**Specifica del database in SQL**

Sommario

[1. Definizione delle relazioni 1](#_Toc91770364)

[2. Definizione dei vincoli 8](#_Toc91770365)

[2.1 Vincoli di cardinalità entità-associazioni 9](#_Toc91770366)

[2.2. Business rules 24](#_Toc91770367)

[2.3. Regole di vincolo derivate dalle fasi di ristrutturazione 27](#_Toc91770368)

# 1. Definizione delle relazioni

DBMS usato: Mysql

create table softwarehouse

(

NomeSoft varchar(50) primary key,

NazSede varchar(50) not null

);

create table videogioco

(

NomeVid varchar(50) primary key,

Prezzo decimal(5,2) not null CHECK (Prezzo >=0 and Prezzo <=999.99),

Sconto tinyint not null DEFAULT 0 CHECK (Sconto >= 0 and Sconto <= 100),

Softwarehouse varchar(50) not null,

LinguaPrimaria varchar(50) not null,

GenerePrimario varchar(50) not null,

ConsolePrimaria varchar(50) not null,

FOREIGN KEY (Softwarehouse) references Softwarehouse(NomeSoft)

on update cascade

on delete cascade,

FOREIGN KEY (LinguaPrimaria) references Lingua(NomeLin)

on update cascade

on delete cascade,

FOREIGN KEY (GenerePrimario) references Genere(NomeGen)

on update cascade

on delete cascade,

FOREIGN KEY (ConsolePrimaria) references Console(NomeCons)

on update cascade

on delete cascade

);

Create table lingua

(

NomeLin varchar(50) primary key

)

Create table genere

(

NomeGen varchar(50) primary key

)

create table espansione

(

NomeEsp varchar(50) primary key,

Prezzo decimal(5,2) not null CHECK (Prezzo >=0 and Prezzo <=999.99),

Sconto tinyint not null DEFAULT 0 CHECK (Sconto >= 0 and Sconto <= 100),

Standalone varchar(3) not null check (Standalone='yes' or Standalone='no'),

Tipo varchar(50) not null check ( Tipo = 'std' or Tipo = 'obj'),

Videogioco varchar(50) not null,

Softwarehouse varchar(50) not null,

LinguaPrimaria varchar(50) not null,

GenerePrimario varchar(50) not null,

ConsolePrimaria varchar(50) not null,

FOREIGN KEY (Videogioco) references Videogioco(NomeVid)

on update cascade

on delete cascade,

FOREIGN KEY (Softwarehouse) references Softwarehouse(NomeSoft)

on update cascade

on delete cascade,

FOREIGN KEY (LinguaPrimaria) references Lingua(NomeLin)

on update cascade

on delete cascade,

FOREIGN KEY (GenerePrimario) references Genere(NomeGen)

on update cascade

on delete cascade,

FOREIGN KEY (ConsolePrimaria) references Console(NomeCons)

on update cascade

on delete cascade

);

create table linguavid

(

Videogioco varchar(50),

Lingua varchar(50),

FOREIGN KEY (Videogioco) references Videogioco(NomeVid)

on update cascade

on delete cascade,

FOREIGN KEY (Lingua) references Lingua(NomeLin)

on update cascade

on delete cascade,

primary key(Videogioco, Lingua)

);

create table linguaesp

(

Espansione varchar(50),

Lingua varchar(50),

FOREIGN KEY (Espansione) references Espansione(NomeEsp)

on update cascade

on delete cascade,

FOREIGN KEY (Lingua) references Lingua(NomeLin)

on update cascade

on delete cascade,

primary key(Espansione, Lingua)

);

create table genvid

(

Videogioco varchar(50),

Genere varchar(50),

FOREIGN KEY (Videogioco) references Videogioco(NomeVid)

on update cascade

on delete cascade,

FOREIGN KEY (Genere) references Genere(NomeGen)

on update cascade

on delete cascade,

primary key(Videogioco, Genere)

);

create table genesp

(

Espansione varchar(50),

Genere varchar(50),

FOREIGN KEY (Espansione) references Espansione(NomeEsp)

on update cascade

on delete cascade,

FOREIGN KEY (Genere) references Genere(NomeGen)

on update cascade

on delete cascade,

primary key(Espansione, Genere)

);

create table console

(

NomeCons varchar(50) primary key,

DataLancio date not null,

Modello varchar(50) not null,

Tipo varchar(50) not null check( Tipo='Casalinga' or Tipo='Portatile')

);

create table portatile

(

NomeCons varchar(50) primary key,

Schermo varchar(50) not null,

FOREIGN KEY (NomeCons) references Console(NomeCons)

on update cascade

on delete cascade

);

create table compatvid

(

Videogioco varchar(50),

Console varchar(50),

FOREIGN KEY (Videogioco) references Videogioco(NomeVid)

on update cascade

on delete cascade,

FOREIGN KEY (Console) references Console(NomeCons)

on update cascade

on delete cascade,

primary key (Videogioco, Console)

);

create table compatesp

(

Espansione varchar(50),

Console varchar(50),

FOREIGN KEY (Espansione) references Espansione(NomeEsp)

on update cascade

on delete cascade,

FOREING KEY (Console) references Console(NomeCons)

on update cascade

on delete cascade,

primary key (Espansione, Console)

);

create table offertamensile

(

Codice smallint primary key AUTO\_INCREMENT,

Mese enum ('January','February','March','April','May','June','July','August','September','October','November','December') not null,

Anno YEAR not null

);

create table CompListaPr

(

Videogioco varchar(50),

OffertaMensile smallint,

foreign key (Videogioco) references Videogioco(NomeVid)

on update cascade

on delete cascade,

foreign key (OffertaMensile) references OffertaMensile(Codice)

on update cascade

on delete cascade,

primary key (Videogioco, OffertaMensile)

);

create table CompListaStr

(

Videogioco varchar(50),

OffertaMensile smallint,

foreign key (Videogioco) references Videogioco(NomeVid)

on update cascade

on delete cascade,

foreign key (OffertaMensile) references OffertaMensile(Codice)

on update cascade

on delete cascade,

primary key (Videogioco, OffertaMensile)

);

create table ListaDesideri

(

Codice smallint primary key AUTO\_INCREMENT,

NomeLista varchar(50) not null,

Utente varchar(50) not null,

foreign key (Utente) references Utente(Nickname)

on update cascade

on delete cascade

)

create table CompListaEsp

(

ListaDesideri smallint,

Espansione varchar(50),

foreign key (ListaDesideri) references ListaDesideri(Codice)

on update cascade

on delete cascade,

foreign key (Espansione) references Espansione(NomeEsp)

on update cascade

on delete cascade,

primary key (ListaDesideri, Espansione)

);

create table CompListaVid

(

ListaDesideri smallint,

Videogioco varchar(50),

foreign key (ListaDesideri) references ListaDesideri(Codice)

on update cascade

on delete cascade,

foreign key (Videogioco) references Videogioco(NomeVid)

on update cascade

on delete cascade,

primary key (ListaDesideri, Videogioco)

);

create table Utente

(

Nickname varchar(50) primary key,

VipPoints smallint not null DEFAULT 0 check (VipPoints>=0),

Nazione varchar(50) not null

)

create table AcquistoVid

(

Utente varchar(50),

Videogioco varchar(50),

Prezzo decimal(5,2) not null DEFAULT 0 CHECK (Prezzo >=0 and Prezzo <=999.99),

Data date not null,

foreign key (Utente) references Utente(Nickname)

on update cascade

on delete cascade,

foreign key (Videogioco) references Videogioco(NomeVid)

on update cascade

on delete cascade,

primary key(Utente, Videogioco)

);

create table AcquistoEsp

(

Utente varchar(50),

Espansione varchar(50),

Prezzo decimal(5,2) not null CHECK (Prezzo >=0 and Prezzo <=999.99),

Data date not null,

foreign key (Utente) references Utente(Nickname)

on update cascade

on delete cascade,

foreign key (Espansione) references Espansione(NomeEsp)

on update cascade

on delete cascade,

primary key(Utente, Espansione)

);

create table RecensioneVid

(

Utente varchar(50),

Videogioco varchar(50),

TitoloRec varchar(50) not null,

Voto tinyint not null check(Voto>=1 and Voto <=10),

TestoRec varchar(500),

foreign key (Utente) references Utente(Nickname)

on update cascade

on delete cascade,

foreign key (Videogioco) references Videogioco(NomeVid)

on update cascade

on delete cascade,

primary key(Utente, Videogioco)

);

create table RecensioneEsp

(

Utente varchar(50),

Espansione varchar(50),

TitoloRec varchar(50) not null,

Voto tinyint not null check(Voto>=1 and Voto <=10),

TestoRec varchar(500),

foreign key (Utente) references Utente(Nickname)

on update cascade

on delete cascade,

foreign key (Espansione) references Espansione(NomeEsp)

on update cascade

on delete cascade,

primary key(Utente, Espansione)

)

create table Obiettivo

(

Codice int primary key AUTO\_INCREMENT,

NomeObj varchar(50) not null,

Videogioco varchar(50) not null,

Descrizione varchar(100) not null,

VipPoints smallint not null default 0 check (VipPoints>=0),

Tipo varchar(50) not null check ( Tipo = 'std' or Tipo = 'gold'),

foreign key (Videogioco) references Videogioco(NomeVid)

on update cascade

on delete cascade

);

create table SbloccoObj

(

Utente varchar(50),

Obiettivo int,

DataSblocco date not null,

foreign key (Utente) references Utente(Nickname)

on update cascade

on delete cascade,

foreign key (Obiettivo) references Obiettivo(Codice)

on update cascade

on delete cascade,

primary key(Utente, Obiettivo)

);

create table Servizio

(

NomeServ varchar(50) primary key,

Descrizione varchar(500) not null,

PianoBase tinyint not null,

foreign key (PianoBase) references Piano(Codice)

on update cascade

on delete no action

)

create table Piano

(

Codice tinyint primary key AUTO\_INCREMENT,

Prezzo decimal(5,2) not null CHECK (Prezzo >=0 and Prezzo <=999.99),

Periodo tinyint not null check(Periodo>=1 and Periodo<=12)

)

create table Abbonamento

(

Codice smallint primary key AUTO\_INCREMENT,

Utente varchar(50) not null,

Servizio varchar(50) not null,

Piano tinyint not null,

DataInizio date not null,

foreign key (Utente) references Utente(Nickname)

on update cascade

on delete cascade,

foreign key (Servizio) references Servizio(NomeServ)

on update cascade

on delete no action,

foreign key (Piano) references Piano(Codice)

on update cascade

on delete no action

);

create table PianoServ

(

Piano tinyint,

Servizio varchar(50),

foreign key (Piano) references Piano(Codice)

on update cascade

on delete cascade,

foreign key (Servizio) references Servizio(NomeServ)

on update cascade

on delete cascade,

primary key (Piano, Servizio)

)

# 2. Definizione dei vincoli

Dobbiamo definire i vincoli di cardinalità tra entità e associazioni, le regole aziendali per lo schema e i vincoli aggiuntivi derivati dalle fasi di ristrutturazione.

## 2.1 Vincoli di cardinalità entità-associazioni

**Vincoli di cardinalità per l’associazione CompListaPr**

Poiché un’occorrenza di OffertaMensile deve essere associata con esattamente 4 occorrenze di CompListaPr l’idea è quella di creare queste occorrenze contestualmente alla creazione di una occorrenza di OffertaMensile per ottenere la cardinalità minima necessaria. Useremo un trigger su OffertaMensile che, nel momento dell’inserimento di un’occorrenza, crea 4 occorrenze in CompListaPr inserendo casualmente dei videogiochi che non siano già presenti. Per la cardinalità massima invece usiamo un trigger in CompListaPr. L’operazione DELETE su CompListaPr non verrà consentita tramite trigger. Le operazioni di modifica saranno consentite per una maggiore libertà di scelta dei videogiochi da associare alle offerte mensili.

**Partecipazione di OffertaMensile a CompListaPr: cardinalità minima = 4**

delimiter //

CREATE TRIGGER VC\_OffertaMensile\_CompListaPr\_CMin

AFTER INSERT ON offertamensile

FOR EACH ROW

BEGIN

set @temp = 0;

set @vid = '';

set @var = false;

WHILE @temp < 4 DO

set @vid = (

SELECT NomeVid

FROM Videogioco

WHERE NomeVid not in (select Videogioco from complistapr)

ORDER BY RAND()

LIMIT 1

);

insert into complistapr(Videogioco, OffertaMensile) VALUES (@vid, new.Codice);

set @temp = @temp + 1;

set @vid = '';

END WHILE;

END //

**Vincolo su CompListaPr per controllo cardinalità massime**

Controlliamo le cardinalità massime di entrambi i ruoli di CompListaPr

delimiter //

CREATE TRIGGER VC\_CompListaPr\_CMax

BEFORE insert ON complistapr

FOR EACH ROW

BEGIN

IF ( (select count(\*) from complistapr where OffertaMensile = new.OffertaMensile) = 4

OR (select count(\*) from complistapr where Videogioco = new.Videogioco) = 1) THEN

signal sqlstate '02000' set message\_text = “cardinalita massima violata!”;

END IF;

END//

**Vincolo su CompListaPr: operazione di delete non concessa**

Non permettiamo operazioni di delete su complistapr, in ogni dato momento la cardinalità deve essere sempre esattamente 4.

# VC\_CompListaPr no delete

delimiter //

CREATE TRIGGER VC\_CompListaPr\_deleteCheck

BEFORE delete ON complistapr

FOR EACH ROW

BEGIN

signal sqlstate '02000' set message\_text = “Non è possibilie eliminare valori da complistapr”;

END//

**Vincolo su CompListaPr: operazione di update**

# VC\_CompListaPr check update

delimiter //

CREATE TRIGGER VC\_CompListaPr\_updateCheck

BEFORE update ON complistapr

FOR EACH ROW

BEGIN

IF ( (new.OffertaMensile = old.OffertaMensile) AND

( new.Videogioco in (select Videogioco from complistapr) ) )

THEN

signal sqlstate '02000' set message\_text = “cardinalità massima violata!il videogioco è già nelle offerte!”;

END IF;

IF ( (new.OffertaMensile != old.OffertaMensile) AND

(new.OffertaMensile in (select OffertaMensile from complistapr where OffertaMEnsile = new.OffertaMensile) ) )

THEN

signal sqlstate '02000' set message\_text = “cardinalità massima violata!Le offerte mensili devono essere esattamente 4!”;

END IF;

END//

**Vincoli di cardinalità per l’associazione CompListaStr**

Stesso criterio del vincolo di cardinalità per l’associazione CompListaPr. Qui dobbiamo fare attenzione al fatto che un videogioco può tornare a far parte dell’offerta anche in mesi successivi.

**Partecipazione di OffertaMensile a CompListaStr cardinalità minima = 20**

# partecipazione di OffertaMensile a CompListaStr cardinalità minima = 20

delimiter //

CREATE TRIGGER VC\_OffertaMensile\_CompListaStr\_CMin

AFTER INSERT ON offertamensile

FOR EACH ROW

BEGIN

set @cont = 0;

set @vid = '';

set @var = false;

set @nvid = 0;

WHILE @cont < 20 DO

set @var = false;

WHILE @var = false DO

#trova un videogioco casuale dalla tabella Videogioco

set @vid = (

SELECT NomeVid

FROM Videogioco

where NomeVid not in (select Videogioco from complistastr where OffertaMensile = new.Codice)

ORDER BY RAND()

LIMIT 1

);

#conta quante volte questo videogioco è stato inserito nelle offerte mensili streaming

set @nvid = (

select count(\*)

from complistastr as c

where c.Videogioco = @vid

);

IF @nvid < 12 THEN

set @var = true;

insert into complistastr(Videogioco, OffertaMensile) VALUES (@vid, new.Codice);

set @cont = @cont + 1;

set @vid = '';

set @nvid = 0;

end if;

END WHILE;

END WHILE;

END //

**Partecipazione di OffertaMensile a CompListaStr: cardinalità massima = 20**

delimiter //

CREATE TRIGGER VC\_OffertaMensile\_CompListaStr\_CMax

BEFORE insert ON complistastr

FOR EACH ROW

BEGIN

IF ( (select count(\*) from complistastr where Videogioco = new.Videogioco) = 12

OR

( (select count(\*) from complistastr where OffertaMensile = new.Offertamensile) = 20))

THEN

signal sqlstate '02000' set message\_text = “cardinalità massima violata!”;

END IF;

END//

**Vincolo su CompListaStr: operazione di delete non concessa**

Non permettiamo operazioni di delete su complistastr, in ogni dato momento la cardinalità deve essere sempre esattamente 20 per la partecipazione di OffertaMensile a CompListaStr.

delimiter //

CREATE TRIGGER VC\_CompListaStr\_deleteCheck

BEFORE delete ON complistastr

FOR EACH ROW

BEGIN

signal sqlstate '02000' set message\_text = “Non è possibilie eliminare valori da complistastr!”;

END//

**Vincolo su CompListaStr: operazione di update**

delimiter //

CREATE TRIGGER VC\_CompListaStr\_updateCheck

BEFORE update ON complistastr

FOR EACH ROW

BEGIN

IF (new.OffertaMensile != old.OffertaMensile AND (new.OffertaMensile in (select OffertaMensile from complistapr where OffertaMEnsile = new.OffertaMensile) ) ) THEN

signal sqlstate '02000' set message\_text = “cardinalità massima violata!Le offerte mensili devono essere esattamente 20!”;

END IF;

END//

**Vincoli di cardinalità per l’associazione AcquistoVid**

**Cardinalità massime:**

delimiter //

CREATE TRIGGER VC\_AcquistoVid\_CMax

BEFORE insert ON acquistovid

FOR EACH ROW

BEGIN

IF ( (select count(\*) from acquistovid where Videogioco = new.Videogioco) = 10E+6

OR

(select count(\*) from acquistovid where Utente = new.Utente) = 320

) THEN

signal sqlstate '02000' set message\_text = “cardinalita massima violata!”;

END IF;

END//

L’operazione di cancellazione è consentita senza vincoli, ipotizziamo possa essere utile nel caso di un ipotetico rimborso oppure per un ban di un utente. L’operazione di modifica non deve essere consentita, non dovrebbe essere possibile assegnare un acquisto di un utente ad un altro utente né tantomeno assegnare un altro prodotto al posto di uno precedentemente acquistato. Lo stesso discorso è applicato all’associazione AcquistoEsp

delimiter //

CREATE TRIGGER VC\_AcquistoVid\_update

BEFORE update ON acquistovid

FOR EACH ROW

BEGIN

signal sqlstate '02000' set message\_text = “Modifiche su AcquistoVid non consentite!”;

END//

**Vincoli di cardinalità per l’associazione AcquistoEsp**

delimiter //

CREATE TRIGGER VC\_AcquistoEsp\_CMax

BEFORE insert ON acquistoesp

FOR EACH ROW

BEGIN

IF ( (select count(\*) from acquistoesp where Espansione = new.Espansione) = 10E+6

OR

(select count(\*) from acquistoesp where Utente = new.Utente) = 80

) THEN

signal sqlstate '02000' set message\_text = “cardinalita massima violata!”;

END IF;

END//

delimiter //

CREATE TRIGGER VC\_AcquistoEsp\_update

BEFORE update ON acquistoesp

FOR EACH ROW

BEGIN

signal sqlstate '02000' set message\_text = “Modifiche su AcquistoVid non consentite!”;

END//

**Vincoli di cardinalità per l’associazione CompListaVid e CompListaEsp**

**Cardinalità massime:** vedere [“Vincoli derivati da ristrutturazione”](#_2.3._Regole_di)

Per le operazioni di update non deve essere consentito di cambiare il codice di una Lista desideri nell’associazione CompListaVid poiché si rischierebbe di assegnare il contenuto di una lista dei desideri di un utente alla lista dei desideri di un altro utente. La modifica del valore di un videogioco è consentita. Lo stesso ragionamento è applicato all’associazione CompListaEsp

delimiter //

CREATE TRIGGER VC\_CompListaVid\_update

BEFORE update ON complistavid

FOR EACH ROW

BEGIN

IF( old.ListaDesideri != new.ListaDesideri) THEN

signal sqlstate '02000' set message\_text = “Modifiche su ListaDesideri di CompListaVid non consentite!”;

END IF;

END//

delimiter //

CREATE TRIGGER VC\_CompListaEsp\_update

BEFORE update ON complistaesp

FOR EACH ROW

BEGIN

IF( old.ListaDesideri != new.ListaDesideri) THEN

signal sqlstate '02000' set message\_text = “Modifiche su ListaDesideri di CompListaEsp non consentite!”;

END IF;

END//

**Vincoli di cardinalità per l’associazione SbloccoObj**

Cardinalità massime:

delimiter //

CREATE TRIGGER VC\_SbloccoObj\_CMax

BEFORE insert ON sbloccoobj

FOR EACH ROW

BEGIN

IF ( (select count(\*) from sbloccoobj where Utente = new.Utente) = 3200

OR

(select count(\*) from sbloccoobj where Obiettivo = new.Obiettivo) = 10E+6

) THEN

signal sqlstate '02000' set message\_text = “cardinalita massima violata!”;

END IF;

END//

Le operazioni di modifica non sono consentite, si rischierebbe di assegnare obiettivi sbloccati da un utente ad altri utenti oppure di assegnare un obiettivo ad un utente diverso da quello che ha realmente sbloccato. Le operazioni di cancellazione sono consentite senza vincoli.

delimiter //

CREATE TRIGGER VC\_SbloccoObj\_update

BEFORE update ON sbloccoobj

FOR EACH ROW

BEGIN

signal sqlstate '02000' set message\_text = “Modifiche su sbloccoobj non consentite!”;

END//

**Vincoli di cardinalità per la partecipazione di Videogioco alle associazioni LinguaVid, GenVid e CompatVid**

**Vincolo per Videogioco: creazione cardinalità minima per la partecipazione a LinguaVid, GenVid e CompatVid**

delimiter //

CREATE TRIGGER VC\_Videogioco\_associazioni\_1\_N\_CMin

AFTER insert ON videogioco

FOR EACH ROW

BEGIN

Insert into linguavid VALUES (new.NomeVid, new.LinguaPrimaria);

Insert into genvid VALUES (new.NomeVid, new.GenerePrimario);

Insert into compatvid VALUES (new.NomeVid, new.ConsolePrimaria);

END//

**Vincoli di cardinalità LinguaVid**

**Cardinalità massime:**

delimiter //

CREATE TRIGGER VC\_LinguaVid\_CMax

BEFORE insert ON linguavid

FOR EACH ROW

BEGIN

IF ( (select count(\*) from linguavid where Videogioco = new.Videogioco) = 10

OR

(select count(\*) from linguavid where Lingua = new.Lingua) = 320

) THEN

signal sqlstate '02000' set message\_text = 'cardinalita massima violata!';

END IF;

END//

Per garantire la cardinalità minima = 1 sul ruolo Videogioco dobbiamo evitare che venga cancellata l’occorrenza con il valore di LinguaPrimaria creata al momento della creazione di una occorrenza di Videogioco. L’operazione di update non viene concessa. Lo stesso criterio è applicato alle associazioni LinguaEsp, GenVid, GenEsp, CompatEsp e CompatVid

**Operazione delete:**

delimiter //

CREATE TRIGGER VC\_LinguaVid\_delete

BEFORE delete ON linguavid

FOR EACH ROW

BEGIN

#Se la lingua dell’occorrenza che si vuole cancellare corrisponde alla lingua primaria del

#videogioco allora non concedere il delete

IF ( old.Lingua = (select LinguaPrimaria from Videogioco where NomeVid = old.Videogioco) ) THEN

signal sqlstate '02000' set message\_text = 'Non è possibile eliminare la lingua primaria!';

END IF;

END//

**Operazione update:**

delimiter //

CREATE TRIGGER VC\_LinguaVid\_update

BEFORE update ON linguavid

FOR EACH ROW

BEGIN

signal sqlstate '02000' set message\_text = 'Update su linguavid non concesso!';

END//

**Vincoli di cardinalità GenVid**

**Cardinalità massime**

delimiter //

CREATE TRIGGER VC\_GenVid\_CMax

BEFORE insert ON genvid

FOR EACH ROW

BEGIN

IF ( (select count(\*) from genvid where Videogioco = new.Videogioco) = 5

OR

(select count(\*) from genvid where Genere = new.Genere) = 320

) THEN

signal sqlstate '02000' set message\_text = 'cardinalita massima violata!';

END IF;

END//

**Operazione delete:**

delimiter //

CREATE TRIGGER VC\_GenVid\_delete

BEFORE delete ON genvid

FOR EACH ROW

BEGIN

IF ( old.Genere = (select GenerePrimario from Videogioco where NomeVid = old.Videogioco) ) THEN

signal sqlstate '02000' set message\_text = 'Non è possibile eliminare il genere primario!';

END IF;

END//

**Operazione update:**

delimiter //

CREATE TRIGGER VC\_GenVid\_update

BEFORE update ON genvid

FOR EACH ROW

BEGIN

signal sqlstate '02000' set message\_text = 'Update su genvid non concesso!';

END//

**Vincoli di cardinalità CompatVid**

**Cardinalità massime**

delimiter //

CREATE TRIGGER VC\_CompatVid\_CMax

BEFORE insert ON compatvid

FOR EACH ROW

BEGIN

IF ( (select count(\*) from compatvid where Videogioco = new.Videogioco) = 4

OR

(select count(\*) from compatvid where Console = new.Console) = 320

) THEN

signal sqlstate '02000' set message\_text = 'cardinalita massima violata!';

END IF;

END//

**Operazione delete:**

delimiter //

CREATE TRIGGER VC\_CompatVid\_delete

BEFORE delete ON compatvid

FOR EACH ROW

BEGIN

IF ( old.Console = (select ConsolePrimaria from Videogioco where NomeVid = old.Videogioco) ) THEN

signal sqlstate '02000' set message\_text = 'Non è possibile eliminare la console primaria!';

END IF;

END//

**Operazione update:**

delimiter //

CREATE TRIGGER VC\_CompatVid\_update

BEFORE update ON compatvid

FOR EACH ROW

BEGIN

signal sqlstate '02000' set message\_text = 'Update su compatvid non concesso!';

END//

**Vincoli di cardinalità per la partecipazione di Espansione alle associazioni LinguaEsp, GenEsp e CompatEsp**

**Vincolo per Espansione: creazione cardinalità minima per partecipazione a LinguaVid, GenVid e CompatVid**

delimiter //

CREATE TRIGGER VC\_Espansione\_associazioni\_1\_N\_CMin

AFTER insert ON espansione

FOR EACH ROW

BEGIN

Insert into linguaesp VALUES (new.NomeEsp, new.LinguaPrimaria);

Insert into genesp VALUES (new.NomeEsp, new.GenerePrimario);

Insert into compaesp VALUES (new.NomeEsp, new.ConsolePrimaria);

END//

**Vincoli di cardinalità LinguaEsp**

**Cardinalità massime**

delimiter //

CREATE TRIGGER VC\_LinguaEsp\_CMax

BEFORE insert ON linguaesp

FOR EACH ROW

BEGIN

IF ( (select count(\*) from linguaesp where Espansione = new.Espansione) = 10

OR

(select count(\*) from linguaesp where Lingua = new.Lingua) = 80

) THEN

signal sqlstate '02000' set message\_text = 'cardinalita massima violata!';

END IF;

END//

**Operazione delete:**

delimiter //

CREATE TRIGGER VC\_linguaesp\_delete

BEFORE delete ON linguaesp

FOR EACH ROW

BEGIN

IF ( old.Lingua = (select LinguaPrimaria from Espansione where NomeEsp = old.Espansione) ) THEN

signal sqlstate '02000' set message\_text = 'Non è possibile eliminare la lingua primaria!';

END IF;

END//

**Operazione Update:**

delimiter //

CREATE TRIGGER VC\_linguaesp\_update

BEFORE update ON linguaesp

FOR EACH ROW

BEGIN

signal sqlstate '02000' set message\_text = 'Update su linguaesp non concesso!';

END//

**Vincoli di cardinalità GenEsp**

**Cardinalità massime**

delimiter //

CREATE TRIGGER VC\_GenEsp\_CMax

BEFORE insert ON genesp

FOR EACH ROW

BEGIN

IF ( (select count(\*) from genesp where Espansione = new.Espansione) = 5

OR

(select count(\*) from genesp where Genere = new.Genere) = 80

) THEN

signal sqlstate '02000' set message\_text = 'cardinalita massima violata!';

END IF;

END//

**Operazione delete:**

delimiter //

CREATE TRIGGER VC\_GenEsp\_delete

BEFORE delete ON genesp

FOR EACH ROW

BEGIN

IF ( old.Genere = (select GenerePrimario from Espansione where NomeEsp = old.Espansione) ) THEN

signal sqlstate '02000' set message\_text = ' Non è possibile eliminare il genere primario!';

END IF;

END//

**Operazione update:**

delimiter //

CREATE TRIGGER VC\_genesp\_update

BEFORE update ON genesp

FOR EACH ROW

BEGIN

signal sqlstate '02000' set message\_text = 'Update su genesp non concesso!';

END//

**Vincoli di cardinalità CompatEsp**

**Cardinalità massime**

delimiter //

CREATE TRIGGER VC\_CompatEsp\_CMax

BEFORE insert ON compatesp

FOR EACH ROW

BEGIN

IF ( (select count(\*) from compatvid where Espansione = new. Espansione) = 4

OR

(select count(\*) from compatvid where Console = new.Console) = 80

) THEN

signal sqlstate '02000' set message\_text = 'cardinalita massima violata!';

END IF;

END//

**Operazione delete:**

delimiter //

CREATE TRIGGER VC\_CompatEsp\_delete

BEFORE delete ON compatesp

FOR EACH ROW

BEGIN

IF ( old.Console = (select ConsolePrimaria from Espansione where NomeEsp = old.Espansione) ) THEN

signal sqlstate '02000' set message\_text = 'Non è possibile eliminare la console primaria!';

END IF;

END//

**Operazione update:**

delimiter //

CREATE TRIGGER VC\_CompatVid\_update

BEFORE update ON compatvid

FOR EACH ROW

BEGIN

signal sqlstate '02000' set message\_text = 'Update su compatvid non concesso!';

END//

**Vincoli di cardinalità per l’associazione PianoServ:**

**Cardinalità minima per ruolo Servizio:**

delimiter //

CREATE TRIGGER VC\_Servizio\_PianoServ\_CMin

AFTER insert ON servizio

FOR EACH ROW

BEGIN

Insert into pianoserv VALUES (new.NomeServ, new.PianoBase);

END//

**Cardinalità massime:**

delimiter //

CREATE TRIGGER VC\_PianoServ\_CMax

BEFORE insert ON pianoserv

FOR EACH ROW

BEGIN

IF ( (select count(\*) from pianoserv where Servizio = new.Servizio) = 3

OR

(select count(\*) from pianoserv where Piano = new.Piano) = 2

) THEN

signal sqlstate '02000' set message\_text = 'cardinalita massima violata!';

END IF;

END//

**Operazione di delete:** dobbiamo garantire la cardinalità minima con il PianoBase

delimiter //

CREATE TRIGGER VC\_PianoServ\_delete

BEFORE delete ON pianoserv

FOR EACH ROW

BEGIN

IF ( old.Piano = (select PianoBase from Servizio where NomeServ = old.Servizio) ) THEN

signal sqlstate '02000' set message\_text = 'Non è possibile eliminare il piano primario!';

END IF;

END//

**Operazione update:**

delimiter //

CREATE TRIGGER VC\_PianoServ\_update

BEFORE update ON pianoserv

FOR EACH ROW

BEGIN

signal sqlstate '02000' set message\_text = 'Update su pianoserv non concesso!';

END//

**Vincoli di cardinalità per l’associazione SviluppoVid**

L’ associazione SviluppoVid è stata accorpata nella relazione Videogioco come risultato della traduzione al modello relazionale, controlliamo quindi la cardinalità massima del ruolo Softwarehouse direttamente nella relazione Videogioco. Lo stesso criterio verrà applicato alle altre associazioni interessate dalla stessa tipologia.

delimiter //

CREATE TRIGGER VC\_SviluppoVid\_CMax

BEFORE insert ON videogioco

FOR EACH ROW

BEGIN

IF ( (select count(\*) from videogioco where Softwarehouse = new.Softwarehouse) = 10 ) THEN

signal sqlstate '02000' set message\_text = 'cardinalita massima delle softwarehouse violata!';

END IF;

END//

**Vincoli di cardinalità per l’associazione SviluppoEsp**

delimiter //

CREATE TRIGGER VC\_SviluppoEsp\_CMax

BEFORE insert ON espansione

FOR EACH ROW

BEGIN

IF ( (select count(\*) from espansione where Softwarehouse = new.Softwarehouse) = 10 ) THEN

signal sqlstate '02000' set message\_text = 'cardinalita massima delle softwarehouse violata!';

END IF;

END//

**Vincoli di cardinalità per l’associazione CompEsp**

delimiter //

CREATE TRIGGER VC\_CompEsp\_CMax

BEFORE insert ON espansione

FOR EACH ROW

BEGIN

IF ( (select count(\*) from espansione where Videogioco = new.Videogioco) = 10 ) THEN

signal sqlstate '02000' set message\_text = 'cardinalita massima dei videogiochi violata!';

END IF;

END//

**Vincoli di cardinalità per l’associazione Lista**

delimiter //

CREATE TRIGGER VC\_Lista\_CMax

BEFORE insert ON listadesideri

FOR EACH ROW

BEGIN

IF ( (select count(\*) from listadesideri where Utente = new.Utente) = 10 ) THEN

signal sqlstate '02000' set message\_text = 'cardinalita massima violata per utenti!';

END IF;

END//

**Vincoli di cardinalità per l’associazione FeedbackVid e RecVid**

delimiter //

CREATE TRIGGER VC\_FeedbackVid\_CMax

BEFORE insert ON recensionevid

FOR EACH ROW

BEGIN

IF ( (select count(\*) from recensionevid where Utente = new.Utente) = 320

OR ((select count(\*) from recensionevid where Videogioco = new.Videogioco) = 10E+6) ) THEN

signal sqlstate '02000' set message\_text = 'cardinalita massima violata!';

END IF;

END//

**Vincoli di cardinalità per le associazioni FeedbackEsp e RecEsp**

delimiter //

CREATE TRIGGER VC\_FeedbackEsp\_CMax

BEFORE insert ON recensioneesp

FOR EACH ROW

BEGIN

IF ( (select count(\*) from recensioneesp where Utente = new.Utente) = 80

OR ((select count(\*) from recensioneesp where Espansione = new. Espansione) = 10E+6) ) THEN

signal sqlstate '02000' set message\_text = 'cardinalita massima violata!';

END IF;

END//

**Vincoli di cardinalità per le associazioni Sottoscrizione, ServizioAbb e AbbPiano**

**Cardinalità massime:**

delimiter //

CREATE TRIGGER VC\_Abbonamento\_CMax

BEFORE insert ON abbonamento

FOR EACH ROW

BEGIN

IF ( (select count(\*) from abbonamento where Utente = new.Utente) = 24

OR ((select count(\*) from abbonamento where Servizio = new.Servizio) = 20E+6)

OR ((select count(\*) from abbonamento where Piano = new.Piano) = 6E+6)

) THEN

signal sqlstate '02000' set message\_text = 'cardinalita massima violata!';

END IF;

END//

**Vincoli di cardinalità per l’associazione CompObj**

**Cardinalità massima:**

delimiter //

CREATE TRIGGER VC\_CompObj\_CMax

BEFORE insert ON obiettivo

FOR EACH ROW

BEGIN

IF ( (select count(\*) from obiettivo where Videogioco = new.Videogioco) = 20 ) THEN

signal sqlstate '02000' set message\_text = 'cardinalita massima violata!';

END IF;

END//

## 2.2. Business rules

|  |
| --- |
| **Regole di vincolo** |
| (RV1) Un utente deve possedere un contenuto multimediale per poterlo recensire |
| (RV2) Un utente deve possedere un contenuto multimediale per poter sbloccare gli obiettivi |
| (RV3) L’utente che sia abbonato Premium deve poter acquistare i prodotti presenti nell’offerta a costo zero |
| **Regole di derivazione** |
| (RD1) Il prezzo reale di un gioco si ottiene sottraendo la percentuale dello sconto al prezzo del contenuto multimediale |
| (RD2) Il punteggio VipPoints dell’utente si ottiene sommando i VipPoints di ogni ObiettivoGold sbloccato |

**(RV1) Un utente deve possedere un contenuto multimediale per poterlo recensire**

delimiter //

CREATE TRIGGER RV1\_videogioco

BEFORE insert ON recensionevid

FOR EACH ROW

BEGIN

IF ( new.Utente not in (select Utente from AcquistoVid where Utente = new.Utente and Videogioco = new.Videogioco) ) THEN

signal sqlstate '02000' set message\_text = 'L’utente non possiede il videogioco!';

END IF;

END//

delimiter //

CREATE TRIGGER RV1\_espansione

BEFORE insert ON recensioneesp

FOR EACH ROW

BEGIN

IF ( new.Utente not in (select Utente from AcquistoEsp where Utente = new.Utente and Espansione = new.Espansione) ) THEN

signal sqlstate '02000' set message\_text = 'L’utente non possiede l’espansione!';

END IF;

END//

**(RV2) Un utente deve possedere un videogioco per poterne sbloccare gli obiettivi**

delimiter //

CREATE TRIGGER RV2\_videogioco

BEFORE insert ON sbloccoobj

FOR EACH ROW

BEGIN

IF ( new.Utente not in (select Utente from AcquistoVid

where Utente = new.Utente and

Videogioco = ( select Videogioco

from Obiettivo

where Codice = new.Obiettivo)

)

) THEN

signal sqlstate '02000' set message\_text = 'L’utente non possiede il videogioco!';

END IF;

END//

**(RV3) e (RD1) possono essere implementati nello stesso trigger**

**(RV3) L’utente che sia abbonato Premium deve poter acquistare i prodotti presenti nell’offerta a costo (RD1) Il prezzo reale di un gioco si ottiene sottraendo la percentuale dello sconto al prezzo del contenuto multimediale**

delimiter //

CREATE TRIGGER RV3\_RD1\_AcquistoVid\_prezzo

BEFORE insert ON acquistovid

FOR EACH ROW

BEGIN

# controllo se l’utente è attualmente abbonato al servizio premium e che il videogioco sia nell’offerta # mensile corrente

If ( new.Utente in ( SELECT Utente

from abbonamento

where Servizio = 'premium'

and

CURDATE() <= ( select ( DATE\_ADD( DataInizio, INTERVAL (select periodo

from piano

where Codice = Abbonamento.Piano) MONTH) - INTERVAL 1 DAY) ) )

AND

New.Videogioco in ( select Videogioco from complistapr where OffertaMensile = (select Codice from offertamensile where Mese = MONTHNAME(CURDATE()) AND YEAR(CURDATE()) ))

) THEN

SET new.Prezzo = 0;

ELSE

# (RD1)

SET @sconto = (select Sconto from Videogioco where NomeVid = new.Videogioco);

SET @prezzoListino = (select Prezzo from Videogioco where NomeVid = new.Videogioco);

SET new.Prezzo = @prezzoListino - (@prezzoListino \* (@sconto/100));

END IF;

END//

**(RD2) Il punteggio VipPoints dell’utente si ottiene sommando i VipPoints di ogni ObiettivoGold sbloccato**

delimiter //

CREATE TRIGGER RD2\_sbloccoobj

AFTER insert ON sbloccoobj

FOR EACH ROW

BEGIN

SET @vippoints = (select VipPoints from Utente where Nickname = new.Utente) +

(Select VipPoints from Obiettivo where Codice = new.Obiettivo);

update Utente

set VipPoints = @vippoints

where Nickname = new.Utente;

END//

## 2.3. Regole di vincolo derivate dalle fasi di ristrutturazione

|  |
| --- |
| **Regole di vincolo derivate dalle fasi di ristrutturazione** |
| (RR1) Un utente può memorizzare fino a 20 contenuti in una lista desideri |
| (RR2) Non possono esistere due obiettivi con lo stesso nome che siano associati allo stesso videogioco |
| (RR3) Non possono esistere due liste desideri aventi lo stesso nome per lo stesso utente |
| (RR4) Non possono esistere due periodi che abbiano lo stesso prezzo per un piano tariffario |
| (RR5) Non possono esistere due stessi mesi nello stesso anno per una offerta mensile |
| (RR6) Un utente non può sottoscrivere uno stesso servizio nella stessa data |
| (RR7) Solo gli obiettivi che siano di tipo “gold” possono avere un punteggio VipPoints diverso da zero |

**(RR1) Un utente può memorizzare fino a 20 contenuti in una lista desideri**

Creiamo una stored procedure poiché dobbiamo fare lo stesso tipo di controllo per CompListaVid e CompListaEsp

delimiter //

CREATE PROCEDURE limit20(IN lista smallint)

BEGIN

SET @cont = 0;

SET @cont = (select count(\*) from complistavid where ListaDesideri = lista) +

(select count(\*) from complistaesp where ListaDesideri = lista);

IF ( @cont = 20 )

THEN

signal sqlstate '02000' set message\_text = 'Capienza massima lista desideri raggiunta!';

END IF;

END//

Quindi creiamo i due trigger di controllo:

delimiter //

CREATE TRIGGER RR1\_CompListaVid

BEFORE INSERT ON complistavid

FOR EACH ROW

BEGIN

CALL limit20(new.ListaDesideri);

END;//

delimiter //

CREATE TRIGGER RR1\_CompListaEsp

BEFORE INSERT ON complistaesp

FOR EACH ROW

BEGIN

CALL limit20(new.ListaDesideri);

END;//

**(RR2) Non possono esistere due obiettivi con lo stesso nome che siano associati allo stesso videogioco**

delimiter //

CREATE PROCEDURE obj\_x\_game(IN nomeObj varchar(50), IN videogioco varchar(50))

BEGIN

IF ( exists ( select \* from obiettivo as obj where (obj.NomeObj = nomeObj) and (obj.Videogioco = videogioco) )) THEN

signal sqlstate '02000' set message\_text = “Non possono esserci due obbiettivi aventi stesso nome per uno stesso videogioco!”;

END IF;

END//

delimiter //

CREATE TRIGGER RR2\_obiettivo\_insert

BEFORE INSERT ON obiettivo

FOR EACH ROW

BEGIN

CALL obj\_x\_game (new.NomeObj, new.Videogioco);

END;//

delimiter //

CREATE TRIGGER RR2\_obiettivo\_update

BEFORE UPDATE ON obiettivo

FOR EACH ROW

BEGIN

CALL obj\_x\_game (new.NomeObj, new.Videogioco);

END;//

**(RR3) Non possono esistere due liste desideri aventi lo stesso nome per lo stesso utente**

delimiter //

CREATE PROCEDURE no2listname (IN list varchar(50), IN user varchar(50))

BEGIN

IF ( exists ( select \* from listadesideri where (NomeLista = list) and (Utente = user) )) THEN

signal sqlstate '02000' set message\_text = 'Non possono esserci due liste desideri aventi stesso nome per uno stesso utente!';

END IF;

END//

delimiter //

CREATE TRIGGER RR3\_obiettivo\_insert

BEFORE INSERT ON listadesideri

FOR EACH ROW

BEGIN

CALL no2listname(new.NomeLista, new.Utente);

END;//

delimiter //

CREATE TRIGGER RR3\_obiettivo\_update

BEFORE UPDATE ON listadesideri

FOR EACH ROW

BEGIN

CALL no2listname(new.NomeLista, new.Utente);

END;//

**(RR4) Non possono esistere due periodi che abbiano lo stesso prezzo per un piano tariffario**

delimiter //

CREATE PROCEDURE no2period (IN period tinyint, IN price decimal(5,2))

BEGIN

IF ( exists ( select \* from piano where (Periodo = period) and (Prezzo = price) )) THEN

signal sqlstate '02000' set message\_text = 'Non possono esserci due periodi aventi stesso prezzo per uno stesso piano!';

END IF;

END//

delimiter //

CREATE TRIGGER RR4\_piano\_insert

BEFORE INSERT ON piano

FOR EACH ROW

BEGIN

CALL no2period (new.Periodo, new.Prezzo);

END;//

delimiter //

CREATE TRIGGER RR4\_piano\_update

BEFORE UPDATE ON piano

FOR EACH ROW

BEGIN

CALL no2period (new.Periodo, new.Prezzo);

END;//

**(RR5) Non possono esistere due stessi mesi nello stesso anno per una offerta mensile**

delimiter //

CREATE PROCEDURE no2months\_year (IN mesep varchar(50), IN annop year)

BEGIN

IF ( exists ( select \* from offertamensile where (Mese = mesep) and (Anno = annop) )) THEN

signal sqlstate '02000' set message\_text = 'Non esistono due stessi mesi nello stesso anno';

END IF;

END//

delimiter //

CREATE TRIGGER RR5\_offertamensile\_insert

BEFORE INSERT ON offertamensile

FOR EACH ROW

BEGIN

CALL no2months\_year (new.Mese, new.Anno);

END;//

delimiter //

CREATE TRIGGER RR4\_offertamensile\_update

BEFORE UPDATE ON offertamensile

FOR EACH ROW

BEGIN

CALL no2months\_year (new.Mese, new.Anno);

END;//

**(RR6) Un utente non può sottoscrivere uno stesso servizio nella stessa data**

delimiter //

CREATE PROCEDURE no2serv\_user (IN serviziop varchar(50), IN utentep varchar(50), IN datap DATE)

BEGIN

IF ( exists ( select \* from abbonamento where (Servizio = serviziop) and (Utente = utentep) and (DataInizio = datap))) THEN

signal sqlstate '02000' set message\_text = ‘Un utente non può sottoscrivere uno stesso servizio nella stessa data!';

END IF;

END//

delimiter //

CREATE TRIGGER RR6\_abbonamento\_insert

BEFORE INSERT ON abbonamento

FOR EACH ROW

BEGIN

CALL no2serv\_user (new.Servizio, new.Utente, new.DataInizio);

END;//

delimiter //

CREATE TRIGGER RR6\_abbonamento\_update

BEFORE UPDATE ON abbonamento

FOR EACH ROW

BEGIN

CALL no2serv\_user (new.Servizio, new.Utente, new.DataInizio);

END;//

**(RR7) Solo gli obiettivi che siano di tipo “gold” possono avere un punteggio VipPoints diverso da zero**

delimiter //

CREATE PROCEDURE noGoldNoPoints(IN tipo varchar(50), IN vipPoints smallint)

BEGIN

IF ( tipo = “std” and vipPoints > 0) THEN

signal sqlstate '02000' set message\_text = “Solo gli obiettivi gold hanno VipPoints!”;

END IF;

END//

delimiter //

CREATE TRIGGER RR7\_obiettivo\_insert

BEFORE INSERT ON obiettivo

FOR EACH ROW

BEGIN

CALL noGoldNoPoints (new.Tipo, new.VipPoints);

END;//

delimiter //

CREATE TRIGGER RR7\_obiettivo\_update

BEFORE update ON obiettivo

FOR EACH ROW

BEGIN

CALL noGoldNoPoints (new.Tipo, new.VipPoints);

END;//