Progetto Basi di Dati

Daniele Maijnelli IN0501000

9 giugno 2023

Indice

1 Introduzione **2**
1.1 Descrizione Progetto 2
1.2 Elenco Operazioni 3

2 Diagramma Entity-Relationship **4**
2.1 Diagramma 4
2.2 Considerazioni 4

3 Documenti aggiuntivi **5**
3.1 Dizionario dei Dati - Entità 5
3.2 Dizionario dei Dati - Relazioni 6
3.3 Vincoli non esprimibili graficamente 6
3.4 Tavola dei Volumi 7

4 Diagramma Entity-Relationship Ristrutturato **8**
4.1 Diagramma 8
4.2 Analisi delle ridondanze 9
4.3 Eliminazione delle generalizzazioni 10
4.4 Partizionamento/accorpamento di concetti 10
4.5 Scelta degli identificatori primari 11

5 Schema Logico **11**
5.1 Schema 11
5.2 Normalizzazione 12

6 Query al database **13**
6.1 User Defined Functions 13
6.2 Implementazione Operazioni 14
6.3 Implementazione Trigger 16

1 Introduzione

1.1 Descrizione Progetto

Un'organizzazione che gestisce un negozio digitale in cui vende videogiochi per PC desidera avere un database che verrà utilizzato dall'applicazione con la quale si accede al negozio digitale.
L'obiettivo della base di dati è quello di avere a disposizione un catalogo dei videogiochi acquistabili nel negozio, dei videogiocatori registrati, dei giochi posseduti ogni videogiocatore e di altre informazioni necessarie per il funzionamento dell'applicazione.
I videogiocatori hanno uno username, un nome, un cognome, una data di nascita, una città di residenza, un indirizzo e dei metodi di pagamento. I metodi di pagamento si dividono in carta di credito e portafoglio, ogni metodo di pagamento registrato ha un ID.
Le carte di credito hanno un nome e un numero della carta. Il portafoglio è una quantità di denaro spendibile solo all'interno del negozio digitale, ogni giocatore ne possiede al più uno, ha un ammontare.
I videogiochi hanno un nome, uno sviluppatore, un editore, una data di rilascio, un prezzo, un numero di vendite. Quando viene effettuata una transazione, si vuole memorizzare quando è avvenuta, una transazione coinvolge un metodo di pagamento, un videogioco (che è acquistato) e un videogiocatore, che riceve il videogioco.
Si evidenzia che è possibile regalare un videogioco ad un altro videogiocatore, in questo caso il metodo di pagamento usato per la transazione non apparterrà al videogiocatore beneficiario della transazione, ogni transazione coinvolge un solo videogioco.
Uno sviluppatore ha un nome, una sede e un numero di dipendenti. Un editore ha un nome, una sede, un numero di dipendenti. Sviluppatore ed editore svolgono attività diverse, ma possono coincidere. Entrambi sono un'azienda, di cui si vuole indicare la sede.
Si vuole rappresentare quando un videogioco riceve un premio, per questioni di pubblicità. I premi hanno un nome, una categoria, un ente che ha fornito il premio, l'anno in cui è stato fornito. Gli enti hanno un nome, un tipo, un numero di fan.

1.2 Elenco Operazioni

- L'applicazione desidera ottenere i nomi dei videogiochi posseduti da un certo videogiocatore assieme a quanti ne ha in totale, assieme al metodo di pagamento utilizzato, ordinati per data di acquisto.
  (2 volte al giorno) (Interattiva)

- L'applicazione desidera ottenere le informazioni di tutti i videogiochi, incluso il numero di vendite, ordinati per numero di vendite (decrescente), assieme al numero di relativi premi ottenuti.
  (1 volta a settimana) (Interattiva)

- L'applicazione desidera memorizzare una nuova transazione all'interno del database.
  (3000 al giorno) (Interattiva)

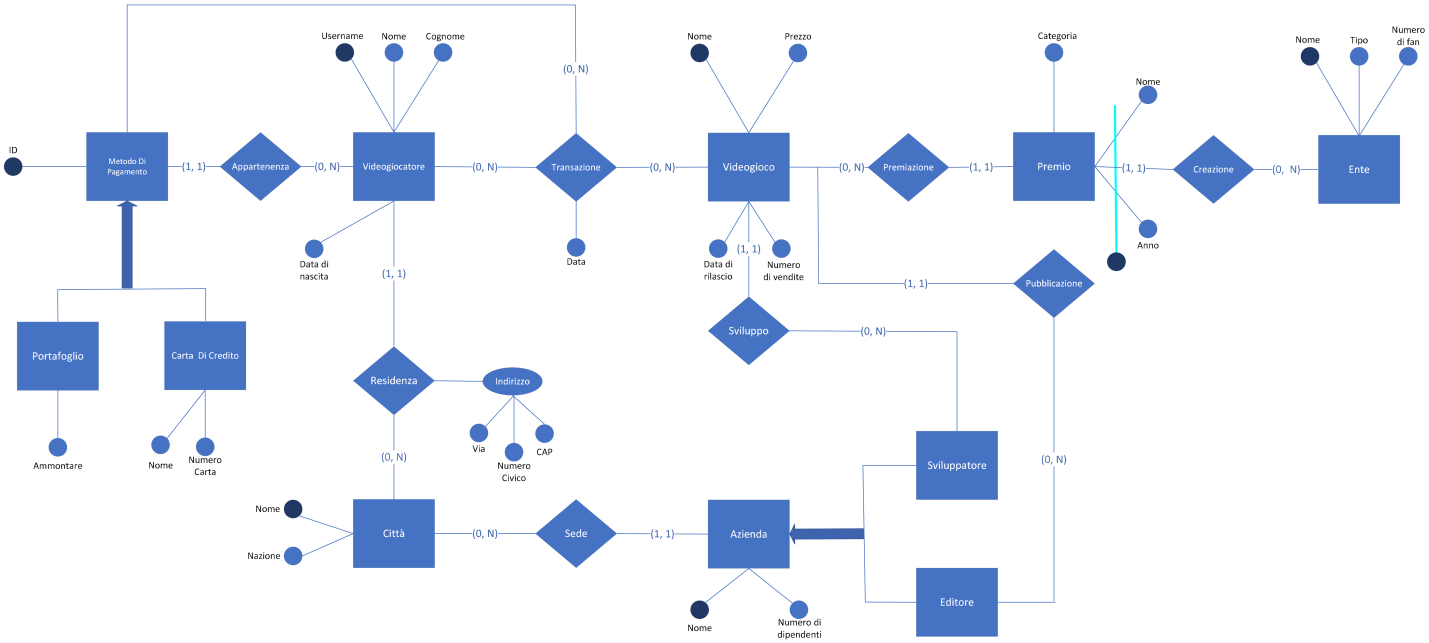
- L'applicazione desidera ottenere le informazioni degli sviluppatori, degli editori con almeno un certo numero di dipendenti e le loro relative sedi, il prezzo del videogioco più caro.
  (2 volta al mese) (Interattiva)

- L'applicazione desidera ottenere, dato un determinato premio, l'ente che l'ha rilasciato, il videogioco che l'ha ricevuto, chi ha sviluppato e chi ha pubblicato il videogioco.
  (2 volte a settimana) (Interattiva)

- L'applicazione desidera aggiornare i dati relativi ai videogiocatori.
  (4 volte al giorno) (Batch)

2 Diagramma Entity-Relationship

2.1 Diagramma



2.2 Considerazioni

Si è deciso di utilizzare una relazione ternaria Transazione perché concettualmente le 3 entità coinvolte sono legate in modo inseparabile nel momento in cui si ha una transazione. Un metodo di pagamento può essere usato per far ottenere videogiochi diversi a videogiocatori diversi. Un videogiocatore può ricevere videogiochi diversi con metodi di pagamento diversi. Un videogioco può essere acquistato con metodi di pagamento diversi per giocatori diversi. Si evidenza che dato che il Videogiocatore coinvolto nella Transazione è quello che riceve il regalo, che non coincide necessariamente con chi effettua l'acquisto, utilizzare due relazioni binarie invece di una ternaria o considerare il Metodo Di Pagamento parte di Videogiocatore, porterebbero ad una modellazione sbagliata.

Si è scelto di poter avere nel database sviluppatori e editori ancora non associati a nessun videogioco, perché in tal modo nuovi sviluppatori ed editori possono essere registrati, senza il vincolo di avere già lavorato su un videogioco. Similmente per gli enti

Le città hanno dalla loro parte delle relazioni una cardinalità (0, N) così che se una città dovesse essere associata ad un videogiocatore, ma non ad un'azienda, questo non crei problemi di vincoli all'interno del database.

La generalizzazione con entità padre Metodo Di Pagamento è di tipo totale ed esclusiva. La generalizzazione con entità padre Azienda è di tipo totale e sovrapposta.

3 Documenti aggiuntivi

3.1 Dizionario dei Dati - Entità

Entità	**Descrizione**	**Attributi**	**Identificatore**
Videogioco	Programma creato per l'intrattenimento dei suoi utenti	Nome, Prezzo, Data di rilascio, Numero di vendite	Nome
Videogiocatore	Utente dell'applicazione	Username, Nome, Cognome, Data di nascita	Username
Metodo Di Pagamento	Metodo con il quale un videogiocatore può fare acquisti nel negozio digitale	ID	ID
Carta Di Credito	Carta di credito registrata all'interno del database	ID, Numero Carta, Nome	ID
Portafoglio	Quantità di denaro spendibile all'interno dell'applicazione	ID, Ammontare	ID
Azienda	Azienda che si occupa di videogiochi in generale	Nome, Numero di dipendenti	Nome
Sviluppatore	Azienda che sviluppa videogiochi	Nome, Numero di dipendenti	Nome
Editore	Azienda che pubblicizza e pubblica videogiochi	Nome, Numero di dipendenti	Nome
Premio	Riconoscimento dato ad un videogioco degno di nota, per scopi di marketing	Nome, Categoria, Anno	Nome, Anno, (R) Creazione
Ente	Ente, che può essere un sito web o una rivista, che tratta di videogiochi	Nome, Tipo, Numero di fan	Nome
Città	Città di cui prendere nota	Nome, Nazione	Nome

3.2 Dizionario dei Dati - Relazioni

Relazione	Descrizione	Componenti	Attributi
Transazione	Acquisto di un videogioco effettuato per un videogiocatore, con un metodo di pagamento	Videogiocatore, Videogioco, Metodo Di Pagamento	Data
Appartenenza	Appartenenza di un metodo di pagamento ad un videogiocatore	Videogiocatore, Metodo Di Pagamento	
Residenza	Città in cui risiede un videogiocatore	Videogiocatore, Città	Indirizzo
Sede	Città in cui è stata fondata una azienda	Azienda, Città	
Sviluppo	Sviluppo di un videogioco	Sviluppatore, Videogioco	
Pubblicazione	Pubblicazione di un videogioco	Editore, Videogioco	
Premiazione	Atto di premiare un videogioco che si è distinto all'interno del mercato videoludico	Premio, Videogioco	
Creazione	Atto di creare un certo premio	Premio, Ente	

3.3 Vincoli non esprimibili graficamente

- Ogni giocatore deve avere al più un portafoglio

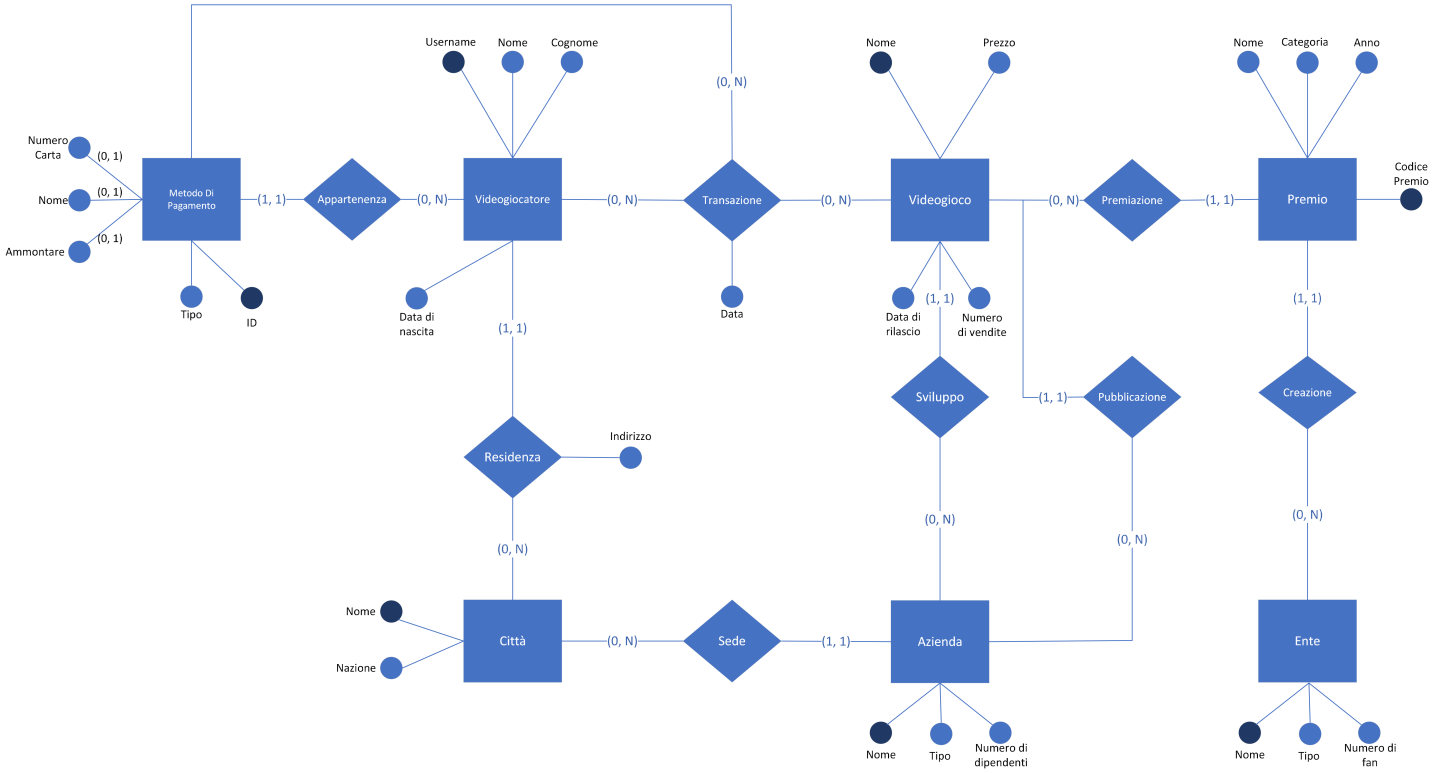
3.4 Tavola dei Volumi

Si considera che all'interno dell'applicazione a regime saranno registrati circa 10000 videogiochi e 100000 videogiocatori. Ogni videogiocatore in media avrà 2 carte di credito registrate, inoltre ogni videogiocatore avrà al più un solo portafoglio (si stima 50000 portafogli in totale), da cui numero metodi di pagamento 50000 + 200000 = 250000. Dato che alcune aziende saranno sia sviluppatore che editore (generalizzazione sovrapposta) la somma delle occorrenze stimate delle entità figlie supera leggermente il totale di occorrenze stimate dell'entità padre, 1400 + 800 = 2200 invece di 2000.

Concetto	**Tipo**	**Volume**
Videogioco	E	10000
Videogiocatore	E	100000
Metodo Di Pagamento	E	250000
Carta Di Credito	E	200000
Portafoglio	E	50000
Azienda	E	2000
Sviluppatore	E	1400
Editore	E	800
Premio	E	500
Ente	E	50
Città	E	5000
Transazione	R	1000000
Appartenenza	R	250000
Residenza	R	100000
Sede	R	2000
Sviluppo	R	10000
Pubblicazione	R	10000
Premiazione	R	500
Creazione	R	500

4 Diagramma Entity-Relationship Ristrutturato

4.1 Diagramma



4.2 Analisi delle ridondanze

L'unica ridondanza presente è dovuta all'attributo "Numero di vendite" di Videogioco, il valore di questo attributo riguarda la seconda e la terza operazione.

- Ottenere le informazioni di tutti i videogiochi, incluso il numero di vendite, ordinati per numero di vendite (decrescente), assieme al numero di relativi premi ottenuti. (1 volta a settimana)

Tabella 1: Presenza di ridondanza

Concetto	**Costrutto**	**Accessi**	**Tipo**
Videogioco	E	10000	L
Premiazione	R	500	L
Premio	E	500	L

Tabella 2: Assenza di ridondanza

Concetto	**Costrutto**	**Accessi**	**Tipo**
Videogioco	E	10000	L
Transazione	R	1000000	L
Premiazione	R	500	L
Premio	E	500	L

- Memorizzare una nuova transazione all'interno del database. (3000 al giorno)

Tabella 3: Presenza di ridondanza

Concetto	**Costrutto**	**Accessi**	**Tipo**
Transazione	R	1	S
Videogioco	E	1	L
Videogioco	E	1	S

Tabella 4: Assenza di ridondanza

Concetto	**Costrutto**	**Accessi**	**Tipo**
Transazione	R	1	S

Considerando doppi gli accessi in scrittura, è possibile contare il numero di accessi totali che si hanno ogni giorno in presenza e in assenza dell'attributo "Numero di vendite" ed eventualmente decidere se mantenerlo nel DB.

In presenza di ridondanza si hanno $(10000 + 500 + 500)/7 \simeq 1500$ accessi al giorno per la prima operazione. $(2 \cdot 2 \cdot 3000) + 3000 = 15000$ accessi al giorno per la seconda, per un totale di 16500.

In assenza di ridondanza si hanno $(10000 + 1000000 + 500 + 500)/7 \simeq 144000$ accessi al giorno per la prima operazione. $2 \cdot 3000 = 6000$ accessi al giorno per la seconda, per un totale di 150000.

Si decide di mantenere la ridondanza, dato che riduce notevolmente il numero di accessi giornalieri al database.

4.3 Eliminazione delle generalizzazioni

Per quanto riguarda la generalizzazione con entità padre Metodo Di Pagamento, dato che le operazioni che riguardano i metodi di pagamento (la prima e la terza) non fanno molta distinzione tra le occorrenze e gli attributi delle entità figlie, si decide di accorpare le entità figlie nel genitore.

Anche per la generalizzazione con entità padre Azienda, si decide di accorpare le entità figlie nel genitore, perché l'operazione che riguardano la generalizzazione la quarta ha anche in questo caso accesso al padre e ai figli contestualizzato. Il fatto che la generalizzazione sia sovrapposta favorisce questa scelta, riducendo la ridondanza dei dati.

4.4 Partizionamento/accorpamento di concetti

L'unica operazione di questo genere che conviene effettuare è la composizione degli attributi semplici "Via", "Numero Civico", "CAP" nell'unico attributo "Indirizzo" appartenente alla relazione "Residenza". Si è scelto di comporre e non il viceversa perché in nessun caso servirà accedere ad uno degli attributi semplici singolarmente, occupare memoria aggiuntiva per essi sarebbe uno spreco.

4.5 Scelta degli identificatori primari

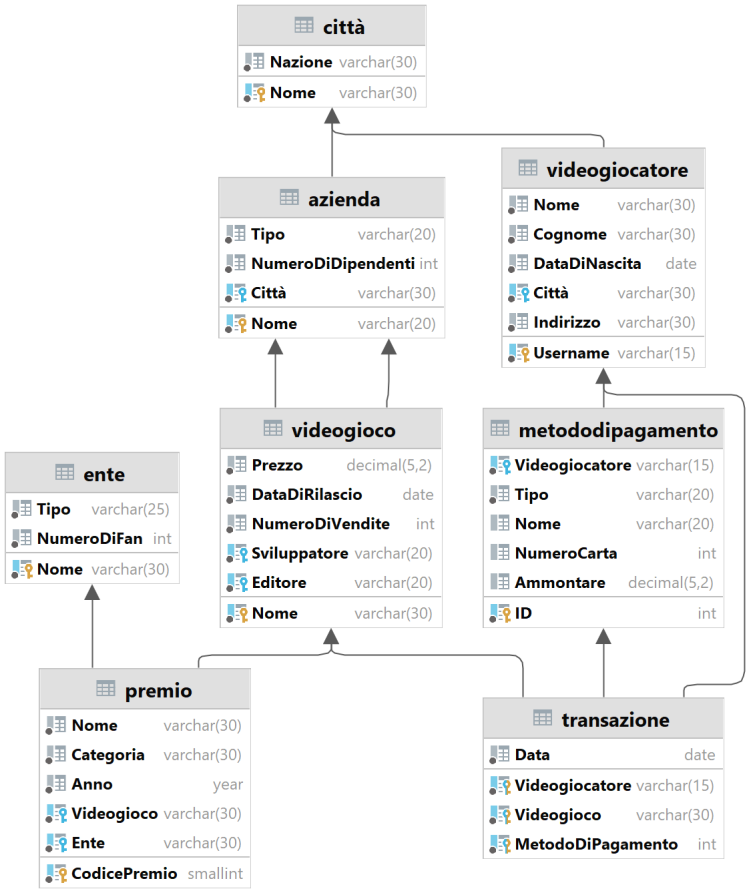
Si preferisce creare un identificatore primario "Codice Premio" per l'entità Premio migliorando le prestazioni, perché altrimenti non soddisfa il requisito di avere un identificatore primario semplice.

- Videogioco : Nome
- Videogiocatore : Username
- Metodo Di Pagamento : ID
- Azienda : Nome
- Premio : Codice Premio
- Ente : Nome
- Città : Nome

5 Schema Logico

5.1 Schema

- **Videogiocatore**(Username, Nome, Cognome, Data di nascita, Città, Indirizzo)
- **Videogioco**(Nome, Prezzo, Data di rilascio, Numero di vendite, Sviluppatore, Editore)
- **Metodo Di Pagamento**(ID, Tipo, Nome, Numero Carta, Ammontare, Videogiocatore)
- **Transazione**(Videogiocatore, Videogioco, Metodo Di Pagamento, Data)
- **Azienda**(Nome, Tipo, Numero di dipendenti, Città)
- **Premio**(Codice Premio, Nome, Categoria, Anno, Videogioco, Ente)
- **Ente**(Nome, Tipo, Numero di fan)
- **Città**(Nome, Nazione)



5.2 Normalizzazione

Il database è in prima forma normale, in tutte le tabelle, ogni colonna è atomica. Il database è in seconda forma normale, in tutte le tabelle, ogni colonna dipende in senso stretto dalla chiave primaria.

La tabella Videogioco non è in terza forma normale, perché ha un campo calcolato "Numero di vendite", servirebbe rimuovere l'attributo, ma si decide di non farlo, perché dall'analisi delle ridondanze risulta conveniente tenerlo nel database.

La tabella "MetodoDiPagamento" non è in terza forma normale, l'attributo "Ammontare" oltre a dipendere da ID, dipende da Videogiocatore. Per risolvere, si dovrebbe aggiungere una tabella di lookup "Portafoglio" con PK Videogiocatore (Foreign Key) e attributo "Ammontare".

Questa nuova tabella comporterebbe modifiche significative alle tabelle "MetodoDiPagamento" e "Transazione", inoltre il risultato sarebbe lo stesso che si sarebbe ottenuto accorpando l'entità genitore "Metodo Di Pagamento" nelle due figlie, ma si è già scelto di non farlo, perché le operazioni non distinguono tra occorrenze delle entità figlie. Dunque per questioni di prestazioni si rinuncia anche in questo caso alla terza forma normale.

6 Query al database

6.1 User Defined Functions

Per ridurre la complessità di alcune operazioni e trigger, si è deciso di aggiungere due User defined function, che verranno utilizzate in altre operazioni.

- Questa UDF prende in input il nome di un videogioco e restituisce il numero di premi assegnati ad esso.

```
delimiter $$
create function numeroDiPremi(videogame varchar(30))
    returns smallint
    deterministic
begin
    return (select count(CodicePremio) from premio where Videogioco = videogame);
end $$
delimiter ;
```

- Questa UDF prende in input in nome di un azienda e restituisce il tipo dell'azienda.

```
create function ottieniTipoAzienda(nomeAzienda varchar(20))
    returns varchar(20)
    deterministic
begin
    return (select Tipo
            from azienda
            where azienda.Nome = nomeAzienda);
end $$
delimiter ;
```

6.2 Implementazione Operazioni

- **Operazione 1** Si è deciso di utilizzare una Stored Procedure, perché svolge in modo autonomo un'operazione complessa.

```
delimiter $$
create procedure mostraLibreriaUtente
    (in Utente varchar(15), out NumeroDiGiochi int)
begin
    select t.Videogioco, m.Tipo, t.Data
    from transazione t
             inner join metododipagamento m
                        on t.MetodoDiPagamento = m.ID
    where t.Videogiocatore = Utente
    order by t.Data;
    select count(*)
    into NumeroDiGiochi
    from transazione
    where transazione.Videogiocatore = Utente;
end $$
delimiter ;
```

- **Operazione 2** Si è deciso di creare una vista per questa operazione, a cui l'applicazione accederà con una semplice query, dato che, da un punto di vista logico, l'operazione richiede di mostrare una tabella. Utilizza la prima UDF.

```
create view informazioniVideogiochi as
select v.*,
       numeroDiPremi( videogame: v.Nome) as numeroPremi
from videogioco v
order by NumeroDiVendite desc;
```

- **Operazione 3** Si è deciso di utilizzare una SP per questa operazione, per migliorare le prestazioni e incapsulare la logica di inserimento dei dati all'interno del DBMS.

```
delimiter $$
create procedure inserisciTransazione(
    utente varchar(15),
    nomevideogioco varchar(30),
    dataTransazione date,
    metodo int
)
begin
    insert into transazione (Videogiocatore, Videogioco, Data, MetodoDiPagamento)
    values (utente, nomevideogioco, dataTransazione, metodo);
end $$
delimiter ;
```

- **Operazione 4** Si è deciso di utilizzare una Stored Procedure, perché svolge in modo autonomo una query complessa, al suo interno c'è una subquery.

```
delimiter $$
create procedure mostraAziende(
    totaleDipendenti int
)
begin
    select a.*,
           (select max(Prezzo)
            from videogioco v
            where (v.Editore = a.Nome or v.Sviluppatore = a.Nome))
              as PrezzoMaggiore
    from azienda a
    where NumeroDiDipendenti >= totaleDipendenti
    order by NumeroDiDipendenti desc;
end $$
delimiter ;
```

- **Operazione 5** Si è deciso di utilizzare una Stored Procedure, perché svolge in modo autonomo una query complessa, ricevendo un parametro di input.

```
delimiter $$
create procedure infoPremio(
    premioID smallint
)
begin
    select p.Nome,
           p.Categoria,
           p.Anno,
           e.Nome,
           e.Tipo,
           v.Nome,
           v.Sviluppatore,
           v.Editore
    from premio p
             inner join ente e
                        on p.Ente = e.Nome
             inner join videogioco v on p.Videogioco = v.Nome
    where p.CodicePremio = premioID;
end $$
delimiter ;
```

6.3 Implementazione Trigger

- Il primo vincolo non esprimibile graficamente viene implementato come un trigger che durante l'inserimento di un nuovo portafoglio, si assicura che non ce ne sia uno già esistente per quel determinato videogiocatore.

```
delimiter $$
create trigger verificaPortafoglio
    before INSERT
    on metododipagamento
    for each row
begin
    declare presenzaPortafoglio int;
    set presenzaPortafoglio = 0;
    select count(Ammontare)
    into presenzaPortafoglio
    from metododipagamento
    where Tipo = 'Portafoglio'
      and metododipagamento.Videogiocatore = new.Videogiocatore;
    if presenzaPortafoglio > 0 and new.Tipo = 'Portafoglio'
    then
        SIGNAL sqlstate '20001'
            SET message_text =
                'I videogiocatori possono avere al più un portafoglio';
    end if;
end $$
delimiter ;
```

- Trigger che verifica che il tipo di una azienda rientri tra quelli previsti.

```
delimiter $$
create trigger verificaTipoAzienda
    before INSERT
    on azienda
    for each row
begin
    if new.Tipo not in ('Editore', 'Sviluppatore', 'Generico')
    then
        SIGNAL sqlstate '20002'
            SET message_text = 'Tipo di azienda inesistente';
    end if;
end $$
delimiter ;
```

- Trigger che verifica che gli attributi Editore e Sviluppatore nella tabella Videogioco siano coerenti con il tipo di Azienda. Utilizza una UDF.

```
delimiter $$
create trigger verificaTipoAziendaInVideogioco
    before INSERT
    on videogioco
    for each row
begin
    declare tipoAzienda varchar(20);
    set tipoAzienda = ottieniTipoAzienda( nomeAzienda: new.Sviluppatore);
    if tipoAzienda not in ('Sviluppatore', 'Generico')
    then
        SIGNAL sqlstate '20003'
            SET message_text = 'Tipo di azienda incoerente';
    end if;
    set tipoAzienda = ottieniTipoAzienda( nomeAzienda: new.Editore);
    if tipoAzienda not in ('Editore', 'Generico')
    then
        SIGNAL sqlstate '20003'
            SET message_text = 'Tipo di azienda incoerente';
    end if;
end $$
delimiter ;
```

- Trigger che verifica che il metodo di pagamento da inserire sia valido.

```
delimiter $$
create trigger verificaTipoMetodoDiPagamento
    before insert
    on metododipagamento
    for each row
begin
    if new.Tipo not in ('Portafoglio', 'Carta di credito')
    then
        signal sqlstate '20004'
            SET message_text = 'Tipo metodo di pagamento non valido';
    elseif (new.Tipo = 'Portafoglio' and
            (new.Ammontare is null or new.NumeroCarta is not null or
             new.Nome is not null))
    then
        signal sqlstate '20005'
            SET message_text = 'Informazioni metodo di pagamento errate';
    elseif (new.Tipo = 'Carta di credito' and
            (new.Ammontare is not null or new.NumeroCarta is null or
             new.Nome is null))
    then
        signal sqlstate '20005'
            SET message_text = 'Informazioni metodo di pagamento errate';
    end if;
end $$
delimiter ;
```