SEMINARI DI INGEGNERIA DEL SOFTWARE

"Traduzione di diagrammi ER in Protégé"

a.a. 2005/2006

Emma Di Pasquale
Teresa Raguso

INDICE

Introduzione

Il presente lavoro ha come obiettivo la creazione di ontologie a partire da diagrammi ER utilizzando come editor Protégé, un tool che consente di implementare schemi concettuali, gestendo in maniera efficace ed intuitiva le relazioni IS_A e l'ereditarietà.
Partendo quindi, dai diagrammi ER dei compiti di basi di dati, abbiamo creato l'ontologia corrispondente ovvero una descrizione formale esplicita dei concetti del dominio. Un'ontologia cioè rappresenta il *modello concettuale* di un mondo, la struttura formale di un pezzo di realtà percepita ed organizzata da chi modella e tenta di formulare uno schema concettuale esaustivo e rigoroso nell'ambito di un certo dominio.
Dovendo quindi, modellare un dominio, le nostre ontologie dovevano essere in grado di esprimere entità, relazioni, attributi, vincoli e corrispondenze con i data base ed è per questo che, laddove possibile, si è cercato di tradurre tutti questi elementi, presenti in ogni schema ER, in corrispondenti concetti, proprietà e restrizioni delle ontologie. Abbiamo quindi progettato le nostre ontologie procedendo nel seguente modo:

- definizione delle classi e organizzazione delle stesse in una gerarchia tassonomica (sottoclassi-superclassi);
- definizione delle proprietà e descrizione dei valori leciti per ciascuna delle classi;
- creazione delle istanze;
- attribuzione dei valori alle proprietà per tutte le istanze create.

Dopo aver utilizzato Protégé per descrivere il dominio di interesse con il procedimento sopra espresso, abbiamo inserito delle regole di inferenza rendendo così completa la nostra base di conoscenza.
Essendo le nostre ontologie realizzate in OWL-DL, abbiamo utilizzato un reasoner (o classificatore) basato sulla Logica Descrittiva: RACER.
Racer è un classificatore che offre un insieme di servizi per ragionare (fare inferenza) sulle basi di conoscenza.
Tra le principali funzionalità offerte da Racer c'è: il **controllo di consistenza** delle classi dell'ontologia. Una classe è detta consistente se possono esistere individui appartenenti ad essa.
Un'altra funzionalità è la **classificazione dell'ontologia** che consente di ottenere la *gerarchia desunta* delle classi, che può essere diversa da quella dichiarata durante la creazione. Abbiamo quindi testato la consistenza delle nostre ontologie e usato la classificazione per estrapolare la gerarchia desunta e stabilire la presenza di equivalenze tra concetti.

1. Preliminari

- **DL:** Per poter sfruttare appieno una ontologia, è fondamentale esprimerla in logica. Le logiche descrittive sono logiche specificamente definite per esprimere la conoscenza in termini di concetti e relazioni, con associate procedure automatiche di ragionamento corretto e completo, e caratterizzate dal punto di vista computazionale. Vengono utilizzate per fornire rigore formale e meccanismi di inferenza a linguaggi concettuali e in particolare a linguaggi per ontologie come OWL.
  I costrutti sintattici di base sono:

  1. concetti atomici (predicati unari);
  2. ruoli atomici (predicati binari);
  3. individui (costanti).

  Una base di conoscenza in DL comprende due componenti:

  1.TBox: contiene la conoscenza intensionale, include la terminologia, ovvero concetti e ruoli del dominio applicativo. Consiste in un insieme di *assiomi terminologici* (inclusione, uguaglianza) che descrivono in che modo i concetti o i ruoli sono in relazione tra loro;

  2.ABox: contiene asserzioni circa gli individui che popolano il mondo in oggetto e quindi la conoscenza estensionale; introduce gli individui, assegnando loro un nome ed asserendo le loro proprietà. Le asserzioni sono di due tipi: *asserzioni di concetto*, che asseriscono l'appartenenza di un individuo ad un concetto ed *asserzioni di ruolo* che asseriscono relazioni tra coppie di individui.

- **OWL:** raccomandazione del W3C, è il linguaggio più diffuso per esprimere ontologie. E' rilasciato in tre diverse versioni, con complessità e potere espressivo crescenti:

  a. *OWL-Lite* è la versione sintatticamente più semplice, da usare nel caso in cui si vogliono definire solo gerarchie di classi e vincoli poco complessi;
  b. *OWL-DL* è la versione intermedia, offre un potere espressivo più elevato di OWL-Lite e mantiene la completezza computazionale (tutte le conclusioni sono computabili, ovvero hanno la garanzia di essere calcolabili) e la decidibilità (tutte le computazioni si concludono in un tempo finito);

c. *OWL-Full* offre la massima espressività senza, però, alcuna garanzia circa completezza e decidibilità.

Con Protégé è possibile scegliere la versione di OWL da utilizzare. Noi abbiamo realizzato le nostre ontologie con OWL-DL perché basato sulle Logiche Descrittive, frammenti decidibili della FOL e quindi di interesse per il ragionamento automatico. Per ontologie conformi ad OWL-DL è possibile calcolare automaticamente la gerarchia di classificazione e verificare le inconsistenze.
Inoltre OWL effettua una assunzione di *mondo aperto* (OWA - Open World Assumption) ovvero l' assunzione di mondo chiuso non viene adottata nelle DL. Quest'ultima assunzione presuppone la conoscenza completa del mondo dell'applicazione, mentre la semantica dell'ABox della DL è compatibile con una situazione di conoscenza parziale, coè di alcune asserzioni si sa che sono vere, di altre che sono false, di altre ancora non si sa nulla.

- **ER:** Sia le ontologie che i Database sono *metodologie di rappresentazione della conoscenza,* si differenziano però in molti aspetti:

  1. Approccio alla realtà da modellare: con un Database si ha una visione più concreta e specifica del mondo; con una ontologia si cerca di modellare concettualmente il mondo (il DB si concentra più sulle istanze, l'ontologia più sulle entità);

  2. Le Ontologie consentono di *ragionare* sul mondo, ovvero di espandere la conoscenza sul mondo, utilizzando *regole di inferenza*, espresse in uno specifico *linguaggio;* la differenza principale fra una base di conoscenza e una base di dati è la possibilità di condurre *ragionamenti* in modo automatico.

  3. Nei DB c'è l' assunzione di mondo chiuso, l'istanza di un database rappresenta esattamente una interpretazione, di conseguenza l'assenza di informazione è interpretata come informazione negativa (falsa); nelle ontologie, invece, c'è l'assunzione di mondo aperto.

2. Protégé

Protégé è una piattaforma open-source che può esportare le ontologie in vari formati: RDF(S), XML Schema e OWL. Basato su Java è estendibile grazie a numerose API e plug-in. Dispone di numerosi ambienti plug-and-play che consentono un rapido sviluppo delle applicazioni.
La piattaforma Protégé consente di utilizzare due modalità per creare le ontologie:

- Il **Protégé-Frames editor**, consente di costruire e popolare le ontologie che sono basate su "frame", secondo il protocollo OKBC;
- Il **Protégé-OWL editor**, consente di costruire ontologie per il *Semantic Web*, in particolare secondo il linguaggio OWL. Un'ontologia OWL può includere descrizioni di classi, di proprietà e le loro istanze.

In questo lavoro è stata usata la seconda modalità ed inoltre è stato selezionato il formato *OWL Files* essendo Protégé in grado di fornire anche formati diversi da OWL; così facendo l'ontologia vuota contiene la sola classe *owl:Thing* che rappresenta l'insieme che contiene tutti gli individui ed è per questo che tutte le classi sono sottoclassi di *owl:Thing*. Oltre al file con estensione *owl*, Protégé crea un ulteriore file con estensione *pprj* che include informazioni aggiuntive relative al progetto, necessarie all'editor alla successiva riapertura.
Protégé ci permette di definire una gerarchia di classi (tassonomia), che corrispondono alle entità dei diagrammi ER presi in esame, ed un insieme di proprietà, che corrispondono alle relazioni.
Per progettare un' ontologia abbiamo bisogno di:

- determinare il **dominio** e lo **scopo** dell'ontologia e quindi di definire i concetti del dominio: in Protégé chiamati *Classes*.
- organizzare i concetti in una **gerarchia** ovvero un insieme di relazioni superclasse/sottoclasse: in Protégé *asserted Hierarchy.*
- definire gli attributi dei concetti, le restrizioni su di essi e le relazioni tra concetti: in Protégé *Properties.*
- definire istanze dei concetti, popolando l' ontologia: in Protégé *Individuals*.

3. ER in Protégé

Classes

Le classi vengono interpretate come insiemi che contengono individui. Possono essere organizzate in una gerarchia di superclassi e sottoclassi meglio conosciuta come tassonomia. In una gerarchia una sottoclasse eredita tutte le proprietà della relativa superclasse. Una caratteristica di OWL-DL è che queste relazioni di sussunzione possono essere calcolate automaticamente dal ragionatore. Inoltre le classi vengono completate da descrizioni relative a condizioni che devono essere soddisfatte da un individuo affinché sia membro di quella classe.

Entità → Classes

Partendo quindi dallo schema concettuale dei diagrammi ER dei compiti di basi di dati abbiamo trasformato ciascuna entità in una classe di Protégé, inserendola nella gerarchia come sottoclasse della classe *Thing*.
Possiamo associare ad ogni classe delle condizioni (*asserted condition*s) che possono essere *necessarie* oppure *necessarie e sufficienti*.
Una classe è detta **primitiva** se è descritta solo attraverso condizioni *necessarie*.
Le condizioni necessarie servono a creare restrizioni sugli individui che possono appartenere ad una classe. In altre parole esse servono ad asserire: "se un individuo appartiene a questa classe allora deve avere queste caratteristiche".
Una classe può essere descritta anche attraverso condizioni *necessarie e sufficienti*. Tali condizioni servono ad asserire: "se un individuo ha queste caratteristiche allora esso fa parte di questa classe".
Le classi **definite** sono classi descritte attraverso condizioni necessarie e sufficienti.
Le condizioni sopra menzionate sono restrizioni (*restrictions*) delle classi ovvero vincoli sull'insieme di individui che possono appartenere ad una classe.
Le restrizioni possono essere di vario tipo. Per esempio le "restrizioni di quantità" sono composte da un quantificatore, da una proprietà e da un argomento (o *filler*).
Le restrizioni che utilizzano il quantificatore esistenziale ($\exists$) sono dette restrizioni esistenziali, quelle che utilizzano il quantificatore universale ($\forall$) sono dette restrizioni universali.
La "restrizione esistenziale" in OWL identifica l'insieme di individui che, per una data proprietà, hanno *almeno* una relazione con gli individui di una specifica classe.
La "restrizione universale", invece, identifica l'insieme di individui che, per una data proprietà, hanno *al più* una relazione con gli individui di una specifica classe.
Tra le condizioni necessarie viene automaticamente inserita da Protégé, anche la super-classe (*Thing* se non è diversamente specificata): la prima condizione necessaria perché un individuo appartenga ad una classe è infatti che esso appartenga anche alla sua super-classe o, meglio, all'insieme delle sue super-classi

(OWL supporta l'ereditarietà multipla). Ogni sottoclasse, inoltre, eredita tutte le condizioni delle superclassi che vengono automaticamente inserite nel gruppo *Inherited*.

Properties

Le *properties* sono relazioni binarie sugli individui, cioè sulle istanze delle classi. Anche per le *properties* è possibile esprimere una gerarchia creando *sub-properties*.

Ci sono due categorie di proprietà:

- OBJECT PROPERTIES, che legano individui ad individui;
- DATATYPE PROPERTIES, che legano individui a valori *datatype* (integers, float, strings, etc).

Relazioni → Object Properties

Per quanto riguarda le relazioni di un diagramma ER, queste vengono trasformate in *Object Properties*; con Protégé è possibile specificare per ogni relazione dominio e codominio.
Il *dominio* di una proprietà è la classe (o l'insieme di classi) ai cui individui appartenenti si può applicare la proprietà.
Il *range* (o codominio) di una proprietà è invece la classe (o l'insieme di classi) i cui individui appartenenti possono essere valori della proprietà.
In altre parole, una proprietà collega individui appartenenti ad un dominio ad individui appartenenti ad un range.
Quando si specificano più classi come dominio (o range) di una proprietà, Protégé interpreta dominio (o range) come l' unione delle classi.
Bisogna tener presente che dominio e range, in Protégé, non vengono visti come vincoli da rispettare, per esempio se consideriamo un dominio (classe) diverso da quello dichiarato per una proprietà, questo non rappresenta un errore a meno che le due classi non siano disjoint.
Le proprietà OWL possono avere diverse caratteristiche, tra cui la transitività, la simmetria e la funzionalità.
Se una proprietà *P* dichiarata **transitiva** relaziona l'individuo *a* con l'individuo *b* e l'individuo *b* con l'individuo *c* allora si può desumere che la proprietà *P* relazioni l'individuo *a* con l'individuo *c* anche se non esplicitamente dichiarato.
Se una proprietà *P* dichiarata **simmetrica** relaziona l'individuo *a* con l'individuo *b* allora si può desumere che la proprietà *P* relazioni l'individuo *b* con l'individuo *a* anche se non esplicitamente dichiarato.
Se una proprietà *P* è dichiarata **funzionale** allora, dato un individuo *a*, esiste al più un individuo *b* che può essere relazionato ad *a* attraverso *P*. Ciò significa che se *P*

relaziona l'individuo *a* con l'individuo *b* e l'individuo *a* con l'individuo *c* allora *b* e *c* sono lo stesso individuo.
Ogni object property può avere una corrispondente **proprietà inversa** (owl:inverseOf): se una proprietà collega un individuo *a* ad un individuo *b*, allora la sua proprietà inversa collegherà l' individuo *b* all' individuo *a* Al fine di poter navigare il diagramma in entrambe le direzioni, abbiamo deciso di creare una relazione inversa per ogni relazione che lega due classi nei diagrammi ER.

Attributi → Datatype Properties

Per poter esprimere gli attributi delle entità, Protégé mette a disposizione le datatype Properties, relazioni che legano individui a valori di tipi ammissibili per XML e/o RDF. Anche in questo caso è possibile specificare il dominio, ovvero la classe a cui l'attributo appartiene e il range che in questo caso è il tipo associato all'attributo (string, int, data, etc..). Per i datatype, però, l'unica caratteristica che può essere associata è di tipo functional.

Restrictions

Vincoli → Restrictions

Alcuni vincoli riguardanti gli schemi ER possono essere tradotti con restrizioni che possono riguardare sia le classi che le proprietà.
Le *restrictions* descrivono un insieme di individui in base al tipo e al numero di relazioni a cui essi partecipano e quindi restringono gli individui che appartengono ad una classe. Possono essere raggruppate in tre grandi categorie:

- Restrizioni di quantità (esistenziale, universale);
- Restrizioni di cardinalità ($\geq$,=,$\leq$);
- Restrizioni "hasValue".

Questi tipi di restrictions sono composti da un quantificatore, una proprietà e un *filler.* I quantificatori usati sono:

- il quantificatore esistenziale ($\exists$), che viene inteso come "almeno uno", "qualche";
- il quantificatore universale ($\forall$), inteso come "solamente".

Una *restriction* descrive una "classe anonima", cioè la classe che contiene tutti gli individui che soddisfano la restrizione. Le *restrictions* vengono usate nella descrizione delle classi per specificare anonime superclassi della classe che viene descritta.

La restrizione esistenziale identifica l'insieme di individui che, per una data proprietà, hanno *almeno una* relazione con individui di una specifica classe.
La restrizione universale identifica l'insieme di individui che, per una data proprietà, hanno *al più* relazioni con individui di una specifica classe.
Il costrutto hasValue ci permette di specificare delle classi sulla base dell'esistenza di *particolari* valori della proprietà. Tuttavia, un individuo sarà un membro di tale classe se almeno *uno* dei valori della sua proprietà è uguale a quello assunto dalla clausola hasValue.
Il linguaggio OWL non assume che gli individui abbiano nome unico e cioè non usa l'UNA (Unique Name Assumption). Perciò con Protégé è possibile asserire che due nomi distinti possono far riferimento allo stesso individuo (owl:sameAs) e analogamente è possibile asserire che due nomi fanno riferimento ad individui differenti (owl:differentFrom).
È anche possibile asserire che n individui sono tutti distinti fra loro (owl:AllDifferent).
OWL non fa alcuna assunzione sulla disgiunzione tra classi ed assume che le classi possano sovrapporsi, se non dichiariamo il contrario; è quindi necessario dichiarare esplicitamente con Protégé che le classi sono disjoint.

Limiti di Protégé

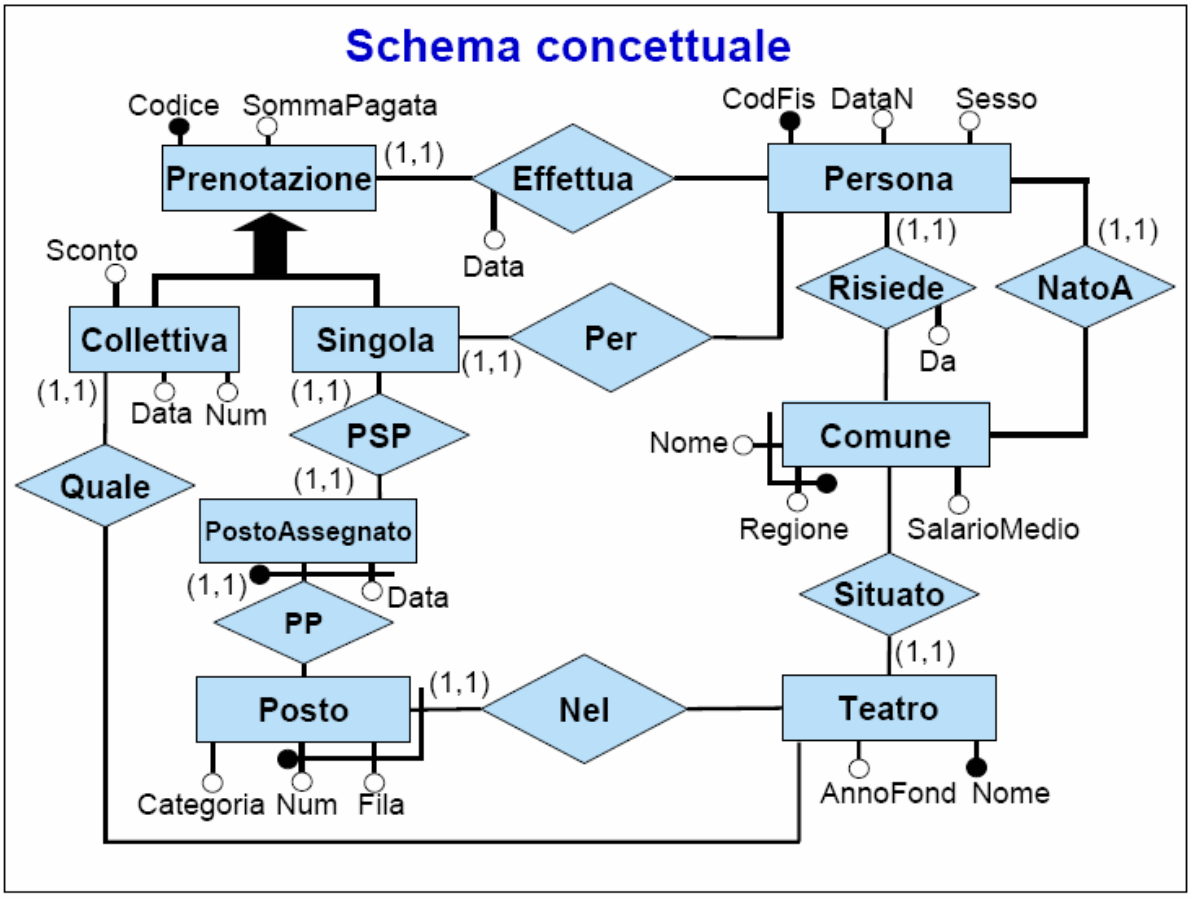
- In OWL-DL non si possono esplicitare le chiavi primarie in quanto non è possibile definire *InverseFunctional* (funzionale inversa) una *datatypeProperty,* mentre questo è possibile scegliendo per Protégé OWL-Full come linguaggio.
- Non si possono esplicitare gli attributi delle relazioni in quanto OWL non permette di inserire una *datatype property* come subproperty di una *object property.*
- Non si può specificare il fatto che gli attributi delle entità siano funzioni totali in quanto il linguaggio usato per comunicare con il ragionatore (DIG) non permette restrizioni sulle cardinalità dei datatypes. Perciò questi sono stati posti *functional* per indicare che ad ogni istanza di una entità può essere associato al massimo un valore del range dell' attributo.

4. Il Ragionatore - Racer

Per le ontologie che rientrano nelle potenzialità di OWL-DL, possiamo usare un ragionatore per inferire informazioni che non sono esplicitamente rappresentate nell'ontologia. Racer è un classificatore DIG compliant e Protégé comunica con questo via HTTP. I servizi di ragionamento offerti da Racer sono:

- **sussunzione** : calcola la gerarchia inferita delle classi dell'ontologia (tramite Classify taxonomy in Protégé);
- **equivalenza**: consiste nel determinare se due o più classi sono equivalenti;
- **consistenza**: consiste nel determinare se le classi dell'ontologia possono avere delle istanze e quindi una classe è considerata inconsistente se non è possibile avere alcuna istanza;
- **istanziazione**: consiste nel determinare gli individui di una classe ad esempio se volessimo estrarre delle informazioni dall'ontologia attraverso una query, non essendo Racer in grado di fare query ma solo di classificare concetti, bisogna considerare tale query come classe ed attraverso la funzione "Compute individuals belonging to classes" si possono calcolare attraverso l'ausilio del ragionatore gli individui che appartengono a tale classe.

- **COMPITO A del 16/12/04**



La traduzione dello schema concettuale sopra riportato in una ontologia OWL utilizzando Protégé , è stata effettuata traducendo:

- le entità in *classi*;
- le relazioni in *object properties*;
- gli attributi in *datatype properties*.

ENTITA'

Non è stato possibile esprimere le chiavi primarie delle varie entità in quanto bisognava esprimere tale vincolo ponendo la datatype property *inverseFunctional* e questo non è ammesso usando OWL DL.
Le classi dell'ontologia, tranne le sottoclassi della gerarchia con la relativa superclasse, sono state poste disjoint a due a due per evitare che gli individui di una classe appartengano ad un'altra classe.

Inoltre tutti gli individui che appartengono alle singole classi sono stati posti differenti tra di loro in quanto OWL non utilizza l'Unique Name Assumption e cioè non assume che gli individui abbiano nome unico.

DifferentIndividuals(a:Terry a:Emma)

Inoltre non è stato possibile specificare il fatto che gli attributi di entità siano funzioni totali, ma sono stati posti *functional* per indicare che ad ogni istanza di un'entità si può associare al massimo un valore del dominio associato all'attributo.

PERSONA
L' entità "Persona" del diagramma ER, è stata tradotta in una *classe* di Protégé.
Essa partecipa alle relazioni "effettua","inversa_di_per", "risiede" e "natoA" ma, mentre per le relazioni "effettua", "inversa_di_per" vi partecipa con cardinalità 0..n, alla relazione "risiede" vi partecipa con cardinalità 1..1. Bisogna quindi, per quest' ultima relazione, porre una *restriction*, ovvero un vincolo all' interno della classe, per esprimere la cardinalità minima con cui l' entità "Persona" partecipa alla relazione "risiede":

Class(a:**Persona** complete
intersectionOf(restriction(a:**risiede**minCardinality(1)) restriction(a:**natoA**
minCardinality(1))))

Complete sta ad indicare che la classe "Persona" è definita, cioè abbiamo almeno una *restriction* tra le condizioni necessarie e sufficienti di questa classe.
Tale classe ha l'attributo "sesso", che è functional, ha come dominio "Persona" e come range una stringa i cui valori possono essere solo (owl:oneOf) "maschile" o "femminile":

DatatypeProperty(a:**sesso** Functional
domain(a:**Persona**)
range(oneOf("**femminile**"^^<http://www.w3.org/2001/XMLSchema#string>
"**maschile**"^^<http://www.w3.org/2001/XMLSchema#string>)))

Gli altri attributi di tale classe sono stati posti tutti *functional:*

DatatypeProperty(a:**codiceFiscale** Functional
domain(a:**Persona**)
range(xsd:**string**))

DatatypeProperty(a:**dataN** Functional
domain(a:**Persona**)
range(xsd:**date**))

PRENOTAZIONE
L' entità "Prenotazione" del diagramma ER, è stata tradotta in una *classe* di Protégé.
Essa nel diagramma ER è padre di una generalizzazione in cui le sottoentità sono "Collettiva" e "Singola"; per esplicitare questo sono state inserite le due classi "Collettiva" e "Singola" come sottoclassi di "Prenotazione" .
Il vincolo "complete" della generalizzazione è stato espresso inserendo la *restriction* "Collettiva U Singola " di tipo "necessaria e sufficiente" tra le "asserted conditions" della classe "Prenotazione".
Essa, inoltre, partecipa alla relazione "effettua" con cardinalità 1..1. Bisogna quindi, porre una *restriction* tra le "condizioni necessarie" della classe "Prenotazione" per esprimere la cardinalità minima con cui l'entità associata partecipa alla relazione "effettua":

Class(a: **Prenotazione** complete
intersectionOf(restriction(a:**inversa_di_effettua** minCardinality(1))
unionOf(a:**Collettiva** a:**Singola**)))
Complete sta ad indicare che la classe "Prenotazione" è una classe definita cioè ha almeno una *restriction* tra le condizioni necessarie e sufficienti.

Complete sta ad indicare che la classe "Prenotazione" è una classe definita cioè ha almeno una *restriction* tra le condizioni necessarie e sufficienti.
Gli attributi di tale classe sono stati posti tutti *functional*:

DatatypeProperty(a:**codice** Functional
domain(a:**Prenotazione**)
range(xsd:**string**))

DatatypeProperty(a:**sommaPagata** Functional
domain(a:**Prenotazione**)
range(xsd:**float**))

COLLETTIVA
L' entità "Collettiva" del diagramma ER, è stata tradotta in una *classe* di Protégé.
Essendo questa un'entità figlia della classe "Prenotazione", erediterà tutte le proprietà e gli attributi della classe padre e inoltre tra le condizioni necessarie Protégé inserirà automaticamente la classe "Prenotazione" per indicare che ogni individuo della classe "Collettiva" è necessariamente un individuo della classe "Prenotazione".

Inoltre, essa partecipa alla relazione "inversa" e vi partecipa con cardinalità 1..1. Bisogna quindi, per quest' ultima relazione, porre una *restriction*, ovvero un vincolo all'interno della classe, per esprimere la cardinalità minima con cui l'entità "Collettiva" partecipa alla relazione "quale":

Class(a:**Collettiva** complete
restriction(a:**quale** minCardinality(1)))

Class(a:**Collettiva** partial
a:**Prenotazione**)

Complete sta ad indicare che la classe "Collettiva" è definita, cioè abbiamo almeno una *restriction* tra le condizioni necessarie e sufficienti di questa classe. *Partial* sta ad indicare che la classe "Collettiva" ha almeno una condizione necessaria.
Gli attributi di tale classe sono stati posti tutti *functional*:

DatatypeProperty(a:**sconto** Functional
domain(a: **Prenotazione**)
range(xsd:**float**))

DatatypeProperty(a:**dataPrenotazioneColl** Functional
domain(a: **Prenotazione**)
range(xsd:**string**))

DatatypeProperty(a:**numeroPrenotazioneSingola** Functional
domain(a:**Prenotazione**)
range(xsd:**int**))

SINGOLA
L' entità "Singola" del diagramma ER, è stata tradotta in una *classe* di Protégé.
Essendo questa un'entità figlia della classe "Prenotazione", erediterà tutte le proprietà e gli attributi della classe padre e inoltre tra le condizioni necessarie Protégé inserirà automaticamente la classe "Prenotazione" per indicare che ogni individuo della classe "Singola" è necessariamente un individuo della classe "Prenotazione".
Inoltre, essa partecipa alle relazioni "psp" e "per" alle quali vi partecipa con cardinalità 1..1. Bisogna quindi, per tali relazione, porre una *restriction*, ovvero un vincolo all'interno della classe, per esprimere la cardinalità minima con cui l'entità "Singola" partecipa alle relazione "psp" e "pp":

Class(a:**Singola** complete
intersectionOf(restriction(a:**psp** minCardinality(1)) restriction(a:**per**
minCardinality(1))))

Class(a:**Singola** partial
a:**Prenotazione**)

Complete sta ad indicare che la classe "Singola" è definita, cioè abbiamo almeno una *restriction* tra le condizioni necessarie e sufficienti di questa classe.
Partial sta ad indicare che la classe "Singola" ha almeno una condizione necessaria.

La classe “Singola” è disgiunta dalla classe “Collettiva”, cioè nessun individuo della classe “Singola” può appartenere alla classe “Collettiva”. Questo vincolo, con Protégé, deve essere inserito in entrambe le classi secondo questa sintassi:

DisjointClasses(a:**Collettiva** a:**Singola**).

COMUNE
L’ entità “Comune” del diagramma ER, è stata tradotta in una *classe* di Protégé.
Essa partecipa alle relazioni “inversa_di_natoA”, “inversa_di_situato”e “inversa_di_risiede” alle quali vi partecipa con cardinalità 0..n .

Class(a:**Comune** partial)

Partial sta ad indicare che la classe “Comune” ha almeno una *restriction* tra le condizioni necessarie.
Gli attributi di tale classe sono stati posti tutti *functional*:

DatatypeProperty(a:**nomeComune** Functional
domain(a:**Comune**)
range(xsd:**string**))

DatatypeProperty(a:**regioneComune** Functional
domain(a:**Comune**)
range(xsd:**string**))

DatatypeProperty(a:**salarioMedio** Functional
domain(a:**Comune**)
range(xsd:**float**))

TEATRO
L’ entità “Teatro” del diagramma ER, è stata tradotta in una *classe* di Protégé.
Essa partecipa alla relazione “inversa_di_nel” con cardinalità 0..n e alla relazione “situato” con cardinalità 1..1.Bisogna, per quest'ultima relazione, porre una *restriction*, ovvero un vincolo all’interno della classe, per esprimere la cardinalità minima con cui l’entità “Teatro” partecipa alle relazione “situato”:

Class(a:**Teatro** complete
restriction(a:**situato** minCardinality(1)))

Complete sta ad indicare che la classe “Teatro” è definita, cioè c'è almeno una *restriction* tra le condizioni necessarie e sufficienti di questa classe.

Gli attributi di tale classe sono stati posti tutti *functional*:

DatatypeProperty(a:**nomeTeatro** Functional
domain(a:**Teatro**)
range(xsd:**string**))

DatatypeProperty(a:**annoFondazione** Functional
domain(a:**Teatro**)
range(xsd:**gYear**))

POSTO

L' entità "Teatro" del diagramma ER, è stata tradotta in una *classe* di Protégé.
Essa partecipa alla relazione "inversa_di_pp" con cardinalità 0..n e alla relazione "nel" con cardinalità 1..1.Bisogna, per quest'ultima relazione, porre una *restriction*, ovvero un vincolo all'interno della classe, per esprimere la cardinalità minima con cui l'entità "Posto" partecipa alle relazione "nel":

Class(a:**Posto** complete
restriction(a:**nel** minCardinality(1)))

Complete sta ad indicare che la classe "Posto" è definita, cioè c'è almeno una *restriction* tra le condizioni necessarie e sufficienti di questa classe.
Gli attributi di tale classe sono stati posti tutti *functional*:

DatatypeProperty(a:**numberoPosto** Functional
domain(a:**Posto**)
range(xsd:**int**))

DatatypeProperty(a:**categoriaPosto** Functional
domain(a:**Posto**)
range(xsd:**int**))

DatatypeProperty(a:**fila** Functional
domain(a:**Posto**)
range(xsd:**string**))

POSTO ASSEGNATO

L' entità "PostoAssegnato" del diagramma ER, è stata tradotta in una *classe* di Protégé.
Essa partecipa alle relazione "pp" e "psp" con cardinalità 1..1. Bisogna, per tali relazioni, porre una *restriction*, ovvero un vincolo all'interno della classe, per esprimere la cardinalità minima con cui l'entità "PostoAssegnato" partecipa alle relazioni "pp" e "psp":

Class(a:**PostoAssegnato** complete

intersectionOf(restriction(a:**pp** minCardinality(1)) restriction(a:**inversa_di_psp** minCardinality(1))))

Complete sta ad indicare che la classe “PostoAssegnato” è definita, cioè c'è almeno una *restriction* tra le condizioni necessarie e sufficienti di questa classe.
Tale classe ha questo attributo che è stato posto *functional*:

DatatypeProperty(a:**dataPostoAssegnato** Functional
domain(a:**PostoAssegnato**)
range(xsd:**date**))

RELAZIONI

Non è stato possibile creare l'attributo "data" della relazione "effettua" in quanto bisognava creare una *datatype property* (rappresentante l'attributo "data") come *subproperty* di una *object property* (rappresentante la relazione "effettua"), ma ciò non è permesso in OWL. Lo stesso vale per l'attributo "da" della relazione "risiede".
Per ogni relazione è stata creata la relazione inversa (cioè ha dominio e range invertiti rispetto alla relazione diretta). Infatti:

- la relazione "effettua" ha come dominio la classe "Persona" e come range la classe "Prenotazione" ed è *inverseFunctional* in quanto la sua inversa "inversa_di_effettua" è *functional* :

  ObjectProperty(a:**effettua** InverseFunctional
  inverseOf(a:**inversa_di_effettua**)
  domain(a:**Persona**)
  range(a:**Prenotazione**))

  La sua relazione inversa, chiamata "inversa_di_effettua", ha come dominio la classe "Prenotazione" e come range la classe "Persona" ed è stata posta *functional* per dire che una persona può effettuare al massimo una prenotazione:

  ObjectProperty(a:**inversa_di_effettua** Functional
  inverseOf(a:**effettua**)
  domain(a:**Prenotazione**)
  range(a:**Persona**))

- la relazione "per" ha come dominio la classe "Singola" e come range la classe "Persona" ed è stata posta *functional* per dire che una prenotazione singola può essere riservata al massimo per una persona:

ObjectProperty(a:**per** Functional
inverseOf(a:**inversa_di_per**)
domain(a:**Singola**)
range(a:**Persona**))

La sua relazione inversa, chiamata "inversa_di_per", ha come dominio la classe "Persona" e come range la classe "Singola":

ObjectProperty(a:**inversa_di_per** InverseFunctional
inverseOf(a:**per**)
domain(a:**Persona**)
range(a:**Singola**))

- la relazione "natoA" ha come dominio la classe "Persona" e come range la classe "Comune" ed è stata posta *functional* per dire che una persona può essere nata al massimo in un comune:

  ObjectProperty(a:**natoA** Functional
  inverseOf(a:**inversa_di_natoA**)
  domain(a:**Persona**)
  range(a:**Comune**))

  La sua relazione inversa, chiamata "inversa_di_natoA", ha come dominio la classe "Comune" e come range la classe "Persona":

  ObjectProperty(a:**inversa_di_natoA** InverseFunctional
  inverseOf(a:**natoA**)
  domain(a:**Comune**)
  range(a:**Persona**))

- la relazione "risiede" ha come dominio la classe "Persona" e come range la classe "Comune" ed è stata posta *functional* per dire che una persona può risiedere al massimo in un comune:

  ObjectProperty(a:**risiede** Functional
  inverseOf(a:**inversa_di_risiede**)
  domain(a:**Persona**)
  range(a:**Comune**))

  La sua relazione inversa, chiamata "inversa_di_risiede", ha come dominio la classe "Persona" e come range la classe "Comune":

  ObjectProperty(a:**inversa_di_risiede** InverseFunctional
  inverseOf(a:**risiede**)

domain(a:**Comune**)
range(a:**Persona**))

- la relazione "situato" ha come dominio la classe "Teatro" e come range la classe "Comune" ed è stata posta *functional* per dire che un teatro può essere situato al massimo in un comune:

  ObjectProperty(a:**situato** Functional
  inverseOf(a:**inversa_di_situato**)
  domain(a:**Teatro**)
  range(a:**Comune**))

  La sua relazione inversa, chiamata "inversa_di_situato", ha come dominio la classe "Comune" e come range la classe "Teatro":

  ObjectProperty(a:**inversa_di_situato** InverseFunctional
  inverseOf(a:**situato**)
  domain(a:**Comune**)
  range(a:**Teatro**))

- la relazione "nel" ha come dominio la classe "Posto" e come range la classe "Teatro" ed è stata posta *functional* per dire che un posto può appartenere al massimo ad un teatro:

  ObjectProperty(a:**nel** Functional
  inverseOf(a:**inversa_di_nel**)
  domain(a:**Posto**)
  range(a:**Teatro**))

  La sua relazione inversa, chiamata "inversa_di_nel", ha come dominio la classe "Teatro" e come range la classe "Posto":

  ObjectProperty(a:**inversa_di_nel** InverseFunctional
  inverseOf(a:**nel**)
  domain(a:**Teatro**)
  range(a:**Posto**))

- la relazione "pp" ha come dominio la classe "PostoAssegnato" e come range la classe "Posto" ed è stata posta *functional* per dire che un posto assegnato è relativo al massimo ad un posto:

  ObjectProperty(a:**pp** Functional
  inverseOf(a:**inverse_of_pp**)

domain(a:**PostoAssegnato**)
range(a:**Posto**))

La sua relazione inversa, chiamata "inversa_di_pp", ha come dominio la classe "Posto" e come range la classe "PostoAssegnato" ed *inverseFunctional* in quanto la sua relazione inversa è *functional*:

ObjectProperty(a:**inversa_di_pp** InverseFunctional
inverseOf(a:**pp**)
domain(a:**Posto**)
range(a:**PostoAssegnato**))

- la relazione "psp" ha come dominio la classe "Singola" e come range la classe "PostoAssegnato" ed è stata posta *functional* per dire che una prenotazione singola è relativa al massimo ad un posto assegnato. Inoltre è *inverseFunctional* in quanto la sua relazione inversa è *functional*:

  ObjectProperty(a:**psp** Functional InverseFunctional
  inverseOf(a:**inversa_di_psp**)
  domain(a:**Singola**)
  range(a:**PostoAssegnato**))

  La sua relazione inversa, chiamata "inversa_di_psp", ha come dominio la classe "PostoAssegnato" e come range la classe "Singola" ed è stata posta *functional* per dire che un posto assegnato è relativo al massimo ad una prenotazione singola. Inoltre è *inverseFunctional* in quanto la sua relazione inversa è *functional*:

  ObjectProperty(a:**inversa_di_psp** Functional InverseFunctional
  inverseOf(a:**psp**)
  domain(a:**PostoAssegnato**)
  range(a:**Singola**))

- la relazione "quale" ha come dominio la classe "Collettiva" e come range la classe "Teatro" ed è stata posta *functional* per dire che una prenotazione collettiva riguarda al massimo un teatro:

  ObjectProperty(a:**quale** Functional
  inverseOf(a:**inversa_di_quale**)
  domain(a:**Collettiva**)
  range(a:**Teatro**))

  La sua relazione inversa, chiamata "inversa_di_quale", ha come dominio la classe "Teatro" e come range la classe "Collettiva":

ObjectProperty(a:**inverse_of_quale** InverseFunctional
inverseOf(a:**quale**)
domain(a:**Teatro**)
range(a:**Collettiva**))

QUERY

Le query richieste nel compito non sono realizzabili.Infatti.

1. **Query1**: Calcolare il codice fiscale ed il sesso delle persone che hanno effettuato almeno una prenotazione nel 2003.

   Non possiamo esprimere il vincolo "almeno una prenotazione " in quanto non ci sono operatori che ce lo permettono (infatti sui singoli individui si può utilizzare il solo operatore hasValue e gli operatori <= e >= possono essere applicati solo su relazioni per limitare la cardinalità) . Inoltre non possiamo estrarre gli attributi codice fiscale e sesso dalla classe Persona;

2. **Query2**: Calcolare il codice fiscale e la data di nascita delle persone che hanno effettuato almeno una prenotazione collettiva per una data di agosto, e per la quale hanno pagato una somma di almeno 200 Euro.

   Non possiamo esprimere i vincoli "almeno una prenotazione" e "una somma di almeno 200 Euro " in quanto non ci sono operatori che ce lo permettono. Inoltre non possiamo estrarre gli attributi codice fiscale e data di nascita dalla classe Persona;

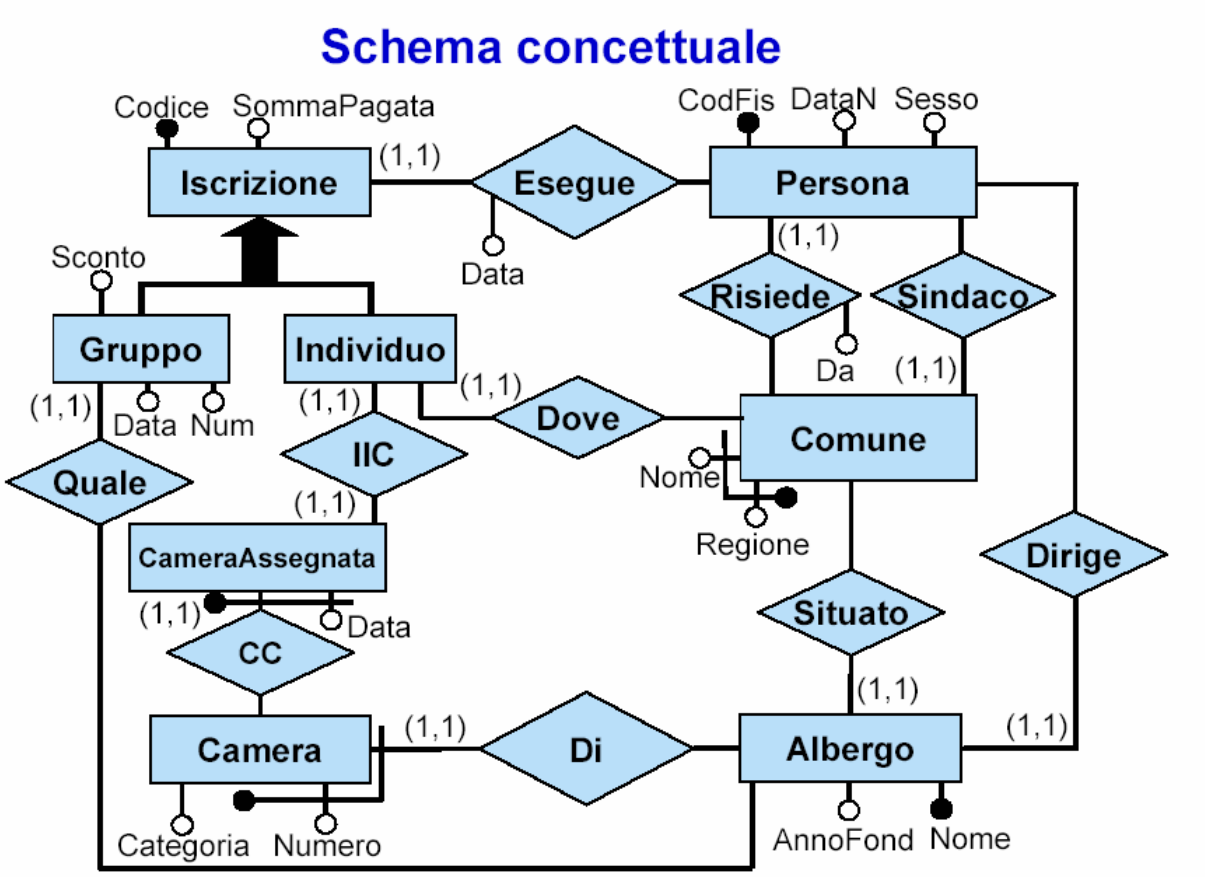
3. **Query3**: Calcolare il comune di residenza e la data di nascita delle donne che hanno effettuato almeno una prenotazione singola per un teatro situato in un comune diverso da quello in cui risiedono.

   Non è una query ad albero, c'è un ciclo e perciò non essendoci variabili in OWL non possiamo esprimere tale query;

4. **Query4**: Per ogni persona che ha effettuato almeno una prenotazione singola per la quale è stato assegnato un posto di categoria 1, contare il numero di prenotazioni (singole o collettive) effettuate.

   Non possiamo esprimere il vincolo "almeno una prenotazione singola" in quanto non ci sono operatori che ce lo permettono. Inoltre non possiamo contare il numero di prenotazioni effettuate in quanto non ci sono operatori che ce lo permettono.

- **COMPITO B del 16/12/04**



La traduzione dello schema concettuale sopra riportato in una ontologia OWL utilizzando Protégé , è stata effettuata traducendo:

- le entità in *classes*;
- le relazioni in *object properties*;
- gli attributi in *datatype properties*.

CLASSI

PERSONA
L' entità "Persona" del diagramma ER, è stata tradotta in una *class* di Protégé.
Essa partecipa alle relazioni "inversa_di_esegue", "risiede", "sindaco" e "dirige" ma, mentre per le relazioni "inversa_di_esegue", "sindaco" e "dirige" vi partecipa con cardinalità 0..*, alla relazione "risiede" vi partecipa con cardinalità 1..1. Bisogna quindi, per quest' ultima relazione, porre una *restriction*, ovvero un vincolo all' interno della classe, per esprimere la cardinalità minima (1) con cui l' entità "Persona" partecipa alla relazione "risiede".

Class(a:**Persona** complete

restriction(a:**risiede** minCardinality(1)))

Complete sta ad indicare che la classe "Persona" è definita, cioè abbiamo almeno una *restriction* tra le condizioni necessarie e sufficienti di questa classe.

Gli **ATTRIBUTI** della classe "Persona" presenti nel diagramma ER, sono stati tradotti in *datatypeProperties* di Protégé.

L' **attributo "codFis"** ha come dominio la classe "Persona" e come range il tipo "string".
Per nostra scelta è stato posto *functional* per indicare la cardinalità 0..1. Non è infatti possibile imporre ai datatype cardinalità 1..1 in quanto non si possono applicare restrizioni sui datatype utilizzando operatori ≥, ≤, =.
Non è stato possibile renderlo chiave della classe in quanto bisognava esprimere tale vincolo ponendo la datatype property *inverseFunctional* e questo non è ammesso usando OWL DL:

DatatypeProperty(a:**Codicefiscale** Functional
domain(a:**Persona**)
range(xsd:**string**))

L' **attributo "dataN"** ha come dominio la classe "Persona" e come range il tipo "date".
Per nostra scelta è stato posto *functional* per indicare la cardinalità 0..1. Non è infatti possibile imporre ai datatype cardinalità 1..1 in quanto non si possono applicare restrizioni sui datatype utilizzando operatori ≥, ≤, =.

DatatypeProperty(a:**dataNascita** Functional
domain(a:**Persona**)
range(xsd:**date**))

L' **attributo "sesso"** ha come dominio la classe "Persona" e come range il tipo "string" i cui valori possono essere solo (owl:oneOf) "maschile" o "femminile".Per nostra scelta è stato posto *functional* per indicare la cardinalità 0..1. Non è infatti possibile imporre ai datatype cardinalità 1..1 in quanto non si possono applicare restrizioni sui datatype utilizzando operatori ≥, ≤ e =, mentre si può imporre quale debba essere l'insieme dei valori ammissibili proprio come avviene per questo attributo:

DatatypeProperty(a:**sesso** Functional
domain(a:**Persona**)
range(**oneOf**("**maschile**"^^<http://www.w3.org/2001/XMLSchema#**string**>
"**femminile**"^^<http://www.w3.org/2001/XMLSchema#**string**>)))

ISCRIZIONE
L' entità "Iscrizione" del diagramma ER, è stata tradotta in una *class* di Protégé.
Essa partecipa alla relazione "esegue"con cardinalità 1..1. Bisogna quindi porre una *restriction*, ovvero un vincolo all' interno della classe, per esprimere la cardinalità minima (1) con cui l' entità "Iscrizione" partecipa alla relazione "esegue".
L' entità "Iscrizione" è, inoltre, una generalizzazione totale delle entità "Gruppo" e "Individuo"; per esplicitare questo è stata creata una *expression* con il tool Protégé che indica che l' unione delle due classi figlie descrive interamente la classe padre "Iscrizione".

Class(a:**Iscrizione** complete
intersectionOf(unionOf(a:**Gruppo** a:**Individuo**) restriction(a:**esegue**
minCardinality(1))))

Complete sta ad indicare che la classe "Iscrizione" è definita, cioè abbiamo almeno una *restriction* tra le condizioni necessarie e sufficienti di questa classe.

Gli **ATTRIBUTI** della classe "Iscrizione" presenti nel diagramma ER, sono stati tradotti in *datatypeProperties* di Protégé.

L' **attributo "codice"** ha come dominio la classe "Iscrizione" e come range il tipo "string". Per nostra scelta è stato posto *functional* per indicare la cardinalità 0..1. Non è infatti possibile imporre ai datatype cardinalità 1..1 in quanto non si possono applicare restrizioni sui datatype utilizzando operatori ≥, ≤, =.
Non è stato possibile renderlo chiave della classe in quanto bisognava esprimere tale vincolo ponendo la datatype property *inverseFunctional* e questo non è ammesso usando OWL DL:

DatatypeProperty(a:**codice** Functional
domain(a:**Iscrizione**)
range(xsd:**string**))

L' **attributo "sommaPagata"** ha come dominio la classe "Iscrizione" e come range il tipo "double".
Per nostra scelta è stato posto *functional* per indicare la cardinalità 0..1. Non è infatti possibile imporre ai datatype cardinalità 1..1 in quanto non si possono applicare restrizioni sui datatype utilizzando operatori ≥, ≤, =.

DatatypeProperty(a:**sommaPagata** Functional
domain(a:**Iscrizione**)
range(xsd:**double**))

GRUPPO
L' entità "Gruppo" del diagramma ER, è stata tradotta in una *class* di Protégé.

Essa partecipa alla relazione "quale"con cardinalità 1..1. Bisogna quindi porre una *restriction*, ovvero un vincolo all' interno della classe, per esprimere la cardinalità minima (1) con cui l' entità "Gruppo" partecipa alla relazione "esegue".
Essendo questa una entità figlia della classe "Iscrizione", erediterà tutte le proprietà e gli attributi della classe padre e inoltre tra le condizioni necessarie Protégé inserirà automaticamente la classe "Iscrizione" per indicare che ogni individuo della classe "Gruppo" è necessariamente un individuo della classe "Iscrizione":

Class(a:**Gruppo** complete
  restriction(a:**quale** minCardinality(1)))
Class(a:**Gruppo** partial
  a:**Iscrizione**)

Complete sta ad indicare che la classe "Gruppo" è definita, cioè abbiamo almeno una *restriction* tra le condizioni necessarie e sufficienti di questa classe.
Partial sta ad indicare che la classe "Gruppo" ha almeno una condizione necessaria.

Secondo la generalizzazione espressa dal diagramma ER, la classe "Gruppo" è disgiunta dalla classe "Individuo", cioè nessun individuo della classe "Gruppo" può appartenere alla classe "Individuo". Questo vincolo, con Protégé, deve essere inserito in entrambe le classi secondo questa sintassi:

DisjointClasses(a:**Individuo** a:**Gruppo**).

Gli **ATTRIBUTI** della classe "Gruppo" presenti nel diagramma ER, sono stati tradotti in *datatypeProperties* di Protégé.

L' **attributo "sconto"** ha come dominio la classe "Gruppo" e come range il tipo "int". Per nostra scelta è stato posto *functional* per indicare la cardinalità 0..1. Non è infatti possibile imporre ai datatype cardinalità 1..1 in quanto non si possono applicare restrizioni sui datatype utilizzando operatori ≥, ≤, =.

DatatypeProperty(a:**sconto** Functional
  domain(a:**Gruppo**)
  range(xsd:**int**))

L' **attributo "data"** ha come dominio la classe "Gruppo" e come range il tipo "date". Per nostra scelta è stato posto *functional* per indicare la cardinalità 0..1. Non è infatti possibile imporre ai datatype cardinalità 1..1 in quanto non si possono applicare restrizioni sui datatype utilizzando operatori ≥, ≤, =.

DatatypeProperty(a:**dataGruppo** Functional
  domain(a:**Gruppo**)
  range(xsd:**date**))

L' **attributo "num"** ha come dominio la classe "Gruppo" e come range il tipo "int".
Per nostra scelta è stato posto *functional* per indicare la cardinalità 0..1. Non è infatti possibile imporre ai datatype cardinalità 1..1 in quanto non si possono applicare restrizioni sui datatype utilizzando operatori ≥, ≤, =.

DatatypeProperty(a:**numeroGruppo** Functional
  domain(a:**Gruppo**)
  range(xsd:**int**))

INDIVIDUO
L' entità "Individuo" del diagramma ER, è stata tradotta in una *class* di Protégé.
Essa partecipa ad entrambe le relazioni "iic" e "dove" con cardinalità 1..1. Bisogna quindi porre due *restrictions*, ovvero due vincoli all' interno della classe, per esprimere la cardinalità minima (1) con cui l' entità "Individuo" partecipa alle due relazioni.
Essendo questa una entità figlia della classe "Iscrizione", erediterà tutte le proprietà e gli attributi della classe padre e inoltre tra le condizioni necessarie Protégé inserirà automaticamente la classe "Iscrizione" per indicare che ogni elemento della classe "Individuo" è necessariamente un elemento della classe "Iscrizione":

Class(a:**Individuo** complete
  intersectionOf(restriction(a:**iic** minCardinality(1)) restriction(a:**dove**
minCardinality(1))))
Class(a:**Individuo** partial
  a:**Iscrizione**)

Complete sta ad indicare che la classe "Individuo" è definita, cioè abbiamo almeno una *restriction* tra le condizioni necessarie e sufficienti di questa classe.
Partial sta ad indicare che la classe "Individuo" ha almeno una condizione necessaria.

Secondo la generalizzazione espressa dal diagramma ER, la classe "Gruppo" è disgiunta dalla classe "Individuo", cioè nessun individuo della classe "Gruppo" può appartenere alla classe "Individuo". Questo vincolo, con Protégé, deve essere inserito in entrambe le classi secondo questa sintassi:

DisjointClasses(a:**Individuo** a:**Gruppo**).

CAMERA ASSEGNATA
L' entità "CameraAssegnata" del diagramma ER, è stata tradotta in una *class* di Protégé.
Essa partecipa ad entrambe le relazioni "inversa_di_iic" e "cc" con cardinalità 1..1. Bisogna quindi porre due *restrictions*, ovvero due vincoli all' interno della classe, per

esprimere la cardinalità minima (1) con cui l' entità "CameraAssegnata" partecipa alle due relazioni:

Class(a: **CameraAssegnata** complete
intersectionOf(restriction(a:**inversa_di_Iic** minCardinality(1)) restriction(a:**cc** minCardinality(1))))

Complete sta ad indicare che la classe "CameraAssegnata" è definita, cioè abbiamo almeno una *restriction* tra le condizioni necessarie e sufficienti di questa classe.

L' **ATTRIBUTO** della classe "CameraAssegnata" presente nel diagramma ER, è stato tradotto in *datatypeProperties* di Protégé.

L' **attributo "data"** ha come dominio la classe "CameraAssegnata" e come range il tipo "date". Per nostra scelta è stato posto *functional* per indicare la cardinalità 0..1. Non è infatti possibile imporre ai datatype cardinalità 1..1 in quanto non si possono applicare restrizioni sui datatype utilizzando operatori ≥, ≤, =.
Non è stato possibile rendere il vincolo di chiave esterna in quanto bisognava esprimere ponendo la datatype property *inverseFunctional* e questo non è ammesso usando OWL DL:

DatatypeProperty(a: **dataCameraAssegnata** Functional
domain(a: **CameraAssegnata**)
range(xsd:**date**))

CAMERA
L' entità "Camera" del diagramma ER, è stata tradotta in una *class* di Protégé.
Essa partecipa alle relazioni "cc" e "di" ma, mentre alla relazione "cc" vi partecipa con cardinalità 0..*, alla relazione "di" vi partecipa con cardinalità 1..1. Bisogna quindi, per quest' ultima relazione, porre una *restriction*, ovvero un vincolo all' interno della classe, per esprimere la cardinalità minima (1)con cui l' entità "Camera" partecipa alla relazione "di":

Class(a:**Camera** complete
restriction(a:**di** minCardinality(1)))

Complete sta ad indicare che la classe "Camera" è definita, cioè abbiamo almeno una *restriction* tra le condizioni necessarie e sufficienti di questa classe.

Gli **ATTRIBUTI** della classe "Camera" presenti nel diagramma ER, sono stati tradotti in *datatypeProperties* di Protégé.

L' **attributo "numero"** ha come dominio la classe "Camera" e come range il tipo "int". Per nostra scelta è stato posto *functional* per indicare la cardinalità 0..1. Non è infatti possibile imporre ai datatype cardinalità 1..1 in quanto non si possono applicare restrizioni sui datatype utilizzando operatori ≥, ≤, =.
Non è stato possibile rendere il vincolo di chiave esterna in quanto bisognava esprimere tale vincolo ponendo la datatype property *inverseFunctional* e questo non è ammesso usando OWL DL:

DatatypeProperty(a:**numeroCamera**Functional
domain(a:**Camera**)
range(xsd:**int**))

L' **attributo "categoria"** ha come dominio la classe "Camera" e come range il tipo "string". Per nostra scelta è stato posto *functional* per indicare la cardinalità 0..1. Non è infatti possibile imporre ai datatype cardinalità 1..1 in quanto non si possono applicare restrizioni sui datatype utilizzando operatori ≥, ≤, =.

DatatypeProperty(a:**categoriaCamera** Functional
domain(a:**Camera**)
range(xsd:**string**))

ALBERGO
L' entità "Albergo" del diagramma ER, è stata tradotta in una *class* di Protégé.
Essa partecipa alle relazioni "inversa_di_di", "inversa_di_quale", "situato" e "dirige" ma, mentre per le relazioni "inversa_di_di" e "inversa_di_quale" vi partecipa con cardinalità 0..*, alla relazioni "situato" e "dirige" vi partecipa con cardinalità 1..1. Bisogna quindi, per quest' ultime relazioni, porre due *restrictions*, ovvero due vincoli all' interno della classe, per esprimere la cardinalità minima (1) con cui l' entità "Albergo" partecipa alle relazioni "situato" e "dirige".

Class(a: **Albergo** complete
intersectionOf(restriction(a:**situato** minCardinality(1))
restriction(a:**inversa_di_dirige** minCardinality(1))))

Complete sta ad indicare che la classe "Albergo" è definita, cioè abbiamo almeno una *restriction* tra le condizioni necessarie e sufficienti di questa classe.

Gli **ATTRIBUTI** della classe "Albergo" presenti nel diagramma ER, sono stati tradotti in *datatypeProperties* di Protégé.

L' **attributo "nome"** ha come dominio la classe "Albergo" e come range il tipo "string".

Per nostra scelta è stato posto *functional* per indicare la cardinalità 0..1. Non è infatti possibile imporre ai datatype cardinalità 1..1 in quanto non si possono applicare restrizioni sui datatype utilizzando operatori ≥, ≤, =.
Non è stato possibile renderlo chiave della classe in quanto bisognava esprimere tale vincolo ponendo la datatype property *inverseFunctional* e questo non è ammesso usando OWL DL:

DatatypeProperty(a:**nomeAlbergo** Functional
domain(a:**Albergo**)
range(xsd:**string**))

L' **attributo "annoFond"** ha come dominio la classe "Albergo" e come range il tipo "string".
Per nostra scelta è stato posto *functional* per indicare la cardinalità 0..1. Non è infatti possibile imporre ai datatype cardinalità 1..1 in quanto non si possono applicare restrizioni sui datatype utilizzando operatori ≥, ≤, =.

DatatypeProperty(a:**annoFondazione** Functional
domain(a:**Albergo**)
range(xsd:**int**))

COMUNE
L' entità "Comune" del diagramma ER, è stata tradotta in una *class* di Protégé.
Essa partecipa alle relazioni "inversa_di_dove", "inversa_di_risiede", "inversa_di_sindaco" e "inversa_di_situato" ma, mentre per le relazioni "inversa_di_dove", "inversa_di_risiede" e "inversa_di_situato" vi partecipa con cardinalità 0..*, alla relazione "inversa_di_sindaco" vi partecipa con cardinalità 1..1. Bisogna quindi, per quest' ultima relazione, porre una *restriction*, ovvero un vincolo all' interno della classe, per esprimere la cardinalità minima (1)con cui l' entità "Comune" partecipa alla relazione "inversa_di_sindaco":

Class(a:**Comune** complete
restriction(a: **inversa_di_sindaco** minCardinality(1)))

Complete sta ad indicare che la classe "Comune" è definita, cioè abbiamo almeno una *restriction* tra le condizioni necessarie e sufficienti di questa classe.

Gli **ATTRIBUTI** della classe "Comune" presenti nel diagramma ER, sono stati tradotti in *datatypeProperties* di Protégé.

L' **attributo "nome"** ha come dominio la classe "Comune" e come range il tipo "string".

Per nostra scelta è stato posto *functional* per indicare la cardinalità 0..1. Non è infatti possibile imporre ai datatype cardinalità 1..1 in quanto non si possono applicare restrizioni sui datatype utilizzando operatori ≥, ≤, =.

DatatypeProperty(a:**nomeComune** Functional
  domain(a:**Comune**)
  range(xsd:**string**))

L' **attributo "regione"** ha come dominio la classe "Comune" e come range il tipo "string".
Per nostra scelta è stato posto *functional* per indicare la cardinalità 0..1. Non è infatti possibile imporre ai datatype cardinalità 1..1 in quanto non si possono applicare restrizioni sui datatype utilizzando operatori ≥, ≤, =.

DatatypeProperty(a:**regione** Functional
  domain(a:**Comune**)
  range(xsd:**string**))

Non è stato possibile rendere il vincolo di chiave esterna in quanto bisognava esprimere tale vincolo con delle variabili così come si fa nella FOL, ma in OWL non esistono variabili.

ESEGUE
La relazione "esegue" è stata tradotta in una *object property* di Protégé.
Ha come dominio la classe "Iscrizione" e come range la classe " Persona ". Per esprimere il limite inferiore della cardinalità 1..1 con cui la classe "Iscrizione" partecipa a questa relazione, è stata posta la restrizione "esegue ≥ 1" nella classe "Iscrizione", cioè indica che una iscrizione è eseguita da almeno una persona, mentre per indicare la cardinalità massima con cui la classe "Iscrizione" partecipa a questa relazione, essa è stata posta a *Functional*, cioè indica che una iscrizione è eseguita da al più una persona :

ObjectProperty(a:**esegue** Functional
  inverseOf(a:**inversa_di_esegue**)
  domain(a:**Iscrizione**)
  range(a:**Persona**))

La relazione inversa, chiamata "inversa_di_esegue", ha come dominio la classe "Persona" e come range la classe " Iscrizione " ed è stata posta *inverseFunctional* in quanto la sua inversa ("esegue") è di tipo *functional* :

ObjectProperty(a:**inversa_di_esegue** InverseFunctional
  inverseOf(a:**esegue**)
  domain(a:**Persona**)

range(a:**Iscrizione**))

Non è stato possibile creare l'attributo "data" della relazione "esegue" in quanto bisognava creare una *datatype property* (rappresentante l'attributo "data") come *subproperty* di una *object property* (rappresentante la relazione "esegue"), ma ciò non è permesso in OWL.

DOVE
La relazione "dove" è stata tradotta in una *object property* di Protégé.
Ha come dominio la classe "Individuo" e come range la classe "Comune". Per esprimere il limite inferiore della cardinalità 1..1 con cui la classe "Individuo" partecipa a questa relazione, è stata posta la restrizione "dove ≥ 1" nella classe "Individuo", cioè indica che una iscrizione individuale è relativa ad almeno un comune, mentre per indicare la cardinalità massima con cui la classe "Individuo" partecipa a questa relazione, essa è stata posta a *Functional*, cioè indica che una iscrizione individuale è relativa ad al più un comune:

ObjectProperty(a:**dove** Functional
inverseOf(a:**inversa_di_dove**)
domain(a:**Individuo**)
range(a:**Comune**))

La sua relazione inversa, chiamata "inversa_di_dove", ha come dominio la classe "Comune" e come range la classe " Individuo " ed è stata posta *inverseFunctional* in quanto la sua inversa ("dove") è di tipo *functional* :

ObjectProperty(a:**inversa_di_dove** InverseFunctional
inverseOf(a:**dove**)
domain(a:**Comune**)
range(a:**Individuo**))

RISIEDE
La relazione "risiede" è stata tradotta in una *object property* di Protégé.
Ha come dominio la classe "Persona" e come range la classe "Comune". Per esprimere il limite inferiore della cardinalità 1..1 con cui la classe "Persona" partecipa a questa relazione, è stata posta la restrizione "risiede ≥ 1" nella classe "Persona", cioè indica che una persona risiede in almeno un comune mentre, per indicare la cardinalità massima con cui la classe "Persona" partecipa a questa relazione, essa è stata posta a *Functional*, cioè indica che una persona risiede in, al più, un comune:

ObjectProperty(a:**risiede** Functional
inverseOf(a:**inversa_di_risiede**)
domain(a:**Persona**)
range(a: **Comune**))

La sua relazione inversa, chiamata "inversa_di_risiede", ha come dominio la classe "Comune" e come range la classe "Persona" ed è stata posta *inverseFunctional* in quanto la sua inversa ("risiede") è di tipo *functional* :

ObjectProperty(a:**inversa_di_risiede** InverseFunctional
inverseOf(a:**risiede**)
domain(a:**Comune**)
range(a:**Persona**))

Non è stato possibile creare l'attributo "da" della relazione "risiede" in quanto bisognava creare una *datatype property* (rappresentante l'attributo "da") come *subproperty* di una *object property* (rappresentante la relazione "risiede"), ma ciò non è permesso in OWL.

SINDACO
La relazione "sindaco" è stata tradotta in una *object property* di Protégé.
Ha come dominio la classe "Persona" e come range la classe "Comune". E' stata posta *inverseFunctional* in quanto la sua inversa ("inversa_di_sindaco") è di tipo *functional*:

ObjectProperty(a:**sindaco** InverseFunctional
inverseOf(a:**inversa_di_ sindaco**)
domain(a:**Persona**)
range(a:**Comune**))

La sua relazione inversa, chiamata "inversa_di_sindaco", ha come dominio la classe "Comune" e come range la classe "Persona". Per esprimere il limite inferiore della cardinalità 1..1 con cui la classe "Comune" partecipa a questa relazione, è stata posta la restrizione "inversa_di_sindaco $\geq$ 1" nella classe "Comune", cioè un comune ha come sindaco almeno una persona mentre per indicare la cardinalità massima con cui la classe "Comune" partecipa a questa relazione, essa è stata posta a *Functional*, cioè indica che un comune ha come sindaco al più una persona:

ObjectProperty(a:**inversa_di_sindaco** Functional
inverseOf(a:**sindaco**)
domain(a:**Comune**)
range(a:**Persona**))

DIRIGE
La relazione "dirige" è stata tradotta in una *object property* di Protégé.
Ha come dominio la classe "Persona" e come range la classe "Albergo". E' stata posta *inverseFunctional* in quanto la sua inversa ("inversa_di_dirige") è di tipo *functional*:

ObjectProperty(a: **dirige** InverseFunctional

inverseOf(a:**inversa_di_dirige**)
domain(a:**Persona**)
range(a:**Albergo**))

La sua relazione inversa, chiamata "inversa_di_ dirige ", ha come dominio la classe "Albergo" e come range la classe "Persona". Per esprimere il limite inferiore della cardinalità 1..1 con cui la classe “Albergo” partecipa a questa relazione, è stata posta la restrizione "inversa_di_dirige ≥ 1" nella classe "Albergo", cioè un albergo ha come direttore almeno una persona, mentre per indicare la cardinalità massima con cui la classe “Albergo” partecipa a questa relazione, essa è stata posta a *Functional*, cioè indica che un albergo ha come direttore al più una persona:

ObjectProperty(a:**inversa_di_dirige** Functional
inverseOf(a:**dirige**)
domain(a:**Albergo**)
range(a:**Persona**))

SITUATO
La relazione "situato" è stata tradotta in una *object property* di Protégé.
Ha come dominio la classe "Albergo" e come range la classe "Comune". Per esprimere il limite inferiore della cardinalità 1..1 con cui la classe “Albergo” partecipa a questa relazione, è stata posta la restrizione "situato ≥ 1" nella classe " Albergo ", cioè indica che un albergo è situato in almeno un comune mentre, per indicare la cardinalità massima con cui la classe “Albergo” partecipa a questa relazione, essa è stata posta a *Functional*, cioè indica che un albergo è situato in, al più, un comune:

ObjectProperty(a:**situato** Functional
inverseOf(a:**inversa_di_situato**)
domain(a:**Albergo**)
range(a:**Comune**))

La sua relazione inversa, chiamata "inversa_di_situato", ha come dominio la classe "Comune" e come range la classe "Albergo" ed è stata posta *inverseFunctional* in quanto la sua inversa (“situato”) è di tipo *functional* :

ObjectProperty(a:**inversa_di_ situato** InverseFunctional
inverseOf(a: **situato**)
domain(a:**Comune**)
range(a: **Albergo**))

DI
La relazione "di" è stata tradotta in una *object property* di Protégé.
Ha come dominio la classe "Camera" e come range la classe " Albergo ". Per esprimere il limite inferiore della cardinalità 1..1 con cui la classe “Camera” partecipa

a questa relazione, è stata posta la restrizione "di ≥ 1" nella classe " Camera ", cioè indica che una camera è di almeno un albergo mentre, per indicare la cardinalità massima con cui la classe "Camera" partecipa a questa relazione, essa è stata posta a *Functional*, cioè indica che una camera è di, al più, un albergo:

ObjectProperty(a:**di** Functional
inverseOf(a:**inversa_di_di**)
domain(a:**Camera**)
range(a: **Albergo**))

La sua relazione inversa, chiamata "inversa_di_di", ha come dominio la classe "Albergo" e come range la classe "Camera" ed è stata posta *inverseFunctional* in quanto la sua inversa ("di") è di tipo *functional* :

ObjectProperty(a:**inversa_di_ di** InverseFunctional
inverseOf(a: **di**)
domain(a: **Albergo**)
range(a: **Camera**))

QUALE
La relazione "quale" è stata tradotta in una *object property* di Protégé.
Ha come dominio la classe "Gruppo" e come range la classe "Albergo". Per esprimere il limite inferiore della cardinalità 1..1 con cui la classe "Gruppo" partecipa a questa relazione, è stata posta la restrizione "quale ≥ 1" nella classe "Gruppo", cioè indica che una prenotazione di gruppo fa riferimento ad almeno un albergo mentre, per indicare la cardinalità massima con cui la classe "Gruppo" partecipa a questa relazione, essa è stata posta a *Functional*, cioè indica che una prenotazione di gruppo fa riferimento ad al più un albergo:

ObjectProperty(a: **quale** Functional
inverseOf(a:**inversa_di_quale**)
domain(a:**Gruppo**)
range(a: **Albergo**))

La sua relazione inversa, chiamata "inversa_di_di", ha come dominio la classe "Albergo" e come range la classe "Camera" ed è stata posta *inverseFunctional* in quanto la sua inversa ("di") è di tipo *functional* :

ObjectProperty(a:**inversa_di_ quale** InverseFunctional
inverseOf(a: **quale**)
domain(a: **Albergo**)
range(a: **Gruppo**))

IIC

La relazione "iic" è stata tradotta in una *object property* di Protégé.
Ha come dominio la classe "Individuo" e come range la classe " CameraAssegnata ". Per esprimere il limite inferiore della cardinalità 1..1 con cui la classe “Individuo” partecipa a questa relazione, è stata posta la restrizione "iic ≥ 1" nella classe " Individuo ", cioè indica che una prenotazione individuale fa riferimento ad almeno una camera assegnata, mentre, per indicare la cardinalità massima con cui la classe “Individuo” partecipa a questa relazione, essa è stata posta a *Functional*, cioè indica che una prenotazione individuale fa riferimento ad al più una camera assegnata. E’ stata inoltre posta anche *inverseFunctional* per indicare che la sua inversa (“inversa_di_iic”) è di tipo functional:

ObjectProperty(a:**iic** Functional InverseFunctional
  inverseOf(a:**inversa_di_Iic**)
  domain(a:**Individuo**)
  range(a:**CameraAssegnata**))

La sua relazione inversa, chiamata "inversa_di_iic", ha come dominio la classe "CameraAssegnata" e come range la classe "Individuo". Per esprimere il limite inferiore della cardinalità 1..1 con cui la classe “CameraAssegnata” partecipa a questa relazione, è stata posta la restrizione "inversa_di_iic ≥ 1" nella classe " CameraAssegnata ", cioè indica che una camera assegnata fa riferimento ad almeno una prenotazione individuale mentre, per indicare la cardinalità massima con cui la classe “CameraAssegnata” partecipa a questa relazione, essa è stata posta a *Functional*, cioè indica che una camera assegnata fa riferimento ad al più una prenotazione individuale. E’ stata inoltre posta anche *inverseFunctional* per indicare che la sua inversa (“iic”) è di tipo functional:

ObjectProperty(a: **inversa_di_Iic** Functional InverseFunctional
  inverseOf(a:**iic**)
  domain(a: **CameraAssegnata**)
  range(a: **Individuo**))

CC
La relazione "cc" è stata tradotta in una *object property* di Protégé.
Ha come dominio la classe "CameraAssegnata" e come range la classe "Camera". Per esprimere il limite inferiore della cardinalità 1..1 con cui la classe “CameraAssegnata” partecipa a questa relazione, è stata posta la restrizione "cc ≥ 1" nella classe "CameraAssegnata ", cioè indica che una camera assegnata è relativa ad almeno una camera mentre, per indicare la cardinalità massima con cui la classe “CameraAssegnata” partecipa a questa relazione, essa è stata posta a *Functional*, cioè indica che una camera assegnata è relativa ad più una camera:

ObjectProperty(a:**cc** Functional
  inverseOf(a:**inversa_di_cc**)

domain(a:**CameraAssegnata**)
range(a:**Camera**))

La sua relazione inversa, chiamata "inversa_di_cc", ha come dominio la classe " Camera " e come range la classe " CameraAssegnata " ed è stata posta *inverseFunctional* in quanto la sua inversa ("cc") è di tipo *functional* :

ObjectProperty(a: **inversa_di_cc** InverseFunctional
inverseOf(a: **cc**)
domain(a: **Camera**)
range(a: **CameraAssegnata**))

E' necessario inoltre specificare che tutti gli individui che appartengono alle singole classi sono diversi tra loro e che tutte le classi sono disgiunte.

DifferentIndividuals(a:Terry a:Emma)

DisjointClasses(a:Camera a:Individuo)
DisjointClasses(a:Individuo a:CameraAssegnata)
DisjointClasses(a:Gruppo a:Comune) [...]

QUERY

Le query richieste nel compito non sono realizzabili.Infatti.

1. **Query1**: Calcolare il codice fiscale ed il sesso delle persone che hanno effettuato almeno una prenotazione nel 2003.

   Non possiamo esprimere il vincolo "almeno una prenotazione " in quanto non ci sono operatori che ce lo permettono (infatti sui singoli individui si può utilizzare il solo operatore hasValue e gli operatori <= e >= possono essere applicati solo su relazioni per limitare la cardinalità) . Inoltre non possiamo estrarre gli attributi codice fiscale e sesso dalla classe Persona;

2. **Query2**: Calcolare il codice fiscale e la data di nascita delle persone che hanno effettuato almeno una prenotazione collettiva per una data di agosto, e per la quale hanno pagato una somma di almeno 200 Euro.

   Non possiamo esprimere i vincoli "almeno una prenotazione" e "una somma di almeno 200 Euro " in quanto non ci sono operatori che ce lo permettono. Inoltre

non possiamo estrarre gli attributi codice fiscale e data di nascita dalla classe Persona;

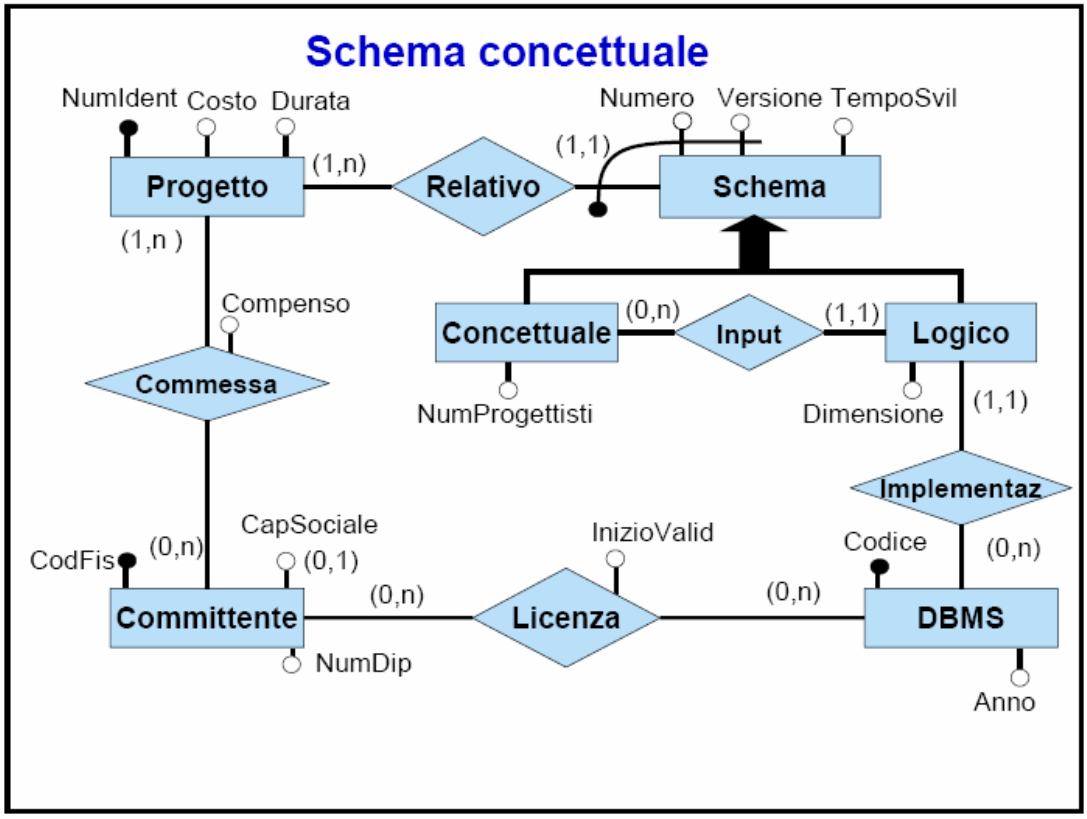
3. **Query3**: Calcolare il comune di residenza e la data di nascita delle donne che hanno effettuato almeno una prenotazione singola per un teatro situato in un comune diverso da quello in cui risiedono.

   Non è una query ad albero, c'è un ciclo e perciò non essendoci variabili in OWL non possiamo esprimere tale query;

4. **Query4**: Per ogni persona che ha effettuato almeno una prenotazione singola per la quale è stato assegnato un posto di categoria 1, contare il numero di prenotazioni (singole o collettive) effettuate.

   Non possiamo esprimere il vincolo "almeno una prenotazione singola" in quanto non ci sono operatori che ce lo permettono. Inoltre non possiamo contare il numero di prenotazioni effettuate in quanto non ci sono operatori che ce lo permettono.

- **COMPITO A del 19/12/02**



La traduzione dello schema concettuale sopra riportato in una ontologia OWL utilizzando Protégé , è stata effettuata traducendo:

- le entità in *classi*;
- le relazioni in object properties;
- gli attributi in datatype properties.

ENTITA'

Non è stato possibile esprimere le chiavi primarie delle varie entità in quanto bisognava esprimere tale vincolo ponendo la datatype property *inverseFunctional* e questo non è ammesso usando OWL DL.
Le classi dell'ontologia, tranne le sottoclassi della gerarchia con la relativa superclasse, sono state poste disjoint a due a due per evitare che gli individui di una classe appartengano ad un'altra classe.

Inoltre tutti gli individui che appartengono alle singole classi sono stati posti differenti tra di loro in quanto OWL non utilizza l'Unique Name Assumption e cioè non assume che gli individui abbiano nome unico.

DifferentIndividuals(a:Terry a:Emma)

Inoltre non è stato possibile specificare il fatto che gli attributi di entità siano funzioni totali, ma sono stati posti *functional* per indicare che ad ogni istanza di un'entità si può associare al massimo un valore del dominio associato.

PROGETTO
L'entità "Progetto" del diagramma ER, è stata tradotta in una *classe* di Protégé.
Essa partecipa alle relazioni "inversa_di_relativo"e "inversa_di_commessa" con cardinalità 1..n. Bisogna quindi, porre una *restriction*, ovvero un vincolo all' interno della classe, per esprimere la cardinalità minima con cui l' entità "Progetto" partecipa alle relazioni "inversa_di_relativo" e "inversa_di_commessa":

Class(a:**Progetto** complete
intersectionOf(restriction(a**:inversa_di_relativo** minCardinality(1))
restriction(a:**inversa_di_commessa** minCardinality(1))))

Complete sta ad indicare che la classe "Progetto" è definita, cioè abbiamo almeno una *restriction* tra le condizioni necessarie e sufficienti di questa classe.
Gli attributi di tale classe sono stati posti *functional*:

DatatypeProperty(a:**numIdent** Functional
domain(a:**Progetto**)
range(xsd:**int**))

DatatypeProperty(a:**costoProgetto** Functional
domain(a:**Progetto**)
range(xsd:**float**))

DatatypeProperty(a:**durataProgetto** Functional
domain(a:**Progetto**)
range(xsd:**time**))

SCHEMA
L'entità "Schema" del diagramma ER, è stata tradotta in una *class* di Protégé.
Essa nel diagramma ER è padre di una generalizzazione in cui le sottoentità sono "Concettuale" e "Logico"; per esplicitare questo sono state inserite le due classi "Concettuale" e "Logico" come sottoclassi di "Schema" .

Il vincolo “complete” della generalizzazione è stato espresso inserendo la *restriction* “Concettuale U Logico” di tipo “necessaria e sufficiente” tra le “asserted conditions” della classe “Schema”.
Essa, inoltre, partecipa alle relazioni “relativo” con cardinalità 1..1. Bisogna quindi, porre una *restriction*, ovvero un vincolo all’ interno della classe, per esprimere la cardinalità minima con cui l’ entità “Schema” partecipa alle relazioni “relativo”:

Class(a:**Schema** complete
intersectionOf(restriction(a:**relativo** minCardinality(1)) unionOf(a:**Concettuale** a:**Logico**)))

Complete sta ad indicare che la classe “Schema” è definita, cioè abbiamo almeno una *restriction* tra le condizioni necessarie e sufficienti di questa classe
Gli attributi di tale classe sono stati posti *functional*:

DatatypeProperty(a:**numeroSchema** Functional
domain(a:**Schema**)
range(xsd:**int**))

DatatypeProperty(a:**versione** Functional
domain(a:**Schema**)
range(xsd:**string**))

DatatypeProperty(a:**tempoDiSviluppo** Functional
domain(a:**Schema**)
range(xsd:**string**))

CONCETTUALE
L’entità “Concettuale” del diagramma ER, è stata tradotta in una *class* di Protégé.
Essendo questa un'entità figlia della classe “Schema”, erediterà tutte le proprietà e gli attributi della classe padre e inoltre tra le condizioni necessarie Protégé inserirà automaticamente la classe “Schema” per indicare che ogni individuo della classe “Concettuale” è necessariamente un individuo della classe “Schema”:

Class(a:**Concettuale** partial a:**Schema**)

Partial sta ad indicare che la classe “Concettuale” ha almeno una condizione necessaria.
Gli attributi di tale classe sono stati posti *functional*:

DatatypeProperty(a:**numProgettisti** Functional
domain(a:**Concettuale**)

range(xsd:**int**))

LOGICO

L' entità "Logico" del diagramma ER, è stata tradotta in una *class* di Protégé.
Essendo questa un'entità figlia della classe "Schema", erediterà tutte le proprietà e gli attributi della classe padre e inoltre tra le condizioni necessarie Protégé inserirà automaticamente la classe "Schema" per indicare che ogni individuo della classe "Logico" è necessariamente un individuo della classe "Schema":

Class(a:**Logico** partial a:**Schema**)

Partial sta ad indicare che la classe "Logico" ha almeno una condizione necessaria.
Essa, inoltre, partecipa alle relazioni "input" e "implementazione" alle quali vi partecipa con cardinalità 1..1. Bisogna quindi, per tali relazione, porre una *restriction*, ovvero un vincolo all'interno della classe, per esprimere la cardinalità minima con cui l'entità "Logico" partecipa alle relazioni "inversa_di_input" e "implementazione":

Class(a:**Logico** complete
intersectionOf(restriction(a:**input** minCardinality(1))
restriction(a:**implementazione** minCardinality(1))))

Complete sta ad indicare che la classe "Logico" è definita, cioè abbiamo almeno una *restriction* tra le condizioni necessarie e sufficienti di questa classe.
La classe "Logico" è disgiunta dalla classe "Concettuale", cioè nessun individuo della classe "Logico" può appartenere alla classe "Concettuale". Questo vincolo, con Protégé, deve essere inserito in entrambe le classi secondo questa sintassi:

DisjointClasses(a:**Concettuale** a:**Logico**).

Gli attributi di tale classe sono stati posti *functional*:

DatatypeProperty(a:**dimSchemaConcettuale** Functional
domain(a:**Logico**)
range(xsd:**int**))

COMMITTENTE

L'entità "Committente" del diagramma ER, è stata tradotta in una *class* di Protégé.
Essa partecipa alle relazioni "commessa" e "licenza" con cardinalità 0..n.

Class(a:**Committente** partial)

Partial sta ad indicare che la classe “Committente” è primitiva cioè ha almeno una condizione necessaria. La condizione necessaria relativa a tale classe è relativa al fatto che "Committente" è sottoclasse di "owl:Thing".
Gli attributi di tale classe sono stati posti *functional*:

DatatypeProperty(a:**codiceFiscale** Functional
domain(a:**Committente**)
range(xsd:**string**))

DatatypeProperty(a:**capitaleSociale** Functional
domain(a:**Committente**)
range(xsd:**float**))

DatatypeProperty(a:**numDipendenti** Functional
domain(a:**Committente**)
range(xsd:**int**))

DBMS
L’entità “Concettuale” del diagramma ER, è stata tradotta in una *class* di Protégé.
Essa partecipa alle relazioni “inversa_di_implementazione” e “inversa_di_licenza” con cardinalità 0..n.

Class(a:**DBMS** partial)

Partial sta ad indicare che la classe “DBMS” ha almeno una condizione necessaria.
Gli attributi di tale classe sono stati posti *functional*:

DatatypeProperty(a:**codiceDBMS** Functional
domain(a:**DBMS**)
range(xsd:**string**))

DatatypeProperty(a:**annoDBMS** Functional
domain(a:**DBMS**)
range(xsd:**int**))

RELAZIONI
L'attributo "compenso" della relazione "commessa" non è stato possibile crearlo in quanto bisognava creare una datatype property (rappresentante l'attributo "compenso") come subproperty di una object property (rappresentante la relazione "commessa"), ma ciò non si può fare in OWL. Lo stesso per l'attributo "inizioValidità" della relazione "licenza".
Per ogni relazione è stata creata la relazione inversa (cioè ha dominio e range invertiti rispetto alla relazione diretta). Infatti:

- la relazione "input" ha come dominio la classe "Concettuale" e come range la classe "Logico" ed è *inverseFunctional* in quanto la sua relazione inversa è "functional".

ObjectProperty(a:**input** InverseFunctional
inverseOf(a:**inversa_di_input**)
domain(a:**Concettuale**)
range(a:**Logico**))

La sua relazione inversa, chiamata "inversa_di_input", ha come dominio la classe "Logico" e come range la classe "Concettuale" ed è stata posta *functional* per dire che uno schema logico ha come input un solo schema concettuale:

ObjectProperty(a:**inverse_of_input** Functional
inverseOf(a:**input**)
domain(a:**Logical**)
range(a:**Conceptual**))

- la relazione "implementazione" ha come dominio la classe "Logico" e come range la classe "DBMS" ed è stata posta *functional* per dire che uno schema logico può essere implementato con al massimo un DBMS:

ObjectProperty(a:**implementazione** Functional
inverseOf(a:**inversa_di_implementazione**)
domain(a:**Logico**)
range(a:**DBMS**))

La sua relazione inversa, chiamata "inversa_di_implementazione", ha come dominio la classe "DBMS" e come range la classe "Logico" ed è "inverseFunctional" in quanto la sua relazione inversa è *functional*:

ObjectProperty(a:**inversa_di_implementazione** InverseFunctional
inverseOf(a:**implementazione**)
domain(a:**DBMS**)
range(a:**Logico**))

- la relazione "commessa" ha come dominio la classe "Committente" e come range la classe "Progetto

ObjectProperty(a:**commessa**
inverseOf(a:**inversa_di_commessa**)
domain(a:**Committente**)
range(a:**Progetto**))

La sua relazione inversa, chiamata "inversa_di_commessa", ha come dominio la classe "Progetto" e come range la classe "Committente":

ObjectProperty(a:**inversa_di commessa**
inverseOf(a:**commessa**)
domain(a:**Progetto**)
range(a:**Committente**))

- la relazione "licenza" ha come dominio la classe "Committente" e come range la classe "DBMS":

  ObjectProperty(a:**licenza**
  inverseOf(a:**inversa_di_licenza**)
  domain(a:**Committente**)
  range(a:**DBMS**))

  La sua relazione inversa, chiamata "inversa_di_licenza", ha come dominio la classe "DBMS" e come range la classe "Committente":

  ObjectProperty(a:**inversa_di_licenza**
  inverseOf(a:**licenza**)
  domain(a:**DBMS**)
  range(a:**Committente**))

- la relazione "relativo" ha come dominio la classe "Schema" e come range la classe "Progetto" ed è stata posta *functional* per dire che uno schema è relativo al massimo ad un Progetto:

  ObjectProperty(a:**relativo** Functional
  inverseOf(a:**inversa_di_relativo**)
  domain(a:**Schema**)
  range(a:**Progetto**))

  La sua relazione inversa, chiamata "inversa_di_relativo", ha come dominio la classe "Progetto" e come range la classe "Schema" ed è *inverseFunctional* in aunto la sua inversa è *functional*:

  ObjectProperty(a:**inversa_di_relativo** InverseFunctional
  inverseOf(a:**relativo**)
  domain(a:**Progetto**)
  range(a:**Schema**))

QUERY

Le query richieste nel compito non sono realizzabili.Infatti.

1. **Query1**: Per ogni schema concettuale che ha richiesto più di 30 giorni per lo sviluppo, si vogliono conoscere i dati relativi al progetto, al numero, e alla versione.

   Non possiamo esprimere il vincolo “più di 30 giorni per lo sviluppo” in quanto non ci sono operatori che ce lo permettono (infatti sui singoli individui si può utilizzare il solo operatore hasValue e gli operatori <= e >= possono essere applicati solo su relazioni per limitare la cardinalità) . Inoltre non possiamo estrarre gli attributi numero e versione da "Progetto";

2. **Query2**: Per ogni schema logico di dimensione maggiore di 100, si vogliono conoscere il tempo di sviluppo, ed il costo del relativo progetto.

   Non possiamo esprimere il vincolo “dimensione maggiore di 100” in quanto non ci sono operatori che ce lo permettono. Inoltre non possiamo estrarre gli attributi "tempo di sviluppo" e "costo" dalla classe Progetto;

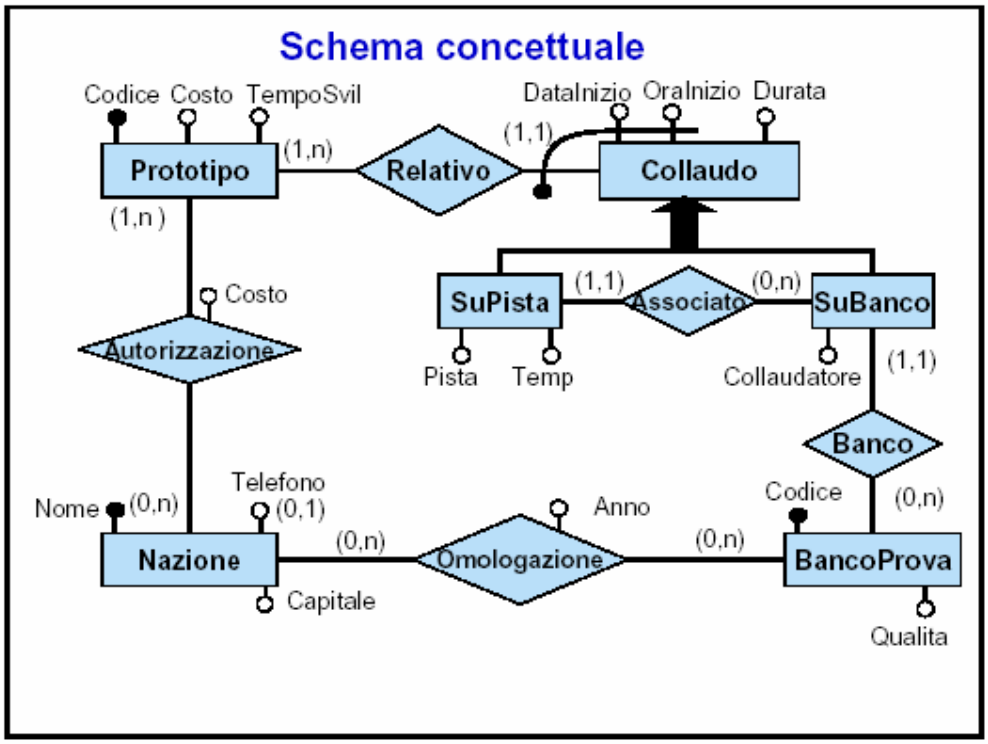
3. **Query3**: Fornire la lista dei progetti per i quali è stato prodotto almeno uno schema logico implementato in un DBMS per il quale almeno un committente del relativo progetto non ha la licenza.

   Non è una query ad albero, c’è un ciclo e perciò non essendoci variabili in OWL non possiamo esprimere tale query;

4. **Query4** Produrre la lista di tutti i progetti il cui costo è inferiore al compenso totale che hanno determinato per l'azienda, dove il compenso totale che un progetto determina per l'azienda è semplicemente la somma dei compensi erogati dai relativi committenti per quel progetto.

   Non possiamo esprimere i vincoli “costo è inferiore al compenso totale” e " la somma dei compensi erogati dai relativi committenti per quel progetto " in quanto non ci sono operatori che ce lo permettono.

- **COMPITO B del 19/12/02**



La traduzione dello schema concettuale sopra riportato in una ontologia OWL utilizzando Protégé , è stata effettuata traducendo:

- le entità in *classes*;
- le relazioni in *object properties*;
- gli attributi in *datatype properties*.

CLASSI

PROTOTIPO
L' entità "Prototipo" del diagramma ER, è stata tradotta in una *class* di Protégé.
Essa partecipa ad entrambe le relazioni "inversa_di_relativo" e "autorizzazione" con cardinalità 1..*. Bisogna quindi, per entrambe le relazioni, porre una *restriction*, ovvero un vincolo all' interno della classe, per esprimere la cardinalità minima (1) con cui l' entità "Prototipo" partecipa alle relazioni:

Class(a:**Prototipo** complete
intersectionOf(restriction(a:**autorizzazione** minCardinality(1))
restriction(a:**inversa_di_relativo** minCardinality(1))))

Complete sta ad indicare che la classe "Prototipo" è definita, cioè abbiamo almeno una *restriction* tra le condizioni necessarie e sufficienti di questa classe.

Gli **ATTRIBUTI** della classe "Prototipo" presenti nel diagramma ER, sono stati tradotti in *datatypeProperties* di Protégé.

L' **attributo "codice"** ha come dominio la classe "Prototipo" e come range il tipo "string".
Per nostra scelta è stato posto *functional* per indicare la cardinalità 0..1. Non è infatti possibile imporre ai datatype cardinalità 1..1 in quanto non si possono applicare restrizioni sui datatype utilizzando operatori ≥, ≤, =.
Non è stato possibile renderlo chiave della classe in quanto bisognava esprimere tale vincolo ponendo la datatype property *inverseFunctional* e questo non è ammesso usando OWL DL:

DatatypeProperty(a:**codicePrototipo** Functional
domain(a:**Prototipo**)
range(xsd:**string**))

L' **attributo "costo"** ha come dominio la classe "Prototipo" e come range il tipo "float".
Per nostra scelta è stato posto *functional* per indicare la cardinalità 0..1. Non è infatti possibile imporre ai datatype cardinalità 1..1 in quanto non si possono applicare restrizioni sui datatype utilizzando operatori ≥, ≤, =.

DatatypeProperty(a:**costoPrototipo** Functional
domain(a:**Prototipo**)
range(xsd:**float**))

L' **attributo "tempoSvil"** ha come dominio la classe "Prototipo" e come range il tipo "int".
Per nostra scelta è stato posto *functional* per indicare la cardinalità 0..1. Non è infatti possibile imporre ai datatype cardinalità 1..1 in quanto non si possono applicare restrizioni sui datatype utilizzando operatori ≥, ≤, =.

DatatypeProperty(a:**tempoSviluppoPrototipo** Functional
domain(a:**Prototipo**)
range(xsd:**int**))

NAZIONE
L' entità "Nazione" del diagramma ER, è stata tradotta in una *class* di Protégé.

Essa partecipa ad entrambe le relazioni "inversa_di_autorizzazione" e "inversa_di_omologazione" con cardinalità 0..*. Non sono necessarie dunque restrizioni:

Class(a:**Nazione** partial)

Partial sta ad indicare che la classe "Nazione" ha almeno una condizione necessaria.
Gli **ATTRIBUTI** della classe "Nazione" presenti nel diagramma ER, sono stati tradotti in *datatypeProperties* di Protégé.

L' **attributo "nome"** ha come dominio la classe "Nazione" e come range il tipo "string".
Per nostra scelta è stato posto *functional* per indicare la cardinalità 0..1. Non è infatti possibile imporre ai datatype cardinalità 1..1 in quanto non si possono applicare restrizioni sui datatype utilizzando operatori ≥, ≤, =.
Non è stato possibile renderlo chiave della classe in quanto bisognava esprimere tale vincolo ponendo la datatype property *inverseFunctional* e questo non è ammesso usando OWL DL:

DatatypeProperty(a:**nomeNazione** Functional
  domain(a:**Nazione**)
  range(xsd:**string**))

L' **attributo "telefono"** ha come dominio la classe "Nazione" e come range il tipo "int".
E' stato posto *functional* per indicare la cardinalità 0..1.

DatatypeProperty(a:**telefono** Functional
  domain(a:**Nazione**)
  range(xsd:**int**))

L' **attributo "capitale"** ha come dominio la classe "Nazione" e come range il tipo "string".
Per nostra scelta è stato posto *functional* per indicare la cardinalità 0..1. Non è infatti possibile imporre ai datatype cardinalità 1..1 in quanto non si possono applicare restrizioni sui datatype utilizzando operatori ≥, ≤, =.

DatatypeProperty(a:**capitale** Functional
  domain(a:**Nazione**)
  range(xsd:**string**))

BANCO PROVA
L' entità "BancoProva" del diagramma ER, è stata tradotta in una *class* di Protégé.

Essa partecipa ad entrambe le relazioni “inversa_di_banco” e “omologazione” con cardinalità 0..*. Non sono necessarie dunque restrizioni:

Class(a:BancoProva partial)

Partial sta ad indicare che la classe “Nazione” ha almeno una condizione necessaria.

Gli **ATTRIBUTI** della classe “BancoProva” presenti nel diagramma ER, sono stati tradotti in *datatypeProperties* di Protégé.

L’ **attributo “codice”** ha come dominio la classe “BancoProva” e come range il tipo “string”.
Per nostra scelta è stato posto *functional* per indicare la cardinalità 0..1. Non è infatti possibile imporre ai datatype cardinalità 1..1 in quanto non si possono applicare restrizioni sui datatype utilizzando operatori ≥, ≤, =.
Non è stato possibile renderlo chiave della classe in quanto bisognava esprimere tale vincolo ponendo la datatype property *inverseFunctional* e questo non è ammesso usando OWL DL:

DatatypeProperty(a: **codiceBancoProva** Functional
  domain(a: **bancoProva**)
  range(xsd:**string**))

L’ **attributo “qualità”** ha come dominio la classe “BancoProva” e come range il tipo “string”.
Per nostra scelta è stato posto *functional* per indicare la cardinalità 0..1. Non è infatti possibile imporre ai datatype cardinalità 1..1 in quanto non si possono applicare restrizioni sui datatype utilizzando operatori ≥, ≤, =.

DatatypeProperty(a:**qualita** Functional
  domain(a:**BancoProva**)
  range(xsd:**string**))

COLLAUDO
L’ entità “Collaudo” del diagramma ER, è stata tradotta in una *class* di Protégé.
Essa partecipa alla relazione “relativo”con cardinalità 1..1. Bisogna quindi porre una *restriction*, ovvero un vincolo all’ interno della classe, per esprimere la cardinalità minima (1) con cui l’ entità “Collaudo” partecipa alla relazione “relativo”.
L’ entità “Iscrizione” è, inoltre, una generalizzazione totale delle entità “SuPista” e “SuBanco”; per esplicitare questo è stata creata una *expression* con il tool Protégé che indica che l’ unione delle due classi figlie descrive interamente la classe padre “Collaudo”.

Class(a:**Collaudo** complete

intersectionOf(unionOf(a: **SuPista** a: **SuBanco)** restriction(a:**relativo** minCardinality(1))))

Complete sta ad indicare che la classe "Collaudo" è definita, cioè abbiamo almeno una *restriction* tra le condizioni necessarie e sufficienti di questa classe.

Gli **ATTRIBUTI** della classe "Collaudo" presenti nel diagramma ER, sono stati tradotti in *datatypeProperties* di Protégé.

L' **attributo "dataInizio"** ha come dominio la classe "Collaudo" e come range il tipo "date".
Per nostra scelta è stato posto *functional* per indicare la cardinalità 0..1. Non è infatti possibile imporre ai datatype cardinalità 1..1 in quanto non si possono applicare restrizioni sui datatype utilizzando operatori ≥, ≤, =.

DatatypeProperty(a:**dataInizioCollaudo** Functional
domain(a:**Collaudo**)
range(xsd:**date**))

L' **attributo "oraInizio"** ha come dominio la classe "Collaudo" e come range il tipo "float".
Per nostra scelta è stato posto *functional* per indicare la cardinalità 0..1. Non è infatti possibile imporre ai datatype cardinalità 1..1 in quanto non si possono applicare restrizioni sui datatype utilizzando operatori ≥, ≤, =.

DatatypeProperty(a:**oraInizioCollaudo** Functional
domain(a:**Collaudo**)
range(xsd:**float**)

Non è stato possibile rendere il vincolo di chiave esterna in quanto bisognava esprimere tale vincolo con delle variabili così come si fa nella FOL, ma in OWL non esistono variabili.

L' **attributo "durata"** ha come dominio la classe "Collaudo" e come range il tipo "float".
Per nostra scelta è stato posto *functional* per indicare la cardinalità 0..1. Non è infatti possibile imporre ai datatype cardinalità 1..1 in quanto non si possono applicare restrizioni sui datatype utilizzando operatori ≥, ≤, =.

DatatypeProperty(a:**durataCollaudo** Functional
domain(a:**Collaudo**)
range(xsd:**float**))

SU PISTA
L' entità "SuPista" del diagramma ER, è stata tradotta in una *class* di Protégé.
Essa partecipa alla relazione "associato" con cardinalità 1..1. Bisogna quindi porre una *restriction*, ovvero un vincolo all' interno della classe, per esprimere la cardinalità minima (1) con cui l' entità "SuPista" partecipa alla relazione.
Essendo questa una entità figlia della classe "Collaudo", erediterà tutte le proprietà e gli attributi della classe padre e inoltre tra le condizioni necessarie Protégé inserirà automaticamente la classe "Collaudo" per indicare che ogni elemento della classe "SuPista" è necessariamente un elemento della classe "Collaudo":

Class(a:**SuPista** complete
 restriction(a:**associato** minCardinality(1)))
 Class(a: **SuPista** partial
 a:**Collaudo**)

Complete sta ad indicare che la classe "SuPista" è definita, cioè abbiamo almeno una *restriction* tra le condizioni necessarie e sufficienti di questa classe.
Partial sta ad indicare che la classe "SuPista" ha almeno una condizione necessaria.

Gli **ATTRIBUTI** della classe "SuPista" presenti nel diagramma ER, sono stati tradotti in *datatypeProperties* di Protégé.

L' **attributo "pista"** ha come dominio la classe "SuPista" e come range il tipo "string".
Per nostra scelta è stato posto *functional* per indicare la cardinalità 0..1. Non è infatti possibile imporre ai datatype cardinalità 1..1 in quanto non si possono applicare restrizioni sui datatype utilizzando operatori ≥, ≤, =.

DatatypeProperty(a: **pista** Functional
 domain(a: **SuPista**)
 range(xsd:**string**))

L' **attributo "temp"** ha come dominio la classe "SuPista" e come range il tipo "float".
Per nostra scelta è stato posto *functional* per indicare la cardinalità 0..1. Non è infatti possibile imporre ai datatype cardinalità 1..1 in quanto non si possono applicare restrizioni sui datatype utilizzando operatori ≥, ≤, =.

DatatypeProperty(a:**temperatura** Functional
 domain(a: **SuPista**)
 range(xsd:**float**))

SU BANCO
L' entità "SuBanco" del diagramma ER, è stata tradotta in una *class* di Protégé.

Essa partecipa alle relazioni “inversa_di_associato” e “banco” ma, mentre per la relazione “inversa_di_associato” vi partecipa con cardinalità 0..*, alla relazione “banco” vi partecipa con cardinalità 1..1. Bisogna quindi, per quest’ ultima relazione, porre una *restriction*, ovvero un vincolo all’ interno della classe, per esprimere la cardinalità minima (1) con cui l’ entità “SuBanco” partecipa alla relazione “banco”.
Essendo questa una entità figlia della classe “Collaudo”, erediterà tutte le proprietà e gli attributi della classe padre e inoltre tra le condizioni necessarie Protégé inserirà automaticamente la classe “Collaudo” per indicare che ogni elemento della classe “SuBanco” è necessariamente un elemento della classe “Collaudo”:

Class(a: SuBanco complete
 restriction(a: banco minCardinality(1)))
Class(a: SuBanco partial
 a: Collaudo)

Complete sta ad indicare che la classe “SuBanco” è definita, cioè abbiamo almeno una *restriction* tra le condizioni necessarie e sufficienti di questa classe.
Partial sta ad indicare che la classe “SuBanco” ha almeno una condizione necessaria.

L’ **ATTRIBUTO** della classe “SuBanco” presente nel diagramma ER, è stato tradotto in *datatypeProperties* di Protégé.

L’ **attributo “collaudatore”** ha come dominio la classe “SuBanco” e come range il tipo “string”.
Per nostra scelta è stato posto *functional* per indicare la cardinalità 0..1. Non è infatti possibile imporre ai datatype cardinalità 1..1 in quanto non si possono applicare restrizioni sui datatype utilizzando operatori ≥, ≤, =.

DatatypeProperty(a: **collaudatore** Functional
 domain(a: **SuBanco**)
 range(xsd:**string**))

RELAZIONI
Per ogni relazione è stata creata la relazione inversa (cioè ha dominio e range invertiti rispetto alla relazione diretta) per permettere la navigazione dello schema in entrambi i versi:

RELATIVO
La relazione "relativo" è stata tradotta in una *object property* di Protégé.
Ha come dominio la classe "Collaudo" e come range la classe " Prototipo ". Per esprimere il limite inferiore della cardinalità 1..1 con cui la classe “Collaudo” partecipa a questa relazione, è stata posta la restrizione " relativo ≥ 1" nella classe " Collaudo ", cioè indica che un collaudo è relativo ad almeno un prototipo, mentre, per indicare la cardinalità massima con cui la classe “Collaudo” partecipa a questa

relazione, essa è stata posta a *Functional*, cioè indica che un collaudo è relativo ad al più un prototipo:

ObjectProperty(a: **relativo** Functional
  inverseOf(a:**inversa_di_ relativo**)
  domain(a: **Collaudo**)
  range(a: **Prototipo**))

La sua relazione inversa, chiamata "inversa_di_relativo", ha come dominio la classe " Prototipo " e come range la classe " Collaudo ". Per esprimere il limite inferiore della cardinalità 1..n con cui la classe “Prototipo” partecipa a questa relazione, è stata posta la restrizione "inversa_di_ relativo ≥ 1" nella classe "Prototipo", cioè indica che un prototipo è relativo ad almeno un collaudo. E’ stata inoltre posta *inverseFunctional* per indicare che la sua inversa (“iic”) è di tipo *functional*:

ObjectProperty(a: **inversa_di_relativo** InverseFunctional
  inverseOf(a: **relativo**)
  domain(a: **Prototipo**)
  range(a: **Collaudo**))

AUTORIZZAZIONE
La relazione "autorizzazione" è stata tradotta in una *object property* di Protégé.
Ha come dominio la classe "Prototipo" e come range la classe "Nazione". Per esprimere il limite inferiore della cardinalità 1..n con cui la classe “Prototipo” partecipa a questa relazione, è stata posta la restrizione " autorizzazione ≥ 1" nella classe " Prototipo ", cioè indica che un prototipo necessita dell’ autorizzazione di almeno una nazione:

ObjectProperty(a: **autorizzazione**
  inverseOf(a:**inversa_di_autorizzazione**)
  domain(a: **Prototipo**)
  range(a: **Nazione**))

La sua relazione inversa, chiamata "inversa_di_autorizzazione", ha come dominio la classe " Nazione " e come range la classe " Prototipo ":

ObjectProperty(a: **inversa_di_autorizzazione**
  inverseOf(a: **autorizzazione**)
  domain(a: **Nazione**)
  range(a: **Prototipo**))

Non è stato possibile creare l'attributo "costo" della relazione "autorizzazione" in quanto bisognava creare una *datatype property* (rappresentante l'attributo "costo")

come *subproperty* di una *object property* (rappresentante la relazione "autorizzazione"), ma ciò non è permesso in OWL.

OMOLOGAZIONE
La relazione "omologazione" è stata tradotta in una *object property* di Protégé.
Ha come dominio la classe "BancoProva" e come range la classe "Nazione".Non presenta cardinalità minime o massime dunque non sono necessarie restrizioni:

ObjectProperty(a: omologazione
 inverseOf(a:**inversa_di_ omologazione**)
 domain(a: **BancoProva**)
 range(a: **Nazione**))

La sua relazione inversa, chiamata " inversa_di_ omologazione ", ha come dominio la classe " Nazione " e come range la classe " BancoProva ":

ObjectProperty(a: **inversa_di_ omologazione**
 inverseOf(a: **omologazione**)
 domain(a: **Nazione**)
 range(a: **BancoProva**))

Non è stato possibile creare l'attributo "anno" della relazione "omologazione" in quanto bisognava creare una *datatype property* (rappresentante l'attributo "anno") come *subproperty* di una *object property* (rappresentante la relazione "omologazione"), ma ciò non è permesso in OWL.

BANCO
La relazione "banco" è stata tradotta in una *object property* di Protégé.
Ha come dominio la classe "SuBanco" e come range la classe "BancoProva". Per esprimere il limite inferiore della cardinalità 1..1 con cui la classe “SuBanco” partecipa a questa relazione, è stata posta la restrizione " banco ≥ 1" nella classe " SuBanco ", cioè indica che un collaudo su banco è relativo ad almeno un banco prova, mentre, per indicare la cardinalità massima con cui la classe “SuBanco” partecipa a questa relazione, essa è stata posta a *Functional*, cioè indica che un collaudo su banco è relativo ad al più un banco prova:

ObjectProperty(a: **banco** Functional
 inverseOf(a:**inversa_di_banco**)
 domain(a: **SuBanco**)
 range(a: **BancoProva**))

La sua relazione inversa, chiamata "inversa_di_ banco", ha come dominio la classe "BancoProva" e come range la classe "SuBanco". E’ stata inoltre posta *inverseFunctional* per indicare che la sua inversa (“banco”) è di tipo *functional*:

ObjectProperty(a: **inversa_di_ banco** InverseFunctional
inverseOf(a: **Banco**)
domain(a: **BancoProva**)
range(a:**SuBanco**))

ASSOCIATO

La relazione "associato" è stata tradotta in una *object property* di Protégé.
Ha come dominio la classe "SuPista" e come range la classe " SuBanco". Per esprimere il limite inferiore della cardinalità 1..1 con cui la classe "SuPista" partecipa a questa relazione, è stata posta la restrizione " associato ≥ 1" nella classe " SuPista ", cioè indica che un collaudo su pista è associato ad almeno un collaudo su banco, mentre, per indicare la cardinalità massima con cui la classe "SuPista" partecipa a questa relazione, essa è stata posta a *Functional*, cioè indica che un collaudo su pista è associato ad al più un collaudo su banco:

ObjectProperty(a: **associato** Functional
inverseOf(a:**inversa_di_ associato**)
domain(a: **SuPista**)
range(a: **SuBanco**))

La sua relazione inversa, chiamata "inversa_di_associato", ha come dominio la classe " SuBanco " e come range la classe " SuPista ". E' stata inoltre posta *inverseFunctional* per indicare che la sua inversa ("associato") è di tipo *functional*:

ObjectProperty(a: **inversa_di_associato** InverseFunctional
inverseOf(a: **SuBanco**)
domain(a: **SuPista**)

E' necessario inoltre specificare che tutti gli individui che appartengono alle singole classi sono diversi tra loro e che tutte le classi sono disgiunte.

DifferentIndividuals(a:Terry a:Emma)

DisjointClasses(a:BancoProva a:SuPista)
DisjointClasses(a:BancoProva a:SuBanco)
DisjointClasses(a:BancoProva a:Collaudo) [...]

QUERY

Le query richieste nel compito non sono realizzabili:

- Query1: Per ogni collaudo più lungo di 60 minuti, si vogliono conoscere il prototipo collaudato e la data e l'ora di inizio del collaudo.

  Non possiamo esprimere il vincolo "più lungo di.."in quanto non ci sono operatori che ce lo permettono.
  Non possiamo estrarre gli attributi data e ora di inizio dalla classe "collaudo".

- Query2: Per ogni collaudo su pista effettuato ad una temperatura minore di 0, si vogliono conoscere la durata ed il tempo di sviluppo del prototipo collaudato.

  Non possiamo esprimere il vincolo "minore di.." in quanto non ci sono operatori che ce lo permettono.
  Non possiamo estrarre gli attributi durata e tempo di sviluppo dalla classe "prototipo".

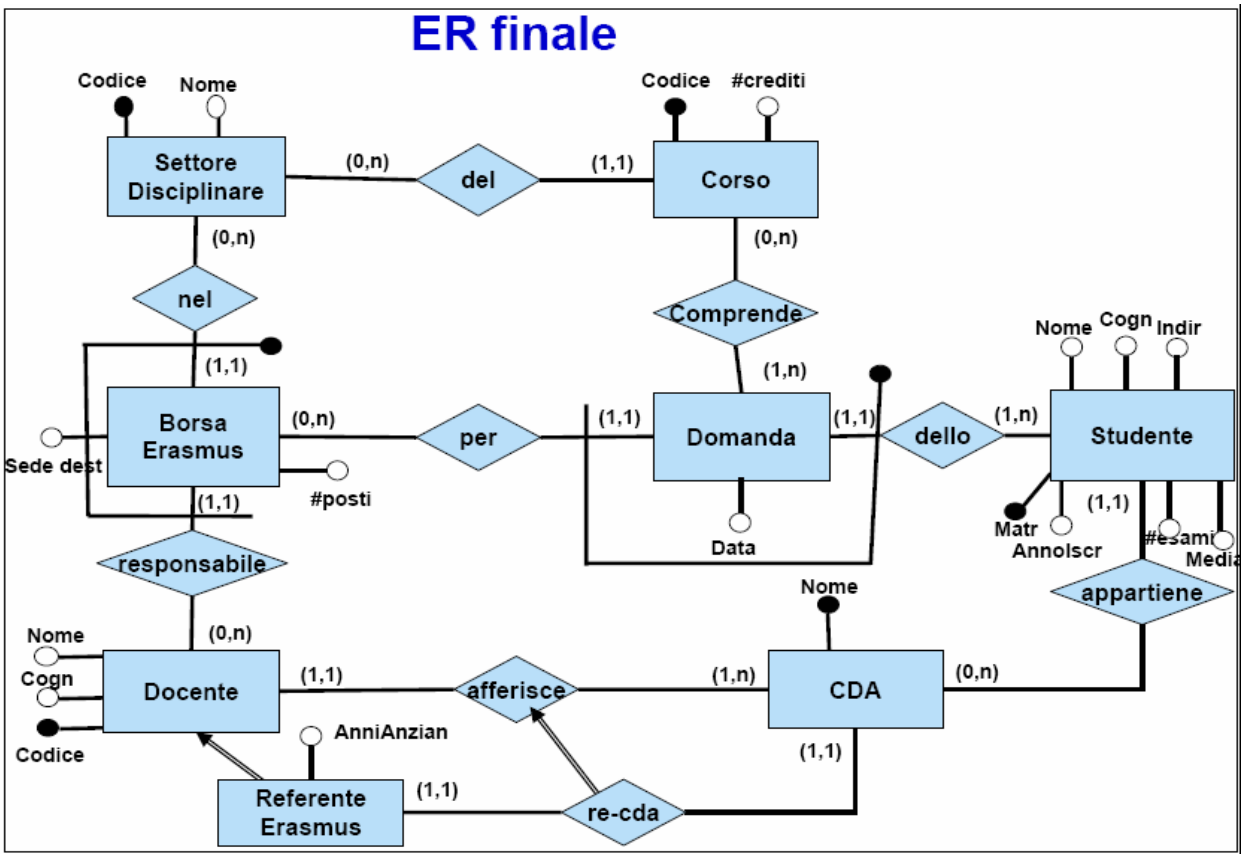
- Query3: Restituire i codici dei prototipi collaudati su almeno un banco che non è omologato in almeno una nazione che ha dato l'autorizzazione al collaudo del prototipo.

  Non è una query ad albero, c'è un ciclo.

- Query4: Produrre la lista di tutti i prototipi per cui il costo totale di autorizzazione del prototipo supera il 5% del costo del prototipo, dove il costo totale di autorizzazione di un prototipo è dato dalla somma di tutti i costi di autorizzazione per quel prototipo nelle varie nazioni.

  Non possiamo esprimere il vincolo "supera il.." in quanto non ci sono operatori che ce lo permettono.
  Non possiamo esprimere la somma di tutti i costi in quanto non ci sono operatori che ce lo permettono.

- **COMPITO A del 19/12/05**



La traduzione dello schema concettuale sopra riportato in una ontologia OWL utilizzando Protégé , è stata effettuata traducendo:

- le entità in *classi*;
- le relazioni in object properties;
- gli attributi in datatype properties.

ENTITA'

Non è stato possibile esprimere le chiavi primarie delle varie entità in quanto bisognava esprimere tale vincolo ponendo la datatype property *inverseFunctional* e questo non è ammesso usando OWL DL.
Le classi dell'ontologia, tranne le sottoclassi della gerarchia con la relativa superclasse, sono state poste disjoint a due a due per evitare che gli individui di una classe appartengano ad un'altra classe.

Inoltre tutti gli individui che appartengono alle singole classi sono stati posti differenti tra di loro in quanto OWL non utilizza l'Unique Name Assumption e cioè non assume che gli individui abbiano nome unico.

DifferentIndividuals(a:Terry a:Emma)

Inoltre non è stato possibile specificare il fatto che gli attributi di entità siano funzioni totali, ma sono stati posti *functional* per indicare che ad ogni istanza di un'entità si può associare al massimo un valore del dominio associato.

SETTORE DISCIPLINARE
L'entità "SettoreDisciplinare" del diagramma ER, è stata tradotta in una *classe* di Protégé. Essa partecipa alle relazioni "del" e "inversa_di_nel" con cardinalità 0..n:

Class(a:**SettoreDisciplinare** partial)

Partial sta ad indicare che la classe "SettoreDisciplinare" è primitiva cioè ha almeno una condizione necessaria.
Gli attributi di tale classe sono stati posti *functional*:

DatatypeProperty(a:**codiceSettoreDisciplinare** Functional
domain(a:**SettoreDisciplinare**)
range(xsd:**string**))

DatatypeProperty(a:**nomeSettoreDisciplinare** Functional
domain(a:**SettoreDisciplinare**)
range(xsd:**string**))

CORSO
L' entità "Corso" del diagramma ER, è stata tradotta in una *classe* di Protégé.
Essa partecipa alle relazioni "comprende" con cardinalità 0..n e alla relazione "inversa_di_del" alla quale vi partecipa con cardinalità 1..1. Bisogna quindi, per quest'ultima relazione, porre una *restriction*, ovvero un vincolo all'interno della classe, per esprimere la cardinalità minima con cui l'entità "Corso" partecipa alle relazioni "inversa_di_del" e "implementazione":

Class(a:**Corso** complete
restriction(a:**inversa_di_del** minCardinality(1)))

Complete sta ad indicare che la classe "Corso" è definita, cioè abbiamo almeno una *restriction* tra le condizioni necessarie e sufficienti di questa classe.
Gli attributi di tale classe sono stati posti *functional*:

DatatypeProperty(a:**codiceCorso** Functional

domain(a:**Corso**)
range(xsd:**string**))

DatatypeProperty(a:**numCrediti** Functional
domain(a:**Corso**)
range(xsd:**int**))

BORSA ERASMUS
L'entità "BorsaErasmus" del diagramma ER, è stata tradotta in una *classe* di Protégé.
Essa partecipa alla relazione "inversa_di_per" con cardinalità 0..n e alle relazioni "nel" e "inversa_di_responsabile" alle quali vi partecipa con cardinalità 1..1. Bisogna quindi, per quest'ultime relazioni, porre una *restriction*, ovvero un vincolo all'interno della classe, per esprimere la cardinalità minima con cui l'entità "BorsaErasmus" partecipa alle relazioni "nel" e "responsabile":

Class(a:**BorsaErasmus** complete
intersectionOf(restriction(a:**nel** minCardinality(1)) restriction(a:**inversa_di responsabile** minCardinality(1))))

Complete sta ad indicare che la classe "Corso" è definita, cioè abbiamo almeno una *restriction* tra le condizioni necessarie e sufficienti di questa classe.
Gli attributi di tale classe sono stati posti *functional*:

DatatypeProperty(a:**numPosti** Functional
domain(a:**BorsaErasmus**)
range(xsd:**int**))

DatatypeProperty(a:**sedeDestinazione** Functional
domain(a:**BorsaErasmus**)
range(xsd:**string**))

DOMANDA
L'entità "Domanda" del diagramma ER, è stata tradotta in una *classe* di Protégé.
Essa partecipa alla relazione "inversa_di_comprende" con cardinalità 1..n, alle relazioni "per" e "dello" con cardinalità 1..1. Bisogna quindi, per tutte le relazioni di tale classe, porre una *restriction*, ovvero un vincolo all'interno della classe, per esprimere la cardinalità minima con cui l'entità "Domanda" partecipa alle relazioni "per" e "dello":

Class(a:**Domanda** complete
intersectionOf(restriction(a:**inversa_di_comprende** minCardinality(1))
restriction(a:**dello** minCardinality(1)) restriction(a:**per** minCardinality(1))))

Complete sta ad indicare che la classe "Corso" è definita, cioè abbiamo almeno una *restriction* tra le condizioni necessarie e sufficienti di questa classe.
Gli attributi di tale classe sono stati posti *functional*:

DatatypeProperty(a:**dataDomanda** Functional
domain(a:**Domanda**)
range(xsd:**date**))

STUDENTE
L'entità "Studente" del diagramma ER, è stata tradotta in una *classe* di Protégé.
Essa partecipa alla relazione "appartiene" con cardinalità 0..n e alle relazioni "dello" e con cardinalità 1..1. Bisogna quindi, per quest'ultima relazione di tale classe, porre una *restriction*, ovvero un vincolo all'interno della classe, per esprimere la cardinalità minima con cui l'entità "Studente" partecipa alle relazioni "inversa_di_dello" e "appartiene":

Class(a:**Studente** complete
intersectionOf(restriction(a**:inversa_di_relativo** minCardinality(1))
restriction(a:**appartiene** minCardinality(1))))

Complete sta ad indicare che la classe "Studente" è definita, cioè abbiamo almeno una *restriction* tra le condizioni necessarie e sufficienti di questa classe.
Gli attributi di tale classe sono stati posti *functional*:

DatatypeProperty(a:nomeStudente Functional
domain(a:**Studente**)
range(xsd:**string**))

DatatypeProperty(a:**cognomeStudente** Functional
domain(a:**Studente**)
range(xsd:**string**))

DatatypeProperty(a:**indirizzoStudente** Functional
domain(a:**Studente**)
range(xsd:**string**))

DatatypeProperty(a:**mediaStudente** Functional
domain(a:**Studente**)
range(xsd:**float**))

DatatypeProperty(a:**numEsami** Functional
domain(a:**Studente**)
range(xsd:**int**))

DatatypeProperty(a:**matricola** Functional
domain(a:**Studente**)
range(xsd:**int**))

DatatypeProperty(a:**annoIscrizione** Functional
domain(a:**Studente**)
range(xsd:**int**))

CDA
L'entità "Studente" del diagramma ER, è stata tradotta in una *classe* di Protégé.
Essa partecipa alle relazioni "inversa_di_afferisce" con cardinalità 1..n e alla relazione "inersa_di_re-cda" con cardinalità 1..1. Bisogna quindi, per tutte le relazioni di tale classe, porre una *restriction*, ovvero un vincolo all'interno della classe, per esprimere la cardinalità minima con cui l'entità "CDA" partecipa alla relazione "inversa_di_afferisce" e "re-cda":

Class(a:**CDA** complete
intersectionOf(restriction(a:**inversa_di_re_cda** minCardinality(1))
restriction(a:**inversa_di_afferisce** minCardinality(1))))

Complete sta ad indicare che la classe "CDA" è definita, cioè abbiamo almeno una *restriction* tra le condizioni necessarie e sufficienti di questa classe.
Gli attributi di tale classe sono stati posti *functional*:

DatatypeProperty(a:**nomeCDA** Functional
domain(a:**CDA**)
range(xsd:**string**))

DOCENTE
L'entità "Docente" del diagramma ER, è stata tradotta in una *classe* di Protégé. Essa è l'entità padre dell'ISA in cui la sottoentità è "ReferenteErasmus"; per esplicitare questo è stata inserita la classe " ReferenteErasmus" come sottoclasse di "Docente" .
Essa partecipa alle relazioni "responsabile" con cardinalità 0..n e alla relazione "afferisce" con cardinalità 1..1. Bisogna quindi, per quest'ultima relazione, porre una *restriction*, ovvero un vincolo all'interno della classe, per esprimere la cardinalità minima con cui l'entità "Docente" partecipa alla relazione "responsabile" e "afferisce":

Class(a:**Docente** complete
restriction(a:**afferisce** minCardinality(1)))

Complete sta ad indicare che la classe "Docente" è definita, cioè abbiamo almeno una *restriction* tra le condizioni necessarie e sufficienti di questa classe.
Gli attributi di tale classe sono stati posti *functional*:

DatatypeProperty(a:**codiceDocente** Functional
domain(a:**Docente**)
range(xsd:**string**))

DatatypeProperty(a:**nomeDocente** Functional
domain(a:**Docente**)
range(xsd:**string**))

DatatypeProperty(a:**cognomeDocente** Functional
domain(a:**Docente**)
range(xsd:**string**))

REFERENTE ERASMUS

L'entità "ReferenteErasmus" del diagramma ER, è stata tradotta in una *classe* di Protégé.
Essendo questa un'entità figlia della classe "Docente", erediterà tutte le proprietà e gli attributi della classe padre e inoltre tra le condizioni necessarie Protégé inserirà automaticamente la classe "Docente" per indicare che ogni individuo della classe "ReferenteErasmus" è necessariamente un individuo della classe "Docente".

Class(a:**ReferenteErasmus** partial a:**Docente**)

Partial sta ad indicare che la classe "ReferenteErasmus" ha almeno una condizione necessaria.
Essa, inoltre, partecipa alla relazione "re-cda" con cardinalità 1..1. Bisogna quindi, per tale relazione, porre una *restriction*, ovvero un vincolo all'interno della classe, per esprimere la cardinalità minima con cui l'entità "ReferenteErasmus" partecipa alla relazione "re-cda":

Class(a:**ReferenteErasmus** complete
restriction(a:**re_cda** minCardinality(1)))

Complete sta ad indicare che la classe "ReferenteErasmus" è definita, cioè abbiamo almeno una *restriction* tra le condizioni necessarie e sufficienti di questa classe.
Gli attributi di tale classe sono stati posti *functional*:

DatatypeProperty(a:**anniAnzianitàRE** Functional
domain(a:**ReferenteErasmus**)
range(xsd:**int**))

RELAZIONI

Per ogni relazione è stata creata la relazione inversa (cioè ha dominio e range invertiti rispetto alla relazione diretta). Infatti:

- la relazione "appartiene" ha come dominio la classe "Studente" e come range la classe "CDA" ed inoltre è *functional* in quanto uno studente può appartenere al massimo ad un CDA:

  ObjectProperty(a:**appartiene** Functional
  inverseOf(a:**inversa_di_appartiene**)
  domain(a:**Studente**)
  range(a:**CDA**))

  La sua relazione inversa, chiamata "inversa_di_appartiene", ha come dominio la classe "CDA" e come range la classe "Studente" ed è *inverseFunctional* per dire che la sua inversa è *functional*:

  ObjectProperty(a:**inversa_di_appartiene** InverseFunctional
  inverseOf(a:**appartiene**)
  domain(a:**CDA**)
  range(a:**Studente**))

- la relazione "per" ha come dominio la classe "Domanda" e come range la classe "BorsaErasmus" ed è stata posta *functional* per dire che una domanda è relativa ad una sola borsa Erasmus:

  ObjectProperty(a:**per** Functional
  inverseOf(a:**inversa_di_per**)
  domain(a:**Domanda**)
  range(a:**BorsaErasmus**))

  La sua relazione inversa, chiamata "inversa_di_per", ha come dominio la classe "BorsaErasmus" e come range la classe "Domanda" ed è *inverseFunctional* in qanto la sua inversa è *functional*:

  ObjectProperty(a:**inversa_di_per** InverseFunctional
  inverseOf(a:**per**)
  domain(a:**BorsaErasmus**)
  range(a:**Domanda**))

- la relazione "comprende" ha come dominio la classe "Corso" e come range la classe "Domanda":

  ObjectProperty(a:**comprende**
  inverseOf(a:**inversa_di_comprende**)

domain(a:**Corso**)
range(a:**Domanda**))

La sua relazione inversa, chiamata "inversa_di_comprende", ha come dominio la classe "Domanda" e come range la classe "Corso":

ObjectProperty(a:**inversa_di_comprende**
inverseOf(a:**comprende**)
domain(a:**Domanda**)
range(a:**Corso**))

- la relazione "nel" ha come dominio la classe "BorsaErasmus" e come range la classe "SettoreDisciplinare" ed è stata posta *functional* per dire che una borsa Erasmus può essere relativa al massimo ad un settore disciplinare:

  ObjectProperty(a:**nel** Functional
  inverseOf(a:**inversa_di_nel**)
  domain(a:**BorsaErasmus**)
  range(a:**SettoreDisciplinare**))

  La sua relazione inversa, chiamata "inversa_di_nel", ha come dominio la classe "SettoreDisciplinare" e come range la classe "BorsaErasmus" ed è *inverseFunctional* in quanto la sua inversa è *functional*:

  ObjectProperty(a:**inversa_di_nel** InverseFunctional
  inverseOf(a:**nel**)
  domain(a:**SettoreDisciplinare**)
  range(a:**BorsaErasmus**))

- la relazione "afferisce" ha come dominio la classe "Docenter" e come range la classe "CDA" ed è stata posta *functional* per dire che un professore può afferire al massimo ad un CDA:

  ObjectProperty(a:**afferisce** Functional
  inverseOf(a:**inverse_of_afferisce**)
  domain(a:**Docente**)
  range(a:**CDA**))

  La sua relazione inversa, chiamata "inversa_di_afferisce", ha come dominio la classe "CDA" e come range la classe "Docente" ed è *inverseFunctional* in quanto la sua inversa è *functional*:

  ObjectProperty(a:**inversa_di_afferisce** InverseFunctional

inverseOf(a:**afferisce**)
domain(a:**CDA**)
range(a:**Docente**))

- la relazione "del" ha come dominio la classe "SettoreDisciplinare" e come range la classe "Corso" ed è stata posta *inverseFunctional* per dire che la sua relazione inversa è *functional*:

ObjectProperty(a:**del** InverseFunctional
inverseOf(a:**inversa_di_del**)
domain(a:**SettoreDisciplinare**)
range(a:**Corso**))

La sua relazione inversa, chiamata "inversa_di_del", ha come dominio la classe "Corso" e come range la classe "SettoreDisciplinare" ed è *functional* in quanto un corso può essere relativo al massimo ad un settore disciplinare:

ObjectProperty(a:**inversa_di_del** Functional
inverseOf(a:**del**)
domain(a:**Corso**)
range(a:**SettoreDisciplinare**))

- la relazione "re-cda" ha come dominio la classe "ReferenteErasmus" e come range la classe "CDA" ed è stata posta *functional* per dire che il referente di un CDA è responsabile di al massimo un CDA. Inoltre tale relazione è anche *InverseFunctional* in quanto la sua relazione inversa è *functional*:

ObjectProperty(a:**re_cda** Functional InverseFunctional
inverseOf(a:**inversa_di_re_cda**)
domain(a:**ReferenteErasmus**)
range(a:**CDA**))

La sua relazione inversa, chiamata "inversa_di_re-cda", ha come dominio la classe "CDA" e come range la classe "ReferenteErasmus" ed è stata posta *functional* per dire che un CDA può avere al massimo un referente.Inoltre tale relazione è anche *inverseFunctional* in quanto la sua relazione inversa è *functional*:

ObjectProperty(a:**inversa_di_re_cda** Functional InverseFunctional
inverseOf(a:**re_cda**)
domain(a:**CDA**)
range(a:**ReferenteErasmus**))
Tale relazione è una subproperty dell'object property "afferisce":

SubPropertyOf(a:**re-cda** a:**afferisce**)
SubPropertyOf(a:**inverse_of_re-cda** a:**inverse_of_afferisce**)

- la relazione "dello" ha come dominio la classe "Domanda" e come range la classe "Studente" ed è stata posta *functional*" per dire che una domanda è relativa al massimo ad uno studente:

  ObjectProperty(a:**relativo** Functional
  inverseOf(a:**inversa_di_relativo**)
  domain(a:**Domanda**)
  range(a:**Studente**))

  La sua relazione inversa, chiamata "inversa_di_dello", ha come dominio la classe "Studente" e come range la classe "Domanda" ed è stata posta *inverseFunctional* per dire che la sua relazione inversa è *functional*:

  ObjectProperty(a:**inversa_di_relativo** InverseFunctional
  inverseOf(a:**relativo**)
  domain(a:**Studente**)
  range(a:**Domanda**))

- la relazione "responsabile" ha come dominio la classe "Docente" e come range la classe "BorsaErasmus" ed è stata posta *inverseFunctional* per dire che la sua relazione inversa è *functional*:

  ObjectProperty(a:**responsabile** InverseFunctional
  inverseOf(a:**inversa_di_responsabile**)
  domain(a:**Docente**)
  range(a:**BorsaErasmus**))

  La sua relazione inversa, chiamata "inversa_di_responsabile", ha come dominio la classe "BorsaErasmus" e come range la classe "Docente" ed è *functional* in quanto una borsa Erasmus può avere al massimo un docente responsabile:

  ObjectProperty(a:**inversa_di_responsabile** Functional
  inverseOf(a:**responsabile**)
  domain(a:**BorsaErasmus**)
  range(a:**Docente**))

QUERY

Le query richieste nel compito non sono realizzabili.Infatti.

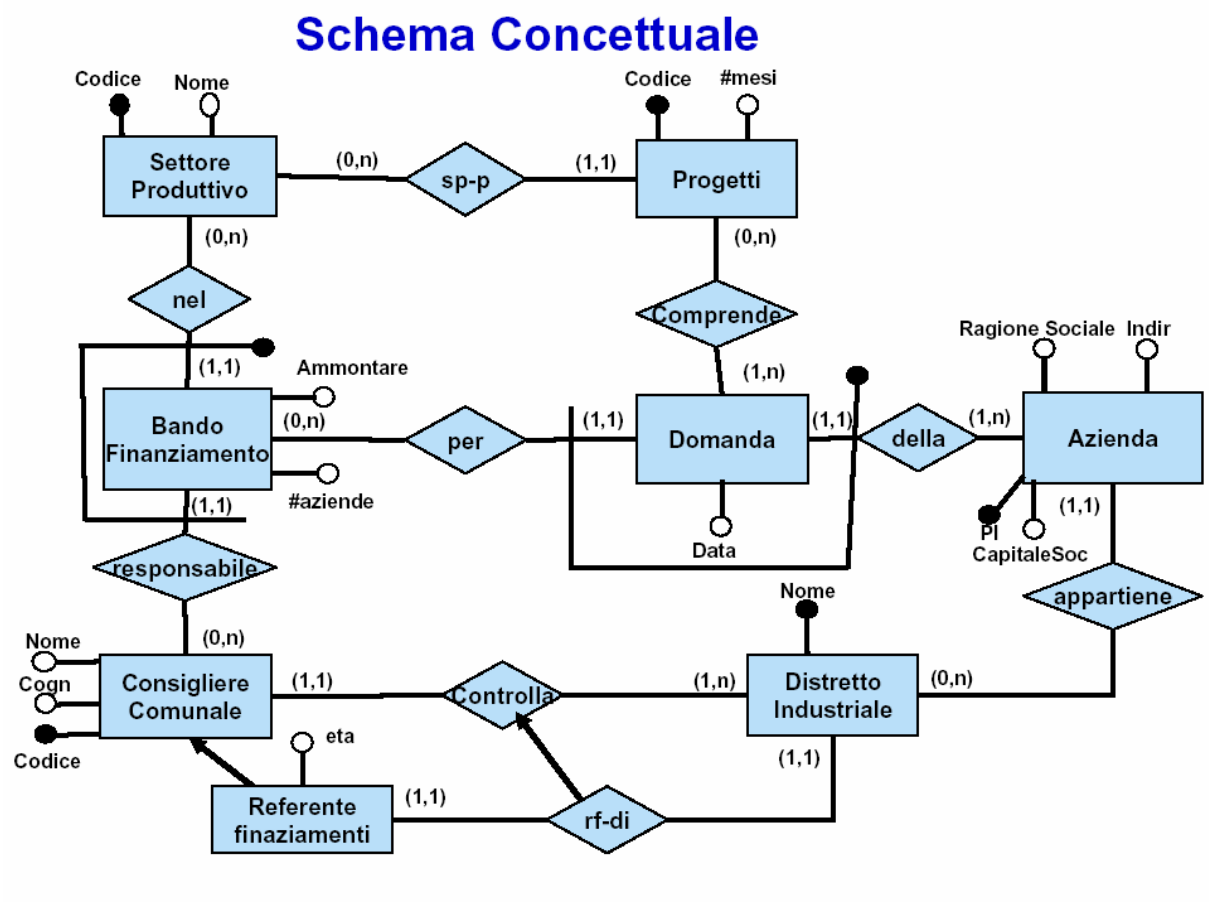
1. **Query1**: Per ogni domanda, restituire la matricola, la media, il numero di esami superati, l'anno di iscrizione ed il consiglio d'area dello studente che l' ha presentata.

   Non possiamo estrarre gli attributi "matricola", "media", "numero di esami superati", "anno di iscrizione" dalla classe "Studente";

2. **Query2**: Restituire le domande per le quali tutti i corsi che lo studente intende frequentare son nel settore disciplinare della borsa alla quale si riferisce la domanda.

   Si tratta di una query non ad albero per cui non può essere realizzata perchè non ci sono variabili in OWL.

- **COMPITO B del 19/12/05**



La traduzione dello schema concettuale sopra riportato in una ontologia OWL utilizzando Protégé , è stata effettuata traducendo:

- le entità in *classes*;
- le relazioni in *object properties*;
- gli attributi in *datatype properties*.

CLASSI

PROGETTI

L' entità "Progetti" del diagramma ER, è stata tradotta in una *class* di Protégé.
Essa partecipa alle relazioni "inversa_di_comprende"e "sp-p" ma, mentre per la relazione "inversa_di_comprende" vi partecipa con cardinalità 0..n, alla relazione "sp-p" vi partecipa con cardinalità 1..1. Bisogna quindi, per quest' ultima relazione, porre una *restriction*, ovvero un vincolo all' interno della classe, per esprimere la cardinalità minima (1) con cui l' entità "Progetti" partecipa alla relazione "sp-p":

Class(a:**Progetti** complete
restriction(<http://www.owl-ontologies.com/unnamed.owl#**sp-p**>
minCardinality(1)))

Complete sta ad indicare che la classe "Progetti" è definita, cioè abbiamo almeno una *restriction* tra le condizioni necessarie e sufficienti di questa classe.

Gli **ATTRIBUTI** della classe "Progetti" presenti nel diagramma ER, sono stati tradotti in *datatypeProperties* di Protégé.

L' **attributo "codice"** ha come dominio la classe "Progetti" e come range il tipo "string".
Per nostra scelta è stato posto *functional* per indicare la cardinalità 0..1. Non è infatti possibile imporre ai datatype cardinalità 1..1 in quanto non si possono applicare restrizioni sui datatype utilizzando operatori ≥, ≤, =.
Non è stato possibile renderlo chiave della classe in quanto bisognava esprimere tale vincolo ponendo la datatype property *inverseFunctional* e questo non è ammesso usando OWL DL:

DatatypeProperty(a:**codiceP** Functional
domain(a: **Progetti**)
range(xsd:**string**))

L' **attributo "numMesi"** ha come dominio la classe "Progetti" e come range il tipo "int".
Per nostra scelta è stato posto *functional* per indicare la cardinalità 0..1. Non è infatti possibile imporre ai datatype cardinalità 1..1 in quanto non si possono applicare restrizioni sui datatype utilizzando operatori ≥, ≤, =.

DatatypeProperty(a: **numMesi** Functional
domain(a: **Progetti**)
range(xsd:**int**))

SETTORE PRODUTTIVO
L' entità "SettoreProduttivo" del diagramma ER, è stata tradotta in una *class* di Protégé.
Essa partecipa ad entrambe le relazioni "inversa_di_nel"e "inversa_di_sp-p" con cardinalità 0..n. Non sono quindi necessarie restrizioni.

Class(a: SettoreProduttivo partial)

Partial sta ad indicare che la classe "SettoreProduttivo" ha almeno una condizione necessaria.

Gli **ATTRIBUTI** della classe "SettoreProduttivo" presenti nel diagramma ER, sono stati tradotti in *datatypeProperties* di Protégé.

L' **attributo "codice"** ha come dominio la classe "SettoreProduttivo" e come range il tipo "string".
Per nostra scelta è stato posto *functional* per indicare la cardinalità 0..1. Non è infatti possibile imporre ai datatype cardinalità 1..1 in quanto non si possono applicare restrizioni sui datatype utilizzando operatori ≥, ≤, =.
Non è stato possibile renderlo chiave della classe in quanto bisognava esprimere tale vincolo ponendo la datatype property *inverseFunctional* e questo non è ammesso usando OWL DL:

DatatypeProperty(a:**codiceSP** Functional
domain(a: **SettoreProduttivo**)
range(xsd:**string**))

L' **attributo "nome"** ha come dominio la classe "SettoreProduttivo" e come range il tipo "string".
Per nostra scelta è stato posto *functional* per indicare la cardinalità 0..1. Non è infatti possibile imporre ai datatype cardinalità 1..1 in quanto non si possono applicare restrizioni sui datatype utilizzando operatori ≥, ≤, =.

DatatypeProperty(a:**nomeSP** Functional
domain(a: **SettoreProduttivo**)
range(xsd:**string**))

BANDO FINANZIAMENTO
L' entità "BandoFinanziamento" del diagramma ER, è stata tradotta in una *class* di Protégé.
Essa partecipa alle relazioni "per", "nel" e "inversa_di_responsabile" ma, mentre per la relazione "per" vi partecipa con cardinalità 0..n, alle relazioni "nel" e "inversa_di_responsabile" vi partecipa con cardinalità 1..1. Bisogna quindi porre due *restrictions*, ovvero due vincoli all' interno della classe, per esprimere la cardinalità minima (1) con cui l' entità "BandoFinanziamento" partecipa alle due relazioni:

Class(a: **BandoFinanziamento** complete
intersectionOf(restriction(a: **inversa_di_responsabile** minCardinality(1))
restriction(a: **nel** minCardinality(1))))

Complete sta ad indicare che la classe "BandoFinanziamento" è definita, cioè abbiamo almeno una *restriction* tra le condizioni necessarie e sufficienti di questa classe.

Gli **ATTRIBUTI** della classe "BandoFinanziamento" presenti nel diagramma ER, sono stati tradotti in *datatypeProperties* di Protégé.

L' **attributo "ammontare"** ha come dominio la classe "BandoFinanziamento" e come range il tipo "float".
Per nostra scelta è stato posto *functional* per indicare la cardinalità 0..1. Non è infatti possibile imporre ai datatype cardinalità 1..1 in quanto non si possono applicare restrizioni sui datatype utilizzando operatori ≥, ≤, =.

DatatypeProperty(a: **ammontare** Functional
domain(a: **BandoFinanziamento**)
range(xsd:**float**))

L' **attributo "numAziende"** ha come dominio la classe "BandoFinanziamento" e come range il tipo "int".
Per nostra scelta è stato posto *functional* per indicare la cardinalità 0..1. Non è infatti possibile imporre ai datatype cardinalità 1..1 in quanto non si possono applicare restrizioni sui datatype utilizzando operatori ≥, ≤, =.

DatatypeProperty(a: **numAziende** Functional
domain(a: **BandoFinanziamento**)
range(xsd:**int**))

Non è stato possibile rendere il vincolo di chiave esterna in quanto bisognava esprimere tale vincolo con delle variabili così come si fa nella FOL, ma in OWL non esistono variabili.

CONSIGLIERE COMUNALE
L' entità "ConsigliereComunale" del diagramma ER, è stata tradotta in una *class* di Protégé.
Essa partecipa alle relazioni "responsabile" e "controlla", ma mentre per la relazione "responsabile" vi partecipa con cardinalità 0..n, alla relazione "controlla" vi partecipa con cardinalità 1..1. Bisogna quindi porre una *restriction*, ovvero un vincolo all' interno della classe, per esprimere la cardinalità minima (1) con cui l' entità "ConsiglierComunale" partecipa alla relazione.
L' entità "ConsiglierComunale" è, inoltre, una generalizzazione totale dell' entità "ReferenteFinanziamenti":

class(a: **ConsiglierComunale** complete
restriction(a: **controlla** minCardinality(1)))

Complete sta ad indicare che la classe "ConsiglierComunale" è definita, cioè abbiamo almeno una *restriction* tra le condizioni necessarie e sufficienti di questa classe.

Gli **ATTRIBUTI** della classe "ConsiglieComunale" presenti nel diagramma ER, sono stati tradotti in *datatypeProperties* di Protégé.

L' **attributo "codice"** ha come dominio la classe "ConsiglieComunale" e come range il tipo "string".
Per nostra scelta è stato posto *functional* per indicare la cardinalità 0..1. Non è infatti possibile imporre ai datatype cardinalità 1..1 in quanto non si possono applicare restrizioni sui datatype utilizzando operatori ≥, ≤, =.
Non è stato possibile renderlo chiave della classe in quanto bisognava esprimere tale vincolo ponendo la datatype property *inverseFunctional* e questo non è ammesso usando OWL DL:

DatatypeProperty(a:**codiceCC** Functional
domain(a: **ConsiglieComunale**)
range(xsd:**string**))

L' **attributo "nome"** ha come dominio la classe "ConsiglieComunale" e come range il tipo "string".
Per nostra scelta è stato posto *functional* per indicare la cardinalità 0..1. Non è infatti possibile imporre ai datatype cardinalità 1..1 in quanto non si possono applicare restrizioni sui datatype utilizzando operatori ≥, ≤, =.

DatatypeProperty(a:**nomeCC** Functional
domain(a: **ConsiglieComunale**)
range(xsd:**string**))

L' **attributo "cognome"** ha come dominio la classe "ConsiglieComunale" e come range il tipo "string".
Per nostra scelta è stato posto *functional* per indicare la cardinalità 0..1. Non è infatti possibile imporre ai datatype cardinalità 1..1 in quanto non si possono applicare restrizioni sui datatype utilizzando operatori ≥, ≤, =.

DatatypeProperty(a:**cognomeCC** Functional
domain(a: **ConsiglieComunale**)
range(xsd:**string**))

REFERENTE FINANZIAMENTI
L' entità "ReferenteFinanziamenti" del diagramma ER, è stata tradotta in una *class* di Protégé.
Essa partecipa alla relazione "rf-di"con cardinalità 1..1. Bisogna quindi porre una *restriction*, ovvero un vincolo all' interno della classe, per esprimere la cardinalità minima (1) con cui l' entità "ReferenteFinanziamenti" partecipa alla relazione "rf-di".
Essendo questa una entità figlia della classe "ConsiglieComunale", erediterà tutte le proprietà e gli attributi della classe padre:

Class(a: **ReferenteFinanziamenti** complete
restriction(<http://www.owl-ontologies.com/unnamed.owl#**rf-di**>
minCardinality(1)))
Class(a: **ReferenteFinanziamenti** partial
a: **ConsigliereComunale**)

Complete sta ad indicare che la classe "ReferenteFinanziamenti" è definita, cioè abbiamo almeno una *restriction* tra le condizioni necessarie e sufficienti di questa classe.
Partial sta ad indicare che la classe "ReferenteFinanziamenti" ha almeno una condizione necessaria.

Gli **ATTRIBUTI** della classe "ReferenteFinanziamenti" presenti nel diagramma ER, sono stati tradotti in *datatypeProperties* di Protégé.

L' **attributo "età"** ha come dominio la classe "ReferenteFinanziamenti" e come range il tipo "int".
Per nostra scelta è stato posto *functional* per indicare la cardinalità 0..1. Non è infatti possibile imporre ai datatype cardinalità 1..1 in quanto non si possono applicare restrizioni sui datatype utilizzando operatori ≥, ≤, =.

DatatypeProperty(a: **età** Functional
domain(a: **ReferenteFinanziamenti**)
range(xsd:**int**))

DISTRETTO INDUSTRIALE
L' entità "DistrettoIndistriale" del diagramma ER, è stata tradotta in una *class* di Protégé.
Essa partecipa alle relazioni "inversa_di_appartiene", "inversa_di_controlla"e "inversa_di_rf-di" ma, mentre per la relazione "inversa_di_ appartiene" vi partecipa con cardinalità 0..n, alla relazione "inversa_di_controlla" vi partecipa con cardinalità 1..n e alla relazione "inversa_di_rf-di" vi partecipa con cardinalità 1..1. Bisogna quindi, per queste ultime due relazioni, porre due *restrictions*, ovvero due vincoli all' interno della classe, per esprimere la cardinalità minima (1) con cui l' entità "DistrettoIndistriale" partecipa alle relazioni "inversa_di_controlla" e "inversa_di_rf-di":

Class(a: **DistrettoIndistriale** complete
intersectionOf(restriction(a: **inversa_di_controlla** minCardinality(1))
restriction(<http://www.owl-ontologies.com/unnamed.owl#**inversa_di_rf-di**>
minCardinality(1))))

L' **ATTRIBUTIO** della classe "DistrettoIndistriale" presente nel diagramma ER, è stato tradotto in *datatypeProperties* di Protégé.

L' **attributo "nome"** ha come dominio la classe "DistrettoIndistriale" e come range il tipo "string".
Per nostra scelta è stato posto *functional* per indicare la cardinalità 0..1. Non è infatti possibile imporre ai datatype cardinalità 1..1 in quanto non si possono applicare restrizioni sui datatype utilizzando operatori ≥, ≤, =.
Non è stato possibile renderlo chiave della classe in quanto bisognava esprimere tale vincolo ponendo la datatype property *inverseFunctional* e questo non è ammesso usando OWL DL:

DatatypeProperty(a:**nomeDI** Functional
domain(a: **DistrettoIndistriale**)
range(xsd:**string**))

AZIENDA
L' entità "Azienda" del diagramma ER, è stata tradotta in una *class* di Protégé.
Essa partecipa alla relazione "inversa_di_della" con cardinalità 1..* e alla relazione "appartiene" con cardinalità 1..1. Bisogna quindi, per entrambe le relazioni, porre una *restriction*, ovvero un vincolo all' interno della classe, per esprimere la cardinalità minima (1) con cui l' entità "Azienda" partecipa alle relazioni:

Class(a: **Azienda** complete
intersectionOf(restriction(a: **appartiene** minCardinality(1)) restriction(a: **inversa_di_della** minCardinality(1))))

Gli **ATTRIBUTI** della classe "Azienda" presenti nel diagramma ER, sono stati tradotti in *datatypeProperties* di Protégé.

L' **attributo "PI"** ha come dominio la classe "Azienda" e come range il tipo "float".
Per nostra scelta è stato posto *functional* per indicare la cardinalità 0..1. Non è infatti possibile imporre ai datatype cardinalità 1..1 in quanto non si possono applicare restrizioni sui datatype utilizzando operatori ≥, ≤, =.
Non è stato possibile renderlo chiave della classe in quanto bisognava esprimere tale vincolo ponendo la datatype property *inverseFunctional* e questo non è ammesso usando OWL DL:

DatatypeProperty(a:**pi** Functional
domain(a: **Azienda**)
range(xsd:**float**))

L' **attributo "CapitaleSoc"** ha come dominio la classe "Azienda" e come range il tipo "float".

Per nostra scelta è stato posto *functional* per indicare la cardinalità 0..1. Non è infatti possibile imporre ai datatype cardinalità 1..1 in quanto non si possono applicare restrizioni sui datatype utilizzando operatori ≥, ≤, =:

DatatypeProperty(a: **CapitaleSoc** Functional
domain(a: **Azienda**)
range(xsd:**float**))

L' **attributo "RagioneSoc"** ha come dominio la classe "Azienda" e come range il tipo "string".
Per nostra scelta è stato posto *functional* per indicare la cardinalità 0..1. Non è infatti possibile imporre ai datatype cardinalità 1..1 in quanto non si possono applicare restrizioni sui datatype utilizzando operatori ≥, ≤, =:

DatatypeProperty(a: **RagioneSoc** Functional
domain(a: **Azienda**)
range(xsd:**string**))

L' **attributo "indirizzo"** ha come dominio la classe "Azienda" e come range il tipo "string".
Per nostra scelta è stato posto *functional* per indicare la cardinalità 0..1. Non è infatti possibile imporre ai datatype cardinalità 1..1 in quanto non si possono applicare restrizioni sui datatype utilizzando operatori ≥, ≤, =:

DatatypeProperty(a: **indirizzo** Functional
domain(a: **Azienda**)
range(xsd:**string**))

DOMANDA
L' entità "Domanda" del diagramma ER, è stata tradotta in una *class* di Protégé.
Essa partecipa alla relazione "comprende" con cardinalità 1..* e alle relazioni "per" e "della" con cardinalità 1..1. Bisogna quindi, per tutte e tre le relazioni, porre una *restriction*, ovvero un vincolo all' interno della classe, per esprimere la cardinalità minima (1) con cui l' entità "Domanda" partecipa alle relazioni:

Class(a: **Domanda** complete
intersectionOf(restriction(a: **comprende** minCardinality(1)) restriction(a: **per** cardinality(1)) restriction(a: **della** cardinality(1))))

L' **ATTRIBUTIO** della classe "Domanda" presente nel diagramma ER, è stato tradotto in *datatypeProperties* di Protégé.

L'attributo **"data"** ha come dominio la classe "Domanda" e come range il tipo "date".

Per nostra scelta è stato posto *functional* per indicare la cardinalità 0..1. Non è infatti possibile imporre ai datatype cardinalità 1..1 in quanto non si possono applicare restrizioni sui datatype utilizzando operatori ≥, ≤, =.

DatatypeProperty(a:**dataDomanda** Functional
  domain(a:**Domanda**)
  range(xsd:**date**))

Non è stato possibile rendere il vincolo di chiave esterna in quanto bisognava esprimere tale vincolo con delle variabili così come si fa nella FOL, ma in OWL non esistono variabili.

RELAZIONI
Per ogni relazione è stata creata la relazione inversa (cioè ha dominio e range invertiti rispetto alla relazione diretta) per permettere la navigazione dello schema in entrambi i versi:

SP-P
La relazione "sp-p" è stata tradotta in una *object property* di Protégé.
Ha come dominio la classe "Progetti" e come range la classe "SettoreProduttivo". Per esprimere il limite inferiore della cardinalità 1..1 con cui la classe "Progetti" partecipa a questa relazione, è stata posta la restrizione " sp-p ≥ 1" nella classe " Progetti ", cioè indica che un progetto è relativo ad almeno un settore produttivo, mentre
per indicare la cardinalità massima con cui la classe "Progetti" partecipa a questa relazione, essa è stata posta a *Functional*, cioè indica che un progetto è relativo ad al più un settore produttivo:

ObjectProperty(<http://www.owl-ontologies.com/unnamed.owl#**sp-p**> Functional
  inverseOf(<http://www.owl-ontologies.com/unnamed.owl#**inversa_di_sp-p**>)
  domain(a: **Progetti**)
  range(a: **SettoreProduttivo**))

La relazione inversa, chiamata "inversa_di_sp-p", ha come dominio la classe " SettoreProduttivo " e come range la classe " Progetti" ed è stata posta *inverseFunctional* in quanto la sua inversa ("sp-p") è di tipo *functional* :

ObjectProperty(<http://www.owl-ontologies.com/unnamed.owl#**inversa_di_sp-p**>
InverseFunctional
  inverseOf(<http://www.owl-ontologies.com/unnamed.owl#**sp-p**>)
  domain(a: **SettoreProduttivo**)
  range(a: **Progetti**))

NEL
La relazione "nel" è stata tradotta in una *object property* di Protégé.

Ha come dominio la classe "BandoFinanziamento" e come range la classe "SettoreProduttivo". Per esprimere il limite inferiore della cardinalità 1..1 con cui la classe “BandoFinanziamento” partecipa a questa relazione, è stata posta la restrizione " nel ≥ 1" nella classe " BandoFinanziamento ", cioè indica che un bando finanziamento è in almeno un settore produttivo, mentre per indicare la cardinalità massima con cui la classe “BandoFinanziamento” partecipa a questa relazione, essa è stata posta a *Functional*, cioè indica che un bando finanziamento è in al più un settore produttivo:

ObjectProperty(a: **nel** Functional
inverseOf(a:**inversa_di_nel**)
domain(a: **BandoFinanziamento**)
range(a: **SettoreProduttivo**))

La relazione inversa, chiamata "inversa_di_nel", ha come dominio la classe "SettoreProduttivo " e come range la classe " BandoFinanziamento " ed è stata posta *inverseFunctional* in quanto la sua inversa (“nel”) è di tipo *functional* :

ObjectProperty(a: **inversa_di_nel** InverseFunctional
inverseOf(a: **nel**)
domain(a: **SettoreProduttivo**)
range(a: **BandoFinanziamento**))

RESPONSABILE
La relazione "responsabile" è stata tradotta in una *object property* di Protégé.
Ha come dominio la classe "ConsiglieреComunale" e come range la classe " BandoFinanziamento ". Essa è stata posta a *inverseFunctional*, in quanto la sua inversa (“inversa_di_responsabile”) è di tipo *functional*:

ObjectProperty(a: **responsabile** InverseFunctional
inverseOf(a:**inversa_di_ responsabile**)
domain(a: **ConsiglieреComunale**)
range(a: **BandoFinanziamento**))

La relazione inversa, chiamata "inversa_di_responsabile", ha come dominio la classe " BandoFinanziamento " e come range la classe " ConsiglieреComunale ". Per esprimere il limite inferiore della cardinalità 1..1 con cui la classe “BandoFinanziamento” partecipa a questa relazione, è stata posta la restrizione " inversa_di_responsabile ≥ 1" nella classe " BandoFinanziamento ", cioè indica che un bando finanziamento è relativo ad almeno un consigliere comunale, mentre per indicare la cardinalità massima con cui la classe “BandoFinanziamento” partecipa a questa relazione, essa è stata posta a *Functional*, cioè indica che un bando finanziamento è relativo ad al più un consigliere comunale:

ObjectProperty(a: **inversa_di_ responsabile** Functional
inverseOf(a: **responsabile**)
domain(a: **BandoFinanziamento**)
range(a: **ConsiglieреComunale**))

CONTROLLA
La relazione "controlla" è stata tradotta in una *object property* di Protégé ed è proprietà padre della relazione "rf-di",
Ha come dominio la classe " ConsiglieреComunale " e come range la classe " DistrettoIndustriale ". Per esprimere il limite inferiore della cardinalità 1..1 con cui la classe "ConsiglieреComunale" partecipa a questa relazione, è stata posta la restrizione " controlla ≥ 1" nella classe " ConsiglieреComunale ", cioè indica che un consigliere comunale controlla almeno un distretto industriale, mentre, per indicare la cardinalità massima con cui la classe "ConsiglieреComunale" partecipa a questa relazione, essa è stata posta a *Functional*, cioè indica che un consiglere comunale controlla al più un distretto industriale:

ObjectProperty(a: **controlla** Functional
inverseOf(a:**inversa_di_controlla**)
domain(a: **ConsiglieреComunale**)
range(a: **DistrettoIndustriale**))

La sua relazione inversa, chiamata "inversa_di_controlla", ha come dominio la classe " DistrettoIndustriale " e come range la classe " ConsiglieреComunale ". Per esprimere il limite inferiore della cardinalità 1..n con cui la classe "DistrettoIndustriale" partecipa a questa relazione, è stata posta la restrizione "inversa_di_controlla ≥ 1" nella classe " DistrettoIndustriale ", cioè indica che un distretto industriale è controllato da almeno un consiglere comunale. E' stata inoltre posta *inverseFunctional* per indicare che la sua inversa ("controlla") è di tipo *functional*:

ObjectProperty(a: **inversa_di_controlla** InverseFunctional
inverseOf(a: **controlla**)
domain(a: **DistrettoIndustriale**)
range(a: **ConsiglieреComunale**))

RF-DI
La relazione "rf-di" è stata tradotta in una *object property* di Protégé.
Ha come dominio la classe "ReferenteFinanziamenti" e come range la classe "DistrettoIndustriale" che eredita dalla relazione padre" Controlla". Per esprimere il limite inferiore della cardinalità 1..1 con cui la classe "ReferenteFinanziamenti" partecipa a questa relazione, è stata posta la restrizione " rf-di ≥ 1" nella classe " ReferenteFinanziamenti ", cioè indica che un referente finanziamenti fa riferimento ad almeno un distretto industriale, mentre, per indicare la cardinalità massima con cui la

classe “ReferenteFinanziamenti” partecipa a questa relazione, essa è stata posta a *Functional*, cioè indica che un referente finanziamenti fa riferimento ad al più un distretto industriale. E’ stata inoltre posta anche *inverseFunctional* per indicare che la sua inversa (“inversa_di_rf-di”) è di tipo functional:

ObjectProperty(<http://www.owl-ontologies.com/unnamed.owl#**rf-di**> Functional InverseFunctional
inverseOf(<http://www.owl-ontologies.com/unnamed.owl#**inversa_di_rf-di**>)
domain(a: **ReferenteFinanziamenti**))

La sua relazione inversa, chiamata "inversa_di_rf-di", ha come dominio la classe " DistrettoIndustriale " e come range la classe " ReferenteFinanziamenti ". Per esprimere il limite inferiore della cardinalità 1..1 con cui la classe “DistrettoIndustriale” partecipa a questa relazione, è stata posta la restrizione "inversa_di_rf-di ≥ 1" nella classe " DistrettoIndustriale ", cioè indica che un distretto industriale fa riferimento ad almeno un referente finanziamenti mentre, per indicare la cardinalità massima con cui la classe “DistrettoIndustriale” partecipa a questa relazione, essa è stata posta a *Functional*, cioè indica che un distretto industriale fa riferimento ad al più un referente finanziamenti. E’ stata inoltre posta anche *inverseFunctional* per indicare che la sua inversa (“rf-di”) è di tipo functional:

ObjectProperty(<http://www.owl-ontologies.com/unnamed.owl# **inversa_di_rf-di** > Functional InverseFunctional
inverseOf(<http://www.owl-ontologies.com/unnamed.owl# **rf-di** >)
domain(a: **DistrettoIndustriale**)
range(a: **ReferenteFinanziamenti**))

APPARTIENE

La relazione "appartiene" è stata tradotta in una *object property* di Protégé.
Ha come dominio la classe "Azienda" e come range la classe "DistrettoIndustriale". Per esprimere il limite inferiore della cardinalità 1..1 con cui la classe “Azienda” partecipa a questa relazione, è stata posta la restrizione " appartiene ≥ 1" nella classe " Azienda ", cioè indica che un progetto è relativo ad almeno un settore produttivo, mentre per indicare la cardinalità massima con cui la classe “Azienda” partecipa a questa relazione, essa è stata posta a *Functional*, cioè indica che un progetto è relativo ad al più un settore produttivo:

ObjectProperty(a: **appartiene** Functional
inverseOf(a:**inversa_di_ appartiene**)
domain(a: **Azienda**)
range(a: **DistrettoIndustriale**))

La relazione inversa, chiamata "inversa_di_appartiene", ha come dominio la classe " DistrettoIndustriale " e come range la classe " Azienda " ed è stata posta *inverseFunctional* in quanto la sua inversa ("appartiene") è di tipo *functional* :

ObjectProperty(a: **inversa_di_ appartiene** InverseFunctional
inverseOf(a: **appartiene**)
domain(a: **DistrettoIndustriale**)
range(a: **Azienda**))

DELLA
La relazione "della" è stata tradotta in una *object property* di Protégé.
Ha come dominio la classe " Domanda " e come range la classe " Azienda ". Per esprimere il limite inferiore della cardinalità 1..1 con cui la classe "Domanda" partecipa a questa relazione, è stata posta la restrizione " della ≥ 1" nella classe " Domanda ", cioè indica che una domanda è di almeno una azienda, mentre, per indicare la cardinalità massima con cui la classe "Domanda" partecipa a questa relazione, essa è stata posta a *Functional*, cioè indica che una domanda è di al più una azienda:

ObjectProperty(a:**della** Functional
inverseOf(a:**inversa_di_della**)
domain(a:**Domanda**)
range(a:**Azienda**))

La sua relazione inversa, chiamata "inversa_di_della", ha come dominio la classe "Azienda " e come range la classe "Domanda". Per esprimere il limite inferiore della cardinalità 1..n con cui la classe "Azienda" partecipa a questa relazione, è stata posta la restrizione "inversa_di_della ≥ 1" nella classe " Azienda ", cioè indica che una azienda è relativa ad almeno una domanda. E' stata inoltre posta *inverseFunctional* per indicare che la sua inversa ("della") è di tipo *functional*:

ObjectProperty(a: **inversa_di_della** InverseFunctional
inverseOf(a: **della**)
domain(a: **Azienda**)
range(a: **Domanda**))

PER
La relazione "per" è stata tradotta in una *object property* di Protégé.
Ha come dominio la classe "Domanda" e come range la classe " BandoFinanziamento ". Per esprimere il limite inferiore della cardinalità 1..1 con cui la classe "Domanda" partecipa a questa relazione, è stata posta la restrizione " per ≥ 1" nella classe " Domanda ", cioè indica che una domanda è per almeno un bando finanziamento, mentre per indicare la cardinalità massima con cui la classe "Domanda" partecipa a

questa relazione, essa è stata posta a *Functional*, cioè indica che una domanda è per al più un bando finanziamento:

ObjectProperty(a: **per** Functional
 inverseOf(a:**inversa_di_ per**)
 domain(a: **Domanda**)
 range(a: **BandoFinanziamento**))

La relazione inversa, chiamata "inversa_di_per", ha come dominio la classe " BandoFinanziamento " e come range la classe " Domanda " ed è stata posta *inverseFunctional* in quanto la sua inversa ("per") è di tipo *functional* :

ObjectProperty(a: **inversa_di_ per** InverseFunctional
 inverseOf(a: **per**)
 domain(a: **BandoFinanziamento**)
 range(a: **Domanda**))

COMPRENDE
La relazione "comprende" è stata tradotta in una *object property* di Protégé.
Ha come dominio la classe "Domanda" e come range la classe " Progetti ". Per esprimere il limite inferiore della cardinalità 1..n con cui la classe "Domanda" partecipa a questa relazione, è stata posta la restrizione " comprende ≥ 1" nella classe " Domanda ", cioè indica che una domanda comprende almeno un progetto:

ObjectProperty(a: **comprende**
 inverseOf(a:**inversa_di_ comprende**)
 domain(a: **Domanda**)
 range(a: **Progetti**))

La relazione inversa, chiamata "inversa_di_comprende", ha come dominio la classe " Progetti " e come range la classe " Domanda ":

ObjectProperty(a: :**inversa_di_ comprende**
 inverseOf(a: **comprende**)
 domain(a: **Progetti**)
 range(a: **Domanda**))

E' necessario inoltre specificare che tutti gli individui che appartengono alle singole classi sono diversi tra loro e che tutte le classi sono disgiunte.

DifferentIndividuals(a:Terry a:Emma)

DisjointClasses(a:ConsigliereComunale a:DistrettoIndustriale)

DisjointClasses(a: ConsiglieреComunale a:BandoFinanziamento)
DisjointClasses(a:ReferenteFinanziario a:Domanda) [...]

QUERY

Le query richieste nel compito non sono del tutto realizzabili. Infatti:

1. **Query1**: Per ogni domanda di risposta ad un bando, restituire la partita IVA, il distretto industriale di appartenenza ed il capitale sociale dell'azienda che l'ha presentata.

   Non possiamo estrarre gli attributi "partita IVA" e "capitale sociale".

2. **Query2**: Restituire le domande di risposta a bandi per le quali tutti i progetti indicati dall'azienda proponente sono nel settore produttivo di interesse del bando di finanziamento alla quale si riferisce la domanda .

   Si tratta di una query non ad albero per cui non può essere realizzata: presenta un ciclo.

3. **Query3**: Un distretto industriale viene detto "rilevante per i finanziamenti" se sono state presentate almeno 10 domande di finanziamento da aziende appartenenti al distretto industriale stesso. Per ciascun distretto industriale rilevante per i finanziamenti, calcolare il numero di domande presentate da aziende ad esso appartenenti.

   Non è possibile contare il numero di domande.

Riferimenti bibliografici

- G. De Giacomo, dispense del corso di "Seminari di ingegneria del software" anno accademico 2005-2006, http://www.dis.uniroma1.it/~degiacomo

- Matthew Horridge, Holger Knublauch, Alan Rector, Robert Stevens, Chris Wroe, ProtégéOWLTutorial, http://protege.stanford.edu/

- www.w3.org/2004/OWL/

- Matthew Horridge,Holger Knublauch, Alan Rector, Robert Stevens, Chris Wroe - The University Of Manchester, Stanford University, http://www.co-ode.org/resources/tutorials/ProtegeOWLTutorial.pdf

- D. Nardi, dispense del corso di "Rappresentazione della conoscenza", http://www.dis.uniroma1.it/~nardi

- D. Calvanese, D. McGuinness, D. Nardi, P. Patel-Schneider, "The Description Logic Handbook" edited by Franz Baader.

- M. Colombetti, appunti delle lezioni 2005-2006 di "Ingegneria della conoscenza"

- N. Capuano, "Ontologie e OWL: Teoria e Pratica"