**Creazione tabelle**

create table SPEDIZIONE(

codice int not null auto\_increment primary key,

Data date not null,

Utente varchar(15) not null,

foreign key (Utente)

references UTENTE(Soggetto)

on delete cascade

on update cascade

);

create table CONTO(

id int not null auto\_increment primary key,

saldo int not null,

Soggetto varchar(15) not null,

foreign key(Soggetto)

references SOGGETTO(CF)

on delete cascade

on update cascade

);

create table NAZIONE(

nome varchar(50) not null primary key

);

create table CITTA(

nome varchar(50) not null primary key,

Nazione varchar(50) not null,

foreign key (Nazione)

references NAZIONE(nome)

on delete cascade

on update cascade

);

create table SOGGETTO(

CF varchar(15) not null primary key,

DataNascita date not null,

email varchar(50) not null,

nome varchar(50) not null,

cognome varchar(50) not null,

Via varchar(50) not null,

civico int not null,

CAP int(5) not null,

Città varchar(50) not null,

foreign key (Via,civico,CAP)

references INDIRIZZO (Via,civico,CAP)

on delete cascade

on update cascade,

foreign key (Città)

references CITTA(nome)

on delete cascade

on update cascade

);

create table UTENTE(

nickname varchar(50),

media\_feedback enum('0','1','2','3',4','5','6','7','8','9','10') not null default '0',

Soggetto varchar(15) not null,

foreign key (Soggetto)

references SOGGETTO(CF)

on delete cascade

on update cascade

);

create table AUTONOMA(

Spedizione int not null,

foreign key (Spedizione)

references SPEDIZIONE(codice)

on delete cascade

on update cascade

);

create table SERVIZIO(

Spedizione int not null,

foreign key (Spedizione)

references SPEDIZIONE(codice)

on delete cascade

on update cascade

);

create table EFFETTUATA(

Data date not null,

Spedizione int not null,

foreign key (Spedizione)

references SPEDIZIONE(codice)

on delete cascade

on update cascade

);

create table NONEFFETTUATA(

Data date not null,

Spedizione int not null,

foreign key (Spedizione)

references SPEDIZIONE(codice)

on delete cascade

on update cascade

);

create table TELEFONO(

num varchar(10) not null primary key,

Soggetto varchar(15) not null,

foreign key (Soggetto)

references SOGGETTO(CF)

on delete cascade

on update cascade

);

create table MOVIMENTO(

importo bigint not null default 0,

DataOra timestamp not null primary key,

Conto int not null,

foreign key (Conto)

references CONTO(id)

on delete cascade

on update cascade

);

create table DIRETTOREMAGAZZINO(

licenza boolean not null,

Soggetto varchar(15) not null,

foreign key (Soggetto)

references SOGGETTO(CF)

on delete cascade

on update cascade

);

create table ANNUNCIO(

codice int auto\_increment not null primary key,

DataOra timestamp not null,

periodo\_usato enum('meno di 1 mese','meno di 6 mesi','meno di 1 anno','più di 1 anno','nuovo') not null default 'nuovo',

prezzo int not null,

descrizione varchar(100) not null,

Soggetto varchar(15) not null,

Categoria varchar(15) not null,

foreign key (Soggetto)

references SOGGETTO(CF)

on delete cascade

on update cascade,

foreign key (Categoria)

references CATEGORIA(nome)

on delete cascade

on update cascade

);

create table CATEGORIA(

nome varchar(15) not null primary key

);

create table FOTO(

percorso varchar(50) not null primary key,

Annuncio int not null,

foreign key (Annuncio)

references ANNUNCIO (codice)

on delete cascade

on update cascade

);

create table ANNUNCIOACQUISTATO(

DataOra timestamp not null,

Annuncio int not null,

Soggetto varchar(15) not null,

foreign key (Annuncio)

references ANNUNCIO (codice)

on delete cascade

on update cascade,

foreign key (Soggetto)

references Soggetto (CF)

on delete cascade

on update cascade

);

create table ANNUNCIOVENDUTO(

DataOra timestamp not null,

Annuncio int not null,

foreign key (Annuncio)

references ANNUNCIO(codice)

on delete cascade

on update cascade

);

create table FEEDBACK(

id int auto\_increment not null primary key,

voto enum('0','1','2','3',4','5','6','7','8','9','10') not null default '0',

AnnuncioAcquistato int not null,

Utente varchar(15) not null,

foreign key (AnnuncioAcquistato)

references AnnuncioAcquistato(Annuncio)

on delete cascade

on update cascade,

foreign key (Utente)

references UTENTE(Soggetto)

on delete cascade

on update cascade

);

create table DIPENDENTE(

nome varchar(15) not null,

cognome varchar(15) not null,

DataNascita date not null,

CF varchar(15) not null primary key

);

create table AZIENDA(

ragione\_sociale varchar(200) not null,

id int auto\_increment not null primary key,

nome varchar(50) not null,

Città varchar(50) not null,

Via varchar(50) not null,

civico int not null,

CAP int(5) not null,

foreign key (Via,civico,CAP)

references INDIRIZZO (Via,civico,CAP)

on delete cascade

on update cascade,

foreign key (Città)

references CITTA(nome)

on delete cascade

on update cascade

);

create table MAGAZZINO(

id int auto\_increment not null primary key,

ragione\_sociale varchar(200) not null,

Via varchar(50) not null,

civico int not null,

CAP int(5) not null,

DirettoreMagazzino varchar(15) not null,

Città varchar(50) not null,

foreign key (Via,civico,CAP)

references INDIRIZZO (Via,civico,CAP)

on delete cascade

on update cascade,

foreign key (Città)

references CITTA(nome)

on delete cascade

on update cascade,

foreign key (DirettoreMagazzino)

references DIRETTOREMAGAZZINO(Soggetto)

on delete cascade

on update cascade

);

create table DIRETTORE(

Dipendente varchar(15) not null,

foreign key (Dipendente)

references DIPENDENTE(CF)

on delete cascade

on update cascade

);

create table impiegatopassato(

fine timestamp not null,

Azienda int not null,

Dipendente varchar(15) not null,

foreign key (Azienda)

references AZIENDA(id)

on delete cascade

on update cascade,

foreign key (Dipendente)

references DIPENDENTE(CF)

on delete cascade

on update cascade

);

create table impiegatocorrente(

inizio timestamp not null,

Azienda int not null,

Dipendente varchar(15) not null,

foreign key (Azienda)

references AZIENDA(id)

on delete cascade

on update cascade,

foreign key (Dipendente)

references DIPENDENTE(CF)

on delete cascade

on update cascade

);

create table compra(

Annuncio int unique not null,

Soggetto varchar(15) unique not null,

foreign key (Annuncio)

references ANNUNCIOACQUISTATO(Annuncio)

on delete cascade

on update cascade,

foreign key (Soggetto)

references SOGGETTO(CF)

on delete cascade

on update cascade

);

create table fornitore(

Azienda int not null,

Magazzino int not null,

foreign key (Azienda)

references AZIENDA(id)

on delete cascade

on update cascade,

foreign key (Magazzino)

references MAGAZZINO(id)

on delete cascade

on update cascade

);

create table INDIRIZZO(

Via varchar(50) not null,

civico int not null,

CAP int(5) not null,

primary key(Via,civico,CAP)

);

**Vincoli**

-**V.Indirizzo**: Un SOGGETTO, un MAGAZZINO e un’AZIENDA non possono avere lo stesso indirizzo di residenza:

* Per ogni istanza (S,Ind) di “IndSog” con S appartenente a SOGGETTO e Ind appartenente a INDIRIZZO e per ogni istanza (Az,Ind’) con Az appartenente a AZIENDA e Ind’ appartenente a INDIRIZZO e per ogni istanza (Mag,Ind’’) con Mag appartenente a MAGAZZINO e Ind’’ appartenente a INDIRIZZO deve valere la regola: Ind != Ind’ != Ind’’;
* **SQL:**
* **Funzione**
* create function ControlloIndirizzi(Via varchar(50), civico int, CAP int) returns int DETERMINISTIC begin
* select count(\*) into @var2 from magazzino
* where magazzino.via=Via and
* magazzino.civico=civico and
* magazzino.CAP=CAP;
* select count(\*) into @var3 from azienda
* where azienda.via=Via and
* azienda.civico=civico and
* azienda.CAP=CAP;
* if(@var2+@var3=0) then return 1;
* end if;
* return 0;
* end $$
* delimiter ;
* **Trigger**
* delimiter $$
* create trigger V\_indirizzo\_soggetto
* before insert on soggetto
* for each row
* begin
* if(ControlloIndirizzi(new.Via,new.civico,new.CAP)=0) then
* signal sqlstate '45000'
* set message\_text='Un SOGGETTO non può avere lo stesso indirizzo di un MAGAZZINO o un\' AZIENDA';
* end if;
* end $$
* delimiter ;
* delimiter $$
* create trigger V\_indirizzo\_magazzino
* before insert on magazzino
* for each row
* begin
* if(ControlloIndirizzi(new.Via,new.civico,new.CAP)=0) then
* signal sqlstate '45000'
* set message\_text='Un MAGAZZINO non può avere lo stesso indirizzo di un SOGGETTO o un\' AZIENDA';
* end if;
* end $$
* delimiter ;
* delimiter $$
* create trigger V\_indirizzo\_azienda
* before insert on azienda
* for each row
* begin
* if(ControlloIndirizzi(new.Via,new.civico,new.CAP)=0) then
* signal sqlstate '45000'
* set message\_text='Un\'AZIENDA non può avere lo stesso indirizzo di un SOGGETTO o un MAGAZZINO';
* end if;
* end $$
* delimiter ;

-**V.DataOra**: La data e l’ora di annuncio acquistato deve essere maggiore alla data e l’ora del medesimo annuncio appena pubblicato:

* Per ogni istanza (AN,AP) di “ISA-AnnAc” con AN appartenente a ANNUNCIO e AP appartenente a ANNUNCIO ACQUISTATO, sia D l’attributo “DataOra” di AN e D’ l’attributo “DataOra” di AC deve valere la regola: D < D’;
* **SQL**:
* **Funzione**
* /\* INIZIO: V.DataOra; \*/
* delimiter $$
* create function ControlloOra(asdf timestamp, codice int) returns int deterministic begin
* set @var1 = (select DataOra from annuncio
* where annuncio.codice=codice);
* if(@var1>asdf) then return 0;
* end if;
* return 1;
* end $$
* delimiter ;
* drop function if exists ControlloOra;
* **Trigger**
* delimiter $$
* create trigger V\_DataOraAcquisto
* before insert on annuncioacquistato
* for each row
* begin
* if(ControlloOra(new.DataOra,new.Annuncio)=0) then
* signal sqlstate '45000'
* set message\_text='La DataOra dell\'annuncio acquistato deve essere maggiore della DataOra dell\'annuncio pubblicato';
* end if;
* end $$
* delimiter ;
* DROP TRIGGER IF EXISTS V\_DataOraAcquisto;
* delimiter $$
* create trigger V\_DataOraVenduto
* before insert on annunciovenduto
* for each row
* begin
* if(ControlloOra(new.DataOra,new.Annuncio)=0) then
* signal sqlstate '45000'
* set message\_text='La DataOra dell\'annuncio venduto deve essere maggiore della DataOra dell\'annuncio pubblicato';
* end if;
* end $$
* delimiter ;
* DROP TRIGGER IF EXISTS V\_DataOraVenduto;
* /\* FINE: V.DataOra; \*/

**-V.SpedizioneData**: La data di una Spedizione effettuata non può essere minore della data della sua Spedizione non effettuata:

* Per ogni istanza (S,E) di “ISA-SpEff” con S appartenente a SPEDIZIONE e E appartenente a EFFETTUATA e per ogni istanza (S,N) con N appartenente a NON EFFETTUATA, sia D l’attributo data di N e D’ l’attributo data di E, deve valere la regola D <= D’;
* **SQL**
* **FUNZIONI**
* /\* INIZIO: Controllo della data \*/
* delimiter $$
* create function ControlloData(x date,Spedizione int) returns int deterministic begin
* select Data into @var1 from noneffettuata
* where noneffettuata.Spedizione=Spedizione;
* if(@var1>x) then return 0; end if;
* return 1;
* end $$
* delimiter ;
* /\* FINE: Controllo della data \*/
* **TRIGGER**
* /\* INIZIO: trigger per l'ISA disgiunta di SPEDIZIONE (EFFETTUATA) \*/
* delimiter $$
* create trigger V\_DataOra
* before insert on effettuata
* for each row
* begin
* if(ControlloData(new.Data,new.Spedizione)=0) then
* signal sqlstate '45000'
* set message\_text='La data deve essere maggiore o uguale alla Data della sua Spedizione non effettuata';
* end if;
* end $$
* delimiter ;
* /\* FINE: trigger per l'ISA disgiunta di SPEDIZIONE (EFFETTUATA) \*/

**-V.SpedizioneISAEffNonEff**: ogni istanza di Spedizione deve partecipare o alla relazione ISA-SpEff o alla relazione ISA-SpN ma non ad entrambe:

* Per ogni SP appartenente a SPEDIZIONE, SP deve fare parte dell’istanza (SP,EFF) di “ISA-SpEff” con EFF appartenente a EFFETTUATA, oppure deve fare parte dell’istanza (SP,NEFF) di “ISA-SpN” con NEFF appartenente a NON EFFETTUATA, ma non può farne parte ad entrambe;
* **FUNZIONI**
* /\* Si è pensato a livello applicazione che quando viene comprato un annuncio viene inserito direttamente in “non effettuato”;
* /\* INIZIO: funzione per controllare se il record che sto per inserire in noneffettuata non sia già in effettuata \*/
* delimiter $$
* create function ControlloISASpedizione(Spedizione int) returns int deterministic begin
* select count(\*) into @var1 from effettuata
* where effettuata.Spedizione=Spedizione;
* if(@var1>0) then return 0; end if;
* return 1;
* end $$
* delimiter ;
* /\* FINE: funzione per controllare se il record che sto per inserire in noneffettuata non sia già in effettuata \*/
* **TRIGGER**
* /\* INIZIO: trigger per l'ISA disgiunta di SPEDIZIONE (NONEFFETTUATA) \*/
* delimiter $$
* create trigger V\_SpedizioneISAEffNonEff\_2
* before insert on noneffettuata
* for each row
* begin
* if(ControlloISASpedizione(new.Spedizione)=0) then
* signal sqlstate '45000'
* set message\_text='Questa spedizione è stata già effettuata';
* end if;
* end $$
* delimiter ;
* /\* FINE: trigger per l'ISA disgiunta di SPEDIZIONE (NONEFFETTUATA) \*/
* /\* INIZIO: trigger per il controllo della data prima dell'inserimento \*/
* delimiter $$
* create trigger V\_DataOra
* before insert on effettuata
* for each row
* begin
* if(ControlloData(new.Data,new.Spedizione)=0) then
* signal sqlstate '45000'
* set message\_text='La data deve essere maggiore o uguale alla Data della sua Spedizione non effettuata';
* end if;
* end $$
* delimiter ;
* drop trigger V\_DataOra;
* /\* FINE: trigger per il controllo della data prima dell'inserimento \*/

**-V.SpedizioneISASpSerISASpAut**: ogni istanza di Spedizione deve partecipare o alla relazione ISA-SpAut o alla relazione ISA-SpSer ma non ad entrambe:

* Per ogni SP appartenente a SPEDIZIONE, SP deve fare parte dell’istanza (SP,EFF) di “ISA-SpSer” con EFF appartenente a SERVIZIO, oppure deve fare parte dell’istanza (SP,NEFF) di “ISA-SpAut” con NEFF appartenente a AUTONOMA, ma non può farne parte ad entrambe;
* **SQL**
* **FUNZIONI**
* /\* INIZIO: controllo che solo un'istanza tra SERVIZIO/AUTONOMA ci può essere \*/
* delimiter $$
* create function ControlloDisgiunzioneServizio(x int) returns int deterministic begin
* select count(\*) into @var1 from servizio
* where servizio.Spedizione=x;
* if(@var1>0) then return 0; end if;
* return 1;
* end $$
* delimiter ;
* /\* FINE: controllo che solo un'istanza tra SERVIZIO/AUTONOMA ci può essere \*/
* /\* INIZIO: controllo che solo un'istanza tra SERVIZIO/AUTONOMA ci può essere \*/
* delimiter $$
* create function ControlloDisgiunzioneAutonoma(x int) returns int deterministic begin
* select count(\*) into @var1 from autonoma
* where autonoma.Spedizione=x;
* if(@var1>0) then return 0; end if;
* return 1;
* end $$
* delimiter ;
* **TRIGGER**
* /\* INIZIO: disgiunzione AUTONOMA da SERVIZIO \*/
* delimiter $$
* create trigger V\_SpedizioneISASpSerISASpAut\_1
* before insert on autonoma
* for each row
* begin
* if(ControlloDisgiunzioneServizio(new.Spedizione)=0) then
* signal sqlstate '45000'
* set message\_text='Questa spedizione è impostata su SERVIZIO';
* end if;
* end $$
* delimiter ;
* /\* FINE: disgiunzione AUTONOMA da SERVIZIO \*/
* /\* INIZIO: disgiunzione SERVIZIO da AUTONOMA \*/
* delimiter $$
* create trigger V\_SpedizioneISASpSerISASpAut\_2
* before insert on servizio
* for each row
* begin
* if(ControlloDisgiunzioneAutonoma(new.Spedizione)=0) then
* signal sqlstate '45000'
* set message\_text='Questa spedizione è impostata su AUTONOMA';
* end if;
* end $$
* delimiter ;
* /\* FINE: disgiunzione SERVIZIO da AUTONOMA \*/

**-V.SoggettoISASogUtISASogDir**: ogni istanza di Soggetto deve partecipare o alla relazione ISA-SogUt o alla relazione ISA-SogDir ma non ad entrambe:

* Per ogni SP appartenente a SOGGETTO, SP deve fare parte dell’istanza (SP,EFF) di “ISA-SogUt” con EFF appartenente a UTENTE, oppure deve fare parte dell’istanza (SP,NEFF) di “ISA-SogDir” con NEFF appartenente a DIRETTORE MAGAZZINO, ma non può farne parte ad entrambe;
* **SQL**
* **FUNZIONI**
* /\* INIZIO: funzione per il controllo disgiunzione UTENTE -> DIRETTORE MAGAZZINO \*/
* delimiter $$
* create function ControlloDisgiunzioneDirettoreMagazzino(x varchar(15)) returns int deterministic begin
* select count(\*) into @var1 from direttoremagazzino
* where direttoremagazzino.Soggetto=x;
* if(@var1>0) then return 0; end if;
* return 1;
* end $$
* delimiter ;
* /\* FINE: funzione per il controllo disgiunzione UTENTE -> DIRETTORE MAGAZZINO \*/
* /\* INIZIO: funzione per il controllo disgiunzione DIRETTORE MAGAZZINO -> UTENTE \*/
* delimiter $$
* create function ControlloDisgiunzioneUtente(x varchar(15)) returns int deterministic begin
* select count(\*) into @var1 from utente
* where utente.Soggetto=x;
* if(@var1>0) then return 0; end if;
* return 1;
* end $$
* delimiter ;
* drop function ControlloDisgiunzioneUtente;
* drop function ControlloDisgiunzioneDirettoreMagazzino;
* /\* FINE: funzione per il controllo disgiunzione DIRETTORE MAGAZZINO -> UTENTE \*/
* **TRIGGER**
* /\* INIZIO: trigger per il controllo della disgiunzione UTENTE -> DIRETTORE MAGAZZINO \*/
* delimiter $$
* create trigger V\_SoggettoISASogUtISASogDir\_1
* before insert on utente
* for each row
* begin
* if(ControlloDisgiunzioneDirettoreMagazzino(new.Soggetto)=0) then
* signal sqlstate '45000'
* set message\_text='Questo UTENTE è già un DIRETTORE MAGAZZINO esistente';
* end if;
* end $$
* delimiter ;
* /\* FINE: trigger per il controllo della disgiunzione UTENTE -> DIRETTORE MAGAZZINO \*/
* /\* INIZIO: trigger per il controllo della disgiunzione DIRETTORE MAGAZZINO -> UTENTE \*/
* delimiter $$
* create trigger V\_SoggettoISASogUtISASogDir\_2
* before insert on direttoremagazzino
* for each row
* begin
* if(ControlloDisgiunzioneUtente(new.Soggetto)=0) then
* signal sqlstate '45000'
* set message\_text='Questo DIRETTORE MAGAZZINO è già un UTENTE esistente';
* end if;
* end $$
* delimiter ;
* /\* FINE: trigger per il controllo della disgiunzione DIRETTORE MAGAZZINO -> UTENTE \*/

**-V.Direttore**: Per ogni AZIENDA deve esistere uno ed un solo DIPENDENTE che è anche un DIRETTORE e che è un impiegato corrente:

* Per ogni istanza (Az,Dip) di “impiegato corrente” con Az appartenente a AZIENDA e Dip appartenente a DIPENDENTE, deve esistere l’istanza (Dip,Dir) di “ISA-Dip” con Dip appartenente a DIRETTORE il cui numero di tale istanza deve essere >0 e <2;
* **SQL**
* **FUNZIONI**
* /\* INIZIO: Controllo numero DIRETTORI \*/
* delimiter $$
* create function ControlloDirettori(x varchar(15)) returns int deterministic begin
* /\* Prima cerchiamo l'azienda per cui lavora x \*/
* select Azienda into @var1 from impiegatocorrente
* where impiegatocorrente.Dipendente=x;
* /\* Poi vediamo quanti direttori ci sono in tale azienda e se sono più di 0 significa che già c'è un direttore \*/
* select count(\*) into @var2 from impiegatocorrente
* inner join direttore on
* direttore.Dipendente=impiegatocorrente.Dipendente
* and impiegatocorrente.Azienda=@var1;
* if(@var2=0) then return 1;
* end if;
* return 0;
* end $$
* delimiter ;
* /\* FINE: Controllo numero DIRETTORI \*/
* **TRIGGER**
* /\* V\_Direttore \*/
* /\* Si è pensato onde evitare che non ci sia nessun direttore per un'azienda che quando si inserisce il primo DIPENDENTE automaticamente esso è il direttore \*/
* /\* Prima di tale operazione varrà eseguito un controllo per appunto controllare quanti direttori ci sono (max 1) \*/
* /\* INIZIO \*/
* delimiter $$
* create trigger V\_Direttore
* after insert on impiegatocorrente
* for each row
* begin
* if(ControlloDirettori(new.Dipendente)=1) then
* insert into direttore(Dipendente)
* values(new.Dipendente);
* end if;
* end $$
* delimiter ;
* /\* FINE \*/
* /\* INIZIO: Trigger per gestire l'inserimento di un solo DIRETTORE \*/
* delimiter $$
* create trigger V\_Direttore\_2
* before insert on direttore
* for each row
* begin
* if(ControlloDirettori(new.Dipendente)=1) then
* set new.Dipendente=new.Dipendente;
* else
* signal sqlstate '45000'
* set message\_text='Esiste già un direttore per questa azienda!';
* end if;
* end $$
* delimiter ;
* /\* FINE: Trigger per gestire l'inserimento di un solo DIRETTORE \*/

-**V.CompraVendiUtente**: Un utente non può comprare un annuncio che egli stesso ha pubblicato:

* Per ogni (U,AP) di “ordine” con U appartenente a UTENTE isa SOGGETTO e AP appartenente a ANNUNCIO, non deve esistere un’istanza (U,AC) con AC appartenente a ANNUNCIO ACQUISTATO dove AC=AP;
* **SQL:**
* **FUNZIONI**
* delimiter $$
* create function Controlla\_proprietario(x int,y varchar(15)) returns int deterministic begin
* /\* controllo che non sta comprando un suo annuncio \*/
* select count(\*) into @var1 from annuncio
* inner join soggetto
* on annuncio.Soggetto=y
* and soggetto.CF=y
* and annuncio.codice=x;
* /\* se è almeno impegnato con questa relazione allora non può comprare \*/
* if(@var1>0) then
* return 0;
* end if;
* /\* altrimenti può comprare \*/
* return 1;
* end $$
* delimiter ;
* /\* FINE: funzione che viene usata per il trigger del vincolo che un utente non può comprare un suo stesso annuncio \*/
* **TRIGGER**
* /\* INIZIO: trigger per controllare che un utente non sta comprando un suo annuncio \*/
* delimiter $$
* create trigger V\_CompraVendiUtente\_1
* before insert on annuncioacquistato
* for each row
* begin
* if(Controlla\_proprietario(new.Annuncio,new.Soggetto)=1) then /\* se restituisce 1 allora può comprarlo, else si \*/
* set new.Annuncio=new.Annuncio;
* set new.Soggetto=new.Soggetto;
* set new.DataOra=new.DataOra;
* else
* signal sqlstate '45000'
* set message\_text='Non puoi comprare un tuo annuncio!';
* end if;
* end $$
* delimiter ;
* /\* FINE: trigger per controllare che un utente non sta comprando un suo annuncio \*/

-**V.DirettoreMagazzino**: Un direttore di un magazzino non può acquistare annunci;

* Per ogni istanza D di DIRETTORE MAGAZZINO isa SOGGETTO non deve esistere l’istanza (D,AC) di “ordine” con AC appartenente a ANNUNCIO ACQUISTATO isa ANNUNCIO;
* **SQL:**
* **FUNZIONI**
* /\* INIZIO: Controllo che chi sta comprando non è un DIRETTORE MAGAZZINO \*/
* delimiter $$
* create function ControllaCompratore(x int,y varchar(15)) returns int deterministic begin
* /\* controllo se y è un DIRETTORE MAGAZZINO \*/
* select count(\*) into @var1 from soggetto
* inner join direttoremagazzino
* on direttoremagazzino.Soggetto=soggetto.CF
* and soggetto.CF=y;
* /\* se @var1 > 0 allora sicuramente è un direttore \*/
* if(@var1>0) then
* return 0;
* end if;
* /\* altrimenti non lo è e quindi può comprare \*/
* return 1;
* end $$
* delimiter ;
* /\* FINE: Controllo che chi sta comprando non è un DIRETTORE MAGAZZINO \*/
* **TRIGGER**
* /\* INIZIO: V\_DirettoreMagazzino \*/
* delimiter $$
* create trigger V\_DirettoreMagazzino
* before insert on annuncioacquistato
* for each row
* follows V\_CompraVendiUtente\_1
* begin
* if(ControllaCompratore(new.Annuncio,new.Soggetto)=0) then /\* se restituisce 0 allora un DIRETTORE MAGAZZINO sta ACQUISTANDO \*/
* signal sqlstate '45000'
* set message\_text='Un DIRETTORE di MAGAZZINO non può comprare un ANNUNCIO!';
* end if;
* end $$
* delimiter ;
* /\* FINE: V\_DirettoreMagazzino \*/

-**V.DipendenteAzienda**: Se un dipendente non lavora più nell’azienda dove lavorava, la data e l’ora di fine deve essere strettamente maggiore della data e l’ora dell’inizio:

* Per ogni istanza (D,A) di “impiegato corrente” con D appartenente a DIPENDENTE e A appartenente a AZIENDA, sia i l’attributo inizio di “impiegato corrente”, se (D,A) è anche un’istanza di “impiegato passato”, sia j l’attributo fine di “impiegato passato”, deve valere che j>i;
* **SQL:**
* **FUNZIONI**
* /\* INIZIO: controllo della data di fine DIPENDENTE \*/
* delimiter $$
* create function ControllaDipendente(x timestamp,y int,z varchar(15)) returns int deterministic begin
* /\* controllo se x (dataora) di z (dipendente) è maggiore della sua data di inizio \*/
* select inizio into @var1 from impiegatocorrente
* where impiegatocorrente.Azienda=y
* and impiegatocorrente.Dipendente=z;
* /\* se @var1<x \*/
* if(@var1<x) then
* return 1;
* end if;
* /\* altrimenti non è corretta la dataora \*/
* return 0;
* end $$
* delimiter ;
* /\* FINE: controllo della data di fine DIPENDENTE \*/
* **TRIGGER**
* /\* INIZIO: V\_DipendenteAzienda \*/
* delimiter $$
* create trigger V\_DipendenteAzienda
* before insert on impiegatopassato
* for each row
* begin
* if(ControllaDipendente(new.fine,new.Azienda,new.Dipendente)=0) then /\* se restituisce 0 allora la data di fine non è corretta\*/
* signal sqlstate '45000'
* set message\_text='La data di fine deve essere maggiore della sua data di inizio';
* end if;
* end $$
* delimiter ;
* /\* FINE: V\_DipendenteAzienda \*/

-**V.SpedizioneServizio**: Se un direttore di magazzino ha pubblicato un annuncio allora la sua eventuale spedizione deve essere per forza di servizio:

* Per ogni istanza (D,AV) di “ordine” con D appartenente a DIRETTORE MAGAZZINO isa SOGGETTO e AV appartenente a ANNUNCIO, se AV appartiene a ANNUNCIO VENDUTO isa ANNUNCIO, sia (AV,SP) con SP appartenente a SPEDIZIONE, SP deve per forza assumere il valore SER appartenente a SERVIZIO e non a AUTONOMA;
* **SQL:**
* **FUNZIONI**
* /\* INIZIO: funzione che controlla che l'annuncio è di un DIRETTORE MAGAZZINO o meno \*/
* delimiter $$
* create function ControlloV\_SpedizioneServizio(x int) returns int deterministic begin
* select count(\*) into @var1 from soggetto,direttoremagazzino,annuncio,annunciovenduto,spedizione
* where soggetto.CF=direttoremagazzino.Soggetto
* and annuncio.Soggetto=soggetto.CF
* and annunciovenduto.Annuncio=annuncio.codice
* and spedizione.AnnuncioVenduto=annunciovenduto.Annuncio
* and spedizione.codice=x;
* /\* se @var1 è uguale a zero allora non è di un DIRETTORE MAGAZZINO e quindi non può essere di SERVIZIO la SPEDIZIONE \*/
* if(@var1=0) then return 0;
* end if;
* return 1;
* end $$
* delimiter ;
* /\* FINE: funzione che controlla che l'annuncio è di un DIRETTORE MAGAZZINO o meno \*/
* **TRIGGER**
* /\* INIZIO: V\_SpedizioneServizio \*/
* delimiter $$
* create trigger V\_SpedizioneServizio
* before insert on servizio
* for each row
* begin
* if(ControlloV\_SpedizioneServizio(new.Spedizione)=0) then /\* se restituisce 0 allora non è di un direttore magazzino e quindi non può essere di servizio la spedizione \*/
* signal sqlstate '45000'
* set message\_text='La spedizione per un UTENTE deve essere per forza AUTONOMA';
* end if;
* end $$
* delimiter ;
* /\* FINE: V\_SpedizioneServizio \*/

-**V.SpedizioneAutonoma**: Se un cliente ha pubblicato un annuncio allora la sua eventuale spedizione deve essere per forza autonoma:

* Per ogni istanza (U,AV) di “ordine” con U appartenente a UTENTE isa SOGGETTO e AV appartenente a ANNUNCIO, se AV appartiene a ANNUNCIO VENDUTO isa ANNUNCIO, sia (AV,SP) con SP appartenente a SPEDIZIONE, SP deve per forza assumere il valore AUT appartenente a AUTONOMA e non a SERVIZIO;
* **SQL:**
* **FUNZIONI**
* /\* INIZIO: funzione che controlla che l'annuncio è di un DIRETTORE MAGAZZINO o meno \*/
* delimiter $$
* create function ControlloV\_SpedizioneServizio\_2(x int) returns int deterministic begin
* select count(\*) into @var1 from soggetto,utente,annuncio,annunciovenduto,spedizione
* where soggetto.CF=utente.Soggetto
* and annuncio.Soggetto=soggetto.CF
* and annunciovenduto.Annuncio=annuncio.codice
* and spedizione.AnnuncioVenduto=annunciovenduto.Annuncio
* and spedizione.codice=x;
* /\* se @var1 è uguale a zero allora non è di un UTENTE e quindi non può essere di SERVIZIO AUTONOMA \*/
* if(@var1=0) then return 0;
* end if;
* return 1;
* end $$
* delimiter ;
* /\* FINE: funzione che controlla che l'annuncio è di un DIRETTORE MAGAZZINO o meno \*/
* **TRIGGER**
* /\* INIZIO: V\_SpedizioneServizio\_2 \*/
* delimiter $$
* create trigger V\_SpedizioneServizio\_2
* before insert on autonoma
* for each row
* begin
* if(ControlloV\_SpedizioneServizio\_2(new.Spedizione)=0) then /\* se restituisce 0 allora non è di un utente e quindi non può essere autonoma la spedizione \*/
* signal sqlstate '45000'
* set message\_text='La spedizione per un DIRETTORE MAGAZZINO deve essere per forza di SERVIZIO';
* end if;
* end $$
* delimiter ;
* /\* FINE: V\_SpedizioneServizio\_2 \*/