon_asin_coverarturl:xsd:string),
lastUpdate(album(album_amazon_asin_album),albu
m_amazon_asin_lastupdate: xsd:date),
```

```
SELECT language.id as language_id,
language.name as language_name,
language.isocode_3t as  language_isocode_3t
FROM language
```

```
~>
```

```
Language(language(language_id)),
languageName(language(language_id),language_na
me: xsd:string),
languageIsoCode(language(language_id),language
_isocode_3t: xsd:string)
```

```
SELECT label.id as label_id,
label.name as label_name,
label.labelcode as label_labelcode,
label.type as label_type,
label.country as label_country
FROM label
```

```
~>
```

```
Label(label(label_id)),
labelName(label(label_id),label_name:
xsd:string),
labelCode(label(label_id),label_labelcode:
xsd:string),
labelType(label(label_id),label_type:
xsd:string)
LabelCountry(label(label_id),country(labelCoun
try))
```

```
SELECT country.id as country_id,
country.name as country_name,
country.isocode as country_isocode
FROM country
```

```
~>
```

```
Country(country(country_id)),
countryName(country(country_id), countryName:
xsd:string),
countryIsoCode(country(country_id),countryIsoC
ode: xsd:string)
```

```
SELECT releases.id as release_id,
releases.barcode as release_barcode,
releases.releasedate as release_releasedate,
releases.catno as release_catno,
releases.country as release_country,
releases.album as release_album
FROM releases
```

```
~>
```

```
Release(release(release_id)),
barcode(release(release_id), release_barcode:
xsd:string),
date(release(release_id), release_releasedate:
xsd:date),
catalogueNumber(release(release_id),
release_catno: xsd:string),
ReleaseCountry(release(release_id),
country(release_country)),
AlbumRelease(album(release_album),
release(release_id))
```

```
SELECT url.id as url_id,
url.url as url_url
```

```
~>
```

```
Url(url(url_id)),
link(url(url_id), url_url: xsd:string)
```

```
FROM url
```

```
SELECT tag.id as tag_id,                          Tag(tag(tag_id)),
tag.name as tag_name,                      ~>     tagName(tag(tag_id), tag_name: xsd:string)
tag.refcount as tag_refcount                      referenceCount(tag(tag_id), tag_refcount:
 FROM tag                                         xsd:string)
```

```
SELECT track.id as track_id,                      Track(track(track_id)),
track.name as track_name,                         trackName(track(track_id), trackName:
track.length  as track_length,             ~>     xsd:string),
track.artist as track_artist                      length(track(track_id), track_length:
FROM track                                        xsd:int),
                                                  TrackArtist(track(track_id),
                                                  artist(track_artist))
```

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

```
SELECT l_artist_label.link0 as link0,             WorksFor(artist(link0), label(link1)),
l_artist_label.link1 as link1,             ~>     worksForType(artist(link0), label(link1),
lt_artist_label.name as name                      name: xsd:string)
FROM l_artist_label, lt_artist_label WHERE
lt_artist_label.id = l_artist_label.link_type
```

```
SELECT l_album_album.link0 as link0,              RelatedAlbum(album(link0),album(link1)),
l_album_album.link1 as link1,              ~>     relatedAlbumType(album(link0), album(link1),
lt_album_album.name as name                       name: xsd:string)
FROM l_album_album, lt_album_album WHERE
lt_album_album.id = l_album_album.link_type
```

```
SELECT l_label_label.link0 as link0,              RelatedLabel(label(link0), label(link1)),
l_label_label.link1 as link1,              ~>     relatedLabelType(label(link0), label(link1),
lt_label_label.name as name                       name: xsd:string)
FROM l_label_label, lt_label_label WHERE
l_label_label.link_type = lt_label_label.id
```

```
SELECT l_album_artist.link0 as link0,             AlbumContributor(album(link0), artist(link1)),
l_album_artist.link1 as link1,             ~>     albumContributorType(album(link0),
lt_album_artist.name as name                      artist(link1), name: xsd:string)
FROM l_album_artist, lt_album_artist WHERE
l_album_artist.link_type = lt_album_artist.id
```

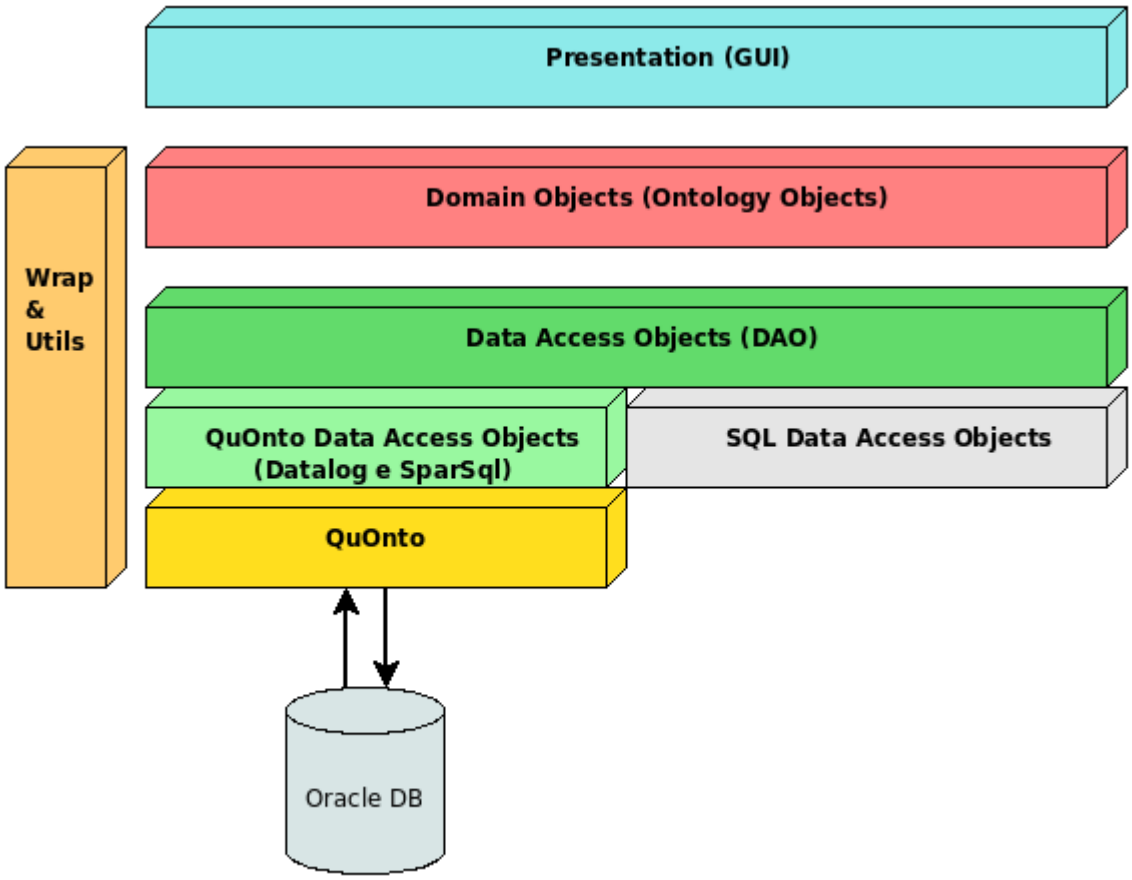
```
SELECT l_artist_artist.link0 as link0,                RelatedArtist(artist(link0), artist(link1)),
l_artist_artist.link1 as link1,                ~>     relatedArtistType(artist(link0),
lt_artist_artist.name as name                         artist(link1), name: xsd:string)
FROM l_artist_artist, lt_artist_artist WHERE
l_artist_artist.link_type =lt_artist_artist.id


SELECT l_artist_track.link0 as link0,                 TrackContributor(track(link1), artist(link1)),
l_artist_track.link1 as link1,                 ~>     trackContributorType(track(link1),
lt_artist_track.name as name                          artist(link0), name: xsd:string)
FROM l_artist_track, lt_artist_track WHERE
l_artist_track.link_type = lt_artist_track.id


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


SELECT l_track_url.link0 as link0,                    TrackUrl(track(link0), url(link1)),
l_track_url.link1 as link1,                    ~>     trackUrkType(track(link0), url(link1), name:
lt_track_url.name as name                             xsd:string)
FROM l_track_url, lt_track_url  WHERE
l_track_url.link_type = lt_track_url.id


SELECT l_label_url.link0 as link0,                    LabelUrl(label(link0), url(link1)),
l_label_url.link1 as link1,                    ~>     labelUrlType(label(link0), url(link1), name:
lt_label_url.name as name                             xsd:string)
FROM l_label_url, lt_label_url  WHERE
l_label_url.link_type = lt_label_url.id


SELECT l_artist_url.link0 as link0,                   ArtistUrl(artist(link0), url(link1)),
l_artist_url.link1 as link1,                   ~>     artistUrlType(artist(link0), url(link1), name:
lt_artist_url.name as name                            xsd:string)
FROM l_artist_url, lt_artist_url WHERE
l_artist_url.link_type = lt_artist_url.id


SELECT albumjoin.album as album,               ~>     AlbumTrack(album(album), track(track))
albumjoin.track as track
FROM albumjoin


SELECT artist_tag.artist as artist,            ~>     ArtistTag(artist(artist), tag(tag))
artist_tag.tag as tag
FROM artist_tag


SELECT track_tag.track as track, track_tag.tag ~>     TrackTag(track(track), tag(tag))
as tag
FROM track_tag
```

```
SELECT l_album_label.link0 as link0,          ~>   AlbumLabel(album(link0), label(link1))
l_album_label.link1 as link1
FROM l_album_label
```

[2.2] Progettazione dell'applicazione

[2.2.1] Struttura del sistema

Durante la progettazione strutturale, sono stati tenuti in forte considerazione i fattori modularità e riuso. La prima scelta, è stata quella di organizzare il sistema nei **3 livelli base di astrazione: presentazione, logica, accesso ai dati**. Tale organizzazione, ormai diventata uno standard in questo tipo di sistemi, fornisce già di per se un grado di modularità molto alto, grazie alla separazione dei tre aspetti fondamentali dell'applicazione. Tuttavia per aumentare le qualità strutturali del sistema finale, sono stati adottati alcuni design patterns, come verrà spiegato nel dettaglio in seguito.



Lo schema sopra riportato, descrive, a grandi linee, la struttura generale del sistema MusicOn. Si nota chiaramente la suddivisione dei tre livelli di astrazione. In particolare, al livello più alto, abbiamo il livello di presentazione implementato sotto forma di GUI, che a sua volta usufruisce dei servizi messi a disposizione dal livello sottostante, ovvero quello della logica applicativa. In questo livello, solitamente, vivono i cosiddetti “Domain Objects” ovvero istanze delle classi che modellano il dominio applicativo, e che hanno il compito di gestire la logica correlata a tale dominio. Al livello più basso troviamo lo strato di accesso ai dati, utilizzato dai domain object come un tramite per accedere alle informazioni.

Qui risiedono di Data Access Objects, ovvero degli oggetti che riflettono la struttura dei dati presenti al livello più basso, grazie ai quali è possibile accedere in modo trasparente, unificato e semplice alle informazioni immagazzinate dai vari sistemi di gestione dei dati. Nello schema notiamo anche la presenza di un modulo “Wrap & Utils”. Tale modulo, verticale rispetto alla gerarchia dell'applicazione, contiene classi utili a semplificare e modularizzare alcune operazioni e l'accesso a servizi messi a disposizione dal sistema MusicOn.

Ora verranno descritti nel dettaglio i vari moduli dell'applicazione.

[2.2.2] I Domain Objects

Come si è già detto, i Domain Objects, sono il cuore dell'applicazione, essendo i responsabili della modellazione e della gestione della logica che sta dietro all'intero sistema. A questo punto, è utile far notare che **la struttura dei domain objects riflette la TBox dell'ontologia** utilizzata per modellare il dominio applicativo. Questo, nella pratica, si traduce nel fatto che avremo una corrispondenza 1 a 1, tra la struttura dell'ontologia e la struttura dei Domain Objects. L'unica differenza è che, se l'ontologia può essere considerata come un insieme di definizioni, asserzioni e vincoli “passivi”, i Domain Object rappresentano degli elementi “attivi” che è possibile manipolare come se fossero degli oggetti del mondo reale, creandoli, distruggendoli, modificandoli, o invocando su di essi operazioni.

Dal punto di vista implementativo, considerato che ogni Domain Object dovrà riflettere il corrispondente concetto o ruolo presente nell'ontologia, la classe corrispondente:

- Avrà lo stesso nome del concetto o del ruolo presente nella TBox.
- Avrà una variabile d'istanza che conterrà l'id dell'oggetto. In particolare tale id corrisponderà all'identificativo dell'oggetto generato da QuOnto mediante l'utilizzo dell'apposita funzione di skolem specificata nel mapping.
- Avrà una variabile d'istanza per ogni attributo del concetto o del ruolo corrispondente.
- Se modella un concetto, avrà una variabile d'istanza per ogni ruolo in cui tale concetto partecipa. In particolare:
    - Se il ruolo corrispondente ha degli attributi, si avrà come variabile d'istanza una classe(molteplicità massima 1) o una lista di classi(molteplicità massima N) corrispondenti a tale ruolo con attributi.
    - Se il ruolo corrispondente non ha attributi, si avrà come variabile d'istanza una classe(molteplicità massima 1) o una lista di classi(molteplicità massima N) che corrispondono al concetto presente dall'altro lato dell'associazione.
- Per ogni oggetto collegato al Domain Object, che non corrisponda ad un attributo, avrà una variabile booleana che varrà “true” se e solo se il corrispondente oggetto è stato caricato dal Database. Questa variabile è utilizzata in MusicOn per gestire il meccanismo di lazy read.

Per chiarire quanto detto, qui di seguito riportiamo come esempio la classe Album che modella l'omonimo concetto dell'ontologia.

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

	protected Artist artist;
	protected boolean artistLoaded;

	protected List<RelatedAlbum> relatedAlbums;
	protected boolean relatedAlbumsLoaded;

	protected Language language;
	protected boolean languageLoaded;

	protected List<Label> labels;
	protected boolean labelsLoaded;

	protected List<Release> releases;
	protected boolean releasesLoaded;

	protected List<AlbumUrl> urls;
	protected boolean urlsLoaded;

	protected List<Track> tracks;
	protected boolean tracksLoaded;

	protected List<AlbumContributor> contributors;
	protected boolean contributorsLoaded;

	protected AlbumAmazon albumAmazon;
	protected boolean albumAmazonLoaded;

	[...]
}
```

Ogni Domain Object utilizzato in MusiOn implementa il meccanismo del **lazy read**. Questo permette di poter caricare a richiesta, e in modo trasparente, gli oggetti collegati ad un Domain Object. In particolare, ogni volta che viene invocato un metodo “getter”, viene verificato che l'oggetto richiesto sia già stato caricato dal Database, in caso contrario si esegue una chiamata al DAO per “riempire” la variabile d'istanza corrispondente. Di seguito si riporta come esempio il codice di uno di tali metodi:

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

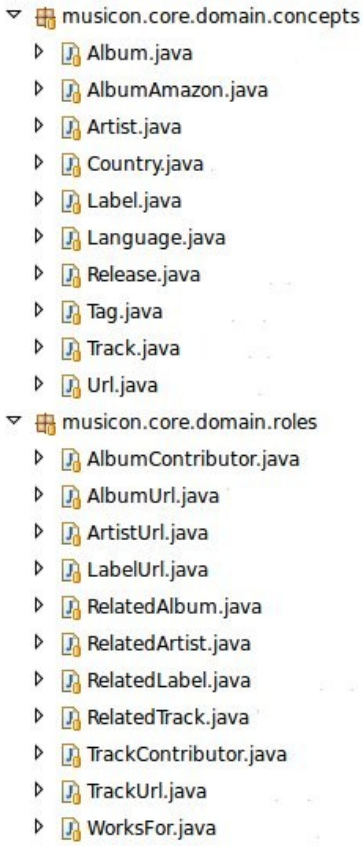
Quindi, uno dei compiti di questo strato applicativo è quello di fornire al livello superiore i meccanismi per “estrarre” dal DB i domain object. Queste operazioni vengono fatte in modo trasparente per l'utente, e l'utilizzo stesso dei DAO viene nascosto e incapsulato in questo strato applicativo. Abbiamo già visto come, a partire da un Domain Object, possiamo ottenere gli oggetti collegati. Invece, per ottenere Domain Object “da zero”, vengono utilizzati degli appositi metodi statici, uno per ogni operazione fondamentale richiesta nelle specifiche dell'applicazione. Ad esempio all'operazione “ricerca di album” corrisponderà un metodo statico nella classe *Album* del tipo

```
public static List<Album> findAlbums(String albumName, String albumArtist,
```

```
                String albumLabel, String startReleaseDate, String endReleaseDate) throws MusicOnException
{
        List<Album> albums = MusicOnManager.getInstance().getDAOFactory().getAlbumDAO().
                        findAlbums(albumName,albumArtist,albumLabel, startReleaseDate, endReleaseDate);
        return albums;
}
```

che come si nota andrà ad invocare il corrispondente metodo nei DAO.

Di seguito si riporta la struttura del package Java contenente di Domain Objects.



[2.2.3] I Data Access Objects

I Data Access Objects, implementano lo strato applicativo di accesso ai dati. In MusicOn i DAO utilizzano il sistema QuOnto per interfacciarsi con le sorgenti, tuttavia grazie all'utilizzo del design patter “Abstract Factory” è possibile sostituire il modulo di accesso ai dati in modo del tutto trasparente per il resto dell'applicazione. In questo strato applicativo, avremo quindi un insieme di DAO astratti, e un insieme di DAO “concreti” (nel nostro caso quelli che utilizzano QuOnto). Tramite la classe *MusiconOnManager* è possibile specificare, in modo globale, l'implementazione del *DAOFactory*, che è l'oggetto incaricato di creare i DAO concreti, e che verrà utilizzato all'interno di tutta l'applicazione. Riportiamo il codice di inizializzazione della classe *MusicOnManager*:

```
public void initialize(String accessMethod) throws MusicOnException
```

```
{
	this.dataAccessMethod = accessMethod;

	[...]
	if(dataAccessMethod.equals("QuontoCQ"))
	{
		daoFactory = new QuontoDAOFactory();
		QuontoManager.getInstance().initialize();
	}
	else
		throw new MusicOnException("Invalid data access method selected in configuration params");
	[...]
}
```

Tramite l'utilizzo dell'"Abstract Factory", dunque, è possibile fornire a MusicOn la capacità di utilizzare altri medoti di accesso ai dati (ad esempio SQL, XPath + Xml), anche se nel nostro caso è stato implementato solo l'accesso tramite QuOnto.

I DAO, solitamente riflettono la struttura logica dei dati sottostanti, quindi nel classico caso di accesso a un database relazionale, avremo un DAO per ogni tabella. Tuttavia, nel nostro caso, la struttura logica dei dati a cui accedere corrisponde con l'ontologia e quindi con lo schema concettuale del dominio applicativo! Si viene dunque ad eliminare il problema dell'"**impedence mismatch**", presente tra i DAO e i Domain Objects (ovvero tra struttura logica e struttura concettuale). Questo si traduce col fatto che avremo un DAO per ogni Domain Object, creando un sistema più semplice e chiaro, e meglio organizzato. Dunque, se vogliamo ottenere tutti gli album di un artista, basta fare riferimento all'oggetto *AlbumDAO*, in particolare al metodo *findByArtist*(...).

Entriamo ora nel dettaglio dei DAO basati su QuOnto. Ogni operazione di accesso ai dati si può scomporre nelle seguenti parti:

1. **Costruzione della query (Datalog o SparSql)**: viene generata la query da inviare al sistema QuOnto. Tale query può essere sia "statica" sia "dinamica", ovvero costruita in base ai parametri di input della funzione. Un semplice esempio di query del secondo tipo è presente nella funzione *QuontoTrackDAO.findTracks(...):*

```
public List<Track> findTracks(String trackName, String albumName,
		String artistName, String[] tags) throws MusicOnException
{
	[...]
	String query = "q(trackVar, trackNameVar, lengthVar, artistVar, artistNameVar," +
	"beginDateVar, endDateVar, sortNameVar, artistTypeVar) : TrackArtist(trackVar, artistVar)," +
	"trackName(trackVar, trackNameVar), length(trackVar, lengthVar), " +
	"artistName(artistVar, artistNameVar), beginDate(artistVar, beginDateVar)," +
	"endDate(artistVar, endDateVar), sortName(artistVar, sortNameVar)," +
	"artistType(artistVar, artistTypeVar)";

	if(trackName != null && trackName.trim().length() != 0)
		query += ", _equal(trackNameVar, '"+trackName+"')";

	if(albumName != null && albumName.trim().length() != 0)
		query += ", AlbumTrack(albumVar, trackVar), albumName(albumVar, '"+albumName+"')";

	if(artistName != null && artistName.trim().length() != 0)
		query += ", _equal(artistNameVar, '"+artistName+"')";


	String oldquery = query;
	List<String> queries = new ArrayList<String>();
	if(tags != null && tags.length != 0)
	{
		//qui dobbiamo costruire una union of conjunctive queries
```

```
                for(String tag: tags)
                {
                        if(tag.trim().length() == 0)
                                continue;
                        query = "" + oldquery;

                        query += ", tagName(tagVar, '"+ tag +"'), TrackTag(trackVar, tagVar)";
                        queries.add(query);
                }
        }
        else
                queries.add(oldquery);
        [...]
}
```

In MusicOn è stato utilizzato il formato Datalog per le query più semplici (90% del totale), mentre per le query più complesse è stato utilizzato SparSql.

2. **Valutazione della query**: Per eseguire le query, viene utilizzata un'apposita classe messa a disposizione da MusicOn, chiama *QuontoManager* che "wrappa" e arricchisce i servizi messi a disposizione da QuOnto. Tale strumento verrà discusso nei paragrafi successivi.

```
IEvaluationResult result = QuontoManager.getInstance().evaluateDatalogQueries(queries.toArray(new
String[queries.size()]));
```

3. **Processamento del risultato**: In questa fase, i dati restituiti dalla query vengono processati e "trasformati" in un Domain Object (o una lista di Domain Objects). Questa operazione è semplificata dall'utilizzo di un altro strumento messo a disposizione da MusicOn, ovvero la classe *ResultsetToObject*, che prende in input un mapping del tipo "numero colonna "-> "nome variabile", il result set e il Domain Object e in automatico estrae i valori dal result set e li assegna alle variabili di istanza del Domain Object (di questo strumento se ne parlerà in seguito)

```
ArrayList<Track> trackList = new ArrayList<Track>();
while (result.nextRow())
{
        Track track = ResultsetToObject.getInstance().convert(result,
                        ResultsetToObject.getInstance().track,
                        1, false, new Track());
        Artist artist = ResultsetToObject.getInstance().convert(result,
                        ResultsetToObject.getInstance().artist,
                        4, false, new Artist());
        track.setArtist(artist);
        track.setArtistLoaded(true);
        trackList.add(track);
}
```

[2.3] Dettagli implementativi

[2.3.1] Interfacciamento con QuOnto e meccanismi di wrapping

Tutto l'interfacciamento con le API del sistema QuOnto è stato incapsulato nella classe *QuontoManager*. In particolare tale classe contiene gli strumenti per:

- **Inizializzare QuOnto**, creando le strutture dati necessarie e specificando TBox e mapping.

```
public void initialize() throws MusicOnException
{
	[...]
	domainFactory = new DomainFactory();
	tbox = new TBoxXMLParser(tboxFile).getTBox();
	abox = GAVMappingABoxXMLParser.getABox(mappingFile, domainFactory, tbox);
	ontology = domainFactory.getNewOntology(tbox, abox);
	quonto = new QuontoCoreInstance(ontology);
	[...]
}
```

- **Invocare query espresse in vari formati,** fornendo meccanismi integrati di logging e gestione degli errori.

```
public IEvaluationResult evaluateDatalogQueries(String queries[]) throws MusicOnException
public IEvaluationResult evaluateDatalogQuery(String query) throws MusicOnException
public IEvaluationResult evaluateSparSqlQuery(String query) throws MusicOnException
public IEvaluationResult evaluateConjunctiveQuery(IConjunctiveQuery query) throws MusicOnException
public IEvaluationResult evaluateUnionOfConjunctiveQueries(IUnionOfConjunctiveQueries ucq) throws
MusicOnException
```

  In queste funzioni inoltre è stato inserito un "workaround" ad un bug di QuOnto, a causa del quale il database lancia un errore di "*maximum open cursors exceeded*" alla query numero 300. A quanto pare è risultato necessario chiamare la funzione

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

  prima dell'esecuzione della query numero 300 per chiudere gli statements sql evidentemente non chiusi in automatico da QuOnto.

[2.3.2] Strumenti di supporto

Per semplificare le operazioni più ricorrenti sono stati implementati alcuni strumenti di supporto. Il più importare è rappresentato dalla classe *ResultsetToObject* che si occupa di trasformare automaticamente un result set ottenuto dalla valutazione di una query (Datalog o SparSql) in un Domain Object. Tale conversione avviene in automatico, sfruttando un mapping definito dall'utente del tipo "variabile d'istanza" → "numero colonna". Tale mapping viene dichiarato come variabile all'interno della classe *ResultsetToObject*, e inizializzata nel suo costruttore, ad esempio:

```
public class ResultsetToObject
{
	[...]
	public final HashMap<String, Integer>  artist = new HashMap<String, Integer>();
	private ResultsetToObject()
	{
		[...]
		i = 0;
		artist.put("objId", new Integer(i++));
		artist.put("name", new Integer(i++));
		artist.put("beginDate", new Integer(i++));
		artist.put("endDate", new Integer(i++));
		artist.put("sortName", new Integer(i++));
		artist.put("type", new Integer(i++));
		[...]
```

```
        }

        [...]
}
```

In questo esempio viene imposto che alla variabile d'istanza chiamata *objId* venga assegnato il valore contenuto nella prima colonna del result set, alla variabile *name* venga assegnato il valore contenuto nella seconda colonna del result set, e così via.

La funzione *convert(...)* realizza nella pratica la conversione utilizzando i meccanismi di reflection messi a disposizione da Java.

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

La funzione prende in input

- Il result set da cui estrarre le informazioni.
- Il mapping “variabile” → “numero colonna”.
- Lo spiazzamento della colonna, ovvero la colonna del result set da cui partire con l'estrazione dei dati.
- L'oggetto in cui copiare i dati.

Inoltre in questa funzione vengono utilizzati una serie di metodi per la conversione delle costanti di QuOnto in tipi di dato Java, visto che la manipolazione di tali costanti è risultata alquanto poco naturale.

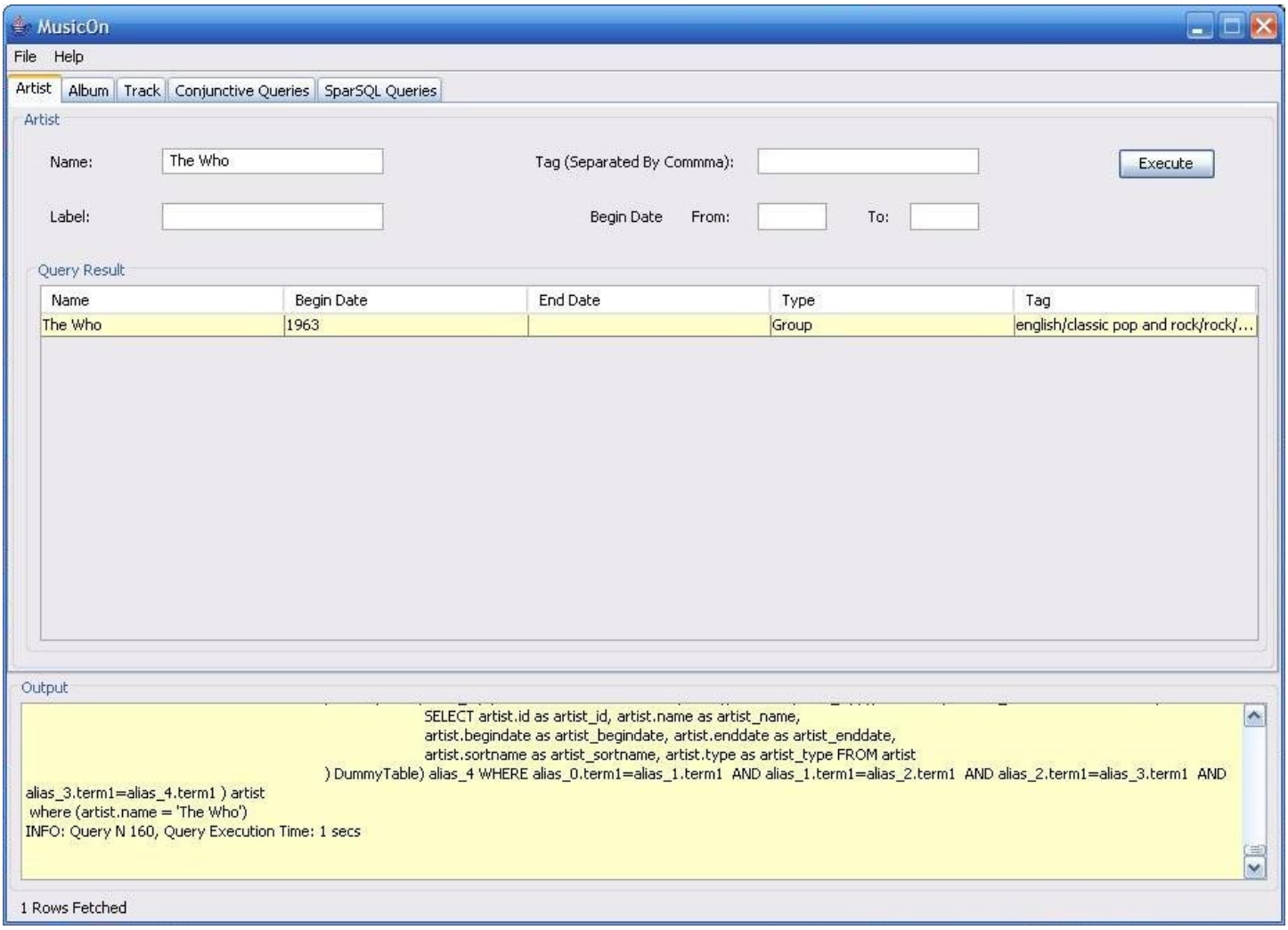
```
public String constantToString(Constant constant)
public int constantToInt(Constant constant)
public Calendar constantToCalendar(Constant constant) throws MusicOnException
```

Sempre per elencare gli strumenti di supporto messi a disposizione da MusicOn, merita una menzione il **parser di query in formato Datalog**, implementato nella classe *QuontoDatalogQueryParser* che ha permesso di utilizzare il formato Datalog in sostituzione di quello XML per esprimere le query congiuntive, con evidente aumento di leggibilità.

[Capitolo 3] Manuale d'uso

Di seguito viene descritta l'interfaccia grafica dell'applicazione e le principali indicazioni sul suo utilizzo.

[3.1] Il tab “Artist”



Nel tab “Artist” sono presenti cinque campi di testo attraverso i quali è possibile eseguire ricerche sugli artisti per:

- “Name”: consente di eseguire ricerche per nome.
- “Label”: consente di eseguire ricerche per casa discografica.
- “Tag”: permette di effettuare ricerche per tag (rock, psychedelic rock...).

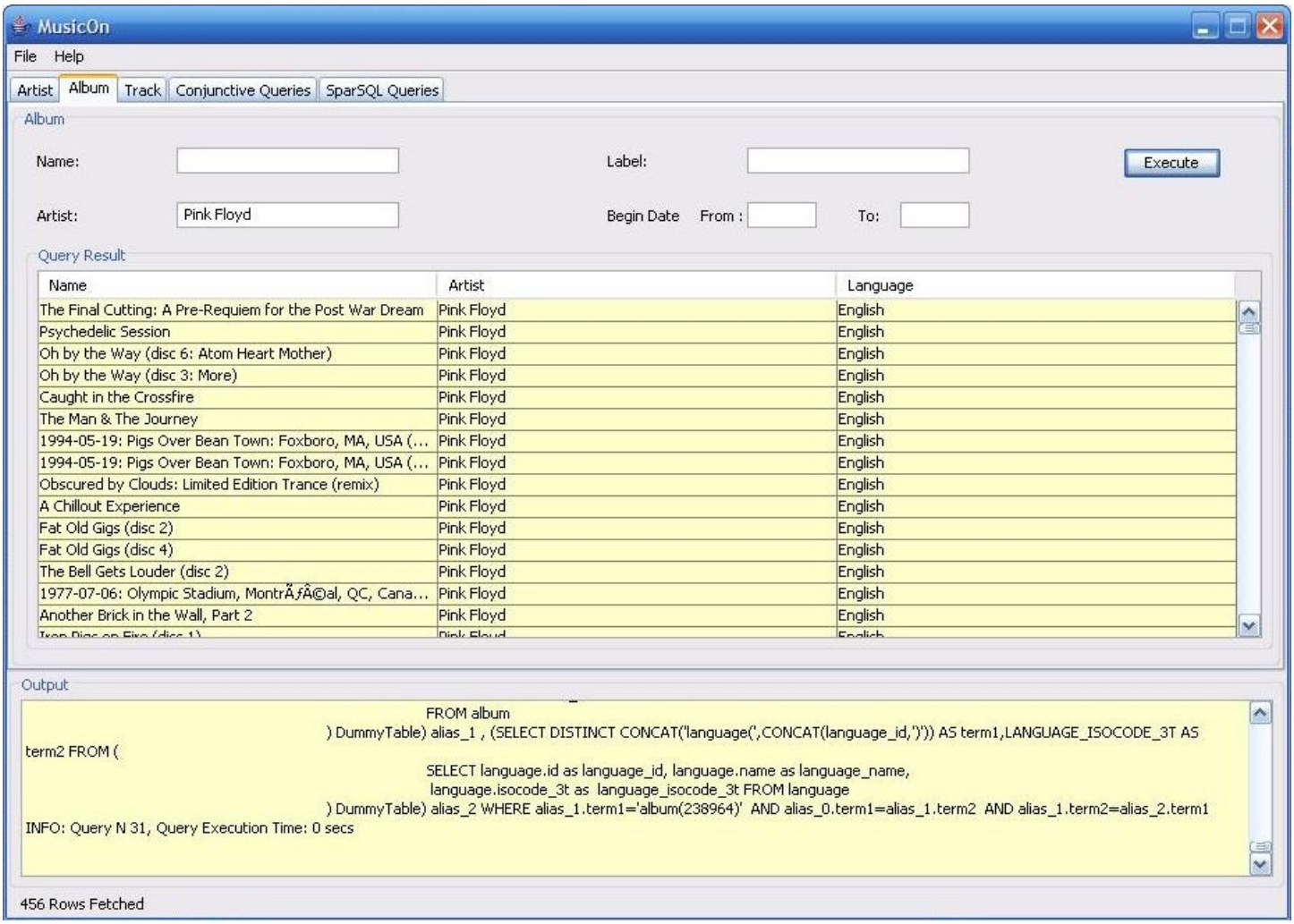
- “BeginDate From” e “To”: campi in cui inserire un periodo temporale in cui cercare tutti gli artisti che abbiano una data di fondazione nell'intervallo definito. Se non viene definito il campo “From” la ricerca viene effettuata a partire dall' anno zero, se non viene specificato il campo “To” la ricerca arriva fino al giorno corrente.

Il pannello “Query Result” contiene la tabella in cui vengono visualizzati i risultati della query effettuata.

Il pannello “Output” contiene un'area di testo in cui vengono stampate informazioni riguardanti la computazione come: le query sparSQL generate, le conjunctive query, le query espanse, il tempo di esecuzione di ciascuna query ed eventuali errori.

Facendo doppio click su una riga della tabella è possibile visualizzare i dettagli degli artisti.

[3.2] Il tab “Album”



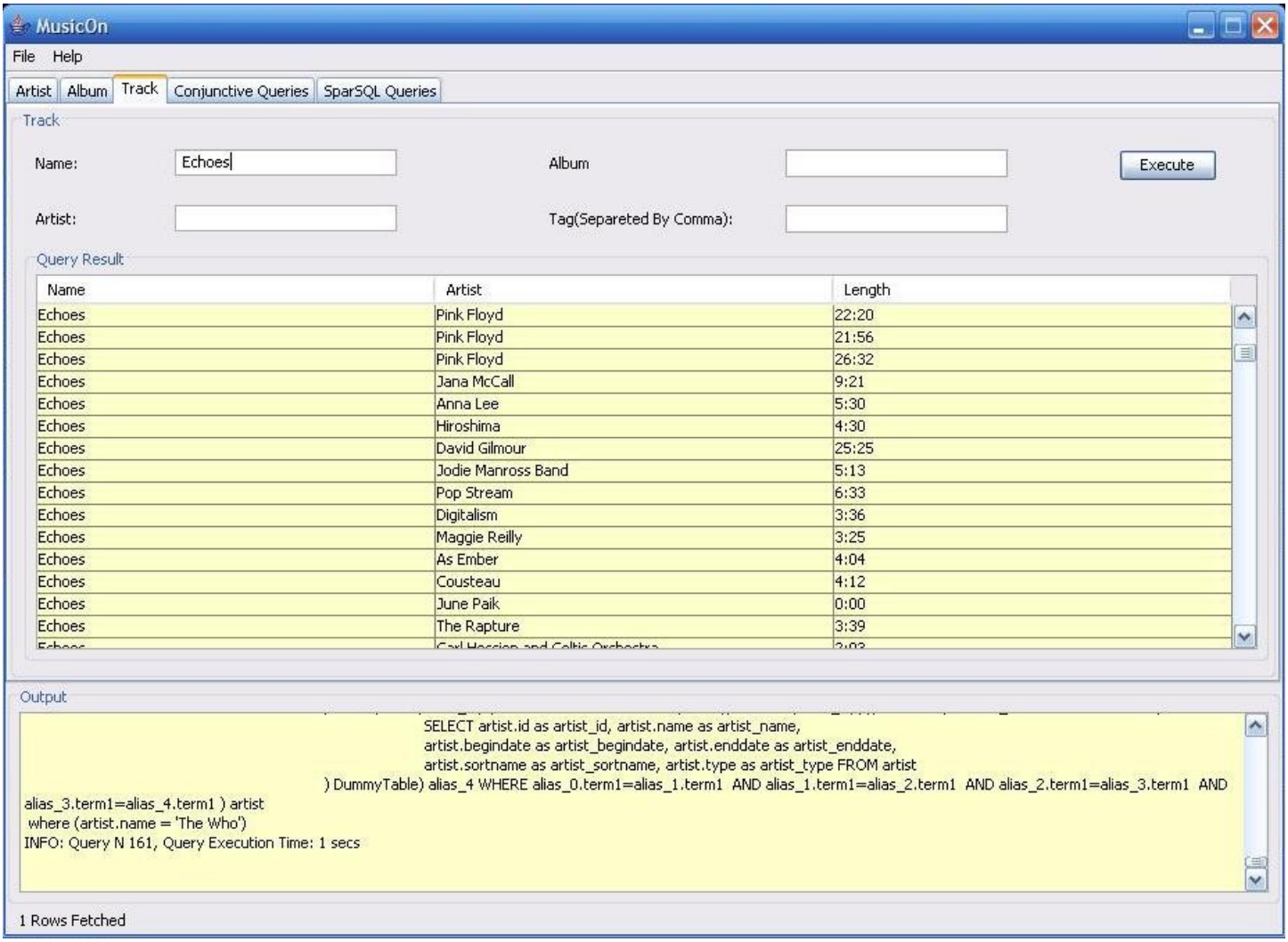
Anche in questo tab sono presenti cinque campi di testo attraverso i quali è possibile eseguire ricerche sugli album in base a:

- “Name”: consente di effettuare ricerche per nome dell' album.
- “Label”: consente di eseguire ricerche per casa discografica.
- “Artist”: permette di effettuare ricerche per nome dell' artista.
- “BeginDate From” e “To”: campi in cui inserire un periodo temporale all'interno del quale ricercare tutti gli album che abbiano una data di pubblicazione in una qualunque nazione nell'intervallo definito. Se non viene definito il campo “From” la ricerca viene effettuata a partire dall' anno zero, se non viene specificato il campo “To” la ricerca arriva fino al giorno corrente.

Anche in questo caso troviamo il pannello “Query Result” che contiene il risultato della ricerca e “Output” in cui sono riportati messaggi di utilità ed eventuali errori.

Facendo doppio click su una riga della tabella è possibile visualizzare i dettagli degli album.

[3.3] Il tab “Track”



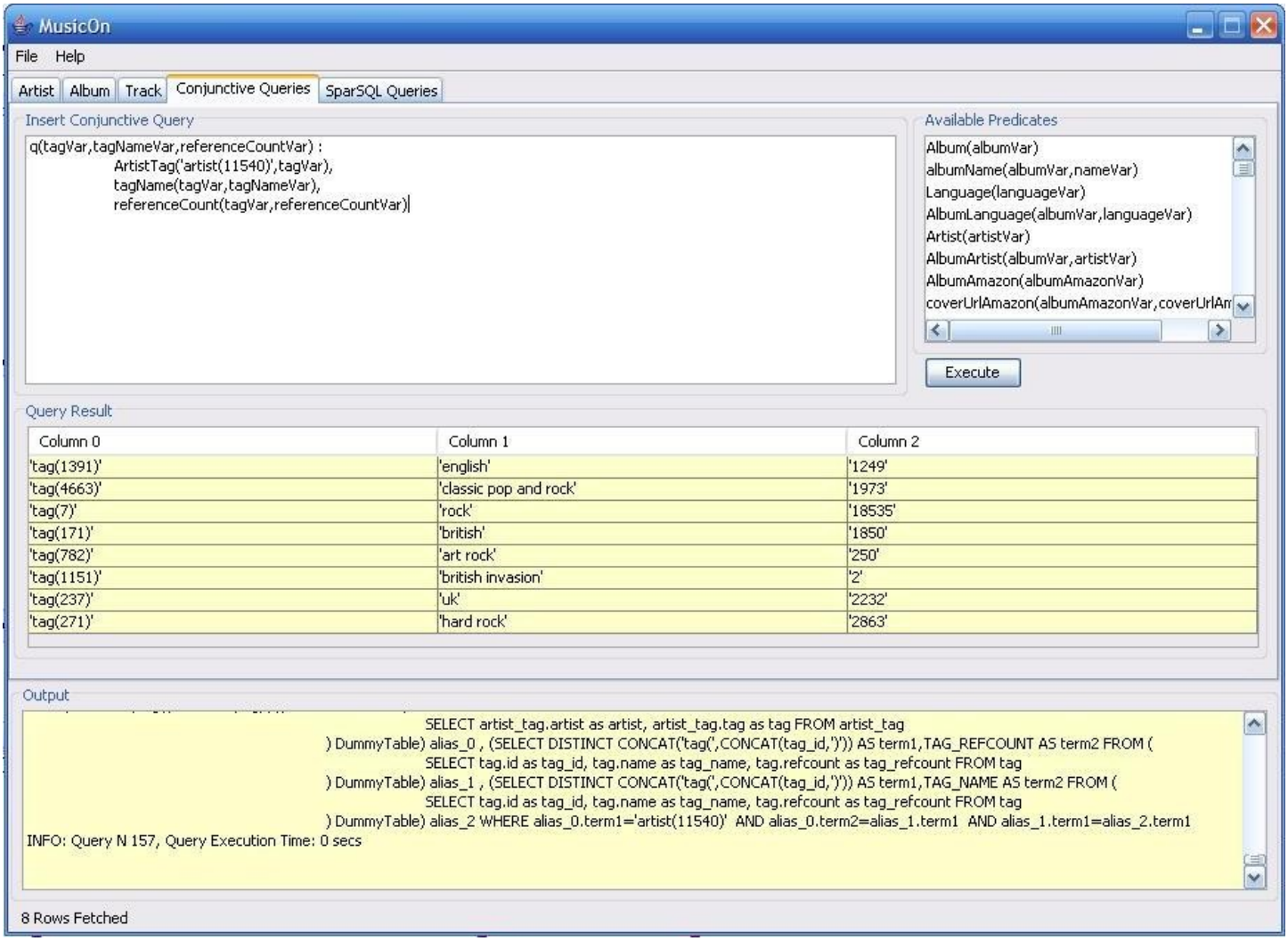
Sono presenti quattro campi di testo attraverso i quali è possibile eseguire ricerche delle canzoni:

- “Name”: consente di effettuare ricerche per nome della canzone.
- “Album”: consente di eseguire ricerche delle tracce per nome dell'album.
- “Artist”: permette di effettuare ricerche di canzoni per nome dell' artista.
- “Tag”: permette di effettuare ricerche di tracce per tag (rock, psychedelic rock...).

Anche qui troviamo i pannelli “Query Result” che contiene la tabella in cui vengono visualizzati i risultati della query effettuata, visualizzando alcuni campi(nome album, artista, lingua) e il pannello “Output”.

Facendo doppio click su una riga della tabella è possibile visualizzare i dettagli di una canzone.

[3.4] Il tab “Conjunctive Queries”



Il tab “Conjunctive Queries” permette di eseguire direttamente delle conjunctive query in sintassi

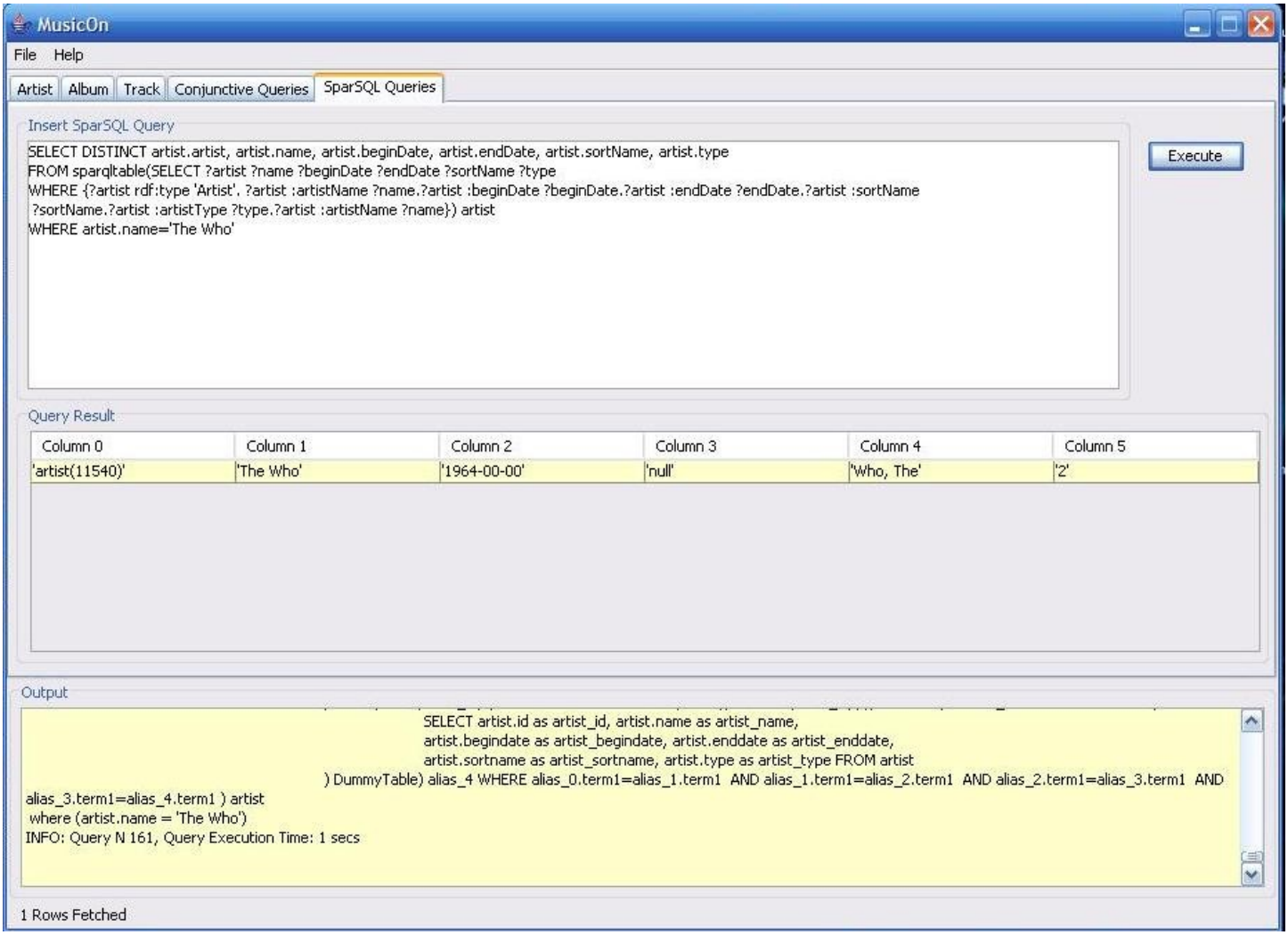
```
q(x): pred1(x,y) , pred2(y,'costante')
```

La lista “Available Predicates” mostra tutti i predicati che è possibile usare nella scrittura della query.

La tabella “Query Result” mostra il risultato della conjunctive query.

Il pannello “Output” contiene un'area di testo in cui vengono stampate informazioni riguardanti la computazione come: le query sparSQL generate, le conjunctive query, le query espanse, il tempo di esecuzione di ciascuna query ed eventuali errori qualora ce ne fossero.

[3.5] Il tab “SparSql Queries”

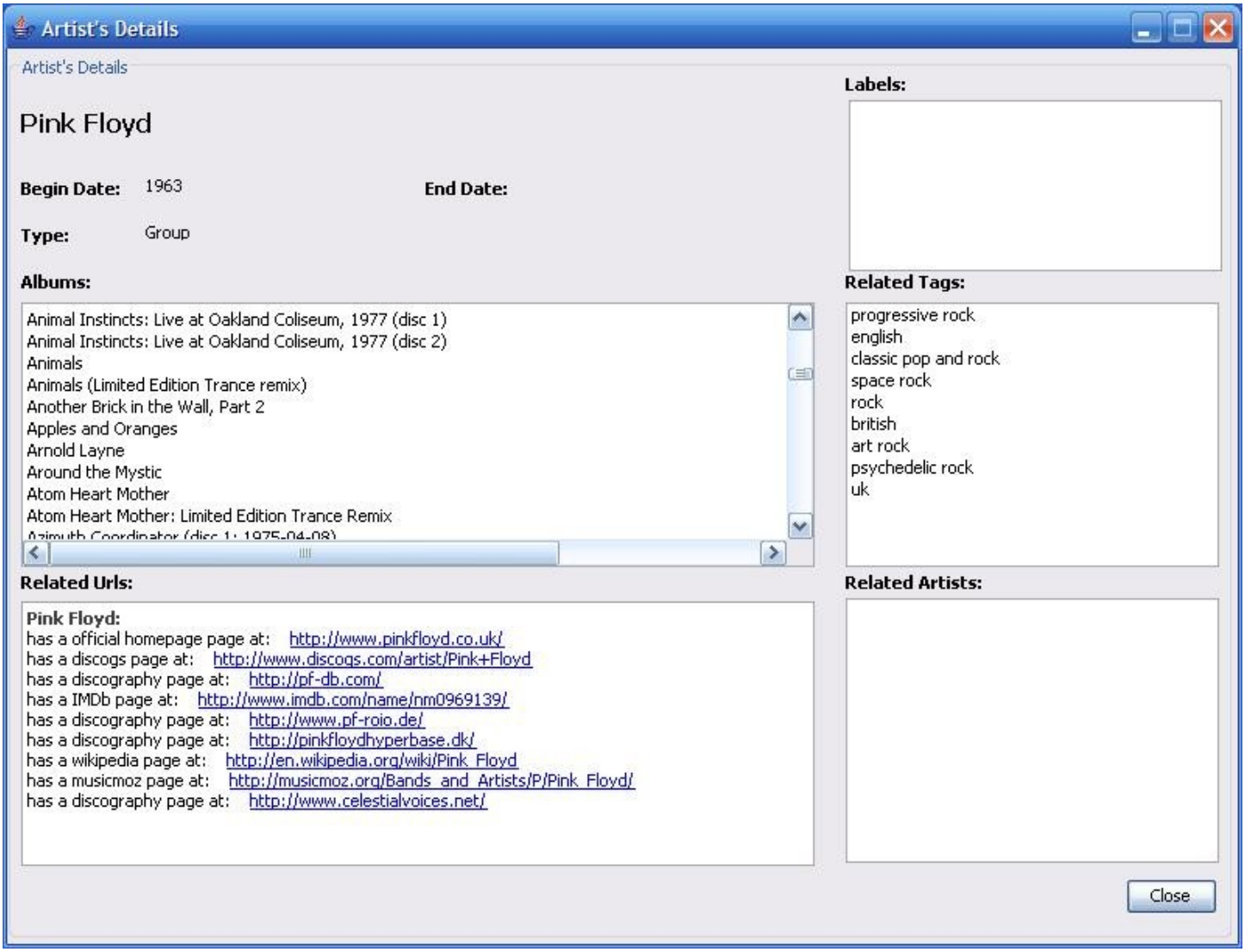


Il tab “SparSQL Queries” permette di eseguire direttamente delle query sparSQL.

La tabella “Query Result” mostra il risultato della query sparSQL.

Il pannello “Output” contiene un'area di testo in cui vengono stampate informazioni riguardanti la computazione come: le query sparSQL generate, le conjunctive query, le query espanse, il tempo di esecuzione di ciascuna query ed eventuali errori qualora ce ne fossero.

[3.6] Finestra dei dettagli di un artista



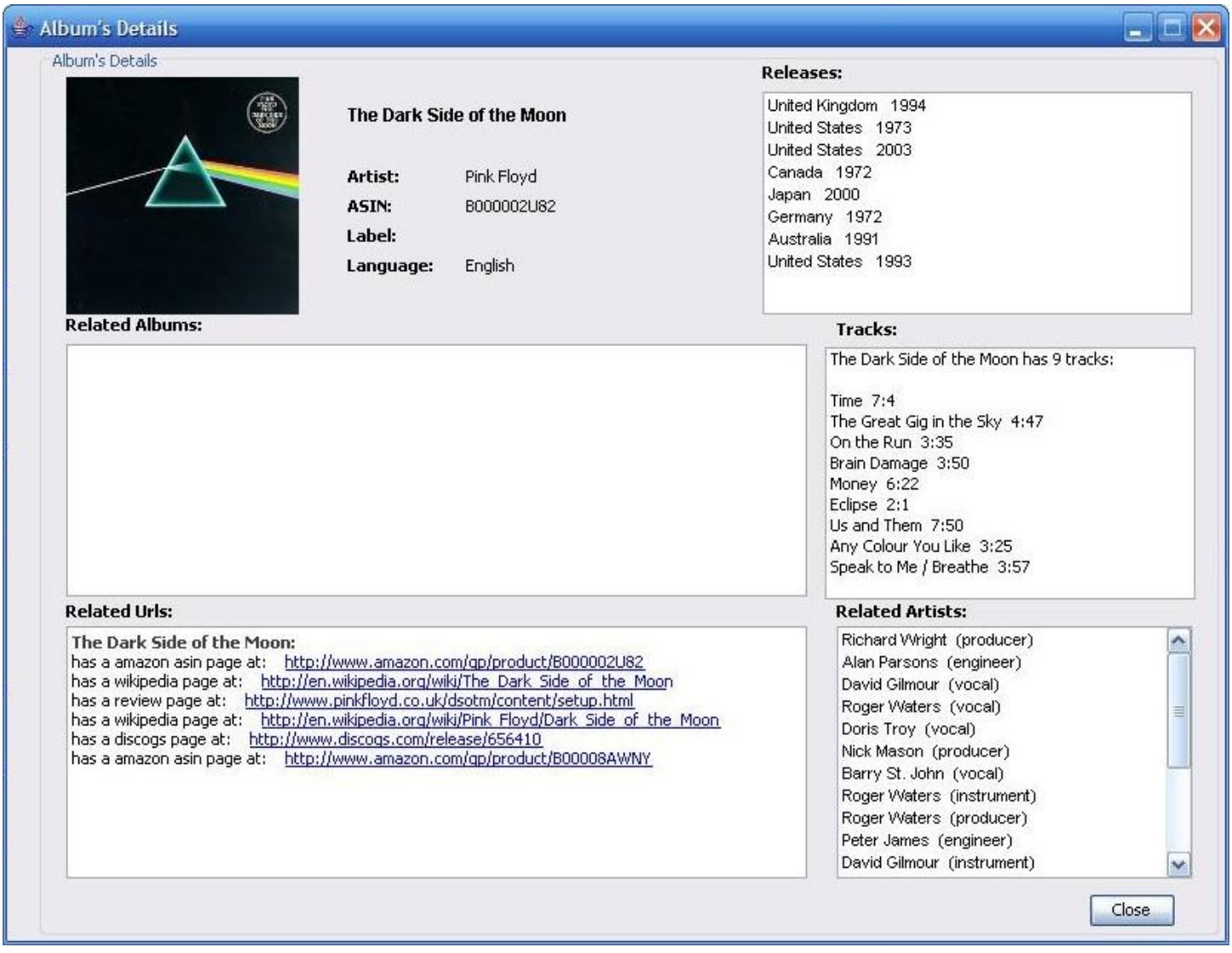
La finestra “Artist Details” visualizzato a seguito di un doppio click su un riga della tabella nel tab “Artist” contiene informazioni dettagliate su un determinato artista, come:

- Nome dell'artista
- Anno di fondazione
- Anno di scioglimento
- Tipo(solista o gruppo).
- L'area di testo “Labels” contiene le case discografiche correlate con l'artista e la relazione tra i due(contratto, fondatore casa discografica...).
- L'area di testo “Albums” contiene l'elenco degli album pubblicati ed anche le contribuzioni

dell'artista ad altri album con il ruolo da essi ricoperto e il nome dell'artista dell'album, al quale hanno prestato il contributo.

- L'area di testo “Related Urls” contiene gli inidrizzi web che contengono informazioni relative all'artista con la tipologia del link(myspace, wikipedia...).
- L'area di testo “Related Tags” contiene tutti i tag associati all'artista.
- L'area di testo “Related Artists” contiene l'elenco degli artisti correlati all'artista con il tipo di legame(ad esempio un solista può essere il cantante di un gruppo).

[3.7] Finestra dei dettagli dell'album

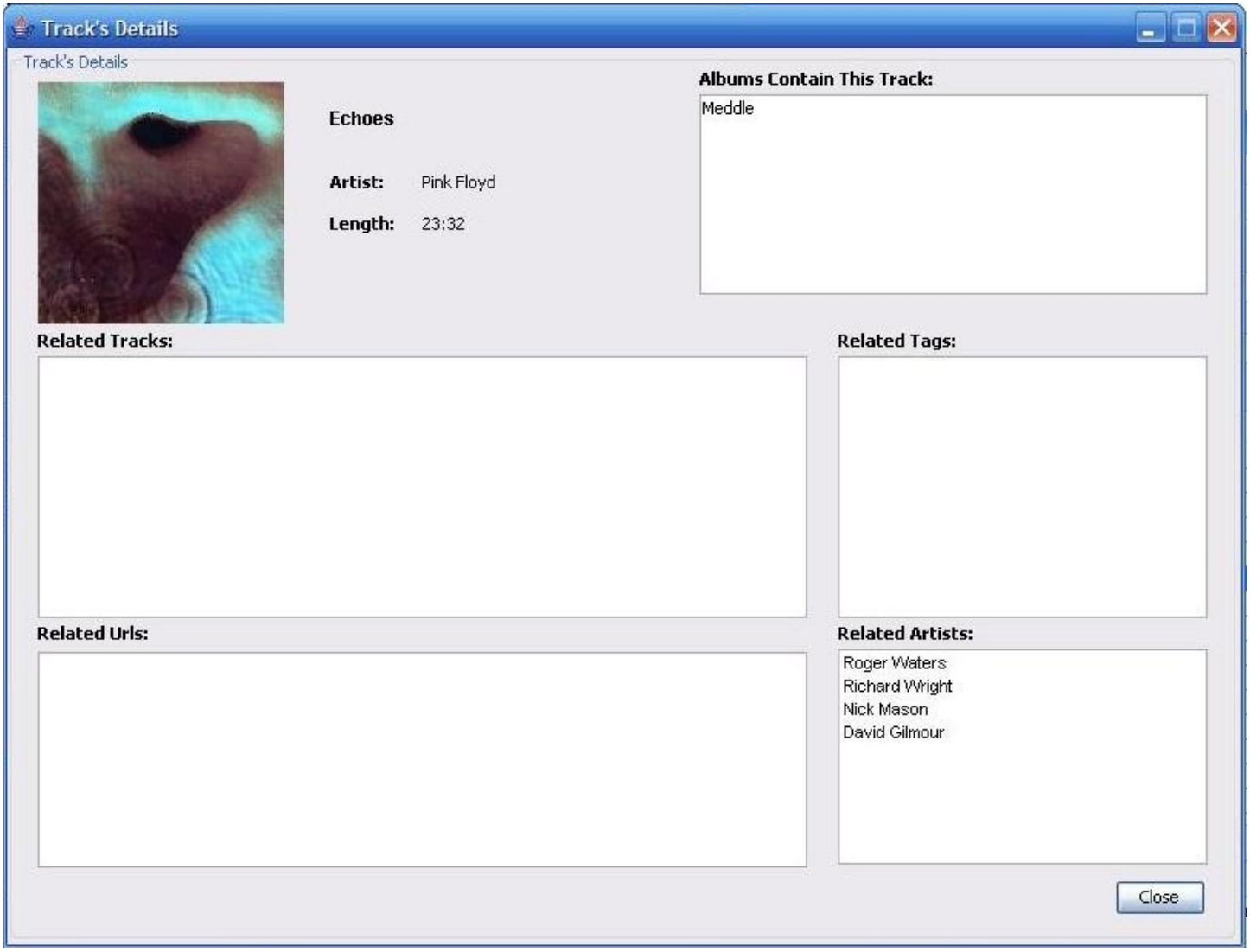


Il pannello “Album Details” visualizzato a seguito di un doppio click su un riga della tabella nel tab “Album” contiene informazioni dettagliate su un album, come:

- Nome dell'album
- La cover art
- L'amazon ASIN
- La casa discografica
- La lingua
- L'area di testo “Releases” contiene l' elenco e delle pubblicazioni nelle varie nazioni con la relativa data.
- L'area di testo “Related Albums” contiene l'elenco degli album correlati all' album selezionato(es. il nome di una raccolta che contiene tale album).
- L'area di testo “Related Urls” contiene gli inidrizzi web che contengono informazioni relative all'album con la tipologia del link(myspace, wikipedia...).
- L'area di testo “Tracks” mostra l' elenco delle tracce contenute nell' album con la relativa durata.

- L'area di testo “Related Artists” contiene l'elenco degli artisti che hanno contribuito all'album, con il tipo di partecipazione(ad esempio come produttore, ingegnere, cantante...).

[3.8] Finestra dei dettagli della traccia



Il pannello “Track Details” visualizzato a seguito di un doppio click su un riga della tabella nel tab “Track” contiene informazioni dettagliate su un traccia, come:

- Nome traccia
- Artista
- Durata
- Cover art dell'album che contiene la traccia.
- L'area di testo “Albums Contains This Track” mostra l' elenco degli album che contengono la canzone in questione.
- L'area di testo “Related Tags” contiene l'elenco dei tag associati alla traccia.

- L'area di testo “Related Urls” contiene gli inidrizzi web che contengono informazioni relative alla canzone con la tipologia del link(myspace, wikipedia...).
- L'area di testo “Related Tracks” mostra l' elenco delle tracce correlate alla traccia selezionata.
- L'area di testo “Related Artists” contiene l'elenco degli artisti che hanno contribuito alla traccia, con il tipo di partecipazione(ad esempio come produttore, ingegnere, cantante...).

[Capitolo 4] Note sull'utilizzo del sistema QuOnto

[4.1] Utilizzo di QuOnto per l'accesso ai dati

In questa sezione verranno analizzati i vantaggi e gli svantaggi sull'utilizzo dell'attuale implementazione del sistema QuOnto.

Sembra doveroso iniziare l'analisi con una delle caratteristiche più utili e innovative del sistema, ovvero la possibilità di eseguire query direttamente sul livello di astrazione concettuale, ovvero sull'ontologia. Durante l'implementazione dell'applicazione questo aspetto ha permesso di r**isolvere in modo del tutto immediato e trasparente il problema dell'impedence mismatch**, eliminando lo sforzo aggiuntivo richiesto per gestire la differenza tra organizzazione logica dei dati e la loro rappresentazione concettuale. Questo passo, di fatto, viene eseguito una volta per tutte con la creazione del mapping ontologia-base di dati, ed è nascosto al resto dell'applicazione. Questa caratteristica, può essere utilmente sfruttata per realizzare una **integrazione semantica dei dati** in modo semplice, intuitivo, e trasparente, anche se questa capacità non è stata sfruttata in MusicOn, essendo stato definito il mapping verso un solo schema di base di dati.

Dal punto di vista pratico, c'è da dire che QuOnto mette a disposizione un insieme di **API molto semplice e intuitiva**, anche se alcuni dettagli potrebbero essere migliorati per renderne più agevole l'utilizzo in un contesto di sviluppo che vada oltre quello accademico. In particolare, la scrittura dell'ontologia e del **mapping sotto forma di file XML** è risultata semplice e immediata anche se un po' prolissa (in MusicOn abbiamo circa 1800 righe per il mapping e 2000 per l'ontologia).

La definizione delle **query congiuntive in formato XML** o direttamente tramite le API si è invece rivelata alquanto ostica e il risultato, sopratutto per le query più complesse, è una scarsa leggibilità e manutenibilità. Un discorso a parte, va fatto per le query scritte in **formato Datalog** la cui scrittura è risultata molto più agevole. Per quanto riguarda le **query SparSql**, invece, c'è da dire che a parte la complessità intrinseca che si affronta manipolando contemporaneamente query Sparql e query Sql, sono risultate abbastanza semplici da scrivere e (ovviamente) molto più potenti delle query datalog.

L'unica pecca incontrata nel modulo **QuontoEql** (il modulo che permette l'invocazione di query SparSql), è l'assenza del supporto **l'operatore Sql LOWER(...)**, che anche se presente nello standard Sql92 non è riconosciuto dal parser SparSql. Questa carenza, nella pratica, ha impedito l'implementazione delle ricerche case-insensitive all'interno del database.

Un altro problema, a cui si è già accennato, è un **bug nella gestione della chiusura degli statement** sql da parte di QuOnto. In pratica, all'esecuzione della query numero 300, all'interno della stessa sessione, il database lancia un errore irreversibile del tipo "*maximum open cursors exceeded*", sintomo del fatto

che gli statements una volta aperti non vengono chiusi, anche se viene chiamato il metodo *close()* sul result set restituito da QuOnto, causando il raggiungimento del limite massimo di cursori contemporaneamente aperti. Tale problema è stato aggirato chiamando la funzione

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

prima della query numero 300, che di fatto chiude tutti gli statements rimasti ancora aperti e permettendo di eseguire senza problemi un numero arbitrario di query.

Un'altra carenza riscontrata nel sistema, è da segnalare nella **gestione dei result set**. In particolare, è risultato poco naturale l'utilizzo di un result set in cui l'accesso alle colonne avviene soltanto tramite un indice numerico, e non tramite un più intuitivo e leggibile identificativo testuale. Inoltre i dati stessi resituiti nel result set, sono tutti modellati sotto forma di oggetti della classe *Constant*, richiedendo quindi un passo aggiuntivo per identificare e convertire tali oggetti nel tipo di dato desiderato.

Un'altra nota negativa, purtroppo, riguarda la **poca leggibilità delle query Sql generat**e dal processo di ragionamento/query unfolding, che oltre ad utilizzare alias poco esplicativi, risultano essere a volte lunghissime rendendo molto difficile il debug (in particolare del mapping).

Un discorso a parte merita l'analisi delle prestazioni, che verrà affrontata nel paragrafo seguente.

[4.2] Analisi delle prestazioni e query di esempio

Si è osservato che, ovviamente, il problema delle prestazioni di QuOnto è molto legato al DBMS utilizzato come sorgente dati e più precisamente, alla **capacità del DBMS di ottimizzare la query** SQL generata da QuOnto. Purtroppo tali query risultano essere molto lunghe e complesse, essendo composte da vari livelli di annidamento, sottoquery, e operatori per la manipolazione delle stringe (CONCAT in particolare). Tale peculiarità, ha creato non pochi problemi, visto che il database di Music Brainz è stato inizialmente installato su DBMS Mysql, che è risultato del tutto inadeguato all'utilizzo con QuOnto e un database di tali dimensioni. Solo in un secondo momento è stata effettuata la migrazione del database su DBMS Oracle XE, che invece si è rivelato sorprendentemente veloce rispetto a MySql. Solo per farsi un idea consideriamo la query congiuntiva

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

che restituisce i nomi degli album dei Pink Floyd. La risultante query SQL generata da QuOnto risulta essere:

```
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'
```

Utilizzando il DBMS MySql tale query veniva eseguita in circa 15 minuti se veniva specificata la clausola DISTINCT, in circa 10 minuti se tale clausola veniva omessa. Con Oracle XE, invece la query impiega solo 1 secondo ad essere eseguita con la clausola DISTINCT!

Abbiamo ipotizzato che uno dei motivi di una tale differenza prestazionale, è da ricercare nel fatto che MySql materializza le tabelle risultanti dalle query annidate, andandole a scrivere fisicamente su disco per poi eseguire la query al livello di annidamento superiore. Questa supposizione è stata fatta dopo aver notato una forte attività di scrittura su disco in seguito all'esecuzione di una query su MySql. Oracle XE, invece, sembra riuscire ad ottimizzare la query in modo tale da non necessitare alcuna materializzazione dei risultati delle query annidate.

Verrà ora fatta una rassegna di alcune query significative con i relativi tempi di esecuzione su DBMS Oracle XE.

Query 1 (Conjunctive Query)

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 2 (SparSql)

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 3 (SparSql)

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 ritorna 1 risultato.