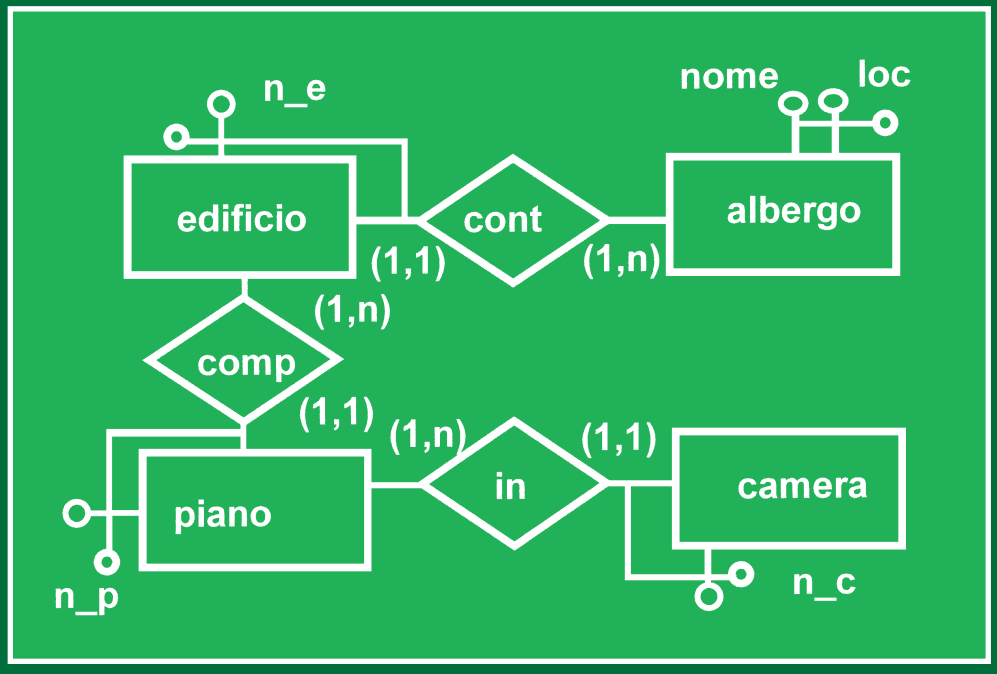
D. Braga ♦ M. Brambilla ♦ A. Campi

ESERCIZIARIO DI BASI DI DATI



SOCIETÀ EDITRICE ESCULAPIO

D. Braga ♦ M. Brambilla ♦ A. Campi

ESERCIZIARIO DI BASI DI DATI

SOCIETÀ EDITRICE ESCULAPIO

ISBN 978-88-7488-328-8

Società Editrice Esculapio s.r.l.
Via Terracini, 30 – 40131 Bologna
www.editrice-esculapio.com – info@editrice-esculapio.it

Impaginazione: Gabriella Gatti e Giancarla Panigali
Stampato da: Legodigit – Lavis (TN)
Printed in Italy

Sommario

Introduzione

Questo testo raccoglie una serie di esercizi relativi all'interrogazione e alla progettazione di basi di dati relazionali, derivati dall'esperienza didattica degli autori negli insegnamenti del corso di laurea in Ingegneria Informatica del Politecnico di Milano.

Il testo si rivolge primariamente agli studenti di corsi di basi di dati erogati per la Laurea di Primo Livello (nelle discipline dell'Ingegneria dell'Informazione), e nella Laurea Magistrale (nelle altre discipline, per chi completi la sua preparazione in informatica con un corso di fondamenti di basi di dati). Più in generale, crediamo che questo testo possa portare un aiuto e un contributo concreto a chiunque debba imparare a confrontarsi con l'utilizzo e la progettazione di basi di dati, anche in ambienti non accademici; pensiamo in particolare ai corsi di formazione per realtà aziendali dove l'uso dei database è una necessità quotidiana.

Dei quattro capitoli del testo, il primo propone esercizi sui tre linguaggi formali comunemente presentati nei corsi di basi di dati, il secondo contiene esercizi di SQL che introducono gradualmente i costrutti e le difficoltà più tipiche nell'utilizzarli, il terzo affronta la progettazione concettuale e logica di basi di dati tramite il modello Entità-Relazione e la sua traduzione in schemi relazionali, il quarto presenta le soluzioni di una selezione di temi d'esame recentemente assegnati (2004-2006) al Politecnico di Milano.

Per quanto riguarda le soluzioni proposte, è importante sottolineare che l'attività di progetto non produce mai risultati univoci, poiché risente inevitabilmente degli stili e delle preferenze dei singoli progettisti; le basi di dati non fanno eccezione a questo principio, e gli autori dichiarano sin d'ora che soluzioni presentate sono influenzate dal loro stile. Non sono certo le uniche soluzioni corrette, dunque; anzi, in vari casi tra i commenti alle soluzioni proposte sono discusse alcune soluzioni alternative, e a volte sono riportati alcuni errori comuni.

Ringraziamo Stefano Ceri, Piero Fraternali, Maristella Matera, Stefano Paraboschi, Davide Martinenghi, Giuseppe Pozzi, Fabio Schreiber, Letizia Tanca e Alessandro Raffio e per i consigli, le osservazioni e i suggerimenti.

Gli autori

Capitolo Primo
Linguaggi formali di interrogazione

Esercizio 1.1: La biblioteca

UTENTE (Codice, Nome, Cognome, Indirizzo, Telefono)
PRESTITO (Collocazione, CodiceUtente, DataPrestito, DataRest)
COPIA (Collocazione, ISBN, DataAcquisizione)
DATILIBRO (ISBN, Titolo, AnnoPub, CasaEd, PrimoAut, Genere)

Le chiavi sono sottolineate. COPIA descrive le copie cartacee, dei libri. I libri sono invece descritti, in quanto "pubblicazioni", dalla relazione DATILIBRO. Così si rappresenta la disponibilità in biblioteca di più copie di uno stesso libro.
La chiave di PRESTITO è triplice per permettere di rappresentare nella base di dati il prestito dello stesso libro allo stesso utente in date diverse (o di più libri allo stesso utente nella stessa data, o dello stesso libro a utenti diversi nello stesso giorno, nel caso limite di una restituzione rapidissima). L'ISBN è un identificatore internazionale univoco per le pubblicazioni (International Standard Book Number).

1.1.1 *Trovare i titoli di tutti i libri pubblicati negli anni '80.*

Algebra relazionale

$\Pi_{Titolo}\ (\sigma_{AnnoPub>=1980\ \wedge\ AnnoPub<1990}\ \text{DATILIBRO})$

Calcolo relazionale

$$\{\ t \mid \exists\ t_D \in \text{DATILIBRO}\ (\ t[Titolo] = t_D[Titolo] \wedge\ t_D[AnnoPub] >= 1980 \wedge\ t_D[AnnoPub] < 1990\ )\ \}$$

Datalog

```
Anni80( Tit ) :- DATILIBRO( _, Tit, AnP, _, _, _ ), AnP >= 1980, AnP < 1990

? - Anni80( X )
```

1.1.2 *Trovare i titoli di tutti i libri* ***non*** *pubblicati negli anni '80.*

Algebra relazionale

Si può procedere (a) negando il predicato dell'interrogazione 1.1.1 tramite la relazione di DeMorgan, (b) usando l'unione o (c) utilizzando un "passaggio al complemento" che riutilizzi l'interrogazione 1.1.1 per differenza:

(a) $\Pi_{Titolo}\ (\sigma_{AnnoPub<1980 \vee AnnoPub>=1990}\ \text{DATILIBRO})$

(b) $\Pi_{Titolo}\ (\sigma_{AnnoPub < 1980}\ \text{DATILIBRO}) \cup \Pi_{Titolo}\ (\sigma_{AnnoPub >= 1990}\ \text{DATILIBRO})$

(c) $\Pi_{Titolo}\ (\text{DATILIBRO} - \sigma_{AnnoPub >= 1980 \wedge AnnoPub < 1990}\ \text{DATILIBRO})$

Attenzione: nel caso (c) è fondamentale effettuare la proiezione su Titolo solo *dopo* aver effettuato la differenza. Non è possibile effettuare la proiezione separatamente sui due insiemi prima di eseguire la differenza in quanto il risultato ne risulterebbe alterato. Del resto, come è noto, nelle trasformazioni di espressioni dell'algebra relazionale non è ammessa la trasformazione di *push* della proiezione rispetto alla differenza. Infatti, per fare un esempio concreto, si consideri un'istanza di base di dati che contiene nella relazione DATILIBRO le sole due tuple seguenti:

[_, "Autobiografia", 1988, _, "Rossi", "Bio"]
[_, "Autobiografia", 1994, _, "Verdi", "Bio"]

Le soluzioni proposte restituiscono "Autobiografia", quella eventualmente ottenuta dall'applicazione errata del *push* restituirebbe l'insieme vuoto.

Calcolo relazionale

Possiamo negare il predicato usato in 1.1.1 (a) applicando DeMorgan oppure (b) lasciando indicata la negazione:

(a) $\{ t \mid \exists\ t_D \in \text{DATILIBRO}$
$(\ t[\text{Titolo}] = t_D[\text{Titolo}] \wedge$
$\mathbf{(}\ t_D[\text{AnnoPub}] < 1980 \vee t_D[\text{AnnoPub}] >= 1990\ \mathbf{)}\ )\ \}$

(b) $\{ t \mid \exists\ t_D \in \text{DATILIBRO}$
$(\ t[\text{Titolo}] = t_D\ [\text{Titolo}] \wedge$
$\neg\ \mathbf{(}\ t_D[\text{AnnoPub}] >= 1980 \wedge t_D[\text{AnnoPub}] < 1990\ \mathbf{)}\ )\ \}$

Datalog

Secondo la logica del "passaggio al complemento" della soluzione (c) in algebra:

Anni80(ISBN) :- DATILIBRO(ISBN, _, AnP, _, _, _), AnP>=1980, AnP<1990

NonAnni80(T) :- DATILIBRO(ISBN, T, _, _, _, _),
¬ Anni80(ISBN)

? - NonAnni80(X)

Si noti che la soluzione proposta è *safe* rispetto alla negazione. Possiamo anche definire direttamente la regola NonAnni80 come unione dei risultati di due regole:

NonAnni80bis(T) :- DATILIBRO(_, T, AnP, _, _, _), AnP < 1980

NonAnni80bis(T) :- DATILIBRO(_, T, AnP, _, _, _), AnP >= 1990

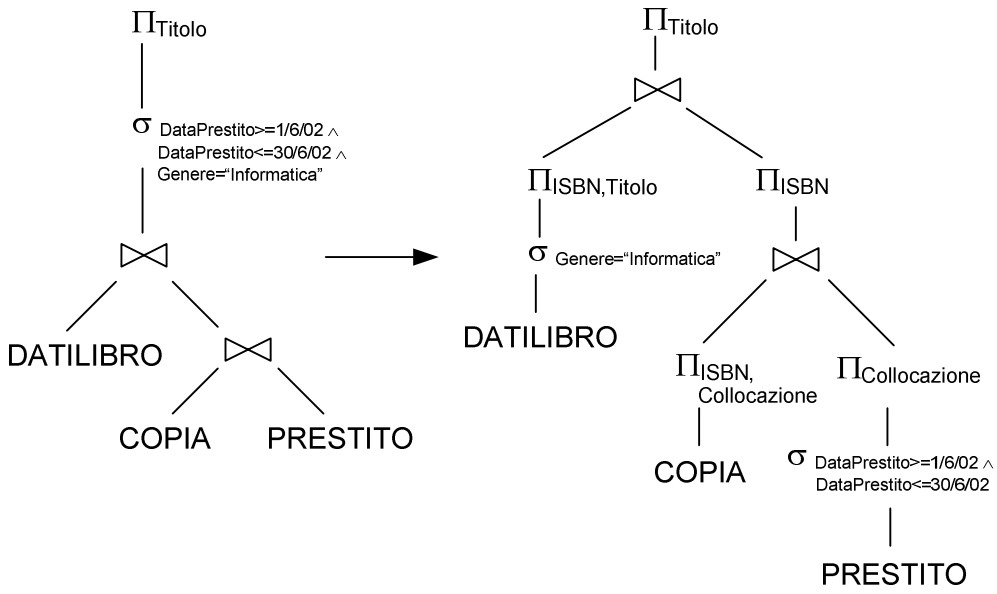
? - NonAnni80bis(X)

1.1.3 *Trovare i titoli dei libri di informatica prestati nel giugno '02.*

Algebra relazionale

$\Pi_{Titolo}(\sigma_{DataPrestito>=1/6/02 \wedge DataPrestito<=30/6/02 \wedge Genere='Info'}(DATILIBRO \bowtie COPIA \bowtie PRESTITO))$

Per ottimizzare è necessario decidere una strategia di associazione per i due join, la cui priorità non è indicata nell'espressione. Scegliamo di associare a destra:



Calcolo Relazionale

$\{ t \mid \exists t_D \in DATILIBRO, \exists t_C \in COPIA, \exists t_P \in PRESTITO$
$(\ t[Titolo] = t_D[Titolo] \wedge$
$t_C[ISBN] = t_D[ISBN] \wedge$
$t_P[Collocazione] = t_C[Collocazione] \wedge$
$t_P[DataPrestito] \geq 1/6/2002 \wedge t_P[DataPrestito] \leq 30/6/2002 \wedge$
$t_D[Genere] = \text{"Informatica"}\ )$
$\}$

Datalog

```
InfoPreGiu02( Tit )  :-  DATILIBRO( Isbn, Tit, _, _, _, "Informatica" ),
                         COPIA( Coll, Isbn, _ ),
                         PRESTITO( Coll, _, DataP, _ ),
                         DataP ≥ 1/6/02,  DataP ≤ 30/6/02

? - InfoPreGiu02( Titolo )
```

1.1.4 *Estrarre nome, cognome e codice degli utenti che non hanno **mai** preso in prestito libri di informatica.*

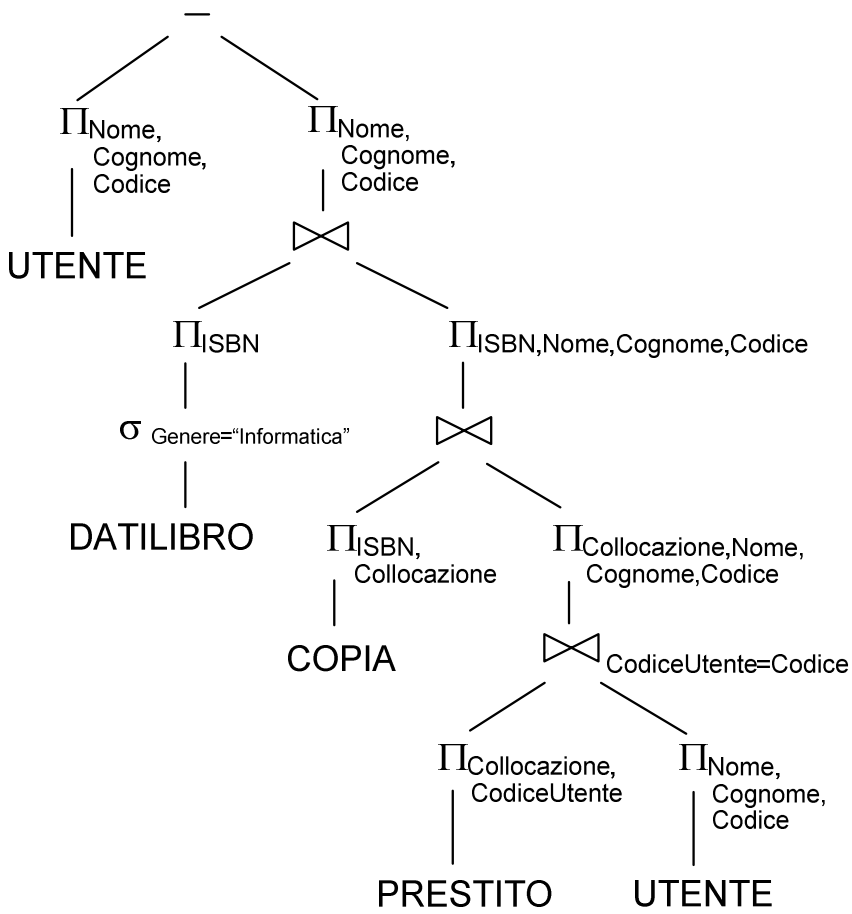
Algebra relazionale

All'insieme di tutti gli utenti sottraiamo gli utenti che figurano come "prestatari" di almeno uno tra i libri di informatica.

$$(\Pi_{\text{Nome, Cognome, Codice}} \text{UTENTE}) - (\Pi_{\text{Nome, Cognome, Codice}} ((\sigma_{\text{Genere="Info"}} \text{DATILIBRO}) \bowtie \text{COPIA} \bowtie \text{PRESTITO} \bowtie_{\text{Codice=CodiceUtente}} \text{UTENTE}))$$

Si noti che è necessario effettuare proiettare gli utenti e il risultato dei join sugli stessi tre attributi, in modo da renderli compatibili prima di eseguire la differenza.

Ottimizzando:



Calcolo relazionale

$\{ t \mid \exists t_U \in$ UTENTE
$(t[Nome,Cognome,Codice] = t_U[Nome,Cognome,Codice] \wedge$
$\neg (\exists t_D \in$ DATILIBRO, $t_C \in$ COPIA, $t_P \in$ PRESTITO
$t_D[ISBN] = t_C[ISBN] \wedge t_C[Collocazione] = t_P[Collocazione] \wedge$
$t_P[CodiceUtente] = t_D[Codice] \wedge t_D[Genere] =$ "Informatica")) $\}$

Datalog

```
PresoInfo( CodU ) :- DATILIBRO( Isbn, _, _, _, _, "Info" ),
                     COPIA( Coll, Isbn, _, _ ), PRESTITO( Coll, CodU, _, _ )

MaiPresoInfo( N, C, Cod ) :- UTENTE( Cod, N, C, _, _ ), ¬ PresoInfo (Cod)

? - MaiPresoInfo ( Nome, Cognome, Codice )
```

1.1.5 *Estrarre i titoli dei libri prestati* ***o*** *acquisiti nel giugno '02.*

Algebra relazionale

Interpretiamo l'***o*** come un or come inclusivo. Costruiamo il risultato come l'unione dei titoli acquisiti e di quelli prestati nel periodo indicato.

$(\Pi_{Titolo}($DATILIBRO $\bowtie (\sigma_{DataAcquisizione>=1/6/02 \wedge DataAcquisizione<1/7/02}$ COPIA$)))$ $\cup$

$(\Pi_{Titolo}($DATILIBRO $\bowtie$ COPIA $\bowtie (\sigma_{DataPrestito>=1/6/02 \wedge DataPrestito<1/7/02}$ PRESTITO$)))$

Si noti che per le acquisizioni è sufficiente considerare le relazioni DATILIBRO e COPIA, mentre per selezionare in base alla data dell'ultimo prestito è necessario includere anche la relazione PRESTITO.

Calcolo relazionale

$\{ t \mid \exists t_D \in$ DATILIBRO, $\exists t_C \in$ COPIA
$(t[Titolo] = t_D[Titolo] \wedge$
$t_C[ISBN] = t_D[ISBN] \wedge$
$((t_C[DataAcquisizione] \geq 1/6/02 \wedge t_C[DataAcquisizione] \leq 30/6/02) \vee$
$(\exists t_P \in$ PRESTITO
$t_P[Collocazione] = t_C[Collocazione] \wedge$
$t_P[DataPrestito] \geq 1/6/2002 \wedge t_P[DataPrestito] \leq 30/6/2002))) \}$

Datalog

```
Risultato( Tit ) :- DATILIBRO( Isbn, Tit, _, _, _, _ ), COPIA( Coll, Isbn, _ ),
                    PRESTITO( Coll, _, DataPre, _ ),
                    DataPre >= 1/6/02, DataPre <= 30/6/02

Risultato( Tit ) :- DATILIBRO( Isbn, Tit, _, _, _, _ ), COPIA( _, Isbn, DataA ),
                    DataA >= 1/6/02, DataA <= 30/6/02

? - Risultato( Titolo )
```

1.1.6 *Per ogni utente, indicare l'ultimo libro preso in prestito (eventuali utenti che non hanno mai preso libri* ***non*** *devono far parte del risultato).*

Algebra relazionale

La costruzione del risultato è divisa in due fasi. Dapprima estraiamo per ogni utente gli ***ultimi prestiti*****,** ottenuti sottraendo all'insieme di tutti i prestiti "i prestiti che hanno un prestito *successivo*", e assegniamo il risultato di tale espressione alla relazione virtuale **UL**. Il semi-join è fatto a pari utente, perché non ha senso confrontare le date di prestiti di utenti diversi.

$$\mathbf{UL} = \text{PRESTITO} - (\text{PRESTITO} \ltimes_{\text{DataPrestito}<\text{DataPrestito} \wedge \text{CodiceUtente}=\text{CodiceUtente}} \text{PRESTITO})$$

Utilizzando poi tale risultato, estraiamo i dati richiesti:

$$\Pi_{\text{Codice, Nome, Cognome, CodiceUtente}} (\text{UTENTE} \bowtie \mathbf{UL} \bowtie \text{COPIA} \bowtie \text{DATILIBRO})$$

Calcolo relazionale

Per ogni utente estraiamo il prestito tale che non esiste un prestito successivo:

$$\{ t \mid \exists t_D \in \text{DATILIBRO}, \exists t_C \in \text{COPIA}, \exists t_P \in \text{PRESTITO}, \exists t_U \in \text{UTENTE} (t[\text{Titolo}] = t_D[\text{Titolo}] \wedge t[\text{Cognome, Nome}] = t_U[\text{Cognome, Nome}] \wedge t_D[\text{ISBN}] = t_C[\text{ISBN}] \wedge t_C[\text{Collocazione}] = t_P[\text{Collocazione}] \wedge t_P[\text{CodiceUtente}] = t_U[\text{Codice}] \wedge \neg (\exists t_{P2} \in \text{PRESTITO} (t_{P2}[\text{CodiceUtente}] = t_P[\text{CodiceUtente}] \wedge t_{P2}[\text{DataPrestito}] > t_P[\text{DataPrestito}]))) \}$$

Datalog

```
NonUltimoPr( coll, cod, data ) :- PRESTITO( coll, cod, data, _ ),
                                  PRESTITO( _, cod, data1, _ ), data < data1

UltimoPrestito( Cog, Nom, Tit ) :- DATILIBRO( isbn, Tit, _, _, _, _ ),
                                   COPIA( coll, isbn, _ ),
                                   PRESTITO( coll, cu, data ),
                                   UTENTE( cu, Cog, Nom ),
                                   ¬ NonUltimoPr( coll, cu, data )

? - UltimoPrestito( cognome, nome, ultimo-titolo )
```

1.1.7 *Trovare i titoli dei libri che non sono stati* ***mai*** *presi in prestito.*

Algebra relazionale

Π_{Titolo} (DATILIBRO – (DATILIBRO$\rtimes$ COPIA $\rtimes$ PRESTITO))

Dall'insieme di tutti i libri sottraiamo quelli "di cui esiste una copia per la quale è stato fatto almeno un prestito". Si noti che la proiezione sul titolo è fatta *dopo* la differenza; la soluzione che la effettua errata, perché estrarrebbe eventuali libri distinti ma aventi lo stesso titolo (possibile nel caso di titoli molto comuni, come ad esempio "memorie" o "autobiografia"):

~~Π_{Titolo} DATILIBRO – Π_{Titolo} (DATILIBRO$\rtimes$ COPIA $\rtimes$ PRESTITO)~~

Calcolo relazionale

$\{ t \mid \exists t_D \in$ DATILIBRO
(t[Titolo] = t_D[Titolo] $\wedge$
$\neg$ ($\exists t_L \in$ COPIA, $\exists t_P \in$ PRESTITO
(t_D[ISBN] = t_L[ISBN] $\wedge$ t_L[Collocazione] = t_P[Collocazione]))) $\}$

Datalog

Preso(Isbn) :- PRESTITO(Coll, _, _, _), COPIA(Coll, Isbn, _)

MaiPreso(Titolo) :- DATILIBRO(Isbn, Titolo, _, _, _, _), $\neg$ Preso(Isbn)

? - MaiPreso(T)

1.1.8 *Trovare gli utenti che hanno preso libri di* ***tutti*** *i generi.*

Algebra relazionale

(Π_{Codice} UTENTE) – (Π_{Codice} ($\Pi_{Codice, Genere}$ (UTENTE $\times$ DATILIBRO) –
$\Pi_{Codice, Genere}$ (UTENTE$\bowtie$ PRESTITO $\bowtie$ COPIA $\bowtie$ DATILIBRO)))

Il prodotto cartesiano genera **tutte** le coppie ‹Utente,Genere›, da cui sottraiamo le coppie effettivamente documentate da un prestito, ottenendo gli "accoppiamenti mancati". Per ulteriore differenza otteniamo gli utenti con tutti gli accoppiamenti.

Calcolo relazionale

$\{ t \mid \exists t_U \in$ UTENTE
(t[CodiceUtente] = t_U[CodiceUtente] $\wedge$
$\neg$ ($\exists t_{DM} \in$ DATILIBRO
$\neg$ ($\exists t_C \in$ COPIA, $\exists t_P \in$ PRESTITO, $\exists t_D \in$ DATILIBRO
(t_D[ISBN] = t_C[ISBN] $\wedge$
t_C[Collocazione] = t_P[Collocazione] $\wedge$
t_P[CodiceUtente] = t_U[CodiceUtente] $\wedge$
t_{DM}[Genere] = t_D[Genere])))) $\}$

Datalog

GenereInBiblio(Gen) :- DATILIBRO(_, _, _, _, _, Gen)

GenereLetto(Ut, Ge) :- PRESTITO(Coll, Ut, _, _), COPIA(Coll, Isbn, _),
DATILIBRO(Isbn, _, _, _, _, Ge)

HaUnGenMaiLetto(Ut) :- UTENTE(Ut, _, _, _, _), GenereInBiblio(Ge),
¬ GenereLetto (Ut, Ge)

LettiTuttiGen(CodUt) :- UTENTE(CodUt, _, _, _, _), ¬ HaUnGenMaiLetto(CodUt)

? - LettiTuttiGen(X)

1.1.9 *Trovare i libri che sono* ***attualmente*** *in prestito*:

La query è piuttosto problematica. Lo schema della base dati richiederebbe, infatti, l'inserimento della data di restituzione già al momento del prelievo. Si può ipotizzare che (a) si inserisca subito una data di restituzione prevista o (b) il valore sia lasciato a null fino all'avvenuta restituzione. Nel caso (a) dovremmo selezionare i prestiti con DataRest successiva alla data corrente, nel caso (b) dovremmo selezionare in base al valore nullo. Tuttavia, nel caso (a) il fatto che la data prevista di restituzione sia trascorsa non garantisce che il libro sia davvero rientrato, così come nel caso (b) occorre sempre ricordare che un valore nullo in un database può avere molte origini e molte spiegazioni (indisponibilità, inapplicabilità, …).
Per dare una risposta più certa a questa (importante!) interrogazione occorrerebbe, in effetti, riprogettare lo schema, rendendo indipendenti le rappresentazioni del prelievo (con eventuale data di restituzione prevista) e dell'effettiva restituzione.

Esercizio 1.2: Variante della biblioteca

AUTORE (Nome, Cognome, Data-N, Nazionalità)
AUTORELIBRO (Nome, Cognome, ISBN)
LIBRO (ISBN, Titolo, Genere, Lingua)
COPIA (Segnatura, ISBN, Scaffale, DataAcq)

In questa variante sono presenti i dati di tutti gli autori. Essendo la corrispondenza tra autori e libri una corrispondenza molti a molti, la relazione AutoreLibro funge da "ponte". Si noti che la corrispondenza tra Autore e AutoreLibro avviene per identità di *coppie di valori* all'interno della singola tupla (l'eventuale integrità referenziale si applica, cioè, alle coppie, non ai nomi e ai cognomi separatamente).

1.2.1 *Trovare tutte le segnature dei libri di Asimov pubblicati in lingua spagnola.*

Algebra relazionale

Disponendo del cognome dell'autore, non è necessario utilizzare la tabella Autore. Operiamo con semi-join sulle restanti tabelle, per risalire dall'autore alla copia.

$\Pi_{\text{Segnatura}}(\text{COPIA} \ltimes (\sigma_{\text{Lingua="spagnola"}} \text{LIBRO}) \ltimes (\sigma_{\text{Cognome="Asimov"}} \text{AUTORELIBRO}))$

Calcolo relazionale

$\{ t \mid \exists t_A \in$ AUTORELIBRO, $\exists t_L \in$ LIBRO, $\exists t_C \in$ COPIA
($t[Segnatura] = t_C[Segnatura] \wedge$
$t_C[ISBN] = t_L[ISBN] \wedge$
$t_L[ISBN] = t_A[ISBN] \wedge$
$t_A[Cognome] =$ "Asimov" $\wedge$
$t_L[Lingua] =$ "spagnola") }

Datalog

```
AsimovEpañol( Segn ) :- COPIA( Segn, isbn, _, _, ),
                        LIBRO( isbn, _, _, "spagnola" ),
                        AUTORELIBRO( _, "Asimov", isbn )

? - AsimovEpañol( Segnatura )
```

1.2.2 *Le collaborazioni internazionali (libri con autori di nazionalità diversa).*

Algebra relazionale

$$\textbf{COLLINT} = \Pi_{Isbn} ((\text{AUTORELIBRO} \bowtie \text{AUTORE}) \ltimes_{Isbn=Isbn \wedge Nazionalità <> Nazionalità} (\text{AUTORELIBRO} \bowtie \text{AUTORE}))$$

Per includere il titolo:

$$\Pi_{Titolo, ISBN} (\text{LIBRO} \bowtie \textbf{COLLINT})$$

Calcolo relazionale

$\{ t \mid \exists t_L \in$ LIBRO,
$\exists t_{AL1}, t_{AL2} \in$ AUTORELIBRO,
$\exists t_{A1}, t_{A2} \in$ AUTORE
($t[ISBN, Titolo] = t_L[ISBN, Titolo] \wedge$
$t_{AL1}[ISBN] = t_L[ISBN] \wedge t_{AL2}[ISBN] = t_L[ISBN] \wedge$
$t_{A1}[Nom,Cogn] = t_{AL1}[Nom,Cogn] \wedge t_{A2}[Nom,Cogn] = t_{AL2}[Nom,Cogn] \wedge$
$t_{A1}[Nazionalità] <> t_{A2}[Nazionalità]$) }

Datalog

```
CollInt( Tit, Isbn ) :- LIBRO( Isbn, Tit, _, _ ),
                        AUTORELIBRO( nom1, cog1, Isbn ),
                        AUTORELIBRO( nom2, cog2, Isbn ),
                        AUTORE( nom1, cog1, _, naz1 ),
                        AUTORE( nom2, cog2, _, naz2 ),  naz1 <> naz2

? - CollInt( Titolo, ISBN )
```

1.2.3 *Libri con autori* ***tutti*** *della stessa nazionalità.*

Algebra relazionale

$\Pi_{Titolo,\ ISBN}$ (LIBRO – (LIBRO $\rtimes$ **COLLINT**))

Calcolo relazionale

$\{ t \mid \exists\ t_L \in$ LIBRO
($t[ISBN, Titolo] = t_L[ISBN, Titolo] \wedge$
$\neg$ ($\exists\ t_{AL1}, t_{AL2} \in$ AUTORELIBRO,
$\exists\ t_{A1}, t_{A2} \in$ AUTORE
($t_{AL1}[ISBN] = t_L[ISBN] \wedge$
$t_{AL2}[ISBN] = t_L[ISBN] \wedge$
$t_{A1}[Nom,Cogn] = t_{AL1}[Nom,Cogn] \wedge$
$t_{A2}[Nom,Cogn] = t_{AL2}[Nom,Cogn] \wedge$
$t_{A1}[Nazionalità] <> t_{A2}[Nazionalità]$))) }

Datalog

Riutilizziamo la definizione della regola CollInt (titolo, isbn), definita nell'esercizio precedente (1.2.3), che estrae tutte le collaborazioni internazionali:

```
CollabSoloNazionale( Tit, Isbn ) :- LIBRO( Isbn, Tit, _, _ ),
                                    ¬ CollInt( Tit, Isbn )

? - CollabSoloNazionale( Titolo, ISBN )
```

1.2.4 *Libri scritti in una lingua* ***diversa*** *dalla nazionalità di* ***tutti*** *gli autori (nella ipotesi semplicistica che ci sia identità dei due domini lingua e nazionalità: sì italiano, francese, inglese, russo,… no belga, creolo, bantu…).*

Algebra relazionale

Estraiamo tutti i libri tranne quelli scritti nella lingua di uno qualsiasi degli autori:

$\Pi_{Titolo,\ ISBN}$ (LIBRO –
(LIBRO $\rtimes_{Isbn=Isbn\ \wedge\ Lingua=Nazionalità}$ (AUTORELIBRO $\bowtie$ AUTORE)))

Calcolo relazionale

$\{ t \mid \exists\ t_L \in$ LIBRO
($t[ISBN,Titolo] = t_L[ISBN,Titolo] \wedge$
$\neg$ ($\exists\ t_{AL} \in$ AUTORELIBRO, $\exists\ t_A \in$ AUTORE
($t_{AL}[ISBN] = t_L[ISBN] \wedge$
$t_{AL}[Nome,Cognome] = t_A[Nome,Cognome] \wedge$
$t_L[Lingua] = t_A[Nazionalità]$))) }

Datalog

LinguaNazCorrispondeAlmenoUno(Isbn) :- LIBRO(Isbn, _, _, LinguaNaz),
AUTORELIBRO(nom, cogn, Isbn),
AUTORE(nom, cogn, _, LinguaNaz)

LinguaDiversaTuttiAutori(Isbn, Tit) :- LIBRO(Isbn, Tit, _, _),
¬ LinguaNazCorrispondeAlmenoUno(Isbn)

? - LinguaDiversaTuttiAutori(Isbn, Titolo)

Esercizio 1.3: Trasporti aerei

Consideriamo il seguente schema:

AEROPORTO (Città, Nazione, NumPiste)
VOLO (IdVolo, GiornoSett, CittaPart, OraPart, CittaArr, OraArr, TipoAereo)
AEREO (TipoAereo, NumPasseggeri, QtaMerci)

Si assume per semplicità che vi sia un solo aeroporto per città, che l'orario si ripeta settimanalmente (come si nota dallo schema di VOLO, dove il tempo figura solo in GiornoSettimana), e che un volo sia effettuato sempre dallo stesso tipo di aereo, indipendentemente dai biglietti venduti (è possibile che si tratti di velivoli diversi, purché dello stesso modello). La quantità di merci è caratteristica del *modello* di aereo, quindi misura la "capacità della stiva", non le merci realmente trasportate.

1.3.1 *Voli da Milano a Roma di martedì oppure di giovedì con più di 200 posti.*

Algebra relazionale

$$\Pi_{IdVolo} \Big(\Big(\sigma_{CittaPart='Milano' \wedge CittaArr='Roma' \wedge (GiornoSett='martedi' \vee GiornoSett='giovedi')} VOLO \Big) \bowtie \big(\sigma_{NumPasseggeri>200} AEREO \big) \Big)$$

Calcolo relazionale

$$\{ t \mid \exists t_V \in VOLO, \exists t_A \in AEREO \; (t[IdVolo] = t_V[IdVolo] \wedge t_V[TipoAereo] = t_A[TipoAereo] \wedge t_V[CittaPart] = \text{“Milano”} \wedge t_V[CittaArr] = \text{“Roma”} \wedge (t_V[GiorSett] = \text{“martedì”} \vee t_V[GiorSett] = \text{“giovedì”}) \wedge t_A[NumPasseggeri] > 200) \}$$

Datalog

VoloMarteGiove(id) :- VOLO(id, “martedì”, _, _, _, _, _)

VoloMarteGiove(id) :- VOLO(id, “giovedì”, _, _, _, _, _)

MilanoRoma200MG(IdVolo) :- VOLO(IdVolo, _, Milano”, _, “Roma”, _, tipo),
AEREO(tipo, numpass, _),
VoloMarteGiove(IdVolo),
numpass > 200

? - MilanoRoma200MG(IdVolo)

1.3.2 *Voli per Roma per soli passeggeri (fatti da aerei che non trasportano merci).*

Algebra relazionale

Π_{IdVolo} ($\sigma_{CittaArr='Roma'}$ VOLO $\rhd\!\!<$ $\sigma_{QtaMerci=0}$ AEREO)

Calcolo relazionale

$\{ t \mid \exists t_V \in$ VOLO, $\exists t_A \in$ AEREO
($t[IdVolo] = t_V[IdVolo] \wedge$
$t_V[TipoAereo] = t_A[TipoAereo] \wedge$
$t_V[CittaArr] =$ “Roma” $\wedge$
$t_A[QtaMerci] = 0$) $\}$

Datalog

RomaSoloPass(IdV) :- VOLO(IdV, _, _, _, “Roma”, _, T),
AEREO(T, _, Qtà), Qtà = 0

? - RomaSoloPass(IdVolo)

1.3.3 *Trovare gli aeroporti che* ***non*** *hanno partenze per Parigi prima delle 8:00.*

Algebra relazionale

$\Pi_{Città}$ AEROPORTO − $\rho_{Città \leftarrow CittaPar}$ $\Pi_{CittaPar}$ $\sigma_{CittaArr=\text{“Parigi”} \wedge OraPar<8:00}$ VOLO

Calcolo relazionale

$\{ t \mid \exists t_A \in$ AEROPORTO
($t[Città] = t_A[Città] \wedge$
$\neg$ ($\exists t_V \in$ VOLO
($t_V[CittaPar] = t_A[Città] \wedge$
$t_V[CittaArr] =$ 'Parigi' $\wedge$
$t_V[OraPar] < 8:00$))) $\}$

Datalog

ParteParigiPrima8(Aer) :- VOLO(_, _, Aer, Ora, 'Parigi', _, _),
Ora < 8:00

NonParteParigiPrima8(A) :- AEROPORTO(A, _, _),
¬ ParteParigiPrima8(A)

? - NonParteParigiPrima8(Città)

Esercizio 1.4: Restrizione dello schema dei trasporti aerei

VOLO (Partenza, Arrivo)

1.4.1 *Determinare tutti i possibili collegamenti tra aeroporti per realizzare i quali si debbano prendere non più di due voli diretti.*

Algebra relazionale

$\text{VOLO} \cup \Pi_{\text{Partenza, Arrivo}} (\ (\rho_{\text{Coinc}\leftarrow\text{Arrivo}} \text{VOLO}) \bowtie (\rho_{\text{Coinc}\leftarrow\text{Partenza}} \text{VOLO}))$

Si noti che il join naturale confronta gli attributi opportunamente ridenominati.

Calcolo relazionale

$\{ t \mid (\exists t_V \in \text{VOLO}$
$(t[\text{Partenza, Arrivo}] = t_V[\text{Partenza, Arrivo}])) \vee$
$(\exists t_{V1} \in \text{VOLO}, \exists t_{V2} \in \text{VOLO}$
$(t[\text{Partenza}] = t_{V1}[\text{Partenza}] \wedge$
$t[\text{Arrivo}] = t_{V2}[\text{Arrivo}] \wedge$
$t_{V1}[\text{Arrivo}] = t_{V2}[\text{Partenza}]) \ \}$

Datalog

Collagamento1(P, A) :- VOLO(P, A)

Collegamento1(P, A) :- VOLO(P, X), VOLO(X, A)

? - Collegamento1(Da, A)

1.4.2 *Determinare tutti i possibili collegamenti tra aeroporti per realizzare i quali si debbano prendere al massimo tre voli diretti.*

Algebra relazionale

$\text{VOLO} \cup$
$\Pi_{\text{Partenza,Arrivo}} (\ (\rho_{\text{Coinc}\leftarrow\text{Arrivo}} \text{VOLO}) \bowtie (\rho_{\text{Coinc}\leftarrow\text{Partenza}} \text{VOLO}) \ ) \cup$
$\Pi_{\text{Partenza,Arrivo}} (\ (\rho_{\text{Coinc1}\leftarrow\text{Arrivo}} \text{VOLO}) \bowtie (\rho_{\text{Coinc1}\leftarrow\text{Partenza, Coinc2}\leftarrow\text{Arrivo}} \text{VOLO})$
$\bowtie (\rho_{\text{Coinc2}\leftarrow\text{Partenza}} \text{VOLO}) \ )$

Calcolo relazionale

$\{ t \mid (\exists t_V \in \text{VOLO}$
$(t[\text{Partenza, Arrivo}] = t_V[\text{Partenza, Arrivo}])) \vee$
$(\exists t_{V1} \in \text{VOLO}, \exists t_{V2} \in \text{VOLO}$
$(t[\text{Partenza}] = t_{V1}[\text{Partenza}] \wedge$
$t[\text{Arrivo}] = t_{V2}[\text{Arrivo}] \wedge$
$t_{V1}[\text{Arrivo}] = t_{V2}[\text{Partenza}]) \vee$
$(\exists t_{V1} \in \text{VOLO}, \exists t_{V2} \in \text{VOLO}, \exists t_{V3} \in \text{VOLO}$
$(t[\text{Partenza}] = t_{V1}[\text{Partenza}] \wedge$
$t[\text{Arrivo}] = t_{V3}[\text{Arrivo}] \wedge$
$t_{V1}[\text{Arrivo}] = t_{V2}[\text{Partenza}] \wedge$
$t_{V2}[\text{Arrivo}] = t_{V3}[\text{Partenza}]) \}$

Datalog

Collegamento2(P, A) :- VOLO(P, A)

Collegamento2(P, A) :- VOLO(P, X), VOLO(X , Y), VOLO(Y, A)

? - Collegamento2(Partenza, Arrivo)

1.4.3 *Determinare tutti i possibili collegamenti tra aeroporti.*

Occorre generalizzare le soluzioni agli esercizi 1.4.1 e 1.4.2 al caso di un numero arbitrario di voli intermedi. È necessaria la ricorsione, possibile solo in Datalog.

Algebra relazionale

L'interrogazione non è esprimibile.

Calcolo relazionale

L'interrogazione non è esprimibile.

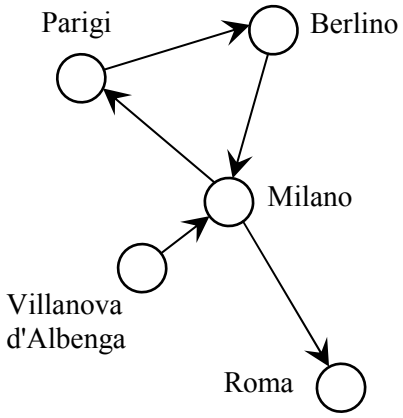
Datalog

CollegamentoN(P, A) :- VOLO(P, A) *(a)*

CollegamentoN(P, A) :- VOLO(P, X),
CollegamentoN(X, A) *(b)*

? - CollegamentoN(Partenza, Arrivo)

Vale la pena di osservare che l'esecuzione della regola *(b)* termina sempre, anche in presenza di eventuali cicli nella base dei fatti (combinazioni di voli per cui si può tornare nell'aeroporto da cui si è partiti). L'esempio seguente mostra un'istanza contenente cicli in cui con tre esecuzioni si raggiunge il *punto fisso* (cioè una situazione in cui anche rivalutando le regole non vi sono nuovi fatti da aggiungere).



Il contenuto della base di fatti VOLO che corrisponde alla situazione illustrata dal grafo qui a lato è riportato qui sotto:

	VOLO Partenza	Arrivo
(1)	Berlino	Milano
(2)	Milano	Parigi
(3)	Milano	Roma
(4)	Parigi	Berlino
(5)	Villanova	Milano

Simuliamo la valutazione della query costruendo il risultato, mostrato nella tabella qui a destra. Accanto alla tabella indichiamo: a sinistra gli indici dei "nuovi" fatti e le esecuzioni delle regole, a destra gli indici dei "vecchi" fatti che causano l'aggiunta dei nuovi.

Inizialmente il risultato è vuoto, quindi la prima esecuzione riguarda la sola regola *(a)*, che genera i primi cinque fatti per copia diretta di VOLO.

La regola *(b)* interviene dalla seconda esecuzione; l'ordine di inserimento nella simulazione è dato dalla valutazione di *(b)* considerando ogni fatto di CollegamentoN per ciascun fatto di VOLO, inserendo solo i fatti non ancora presenti.

		CollegamentoN Decollo	Atterraggio	
1:(a)	[1]	Berlino	Milano	(1)
	[2]	Milano	Parigi	(2)
	[3]	Milano	Roma	(3)
	[4]	Parigi	Berlino	(4)
	[5]	Villanova	Milano	(5)
2:(b)	[6]	Berlino	Parigi	(1)+[2]
	[7]	Berlino	Roma	(1)+[3]
	[8]	Milano	Berlino	(2)+[4]
	[9]	Parigi	Milano	(4)+[1]
	[10]	Villanova	Parigi	(5)+[2]
	[11]	Villanova	Roma	(5)+[3]
3:(b)	[12]	Berlino	Berlino	(1)+[8]
	[13]	Milano	Milano	(2)+[9]
	[14]	Parigi	Parigi	(4)+[6]
	[15]	Parigi	Roma	(4)+[7]
	[16]	Villanova	Berlino	(5)+[8]

Solo alla terza esecuzione si "scopre" la possibilità di ritornare al punto di partenza percorrendo l'unico ciclo del grafo – è una naturale conseguenza del fatto che a nessun volo di "andata" corrisponde un "ritorno" diretto in VOLO. Una quarta esecuzione non aggiungerebbe alcun nuovo fatto: si è raggiunto il *punto fisso*.

Si noti che un'esplorazione ricorsiva del grafo dovrebbe, per evitare di entrare in ciclo, marcare esplicitamente i nodi – quella proposta è una "elegante e concisa" specifica intensionale delle proprietà di raggiungibilità di una rete. Il numero di esecuzioni è pari alla lunghezza dei più lunghi tra i "cammini minimi" possibili tra tutte le coppie di nodi, poiché ogni coppia è aggiunta una sola volta, e le connessioni possibili sono enumerate in ordine di lunghezza crescente.

Esercizio 1.5: Centraline di rilevamento dell'inquinamento

CENTRALERILEVAMENTO (Codice, Indirizzo, CodComune)
OSSERVAZIONE (Centrale, Data, Ora, PM10, CO, NOx, HC)
COMUNE (Codice, Nome, Regione)
SOGLIA (Regione, SogliaPM10, SogliaCO, SogliaNOx, SogliHC)

1.5.1 *Estrarre le centrali di rilevamento che non hanno prodotto alcuna osservazione in un qualche giorno del 2001. Si assuma che esista nella base di dati almeno una osservazione per ogni giorno dell'anno (in nessun giorno, cioè, tutte le centrali sono state contemporaneamente spente).*

Una difficoltà in Algebra e Datalog è l'indisponibilità di un "calendario" dei giorni del 2001, da usarsi per escludere la presenza di osservazioni per uno di tali giorni.

Algebra relazionale

$$\Pi_{Codice} \; (\; (\; (\Pi_{Codice} \; \text{CENTRRILEV}) \times (\Pi_{Data} \; \sigma_{Data>31/12/00 \wedge Data<1/1/02} \; \text{OSSERVAZIONE}) \;) \; - \; (\; \rho_{Codice \leftarrow Centrale} \; (\Pi_{Centrale,\, Data} \; \text{OSSERVAZIONE}) \;) \;)$$

Il "calendario" è costruito sfruttando l'ipotesi, menzionata nel testo, secondo la quale per ogni giorno esiste almeno un'osservazione. In base a questa informazione, possiamo definire il calendario proiettando su Data le osservazioni del 2001. Il prodotto cartesiano genera tutte le coppie ⟨Centrale, Data⟩ da cui sottraiamo le coppie effettivamente corrispondenti a un'osservazione.

Calcolo relazionale

$$\{t \mid \exists\, t_C \in \text{CENTRALERILEVAMENTO} \; (t[Codice] = t_C[Codice] \wedge \neg (\exists\, t_O \in \text{OSSERVAZIONE} \; (\; t_O[Data] > 31/12/00 \; \wedge \; t_O[Data] < 1/1/02 \wedge t_O[Centrale] = t_C[Codice]) \;) \;) \}$$

Si noti quanto è più agevole negare l'esistenza di un'osservazione con ¬ (∃…) !

Datalog

```
Calendario2001( Data ) :- OSSERVAZIONE( _,Data,_,_,_,_,_ ),
                          Data < 1/1/02, Data > 31/12/00

EsisteOss( Ce, Da ) :- OSSERVAZIONE( Ce,Da,_,_,_,_,_ )

CentrSpentaAlmeno1DìNel2001( C ) :- CENTRALERILEVAMENTO( C,_,_ ),
                                    Calendario2001( D ), ¬ EsisteOss( C, D )

? - CentrSpentaAlmeno1GiornoNel2001( X )
```

Il principio è lo stesso adottato nella soluzione in algebra relazionale.

Capitolo Secondo
SQL

Esercizio 2.1: Semplicissimi esercizi introduttivi

IMPIEGATO (Nome, Qualifica, Salario, Dip)
ACQUISTO (IdCliente, DataDiOrdinazione, Prodotto, Quantità, Prezzo)
CLIENTE (Id, Nome, Città, Nazione)

Di tutti gli acquisti del cliente con id=10449, mostrare prodotti, quantità e prezzi:

```
select Prodotto, Prezzo
from Acquisto
where id = 10449
```

Nome, qualifica e salario di tutti i programmatori con salario di almeno 50000 €::

```
select Nome, Qualifica, Salario
from Impiegato
where Salario >= 50000 and Qualifica = 'Programmatore'
```

Nome e salario degli impiegati dei dipartimenti 'Vendite' e 'Sviluppo':

```
select Nome, Salario
from Impiegato
where Dip = 'Vendite' or Dip = 'Sviluppo'
```

I dati degli impiegati del dipartimento Vendite in ordine di stipendio crescente:

```
select *
from Impiegato
where Dip = 'Vendite'
order by Salario
```

Tutti i dati degli impiegati del dipartimento Vendite, in ordine ascendente di stipendio e, a parità di stipendio, in ordine discendente di età:

```
select *
from Impiegato
where dip = 'Vendite'
order by Salario, Età desc
```

L'elenco dei dipartimenti dell'azienda (ogni dipartimento compaia una sola volta):

```
select distinct Dip
from Impiegato
order by Dip
```

Lo stipendio medio degli impiegati:

```
select avg(Salario)
from Impiegato
```

Lo stipendio medio degli ingegneri:

```
select avg(Salario)
from Impiegato
where Qualifica like 'Ing%'
```

Il massimo stipendio di ogni dipartimento:

```
select Dip, max(Salario)
from Impiegato
group by Dip
```

N.B. Se c'è raggruppamento, solo gli attributi di raggruppamento *possono* comparire nella clausola select senza essere argomento di funzioni aggregate (possono, ma… *non devono*).

Il massimo stipendio di ogni dipartimento che abbia almeno 20 impiegati:

```
select Dip, max(Salario)
from Impiegato
group by Dip
having count(*) > 20
```

Tutti i dati degli impiegati Cole, Jones e Tibbs:

```
select *
from Impiegato
where Nome in ('Cole', 'Jones', 'Tibbs')
```

oppure

```
where Nome='Cole' or Nome='Jones' or Nome='Tibbs'
```

N.B.: per estrarre *tutti i dati di tutti gli impiegati **tranne** Cole, Jones e Tibbs*:

```
where Nome not in ('Cole', 'Jones', 'Tibbs')
```

oppure

```
where Nome<>'Cole' and Nome<>'Jones' and Nome<>'Tibbs'
```

Tutti i dati degli impiegati di età compresa tra 25 e 35 anni:

```
select *
from Impiegato
where Età between 25 and 35
```

oppure

```
where Età >= 25 and Età <= 35
```

Estrarre, per ogni acquisto, il nome del cliente e il prezzo del prodotto:

```
select nome, prezzo
from cliente, acquisto
where cliente.id = acquisto.idcliente
```

oppure

```
select nome, prezzo
from cliente C join acquisto A on C.id = A.idcliente
```

Esercizio 2.2: Aeroporti (lo schema è già stato commentato al paragrafo 1.3)

AEROPORTO (Città, Nazione, NumPiste)
VOLO (IdVolo, GiornoSett, CittàPart, OraPart, CittàArr, OraArr, TipoAereo)
AEREO (TipoAereo, NumPasseggeri, QtaMerci)

2.2.1 *Trovare le città da cui partono voli diretti a Roma, ordinate alfabeticamente:*

```
select CittàPar
from Volo
where CittàArr = 'Roma'
order by CittàPar
```

2.2.2 *Trovare le città con un aeroporto di cui non è noto il numero di piste:*

```
select Città
from Aeroporto
where NumPiste is null
```

2.2.3 *Di ogni volo misto (merci e passeggeri) estrarre codice e i dati sul trasporto:*

```
select IdVolo, NumPasseggeri, QtaMerci
from  Volo as V, Aereo as A
where  V.TipoAereo = A.TipoAereo and
       NumPasseggeri > 0 and QtaMerci > 0
```

oppure

```
select IdVolo, NumPasseggeri, QtaMerci
from Volo V join Aereo A on V.TipoAereo = A.TipoAereo
where NumPasseggeri > 0 and QtaMerci > 0
```

2.2.4 *Le nazioni da cui parte e in cui arriva il volo AZ274:*

```
select A1.Nazione, A2.Nazione
from (Aeroporto A1 join Volo on A1.Città = CittàArr)
            join Aeroporto A2 on CittàPar = A2.Città
where IdVolo = 'AZ274'
```

Per includere le due nazioni occorre fare due volte il join con AEROPORTO: A2 rappresenta l'aeroporto di partenza, mentre A1 l'aeroporto d'arrivo.

2.2.5 *I tipi di aereo e il corrispondente numero di passeggeri per i tipi di aereo usati nei voli che partono da Torino. Se i dati dell'aereo non sono disponibili visualizzare solo il tipo (cioè se non esiste una tupla corrispondente nella tabella Aereo – quindi non c'è integrità referenziale*).

```
select Volo.TipoAereo, NumPasseggeri
from Volo V left join Aereo A
        on V.TipoAereo = A.TipoAereo
where CittàPar = 'Torino'
```

2.2.6 *Trovare l'aeroporto italiano con il maggior numero di piste:*

```
select Città, max(NumPiste)
from  Aeroporto
where Nazione = 'Italia'
```

← ***ERRORE TIPICO***

La soluzione precedente è sintatticamente scorretta perché inserisce nella clausola `select` un attributo e un aggregato senza che si sia raggruppato per il primo.

```
select Città, max(NumPiste)
from  Aeroporto
where Nazione = 'Italia'
group by Città
```

← ***ERRORE TIPICO***

Questa soluzione è invece sintatticamente corretta ma semanticamente scorretta: il comportamento di questa interrogazione è di raggruppare per città, e quindi mostrare tutte le città e il massimo numero di piste per ognuna di tali città. Siccome la città è chiave per la tabella Aeroporto, di fatto l'interrogazione estrae *tutte* le città, con il rispettivo numero di piste!

```
select Città, NumPiste
from  Aeroporto
where Nazione = 'Italia' and
      NumPiste = ( select max(numPiste)
                   from Aeroporto
                   where Nazione = 'Italia' )
```

← ***Versione corretta***

2.2.7 *Di ogni nazione, trovare quante piste ha l'aeroporto con più piste:*

```
select Nazione, max(NumPiste)
from  Aeroporto
group by Nazione
```

2.2.8 *Di ogni nazione, trovare quante piste ha l'aeroporto con più piste, purché le piste siano almeno 3:*

```
select Nazione, max(NumPiste)
from  Aeroporto
group by Nazione
having max(NumPiste) > 2
```

Questa soluzione dapprima raggruppa **tutte** le tuple, e poi considera solo i gruppi di tuple (a pari nazione) in cui il massimo numero di piste sia 3 o più. In alternativa avremmo potuto da subito scartare gli aeroporti con un numero insufficiente di piste e poi raggruppare le tuple restanti a pari nazione:

```
select Nazione, max(NumPiste)
from  Aeroporto
where NumPiste > 2
group by Nazione
```

Per includere nel risultato anche la città bisogna cambiare strategia (non si può inserire l'attributo città nella target list, perché esso non compare nella group by).

2.2.9 *Trovare le città in cui si trovano gli aeroporti con più piste di ogni nazione, indicando città, nazione e numero di piste:*

```
select *
from Aeroporto A
where NumPiste = ( select max(NumPiste)
                   from Aeroporto B
                   where B.Nazione = A.Nazione )
```

Alternativamente, riutilizzando la soluzione data alla query 2.2.7:

```
select *
from Aeroporto
where (Nazione, NumPiste) in ( select Nazione, max(NumPiste)
                               from Aeroporto
                               group by Nazione)
```

2.2.10 *Trovare gli aeroporti da cui **partono** voli internazionali:*

```
select distinct CittàPar
from  (Aeroporto as A1 join Volo on CittàPar = A1.Città)
            join Aeroporto as A2 on CittàArr = A2.Città
where A1.Nazione <> A2.Nazione
```

Il distinct è "essenziale" per la chiarezza e leggibilità del risultato.

2.2.11 *Trovare il numero **totale** di **partenze** internazionali del giovedì:*

```
select count(*)
from  (Aeroporto as A1 join VOLO on CittàPar = A1.Città)
            join Aeroporto as A2 on CittàArr = A2.Città
where A1.Nazione <> A2.Nazione and GiornoSett = 'Giovedì'
```

2.2.12 *Trovare il numero di **aeroporti** da cui partono voli internazionali il giovedì:*

```
select count( distinct CittàPar )
from (Aeroporto as A1 join Volo on CittàPar = A1.Città)
            join Aeroporto as A2 on CittàArr = A2.Città
where A1.Nazione <> A2.Nazione and GiornoSett = 'Giovedì'
```

2.2.13 *Trovare il numero di partenze internazionali del giovedì **da ogni aeroporto**:*

```
select CittàPar, count(*) as NumPartInt
from (Aeroporto as A1 join Volo on CittàPar = A1.Città)
            join Aeroporto as A2 on CittàArr = A2.Città
where A1.Nazione <> A2.Nazione and GiornoSett = 'Giovedì'
group by CittàPar
```

2.2.14 *Trovare le città francesi da cui partono più di 20 voli* ***a settimana*** *per la Germania:*

```
select CittàPar, count(*) as NumVoliGer
from  (Aeroporto as A1 join Volo on CittàPar = A1.Città)
           join Aeroporto as A2 on CittàArr = A2.Città
where A1.Nazione = 'Francia' and A2.Nazione = 'Germania'
group by CittàPar
having count(*) > 20
```

2.2.15 *Trovare il numero di* ***voli del giovedì*** *di ogni aeroporto* ***da cui partono almeno 100 voli a settimana****:*

```
select CittàPart, count(*)          ← ERRORE TIPICO
from  Volo
where GiornoSett = 'Giovedì'
group by CittàPart
having count(*) > 100
```

Il secondo count deve contare **tutti** i voli dell'aeroporto, non solo quelli del giovedì. La clausola where è valutata *prima* di group by e having, e ne influenza la valutazione (solo le tuple che "passano" la where subiscono il raggruppamento).

```
select CittàPart, count(*)          ← Versione corretta
from  Volo
where GiornoSett = 'Giovedì' and
      CittàPart in ( select CittàPart
                     from Volo
                     group by CittàPart
                     having count(*) > 100 )
group by CittàPart
```

Esercizio 2.3: Gestione Ordini

PRODOTTI (Codice, Nome, Descrizione, Peso, Costo)
LINEEORDINE (Ordine, Prodotto, Quantità, Ammontare)
ORDINI (CodOrdine, Cliente, Data, Totale, Sconto)
CLIENTI (Codice, Nome, Indirizzo, Categoria)

2.3.1 *Trovare l'ordine più* ***pesante*** *(si badi a sommare i pesi in base alle quantità):*

Calcoliamo il risultato attraverso una view intermedia.

```
create view PesoOrdini as
  select Ordine, sum(Quantità*Peso) as PesoTot
  from LineeOrdini join Prodotti on Prodotto = Codice
  group by Ordine
```

```
select *
from Ordini join PesoOrdini on (CodOrdine = Ordine)
where PesoTot = ( select max(PesoTot)
                  from PesoOrdini )
```

2.3.2 *Formulare un comando che assegna il valore 'Abituale' all'attributo Categoria per i clienti che hanno ordinato più di 5 volte lo stesso prodotto:*

```
update Clienti
   set Categoria = 'Abituale'
   where Codice in ( select Cliente
                     from Ordini join LineeOrdine
                            on CodOrdine = Ordine
                     group by Cliente, Prodotto
                     having count(*) > 5 )
```

2.3.3 *Formulare un'asserzione SQL che verifichi che per ogni ordine la somma dell'ammontare delle linee sia pari alla somma tra Totale e Sconto:*

```
create assertion OrdineCorretto check
  ( not exists ( select *
                 from Ordini O
                 where Totale + Sconto <> all
                      ( select sum(Ammontare)
                        from LineeOrdine
                        where Ordine = CodOrdine) ) )
```

Esercizio 2.4: Campionato di calcio

Il seguente schema relazionale descrive i campionati di calcio. La tabella CLASSIFICA contiene una "fotografia" della situazione di ogni squadra alla fine di ogni giornata di ogni campionato (anche Giornata, infatti, è parte della chiave).

PARTITA (Anno, Giornata, SquadraCasa, SquadraOspite, RetiCasa, RetiOspite)
CLASSIFICA (Anno, Giornata, Squadra, TotPunti, Posizione)
SQUADRA (Nome, Citta, NomeStadio)

2.4.1 *Trovare la squadra che ha conseguito il massimo numero di vittorie:*

```
create view Vittoria (Anno, Giorn, Squadra) as
( select Anno, Giornata, SquadraCasa
  from Partita
  where RetiCasa > RetiOspite )
      union (all)
( select Anno, Giornata, SquadraOspite
  from Partita
  where RetiOspite > RetiCasa )
```

Aggiungere all'operatore `union` l'opzione `all` **non** è necessario, perché le due query non possono restituire duplicati (in ogni giornata una squadra non può giocare sia in casa sia fuori casa!). Usando la view scriviamo poi la soluzione:

```
select Squadra
from Vittoria
group by Squadra
having count(*) >= all ( select count(*)
                         from Vittoria
                         group by Squadra )
```

2.4.2 *Definire in SQL un comando di aggiornamento che assegni all'attributo TotPunti il saldo dei punti fino alla giornata corrente:*

Definiamo due comandi da eseguirsi in sequenza. Il primo *assegna* i punti dovuti alle vittorie, il secondo incrementa il valore con i punti dovuti ai pareggi.

```
update Classifica C
set TotPunti = ( select count(*) * 3
                 from Vittoria V
                 where V.Anno = C.Anno and
                       V.Giornata <= C.Giornata and
                       V.Squadra = C.Squadra )
update Classifica C
set TotPunti = TotPunti+( select count(*)
                          from Partita P
                          where P.Anno = C.Anno
                            and P.Giornata <= C.Giornata
                            and (P.SquadraCasa = C.Squadra or
                                 P.SquadraOspite = C.Squadra)
                            and P.RetiCasa = P.RetiOspite )
```

2.4.3 *Definire in SQL un comando di aggiornamento che assegni all'attributo Posizione di Classifica la posizione in classifica della squadra, assumendo che TotPunti già rappresenti i punti conquistati dalla squadra fino a quella giornata (inclusa):*

La posizione in classifica è (per definizione) pari a 1 più il numero di squadre che, a quella giornata, hanno accumulato un maggior numero di punti. Questa soluzione tratta correttamente anche il parimerito.

```
update Classifica C
set Posizione = 1 + ( select count(*)
                      from Classifica C1
                      where C1.Anno = C.Anno
                          and C1.Giornata = C.Giornata
                          and C1.TotPunti > C.TotPunti )
```

Esercizio 2.5: Corsi universitari

STUDENTI (Matr, Nome, Residenza,Telefono, CorsoDiLaurea, Anno, Sede)
ISCRIZIONI (MatrStud, Corso, Anno, Data)
CORSIANNI (CodCorso, Anno, Docente, Semestre, NroStudenti)
ABBINAMENTI (CodCorso, CorsoLaurea)
CORSI (CodCorso, Titolo, Crediti, Sede)

2.5.1 *Formulare un comando SQL che per tutti i corsi del 2000 assegna all'attributo NumStudenti il numero di studenti che risultano iscritti al corso:*

```
update corsiAnni CA
    set NroStudenti = ( select count(*)
                        from Iscrizioni
                        where Anno = 2000 and
                              CA.CodCorso = CodCorso )
    where Anno = 2000
```

2.5.2 *Formulare un'interrogazione SQL che permette di estrarre gli studenti che hanno seguito dei corsi che non sono abbinati al proprio corso di laurea:*

```
select Matr
from Studenti S, Iscrizioni I, Abbinamenti A
where S.Matr = I.MatrStud and I.Corso = A.CodCorso and
      S.CorsoDiLaurea <> A.CorsoLaurea
```

Esercizio 2.6: Catalogo di fornitura prodotti

FORNITORI (CodiceFornitore, Nome, Indirizzo, Città)
PRODOTTI (CodiceProdotto, Nome, Marca, Modello)
CATALOGO (CodiceFornitore, CodiceProdotto, Costo)

2.6.1 *Trovare i nomi dei fornitori di prodotti IBM:*

```
select distinct F.Nome
from Fornitori as F, Prodotti as P, Catalogo as C
where F.CodiceFornitore = C.CodiceFornitore and
      C.CodiceProdotto = P.CodiceProdotto and
      P.Marca = 'IBM'
```

2.6.2 *Trovare i codici dei prodotti distribuiti da almeno 2 fornitori:*

```
select distinct C.CodiceProdotto
from Catalogo C, Catalogo C1
where C.CodFornitore > C1.CodFornitore and
      C.CodiceProdotto = C1.CodiceProdotto
```

Si noti che al posto del > si sarebbe potuto usare < >. Tuttavia, attraverso l'uso del > si "dimezza" la dimensione della tabella coinvolta.
In alternativa, possiamo raggruppare i prodotti per codice e contarne le occorrenze:

```
select CodiceProdotto
from Catalogo
group by CodiceProdotto
having count(*) > 1
```

2.6.3 *Di ogni prodotto calcolare il costo medio di fornitura in ciascuna città:*

```
select CodiceProdotto, Città, avg(costo) as CostoMedio
from Catalogo C, Fornitori F
where C.CodiceForn = F.CodiceForn
group by Città, CodiceProdotto
```

N.B.: L'ordine degli attributi che compaiono nella clausola group by non è mai importante! (a differenza della clausola order by)

2.6.4 *Trovare il codice dei prodotti più costosi tra quelli distribuiti dai fornitori presenti a Roma:*

```
select distinct C.CodiceProdotto
from Fornitori as F, Catalogo as C
where F.CodiceFornitore = C.CodiceFornitore and
      F.Città = 'Roma' and
      C.Costo = ( select max(Costo)
                  from Fornitori as F1, Catalogo as C1
                  where F1.CodiceFornitore=C1.CodiceFornitore
                    and F1.Città = 'Roma' )
```

2.6.5 *Trovare i nomi dei fornitori che distribuiscono tutti i prodotti in catalogo (cioè tutti i prodotti noti al sistema):*

Possiamo estrarre i fornitori per i quali ci sono nel catalogo tante tuple quanti sono i prodotti. Si noti la presenza dell'attributo Nome nella group by, che non modifica il raggruppamento ma consente di estrarre l'attributo nella clausola select.

```
select Nome
from Fornitori F join Catalogo C
        on F.CodiceFornitore = C.CodiceFornitore
group by F.CodiceFornitore, Nome
having count(*) = ( select count(*) from Prodotti )
```

In alternativa, possiamo estrarre tutti i fornitori "tali per cui non esiste un prodotto in catalogo che essi non distribuiscono":

```
select Codice, Nome
from Fornitori as F
where not exists
       ( select P.CodiceProdotto
         from Prodotti as P
         where not exists
                ( select C1.CodiceProdotto
                  from Catalogo as C1
                  where C1.CodiceFornitore=F.CodiceFornitore
                    and C1.CodiceProd = P.CodiceProd ) )
```

Esercizio 2.7: Campionato di calcio (variante)

PARTITA (Codice, Campionato, Serie, Data, Stadio)
RISULTATO (Squadra, Partita, Punti, Reti)

Lo schema è relativo ai campionati di serie A, B, C. Ad ogni *partita giocata* è assegnato un codice univoco, per comodità. Ogni partita è (ovviamente) giocata da due squadre. Si noti che la registrazione dei punti è *ridondante*, in quanto deducibile dal confronto delle reti di squadre diverse nella stessa partita, ma molto *comoda* (occorrono due join per confrontare le reti segnate dalle due squadre).

2.7.1 *Trovare le squadre che hanno giocato sempre e solo in serie A:*

```
select distinct Squadra
from Risultato
where Squadra not in
      ( select distinct Squadra
        from Risultato join Partita on Partita = Codice
        where Serie <> 'A' )
```

2.7.2 *Trovare la squadra che ha conquistato più punti in un campionato di serie A:*

Con una view calcoliamo i punti totalizzati in ogni campionato da ogni squadra.

```
create view PuntiTotInA(Camp, Squadra, PTot) as
( select Campionato, Squadra, sum(Punti)
  from Partita, Risultato
  where Codice = Partita and Serie = 'A'
  group by Squadra, Campionato )

select Squadra, PTot
from PuntiTotInA
where PTot = ( select max(PTot)
               from PuntiTotInA )
```

2.7.3 *Costruire il calendario, con i risultati, delle 34 giornate del campionato di serie A '01/'02 in ordine cronologico. Si noti che il calendario ha 34*9 = 306 tuple (una tupla per ogni partita, campionato a 18 squadre):*

Attenzione: lo schema non permette di stabilire con certezza (né di calcolare per tutte le squadre) chi ha giocato in casa; includiamo allora nel risultato anche lo Stadio, che può aiutare a capirlo.
Però resta ancora il problema dei derby e dei campi neutri… quindi imponiamo di visualizzare *prima* il nome della *vincitrice*, e in caso di *pareggio* di adottare, ad esempio, il criterio *lessicografico*.

```
select Data,r1.Squadra,r2.Squadra,r1.Reti,r2.Reti,Stadio
from Partita, Risultato r1, Risultato r2
where Serie = 'A' and
      Campionato = '01-02' and
      r1.Partita = Codice and
      r2.Partita = Codice and
      ( r1.Reti > r2.Reti
            or
        r1.Reti = r2.Reti and r1.Squadra < r2.Squadra )
order by Data
```

Esercizio 2.8: Andiamo al cinema

REGISTA (Nome, DataNascita, Nazionalità)
ATTORE (Nome, DataNascita, Nazionalità)
INTERPRETA (Attore, Film, Personaggio)
FILM (Titolo, NomeRegista, Anno)
PROIEZIONE (NomeCin, CittàCin, TitoloFilm)
CINEMA (Città, NomeCinema, #Sale, #Posti)

Si noti che, in virtù della chiave della tabella INTERPRETA, un attore può interpretare lo stesso personaggio in più film (tipicamente nel caso di sequel o serie), più personaggi nello stesso film (vi sono numerosi esempi); è anche possibile che un personaggio sia interpretato da più di un attore (da giovane e da vecchio, ad esempio). Si noti poi che le proiezioni sono indicate senza alcun riferimento temporale, quindi sono da interpretarsi come "attualmente in proiezione" – il database non rappresenta cioè la serie storica degli spettacoli. Infine, si noti che possiamo associare un numero arbitrario di proiezioni a un cinema, e non c'è modo di legare i film alle sale (delle quali si conosce solamente il numero, per ciascun cinema).

2.8.1 *Scriviamo i comandi SQL che definiscono alcune parti di questo schema*:

```
create table REGISTA (
  Nome varchar(30) primary key,
  DataNascita date,
  Nazionalità varchar(30) not null )

create domain AnnoFilm integer > 1870

create table FILM (
  Titolo varchar(100) primary key,
  NomeRegista varchar(30) references REGISTA(Nome)
                          on delete no action
                          on update cascade,
  Anno AnnoFilm )

create table PROIEZIONE (
      NomeCin varchar(30),
      CittàCin varchar(30),
      TitoloFilm varchar(100) references FILM(Titolo)
                                      on delete set null
                                      on update cascade,
      primary key (NomeCinema, Città, TitoloFilm),
      foreign key (NomeCin, CittàCin)
                  references CINEMA(NomeCinema, Città)
                                  on delete set null
                                  on update cascade )
```

ATTENZIONE! TitoloFilm è parte della chiave! Quindi la politica di reazione al cambiamento del titolo di un film non può essere "set null": può essere solamente no action o cascade. Lo stesso discorso vale per NomeCin e per CittàCin.

2.8.2 *Modifichiamo* ***lo schema****, aggiungendo un attributo 'Tipo' alla tabella Film:*

```
alter table FILM
  add column Tipo varchar(20) default 'Normale'
```

2.8.3 *Diamo ora al nuovo attributo il valore "Flop" se il film è attualmente in proiezione in meno di 10 cinema:*

```
update FILM set Tipo = 'Flop'
     where 10 > ( select count(*)
                  from PROIEZIONE
                  where TitoloFilm = Titolo )
```

Prima di passare alle interrogazioni, esprimiamo il seguente vincolo, che, come si è visto precedentemente, non è espresso strutturalmente dallo schema:

2.8.4 *(Vincolo) In nessun cinema possono esserci in proiezione più film di quante sono le sale del cinema.*

Il vincolo può essere espresso nella definizione della tabella CINEMA:

```
create table CINEMA (
 Città      varchar(30),
 NomeCinema varchar(30),
 #Sale      integer check( #Sale > 0 and
                           #Sale >= (select count(*)
                                     from PROIEZIONE
                                     where Nome=NomeCinema
                                       and Città=CittàCin) ),
 #Posti integer > 0 )
```

Oppure può essere espresso in forma di asserzione:

```
create assertion NonTroppiFilm check (
  not exists ( select *
               from  CINEMA C
               where #Sale < (select count(*)
                              from PROIEZIONE
                              where C.Nome=NomeCinema
                                and C.Città=CittàCin) ) )
```

2.8.5 *Estrarre le nazionalità dei registi che hanno diretto qualche film nel 1992 ma non hanno diretto alcun film nel 1993:*

```
select distinct Nazionalità
from REGISTA
where Nome in ( select NomeRegista
                from FILM
                where Anno = 1992 )
  and Nome not in ( select NomeRegista
                    from FILM
                    where Anno = 1993 )
```

In alternativa si può usare **`except`** (a patto di discriminare correttamente i registi). **Non** si può invece assolutamente usare un join tra FILM e REGISTA seguito dalla condizione:

~~`where Anno = 1992 and Anno <> 1993`~~

perché la clausola `where` agisce a livello di tupla (in ogni tupla `Anno` ha un solo valore e se il valore di `Anno` è 1992 esso è banalmente diverso da 1993).

```
select Nazionalità
from REGISTA
where Nome in
      ( select NomeRegista
        from FILM where Anno = 1992
            except
        select NomeRegista
        from FILM where Anno = 1993 )
```

Non si può usare **except** direttamente se nella clausola select non sono inclusi attributi discriminanti per l'esclusione. Ad esempio, nella soluzione seguente un solo regista di nazionalità *n* che abbia girato un film nel '93 farebbe sparire dalla soluzione il valore *n*, anche se ci fossero molti altri suoi connazionali che soddisfano l'interrogazione. **Attenzione**: si ricordi che in SQL gli operatori insiemistici (senza all) eliminano i duplicati dagli operandi a cui sono applicati.

```
select Nazionalità
from FILM join REGISTA on NomeRegista = Nome
where Anno = 1992

      except

select Nazionalità
from FILM join REGISTA on NomeRegista = Nome
where Anno = 1993
```

2.8.6 Trovare le date di nascita dei registi che hanno diretto film in proiezione sia a Torino sia a Milano:

```
select distinct DataNascita
from REGISTA join FILM on Nome = NomeRegista
where Titolo in
         ( select TitoloFilm
           from PROIEZIONE
           where CittàCin = 'Milano' )
  and Titolo in
         ( select TitoloFilm
           from PROIEZIONE
           where CittàCin = 'Torino' )
```

Qui è ancor più evidente che *`where Città = 'Torino' and Città = 'Milano'`* sarebbe un predicato palesemente auto-contraddittorio.

In alternativa si può usare **intersect** (sempre utilizzando la chiave).

```
select DataNascita
from REGISTA
where Nome in
  ( select NomeRegista
    from FILM join PROIEZIONE on Titolo = TitoloFilm
    where Città = 'Milano'
      intersect
    select NomeRegista
    from FILM join PROIEZIONE on Titolo = TitoloFilm
    where Città = 'Torino' )
```

Non si può usare **`intersect`** direttamente se nella clausola select non sono inclusi attributi discriminanti per l'esclusione:

```
select DataNascita
from (FILM join REGISTA on Titolo = Nome)
         join PROIEZIONE on Titolo = TitoloFilm
where Città = 'Milano'

     intersect

select DataNascita
from (FILM join REGISTA on Titolo = Nome)
         join PROIEZIONE on Titolo = TitoloFilm
where Città = 'Torino'
```

Due registi *nati lo stesso giorno* tali che i film dell'uno siano stati proiettati *solo* a Torino e quelli dell'altro *solo* a Milano contribuirebbero erroneamente alla soluzione con la loro data di nascita.

2.8.7 *Trovare i nomi dei registi che hanno diretto nel 1993 più film di quanti ne abbiano diretti nel '92:*

```
select NomeRegista
from FILM as F
where Anno = 1993
group by NomeRegista
having count(*)  >  ( select count(*)
                      from  FILM  as  F1
                      where F1.NomeRegista = F.NomeRegista
                        and Anno = 1992 )
```

Che cosa succede se si invertono le condizioni sugli anni e il predicato di confronto, come nella versione seguente? È ancora corretta la soluzione seguente?

```
select NomeRegista
from FILM as F
where Anno = 1992
group by NomeRegista
having count(*) < ( select count(*)
                    from FILM as F1
                    where F1.NomeRegista = F.NomeRegista
                      and Anno = 1993 )
```

NO, è errata: non estrae i registi che hanno diretto film nel '93 ma non nel '92.

Si può invece usare una vista intermedia:

```
create view NumPerAnno (Nom, Ann, Num) as
select NomeRegista, Anno, count(*)
from FILM
group by NomeRegista, Anno

select Nom as NomeRegistaCercato
from NumPerAnno N1
where Ann = 1993 and
      Nom not in ( select Nom
                   from NumPerAnno N2
                   where N2.Ann = 1992 and
                         N1.Num <= N2.Num )
```

N.B.: un self join su NumPerAnno perderebbe chi non ha girato film nel 1992.

2.8.8 *Estrarre i film proiettati nel maggior numero di cinema di Milano:*

```
select TitoloFilm, count(*) as NumCin
from PROIEZIONE
where Città = 'Milano'
group by TitoloFilm
having count(*) >= all ( select count(*)
                         from PROIEZIONE
                         where Città = 'Milano'
                         group by TitoloFilm )
```

NumCin non è richiesto dalla specifica, ma migliora la leggibilità. In alternativa:

```
create view ProiezMilano (Titolo, Num) as
select TitoloFilm, count(*)
from PROIEZIONE
where Città = 'Milano'
group by TitoloFilm
```

```
select Titolo
from ProiezMilano
where Num = ( select max(Num)
              from ProiezMilano )
```

2.8.9 *Trovare gli attori che hanno interpretato più personaggi in uno stesso film:*

```
select distinct P1.Attore
from INTERPRETA P1,
     INTERPRETA P2
where P1.Attore = P2.Attore and
      P1.Film = P2.Film and
      P1.Personaggio <> P2.Personaggio
```

oppure

```
select distinct Attore
from INTERPRETA
group by Attore, Film
having count(*) > 1
```

2.8.10 *Trovare i film in cui recita un solo attore, che, però, interpreta più personaggi:*

Imponiamo le opportune condizioni sul numero di tuple dei gruppi di interpretazioni relative a uno stesso film:

```
select Film
from INTERPRETA
group by Film
having count(*) > 1 and
       count(distinct Attore) = 1
```

oppure

```
select X.Film
from INTERPRETA X,
     INTERPRETA Y
where X.Attore = Y.Attore and
      X.Film = Y.Film and
      X.Personaggio <> Y.Personaggio

  except

select M.Film
from INTERPRETA M,
     INTERPRETA P
where M.Film = P.Film and
      M.Attore <> P.Attore
```

2.8.11 *Trovare gli attori italiani che non hanno mai recitato con altri italiani:*

```
select Nome
from ATTORE A1
where Nazionalità = 'Italiana' and not exists (
       select *
       from INTERPRETA I1, INTERPRETA I2, ATTORE A2
       where I1.Titolo=I2.Titolo and
             I2.Attore=A2.Nome and I1.Attore=A1.Nome and
             A2.Nazionalità='Italiana' and A1.Nome<>A2.Nome )
```

L'ultima condizione serve a non ritrovare nella query annidata lo stesso attore A1 (altrimenti ogni attore risulterebbe aver recitato con se stesso in ogni suo film!).

2.8.12 *Trovare i registi che hanno recitato in (almeno) un loro film:*

```
select distinct NomeRegista
from FILM join INTERPRETA on Titolo = Film
where NomeRegista = Attore
```

2.8.13 *Trovare i registi che hanno recitato in almeno 4* ***loro*** *film interpretandovi in totale almeno 5 personaggi diversi:*

```
select NomeRegista
from FILM join INTERPRETA on Titolo = Film
where NomeRegista = Attore
group by NomeRegista
having count( distinct Titolo ) >= 4 and
       count( distinct Personaggio ) >= 5
```

2.8.14 *I registi che hanno recitato in* ***tutti*** *i loro film:*

```
select NomeRegista
from  REGISTA R
where not exists ( select * from FILM F
                   where not exists (select * from INTERPRETA
                                     where R.Nome = F.Regista
                                       and Film = F.Titolo
                                       and R.Nome = Attore) )
```

Esercizio 2.9: Quotazioni di borsa

TITOLO (Codice, Nome, …)
SCAMBIO (Titolo, Data, Orario, Acquirente, Venditore, Qtà, Prezzo)
QUOTAZIONE (Data, Titolo, Volume, Max, Min, Ufficiale, Riferimento)

SCAMBIO rappresenta l'avvenuto trasferimento di un lotto di titoli in un preciso momento a un dato prezzo unitario.
QUOTAZIONE riassume i dati di scambio di ogni titolo per ogni giornata (come da servizio RAI Televideo dopo la chiusura).

Volume: controvalore (in K€) scambiato in una giornata.
Ufficiale: prezzo medio nel giorno (pesato rispetto al volume).
Riferimento: prezzo medio pesato sull'ultimo **10%** di volume scambiato (gli scambi più prossimi alla chiusura).

Si chiede di scrivere i comandi che, dopo ogni giornata, aggiornano la tabella QUOTAZIONE in base ai nuovi scambi effettuati, aggiungendovi *una tupla per ogni titolo* del mercato. Così si potranno ottenere tutte le quotazioni di una giornata selezionando semplicemente da QUOTAZIONE in base alla Data, e l'andamento storico dei titoli selezionando invece in base al Titolo.

```
insert into Quotazione
( select Data, Titolo, sum(Qtà*Prezzo)/1000, max(Prezzo),
         min(Prezzo), sum(Qtà*Prezzo)/sum(Qtà), null
   from Titolo left join Scambio on Titolo = Codice
   where Data = ( select max(Data)
                  from Scambio )
      or Data is null
   group by Titolo, Data )
```

Il comando precedente calcola e aggiunge una tupla (relativa all'ultima giornata, cioè la più recente tra le date degli scambi) per *tutti* i titoli, cioè anche per quelli *non* scambiati in quella giornata, grazie all'outer join (gli aggregati per i gruppi relativi a quei titoli forniscono valori nulli).
Il valore di *Riferimento* (posto a null) sarà inserito in un secondo momento.
Nella clausola 'where' la condizione "or Data is null" è necessaria per trattenere le tuple relative ai titoli non scambiati, introdotte dall'outer join.
Questa soluzione è migliorabile, poiché è preferibile avere esplicitamente il valore 0 per il *Volume* relativo ai titoli non scambiati, essendo un dato esatto. I prezzi max, min, uff. e di rif. sono invece giustamente a null, in quanto *indefiniti*.

I titoli ***non*** scambiati possono essere trattati "separatamente".

```
insert into Quotazione
( (select Data, Titolo, sum(Qtà*Prezzo)/1000, max(Prezzo),
          min(Prezzo), sum(Qtà*Prezzo)/sum(Qtà), null
   from Scambio
   where Data = ( select max(Data)
                  from Scambio )
   group by Titolo, Data)
      union
  (select Data, Codice, 0, null, null, null, null
   from Titolo, Scambio
   where Data = ( select max(Data)
                  from Scambio )
     and (Data, Codice) not in ( select Data, Titolo
                                 from Scambio ) )     )
```

Per calcolare il valore di Riferimento occorre dapprima estrarre (per ogni titolo e per l'ultima giornata) l'insieme degli scambi tali per cui la somma dei loro volumi meglio approssima il 10% del volume totale giornaliero di scambi. Per far ciò ci appoggiamo ad una vista. È meglio approssimare per eccesso che per difetto, se riteniamo che nel caso ci sia *un solo* scambio sia preferibile che il Riferimento coincida con l'Ufficiale (rispetto a considerarlo 0 o null). La vista sceglie gli "scambi che sono nell'ultimo 10% o più", cioè tutti quelli tali per cui il volume scambiato *fino alla loro ora di effettuazione (compresa)* è maggiore del 90% del volume totale. Il volume totale può ovviamente essere letto dalla tabella QUOTAZIONE, dato che il precedente comando è già stato eseguito.

```
create view ScambiRecentiUltimo10 (Tit, Ora, Prez, Qtà) as
 select S.Titolo, S.Orario, S.Prezzo, S.Qtà
 from Scambio S join Scambio S1 on
         ( S.Titolo = S1.Titolo and S.Data = S1.Data )
 where S.Orario >= S1.Orario and
       S.Data = ( select max(Data)
                  from Scambio )
 group by S.Titolo, S.Orario, S.Prezzo, S.Qtà
 having sum(S1.Prezzo*S1.Qtà) > ( select 900 * Volume
                                  from Quotazione Q
                                  where Q.Data = S.Data and
                                        Q.Titolo = S.Titolo )

update Quotazione Q
set Q.Riferimento = ( select sum(X.Qtà*X.Prezzo) / sum(X.Qtà)
                      from ScambiRecentiUltimo10 as X
                      where X.Tit = Q.Titolo )
where Q.Data = ( select max(Data)
                 from Scambio )
```

Esercizio 2.10: Officina riparazioni auto

Il seguente schema relazionale è relativo a un database che registra le riparazioni effettuate da una catena di officine meccaniche.

OFFICINA (Nome, Indirizzo, Direttore)
RIPARAZIONE (Nome-Off, Num-Fattura, Targa, Tipo, Data, Importo)
AUTO (Targa, Proprietario)

2.10.1 *Elencare targa e data relative a tutte le riparazioni di auto il cui proprietario ha lo stesso nome del direttore dell'officina:*

```
select X.Targa, Y.Data
from AUTO as X, RIPARAZIONE as Y, OFFICINA
where X.Proprietario = Direttore and
      X.Targa = Y.Targa and
      Nome = Y.Nome-Off
```

2.10.2 *Elencare le officine che hanno svolto più di 50 riparazioni nel Febbraio 1993 ma non hanno mai riparato un'auto Fiat:*

```
( select Nome-Off
  from RIPARAZIONE
  where Data between 01/02/93 and 28/02/93
  group by Nome-Off
  having count(*) > 50 )

      except

( select Nome-Off
  from RIPARAZIONE
  where Tipo like '%Fiat%')
```

2.10.3 *Trovare l'officina che ha svolto più riparazioni di auto Fiat nel Febbraio 1993 (usando una vista):*

```
create view NUMFIAT (Nome, Num) as
  select Nome-Off, count(*)
  from RIPARAZIONE
  where Tipo = 'Fiat' and
        Data between 1/2/93 and 28/2/93
  group by Nome-Off

select X.Nome
from NUMFIAT as X,
where X.Num = ( select max(Num)
                from NUMFIAT )
```

Esercizio 2.11: Variante dell'officina

Dato il seguente schema relazionale, che descrive anche le informazioni dei clienti delle officine:

OFFICINA (ID, Sede, Direttore, Fatturato-92)
RIPARAZIONE (ID-Officina, ID-Cliente, ID-Ripar, Tipo_Riparazione,
Tipo_Auto, Data, Pagamento)
CLIENTE (ID-Cliente, Nome, Indirizzo)

2.11.1 *Determinare nome e indirizzo dei clienti che hanno pagato qualche riparazione relativa ad auto Fiat il cui importo (individuale) sia superiore a 3000 € in officine fuori Milano:*

Estraiamo i clienti che hanno delle riparazioni più costose di *3000 €*, eseguite presso officine con sede **non** in Milano.

```
select Nome, Indirizzo
from Cliente
where ID-Cliente in
     ( select ID-Cliente
       from Riparazione
       where Pagamento > 3000 and
             Tipo_auto like '%FIAT%' and
             ID-Officina not in
                      ( select Numero
                        from Officina
                        where Sede like '%MILANO%' ) )
```

2.11.2 *Trovare nome e indirizzo dei clienti che hanno pagato riparazioni nel 1992 per un ammontare superiore al 3% del fatturato dell'officina nel 1992:*

```
select Nome, Indirizzo
from Cliente
where ID-Cliente in
  ( select ID-Cliente
    from Riparazione R
    where Data between 1/1/1992 and 31/12/1992
    group by ID-Cliente, ID-Officina
    having sum(Pagamento) > all
            ( select Fatturato-92 * 0.03
              from Officina
              where Numero = R.ID-Officina ) )
```

Si noti che si utilizza **`> all`** sebbene l'insieme risultante contiene sempre un solo elemento (il fatturato dell'officina presso cui viene eseguita la riparazione).

2.11.3 *Determinare nome e indirizzo dei clienti che hanno pagato più di dieci riparazioni nel 1991 ma non hanno pagato alcuna riparazione nel 1992:*

```
select Nome, Indirizzo
from Cliente
where ID-Cliente in
     ( ( select ID-Cliente
         from Riparazione
         where Data between 1/1/1991 and 31/12/1991
         group by ID-Cliente
         having count(*) > 10)
             except
       ( select ID-Cliente
         from Riparazione
         where Data between 1/1/1992 and 31/12/1992 ) )
```

Esercizio 2.12: Gran premio automobilistico

Il seguente schema descrive i risultati di competizioni automobilistiche che si svolgono su più gare:

GRANPREMIO (Nazione, Anno, Data, Circuito)
PARTECIPAZIONE (Nazione, Anno, Pilota, Scuderia, PosInProva, PosInGara, Squalifica, Punti)
PILOTI (Nome, Nazione, DataNascita)

2.12.1 *Trovare le scuderie con due piloti in cui, le volte che i piloti hanno conquistato in gara posizioni consecutive, uno dei due piloti ha sempre preceduto l'altro:*

```
select distinct P1.Scuderia
from Partecipazione as P1, Partecipazione as P2
where P1.Scuderia = P2.Scuderia and
      P1.Nazione = P2.Nazione and P1.Anno = P2.Anno and
      P1.Posizione = P2.PosInGara + 1 and
      not exists ( select *
                   from Partecipazione P3, Partecipazione P4
                   where P3.Scuderia = P4.Scuderia and
                         P3.Nazione = P4.Nazione and
                         P3.Anno = P4.Anno and
                         P4.Posizione = P3.PosInGara + 1 and
                         P3 = P1 and
                         P4 = P2 )
```

2.11.2 *Creare una vista che restituisce per ogni nazione il pilota di quella nazione che ha conquistato il maggior numero di punti, ordinati in modo decrescente rispetto ai punti:*

```
create view Classifica(Nazione, Nome, Punti) as
select Piloti.Nazione, Piloti.Nome, sum(Piloti.Punti)
from Piloti inner join Partecipazione
            on Piloti.Nome = Partecipazione.Pilota
group by Piloti.Nome

create view Migliori(Nazione, Nome) as
select Nazione, Nome
from Classifica as X
where Punti = ( select max( Punti )
                from Classifica
                where X.Nazione = Nazione )
order by Punti desc
```

Capitolo Terzo
Progettazione concettuale

Questo capitolo affronta il tema della progettazione di basi di dati. In particolare, si occupa della modellazione concettuale secondo il modello entità relazione e della traduzione di tale modello nel modello logico.

Esercizio 3.1: Servizio autobus
Si ha una società che gestisce un servizio di trasporto con autobus. L'azienda possiede un certo numero di autobus, di diversi modelli, che vengono utilizzati su diversi percorsi (un autobus può eseguire corse su percorsi diversi, così come gli autisti e i controllori possono utilizzare diversi autobus e diversi percorsi). Il personale dell'azienda è costituito da autisti e controllori (che svolgono servizio su determinate corse), addetti alla vendita (che risiedono in alcune fermate) e addetti alla amministrazione ed alla manutenzione che lavorano alla sede centrale. Di ogni dipendente bisogna memorizzare le solite informazioni anagrafiche. Di ogni percorso si devono memorizzare il luogo di partenza, di arrivo e le fermate intermedie. I mezzi sono conservati durante la notte in depositi, ogni autobus è associato a un particolare deposito. Ogni autista è associato a un deposito, e la prima e l'ultima corsa della giornata verranno effettuate sempre su un autobus del deposito a cui è assegnato, ma le altre corse possono essere svolte anche su mezzi facenti capo ad altri depositi.

Soluzione – Modello concettuale (Entità-Relazione)
Il modello *Entità-Relazione* (*Entity-Relationship*, o *ER*) è un *modello concettuale* per la progettazione di basi di dati basato sui concetti di entità e relazione. Un'*entità (entity)* è un'astrazione che descrive una categoria di oggetti del mondo reale, mentre una *relazione (relationship)* è un'associazione tra due o più entità. Le entità sono descritte attraverso *attributi* (proprietà degli oggetti), mentre le relazioni possono essere dettagliate grazie al concetto di *cardinalità*, che specifica, per ogni istanza di un oggetto, il numero minimo e massimo di istanze che possono essere associate a quell'oggetto. Il modello ER ammette anche *gerarchie di generalizzazione,* che rappresentano un legame gerarchico tra un'entità padre e una o più entità figlie, che sono casi particolari del padre. Le entità figlie ereditano tutte le proprietà del padre, e le loro istanze sono anche istanze dell'entità padre.

La progettazione di modelli concettuali è supportata da diverse metodologie, che propongono differenti processi di sviluppo: top-down, bottom-up, a macchia d'olio, ecc. Nella soluzione di esercizi di dimensioni medio-piccole, gli approcci top-down e a macchia d'olio sono i più usati. Tipicamente si procede con un'attenta lettura del testo, sottolineando i concetti fondamentali e sfruttandoli come elementi base del modello. Per questo primo esempio si riporta lo sviluppo passo-passo di una possibile soluzione, seguendo l'approccio top-down.

La prima fase della progettazione consiste nell'identificare le entità e le relazioni principali. Le relazioni potranno essere raffinate successivamente.

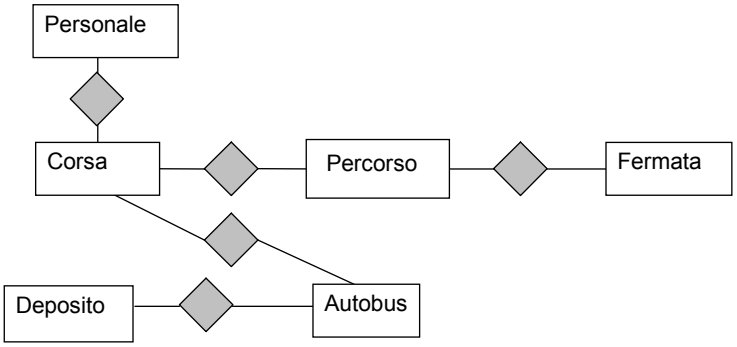


Figura 3.1 – Modello ER fase 1

Successivamente, si tracciano relazioni più precise, includendo anche i nomi, e le gerarchie di generalizzazione più importanti.

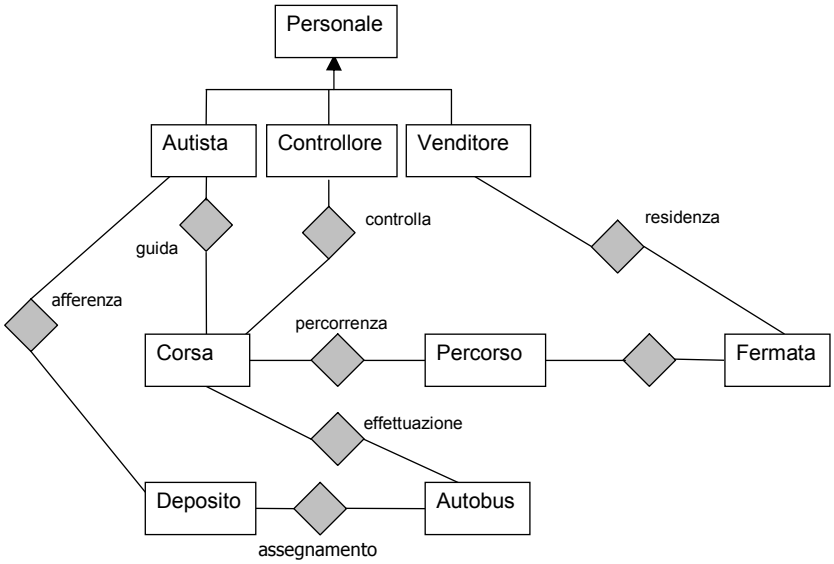


Figura 3.2 – Modello ER fase 2

In una terza fase si rappresentano i dettagli di tutte le gerarchie e le relazioni, indicandone anche i nomi, e si specificano gli identificatori delle entità.

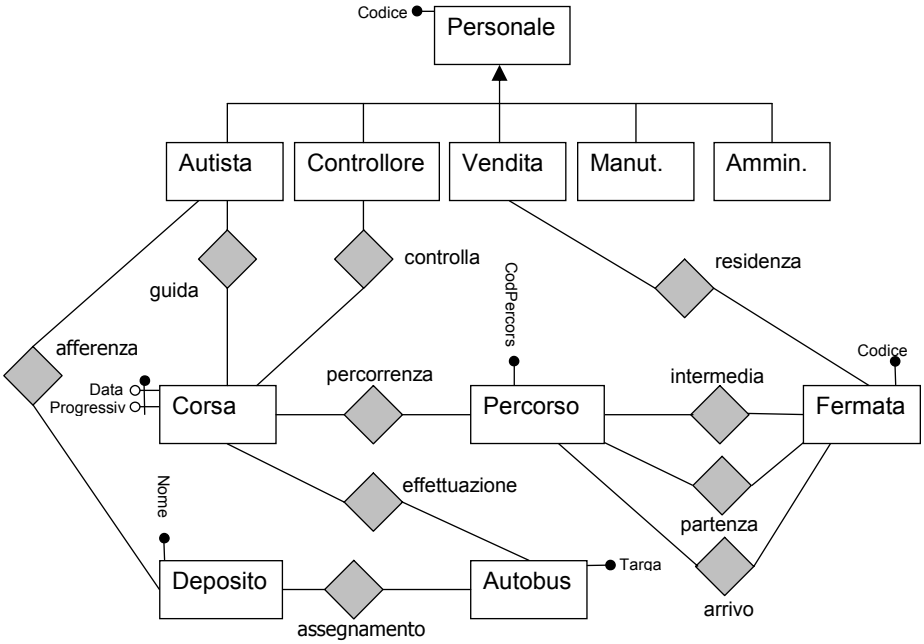


Figura 3.3 – Modello ER fase 3

Infine, si completa lo schema con tutti gli attributi, indicandone nome e cardinalità, e con tutti i dettagli delle relazioni (cardinalità minima e massima) e delle gerarchie (indicandone la totalità e l'esclusività). Il risultato finale è rappresentato in Fig. 3.4.

Bisogna osservare che, anche partendo da testi brevi come quelli trattati qui, il risultato ottenuto non è praticamente mai univoco, in quanto le specifiche sono soggette all'interpretazione del progettista, che le raffina e le dettaglia dove carenti.

Nel nostro esempio, nel progettare il concetto di itinerario si è fatta l'ipotesi che un percorso non preveda intersezioni, cioè l'autobus si fermi alla stessa fermata solo una volta in un dato percorso. Inoltre, le relazioni partenza e arrivo dell'entità Percorso sono ridondanti, in quanto è possibile risalirvi esaminando l'attributo della relazione intermedia. Si è scartata tale soluzione in quanto si ipotizza che le fermate di inizio e fine di un percorso siano proprietà che entrano a far parte di una interrogazione con una certa frequenza. Si noti che l'identificatore CodPercorso permette di distinguere percorsi con gli stessi estremi ma itinerari differenti (ad esempio Menaggio-Como via panoramica è diverso da Menaggio-Como via lago).

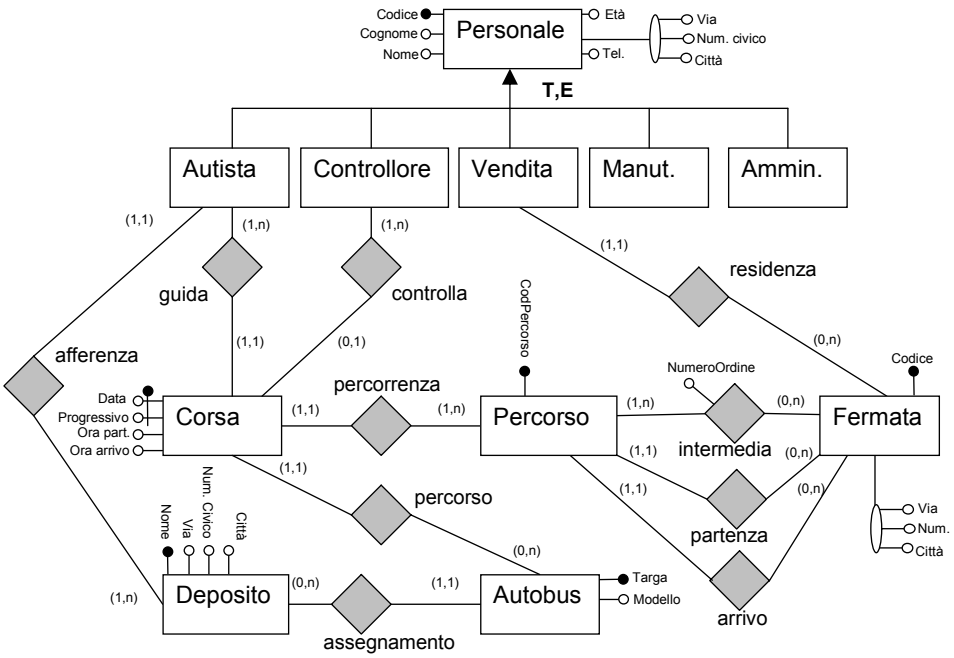


Figura 3.4 – Modello ER completo

Soluzione – Modello logico (Relazionale)

Il modello *Relazionale* è un modello che definisce a *livello logico* la struttura di un database. Il modello relazionale si basa sul concetto di *relazione* (*relation*), nel senso matematico del termine: una relazione è un sottoinsieme del prodotto cartesiano di n insiemi. Il concetto di relazione corrisponde sostanzialmente ad una tabella di database. Una relazione è composta da n attributi. Gli oggetti (o istanze) memorizzati in una relazione prendono il nome di *tuple* o *n-uple*. Il passaggio da modello concettuale a modello logico è guidato da regole che prevedono la ristrutturazione del modello ER e la sua traduzione nel modello relazionale.

La realizzazione del modello logico prevede due passi fondamentali: la ristrutturazione dello schema ER e la traduzione nel modello relazionale.

La ristrutturazione dello schema ER include i seguenti passi:

- *Eliminazione di ridondanze* non necessarie;
- *Partizionamento o accorpamento* di concetti: i casi più tipici sono quello di trasformazione di attributi strutturati in attributi semplici o in entità indipendenti, oppure quello di trasformazione di attributi multivalore in entità a sé stanti, associate all'entità principale. In particolare, gli attributi composti da più sottoattributi possono essere trasformati per aggregazione di tutti i

sottoattributi in un unico attributo semplice generale, che contiene tutte le informazioni in modo indiscriminato, oppure separazione dei sottoattributi in singoli attributi semplici. Per gli attributi multivalore la soluzione migliore è in genere quella di realizzare un'entità debole corrispondente al singolo attributo, collegata attraverso un'associazione all'entità principale.

- *Eliminazione delle gerarchie* di generalizzazione, inglobando le entità figlie all'interno del padre (risoluzione o collasso verso l'alto, che prevede la scomparsa delle entità "figlie"), spezzandole in entità indipendenti ed eliminando il padre (risoluzione o collasso verso il basso, che prevede la scomparsa dell'entità "padre"), o infine trasformandole in associazioni uno-a-uno che traducono il rapporto di ereditarietà (risoluzione "ibrida").
- *Scelta della chiave primaria*, per le entità che avessero più di un identificatore. Questo passo consiste nella scelta come chiave primaria di uno e un solo identificatore per ogni entità. In questo passaggio è anche necessario trasformare tutti gli identificatori esterni presenti delle entità deboli in identificatori semplici.

A valle della ristrutturazione dello schema ER si effettua la traduzione di *entità* e *relazioni* (o associazioni, relationship) del modello ER in *relazioni* (relation) del modello relazionale. Per evitare ambiguità dovute alla lingua italiana, d'ora innanzi chiameremo *associazioni* le relationship del modello ER e *relazioni* le relation del modello relazionale. Per la traduzione si possono applicare le seguenti regole:

- Ogni entità viene tradotta in una relazione (tabella), avente come attributi gli attributi di quell'entità e come chiave l'identificatore primario.
- Ogni associazione molti a molti (cioè con cardinalità massima pari a N in entrambe le direzioni) viene tradotta in una relazione (tabella) avente come attributi gli attributi dell'associazione più le chiavi delle entità associate.
- Ogni associazione uno a molti (cioè con cardinalità massima pari a 1 in una delle due direzioni) può essere tradotta inserendo un attributo aggiuntivo sulla entità da cui la cardinalità massima è 1. Tale attributo conterrà la chiave dell'oggetto associato all'oggetto corrente. Si introduce così una chiave esterna (foreign key) e, con essa, un vincolo di integrità referenziale.
- Ogni associazione uno a uno (cioè con cardinalità massima pari a 1 in entrambe le direzioni) può essere tradotta inserendo un attributo aggiuntivo su una qualunque delle entità collegate; tale attributo diviene una foreign key.

Poiché la realizzazione del modello logico è un'attività in gran parte meccanica, nel seguito esso verrà presentato solo per un numero limitato di esercizi, lasciando al lettore la modellazione logica degli altri casi.

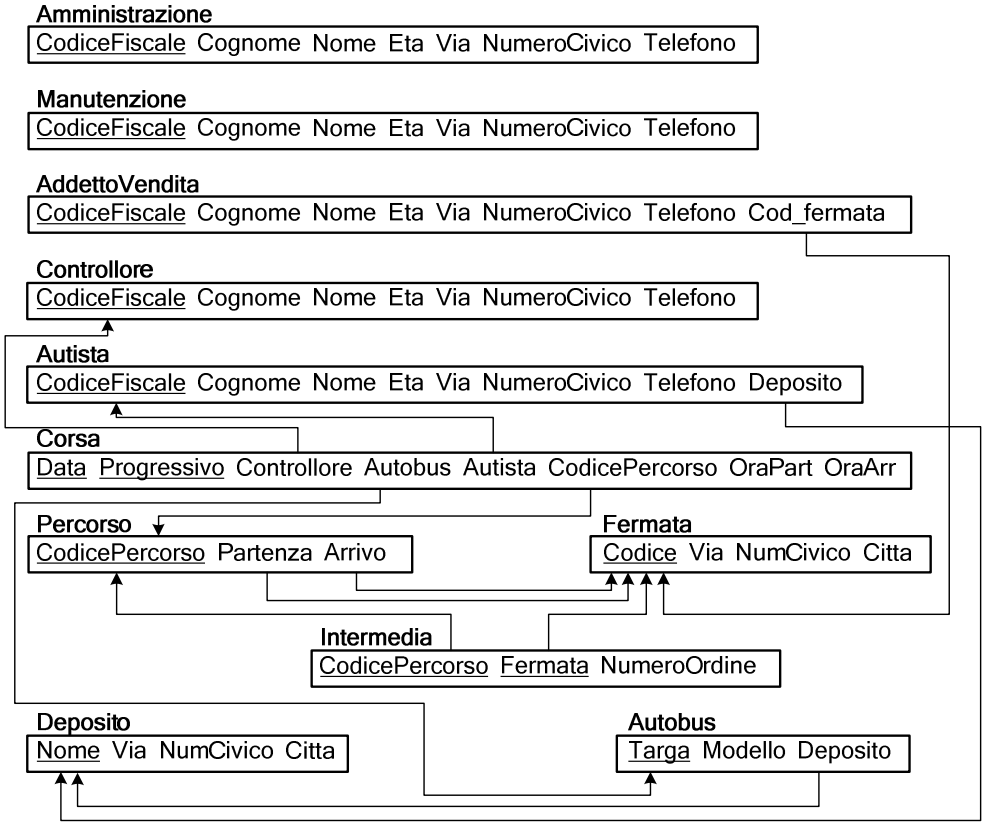


Figura 3.5 – Modello relazionale (o logico)

Il modello logico è a volte rappresentato in forma testuale, attraverso notazioni simili alla seguente. Anche in questo caso è possibile rappresentare i cammini di join tra le relazioni, qui non riportati per preservare la leggibilità del testo. In generale, raccomandaiamo la rappresentazione a diagramma, in cui i cammini di join sono più leggibili e che può essere più facilmente messa in corrispondenza con la topologia dello schema E-R da cui è derivato il progetto logico.

```
Amministrazione(CodiceFiscale, Cognome, Nome, ...)
Manutenzione(CodiceFiscale, Cognome, Nome, ...)
AddettoVendita(CodiceFiscale, Cognome, ..., Cod_fermata)
Controllore(CodiceFiscale, Cognome, Nome, ...)
Autista(CodiceFiscale, Cognome, Nome, ..., Deposito)
Corsa(Data, Progressivo, Autista, Autobus, Controllore,
      CodicePercorso, Ora_part, Ora_arr)
Autobus(Targa, Modello, Deposito)
```

```
Percorso(CodicePercorso, Partenza, Arrivo)
Intermedia(CodicePercorso, Fermata, NumeroOrdine)
Fermata(Codice, Via, Numero civico, Citta)
Depositi(Nome, Via, Numero civico, Citta)
```

La gerarchia di generalizzazione totale ed esclusiva su Personale è stata collassata verso il basso, poiché esistono diverse associazioni che interessano le singole sotto-entità. Si ipotizza, infatti, che le interrogazioni siano più strettamente legate alla mansione svolta dai singoli dipendenti che ai dati anagrafici generali.

Esercizio 3.2: Ricerca paleontologica
Si vuole realizzare una base di dati per la comunità scientifica di ricerca paleontologica. Si devono memorizzare i dati riguardanti i reperti fossili di vertebrati custoditi dai musei. I reperti sono caratterizzati dal luogo e dall'anno di ritrovamento, dal ricercatore responsabile della scoperta, dal museo e dalla sala in cui sono custoditi e dalla specie presunta di appartenenza. Ogni reperto può essere attribuito a diverse specie, con diverso grado di probabilità. Ad ogni specie possono essere associati più nomi, qualora diversi ricercatori abbiano fornito lo stesso nome a specie diverse: in tal caso il nome ufficiale è il nome più vecchio. I musei sono caratterizzati dalle sale, dai loro ricercatori e dal loro direttore (che può essere un paleontologo o un ricercatore di altra materia).

Soluzione

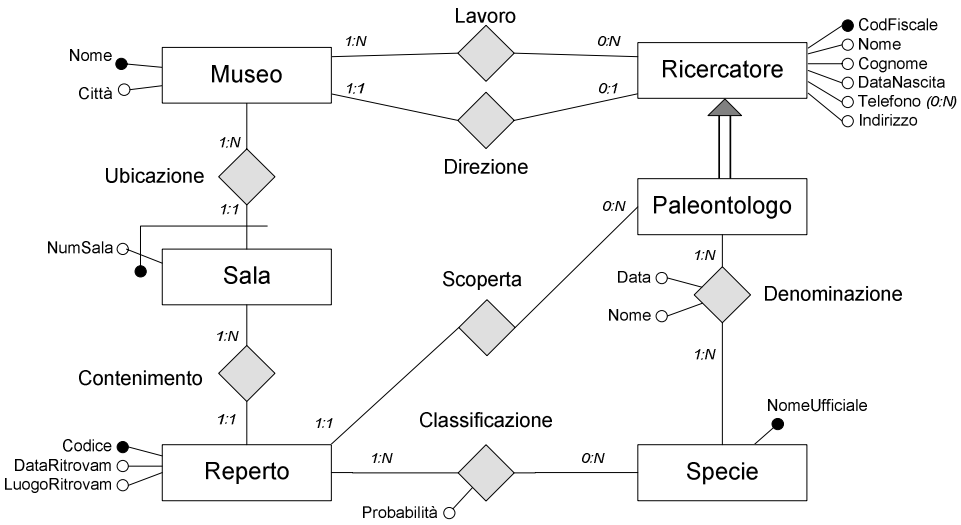


Figura 3.6 – Modello ER

La traduzione verso il modello logico avviene secondo il processo standard descritto nel precedente esercizio. La gerarchia di generalizzazione Ricercatore-

Paleontologo è risolta verso l'alto, introducendo in Ricercatore l'attributo IsPaleo di tipo boolean: se questo assume valore *true*, il ricercatore sarà un Paleontologo. In questo modo si perde, però, il vincolo secondo il quale solo un paleontologo può scoprire reperti, che dovrà essere espresso nella base dati con un'opportuna asserzione o clausola check che coinvolga l'attributo IsPaleo.

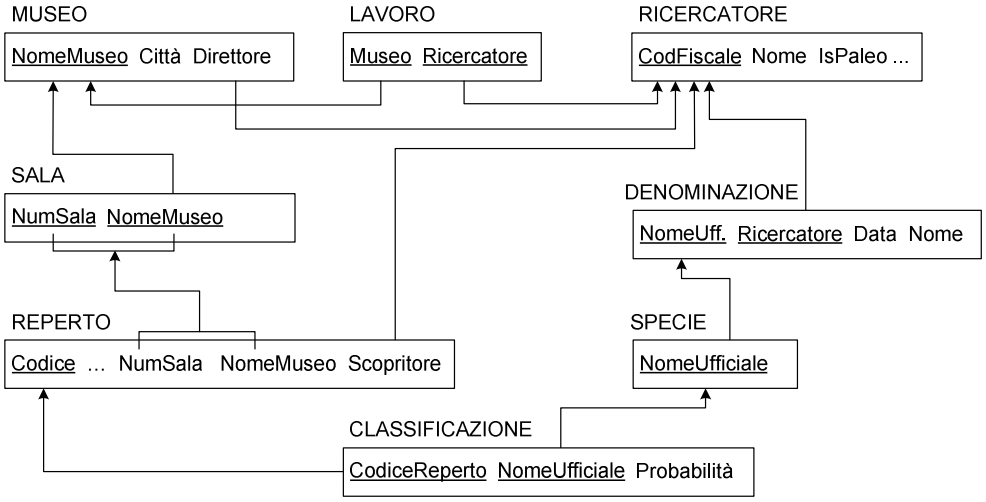


Figura 3.7 – Modello logico

Esercizio 3.3: Agenzia viaggi
Una agenzia di viaggi gestisce informazioni relative ai propri clienti e ai loro viaggi. Per ciascun cliente è noto se si tratta di una impresa (caratterizzata da una ragione sociale ed una partita iva) o un privato; in ogni caso, sono noti indirizzo (via, città, cap), vari recapiti telefonici e un identificativo del cliente, assegnato dall'agenzia. Ciascun viaggio è associato ad un cliente (anche se può essere svolto da più persone) ed è caratterizzato da un identificativo, un costo ed una durata complessiva. Il costo comprende anche diritti di agenzia, oltre al costo dei vari servizi che compongono ciascun viaggio. Tali servizi sono:

- I tragitti, da un luogo ed ora di partenza ad un luogo ed ora di arrivo, che possono svolgersi in treno, bus, aereo oppure nave. Ciascun tragitto ha un numero di partecipanti, un costo individuale ed una classe, supposta identica per tutti i partecipanti. Per i viaggi in aereo è nota la compagnia aerea. Per i viaggi notturni in treno oppure in nave possono essere utilizzate varie cuccette (che possono essere singole, doppie o triple).
- I pernottamenti, che si svolgono in notti consecutive dall'inizio alla fine del viaggio. Per ciascuna notte di pernottamento è noto un costo complessivo. Nel caso il pernottamento si svolga in un hotel, è noto il numero ed il tipo (singola, doppia, tripla) di stanze e il trattamento (solo pernottamento, mezza pensione,

pensione completa), che si suppone identico per tutti i partecipanti. Nel caso il pernottamento si svolga in abitazioni, è noto il loro numero complessivo di posti letto. In entrambi i casi, è noto l'indirizzo e il numero telefonico. Infine, taluni pernottamenti si svolgono durante l'effettuazione di un tragitto; in tal caso, il loro costo è nullo in quanto già coperto dal costo del tragitto.

- Gli spettacoli, caratterizzati da un titolo, un luogo, una data, un costo individuale e un numero di partecipanti.

Soluzione

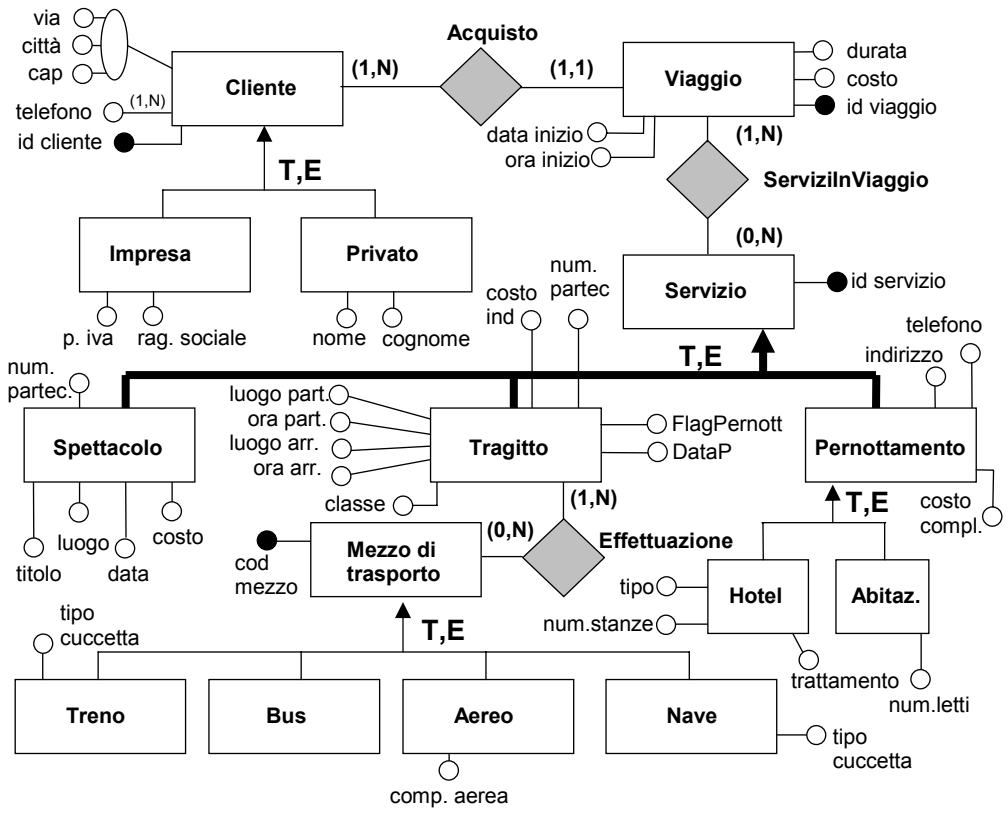


Figura 3.8 – Modello ER

Si noti che alcuni vincoli di integrità che non sono espressi dallo schema sono:

- Il numero di partecipanti, attributo delle entità spettacolo e tragitto, deve essere minore o uguale al numero di partecipanti dell'entità viaggio;
- Il costo individuale di ciascun servizio deve essere inferiore al costo totale del viaggio.

Esercizio 3.4: Laboratori didattici
Si deve progettare una base di dati relativa alla gestione delle prenotazioni dei posti di un laboratorio didattico di una università. Ogni studente è caratterizzato dalla propria matricola, nome, cognome, data e luogo di nascita, residenza, recapito telefonico. Gli studenti frequentano alcuni laboratori didattici. I laboratori didattici contengono un insieme di posti di lavoro e un insieme di risorse. Ad ogni posto di lavoro sono assegnate alcune risorse (unità di calcolo, stampanti, applicazioni). Alcune delle risorse sono rese disponibili a tutti gli studenti senza controlli, altre vengono assegnate agli studenti che frequentano determinati laboratori, previa autorizzazione. Lo studente può utilizzare un posto di lavoro solo se effettua una prenotazione. Si deve tenere traccia di tutte le prenotazioni e di tutte le volte che lo studente utilizza un posto di lavoro. Ogni laboratorio ha un solo responsabile, il quale si può occupare di un solo laboratorio.

Soluzione ERRATA

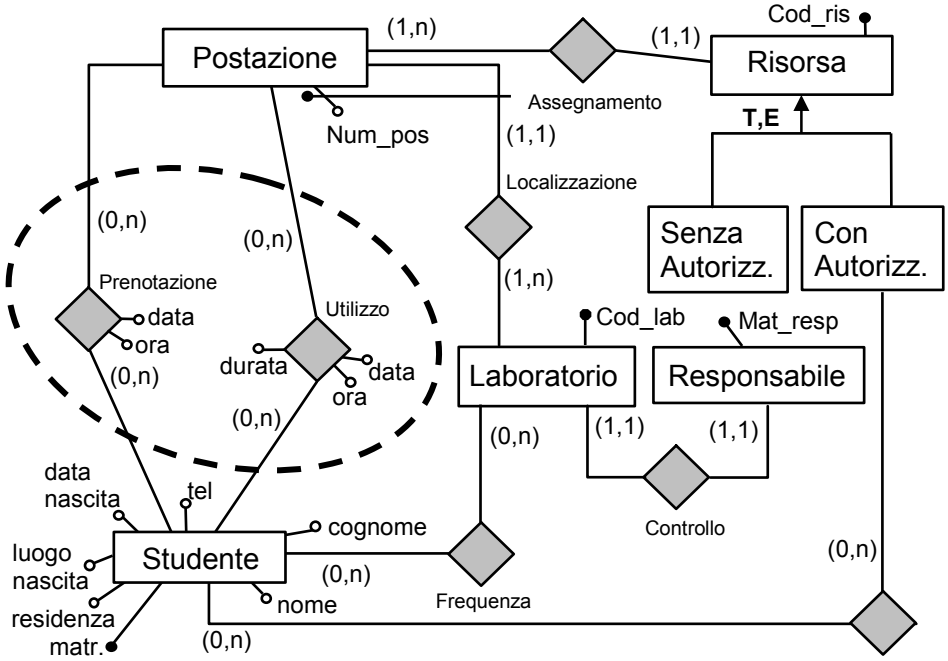


Figura 3.9 – Modello ER con un tipico errore

La soluzione soprastante è errata in virtù dell'interpretazione delle associazioni Prenotazione e Utilizzo come relazioni algebriche. Infatti, ricordiamo che Prenotazione $\subseteq$ Postazione $\times$ Studente, e Utilizzo $\subseteq$ Postazione $\times$ Studente. Prenotazione e Utilizzo sono quindi due *insiemi di coppie*, e, in quanto insiemi, non ammettono duplicati. Non è ammesso dal modello, cioè, che per una stessa coppia

$\langle p, s \rangle \in$ Postazione × Studente sussista più di un'istanza di ogni associazione, indipendentemente dai valori degli (eventuali) attributi specificati sull'associazione stessa (per le due associazioni che stiamo considerando, data, ora, e durata).

Nel nostro caso, l'effetto di questa restrizione è evidente: uno studente potrebbe prenotare solo una volta ogni postazione (non è ammesso, infatti, distinguere le diverse istanze di associazione in base a data e ora, perché sarebbero comunque due associazioni tra gli *stessi* elementi $\langle p, s \rangle$). Del resto, non è accettabile introdurre surrettiziamente il vincolo per cui gli studenti, una volta "consumata" una data postazione, non potranno più utilizzarla in seguito, per quanto detto chiaramente nel testo della specifica ("*...di tutte le volte che lo studente utilizza un posto di lavoro...*").

Per rappresentare correttamente la possibilità di tornare a occupare lo stesso posto dobbiamo ristrutturare lo schema, introducendo due entità deboli di supporto:

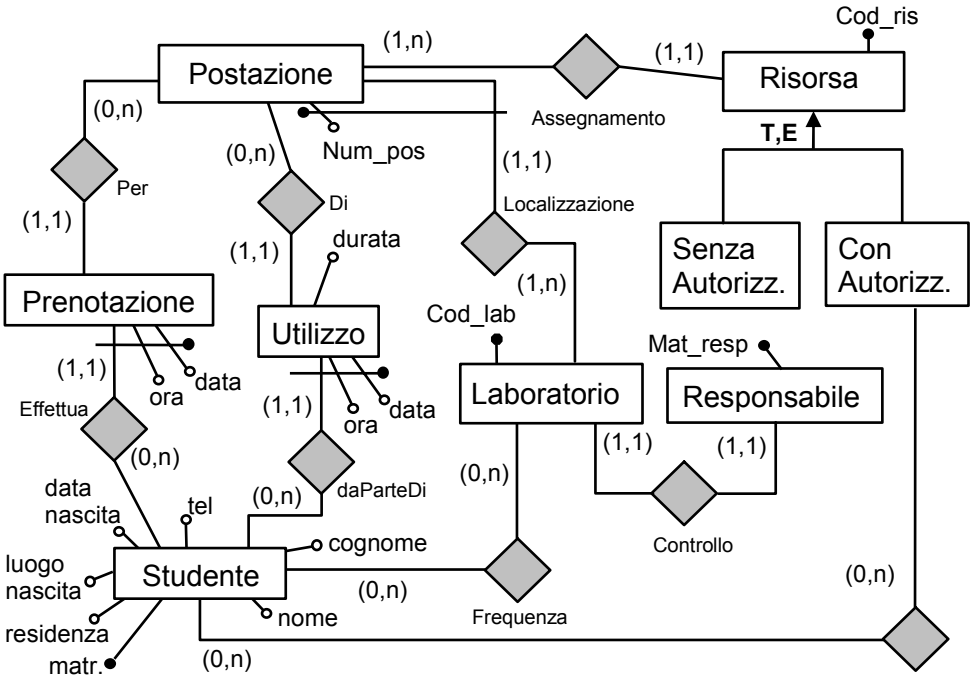


Figura 3.10 – Modello ER corretto

Segue il modello logico dei dati. Si noti che la data e l'ora compaiono nelle chiavi delle tabelle **Prenotazione** e **Utilizzo**. In alternativa, l'identificazione debole sarebbe potuta avvenire verso la postazione invece che verso lo studente (e sarebbe stato forse anche più intuitivo). Si noti, però, che in tal caso per costruire la chiave delle due tabelle derivate avremmo dovuto risalire fino al laboratorio (ci sono due entità deboli "in cascata"), e sarebbe stata una soluzione assai inefficiente (tipicamente, in questi casi, si introduce poi un id numerico nel DB fisico).

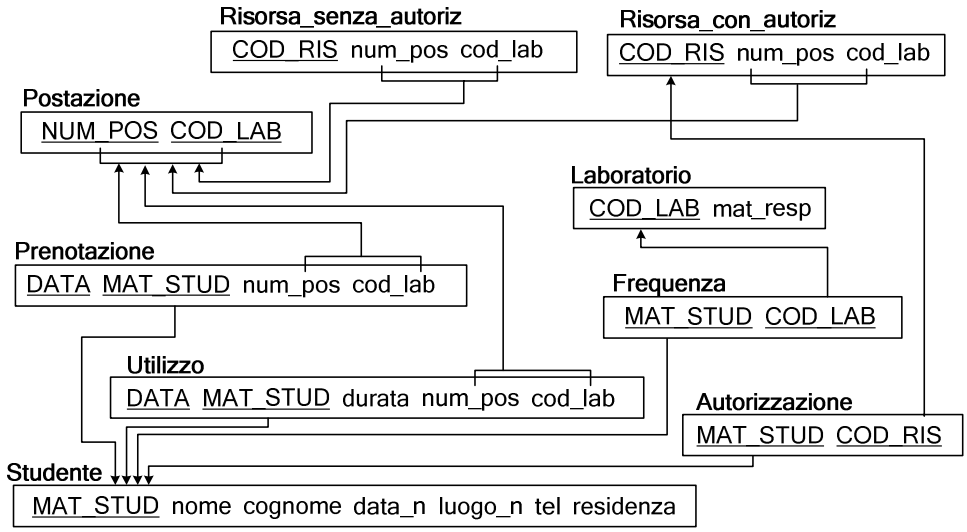


Figura 3.11 – Modello logico

Esercizio 3.5: Noleggio furgoni
Una piccola società di noleggio di furgoni vuole automatizzare la gestione del noleggio. I furgoni, caratterizzati da un tipo (in genere, una coppia: modello, marca del costruttore), una capienza (in termini sia di posti a sedere, sia di metri cubi disponibili nel vano posteriore), una data di immatricolazione e una targa, vengono noleggiati a clienti, caratterizzati da nome, indirizzo, ragione sociale (nel caso di imprese), una partita IVA (obbligatoria per le imprese) e un numero di carta di credito (obbligatorio per i privati). Alcuni clienti convenzionati godono di uno sconto speciale sulle tariffe di noleggio. I dati del noleggio prevedono: data e ora di inizio, data e ora concordata di fine, costo globale pattuito per il noleggio, eventuale copertura assicurativa aggiuntiva, eventuale estensione del noleggio pattuita col cliente durante il noleggio stesso, costo di ogni settimana, giornata, ed ora aggiuntiva (per un massimo di tre ore di ritardo sulla consegna); questi ultimi tre dati dipendono esclusivamente dal tipo di furgone noleggiato. I dati relativi al noleggio vengono rilevati all'atto della prenotazione; possono cioè far riferimento a noleggi che avverranno nel futuro, oppure ad un noleggio in corso, oppure infine ad un noleggio passato; i dati vengono cancellati dopo due mesi dalla data di termine del noleggio. Inoltre, assieme a ciascun cliente vengono registrate le eventuali multe ricevute durante uno specifico noleggio, con indicazione del luogo, dell'importo e della causale; sono evidentemente possibili varie multe in relazione ad uno specifico noleggio. Nel progettare la base di dati, è opportuno tenere presente che deve essere possibile interrogare la base di dati per ricavare l'elenco dei furgoni disponibili ad una certa data ed il loro costo giornaliero e settimanale.

Soluzione

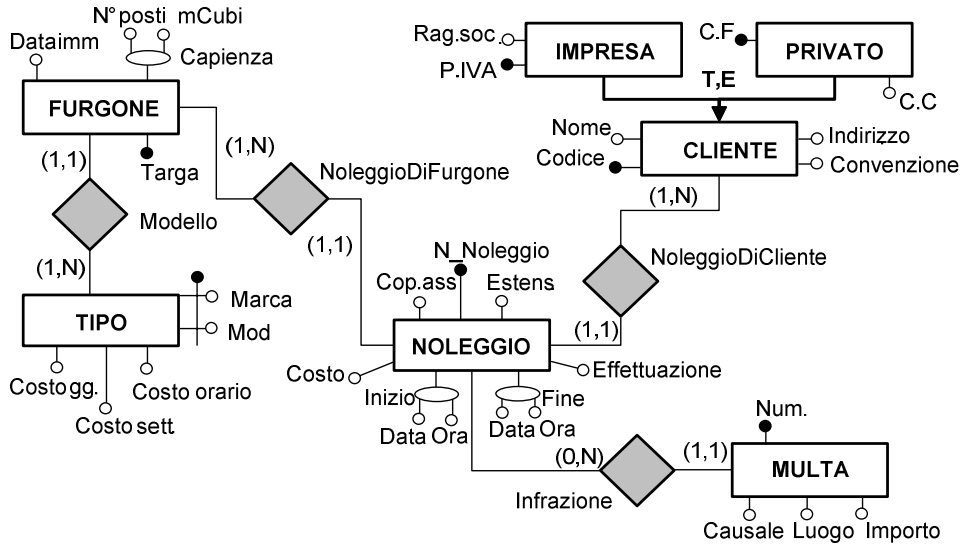


Figura 3.12 – Modello ER

Si noti che diversi vincoli non possono essere espressi nel modello: la quantità di ore massima di estensione, il fatto che un furgone non può essere associato a più noleggi contemporanei, ecc.
Riportiamo di seguito lo schema logico corrispondente. Si noti che la gerarchia di generalizzazione è stata collassata verso l'alto, in quanto si è ritenuta prioritaria l'associazione verso noleggio per tutti i clienti.

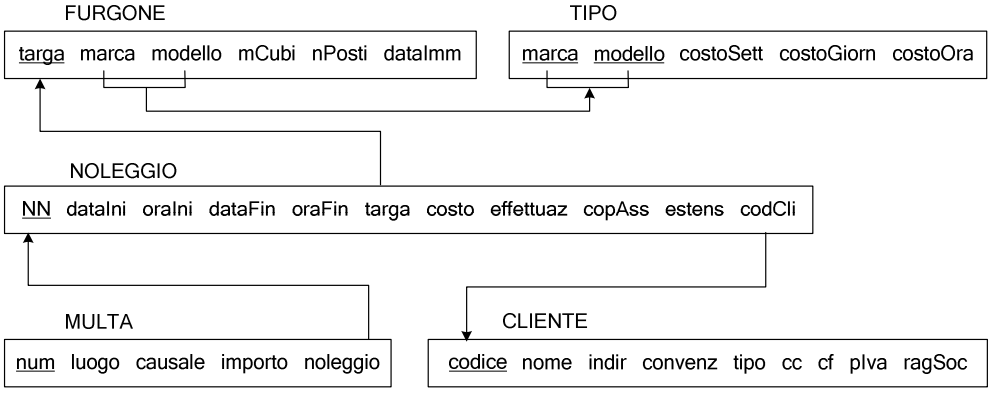


Figura 3.13 – Modello logico

Esercizio 3.6: Applicazioni software

Una ditta produttrice di applicazioni software deve realizzare una base di dati relativa al personale, alle applicazioni prodotte e ai clienti. Di ogni persona sono noti i dati anagrafici e la residenza. Il personale è diviso in programmatori, amministratori, venditori e dirigenti. Alcuni programmatori lavorano allo sviluppo delle applicazioni, mentre altri forniscono un supporto ai clienti; vi sono anche programmatori che rivestono entrambe le funzioni. Le applicazioni si dividono in applicazioni finite per cui esiste un prezzo di listino, applicazioni in stadio 2 (che vengono distribuite ad alcuni dei clienti in modo tale che vengano scoperti gli errori), applicazioni in stadio 1 (che sono nella prima fase dello sviluppo e che non vengono distribuite all'esterno. I clienti possono essere persone fisiche (caratterizzate dalle informazioni anagrafiche e dal codice fiscale) o ditte (caratterizzate dalla ragione sociale, dalla sede e dal codice fiscale). I clienti possono avere un contratto di manutenzione, per cui viene loro assegnato un particolare programmatore che fornisce assistenza quando sorgono problemi, oppure possono essere privi del contratto ed in tale caso ogni richiesta di intervento viene fatturata a parte e gestita da uno qualsiasi dei programmatori disponibili.

Soluzione

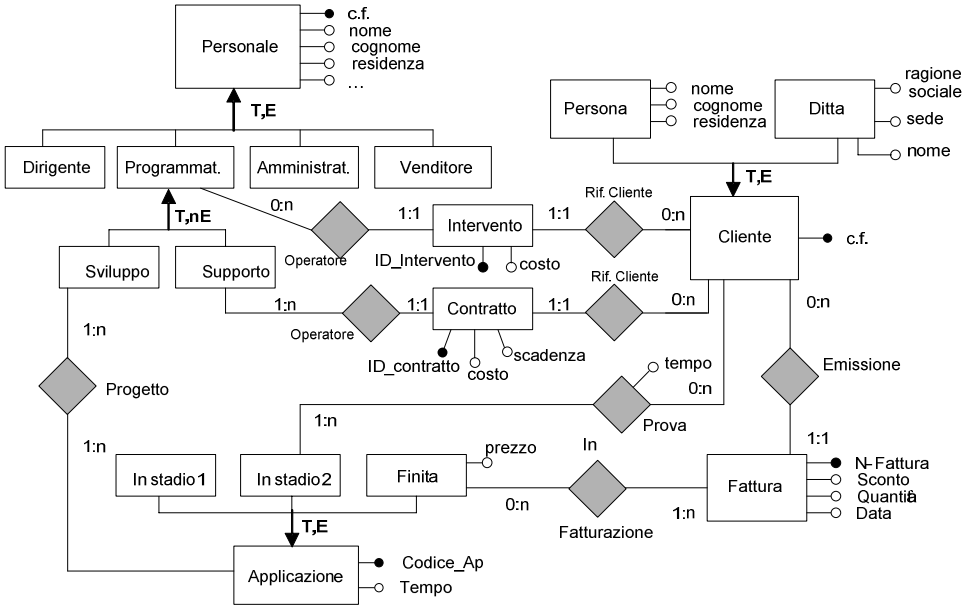


Figura 3.14 – Modello ER

Esercizio 3.7: Campionato di calcio
Si vuole realizzare una base di dati che descriva le informazioni relative al campionato di calcio. Per ogni partita, descrivere il girone e la giornata in cui si è svolta, il numero progressivo nella giornata (es. prima partita, seconda partita, ecc.), la data con giorno, mese, anno, le squadre coinvolte nella partita con nome, città della squadra e allenatore, ed infine per ciascuna squadra se ha giocato in casa. Si vogliono conoscere i giocatori che giocano in ogni squadra con la loro data di nascita e il loro ruolo principale. Si vuole conoscere per ogni giornata, quanti punti ha ogni squadra. Si vogliono anche conoscere, per ogni giornata, i giocatori di ogni squadra che hanno giocato ed il ruolo che ha giocato ogni giocatore (i ruoli, perciò, possono cambiare di partita in partita). Per ogni partita, si vuole rappresentare l'arbitro, con nome, cognome, città e regione di nascita. Distinguere le partite giocate regolarmente da quelle rinviate. Per quella rinviate, rappresentare la data in cui si sono effettivamente giocate. Distinguere anche le partite giocate in una città diversa da quella della squadra ospitante; per questo si vuole rappresentare la città in cui si svolgono, nonché il motivo della variazione di sede. Dei giocatori interessa anche la città e la regione di nascita.

Soluzione

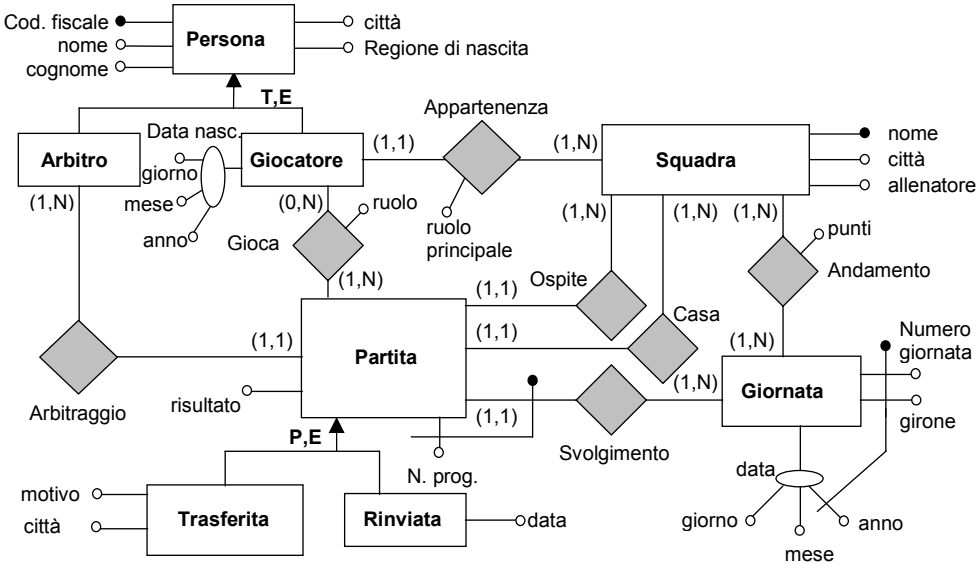


Figura 3.15 – Modello ER

Esercizio 3.8: Festival cinematografico

Si deve progettare una base di dati per il supporto all'organizzazione di un festival cinematografico. L'organizzazione impiega del personale a tempo indeterminato, coadiuvato da personale assunto a tempo determinato e da volontari. Il Festival fornisce dei tesserini alle persone accreditate, ovvero le persone che partecipano al festival per motivi professionali (attori, registi, critici, membri della giuria). Bisogna tener traccia di chi ha autorizzato l'emissione del tesserino. Alcuni membri del personale si devono occupare di assistere i membri della giuria, organizzando i pernottamenti e i trasporti tra l'aeroporto e la sede del festival.

Il Festival è strutturato in rassegne, una destinata ai film in concorso e le altre su temi diversi. Ogni film viene presentato nell'ambito di una sola rassegna. Di ogni film vengono organizzate alcune proiezioni, di cui si conosce la sala, il giorno, l'ora e il ricavo ottenuto dalla vendita dei biglietti. Almeno una proiezione dei film in concorso è dedicata ai membri della giuria.

Si deve tenere traccia di quali sono i film che i registi e gli attori accreditati sono venuti a presentare, e se gli attori hanno o meno un ruolo da protagonista nel film. Ogni giorno la giuria si riunisce in una sala. Al termine del Festival, la giuria assegna un certo numero di premi (che possono anche essere a pari merito): miglior film, secondo miglior film, miglior attore e attrice (protagonista e non protagonista).

Soluzione

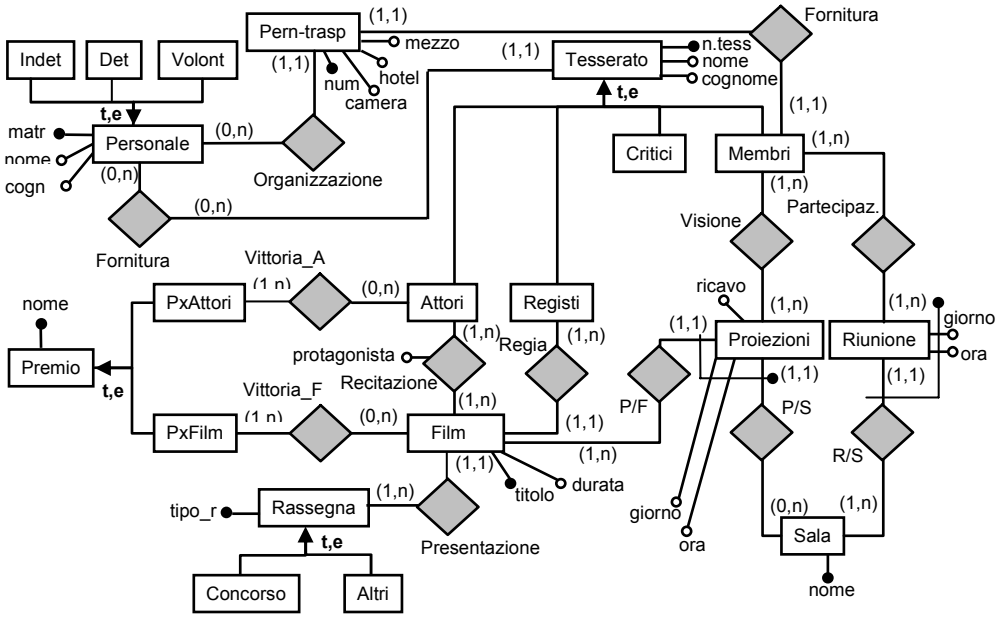


Figura 3.16 – Modello ER

Capitolo Quarto
Temi d'esame risolti

Questo capitolo contiene dieci temi d'esame assegnati nel corso di Basi di Dati al Politecnico di Milano negli anni accademici 03/04, 04/05 e 05/06.

Attenzione: le soluzioni proposte riflettono sempre ***una interpretazione*** delle specifiche, che non pretende di essere l'unica corretta rispetto alla semantica del linguaggio naturale (che è sempre inevitabilmente ambigua, in qualche misura). Ciò è particolarmente vero nel caso della progettazione concettuale. Per quanto riguarda invece le interrogazioni, in alcuni casi in cui sono possibili più interpretazioni significativamente diverse proponiamo le soluzioni corrispondenti.

I testi dei temi d'esame richiedono che la soluzione in algebra relazionale sia ottimizzata. Per motivi di spazio, e per non appesantire la soluzione con parti che ne compromettano la leggibilità, abbiamo preferito tralasciare in questa sezione l'applicazione sistematica delle trasformazioni di equivalenza, privilegiando piuttosto formulazioni che aiutino a osservare l'analogia con la formulazione negli altri linguaggi. Del resto, l'ottimizzazione è in gran parte un processo meccanico.

Per le stesse ragioni di brevità proponiamo solo gli schemi ER come soluzione alle parti di progettazione concettuale, lasciando la progettazione logica per esercizio. I commenti agli schemi concettuali sono limitati al minimo, perché è bene abituarsi a produrre schemi autoesplicativi. A rigore, uno schema che necessita di molti commenti e di molte precisazioni denuncia da sé la sua scarsa qualità.

Allievi INF e TLC - Proff. S. Ceri e M. Matera
Prove in Itinere – Novembre 2003 / Gennaio 2004

UTENTE (IdUsr, Nome, Password, Categoria)
PAGINA (URL, Titolo, DimInByte, Tipo, Proprietario)
ACCESSO (IdUsr, Data, Ora, UrlPag, PaginaProvenienza)
COLLEGAMENTO (PagFrom, PagTo)

Lo schema soprastante si riferisce agli accessi alle pagine di un sito Web da parte di utenti registrati. Memorizza inoltre le informazioni relative al collegamento tra le pagine (attraverso i link) e per ogni accesso registra la pagina di provenienza. Ogni pagina ha un utente – rappresentato tramite il suo identificatore – come "proprietario". Per la pagina iniziale di ogni navigazione, la pagina di provenienza è posta a **Null**.

A. DDL (2 punti)

Scrivere i comandi SQL per creare le tabelle PAGINA e COLLEGAMENTO, effettuando opportune e ragionevoli ipotesi sui domini, sui vincoli e sulle reazioni ai cambiamenti.

```
create table PAGINA (
  URL             varchar(255) primary key,
  Titolo          varchar(255),
  DimInByte       integer > 0,
  Tipo            varchar(20) default 'personal',
  Proprietario    varchar(30) references UTENTE(IdUsr)
                       on update cascade on delete set null )
```

Il valore null assegnato al proprietario a seguito della cancellazione di un utente permette di identificare le pagine che nessuno più mantiene (o di cui comunque non si sa chi sia il proprietario).

```
create table COLLEGAMENTO (
  PagFrom   varchar(255) references PAGINA(URL)
                on update cascade on delete cascade,
  PagTo     varchar(255) references PAGINA(URL)
                on update cascade on delete no action,
  primary key (PagFrom, PagTo)  )
```

In caso di update del link scegliamo di aggiornare i collegamenti di conseguenza. In caso di cancellazione di una delle pagine, la politica è differenziata:

- il cascade su PageFrom garantisce che la tabella COLLEGAMENTO venga automaticamente aggiornata (e mantenuta "minima")
- il no action su PageTo garantisce che non si possano cancellare pagine quando il sito ancora contiene collegamenti attivi a quelle pagine – garanzia di integrità.

Per completezza illustriamo la definizione delle altre tabelle:

```
create table UTENTE (
  IdUsr       varchar(30) primary key,
  Nome        varchar(255),
  Password    varchar(20),
  Categoria   varchar(20) default 'guest'  )

create table ACCESSO (
  IdUsr       varchar(30) references UTENTE(idusr),
  Data        date,
  Ora         time,
  UrlPag      varchar(255) references PAGINA(URL)
                   on update cascade on delete cascade,
  PaginaProvenienza  varchar(255),
  primary key (IdUsr, Data, Ora, UrlPag)  )
```

Un vincolo di integrità referenziale tra la coppia (UrlPag, PaginaProvenienza) e COLLEGAMENTO (PagTo, PagFrom) potrebbe aver senso (la navigazione può solo seguire link validi), ma non è applicabile a causa della scelta di usare il Null per caratterizzare le pagine iniziali di ogni percorso di navigazione. Infatti, PagFrom è parte della chiave di Collegamento, e non può assumere valori nulli.

B. LINGUAGGI FORMALI (7 punti)

1. Esprimere in Algebra Relazionale ottimizzata, Calcolo Relazionale e Datalog la seguente interrogazione: *Trovare le pagine "flop", cioè quelle che sono state visitate soltanto dai loro proprietari.*

Algebra relazionale

Π_{URL} ACCESSO – Π_{URL} (ACCESSO $\ltimes_{UrlPag=URL \wedge IdUsr<>Proprietario}$ PAGINA)

Calcolo relazionale

$\{ t \mid \exists\, t_A \in$ ACCESSO
$(t[UrlPag] = t_A[UrlPag] \wedge$
$\neg (\exists\, t_{A2} \in$ ACCESSO, $\exists\, t_P \in$ PAGINA
$t_{A2}[UrlPag] = t_A[UrlPag] \wedge$
$t_{A2}[UrlPag] = t_P[UrlPag] \wedge$
$t_{A2}[IdUsr] <> t_P[Proprietario])) \}$

Datalog

```
AccDaNonProprietario ( Url ) :-  ACCESSO ( IdU, _, _, Url, _ ),
                                 PAGINA ( Url, _, _, _, IdP ),  IdP <> IdU

AccSoloProprietario ( UrlFlop ) :-  ACCESSO ( _, _, _, UrlFlop, _ ),
                                    ¬ AccDaNonProprietario ( UrlFlop )

? - AccSoloProprietario ( X )
```

2. Esprimere in algebra relazionale ottimizzata la seguente interrogazione: *Trovare le pagine per cui l'ultimo utente che le ha consultate è il loro proprietario* [si presti attenzione al fatto che l'istante di accesso è contraddistinto da data e ora].

PAGINA $\ltimes_{URL=UrlPag \wedge Proprietario=IdUsr}$
((ACCESSO – (ACCESSO $\ltimes_{UrlPag=UrlPag \wedge Data<Data}$ ACCESSO)) –
(ACCESSO $\ltimes_{UrlPag=UrlPag \wedge Data=Data \wedge Ora<Ora}$ ACCESSO))

3. Usando la ricorsione, esprimere la seguente interrogazione: *Sapendo che l'utente "Alex" ha visitato la pagina "p001" in data "03/09/03" alle ore "13:20:05", estrarre tutte le pagine cui l'utente ha fatto accesso successivamente.*

Raggiunge ("p001") :- ACCESSO ("Alex", "03/09/03", "13:20:05" "p001",_)

Raggiunge (X) :- ACCESSO ("Alex", Data, Ora, X, Y),
Raggiunge (Y),
Data >= 03/09/03, Ora > 13:20:05

? - Raggiunge (P)

Facoltativo (1 punto) Discutere come realizzare un insieme di regole che traccino il percorso seguito da Alex dopo l'accesso alla pagina "p001" in data "03/09/03" alle ore "13:20:05", cioè che estraggano il numero progressivo di accesso, l'ora e la pagina acceduta, limitandosi ad osservare gli accessi del giorno corrente (cioè in data "03/09/03").

PercorsoAlex (Pag, Ora, **1**) :- ACCESSO ("Alex", "03/09/03", Ora, Pag, _),
Ora = "13:20:05", Pag = "p001"

Next (Pg, Or2, NumPr+**1**) :- ACCESSO ("Alex", "03/09/03", Or2, Pg, from),
PercorsoAlex (from, Or1, NumPr), Or2 > Or1.

PercorsoAlex (Pg, Or, Np) :- Next (Pg, Or, Np), ¬ Futuro (Pg, Or, Np)

Futuro (Pag, O2, N) :- Next (Pag, O2, N), Next (Pag, O1, N), O2 > O1

Si noti che la soluzione proposta utilizza, nella prima regola, la possibilità di inserire costanti nel letterale che costituisce la testa. Nella seconda regola, inoltre, si è ipotizzato di poter effettuare operazioni aritmetiche sui valori estratti dai letterali del corpo – un incremento di una unità, nella fattispecie.

C. Interrogazioni in SQL (8 punti)

1. *Estrarre la dimensione totale in KByte delle pagine scaricate dagli "studenti" nel luglio 2003.*

```
select sum(DimInByte)/1024
from (ACCESSO A join UTENTE U on A.IdUsr = U.IdUsr)
      join PAGINA on UrlPag = URL
where Categoria = 'Studente' and
      Data between 01/07/03 and 31/07/03
```

2. *Trovare i titoli delle pagine che non sono state visitate dai loro proprietari nell'ottobre 2003.*

```
select distinct Titolo
from PAGINA P
where URL not in (select UrlPag
                  from ACCESSO
                  where IdUsr = P.Proprietario and
                        Data between 1/10/03 and 31/10/03)
```

3. *Determinare la categoria di utenti (diversa da "studenti") che ha effettuato il maggior numero di accessi a pagine di tipo "didattica".*

```
select Categoria
from (ACCESSO A join UTENTE U on A.IdUsr = U.IdUsr)
      join PAGINA on UrlPag = URL
where Categoria <> 'Studenti' and Tipo = 'Didattica'
group by Categoria
having count(*) >= all
   ( select count(*)
     from (ACCESSO A join UTENTE U on A.IdUsr = U.IdUsr)
            join PAGINA on UrlPag = URL
     where Categoria <> 'Studenti' and Tipo = 'Didattica'
     group by Categoria )
```

oppure

```
create view SOMMA_ACCESSI(Cat, Num) as
select Categoria, Count(*)
from (ACCESSO A join UTENTE U on A.IdUsr = U.IdUsr)
      join PAGINA on UrlPag = URL
where Categoria <> 'Studenti' and Tipo = 'Didattica'
group by Categoria

select Categoria
from SOMMA_ACCESSI
where Num = ( select max(Num)
              from SOMMA_ACCESSI )
```

4. *Trovare gli utenti che hanno visitato almeno 10 pagine diverse e non hanno mai iniziato le loro navigazioni prima delle 13:00 [l'inizio di navigazione viene caratterizzato da un accesso privo di pagina di provenienza].*

```
select IdUsr, Nome
from UTENTE
where IdUsr in ( select IdUsr
                 from ACCESSO
                 group by IdUsr
                 having count(distinct UrlPag) >= 10 )
```

```
and IdUsr not in ( select IdUsr
                   from ACCESSO
                   where Ora < 13:00:00 and
                         PaginaProvenienza is null )
```

D. PROGETTO CONCETTUALE E LOGICO

Una città vuole attivare un servizio di "autobus a prenotazione" che raggiunga, a richiesta, utenti distribuiti sul territorio della città. Si vuole progettare il database che contiene i dati necessari.

Per accedere al servizio, gli utenti si devono iscrivere via Internet, indicando il loro nome, cognome, indirizzo, e-mail, estremi di un documento identificativo e, se disponibile, un numero di telefono cellulare. Inoltre, devono pagare un abbonamento che dà diritto ad un certo numero di corse e che può essere rinnovato; l'abbonamento si può pagare su Internet tramite carta di credito oppure presso varie sedi della compagnia municipale degli autobus. Ciascun utente è quindi caratterizzato da un numero di corse già acquistate, che viene ridotto quando prenota il servizio.

Ciascuna prenotazione ha un orario di partenza e un orario massimo di arrivo e fa riferimento a collegamenti tra due punti specifici della città; l'elenco dei punti raggiungibili, suddivisi in zone e caratterizzati da un nome di località, è pure disponibile su Internet. Ogni sera, il sistema raccoglie le prenotazioni, pianifica le corse, e talvolta avverte gli utenti di eventuali variazioni di orario, ad esempio se il luogo di arrivo non è raggiungibile entro il tempo limite indicato. La comunicazione avviene tramite SMS oppure e-mail e dà luogo a modifiche delle prenotazioni. Il sistema predispone poi un piano-corsa che risulta disponibile a ciascun autista; sul piano-corsa, gli autisti rilevano eventuali mancanze degli utenti nei luoghi previsti; gli autisti possono anche interagire con gli utenti tramite telefono cellulare. Al termine della corsa, eventuali mancanze di utenti o ritardi rispetto al massimo orario di arrivo vengono registrati nel database.

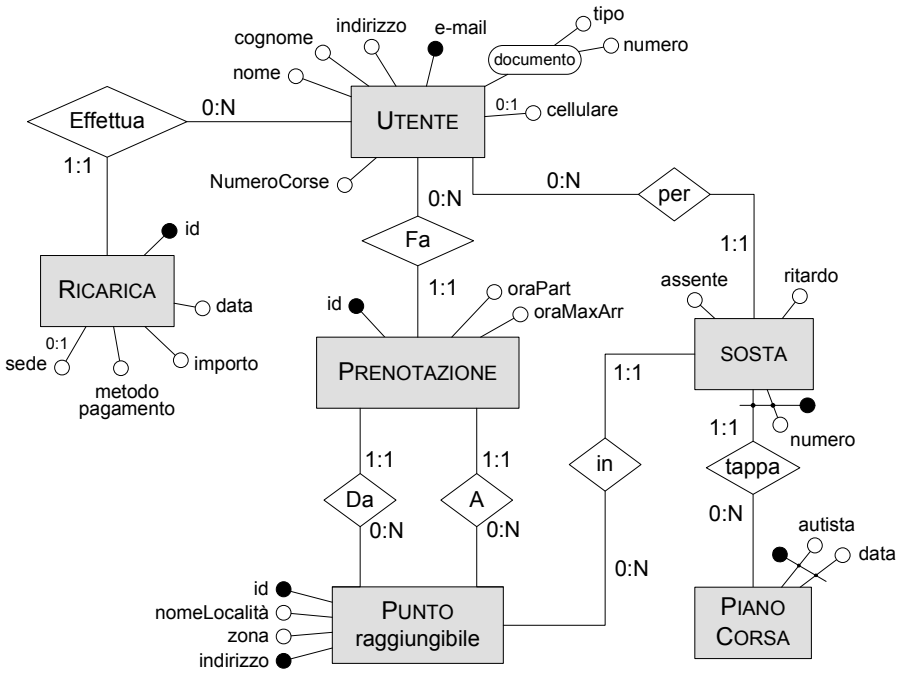
- *Ricordare che il progetto è valutato per completezza, correttezza, leggibilità, minimalità e autoesplicatività, e che anche il progetto logico è un **grafo** i cui nodi e archi devono essere disposti coerentemente col progetto concettuale.*
- *Ricordare anche di specificare gli **identificatori** di tutte le entità e le **cardinalità** di tutte le associazioni, e di disporre i due grafi su due facciate affiancate, in modo da poterli osservare simultaneamente.*[1]

Lo schema concettuale è nella pagina successiva.

Si noti che la soluzione proposta introduce il concetto di "sosta", che rappresenta una tappa nel viaggio e una "riga" nel piano-corsa cartaceo (in pratica un

[1] Queste indicazioni, riportate in tutti i temi d'esame, in seguito saranno omesse per brevità.

appuntamento con un cliente da parte di un autista). L'autista annota assenze e ritardi sul piano cartaceo, ma a fine corsa le annotazioni sono riportate nel database, dove resta permanentemente registrata la composizione dei piani e dei percorsi effettuati.



Allievi INF e TLC - Proff. S. Ceri e M. Matera
Primo Appello – 13 Febbraio 2004

PROPRIETARIOAUTO (IdProp, Targa, DataAcq, DataVendita, Nome, ComuneResid, ProvResid)
AUTOIMM (Targa, ComImm, ProvImm, DataImm, Marca, Modello, Colore, PotenzaKW)
DEMOLIZIONE (Targa, Data, Luogo, NomeDemolitore)
PRA (Comune, Provincia, Regione)

Lo schema si riferisce alle immatricolazioni, ai passaggi di proprietà e alle demolizioni di auto in Italia. Si ipotizzi che ogni auto possa essere immatricolata una sola volta, che più persone possano essere comproprietarie di una auto (nel caso, le date di acquisto devono coincidere) e che l'attributo DataVendita valga **Null** fino all'avvenuta trascrizione degli atti di vendita.

A. DDL (2 punti)

Scrivere i comandi SQL per
1) Creare le tabelle AUTOIMM e PRA, effettuando opportune ragionevoli ipotesi su domini, vincoli e reazioni ai cambiamenti.
2) Aggiungere alla tabella PRA l'attributo TipoSede, che assuma il valore "Normale" in assenza di altre indicazioni.

```
create table AUTOIMM (
  Targa      varchar(7) primary key,
  ComImm     varchar(127),
  ProvImm    varchar(2),
  DataImm    date > 01/01/1900   default sysdate(),
  Marca      varchar(30),
  Modello    varchar(30),
  Colore     varchar(30),
  PotenzaKW integer between 1 and 1000,
  foreign key (ComImm, ProvImm)
                references PRA(ComImm, ProvImm)
                   on delete no action on update no action )

create table PRA (
  Comune      varchar(127),
  Provincia  varchar(127),
  Regione     varchar(25),
  primary key (Comune, Provincia)  )
```

B. LINGUAGGI FORMALI (7 punti)

1. Esprimere in Algebra Relazionale ottimizzata, Calcolo Relazionale e Datalog la seguente interrogazione: *trovare i proprietari che hanno acquistato sempre e solo auto nuove.*

Algebra relazionale

$\Pi_{IdProp,Nome}$ PROPAUT – $\Pi_{IdProp,Nome}$ (PROPAUT $\ltimes_{Targa=Targa \wedge DataAcq>DataAcq}$ PROPAUT)

Attenzione! La seguente soluzione è errata, e il motivo può apparire subdolo:

~~$\Pi_{IdProp, Nome}$ (PROPAUT – (PROPAUT $\triangleright\!\!<_{Targa=Targa \wedge DataAcq> DataAcq}$ PROPAUT))~~

La tabella PROPRIETARIOAUTO contiene valori replicati per tutte le coppie di attributi (IdProp, Nome) per le tuple relative alle auto possedute (oggi o in passato) da una stessa persona. Se un proprietario *p* è (o è stato) in possesso di un'auto usata di targa t_1 ma ha posseduto in precedenza (o possiede anche) un'altra auto t_2 acquistata nuova, *p* non deve appartenere al risultato. Nella soluzione errata la tupla con t_1 è estratta dal semi-join e correttamente sottratta all'insieme di tutte le proprietà, ma *p* resta "in gioco" in virtù dell'altro acquisto, documentato dalla tupla con la targa t_2, che non è eliminata. È necessario quindi che, all'applicazione della differenza, non sopravvivano informazioni come t_1 *e* t_2 (ad esempio lo stesso vale per le date); così, per effetto dell'eliminazione dei duplicati, ogni proprietario compare al più una volta e non rischia di "sopravvivere" alla differenza per effetto dei dati di "qualche altra auto nuova" da lui posseduta oltre a quella usata.

Calcolo relazionale

Si consideri innanzitutto la soluzione seguente, che sembra giusta ma non lo è, in quanto considera una proprietà sola, tale per cui non esiste, ***per quell'auto***, un proprietario precedente. Essa estrae tutti i proprietari di ***almeno una*** auto nuova, non di sole auto nuove. Per scriverla correttamente si può quantificare universalmente (con $\forall$) oppure escludere la presenza di qualsiasi altra auto usata di proprietà dello stesso proprietario – come nella versione proposta sotto.

~~$\{ t \mid \exists t_P \in$ PROPRIETARIOAUTO~~ ***–ERRATA–***
~~$(t[IdProp, Nome] = t_P[IdProp, Nome] \wedge$~~
~~$\neg (\exists t_{P2} \in$ PROPRIETARIOAUTO~~
~~$t_{P2}[Targa] = t_P[Targa] \wedge$~~
~~$t_{P2}[DataAcq] < t_P[DataAcq]$)~~
~~)~~
~~$\}$~~

$\{ t \mid \exists t_P \in$ PROPRIETARIOAUTO ***CORRETTA***
$(t[IdProp, Nome] = t_P[IdProp, Nome] \wedge$
$\neg (\exists \mathbf{t_{P1}} \in$ PROPRIETARIOAUTO, $\exists \mathbf{t_{P2}} \in$ PROPRIETARIOAUTO
$\mathbf{t_P}[IdProp] = \mathbf{t_{P2}}[IdProp] \wedge$
$t_{P1}[Targa] = t_{P2}[Targa] \wedge$
$t_{P1}[DataAcq] < t_{P2}[DataAcq]))$
$\}$

Datalog

CompratoUsatoUnaVolta (P) :- PROPRIETARIOAUTO (P, Targa, Data1, _, _, _),
PROPRIETARIOAUTO (_, Targa, Data2, _, _, _),
Data1 > Data2

CompraSoloNuove (Id, Nome) :- PROPRIETARIOAUTO (Id, _, _, _, Nome, _),
¬ CompratoUsatoUnaVolta (Id)

? - CompraSoloNuove (X, Y)

2. Esprimere in algebra relazionale ottimizzata la seguente interrogazione: *trovare i proprietari che hanno posseduto auto tutte delle stesso colore:*

$\Pi_{IdProp, Nome}$ PRO –
$\Pi_{IdProp, Nome} ((PRO \bowtie AUT) \ltimes_{IdProp=IdProp \wedge Colore<>Colore} (PRO \bowtie AUT))$

C. Interrogazioni in SQL (8 PUNTI)

1. *Calcolare, per ogni regione, la potenza media in KW delle auto immatricolate nel 2003.*

```
select Regione, avg(PotenzaKW)
from AUTOIMM A join PRA P
     on ( ComImm = Comune and ProvImm = Provincia )
where DataImm between 01/01/03 and 31/12/03
group by Regione
```

2. *Trovare le targhe delle auto demolite senza mai subire passaggi di proprietà.*

~~*select Targa*~~
~~*from DEMOLIZIONE D*~~ ***ERRATA!*** *(ci sono le multiproprietà…)*
~~*where 1 = (select count(*)*~~
~~*from PROPRIETARIOAUTO*~~
~~*where Targa = D.Targa)*~~

```
select Targa
```

```
from DEMOLIZIONE D
where 1 = ( select count(distinct DataAcq)
            from PROPRIETARIOAUTO
            where Targa = D.Targa )
```

oppure anche, ad esempio:

```
( select Targa          Tutte le auto demolite…
   from DEMOLIZIONE D )
    except
( select Targa          …meno quelle che sono state vendute almeno una volta
  from PROPRIETARIOAUTO
  where DataVendita IS not NULL )
```

o ancora:

```
select Targa            Tutte le auto demolite che non hanno avuto due proprietari
from DEMOLIZIONE
where Targa not in
   ( select A.Targa
     from PROPAUTO A join PROPAUTO B on A.Targa = B.Targa
     where A.DataAsq <> B.DataAcq )
```

3. *Determinare il modello di auto più "gradito" agli automobilisti della provincia di Savona (si conti come "preferenza" ogni acquisto del nuovo o dell'usato).*

```
select Modello
from AUTOIMM A join PROPRIETARIOAUTO P on A.Targa = P.Targa
where ProvResid = 'SV'
group by Modello
having count(*) >= all
          ( select count(*)
            from AUTOIMM A join PROPAUTO P on A.Targa = P.Targa
            where ProvResid = 'SV'
            group by Modello )
```

oppure

```
create view PREFERENZE-SAVONA(Mod, Num) as
select Modello, Count(*)
from AUTOIMM A join PROPRIETARIOAUTO P on A.Targa = P.Targa
where ProvResid = 'SV'
group by Modello

select Mod
from PREFERENZE-SAVONA
where Num = ( select max(Num)
              from PREFERENZE-SAVONA )
```

4. *Costruire il registro delle auto in circolazione, nella forma (Targa, Modello, Proprietario, ComuneResPr, ProvinciaResPr), considerando solo l'ultimo proprietario di ogni auto non ancora demolita.*

```
select Targa, Modello, IdProp as Proprietario,
       ComuneRes as ComuneResPr, ProvRes as ProvResPr
from AUTOIMM A join PROPRIETARIOAUTO P on A.Targa = P.Targa
where A.Targa not in ( select Targa
                       from DEMOLIZIONE )
  and not exists ( select *
                   from PROPRIETARIOAUTO
                   where Targa = P.Targa and
                         DataAcq > P.DataAcq )
```

O, ancora più semplicemente:

```
select A.Targa, Modello, IdProp as Proprietario,
       ComuneRes as ComuneResPr, ProvRes as ProvResPr
from AUTOIMM A join PROPRIETARIOAUTO P on A.Targa = P.Targa
where A.Targa not in ( select Targa
                       from DEMOLIZIONE )
  and DataVendita IS NULL
```

ricordando che una data di vendita a **Null** significa che P è l'ultimo proprietario.

D. PROGETTO CONCETTUALE E LOGICO

Si deve progettare una base di dati per un ambulatorio che effettua test clinico-biologici e analisi radiologiche. Il sistema memorizzerà i test che l'ambulatorio è in grado di svolgere e per ogni tipo di test il nome, una breve descrizione e il costo della prestazione. Per i soli test clinico-biologici, è necessario memorizzare i valori di riferimento (valore minimo e massimo), che poi possono essere utili per redigere l'esito dei test eseguiti dai pazienti.

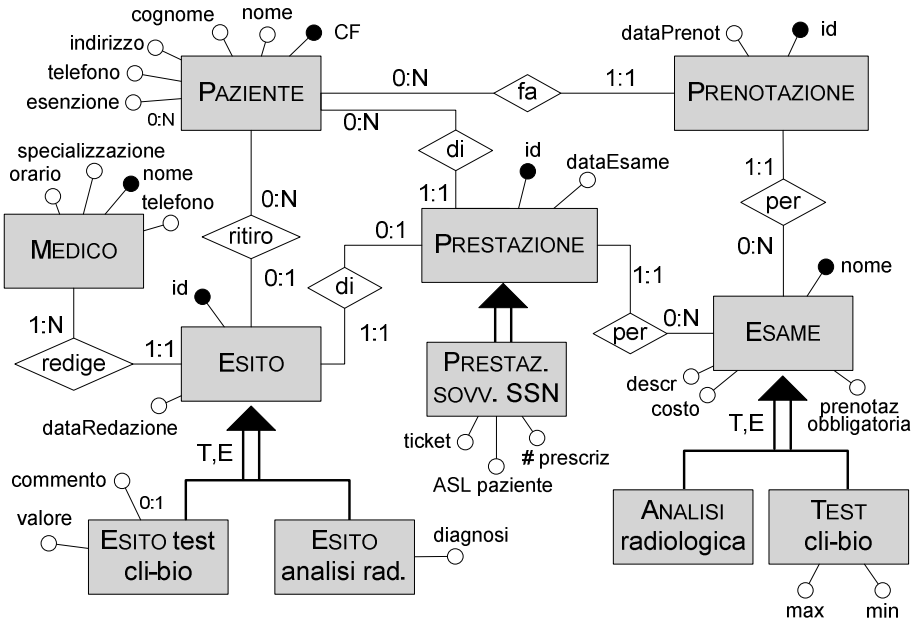
Per i pazienti che eseguono le analisi, è necessario memorizzare nome, cognome, codice fiscale, indirizzo, recapito telefonico e eventuali esenzioni. Per ogni richiesta di prestazione, sarà necessario registrare se il paziente intende usufruire della sovvenzione del Servizio Sanitario Nazionale. In tal caso, dovrà essere memorizzato anche il numero della prescrizione medica, la ASL di appartenenza del paziente e l'importo del "ticket" da pagare, che dipenderà dal tipo di esenzione eventualmente posseduta dal paziente.

Per alcuni test clinico-biologici, i pazienti devono necessariamente prenotarsi in anticipo e il test potrà essere svolto solo in presenza di prenotazione; per tutti gli altri sarà invece sufficiente presentarsi presso l'ambulatorio.

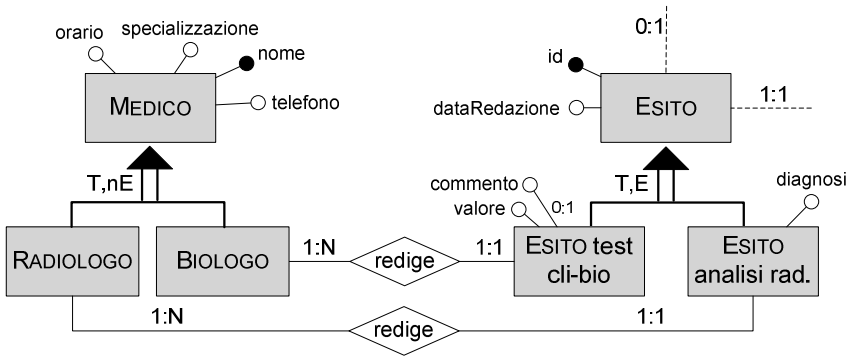
La base di dati deve inoltre memorizzare l'esito degli esami. L'esito di un esame radiologico è una diagnosi definita dal medico radiologo. Per i test clinico-biologici, l'esito consiste invece in un risultato numerico, da confrontare con i

valori di riferimento definiti per lo specifico test, ed eventualmente in un commento che evidenzi che il risultato è al di fuori dell'intervallo di tali valori.

Per ogni esito dovrà inoltre essere registrata la data in cui viene redatto e il biologo o il radiologo responsabile; per questi ultimi dovrà inoltre essere indicato un recapito telefonico e un orario in cui potranno essere contattati per eventuali chiarimenti. Nel caso di mancato ritiro di un esito entro una settimana dalla data di preparazione, la segreteria dell'ambulatorio contatterà telefonicamente l'utente.



Si noti che in questa soluzione i biologi e i radiologi sono distinti dalla specializzazione, ma occorre esprimere un vincolo a parte per imporre che redigano solamente gli esiti degli esami per cui hanno competenza. In alternativa possiamo introdurre una generalizzazione anche per i medici, sdoppiando redige:



Allievi INF e TLC - Proff. S. Ceri e M. Matera
Secondo Appello – 28 Giugno 2004

DEPLIANT (ID, Titolo, Compositore, NumPag, Formato, Data, Durata)
CONTIENE (D-ID, P-ID, Foto, Descrizione, Sconto)
PRODOTTO (P-ID, Nome, Categoria, CostoListino)

Ogni depliant descrive un insieme di prodotti e ha un titolo, un compositore, un numero di pagine, un formato, la data di pubblicazione e una durata di validità. Lo sconto è espresso in percentuale. Esprimere in SQL le seguenti interrogazioni:

A. SQL (9 punti)

1. *Estrarre il depliant valido (rispetto alla data corrente sysdate()) che presenta lo sconto più alto relativo al prodotto MSOffice.*

a) Interpretando 'MSOffice' come P-ID (Data di tipo **date**, Durata di tipo **interval**)

```
select ID, Titolo
from  DEPLIANT join CONTIENE on ID = D-ID
where sysdate() between Data and (Data + Durata) and
   P-ID = 'MSOffice' and
   Sconto = ( select max(Sconto)
              from DEPLIANT join CONTIENE on ID = D-ID
              where sysdate between Data and (Data+Durata)
                and P-ID = 'MSOffice' )
```

b) Se invece si interpreta MSOffice come Nome (e quindi, in teoria, più prodotti con tale nome possono partecipare a un depliant con sconti diversi), a rigore il confronto dovrebbe essere fatto tra i massimi sconti di ogni depliant (sempre rispetto ai prodotti della "famiglia" MSOffice)

```
select ID
from (DEPLIANT join CONTIENE C on ID = D-ID)
      join PRODOTTO P on C.P-ID = P.P-ID
where sysdate() between Data and (Data + Durata) and
      Name = 'MSOffice'
group by ID
having max(sconto) >= all
       ( select max(sconto)
         from  (DEPLIANT join CONTIENE C on ID = D-ID)
                join PRODOTTO P on C.P-ID = P.P-ID
         where sysdate() between Data and (Data + Durata) and
               Name = 'MSOffice'
         group by ID )
```

2. *Estrarre i depliant che non contengono prodotti il cui costo scontato è inferiore a 1000 Euro e il cui numero di pagine supera il triplo dei prodotti presenti.*

Considerando lo Sconto un intero compreso tra 0 e 100:

```
select ID, Titolo
from  DEPLIANT X
where ID not in
     ( select D-ID
       from CONTIENE C join PRODOTTO P on C.P-ID=P.P-ID
       where 1000 > (CostoListino*(100-Sconto)/100) )
  and NumPag > ( select 3 * count(*)
                 from CONTIENE
                 where D-ID = X.ID )
```

3. *Estrarre il nome dei prodotti per i quali esistano depliant in cui la differenza fra lo sconto più alto e lo sconto più basso praticato sia superiore al 10 per cento e la cui categoria sia "PC" oppure "Mouse".*

Considerando lo scostamento tra gli sconti sullo stesso prodotto in depliant diversi:

```
create view MAX-DELTA ( Prod, Delta ) as
select P-ID, max(Sconto) - min(Sconto)
from CONTIENE
group by P-ID
select distinct Nome
from  PRODOTTO
where Categoria in ( 'PC', 'Mouse' ) and
      P-ID in ( select Prod
                from MAX-DELTA
                  where Delta > 10 )
```

Considerando invece lo scostamento tra gli sconti su diversi prodotti in uno stesso depliant:

```
create view MAX-D ( Dep, Delta ) as
select D-ID, max(Sconto) - min(Sconto)
from CONTIENE
group by D-ID

select distinct Nome
from  PRODOTTO
where Categoria in ('PC', 'Mouse' ) and
      P-ID in ( select P-ID
                from MAX-D join CONTIENE on Dep = D-ID
                where Delta > 10 )
```

B. LINGUAGGI FORMALI DI INTERROGAZIONE (5 punti)

1. Esprimere in Calcolo e Datalog l'interrogazione 1. dell'esercizio precedente.

Nell'interpretazione **a**)

Calcolo relazionale

$$
\begin{aligned}
&\{ t \mid \exists\, t_D \in \text{DEPLIANT}, \exists\, t_C \in \text{CONTIENE} \\
&\quad (t[\text{ID}] = t_D[\text{ID}] \wedge \\
&\quad\ t_C[\text{P-ID}] = \text{`MSOffice'} \wedge \\
&\quad\ t_D[\text{ID}] = t_C[\text{D-ID}] \wedge \\
&\quad\ \text{sysdate}() >= t_D[\text{Data}] \wedge\ \text{sysdate}() <= (t_D[\text{Data}] + t_D[\text{Durata}]) \wedge \\
&\quad\ \neg (\ \exists\, t_{D2} \in \text{DEPLIANT}, \exists\, t_{C2} \in \text{CONTIENE} \\
&\qquad\ (t_{C2}[\text{P-ID}] = \text{`MSOffice'} \wedge \\
&\qquad\ t_{D2}[\text{ID}] = t_{C2}[\text{D-ID}] \wedge \\
&\qquad\ \text{sysdate}() >= t_{D2}[\text{Data}] \wedge \text{sysdate}() <= (t_{D2}[\text{Data}] + t_{D2}[\text{Durata}]) \wedge \\
&\qquad\ t_{C2}[\text{Sconto}] > \mathbf{t_C}[\text{Sconto}] \ ) \ ) \\
&\quad) \ \}
\end{aligned}
$$

Datalog

```
ScontoValidoProdotto ( Dep, Prod, Sconto ) :-
                                DEPLIANT ( Dep, _, _, _, _, Data, Durata ),
                                CONTIENE ( Dep, Prod, _, _, Sconto ),
                                sysdate() >= Data, sysdate() <= ( Data + Durata )

ScontoOfficeNonMax (Dep) :- ScontoValidoProdotto ( Dep, 'MSOffice', Sconto ),
                            ScontoValidoProdotto ( _, 'MSOffice', Sconto1 ),
                            Sconto < Sconto1

ScontoOfficeMax ( Dep ) :- ScontoValidoProdotto ( Dep, 'MSOffice', _ ),
                           ¬ ScontoOfficeNonMax ( Dep )

? - ScontoOfficeMax ( X )
```

C. DDL (2 punti)

Scrivere i comandi SQL per

1. Creare la tabella CONTIENE, effettuando opportune ragionevoli ipotesi su domini e vincoli e reazioni ai cambiamenti.
2. Eliminare dalla tabella CONTIENE l'attributo Foto.

```
create table CONTIENE (
   D-ID varchar(10) references DEPLIANT(ID)
                  on update cascade on delete cascade,
   P-ID varchar(10) references PRODOTTO(P-ID)
                  on update cascade on delete no action,
   Foto blob,
   Descrizione varchar(255),
   Sconto integer not null between 0 and 100,
   primary key (D-ID, P-ID)  )
```

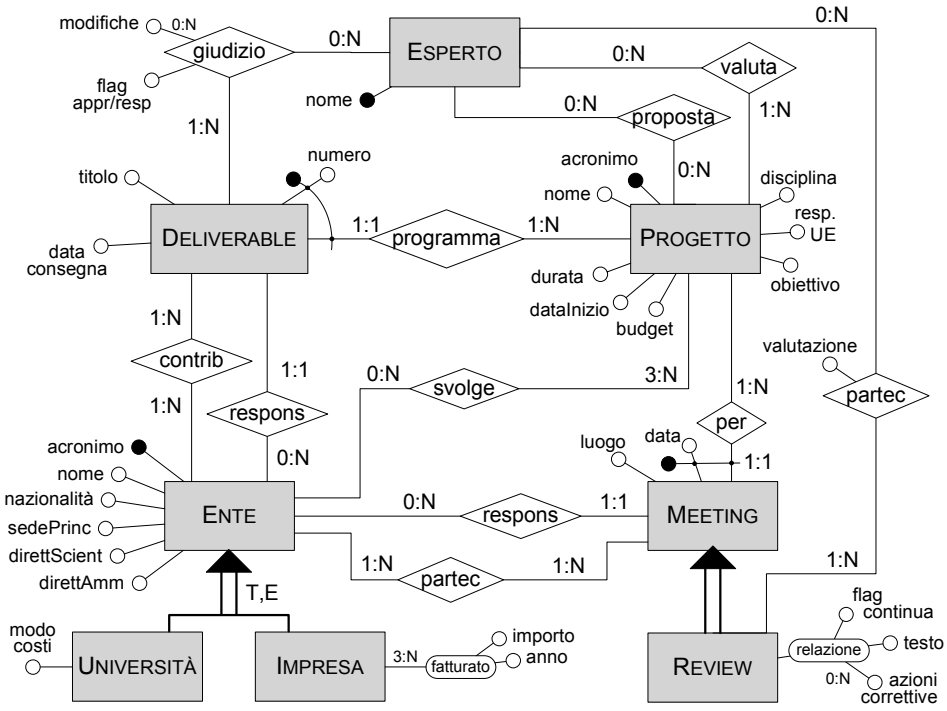
Alla cancellazione di un depliant non ha senso conservare l'elenco dei prodotti che vi appartengono. Invece vietiamo la cancellazione di un prodotto finché restano nel DB depliant relativi a quel prodotto.

```
alter table CONTIENE drop column Foto
```

D. PROGETTO CONCETTUALE E LOGICO

Si vuole costruire un database che descrive la partecipazione di università e imprese a progetti europei. Tali enti hanno un nome, un acronimo, una nazionalità e sede principale, un direttore scientifico e un direttore amministrativo. Le università hanno la possibilità di imputare costi secondo due modalità differenti, le imprese devono indicare il fatturato negli ultimi tre anni di attività. Ciascun progetto, caratterizzato da un acronimo, un nome, una data di inizio, una durata ed un budget, è svolto da un consorzio di almeno tre enti "partner". Il progetto ha un obiettivo e si inserisce all'interno di una disciplina. Ogni progetto si impegna a consegnare entro tempi predefiniti un insieme di risultati ("deliverable"), cui partecipano un certo numero di partner, di cui uno è designato come responsabile. Durante il progetto si svolgono numerosi incontri tecnici, cui partecipano alcuni dei partner.

Ciascun progetto è affidato ad un responsabile, scelto tra i dipendenti della Comunità Europea, che ne cura la corretta esecuzione. Per fare ciò, il responsabile nomina dei revisori scelti all'interno di un insieme di esperti già preventivamente approvati dalla Comunità Europea; ciascun consorzio di partner di progetto può proporre alcuni esperti. La valutazione dei progetti avviene durante particolari incontri tecnici di revisione ("review"), in cui il progetto presenta i suoi risultati in base alla loro scadenza di realizzazione; al termine della review, i revisori danno una valutazione scritta del progetto e in particolare possono approvare ciascun risultato oppure chiederne una modifica oppure infine respingerlo, e prendere decisioni complessive circa la continuazione del progetto e le azioni correttive che il progetto dovrà svolgere entro la successiva review.



Si è ritenuto che i giudizi sui deliverable e le valutazioni sui progetti siano formulati dai singoli reviewer in modo "individuale" e memorizzati in modo da tener traccia dei contributi distinti, e che sia poi il dipendente responsabile per la Comunità Europea a fonderli in un giudizio sintetico sul progetto, che costituisce la relazione finale di ogni incontro di revisione. Per questo motivo l'attributo valutazione è sull'associazione partec e non sull'entità REVIEW, così come gli attributi modifiche e flag appr/resp sono sull'associazione giudizio e non sono sull'entità DELIVERABLE.

Allievi INF e TLC - Proff. S. Ceri e M. Matera
Terzo Appello – 10 Settembre 2004

Si consideri il seguente schema, relativo a un sistema di prenotazione e noleggio di biciclette, in cui le indicazioni orarie possono assumersi piene (dalle 8 alle 19):

CLIENTE (Num, Nome, TipoDoc, NumeroDoc)
PRENOTA (Num-Cli, TipoBici, Giorno, OraInizio, OraFine)
USA (Num-Cli, Num-Bici, Giorno, OraInizio, OraFine, Costo)
BICICLETTA (Num-Bici, TipoBici, CostoOra, CostoGiorno)

A. DDL (2 punti)

Scrivere i comandi SQL per creare le tabelle PRENOTA e BICICLETTA, effettuando opportune ragionevoli ipotesi su domini, vincoli e reazioni ai cambiamenti.

```
create Table BICICLETTA (
  Num-Bici     integer primary key,
  TipoBici     varchar(50) not null,
  CostoOra     decimal(4,2),
  CostoGiorno  decimal(5,2)  )

create table PRENOTA (
  Num-Cli  integer references CLIENTE(Num)
                  on delete cascade on update cascade,
  TipoBici varchar(50) default 'City Bike'
                          check TipoBici in ( select TipoBici
                                              from BICICLETTA )
  Giorno  date,
  OraInizio  integer between 8 and 18,
  OraFine    integer check ( OraFine-OraInizio > 0 and
                             OraFine between 9 and 19),
  Primary key (Num-Cli, TipoBici, Giorno)  )
```

N.B.: **non è possibile** accendere un vincolo di integrità referenziale tra il TipoBici della tabella PRENOTA e il TipoBici della tabella BICICLETTA, poiché in BICICLETTA TipoBici ***non è unique***.
Si rammenta infatti che, affinché sia possibile definire una chiave esterna, i valori esterni devono (collettivamente) essere unique – *non chiave, ma neanche meno di unique* – e che ciò è necessario, ad esempio, per poter applicare in modo non ambiguo le politiche di reazione a modifiche e cancellazioni nella tabella esterna (la tabella "padre").

Ciononostante, è del tutto sensato imporre il vincolo che non si possa prenotare una bicicletta di un tipo di cui il noleggiatore non possegga alcun esemplare (che sia o che non sia in prestito al momento della prenotazione). Per esprimere il vincolo si può usare una **check** che ha lo stesso potere vincolante della chiave esterna ma non garantisce alcuna politica di reazione ai cambiamenti.

B. LINGUAGGI FORMALI (7 punti)

1. Esprimere in Algebra Relazionale ottimizzata, Calcolo Relazionale e Datalog la seguente interrogazione: *trovare il nome dei clienti che hanno prenotato bici di tipo "da corsa" senza mai usarle.*

Algebra relazionale

A) Prima interpretazione: *clienti che hanno qualche prenotazione ma nessun uso*

a1) Una prima idea per una soluzione: join tra chi non ha mai usato una bici da corsa e chi ne ha prenotata una:

$$\Pi_{Nome} \Big(\big(CLI - (CLI \rtimes_{Num=Num\text{-}Cli} (USA \rtimes (\sigma_{Tipo='Corsa'} BIC)))\big) \rtimes_{Num=Num} (CLI \rtimes_{Num=Num\text{-}Cli} (\sigma_{Tipo='Corsa'} PRE)) \Big)$$

a2) Si può anche pensare di sottrarre i clienti che hanno usato una bici da corsa da quelli che ne hanno prenotata una:

$$\Pi_{Nome} \Big((CLI \rtimes_{Num=Num\text{-}Cli} (\sigma_{Tipo='Corsa'} PRE)) - (CLI \rtimes_{Num=Num\text{-}Cli} (USA \rtimes (\sigma_{Tipo='Corsa'} BIC))) \Big)$$

o, forse ancora più chiaro…:

$$\Pi_{Nome} (CLI \rtimes_{Num=Num\text{-}Cli} (\Pi_{Num\text{-}Cli} (\sigma_{Tipo='Corsa'} PRE) - \Pi_{Num\text{-}Cli} (USA \rtimes (\sigma_{Tipo='Corsa'} BIC))))$$

B) Seconda interpretazione: *clienti tali che a nessuna delle loro prenotazioni di bici da corsa (e una deve esistere di sicuro) corrisponda un uso di una bici da corsa nello stesso giorno* (quindi gli eventuali usi di bici da corsa di tali clienti, che pure possono esserci, non devono però avere una corrispondente prenotazione). Si può riformulare l'interrogazione in "tutti i clienti con almeno una prenotazione di bici da corsa meno tutti quelli con una prenotazione di bici da corsa a cui corrisponde un uso per lo stesso giorno e ora":

$$\Pi_{Nome}(CLI \ltimes_{Num=Num\text{-}Cli} (\Pi_{Num\text{-}Cli}(\sigma_{Tipo='Corsa'}PRE) - \Pi_{Num\text{-}Cli}(\sigma_{Tipo='Corsa'}PRE \ltimes_{Gior=Gior \wedge OraInizio=OraInizio \wedge Num\text{-}Cli=Num\text{-}Cli} USA \ltimes_{Num\text{-}Bici=Num\text{-}Bici} \sigma_{Tipo='Corsa'}BIC)))$$

N.B.: A) e B) sono in generale diversi, ma se la prenotazione è obbligatoria sono equivalenti. Si noti però che lo schema non permette in alcun modo di arguire se la prenotazione sia obbligatoria o no.

Calcolo relazionale (secondo l'interpretazione A):

$\{ t \mid \exists t_P \in$ PRENOTA, $\exists t_C \in$ CLIENTE *(Esiste una prenotazione)*
$(t[Num, Nome] = t_C[Num, Nome] \wedge$
$t_P[Num\text{-}Cli] = t_C[Num] \wedge$ *("join" tra cliente e prenotazione)*
$t_P[Tipo] =$ "Corsa"$\wedge$ *(per una bici da corsa)*
$\neg (\exists t_B \in$ BICICLETTA, $\exists t_U \in$ USA *(però non esiste alcun uso)*
$(t_U[Num\text{-}Cli] = t_P[Num\text{-}Cli] \wedge$ *(dello **stesso** cliente)*
$t_B[Num\text{-}Bici] = t_U[Num\text{-}Bici] \wedge$ *(join)*
$t_B[TipoBici] =$ "Corsa")) *(per una bici da corsa)*
)
}

Datalog (secondo l'interpretazione A):

```
UsataBiciTipo ( C, T )   :-  BICICLETTA ( N, T, _, _ ),
                             USA ( C, N, _, _, _, _ )

PrenotaMaNonUsa ( Nom )  :-  CLIENTE ( Cliente, Nom, _, _ ),
                             PRENOTA ( Cliente, "Corsa", _, _, _ )
                             ¬ UsataBiciTipo ( Cliente, "Corsa" ),

? - PrenotaMaNonUsa ( Nom )
```

2. Esprimere in algebra relazionale ottimizzata la seguente interrogazione: *trovare il tipo dell'ultima bicicletta usata da Mario Rossi.*

Intendendo da "un" Mario Rossi; il nome, infatti, non è chiave.

Usiamo *l'assegnamento* (=, o *relazione virtuale*) per rendere più leggibile la soluzione. Si rammenta che l'assegnamento consiste nel definire una "sottoquery" il cui nome rappresenta la relazione calcolata come risultato dell'espressione associata, che si può poi liberamente usare in altre espressioni:

UltimoUsoMarioRossi =

$$(\text{USA} \ltimes_{\text{Num-Cli=Num}} (\sigma_{\text{Nome='MarioRossi'}} \text{CLI}))$$
$$-$$
$$((\text{USA} \ltimes_{\text{Num-Cli=Num}} (\sigma_{\text{Nome='MarioRossi'}} \text{CLI})) \ltimes_{\text{Num-Cli=Num-Cli} \wedge \textbf{(Giorno<Giorno} \vee \textbf{Giorno=Giorno} \wedge \textbf{OraInizio < OraInizio)}} (\text{USA} \ltimes_{\text{Num-Cli=Num}} (\sigma_{\text{Nome='MarioRossi'}} \text{CLI})))$$

Soluzione: $\Pi_{\text{TipoBici}} (\text{BIC} \ltimes \text{UltimoUsoMarioRossi})$

Si noti che i semi-join con il cliente sono entrambi necessari per restringere l'insieme degli usi da sottrarre ai soli usi di Mario Rossi. Diversamente si considererebbero solo gli ultimi usi *in assoluto*, e non gli ultimi tra i soli usi di qualche Mario Rossi).
Si noti anche che la soluzione si basa sul fatto che, anche se in uno stesso giorno uno stesso cliente non può usare la stessa bici più di una volta, non è detto che non usi due bici diverse nello stesso giorno. Per questo motivo non è sufficiente la condizione $\ltimes_{\text{Num-Cli=Num-Cli} \wedge \text{Giorno<Giorno}}$.

C. Interrogazioni in SQL (7 punti)

1. *Elencare il nome dei clienti che hanno usufruito di una bicicletta di tipo diverso da quello da loro prenotato [nota: far corrispondere prenotazioni e relativi usi].*

```
select Nome
from CLIENTE C, PRENOTA P, USA U, BICICLETTA B
where C.Num = P.Num-Cli and
      C.Num = U.Num-Cli and
      U.Num-Bici = B.Num-Bici and
      (P.Giorno, P.OraInizio, P.OraFine)
          =
      (U.Giorno, U.OraInizio, U.OraFine) and
      B.TipoBici <> P.TipoBici
```

Si noti l'uso del costruttore di tupla per confrontare l'uguaglianza di una tripla di valori. Non è particolarmente rilevante che l'uso corrisponda alla prenotazione anche per quanto riguarda l'ora di riconsegna della bicicletta (la condizione sull'ora di riconsegna è indicata in corsivo e può essere omessa).

2. *Elencare il nome dei clienti che hanno prenotato una bicicletta senza usarla per più di una volta, e che non hanno mai usato una bicicletta.*

Intersechiamo chi non ha mai usato alcuna bici con chi ha almeno due prenotazioni (intendendo "due generiche prenotazioni", non "due volte la **stessa** bici").

```
select Nome, Num
```

```
from CLIENTE
where Num not in (select Num-Cli from USA)
      intersect
select Nome, Num
from CLIENTE join PRENOTA on Num = Num-Cli
group By Nome, Num
having count(*) > 1
```

3. *Per ogni cliente, indicare la bicicletta usata per il maggior numero di ore ed il corrispondente numero totale di ore.*

In caso di parità di "monte ore" tra due o più bici per lo stesso cliente, estraiamo tutte le coppie (C.Num, U.Num-Bici) relative a quel cliente.

```
select Num, Nome, NumBici, sum(OraFine-OraInizio) as TotOre
from CLIENTE C join USA U on Num=Num-Cli
group by Num, NumBici, Nome
having sum(OraFine-OraInizio) >= all
                        ( select sum(OraFine-OraInizio)
                          from USA U2
                          where C.Num=U2.Num-Cli
                          group by NumBici )
```

N.B.: L'aggiunta di Nome alla target list rende più leggibile il risultato dell'interrogazione. Per poter includere tale attributo, però, occorre che sia anche nella clausola `group by`. Aggiungerlo, peraltro, non modifica la struttura del raggruppamento (infatti, se nel gruppo ci sono solo tuple relative allo stesso cliente, allora anche il suo Nome sarà uguale). Lo stesso varrebbe per il TipoBici, se si volesse includerlo nello schema della tabella risultato.

Se invece avessimo voluto trovare **per ogni bici** qual è il cliente che le è più affezionato, avremmo usato la stessa struttura per la query esterna ma una (leggermente!!) diversa query annidata:

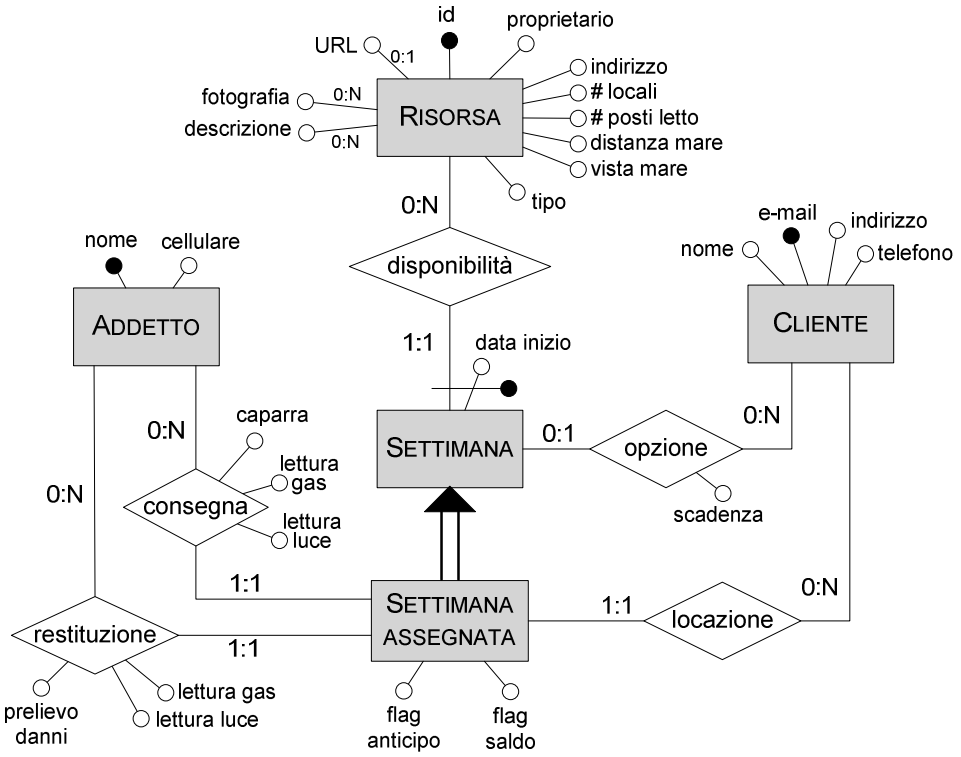
```
select NumBici, TipoBici, Num, Nome,
       sum(OraFine-OraInizio) as TotOre
from CLIENTE C join USA U on Num=Num-Cli
group by Num, NumBici, Nome, TipoBici
having sum(OraFine-OraInizio) >= all
                        ( select sum(OraFine-OraInizio)
                          from USA U2
                          where U.Num-Bici=U2.Num-Bici
                          group by Num-Cli )
```

D. PROGETTO CONCETTUALE E LOGICO

Un'agenzia turistica regionale gestisce tutte le informazioni per consentire la ricerca su Web e la locazione di case e villini. Ogni "risorsa" ha una collocazione, il numero di locali e di posti-letto, la distanza dal mare e la vista del mare. Alcune risorse hanno un sito Web proprio (indicato tramite URL), altre hanno una serie di fotografie e di testi descrittivi che vengono gestite dalla agenzia e presentate su Web.

Gli affitti vengono definiti a settimana, nei mesi estivi. Le varie settimane vengono definite dai proprietari come destinate all'affitto, e poi vengono progressivamente prenotate dai clienti; ciascuna settimana ha un prezzo differente.

I clienti sono noti in termini di nome, indirizzo, recapito telefonico ed e-mail. Il processo di prenotazione prevede: l'opzione di una casa (per una settimana), il pagamento dell'anticipo (30%), il pagamento del saldo (70%). Il processo di effettivo uso di una casa prevede i due momenti di consegna e restituzione della casa, che vengono curati da un addetto della agenzia, di cui è noto nome e numero di cellulare. Alla consegna viene versato un ulteriore ammontare per la copertura di eventuali danni, che è restituito al termine del soggiorno salvo prelievi che compensino i danni arrecati. Alla restituzione vengono letti i consumi di energia, a carico dei clienti.



Allievi INF e TLC - Proff. A. Campi, S. Ceri e G. Pozzi
Prove in Itinere –Novembre 2004 / Gennaio 2005

A. SQL (8 punti)

Si considerino le tabelle che rappresentano i dati di un gioco a quiz dove un concorrente deve rispondere alle domande proposte scegliendo la risposta tra un insieme di risposte predefinito per quella domanda.

CONCORRENTE (CodiceFiscale, Nome, Cognome, Città, Vincita)
DOMANDA (Id, Testo, Premio, IdRispostaCorretta)
RISPOSTA (IdDomanda, IdRisposta, Testo)
RISPOSTACONCORRENTE (CodiceFiscale, IdDomanda, IdRisposta)

Si noti che per ogni domanda esistono più risposte, di cui solo una è corretta; si assuma inoltre che tutte le domande siano poste a tutti i concorrenti.

1. *Estrarre il testo delle domande alle quali nessun concorrente ha mai risposto in modo errato.*

```
select Id, Testo
from DOMANDA D
where not exists
       ( select *
         from RISPOSTACONCORRENTE
         where IdDomanda = D.Id and
               IdRisposta <> D.IdRispostaCorretta )
```

oppure

```
select Id, Testo
from DOMANDA D
where IdRispostaCorretta = all ( select IdRisposta
                                 from RISPOSTACONCORRENTE
                                 where D.Id = IdDomanda )
```

oppure ancora

```
select Id, Testo
from DOMANDA)
   except
select Id, Testo
from DOMANDA join RISPOSTACONCORRENTE on Id = IdDomanda
where IdRisposta <> IdRispostaCoretta
```

Assolutamente NON, invece:
(Errore grave! chiede che nessuno abbia mai indovinato nulla!)

```
select Id, Testo
from DOMANDA
where not exists
        ( select *
          from DOMANDA join RISPOSTACONCORRENTE
               on Id = IdDomanda
          where IdRisposta <> IdRispostaCoretta )
```

2. *Estrarre le domande che tutti i vincitori di più di 1000 euro hanno sbagliato.*

Siccome tutte le domande sono poste a tutti i concorrenti, posso raggruppare **per domanda** le sole risposte sbagliate da vincitori di più di 1000 euro, e trattenere solo i gruppi con tante risposte quanti vincitori:

```
select D.Id
from DOMANDA D, RISPOSTACONCORRENTE RC, CONCORRENTE C
where RC.CodFis = C.CodFis and
      D.Id = RC.IdDomanda and
      RC.IdRisp <> D.RispCorr and
      C.Vincita > 1000
group by D.Id
having count (*) = ( select count(*)
                     from CONCORRENTE
                     where Vincita > 1000 )
```

In alternativa, si può definire una vista per individuare – ad esempio – tutte le risposte **giuste** dei concorrenti "ricchi", cercando poi le domande che non vi compaiono (perché nessun "ricco" ha risposto correttamente).

```
Crete view RispOkRicchi (IdDom, IdRisp) as
select Id, IdRispostaCorretta
from (DOMANDA join RISPOSTACONCORRENTE RC
      on Id = IdDomanda) join CONCORRENTE C
       on RC.Codice Fiscale = C.CodiceFiscale
where C.Vincita > 1000 and
      RC.IdRisposta = IdRispostaCorretta

select Id
from DOMANDA
where Id not in ( select IdDom
                  from RispOkRicchi )
```

Oppure ancora si può applicare direttamente la definizione: scelgo una domanda se la sua risposta corretta non compare tra quelle date dai ricchi (per quella domanda).

```
select ID
from DOMANDA
where (Id, IdRispostaCorretta) <> all          Oppure "not in"
        ( select IdDomanda, IdRisposta
          from CONCORRENTE C join RISPOSTACONCORRENTE RC
                 on C.CodFisc = RC.CodFisc
          where Vincita > 1000 )
```

3. *Scrivere un comando di update che assegna al campo Vincita di ogni Concorrente il valore calcolato in base alle risposte che ha dato.*

```
update CONCORRENTE C
set Vincita = ( select sum(Premio)
                from DOMANDA join RISPOSTACONCORRENTE R
                       on (Id = IdDomanda and
                           IdRisposta = IdRispostaCoretta)
                where R.CodiceFiscale = C.CodiceFiscale )
```

B. LINGUAGGI FORMALI DI INTERROGAZIONE (6 punti)

1. Esprimere in Algebra ottimizzata, Calcolo **e** Datalog la prima interrogazione dell'esercizio precedente.

Algebra relazionale

$$\Pi_{Id,Testo} (\text{DOMANDA} - (\text{DOMANDA} \rtimes_{Id=IdDomanda \wedge IdRispostaCorretta <> IdRisposta} \text{RISCON}))$$

Calcolo relazionale

$$\{ t \mid \exists t_D \in \text{DOMANDA} \quad \textit{(Esiste la domanda)}$$
$$(t[Id, Testo] = t_D[Id, Testo] \wedge$$
$$\neg (\exists t_R \in \text{RISCON} \quad \textit{(e non c'è alcuna risposta)}$$
$$(t_R[IdDomanda] = t_{\mathbf{D}}[Id] \wedge \quad \textit{(alla **stessa** domanda)}$$
$$t_R[IdRisposta] <> t_{\mathbf{D}}[IdRispostaCorretta]) \quad \textit{(che sia sbagliata)}$$
$$)) \}$$

Datalog

```
RispostaSbagliata ( IdDom ) :- DOMANDA ( IdDom, _, _, IdCorretta ),
                               RISPOSTACONCORRENTE ( _, IdDom, IdErrore ),
                               IdCorretta <> IdErrore
```

DomandaMaiSbagliata (Id, Testo) :- DOMANDA (Id, Testo, _, _),
¬ RispostaSbagliata (Id)

? - DomandaMaiSbagliata (X, Y)

2. Esprimere in **due** linguaggi a scelta tra Algebra, Calcolo e Datalog l'interrogazione che estrae il concorrente di Milano che ha vinto più soldi.

Algebra relazionale

$\Pi_{CodFisc,Nome,Cognome}(\sigma_{Città='Milano'} CONCORRENTE -$
$(\sigma_{Città='Milano'} CONCORRENTE \rtimes_{Vincita<Vincita} \sigma_{Città='Milano'} CONCORRENTE))$

Calcolo relazionale

$\{ t \mid \exists t_C \in CONCORRENTE$ *(Esiste il concorrente)*
$(t[CodFisc, Nome, Cognome] = t_C[CodFisc, Nome, Cognome] \wedge$
$t_C[Città] = 'Milano' \wedge$ *(di Milano)*
$\neg (\exists t_{C2} \in CONCORRENTE$ *(e non c'è alcun concorrente)*
$(t_{C2}[Città] = 'Milano' \wedge$ *(di Milano con)*
$t_{C2}[Vincita] > t_C[Vincita])$ *(una vincita superiore a **quella**)*
$)$
$)$
$\}$

Datalog

VincitaMilaneseSuperata(CF) :- CONCORRENTE (CF, _, _, 'Milano', VincBassa),
CONCORRENTE (_, _, _, 'Milano', VincAlta),
VincAlta > VincBassa

VincitaMassimaMilano (CodFisc) :- CONCORRENTE (CodFisc, _, _, _, _),
¬ VincitaMilaneseSuperata (CodFisc)

? - VincitaMassimaMilano (X)

C. DDL (2 punti)

Scrivere i comandi SQL per:

1. Creare la tabella RISPOSTACONCORRENTE, effettuando opportune e ragionevoli ipotesi su domini e vincoli e reazioni ai cambiamenti.

```
create table RISPOSTACONCORRENTE (
  CodiceFiscale char[16]                                  /* Non varchar!! */
         references CONCORRENTE(CodFisc)
                on delete no action on update cascade,
  IdDomanda integer,
  IdRisposta char,                    /* può bastare - ad esempio: a, b, c… ! */
  primary key (CodiceFiscale, IdDomanda),
  foreign key (IdDomanda, IdRisposta)
         references RISPOSTA(IdDomanda, IdRisposta)
                on delete cascade on update cascade  )
```

È fondamentale che l'integrità referenziale sia applicata alla **coppia** di valori in RISPOSTA. Ogni altra chiave esterna o è sintatticamente scorretta o rischia di non vincolare la risposta ad essere una risposta lecita (ad esempio, vincolare solo l'IdDomanda ad appartenere a DOMANDA ma non l'IdRisposta permetterebbe di rappresentare una risposta 'd' a una domanda che prevede solo tre opzioni).

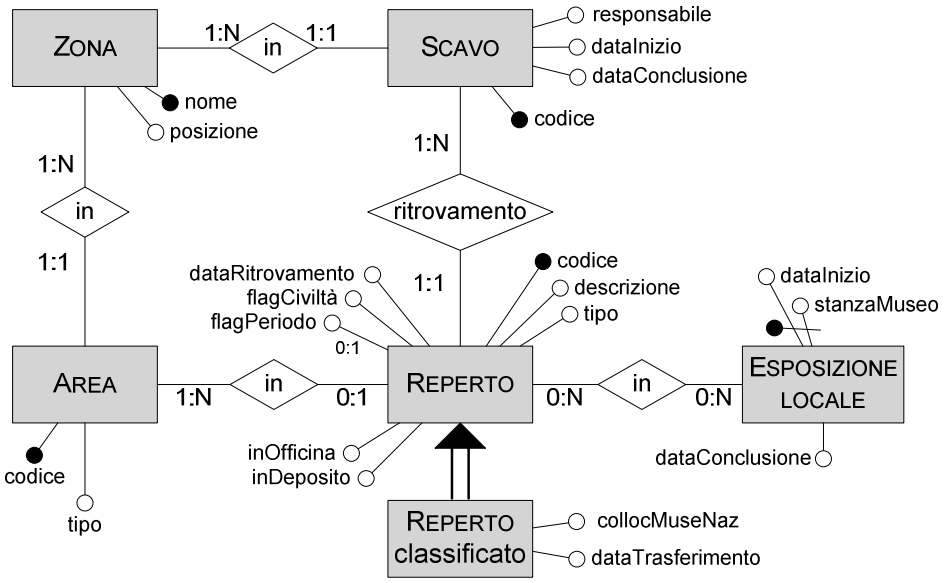
2. Aggiungere alla tabella CONCORRENTE l'attributo Indirizzo.

```
alter table CONCORRENTE add column Indirizzo varchar(255)
```

D. PROGETTO CONCETTUALE E LOGICO

Una missione archeologica mantiene un database delle attività di scavo. Gli scavi avvengono in zone geografiche che hanno un nome, una posizione geografica e, per ogni attività di scavo relativa, la data di inizio e di conclusione dello scavo e il nominativo della persona responsabile dello scavo. Durante ogni scavo, vengono trovati reperti archeologici (anfore, vasi, gioielli, sculture, armi, ...) che vengono inizialmente attribuiti ad una civiltà e poi attribuiti ad un intervallo temporale. Ogni zona geografica comprende un certo numero di aree che vengono classificate come palazzi, abitazioni, templi, piazze, vie; ogni reperto archeologico può essere trovato all'interno di un'area.

I reperti, trovati in una certa data, vengono progressivamente acquisiti dalla missione e trasferiti nell'officina per il restauro, completato il quale vengono destinati al deposito collocato vicino allo scavo, e possono essere periodicamente esposti nel museo archeologico collocato vicino agli scavi; per ogni esposizione, è nota la data di inizio, la data di fine e la collocazione in una stanza del museo. Alcuni reperti sono definitivamente trasferiti al museo archeologico nazionale; in tal caso si conosce la loro collocazione in una sala del museo e la data di trasferimento al museo.



Si noti che il ciclo di associazioni non è ridondante, perché nessuna associazione è deducibile dalle altre.

Si noti anche che sussiste un vincolo sui valori degli attributi inOfficina e inDeposito, che non possono essere posti entrambi a true contemporaneamente (mentre possono essere entrambi false).

Allievi INF e TLC - Proff. S. Ceri e A. Campi
Primo Appello – 23 Febbraio 2005

UTENTE (NickName, pwdHash, Nome, DataNascita, Sesso, Nazione,
CittàResidenza, Foto, e-mail)
MESSAGGIO (ID, NickMitt, NickDest, Data, Ora, Testo)
CONTATTO (NickAccount, NickContatto, DataInserimento, Gruppo)

Lo schema si riferisce a un sistema centralizzato di messaggistica istantanea, in cui gli utenti registrati salvano sul server tutti i messaggi scambiati e la loro "contact-list", in cui i contatti di ogni account possono essere ripartiti in gruppi (il valore Null per l'attributo Gruppo indica un contatto non classificato).

A. DDL (1 punto)

Scrivere i comandi SQL per

1. Creare la tabella CONTATTO, effettuando opportune ragionevoli ipotesi su domini, vincoli e reazioni ai cambiamenti.

```
create table CONTATTO (
  NickAccount varchar(16) references UTENTE (NickName)
                on update cascade on delete cascade,
  NickContatto varchar(16) references UTENTE (NickName)
                on update no action on delete cascade,
  DataInserimento date,
  Gruppo varchar(16),
  primary key (NickAccount, NickContatto)  )
```

Al momento della cancellazione di un utente sono automaticamente rimosse la sua contact-list e il contatto relativo all'utente cancellato da tutte le contact-list degli altri utenti. Non è invece consentita la modifica del NickName di un utente, se questo compare nella contact-list di almeno un altro utente (che non lo riconoscerebbe più!).

B. LINGUAGGI FORMALI (7 punti)

1. Esprimere in Algebra Relazionale ottimizzata, Calcolo Relazionale e Datalog la seguente interrogazione: *trovare, per ogni utente, l'utente a cui ha inviato il messaggio più recente.*

Algebra relazionale

$$\Pi_{NickMitt,NickDest} (\quad (MESS - (MESS \rtimes_{NickMitt=NickMitt \wedge Data<Data} MESS)) - (MESS \rtimes_{NickMitt=NickMitt \wedge Data=Data \wedge Ora<Ora} MESS) \quad)$$

Calcolo relazionale

$$\{ t \mid \exists t_M \in MESSAGGIO \mid (t[NickMitt,NickDest] = t_M[NickMitt,NickDest] \wedge \neg (\exists t_{MSuc} \in MESSAGGIO \mid t_{MSuc}[NickMitt] = t_M[NickMitt] \wedge (t_{MSuc}[Data] > t_M[Data] \vee t_{MSuc}[Data] = t_M[Data] \wedge t_{MSuc}[Ora] > t_M[Ora]))) \}$$

Datalog

```
MessaggioNonUltimo ( Mes ) :-  MESSAGGIO ( Mes, Mit, _, D1, _, _ ),
                               MESSAGGIO ( _, Mit, _, D2, _, _ ),  D2 > D1

MessaggioNonUltimo ( Mes ) :-  MESSAGGIO ( Mes, Mit, _, D, H1, _ ),
                               MESSAGGIO ( _, Mit, _, D, H2, _ ),  H2 > H1

UltimoUtenteContattato ( Mit, Des ) :-  MESSAGGIO ( Mes, Mit, Des, _, _, _ ),
                                        ¬ MessaggioNonUltimo ( Mes )

? - UltimoUtenteContattato ( X, Y )
```

2. Esprimere in un linguaggio formale a scelta la seguente interrogazione: *trovare tutte le coppie di utenti della stessa città che non si sono mai scritti alcun messaggio, neanche monodirezionalmente [si cerchi di evitare che le coppie compaiano più volte nel risultato]*.

Algebra relazionale

$$\Pi_{Nick,Nick2} (UTENTE \bowtie_{\mathbf{Nick<Nick2} \wedge Nazione=Nazione \wedge CittàRes=CittàRes} (\rho_{Nick2 \leftarrow Nick} UTENTE))$$

$$-$$

$$\Pi_{Nick,Nick2} ((\rho_{Nick,Nick2 \leftarrow NickDest,NickMitt} MESS) \cup (\rho_{Nick,Nick2 \leftarrow NickMitt,NickDest} MESSAGGIO))$$

Si noti che la garanzia che le stesse coppie non compaiano più volte nel risultato è data dal fatto che ogni coppia di NickName (che sono chiave per la relazione UTENTE) è inclusa solo se il primo nick è lessicograficamente precedente al

secondo (Nick<Nick2), escludendo la possibilità di includere la stessa coppia in ordine inverso.
Si noti anche che per limitare gli errori di valutazione della "conterraneità" uguagliamo anche la nazione, non solo il nome della città (ad esempio ci sono città di nome "Florence" in Italia, Alabama, Arizona, California, Colorado, Illinois, Kansas, Kentucky, Massachusetts, Minnesota, Mississippi, Missouri, Montana, New York, Oregon, South Carolina, South Dakota, Texas, ...).
Si noti infine che la ridenominazione è formalmente necessaria per estrarre le coppie, quindi gli altri due linguaggi erano probabilmente da preferire, sfruttando il grado di libertà lasciato dalla traccia.

Calcolo relazionale

$$
\begin{aligned}
&\{ t \mid \exists\, t_{U1} \in \text{UTENTE}, \exists\, t_{U2} \in \text{UTENTE}, \mid \\
&\quad (t[\text{Nick1}] = t_{U1}[\text{NickName}] \wedge \\
&\quad\ t[\text{Nick2}] = t_{U2}[\text{NickName}] \wedge \\
&\quad\ t_{U1}[\text{CittàResidenza}] = t_{U2}[\text{CittàResidenza}] \wedge \\
&\quad\ t_{U1}[\text{Nazione}] = t_{U2}[\text{Nazione}] \wedge \\
&\quad\ \mathbf{t_{U1}[NickName] > t_{U2}[NickName]} \wedge \\
&\quad\ \neg (\exists\, t_M \in \text{MESSAGGIO} \mid \\
&\qquad\ t_M[\text{NickMitt}] = t_{U1}[\text{NickMitt}] \wedge t_M[\text{NickMitt}] = t_{U2}[\text{NickDest}] \\
&\qquad\ \vee \\
&\qquad\ t_M[\text{NickMitt}] = t_{U2}[\text{NickMitt}] \wedge t_M[\text{NickMitt}] = t_{U1}[\text{NickDest}] \ ) \\
&\quad) \\
&\}
\end{aligned}
$$

Datalog

```
ScambiatisiMessaggi ( Ut1, Ut2 )  :-  MESSAGGIO ( _, Ut1, Ut2, _, _, _ )

ScambiatisiMessaggi ( Ut1, Ut2 )  :-  MESSAGGIO ( _, Ut2, Ut1, _, _, _ )

Conterranei ( Ut1, Ut2 ) :-  UTENTE  ( Ut1, _, _, _, _, Naz, Cit, _, _ ),
                             UTENTE  ( Ut2, _, _, _, _, Naz, Cit, _, _ ),  Ut1 > Ut2

ConterraneiNonComunicanti ( Ut1, Ut2 ) :-  Conterranei ( Ut1, Ut2 ),
                                           ¬ ScambiatisiMessaggi ( Ut1, Ut2 )

? - ConterraneiNonComunicanti ( A, B )
```

Si noti che la condizione di ordine lessicografico può essere imposta nella regola "Conterranei", poiché la residenza nella stessa città è una "relazione simmetrica", ma non nella regola "ScambiatisiMessaggi", poiché si potrebbero perdere coppie di utenti che si sono scambiati un solo messaggio.

3. Esprimere in datalog la seguente interrogazione: *dire se l'utente Alice può, in linea di principio, recapitare un messaggio all'utente Bob, anche indirettamente (cioè ricorrendo eventualmente agli "amici degli amici degli amici...").*

Con una regola ricorsiva decidiamo se Bob è nella contact-list di Alice (recapito diretto), o se il recapito può avvenire (ricorsivamente) in modo indiretto. Otteniamo una risposta sì/no vincolando tutte le variabili della regola con dei parametri (**no**: la query restituisce l'insieme vuoto, **sì**: restituisce ["Alice", "Bob"]).

PuòRecapitare (A, B) :- CONTATTO (A, B, _, _)

PuòRecapitare (A, B) :- CONTATTO (A, X, _, _),
PuòRecapitare (X, B)

? - PuòRecapitare ("Alice", "Bob")

C. Interrogazioni in SQL (9 punti)

1. *Trovare gli utenti che non hanno ricevuto alcun messaggio nel giorno del loro compleanno del 2004 [si usino ad esempio le funzioni di sistema d.year(), d.month(), d.day() - che restituiscono un intero - dove d rappresenta un attributo di tipo date].*

```
select NickName
from UTENTE U
where not exists
   ( select *
     from MESSAGGIO M
     where U.NickName = M.NickDest and
           U.DataNascita.month() = M.Data.month() and
           U.DataNascita.day() = M.Data.day() and
           M.data.year() = 2004)
```

2. *Trovare, per ogni giorno del mese di gennaio 2005, la città i cui residenti hanno inviato il maggior numero di messaggi.*

Possiamo effettuare il join tra la tabella dei messaggi di gennaio '05 e gli utenti-mittenti, raggruppare tale tabella per giorno e per città, e successivamente estrarre il massimo di ogni giorno:

```
create view MessCittGen05 (Città, GiornoC, NumInvii) as
select CittàResidenza, Data, count(*)
from MESSAGGIO join UTENTE on NickMitt = NickName
where Data between 1/1/05 and 31/1/05
group by CittàResidenza, Data
```

```
create view MaxMessGen05 (GiornoM, MaxInvii) as
select GiornoC, max(NumInvii)
from MessCittGen05
group by GiornoC

select Giorno, Città
from MessCittGen05 join MaxMessGen05 on GiornoC = GiornoM
where NumInvii = MaxInvii
```

N.B.: in caso di "parimerito" estrae tutte le città con lo stesso numero di messaggi.

3. *Trovare gli utenti (qualora ce ne fossero) che hanno inviato lo stesso numero di messaggi a tutti gli utenti presenti nella loro contact list.*

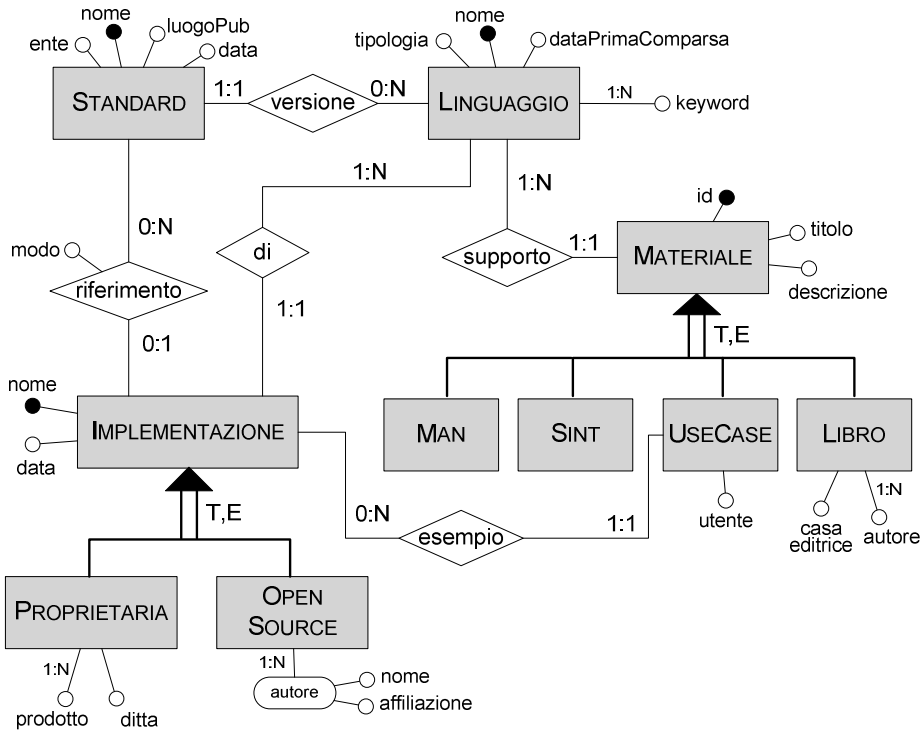
Ad esempio si può preparare una view che conta i messaggi inviati da ogni utente a ogni altro, e selezionare gli utenti che hanno il massimo pari al minimo nella view:

```
create view ContaInvii (Sender, Receiver, NumMess) as
select NickMitt, NickDest, count(*)
from MESSAGGIO U
group by NickMitt, NickDest

select Sender
from ContaInvii
group by Sender
having min(NumMess) = max(numMess)
```

D. PROGETTO CONCETTUALE E LOGICO

Si vuole raccogliere un database relativo ai linguaggi di programmazione. Ciascun linguaggio ha un nome, una data di prima comparsa, una tipologia e un insieme di parole riservate. Alcuni linguaggi hanno varie successive versioni standard; ciascuno standard è caratterizzato da un nome, da un ente di standardizzazione, da una data e da un luogo di pubblicazione. Inoltre, ogni linguaggio di programmazione è associato a vari materiali di riferimento, che comprendono manuali, definizioni della sintassi, "casi d'uso", libri di tipo didattico. I "casi d'uso" coinvolgono un utente di cui è noto il nome. I libri sono caratterizzati da uno o più autori e dalla casa editrice del libro.

Esistono poi varie implementazioni di ciascun linguaggio, che possono essere open source o proprietarie; per le implementazioni open source è noto il nome e l'affiliazione degli autori, per le implementazioni proprietarie è nota la ditta di software autrice della implementazione e la disponibilità di (vari) prodotti sul mercato. Ogni implementazione è poi descritta dalla sua data di realizzazzione ed associata a "casi d'uso". Le implementazioni possono fare riferimento (completamente oppure parzialmente) alle versioni standard del linguaggio.


nome
ente
luogoPub
data
STANDARD
1:1
versione
0:N
nome
tipologia
dataPrimaComparsa
LINGUAGGIO
1:N
keyword
1:N
1:N
0:N
modo
riferimento
0:1
di
1:1
supporto
1:1
id
MATERIALE
titolo
descrizione
T,E
nome
data
IMPLEMENTAZIONE
MAN
SINT
USECASE
LIBRO
utente
casa editrice
1:N
autore
T,E
0:N
esempio
1:1
PROPRIETARIA
OPEN SOURCE
1:N
prodotto
ditta
1:N
autore
nome
affiliazione

Allievi INF e TLC - Proff. A. Campi, S. Ceri, G. Pozzi
Secondo Appello – 7 Luglio 2005

Si considerino le tabelle seguenti, che rappresentano i dati relativi alle proprietà immobiliari gestite da uno studio d'amministrazione di più condomini.

CONDOMINIO (Codice, Via, Civico, Città)

APPARTAMENTO (Condominio, Numero, Scala, Piano, CodFiscProprietario, Millesimi)

PROPRIETARIO (CodiceFiscale, Nome, Cognome, DataNascita)

A. DDL (2 punti)

Scrivere i comandi SQL per

1. Creare le tabelle CONDOMINIO e APPARTAMENTO, effettuando opportune e ragionevoli ipotesi su domini, vincoli e reazioni ai cambiamenti. In particolare si consideri il vincolo per cui i millesimi sono esattamente mille per ogni condominio.

```
create table CONDOMINIO (
  Codice    varchar(16) primary key,
  Via       varchar(64),
  Civico    integer > 0,
  Città     varchar(64)  )

create table APPARTAMENTO (
  Condominio  varchar(16) references CONDOMINIO(Codice)
                  on delete no action on update cascade,
  Numero       integer > 0,
  Scala        varchar(4),
  Piano        integer,
  CodFiscProprietario char(16)
              references PROPRIETARIO(CodiceFiscale)
                 on delete set null          l'appartamento esiste
                 on update set null,         indipendentemente!!
  Millesimi numeric(6,2) check
     ( Millesimi > 0 and
       1000,00 = ( select sum(Millesimi)
                   from APPARTAMENTO A
                   where A.Condominio=Condominio) ),
  primary key (Condominio, Numero)  )
```

B. SQL (8 punti)

1. *Creare una view che mostri di ogni condominio il proprietario con più millesimi (si tenga presente che uno stesso proprietario può possedere più appartamenti in uno stesso condominio).*

```
create view PropDominante(CodCondominio, CodFisc,
                          Nome, Cognome) as
select Condominio, CodFiscProprietario, Nome, Cognome
from  APPARTAMENTO A  join PROPRIETARIO
                      on CodFiscProprietario = CodiceFiscale
group by Condominio, CodFiscProprietario, Nome, Cognome
having sum(Millesimi) >= all
                  ( select sum(Millesimi)
                    from  APP A1
                    where A1.Condominio = A.Condominio
                    group by CodFiscProprietario )
```

N.B.: l'aggiunta di Nome e Cognome alla target list (e quindi del join con la tabella PROPRIETARIO) è ortogonale alla soluzione, poiché non cambia la struttura del raggruppamento (il codice fiscale implica univocamente il nome del proprietario).

2. *Estrarre la lista di tutti i proprietari di appartamenti nel condominio in Via Rubattino 8 a Milano (che si può supporre essere unico) ordinata per totale di millesimi posseduti (si tenga presente che uno stesso proprietario può possedere più appartamenti in un condominio).*

```
select CodFiscProprietario, Cognome, Nome, sum(Millesimi)
from  (APPARTAMENTO join PROPRIETARIO
       on CodFiscProprietario =  CodiceFiscale)
        join CONDOMINIO on Condominio = Codice
where Via = 'Rubattino' and Civico = 8 and
      Città = 'Milano'
group by CodFiscProprietario, Nome, Cognome
order by 4 desc, 2
```

Si noti che, a parità di millesimi, i proprietari sono ordinati per cognome.

3. *Estrarre il codice fiscale di tutti coloro che posseggono in totale almeno sette appartamenti, distribuiti in almeno tre condomini distinti.*

```
select CodFiscProprietario
from APPARTAMENTO
group by CodFiscProprietario
having count(*) > 6 and
       count(distinct Condominio) > 2
```

C. LINGUAGGI FORMALI (6 punti)

1. Esprimere in Algebra ottimizzata, Calcolo e Datalog l'interrogazione che estrae nome e cognome del proprietario dell'appartamento più grande del condominio in Via Rubattino 8 a Milano (che si può supporre essere unico).

N.B. Intendiamo con "più grande" l'appartamento col più alto valore in millesimi.

Algebra relazionale

$$\Pi_{\text{Nome, Cognome}} (\Pi_{\text{Nome,Cognome,CodiceFiscale}} \text{PRO} \ltimes_{\text{CodiceFiscale=CodFiscProp}} ((\text{APP} \ltimes_{\text{Cond=Cod}} (\sigma_{\text{Via='Rubattino'} \wedge \text{Città='Milano'} \wedge \text{Civ=8}} \text{CON})) - ((\text{APP} \ltimes_{\text{Cond=Cod}} (\sigma_{\text{Via='Rubattino'} \wedge \text{Città='Milano'} \wedge \text{Civ=8}} \text{CON})) \ltimes_{\text{Millesimi<Millesimi}} (\text{APP} \ltimes_{\text{Cond=Cod}} (\sigma_{\text{Via='Rubattino'} \wedge \text{Città='Milano'} \wedge \text{Civ=8}} \text{CON})))))$$

Oppure, più leggibile, usando un assegnamento alla relazione virtuale APP_RUB:

$$\text{APP\_RUB} = \Pi_{\text{Cond,Numero,CodFiscProp}} (\text{APP} \ltimes_{\text{Condominio=Codice}} (\sigma_{\text{Via='Rubattino'} \wedge \text{Città='Milano'} \wedge \text{Civico=8}} \text{CON}))$$

In questo modo la soluzione diventa:

$$\Pi_{\text{Nome, Cognome}} (\Pi_{\text{Nome,Cognome,CodiceFiscale}} \text{PRO} \ltimes_{\text{CodiceFiscale=CodFiscProp}} (\text{APP\_RUB} - (\text{APP\_RUB} \rhd_{\text{Millesimi<Millesimi}} \text{APP\_RUB})))$$

Calcolo relazionale

$$\{ t \mid \exists t_P \in \text{PROPRIETARIO}, \exists t_A \in \text{APPARTAMENTO}, \exists t_C \in \text{CONDOMINIO} (t[\text{CodFisc, Nome, Cognome}] = t_P[\text{CodFisc, Nome, Cognome}] \wedge t_A[\text{Condominio}] = t_C[\text{Codice}] \wedge t_A[\text{CodFiscProprietario}] = t_P[\text{CodiceFiscale}] \wedge t_C[\text{Via}] = \text{'Rubattino'} \wedge t_C[\text{Civico}] = 8 \wedge t_C[\text{Città}] = \text{'Milano'} \wedge \neg (\exists t_{A1} \in \text{APPARTAMENTO} \mid t_{A1}[\text{Condominio}] = \mathbf{t_A}[\text{Condominio}] \wedge \quad \textit{stesso condominio} \quad t_{A1}[\text{Millesimi}] > \mathbf{t_A}[\text{Millesimi}]) \quad \textit{dimensione maggiore}) \}$$

Datalog

```
AppRubattinoPiccolo(Co, Num) :- CONDOMINIO ( Co, 'Rubattino', 8, 'Milano' ),
                                APPARTAMENTO ( Co, Num, _, _, _, M1 ),
                                APPARTAMENTO ( Co, _, _, _, _, M2 ),
                                M1 < M2

AppRubattinoGrande(Co, Num) :- CONDOMINIO ( Co, 'Rubattino', 8, 'Milano' )
                               APPARTAMENTO ( Co, Num, _, _, _, _ ),
                               ¬ AppRubattinoPiccolo ( Co, Num )

ProprietarioMaxRub ( CF, N, C ) :- PROPRIETARIO ( CF, N, C, _ ),
                                   APPARTAMENTO ( Cond, Num, _, _, CF, _ ),
                                   AppRubattinoGrande ( Cond, Num )

? - ProprietarioMaxRub ( X, Y, Z )
```

2. Esprimere in Datalog l'interrogazione che estrae tutti i proprietari di uno e un solo appartamento.

Basta escludere dal risultato i proprietari di due o più appartamenti, che a loro volta sono definiti come tutti i proprietari di due appartamenti diversi (cioè appartamenti di condomini diversi **oppure** caratterizzati da numero diverso)

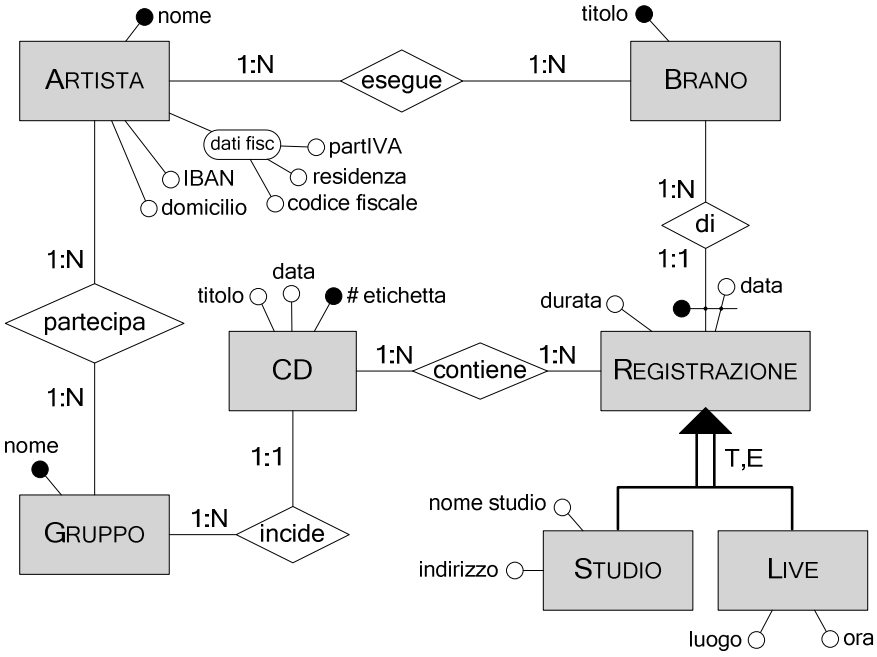
```
ProprietarioDueAppart ( CF ) :- PROPRIETARIO ( CF, _, _, _ ),
                                APPARTAMENTO ( Cond1, _, _, _, CF, _ ),
                                APPARTAMENTO ( Cond2, _, _, _, CF, _ ),
                                Cond1 <> Cond2

ProprietarioDueAppart ( CF ) :- PROPRIETARIO ( CF, _, _, _ ),
                                APPARTAMENTO ( _, Num1, _, _, CF, _ ),
                                APPARTAMENTO ( _, Num2, _, _, CF, _ ),
                                Num1 <> Num2

PropUnSoloAppartamento ( CF, N, C ) :- PROPRIETARIO ( CF, N, C, _ ),
                                       ¬ ProprietarioDueAppart ( CF )

? - PropUnSoloAppartamento ( X, Y, Z )
```

D. PROGETTO CONCETTUALE E LOGICO

Una casa discografica mantiene traccia delle attività di registrazione dei vari artisti, che portano alla preparazione dei CD. I CD sono incisi da gruppi, caratterizzati da un nome d'arte; il caso di CD incisi da singoli artisti viene

rappresentato creando nel DB un gruppo denominato esattamente col nome dell'artista. Ogni artista ha un nome, un domicilio, dati fiscali e coordinate bancarie; lo stesso artista può appartenere a più gruppi. Ogni CD ha un titolo, una data di pubblicazione e un numero di etichetta.

Il CD si compone di vari brani, cui partecipano alcuni degli artisti del gruppo (non necessariamente tutti); ciascun brano ha un titolo e una durata. Ciascun brano è inciso o in uno studio oppure dal vivo. Nel caso di registrazione in uno studio, si indicano la data di incisione e il nome e l'indirizzo dello studio. Nel caso di registrazione dal vivo, la data, il luogo e l'ora del concerto. Di alcuni brani, esistono varie registrazioni alternative, sempre con gli stessi artisti coinvolti.



Il testo presenta una certa ambiguità tra i concetti di *brano* e di *registrazione* di un brano. Poiché un brano può avere più registrazioni, ed evidentemente di norma solo una di tali registrazioni entra in un particolare CD, associamo i CD alle registrazioni e no ai brani (è poi ovviamente possibile risalire al brano tramite l'associazione di, che ha cardinalità 1:1).

In questo modo, tra l'altro, eventualmente più di una versione dello stesso brano (live o studio) possono entrare nello stesso CD.

Per le stesse ragioni consideriamo la durata un attributo della registrazione, perché diverse registrazioni dello stesso brano possono avere lunghezze diverse, seppure leggermente.

Allievi INF e TLC - Proff. A. Campi, S. Ceri, G. Pozzi
Terzo Appello – 9 Settembre 2005

Si consideri il seguente schema, relativo a un campionato mondiale di pallavolo:

GIOCATORE (NumTessera, Nome, Squadra, Altezza, DataNascita, PresenzeInNazionale)

SQUADRA (Nazione, Allenatore, NumPartiteVinte)

PARTITA (IdPartita, Data, Squadra1, Squadra2, SetVintiSquadra1, SetVintiSquadra2, Arbitro)

PARTECIPAZIONE (IdPartita, TesseraGiocatore, Ruolo, PuntiRealizzati)

A. DDL (2 punti)

1. Scrivere i comandi SQL per creare le tabelle PARTECIPAZIONE e PARTITA, effettuando opportune e ragionevoli ipotesi su domini, vincoli e reazioni ai cambiamenti.

```
create table PARTITA (
  IdPartita integer primary key,
  Data      date not null,
  Squadra1  varchar(25) references SQUADRA(Nazione)
               on delete no action on update cascade,
  Squadra2  varchar(25) references SQUADRA(Nazione)
               on delete no action on update cascade,
  SetVintiSquadra1 integer,
  SetVintiSquadra2 integer check
     ( SetVintiSq1=3 and SetVintiSq2 between 0 and 2
             or
       SetVintiSq2=3 and SetVintiSq1 between 0 and 2 ),
  Arbitro   varchar(50) )

create table PARTECIPAZIONE (
  IdPartita  integer references PARTITA(IdPartita)
                on delete no action on update cascade,
  TesseraGiocatore integer
         references GIOCATORE(NumTessera)
                on delete no action on update cascade,
  Ruolo  varchar(15) not null,
  PuntiRealizzati  integer >= 0,
  primary key (IdPartita, TesseraGiocatore)  )
```

B. Linguaggi Formali (7 punti)

1. Esprimere in Algebra Relazionale ottimizzata, Calcolo Relazionale e Datalog l'interrogazione che trova il nome del giocatore *più alto* tra quelli nati dopo il 9/9/85.

Algebra relazionale

$$\Pi_{\text{NumTessera, Nome}} (\quad (\Pi_{\text{NumTessera,Nome,Altezza}} \sigma_{\text{DataNascita}>9/9/1985} \text{GIOCATORE}) - (\quad (\Pi_{\text{NumTessera,Nome,Altezza}} \sigma_{\text{DataNascita}>9/9/1985} \text{GIOCATORE}) \ltimes_{\text{Altezza}<\text{Altezza}} (\Pi_{\text{NumTessera,Nome,Altezza}} \sigma_{\text{DataNascita}>9/9/1985} \text{GIOCATORE}) \quad) \quad)$$

Oppure, usando un assegnamento a una relazione virtuale:

$$\text{Giovane} = \Pi_{\text{NumTessera,Nome,Altezza}} \sigma_{\text{DataNascita}>9/9/1985} \text{GIOCATORE}$$

$$\Pi_{\text{NumTessera, Nome}} (\quad \text{Giovane} - (\quad \text{Giovane} \ltimes_{\text{Altezza}<\text{Altezza}} \text{Giovane} \ ) \quad)$$

Calcolo relazionale

$$\{ t \mid \exists t_G \in \text{GIOCATORE} \mid (t[\text{NumTessera, Nome}] = t_G[\text{NumTessera, Nome}] \wedge t_G[\text{DataNascita}] > 9/9/1985 \wedge \neg (\exists t_{G1} \in \text{GIOCATORE} \mid t_{G1}[\text{Altezza}] > \mathbf{t_G}[\text{Altezza}] \wedge t_{G1}[\text{DataNascita}] > 9/9/1985)) \}$$

Datalog

```
Giovane ( Tes, Nom, Alt ) :- GIOCATORE ( Tes, Nom, _, Alt, DataN, _ ),
                             DataN > 9/9/85

GiovaneBasso ( Tes ) :- Giovane ( Tes, _, A1 ), Giovane ( _, _, A2), A1 < A2

GiovaneAlto ( T, N ) :- Giovane ( T, N, _ ), ¬ GiovaneBasso ( T )

? - GiovaneAlto ( X, Y )
```

2. Esprimere in **due** linguaggi formali a scelta l'interrogazione che trova i nomi dei giocatori che hanno giocato *sempre* nello stesso ruolo (estraendo anche tale ruolo nel risultato).

Diamo comunque la soluzione in tutti e tre i linguaggi:

Algebra relazionale

$$\Pi_{NumTes,Nome,Ruolo}(\text{GIOCATORE} \bowtie_{NumTes=TesGioc} ((\Pi_{TesGioc,Ruolo} \text{PARTEC}) - (\Pi_{TesGioc,Ruolo} (\text{PARTEC} \rtimes_{TesGioc=TesGioc \wedge Ruolo<>Ruolo} \text{PARTEC}))))$$

Si noti che le coppie (TesGioc, Ruolo) dei giocatori a ruolo non unico sono eliminate tutte, poiché <> è una relazione simmetrica.

Calcolo relazionale

$$\{ t \mid \exists t_G \in \text{GIOCATORE} \mid \exists t_P \in \text{PARTECIPAZIONE} (t[\text{NumTessera, Nome}] = t_G[\text{NumTessera, Nome}] \wedge t[\text{Ruolo}] = t_P[\text{Ruolo}] \wedge t_G[\text{NumTessera}] = t_P[\text{TesseraGiocatore}] \wedge \neg (\exists t_{P1} \in \text{PARTECIPAZIONE} \mid t_{P1}[\text{TesseraGiocatore}] = \mathbf{t_G}[\text{NumTessera}] \wedge t_{P1}[\text{Ruolo}] <> \mathbf{t_P}[\text{Ruolo}])) \}$$

stesso giocatore

ruolo diverso

Datalog

```
GiocaDueRuoli ( Tes ) :- PARTECIPAZIONE ( _, Tes, R1, _ ),
                         PARTECIPAZIONE ( _, Tes, R2, _ ),
                         R1 <> R2

GiocaUnRuoloSolo ( T, N, R ) :- PARTECIPAZIONE ( _, T, R, _ ),
                                ¬ GiocaDueRuoli ( T ),
                                GIOCATORE ( T, N, _, _, _, _ )

? - GiocaUnRuoloSolo ( Tessera, Nome, Ruolo )
```

C. Interrogazioni in SQL (7 punti)

1. *Estrarre i giocatori che hanno giocato in tutte le partite della loro squadra.*

```
select *
from GIOCATORE G
where not exists (
     select *
     from PARTITA P
     where G.Squadra = Squadra1 or
           G.Squadra = Squadra2 and not exists (
                 select *
                 from PARTECIPAZIONE
                 where IdPartita = P.IdPartita and
                       TesseraGiocatore = G.NumTessera ) )
```

Confrontare direttamente il valore di PresenzeInNazionale col numero di partite giocate dalla sua squadra significa fare l'ipotesi che tutte le presenze del giocatore siano relative all'attuale campionato europeo. A rigore, lo schema non offre giustificazioni per questa interpretazione (anzi, sarebbe uno schema ridondante, poiché conterrebbe un dato derivabile), a meno che i giocatori non siano tutti esordienti. Nel caso, comunque, si potrebbe scrivere:

```
select *
from GIOCATORE G
where PresenzeInNazionale = ( select count(*)
                              from PARTITA
                              where G.Squadra = Squadra1 or
                                    G.Squadra = Squadra2 )
```

2. *Estrarre il numero totale di set vinti da ogni squadra.*

```
create view FlattenSetVinti (Squadra, SetVinti) as
( select Squadra1, SetVintiSquadra1 from PARTITA
      union all
  select Squadra2, SetVintiSquadra2 from PARTITA )

select Squadra, sum(SetVinti)
from FlattenSetVinti
group by Squadra
```

3. *Sapendo che una partita è vinta quando il numero di set vinti è pari a 3, scrivere un comando SQL che assegna l'attributo NumPartiteVinte della tabella SQUADRA, in base ai dati contenuti nella tabella Partita.*

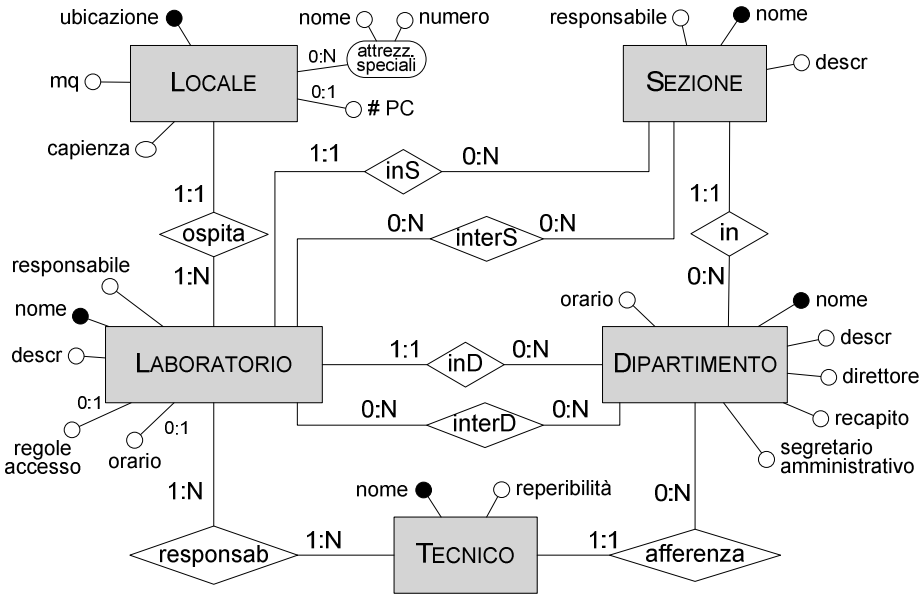
```
update SQUADRA S
set NumPartiteVinte = ( select count(*)
                        from FlattenSetVinti
                        where Squadra = S.Nazione and
                              SetVinti = 3 )
```

dove la vista FlattenSetVinti è definita nella soluzione dell'esercizio precedente.

D. PROGETTO CONCETTUALE E LOGICO

Il Politecnico costituisce l'archivio dei suoi laboratori. Ogni laboratorio si colloca all'interno di un Dipartimento e talvolta all'interno di una Sezione del Dipartimento (ma non tutti i Dipartimenti sono divisi in Sezioni). Ogni Dipartimento ha un nome, una descrizione, un direttore, un segretario amministrativo, un recapito principale. Ogni Sezione ha un nome, una descrizione, un responsabile. I Laboratori possono essere InterDipartimentali (fare riferimento, cioè, a più Dipartimenti) oppure InterSezionali (fare riferimento a più Sezioni). Ogni laboratorio ha un nome, una descrizione e un responsabile.

I laboratori sono collocati in uno o più locali, di cui è nota l'ubicazione, la capienza massima (in termini di numero di persone che possono essere contemporaneamente presenti) e la metratura. Taluni locali hanno caratteristiche particolari: hanno un certo numero di attrezzature speciali (ad esempio: colonne di distillazione, stazioni di montaggio dei robot) e sono dotati di un certo numero di computer fissi. Ogni laboratorio è assegnato ad uno o più tecnici, e ciascun tecnico può operare su più laboratori; i tecnici hanno un nome, un dipartimento di appartenenza e un orario settimanale di reperibilità. Alcuni laboratori hanno un orario di apertura agli studenti e hanno regole di accesso particolari, mentre altri laboratori si adeguano all'orario di apertura dell'edificio che li ospita.



È ragionevole che i laboratori che occupano più locali siano comunque collocati in locali siti tutti in uno stesso edificio; questo vincolo riguarda i valori, e non è esprimibile nel modello ER. Il termine "edificio" dell'ultima frase è stato inteso come sinonimo di "dipartimento", poiché si dice che ogni laboratorio si colloca in un dipartimento. Si è quindi specificato un attributo orario anche sui dipartimenti.

Allievi INF –Prof. S. Ceri
Pre-Appello – 10 Gennaio 2006

Il seguente schema è relativo a un'agenzia immobiliare che gestisce contratti di affitto. I clienti possono essere proprietari e/o affittuari di più immobili contemporaneamente, ma ogni immobile in ogni momento ha *un solo proprietario* e *al massimo un inquilino* (intestatario del contratto), se non è sfitto.

CLIENTE (CodiceFiscale, Nome, Cognome, Nazionalità, Telefono, e-mail)

IMMOBILE (CodImmobile, NumeroDiInterno, MetriQuadri, Locali, Piano, Via, NumCivico, Città)

CONTRATTO (CodImmobile, DataInizio, DataFine, CFProprietario, CFAffittuario, AffittoMensile, PercentualeProvvigione)

A. SQL (11 punti)

1. *Trovare l'immobile col più alto rapporto tra affitto e metratura*.

a) Considerando solo i contratti attivi al momento dell'interrogazione:

```
select CodImmobile, AffittoMensile/MetriQuadri as Rapporto
from CONTRATTO C join IMMOBILE I on C.CodImmobile=I.CodImmobile
where sysdate() between C.DataInizio and C.DataFine and
      AffittoMensile/MetriQuadri =
        ( select max (AffittoMensile/MetriQuadri)
          from CONTRATTO C join IMMOBILE I on C.CodImm=I.CodImm )
```

È ragionevole ipotizzare che le date di inizio e di fine dei contratti siano sempre note, anche se future, poiché normalmente i contratti contengono una data di scadenza (così come si può ipotizzare che in alcuni casi possano essere Null).

b) Considerando invece il massimo "storico" relativo a ogni immobile (uguale, ma senza sysdate()):

```
select CodImmobile,
       AffittoMensile/MetriQuadri as Rapporto
from CONTRATTO C join IMMOBILE I on C.CodImmobile=I.CodImmobile
where AffittoMensile/MetriQuadri =
        ( select max (AffittoMensile/MetriQuadri)
          from CONTRATTO C join IMMOBILE I on C.CodImm=I.CodImm )
```

Oppure con una view:

```
create view Rapporti( Imm, Rapp ) as
select CodImmobile, AffittoMensile/MetriQuadri
from IMMOBILE I join CONTRATTO C
       on C.CodImmobile = I.CodImmobile

select Imm
from Rapporti
where Rapp = ( select max(Rapp)
               from Rapporti )
```

2. *Estrarre con due viste distinte (a) i clienti che siano oggi proprietari di un immobile in cui in precedenza hanno abitato in affitto e (b) i clienti che abitano oggi in affitto in un immobile di cui in precedenza sono stati proprietari.*

```
create view PrimaPropPoiAff (CF, N, C) as
select CodiceFiscale, Nome, Cognome
from CLIENTE
where CodiceFiscale in
  ( select CP.CFProprietario
    from CONTRATTO CP join CONTRATTO CA
         on CP.CodImmobile = CA.CodImmobile
    where sysdate() between CA.DataInizio
                        and CA.DataFine and
          CP.CFProprietario = CA.CFAffittuario and
          CP.DataFine < CA.DataInizio )

create view PrimaAffPoiProp …
```

…poi ***UGUALE***

ma con `sysdate() between CP.DataInizio and CP.DataFine`

e con `CA.DataFine < CP.DataInizio`

3. *Scrivere un comando SQL che, per tutti i contratti relativi a Milano, aumenta del 2% l'affitto mensile e di un punto percentuale la provvigione trattenuta a beneficio dell'agenzia.*

```
update Contratto
set PercentualeProvvigione = PercentualeProvvigione+1,
    AffittoMensile = AffittoMensile * 1.02
where sysdate() between DataInizio and DataFine and
      CodImmobile in ( select CodImmobile
                       from Immobile
                       where Città = 'Milano' )
```

B. Linguaggi Formali (6 punti)

1. Esprimere in Algebra Relazionale ottimizzata, in Calcolo Relazionale **e** in Datalog l'interrogazione che trova gli immobili che non sono mai stati affittati a cittadini Italiani.

Algebra relazionale

$$(\Pi_{CodImm} \text{IMMOBILE}) \; - \;$$
$$(\Pi_{CodImm} \; (\; (\Pi_{CodImm,CFAffitt} \text{CONTR}) \rtimes_{CFAff=CodFis} (\Pi_{CoFis} \sigma_{Naz=\text{“Italiana”}} \text{CLIENTE}) \;) \;)$$

Calcolo relazionale

$$\{ t \mid \exists\, t_I \in \text{IMMOBILE} \mid$$
$$(t[\text{CodImmobile}] = t_I[\text{CodImmobile}] \wedge$$
$$\neg (\exists\, t_C \in \text{CONTRATTO}, \exists\, t_P \in \text{CLIENTE} \mid$$
$$t_C[\text{CodImmobile}] = \mathbf{t_I}[\text{CodImmobile}] \wedge$$
$$t_C[\text{CFAffittuario}] = t_P[\text{CodiceFiscale}] \wedge$$
$$t_P[\text{Nazionalità}] = \text{“Italiana”} \;) \;) \; \}$$

Datalog

```
AffittatoUnaVoltaItaliano ( Imm ) :-  CONTRATTO ( Imm, _, _, _, cfa, _, _ ),
                                      CLIENTE ( cfa, _, _, “Italiana”, _, _ )

MaiAffittatoIta ( CI ) :-  IMMOBILE ( CI, _, _, _, _, _, _, _, _ ),
                           ¬ AffittatoUnaVoltaItaliano ( CI )

? - MaiAffittatoIta ( X )
```

2. Esprimere in uno a scelta dei tre linguaggi formali l'interrogazione che estrae gli immobili che sono rimasti sfitti per tutto il 2005 (2 p.)

Ragionando in modo sistematico, possiamo dividere i contratti in quattro tipi:

1) finiscono prima del 1/1/05 ***oppure*** *iniziano dopo il 31/12/05 (cioè non hanno giorni del 2005);*

2) iniziano in un giorno del 2005;

3) finiscono in un giorno del 2005;

4) iniziano prima del 2005 ***e*** *finiscono dopo il 2005;*

Le classi 2) e 3) non sono disgiunte, ma non è un problema. Ci interessano gli immobili che *non* hanno *alcun* contratto delle tipologia 2, 3 oppure 4.

Datalog:

```
ConUnGiornoNel2005 ( CI )  :-  CONTRATTO ( CI, DataI, DataF, _, _, _, _ ),
                               DataI < 1/1/05,  DataF > 31/12/05

ConUnGiornoNel2005 ( CI )  :-  CONTRATTO ( CI, DataI, _, _, _, _, _ ),
                               DataI > 1/1/05,  DataI < 31/12/05

ConUnGiornoNel2005 ( CI )  :-  CONTRATTO ( CI, _, DataF, _, _, _, _ ),
                               DataF > 1/1/05,  DataF < 31/12/05

Sfitto2005 ( C )  :-  IMMOBILE ( C, _, _, _, _, _, _, _ ),
                      ¬ ConUnGiornoNel2005 ( C )
? - Sfitto2005 ( X )
```

In modo più brillante e compatto, sfruttando implicitamente l'ovvio vincolo di integrità che vuole che la data di fine di ogni contratto sia successiva alla sua data di inizio, si possono sostituire le tre regole precedenti con la seguente:

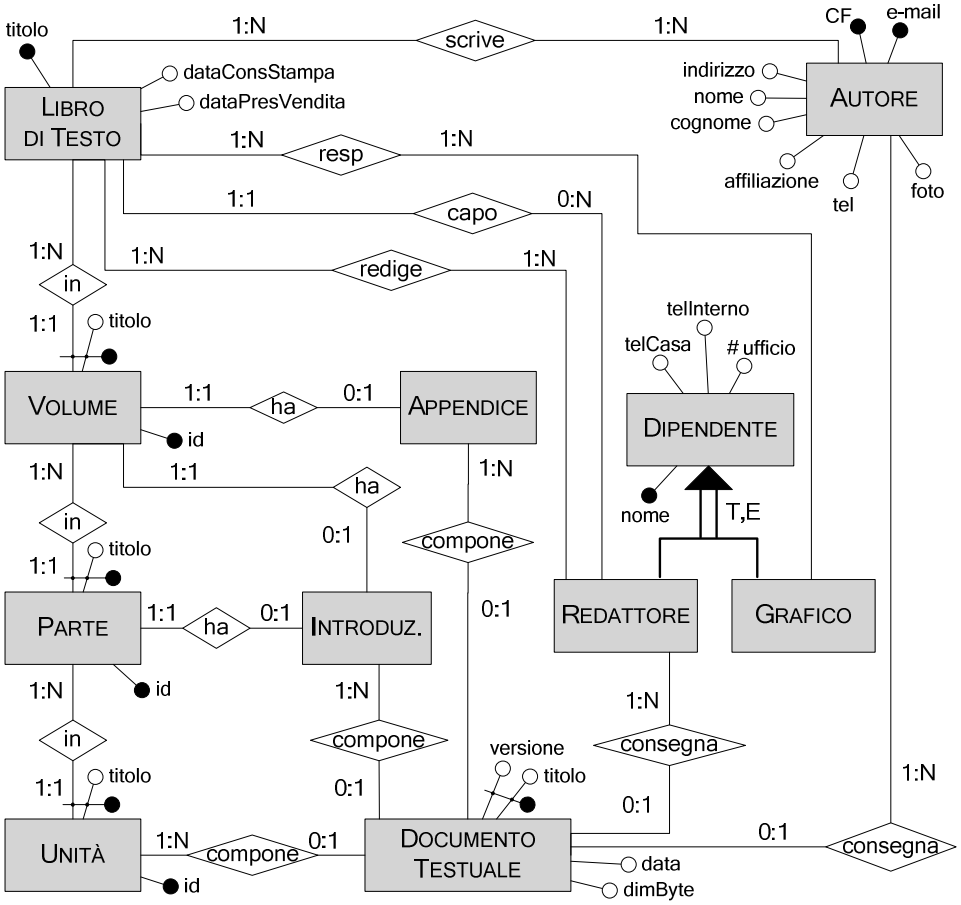
```
ConUnGiornoNel2005 ( CI )  :-  CONTRATTO ( CI, DI, DF, _, _, _, _ ),
                               DI <= 31/12/05,  DF >= 1/1/05
```

N.B.: il predicato della regola soprastante si può derivare negando la definizione del tipo di contratto 1) definito sopra, e applicando la relazione di DeMorgan.

D. PROGETTO CONCETTUALE E LOGICO

Una casa editrice deve gestire i materiali dei suoi autori e redattori per la costruzione di libri di testo. Ogni libro di testo è suddiviso in volumi, ogni volume in parti, ogni parte in unità. Le unità hanno un titolo e sono associate a vari documenti testuali, consegnati da un autore o da un redattore; ciascun documento ha un titolo, un numero di versione, una data, una dimensione in byte. Le parti hanno un titolo e una introduzione. I volumi hanno un titolo, una introduzione e varie appendici; le appendici hanno un titolo, e ciascuna introduzione e/o appendice è associata a vari documenti testuali.

Ogni libro ha vari autori, un capo-redattore, un responsabile grafico e vari altri redattori. I libri hanno una data di consegna agli stampatori e una successiva data di presentazione ai venditori. Gli autori hanno nome, codice fiscale, numero telefonico, indirizzo fisico e di e-mail, affiliazione, fotografia. Redattori e grafici hanno nome, numero di ufficio, numero telefonico interno, indirizzo e numero telefonico domestico. Gli autori sono responsabili dei "manoscritti", mentre i redattori sono responsabili dei documenti dalla fase di consegna alla fase di stampa, coprendo le prime, seconde e terze bozze.



La distinzione tra manoscritti e prime, seconde e terze bozze risulta modellata nei valori dell'attributo versione dei documenti testuali. In pratica la prima versione di ogni documento è il manoscritto dell'autore, mentre le versioni successive sono le bozze corrette e adattate dai redattori.

Allievi INF - Prof. S. Ceri
Primo Appello – 16 Febbraio 2006

Il seguente schema è relativo ai ricoveri effettuati da un ospedale. I pazienti sono identificati tramite il *N*umero di *T*essera *S*anitaria.
Ogni posto letto può divenire libero al mattino o alla sera di ogni giorno; un posto occupato diviene libero a partire dal mattino successivo se la dimissione avviene dopo le ore 12 di un qualsiasi giorno, e a partire dal pomeriggio dello stesso giorno se la dimissione avviene prima delle ore 12; per questo motivo la chiave della tabella PostiLetto contiene un flag che distingue i due momenti (assumendo i valori "M" e "S").

AMMISSIONE (NTS, Data, Ora, Nome, Sesso, Reparto)

DIMISSIONE (NTS, Data, Ora, Reparto)

POSTILETTO (Reparto, Data, MattSera, LiberiMaschi, LiberiFemmine)

A. SQL (11 punti)

1. *Esprimere tramite un'opportuna asserzione SQL il vincolo per cui non si possono ricoverare maschi nel reparto Ginecologia.*

```
create assertion Gineceo
check ( 0 = ( select count(*)
              from AMMISSIONE
              where Reparto = 'ginecologia' and
                    Sesso = 'M' ) )
```

o, equivalentemente:

```
create assertion AltroGineceo
check ( not exists ( select *
                     from AMMISSIONE
                     where Reparto = 'ginecologia' and
                           Sesso = 'M' ) )
```

2. *Calcolare il numero di pazienti attualmente ricoverati in ciascun reparto dell'ospedale (si noti che i pazienti ricoverati sono quelli ammessi e non ancora dimessi, e si badi a generare un risultato che abbia la struttura* [<*nome_reparto*>, <*numero_ricoveri*>])

```
select Reparto, count(*) as NumeroRicoveriAttuali
from AMMISSIONE A
where not exists ( select *
                   from DIMISSIONE
                   where NTS = A.NTS and
                         ( Data > A.Data or
                           Data = A.Data and Ora > A.Ora ) )
group by Reparto
```

3. *Trovare il paziente (o i pazienti) con la degenza più lunga, misurata in giorni (si trascurino cioè gli orari di ammissione e dimissione, e si ipotizzi che una operazione di differenza sulle date restituisca un numero intero che rappresenta il numero di giorni, anziché un valore di tipo interval).*

Costruiamo una view che ad ogni AMMISSIONE associa la relativa DIMISSIONE (la prima successiva in ordine di tempo):

```
create view Durata (NTS, Din, Dout) as
select A.NTS, A.Data, D.Data
from AMMISSIONE A join DIMISSIONE D on A.NTS = D.NTS
where A.Data < D.Data and
      not exists ( select *
                   from DIMISSIONE D2
                   where D2.NTS = A.NTS and
                         D2.Data between A.Data and D.Data )
```

Ora possiamo estrarre la degenza più lunga, calcolandone la durata in giorni:

```
select Nome, Din as DataIn, Dout-Din as DurataInGiorni
from AMMISSIONE join Durata on NTS = NTS
where Dout-Din >= all
               ( select Dout - Din
                 from AMMISSIONE join Durata on NTS = NTS )
```

Volendo considerare anche i minuti, possiamo fare l'ipotesi che una differenza di orari restituisca un numero intero di minuti (numeri anche negativi), anziché valori di tipo interval:

```
create view Durata (NTS, Di, Oi, Do, Oo) as
select A.NTS, A.Data, A.Ora, D.Data, D.Ora
from AMMISSIONE A join DIMISSIONE D on A.NTS = D.NTS
where A.Data < D.Data and
      not exists ( select * from DIMISSIONE D2
                   where D2.NTS = A.NTS and
                         D2.Data between A.Data and D.Data )
```

```
select Nome, Di as DataIn,
       ((Do-Di)*1440 + (Oo-Oi))/1440 As DurataInGiorni
from AMMISSIONE join Durata on NTS = NTS
where (Do – Di)*1440 + (Oo – Oi) >= all
                  ( select (Do – Di)*1440 + (Oo – Oi)
                    from AMMISSIONE join Durata on NTS = NTS )
```

Si noti che il contributo della differenza (Oo – Oi) può anche essere (giustamente) negativo.

Queste soluzioni comunque non considerano i pazienti attualmente ricoverati (cioè i pazienti non ancora dimessi). Per considerarli è sufficiente modificare la definizione della vista Durata (in *corsivo* le parti relative alla versione che considera i minuti):

```
create view InOut (NTS, Di, Oi, Do, Oo) as
select A.NTS, A.Data, A.Ora, D.Data, D.Ora
from AMMISSIONE A join DIMISSIONE D on A.NTS = D.NTS
where A.Data < D.Data and not exists (
         select * from DIMISSIONE D2
         where D2.NTS = A.NTS and
               D2.Data between A.Data and D.Data )

create view OnlyIn (NTS, Di, Oi, Do, Oo) as
select A.NTS, A.Data, A.Ora, SysDate(), SysTime()
from AMMISSIONE A
where not exists ( select * from DIMISSIONE D
                   where D.NTS = A.NTS and
                         D.Data > A.Data  )

create view Durata (NTS, Di, Oi, Do, Oo) as
( select * from InOut
    union
  select * from OnlyIn )
```

Un'interpretazione diversa potrebbe essere quella di sommare il *totale* dei giorni di degenza di ogni paziente, anche in intervalli non consecutivi, e individuare il paziente che ha complessivamente trascorso più giorni nell'ospedale (considerando quindi i ricoveri in tutti i reparti). Questa versione è lasciata per esercizio.

B. Linguaggi Formali (5 punti)

1. Estrarre in Algebra Relazionale ottimizzata, Calcolo Relazionale **e** Datalog il Nome dei pazienti dell'ospedale che sono stati ricoverati sempre e solo nel reparto Ortopedia.

Algebra relazionale

Π_{Nome} (($\Pi_{NTS,\ Nome}$ AMMISSIONE) – ($\Pi_{NTS,Nome}$ $\sigma_{Reparto<>\text{“Ortopedia”}}$ AMMISSIONE)))

Calcolo relazionale

{ t | ∃ t_A ∈ AMMISSIONE |
(t[Nome] = t_A[Nome] ∧
¬ (∃ t_{A2} ∈ AMMISSIONE |
t_{A2}[NTS] = $\mathbf{t_A}$[NTS] ∧
t_{A2}[Reparto] <> “Ortopedia”)) }

Datalog

```
AltroReparto ( NTS ) :- AMMISSIONE ( NTS, _, _, _, _, Rep ),
                        Rep <> “Ortopedia”

SoloOrtopedia ( NTS ) :- AMMISSIONE ( NTS, _, _, _, _, _ ),
                         ¬ AltroReparto ( NTS )

? - SoloOrtopedia ( X )
```

2. [**FACOLTATIVO**] *Estrarre in uno a scelta dei tre linguaggi formali i pazienti che, da quando è attivo il sistema di monitoraggio, sono stati ricoverati almeno una volta in tutti i reparti.*

Algebra relazionale

Π_{NTS} AMM – (Π_{NTS} ((Π_{NTS} AMM × $\Pi_{Reparto}$ AMM) – ($\Pi_{NTS,\ Reparto}$ AMM)))

Calcolo relazionale

{ t | ∃ t_A ∈ AMMISSIONE |
(t[NTS] = t_A[NTS] ∧
¬ (∃ t_R ∈ AMMISSIONE |
¬ (∃ t_{A2} ∈ AMMISSIONE |
t_{A2}[NTS] = $\mathbf{t_A}$[NTS] ∧
t_{A2}[Reparto] = $\mathbf{t_R}$[Reparto])))
}

Datalog

Reparti (Rep) :- AMMISSIONE (_, _, _, _, _, Rep)

Pazienti (NTS) :- AMMISSIONE (NTS, _, _, _, _, _)

RepartoVisitato (NTS, Rep) :- AMMISSIONE (NTS, _, _, _, _, Rep)

MancaQualcheReparto (NTS) :- Reparti (Rep), Pazienti (NTS),
¬ RepartoVisitato (NTS, Rep)

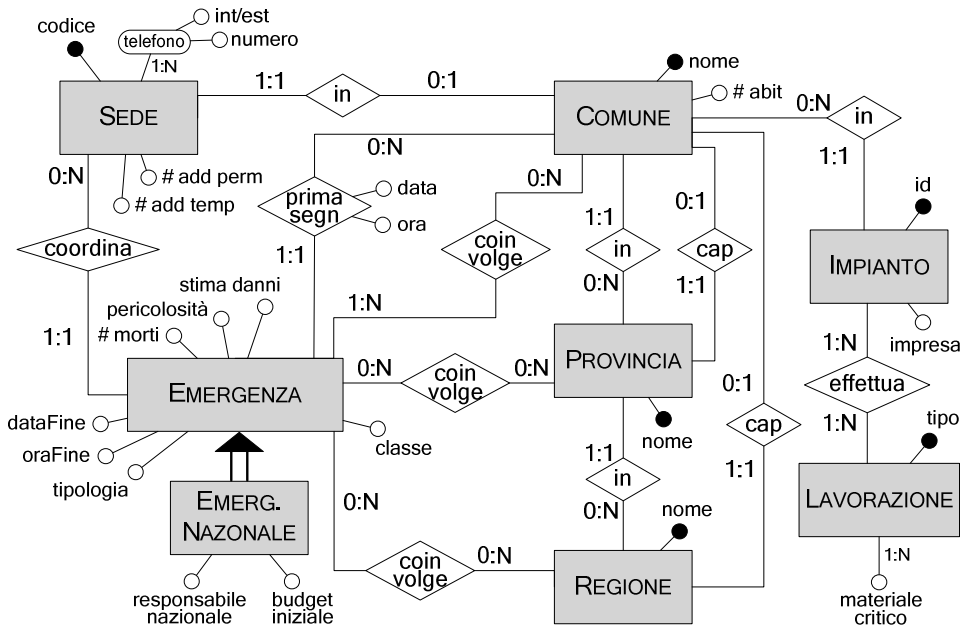
LiHannoVisitatiTutti (NTS) :- Pazienti (NTS),
¬ MancaQualcheReparto (NTS)

? - LiHannoVisitatiTutti (X)

D. PROGETTO CONCETTUALE E LOGICO

La protezione civile gestisce la base di dati delle "emergenze", ove si raccolgono informazioni iniziali relative a questi eventi. Il territorio nazionale è diviso in Regioni, Provincie e Comuni, ciascuno caratterizzato da un nome e un numero di abitanti; province e regioni hanno un comune capoluogo. Presso alcuni comuni sono localizzate le sedi della protezione civile, caratterizzate da un proprio codice, un numero di addetti permanenti e di addetti aggiunti (volontari), vari numeri telefonici ad uso interno e/o esterno. Sempre presso i comuni sono localizzati gli impianti, caratterizzati dal nome dell'impresa e da un numero variabile di lavorazioni pericolose; ciascuna lavorazione pericolosa ha una tipologia e coinvolge un certo numero di materiali critici.

La protezione civile riceve la segnalazione di emergenze, che vengono inizialmente associate ad un comune (di prima segnalazione) e che poi vengono via via associate ad altri comuni; talvolta è coinvolta una intera provincia o regione. Ogni emergenza ha un inizio (il tempo della segnalazione), una fine (il tempo in cui l'emergenza viene definita come "completata" dall'ufficio), una tipologia (ad esempio: nevicata, frana, incendio, attentato), un ufficio responsabile (a cui viene assegnato il coordinamento delle attività locali). Per ogni emergenza, si memorizzano dati relativi alla mortalità (presenza di persone decedute a causa dell'evento), pericolosità (stima iniziale delle persone a rischio di vita a causa dell'evento), impatto economico (stima iniziale dei possibili danni); a partire da questi elementi, si calcola la sua classe (misura di importanza). Ogni emergenza di classe A oppure B deve essere gestita anche a livello nazionale, individuando un responsabile nell'ufficio centrale di coordinamento e stabilendo un budget iniziale per coprire le spese di primo intervento.



Si è ritenuto che l'ufficio responsabile del coordinamento di ogni emergenza sia una sede della protezione civile o comunque faccia capo ad una di tali sedi, immaginando che sai di norma la sede più vicina. Del resto, se anche il controllo effettivo è esercitato da parte di un'unità di crisi che si stabilisce sul posto, tale unità farà comunque riferimento alla sede più vicina.
Si noti che l'attributo classe per l'entità EMERGENZA è derivabile dagli altri attributi, ma abbiamo ritenuto opportuno rappresentarlo indipendentemente.

Questo testo raccoglie una serie di esercizi relativi all'interrogazione e alla progettazione di basi di dati relazionali. Il testo si rivolge primariamente agli studenti di corsi di basi di dati erogati nella Laurea di Primo Livello (nelle discipline dell'Ingegneria dell'Informazione), e nella Laurea Magistrale (nelle altre discipline), oltre che ai partecipanti a corsi di formazione su basi di dati per realtà aziendali e per programmi di Master.

DANIELE BRAGA è Ricercatore al Politecnico di Milano presso la Facoltà di Ingegneria dell'Informazione, dove ha ottenuto nel 2005 il Dottorato di Ricerca. Il lavoro di ricerca riguarda i linguaggi di interrogazione e manipolazione di dati semi-strutturati. Il lavoro più cospicuo è la specifica e l'implementazione di un linguaggio grafico per interrogare tali dati. Altri temi di ricerca riguardano l'integrazione di servizi Web e motori di ricerca basata su tecniche sintattiche e semantiche, la minimizzazione automatica di vincoli, e la modellazione concettuale di applicazioni Web. È titolare di due corsi curricolari: Informatica I per il corso di laurea in Ingegneria dell'Automazione e Progetto di Basi di Dati per quello in Ingegneria Informatica. Ha svolto e svolge seminari didattici nell'ambito di altri corsi, anche presso altri atenei (Basi di Dati 1 e 2, Informatica II, Ingegneria del Software, Sistemi Operativi). Coordina e supervisiona l'attività di tesi svolta da numerosi studenti del Politecnico nell'area Basi di Dati.

MARCO BRAMBILLA è Ricercatore al Politecnico di Milano presso la Facoltà di Ingegneria dell'Informazione dal febbraio 2005, anno in cui ha conseguito Dottorato di Ricerca. È docente del corso di Tecnologie per i Sistemi Informativi e del corso Informatica II; svolge e ha svolto esercitazioni e seminari didattici per corsi legati all'informatica di base (Informatica I), alle basi di dati (Basi Dati 1 e 2) e alle applicazioni Web (Tecnologie Informatiche per il Web), sia in ambito industriale che accademico, oltre che corsi specialistici all'interno di master post-laurea. Svolge attività di ricerca nei seguenti ambiti: metodologie e linguaggi visuali per la progettazione di applicazioni Web, design pattern per il Web, composizione di Web services e integrazione con applicazioni Web tradizionali, modelli e metodi per la progettazione di applicazioni basate su workflow. Su questi temi supervisiona svariate attività di tesi all'interno del Politecnico. È coautore di diversi testi didattici e scientifici pubblicati a livello italiano e internazionale.

ALESSANDRO CAMPI è Ricercatore al Politecnico di Milano presso la Facoltà di Ingegneria dell'Informazione, dove ha ottenuto nel 2004 il Dottorato di Ricerca. La sua attività di ricerca si concentra sull'estensione dei linguaggi di interrogazione e manipolazione di dati semi-strutturati, sulle tecniche di progettazione e verifica di interfacce grafiche e sulla modellazione di applicazioni Web secondo metodologie model-driven. Da tempo si dedica anche alla progettazione di piattaforme di e-learning. Ha svolto esercitazioni e seminari didattici in vari corsi curricolari del Politecnico di Milano sia nel corso di laurea in Ingegneria Informatica, sia nel Master in Editoria Multimediale: è docente del corso di Informatica 3 e del corso di Progetto di Basi di Dati, collabora ai corsi di Basi di Dati 2, Informatica II e Ingegneria del Software. Coordina e supervisiona le attività di tesi svolte dagli studenti all'interno del gruppo basi di dati del Politecnico di Milano.

ISBN 978-88-7488-328-8

9 788874 883288 >

Euro 15,00

SOCIETÀ EDITRICE ESCULAPIO

www.editrice-esculapio.it