

**Università degli Studi di Trieste
Ingegneria Elettronica e Informatica**

**Progetto Basi di Dati
“GameStore”**

**Denis Malasi
IN0500804**

Indice

1. Requisiti della base di dati	3
2. Glossario dei termini	4
3. Diagramma Entity-Relationship	5
4. Dizionario dei dati (Entità)	6
5. Dizionario dei dati (Relazioni)	7
6. Vincoli non esprimibili graficamente, Tabella dei volumi	8
7. Azioni di interesse	9
8. Analisi ridondanze, Eliminazione attributi composti, Scelta identificatori primari	10
9. Diagramma Entity-Relationship Ristrutturato	11
10. Schema Logico	12
11. Normalizzazione	13
12. Query SQL per la creazione del Database	14
13. Query SQL per interagire col Database	16
14. Implementazione Java	18
15. Triggers	21

Requisiti della Base di Dati

Si vuole realizzare una base di dati che gestisca un negozio virtuale chiamato **GameStore** che permette alle persone di scaricare giochi per il cellulare.

Ogni gioco viene pubblicato da uno **sviluppatore** (in genere sono case di sviluppo o Software House) di cui si è interessati a conosce la loro Partita IVA, il loro nome, il sito web, il telefono, la mail e la sede legale (indirizzo, città, Cap e la sigla della provincia).

I **giochi** possono essere gratuiti o a pagamento e sono identificati da un codice univoco. Inoltre, di un gioco si vogliono sapere il suo titolo, le sue dimensioni, l'età minima per giocarci (che va dai 3 ai 18 anni), una breve descrizione, la sua data di pubblicazione, il prezzo e lo sviluppatore del gioco stesso.

Si possono creare degli **utenti** ed ognuno avrà il suo Nickname unico. Per iscriversi a questo negozio virtuale l'utente dovrà inserire il suo nome, cognome, la sua data di nascita, il suo telefono, la sua mail, il suo indirizzo (indirizzo, città, Cap e la sigla della provincia) ed infine la password del suo account.

Ogni utente può stringere un' **amicizia** con un altro utente. Si desidera sapere la data in cui si è “stretta” questa amicizia.

Si vuole registrare la cronologia delle **versioni** di un gioco. Per ogni versione si vuole registrare il tipo (alfa, beta, release), il numero della versione, una descrizione (facoltativa) e la data di pubblicazione della versione.

Un utente può acquistare più giochi e per ogni gioco si vogliono sapere i **dati** di questo **gioco**, cioè quando è stato acquistato e le ore di gioco dell'utente.

Dopo aver acquistato un gioco, l'utente può lasciarci una **recensione**. La recensione è composta da un titolo, una valutazione in stelle (da 1 a 5), una data e da un testo (facoltativo).

Dato che alcuni giochi sono a pagamento bisogna registrare anche i dati della carta di credito di ogni utente.

Quindi ogni utente avrà dei **dati confidenziali** (password dell'account e dati della carta di credito) che dovranno essere salvati da parte e crittati.

Prima di fare una recensione l'utente deve aver giocato almeno 1 ora a quel gioco (quindi un utente non può recensire un gioco se non l'ha comprato).

Un utente può recensire un singolo gioco una sola volta.

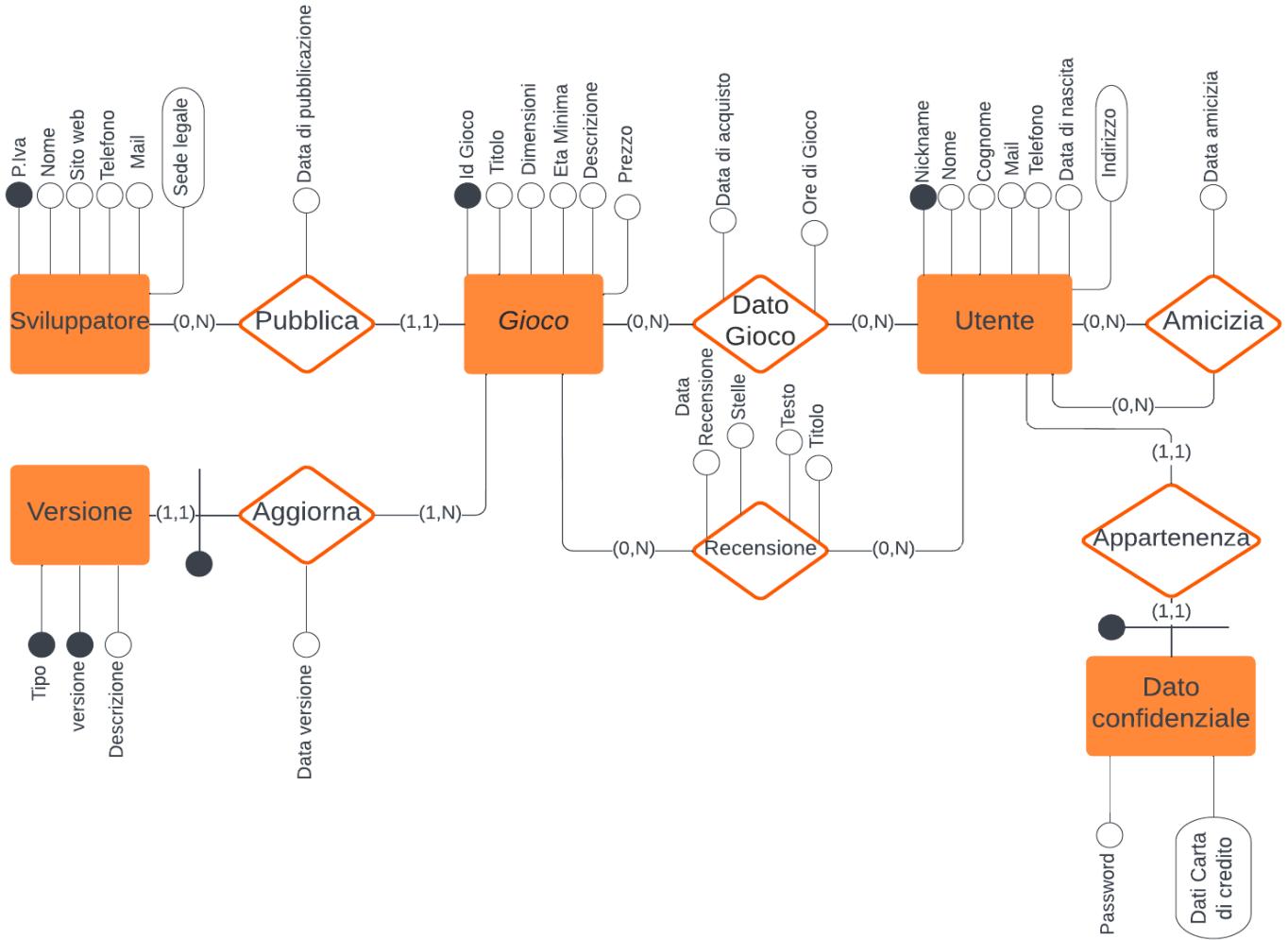
Un utente che non ha l'età minima richiesta dal gioco non può scaricarlo.

Se un gioco viene pubblicato senza alfa o beta nella cronologia delle versioni apparirà “Release 1.00” e come data avrà la data di pubblicazione del gioco

Glossario dei termini

Termine	Definizione	Collegamento
Sviluppatore	Colui che programma e distribuisce il gioco	Gioco
Gioco	Prodotto pubblicato nel negozio che può essere scaricato o recensito	Sviluppatore, Utente, Versione
Utente	Colui che usufruisce del gioco	Gioco, Utente, Dato confidenziale
Amicizia	Relazione fra 2 utenti	Utente
Versione	Nuove versioni del gioco che lo migliorano costantemente	Gioco
Dato Gioco	Data di acquisto del gioco e le ore giocate da un utente specifico	Gioco, Utente
Recensione	Valutazione che un’utente lascia ad un gioco a cui ha giocato	Gioco, Utente
Dato Confidenziale	Dati importanti e privati di ogni utente	Utente

Diagramma Entity-Relationship



Per “Dato Gioco” si intende una relazione che racchiuda l’utente e il gioco che ha scaricato

La cardinalità di Gioco in aggiorna è (1,N) perché ci sarà almeno una versione “Release 1.00” del gioco con la data di versione uguale alla data di pubblicazione del gioco.

Dizionario dei dati (Entità)

Entità	Attributi	Precisazioni	Identificatore
Sviluppatore	Partita Iva, Nome, Sito web, Mail, Telefono, Sede Legale	Sede Legale è un attributo composto: Indirizzo, città, Cap e Provincia.	Partita Iva
Gioco	Id Gioco, Titolo, Dimensioni, Età Minima, Descrizione, Prezzo	Età Minima è l'età minima che deve avere l'utente per giocare al gioco	Id Gioco
Utente	Nickname, Nome, Cognome, Mail, Telefono, Indirizzo, Data di nascita	Indirizzo è un attributo composto: Indirizzo, città, Cap e Provincia.	Nickname
Versione	Tipo, Versione, Descrizione	Tipo può essere: alfa, beta, release. Versione è il numero della versione.	Tipo, Versione
Dato Confidenziale	Password, Dati carta di credito	Dati carta di credito è un attributo composto: Numero carta, scadenza, CVV.	

Dizionario dei dati (Relazioni)

Relazione	Componenti	Attributi	Precisazioni
Pubblica	Sviluppatore, Gioco	Data di pubblicazione	Data in cui lo sviluppatore pubblica la prima versione del suo gioco sullo store
Aggiorna	Versione, Gioco	Data Versione	Data di uscita della nuova versione di un gioco
Dato Gioco	Utente, Gioco	Data di acquisto, ore di gioco	Ore di Gioco sono le ore che ha speso un utente su un singolo gioco
Recensione	Utente, Gioco	Titolo, Stelle, Data Recensione, Testo	Le stelle vanno da 1 (pessimo) a 5 (ottimo)
Amicizia	Utente, Utente	Data Amicizia	Data in cui i 2 utenti sono diventati amici
Appartenenza	Utente, Dato Confidenziale	—	—

Vincoli non esprimibili graficamente

- Prima di fare una recensione ad un gioco l'utente deve aver giocato almeno 1 ora a quel gioco.
- Un utente può recensire un gioco solo se l'ha comprato.
- Un utente può recensire un singolo gioco una sola volta.
- La data della recensione non può essere precedente alla data di pubblicazione del gioco o precedente alla data di acquisto del gioco.
- Le stelle in una recensione vanno da 1 a 5.
- L'età minima del gioco va dai 3 ai 18 anni.
- Un utente che non ha l'età minima richiesta dal gioco non può scaricarlo.
- Il tipo di versione può essere solo uno tra: alfa, beta, release.
- Vincoli nelle date delle versioni dei giochi (Es: la release 2.00 non può avere la data di pubblicazione prima della release 1.00 dello stesso gioco, la alfa 2.00 non può essere rilasciata dopo della beta 2.00, ecc.)

Tabella dei Volumi

Suppongo che questo negozio virtuale sia stato reso pubblico da poco tempo e che sia ancora in crescita. Quindi gli utenti, i giochi, gli sviluppatori e le azioni sul DB saranno poche.

Concetto	Tipo	Volume
Sviluppatore	E	20
Gioco	E	100
Utente	E	2000
Dato confidenziale	E	2000
Versione	E	400
Dato Gioco (giochi scaricati)	R	5000
Amicizia	R	8000
Recensione	R	2000
Pubblica	R	5
Aggiorna	R	4
Appartenenza	R	1

Azioni di interesse

Azione	Tipo	Frequenza
Aggiunta di un nuovo utente	Interattiva	50/giorno
Aggiunta di un nuovo sviluppatore	Interattiva	1/settimana
Aggiunta di un nuovo gioco	Interattiva	5/settimana
Aggiunta di una recensione	Interattiva	10/giorno
Aggiunta di una versione	Interattiva	10/settimana
Aggiunta dati confidenziali	Interattiva	50/giorno (si inseriscono insieme ai dati dell'utente)
Aggiunta di una amicizia	Interattiva	10/giorno
Aggiunta dati Gioco (utente scarica un gioco)	Interattiva	80/giorno
Visualizzazione del proprio catalogo giochi da parte di un utente	Interattiva	100/giorno
Ricavare la lista dei primi 10 utenti con più ore di gioco tra tutti i giochi	batch	1/giorno
Ricavare la lista dei primi 5 giochi gratis con più download	batch	1/giorno
Ricavare la lista dei primi 5 giochi a pagamento con più download	batch	1/giorno
Ricavare la lista dei primi 5 giochi più giocati (con più ore di gioco totali)	batch	1/giorno
Incrementare le ore di gioco di un utente appena smette di giocare	batch	300/giorno

Il negozio virtuale offrirà ai suoi utenti, nella schermata principale della sua applicazione, varie classifiche aggiornate quotidianamente che mostreranno gli utenti con più ore giocate, i giochi con più download, etc.

Inoltre, ogni volta che utente chiuderà un gioco, ci sarà un programma che dati in ingresso il nickname dell’utente, l’id del gioco e le ore giocate nella sessione appena conclusa incrementerà l’attributo Ore_Giocate.

Analisi ridondanze

Il problema non presenta ridondanze da analizzare

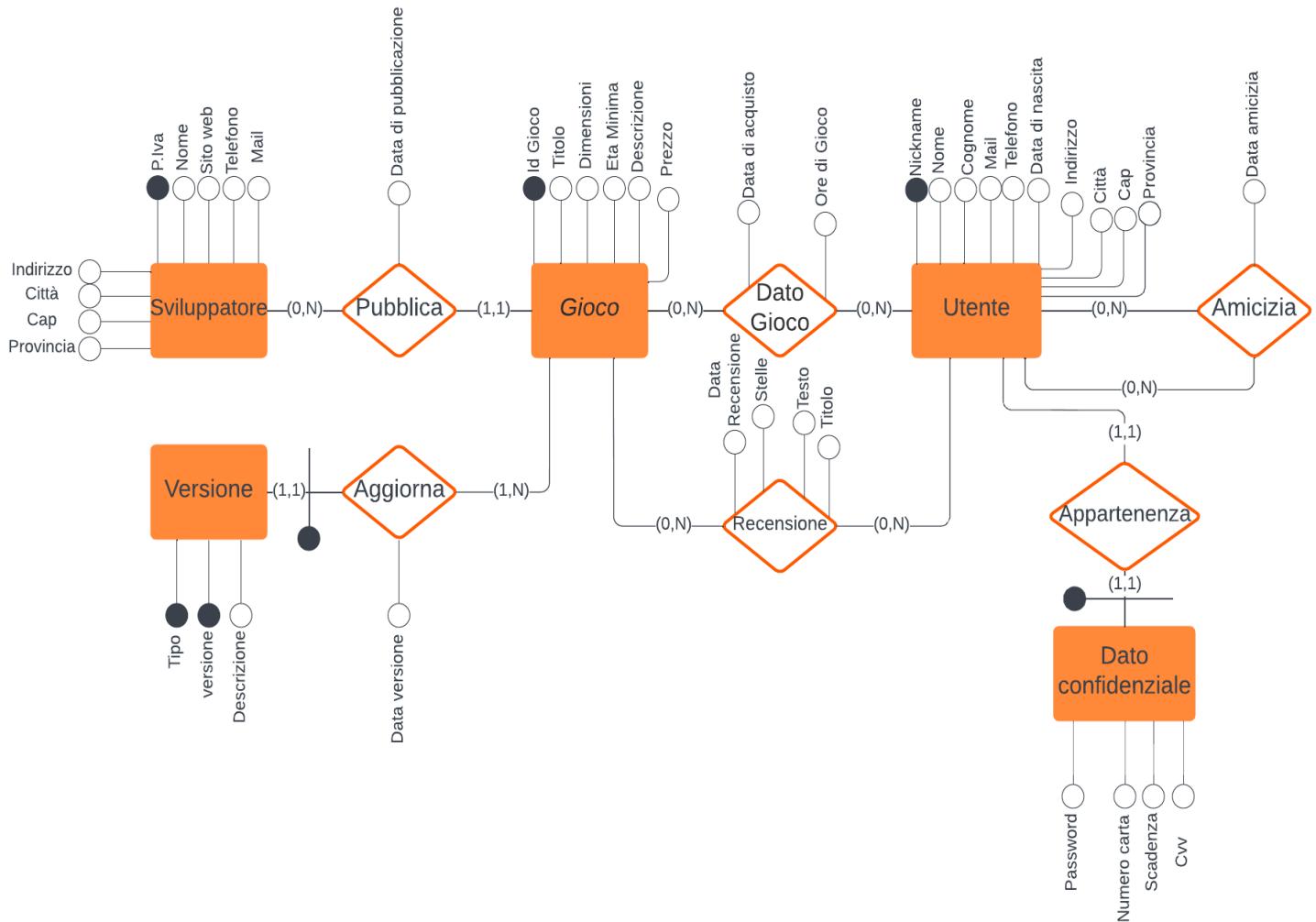
Eliminazione attributi composti

Gli attributi composti vengono scomposti in attributi più semplici.
‘Indirizzo’ e ‘sede legale’ vengono scomposti in: indirizzo, città, Cap e provincia.
Mentre ‘Dati carta di credito’ viene scomposta in: numero carta, scadenza e Cvv.

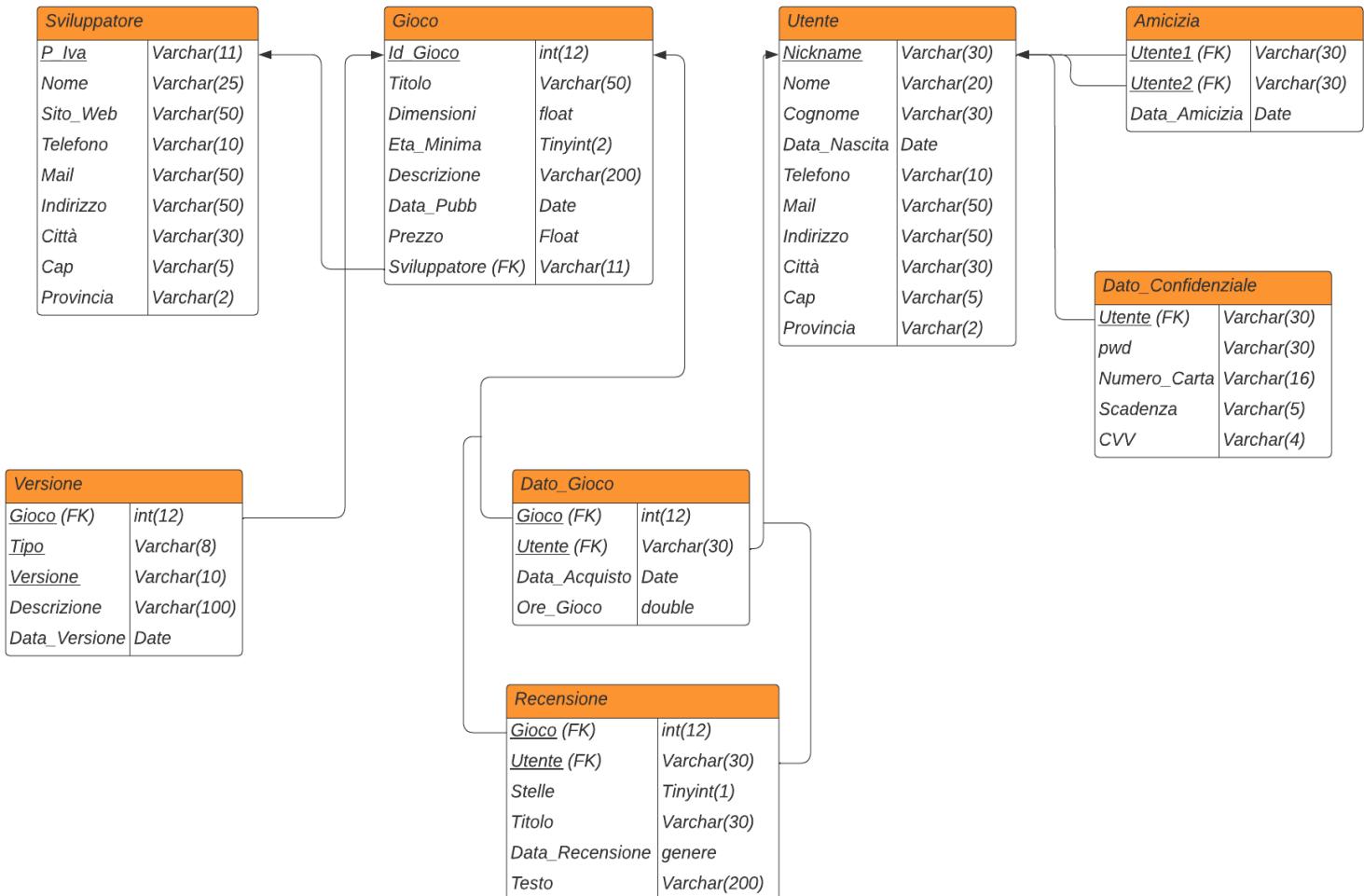
Scelta identificatori primari

Entità	Identificatore
Sviluppatore	Partita Iva
Gioco	Id Gioco
Utente	Nickname
Versione	Tipo, Versione, Id Gioco (esterno)
Dato Confidenziale	Nickname (esterno)

Diagramma Entity-Relationship Ristrutturato



Schema Logico



Le chiavi primarie sono sottolineate, mentre le chiavi secondarie sono contrassegnate con un ‘FK’.

- Sviluppatore (P_Iva*, Nome, Sito_Web, Telefono, Mail, Indirizzo, Città, Cap, Provincia)
- Utente (Nickname*, Nome, Cognome, Data_Nascita, Telefono, Mail, Indirizzo, Città, Cap, Provincia)
- Gioco (Id_Gioco*, Titolo, Dimensioni, Eta_Minima, Descrizione, Data_Pubb, Prezzo, Sviluppatore[Sviluppatore])
- Amicizia (Utente1[Utente]*, Utente2[Utente]*, Data_Amicizia)

- Dato_Confidenziale (Utente[Utente]*, pwd, Numero_Carta, Scadenza, CVV)
- Versione (Gioco[Gioco]*, Tipo*, Versione*, Descrizione, Data_Versione)
- Dato_Gioco (Gioco[Gioco]*, Utente[Utente]*, Data_Acquisto, Ore_Gioco)
- Recensione (Gioco[Gioco]*, Utente[Utente]*, Stelle, Titolo, Data_Recensione, Testo)

Normalizzazione

La base di dati è in prima forma normale (**1FN**) in quanto tutte le colonne sono atomiche.

La base di dati è in prima forma normale (**2FN**) in quanto ciascuna colonna dipende dalla chiave primaria.

Si nota che le tabelle Sviluppatore e Utente non sono in terza forma normale (**3FN**) perché la colonna Cap dipende da Città, Provincia e Indirizzo.

Per ovviare a questo problema si potrebbe aggiungere una tabella “Posizione” che ha come attributi: Indirizzo, Città, Cap e Provincia.

Tuttavia, si è deciso di non creare questa tabella in quanto finirebbe solo per appesantire il progetto senza portare ad alcun vantaggio concreto.

Query SQL per la creazione del Database

Di seguito vengono rappresentate le query per la creazione delle sole tabelle con dei vincoli da rispettare in certi attributi. Le altre tabelle vengono ommesse per semplicità.

Creazione tabella Gioco

```
19  ↗create table if not exists gioco(
20      id_Gioco int(12) primary key auto_increment,
21      titolo varchar(50) not null,
22      Dimensioni float not null,
23      Eta_Minima tinyint(2) not null,
24      Descrizione varchar(200) not null,
25      Data_Pubb date not null,
26      prezzo float,
27      Sviluppatore varchar(11),
28      constraint FK_Sviluppatore foreign key (Sviluppatore) references sviluppatore(P_Iva),
29      constraint Con_Eta check (Eta_Minima between 3 and 18)
30  ↘);
```

Il vincolo a riga 29 serve per far sì che il gioco non abbia un'età minima inferiore a 3 o superiore a 18 anni.

Creazione tabella Versione

```
46  ↗create table if not exists versione(
47      Gioco int(12),
48      tipo varchar(8),
49      versione varchar(10),
50      descrizione varchar(100),
51      data_Versione date not null,
52      primary key (Gioco,tipo,versione),
53      constraint FK_GiocoVersione foreign key (Gioco) references gioco(id_Gioco) ON DELETE cascade ON UPDATE CASCADE,
54      constraint Con_versione check (tipo in ('alfa','beta','release'))
55  ↘);
```

Il Vincolo a riga 54 serve per dire che l'attributo tipo della versione può essere solo uno tra: alfa, beta, release.

Creazione tabella Recensione

```
67  ↗create table if not exists recensione(
68      Gioco int(12),
69      Utente varchar(30),
70      stelle tinyint(1) not null,
71      titolo varchar(30) not null,
72      data_Recensione date not null,
73      testo varchar(200),
74      primary key (Gioco,Utente),
75      constraint FK_GiocoRecensione foreign key (Gioco) references gioco(id_Gioco) ON DELETE cascade ON UPDATE CASCADE,
76      constraint FK_UtenteRecensione foreign key (Utente) references Utente(Nickname) ON DELETE cascade ON UPDATE CASCADE,
77      constraint Con_Stelle check ( stelle between 1 and 5)
78  ↘);
```

Il vincolo a riga 77 serve per far in modo che le stelle stiano tra 1 e 5.

Query SQL per interagire col Database

1) Visualizzare i primi 5 giochi più giocati (con più ore di gioco in totale)

```
select g.id_Gioco, g.titolo, truncate(sum(Ore_Gioco), 1) as Totale_Ore from dato_Gioco inner join gioco g on dato_Gioco.Gioco = g.id_Gioco  
group by Gioco  
order by Totale_Ore desc  
limit 5;
```

2) Visualizzare i primi 5 giochi gratis con più download

```
select g.titolo, COUNT(*) as Numero_Download from gioco g inner join dato_Gioco dG on g.id_Gioco = dG.Gioco  
where g.prezzo is null  
group by titolo  
order by Numero_Download desc  
limit 5;
```

3) Visualizzare i primi 5 giochi a pagamento con più download

```
select g.titolo, COUNT(*) as Numero_Download from gioco g inner join dato_Gioco dG on g.id_Gioco = dG.Gioco  
where g.prezzo is not null  
group by titolo  
order by Numero_Download desc  
limit 5;
```

4) Visualizzare i primi 10 utenti con più ore giocate in totale

```
select Nickname, truncate(Sum(dG.Ore_Gioco), 1) as Totale_Ore from Utente inner join dato_Gioco dG on Utente.Nickname = dG.Utente  
group by Nickname  
order by Totale_Ore desc  
limit 10;
```

5) Dato l'id di gioco, visualizzare il giocatore che ha fatto più ore su quel gioco

```
select Ore_Gioco, Utente from dato_Gioco  
where Gioco=?  
ORDER BY Ore_Gioco desc  
limit 1;
```

6) Dato il nickname di un utente, visualizzare la lista dei giochi scaricati con le ore di gioco

```
select titolo,dg.Ore_Gioco from gioco inner join dato_Gioco dg on gioco.id_Gioco = dg.Gioco  
where dg.Utente=?;
```

7) Dato il nickname di un utente, visualizzare i soldi che ha speso per acquistare i giochi

```
select TRUNCATE(sum(g.prezzo),2) as somma_Prezzi from dato_Gioco inner join gioco g on Gioco=g.id_Gioco  
where Utente=?;
```

8) Incrementare le ore di gioco di un utente

```
UPDATE dato_Gioco  
set Ore_Gioco=Ore_Gioco + oreGiocoNuove  
where dato_Gioco.Utente=utente and dato_Gioco.Gioco=gioco;
```

Implementazione Java

Non sono state usate stored procedures che sono state sostituite da procedure Java in quanto sono più facili per il debug

```
public void GiochiPiùOre() throws SQLException {
    Statement stmt= conn.createStatement();
    PreparedStatement preparedStatement;
    ResultSet rs=stmt.executeQuery( sql: "select g.id_Gioco,g.titolo,truncate(sum(Ore_Gioco),1) as Totale_Ore "+
        "from dato_Gioco inner join gioco g on dato_Gioco.Gioco = g.id_Gioco\n" +
        "group by Gioco\n" +
        "order by Totale_Ore desc\n" +
        "limit 5;");
    System.out.println("---Classifica giochi più giocati---");
    int indice=0;
    while (rs.next()){
        preparedStatement= conn.prepareStatement( sql: "select Ore_Gioco, Utente from dato_Gioco \n"+
            "where Gioco=?\n" +
            "ORDER BY Ore_Gioco desc\n" +
            "limit 1;");
        preparedStatement.setInt( parameterIndex: 1,rs.getInt( columnLabel: "id_Gioco"));
        ResultSet resultSet=preparedStatement.executeQuery();
        resultSet.next();
        double ore=rs.getDouble( columnLabel: "Totale_Ore");
        double oreGiocatore=resultSet.getDouble( columnLabel: "Ore_Gioco");
        System.out.println((++indice)+" "+rs.getString( columnLabel: "titolo")+", Ore di gioco: "+(int) ore+"h "+((int)((ore-(int)ore)*60)+1)+
            "m , Utente con più ore: "+ resultSet.getString( columnLabel: "Utente") +"+ "+ (int)oreGiocatore + "h "+((int)((oreGiocatore-(int)oreGiocatore)*60)+1)+"m");
    }
}
```

Questa funzione in java stampa la lista dei primi 5 giochi più giocati, inoltre stampa l'utente con più ore di gioco per ognuno dei 5 giochi.
Sono state utilizzate le query 1) e 5).

```
public void GiochiGratisScaricati() throws SQLException {
    Statement stmt= conn.createStatement();
    ResultSet rs=stmt.executeQuery( sql: "select g.titolo, COUNT(*) as Numero_Download \n"+
        "from gioco g inner join dato_Gioco dG on g.id_Gioco = dG.Gioco \n" +
        "where g.prezzo is null \n" +
        "group by titolo \n" +
        "order by Numero_Download desc \n" +
        "limit 5;");
    System.out.println("---Classifica giochi gratis più giocati---");
    int indice=0;
    while (rs.next())
        System.out.println((++indice)+" "+rs.getString( columnLabel: "titolo")+", Numero di download: "+rs.getInt( columnLabel: "Numero_Download"));
}
```

Questa funzione stampa la lista dei primi 5 giochi gratis più scaricati.
È stata utilizzata la query 2).

```

public void GiochiPagamentoScaricati() throws SQLException {
    Statement stmt= conn.createStatement();
    ResultSet rs=stmt.executeQuery( sql: "select g.titolo, COUNT(*) as Numero_Download \n"+
        "from gioco g inner join dato_Gioco dG on g.id_Gioco = dG.Gioco \n"+
        "where g.prezzo is not null \n"+
        "group by titolo \n"+
        "order by Numero_Download desc \n"+
        "limit 5");
    System.out.println("---Classifica giochi a Pagamento più giocati---");
    int indice=0;
    while (rs.next())
        System.out.println((++indice)+" "+rs.getString( columnLabel: "titolo")+", Numero di download: "+rs.getInt( columnLabel: "Numero_Download"));
}

```

Questa funzione stampa la lista dei primi 5 giochi a pagamento più scaricati.
È stata utilizzata la query 3).

```

public void OreGiocoUtenti() throws SQLException {
    Statement stmt= conn.createStatement();
    ResultSet rs=stmt.executeQuery( sql: "select Nickname, truncate(Sum(dG.Ore_Gioco),1) as Totale_Ore \n" +
        "from Utente inner join dato_Gioco dG on Utente.Nickname = dG.Utente\n" +
        "group by Nickname\n" +
        "order by Totale_Ore desc\n" +
        "limit 10");
    System.out.println("---Classifica giocatori con più ore di gioco---");
    int indice=0;
    while (rs.next()){
        double ore=rs.getDouble( columnLabel: "Totale_Ore");
        System.out.println((++indice)+" "+rs.getString( columnLabel: "nickname")+", Ore di gioco: "+(int)ore+"h "+((int)((ore-(int)ore)*60)+1)+"m");
    }
}

```

Questa funzione stampa i primi 10 utenti con più ore di gioco in totale.
È stata utilizzata la query 4).

```

public void CatalogoUtente(String nick) throws SQLException {
    PreparedStatement preparedStatement;
    ResultSet rs;
    preparedStatement=conn.prepareStatement( sql: "select titolo,dg.Ore_Gioco \n" +
        "from gioco inner join dato_Gioco dg on gioco.id_Gioco = dg.Gioco\n" +
        "where dg.Utente=?;");
    preparedStatement.setString( parameterIndex: 1,nick);

    rs=preparedStatement.executeQuery();
    int indice=0;
    System.out.println("---Catalogo giochi dell'utente "+nick+" ---");
    while (rs.next()){
        double ore=rs.getDouble( columnLabel: "Ore_Gioco");
        System.out.println((++indice)+" "+rs.getString( columnLabel: "titolo")+ ", Ore Giocate: "+(int)ore+"h "+((int)((ore-(int)ore)*60)+1)+"m");
    }
    preparedStatement=conn.prepareStatement( sql: "select TRUNCATE(sum(g.prezzo),2) as somma_Prezzi \n" +
        "from dato_Gioco inner join gioco g on Gioco=g.id_Gioco\n" +
        "where Utente=?");
    preparedStatement.setString( parameterIndex: 1,nick);
    rs=preparedStatement.executeQuery();
    rs.next();
    System.out.println("Soldi spesi: "+rs.getDouble( columnLabel: "somma_Prezzi")+"€");
}

```

Questa funzione, dato il nickname di un utente, stampa il suo catalogo dei giochi (i giochi che ha scaricato) e i soldi che ha speso per acquistarli.
Sono state utilizzate le query 6) e 7).

```

public void IncrementaOre(String n,int g,int o) throws SQLException {
    PreparedStatement preparedStatement= conn.prepareStatement( sql: "UPDATE dato_Gioco\n" +
        "        set Ore_Gioco=Ore_Gioco + ?\n" +
        "        where dato_Gioco.Utente=? and dato_Gioco.Gioco=?;");
    preparedStatement.setInt( parameterIndex: 1,o);
    preparedStatement.setString( parameterIndex: 2,n);
    preparedStatement.setInt( parameterIndex: 3,g);
    preparedStatement.execute();
}

```

Questa funzione prende in ingresso il nickname di un utente, l'id di un gioco e le ore giocate e incrementa l'attributo Ore_Giocate.
È stata utilizzata la query 8).

Triggers

```
DELIMITER $$  
create TRIGGER VincoloAmicizia  
before insert on amicizia  
for each row  
begin  
    DECLARE data date;  
    select data_Amicizia into data from amicizia where Utente1=new.Utente2 and Utente2=new.Utente1;  
    if data is not null then  
        SIGNAL sqlstate '45001' SET message_text = 'esiste già questa amicizia';  
    end if;  
end;  
DELIMITER $$
```

Questo Trigger è stato creato per impedire che vengano inserite amicizie duplicate. Es: se c'è già un'amicizia tra Gianni (utente1) e Mattia (utente2), potrebbe essere inserita un'amicizia superflua tra Mattia (utente1) e Gianni (utente2) senza che il database se ne accorga perché la chiave primaria non è la stessa.

```
DELIMITER $$  
create TRIGGER VincoloEta  
before insert on dato_Gioco  
for each row  
begin  
    DECLARE eta int;  
    DECLARE anno int;  
    select Eta_Minima into eta from gioco where id_Gioco=new.Gioco;  
    select YEAR(data_Nascita) into anno from Utente where Nickname=new.Utente;  
    if (2022-anno) < eta then  
        SIGNAL sqlstate '45001' SET message_text = 'Non hai l eta minima per scaricare il gioco';  
    end if;  
end;  
DELIMITER $$
```

Questo Trigger serve per impedire che un utente scarichi senza che abbia l'età minima per giocarci.

```

DELIMITER $$

Create TRIGGER InserimentoVersione
after insert on gioco
for each row
begin
    insert into versione (Gioco, tipo, versione, descrizione,data_Versione) values (new.id_Gioco,'release','1.00','Versione Release',new.Data_Pubb);
end$$
delimiter ;

```

Non appena viene pubblicato un gioco aggiunge la versione ‘Release’ 1.00 nella tabella versioni.

```

DELIMITER $$

Create TRIGGER VincoloDataGioco
before insert on dato_Gioco
for each row
begin
    DECLARE data DATE;
    SELECT Data_Pubb into data from gioco where gioco.id_Gioco=new.Gioco;
    if data > NEW.Data_Acquisto then
        SIGNAL sqlstate '45001' SET message_text = 'Non puoi comprare un gioco che non è ancora uscito';
    end if;
end$$
DELIMITER ;

```

Questo Trigger serve per assicurarsi che i dati in ingresso siano giusti. In questo caso se la data di acquisto è prima della data di pubblicazione del gioco da errore.

```

2      DELIMITER $$ 
3  Create TRIGGER VincoliRecensioni
4    before insert on recensione
5    for each row
6  begin
7    DECLARE ore double;
8    DECLARE data_Pubb DATE;
9    DECLARE data_Acq DATE;
10   DECLARE num int;
11   SELECT Ore_Gioco into ore from dato_Gioco where dato_Gioco.Gioco=new.Gioco and dato_Gioco.Utente=new.Utente;
12   SELECT Data_Acquisto into data_Acq from dato_Gioco where dato_Gioco.Gioco=new.Gioco and dato_Gioco.Utente=new.Utente;
13   SELECT Data_Pubb into data_Pubb from gioco where gioco.id_Gioco=new.Gioco;
14   SELECT count(*) into num from recensione where gioco=new.Gioco and utente=new.utente; -- si può ommettere
15   if ore is null then
16     SIGNAL sqlstate '45001' SET message_text = 'Non hai ancora scaricato il gioco';
17   elseif ore < 1 then
18     SIGNAL sqlstate '45001' SET message_text = 'Hai giocato meno di 1h non puoi recensire il gioco';
19   elseif new.data_Recensione < data_Acq then
20     SIGNAL sqlstate '45001' SET message_text = 'Controlla la data della recensione';
21   elseif new.data_Recensione < data_Pubb then
22     SIGNAL sqlstate '45001' SET message_text = 'Non puoi recensire un gioco che non è ancora uscito';
23   elseif num >=1 then -- si puo ommettere
24     SIGNAL sqlstate '45001' SET message_text = 'Non puoi recensire più di una volta lo stesso gioco';
25   end if;
26 end$$
27 DELIMITER ;

```

Questo Trigger impone dei vincoli sulle recensioni:

- Non si può recensire un gioco se non è stato comprato o se non è stato giocato dall'utente per almeno un'ora.
- Un utente non può recensire lo stesso gioco più di una volta.
- Vincoli sulle date delle recensioni.

Le righe commentate potevano anche essere ommesse perché è impossibile per un utente recensire lo stesso gioco più di una volta dato che si avrebbe una chiave primaria duplicata. Tuttavia si è deciso di lasciarle per rendere più chiaro l'errore a chi inserisce i dati.

```

DELIMITER $$

Create TRIGGER VincoloVersione
before insert on versione
for each row
begin
    DECLARE dataRelease DATE;
    DECLARE dataReleasePrec DATE;
    DECLARE dataBeta DATE;
    DECLARE dataAlfa DATE;
    DECLARE str varchar(10);
    SELECT tipo into str from versione where Gioco=new.Gioco order by tipo desc limit 1;
    SELECT data_Versione into dataRelease from versione where Gioco=new.Gioco and tipo='release' and versione=new.versione;
    SELECT data_Versione into dataBeta from versione where Gioco=new.Gioco and tipo='beta' and versione=new.versione;
    SELECT data_Versione into dataAlfa from versione where Gioco=new.Gioco and tipo='alfa' and versione=NEW.versione;
    SELECT data_Versione into dataReleasePrec from versione where Gioco=new.Gioco and tipo='release' and versione=(left(new.versione,1) - 1);

    if new.tipo='beta' then
        if dataRelease is not null and new.data_Versione > dataRelease then
            SIGNAL sqlstate '45001' SET message_text = 'La beta deve essere più vecchia della release della stessa versione';
        elseif dataAlfa is not null and NEW.data_Versione < dataAlfa then
            SIGNAL sqlstate '45001' SET message_text = 'La alfa deve essere più vecchia della beta della stessa versione';
        elseif dataReleasePrec is not null and new.data_Versione < dataReleasePrec then
            SIGNAL sqlstate '45001' SET message_text = 'La release della versione precedente deve essere più vecchia della beta';
        end if;
    end if;

    if new.tipo='alfa' then
        if dataBeta is not null and new.data_Versione > dataBeta then
            SIGNAL sqlstate '45001' SET message_text = 'La alfa deve essere più vecchia della beta della stessa versione';
        elseif dataReleasePrec is not null and new.data_Versione < dataReleasePrec then
            SIGNAL sqlstate '45001' SET message_text = 'La release della versione precedente deve essere più vecchia della alfa';
        elseif dataRelease is not null and new.data_Versione > dataRelease then
            SIGNAL sqlstate '45001' SET message_text = 'La alfa deve essere più vecchia della release della stessa versione';
        end if;
    end if;

    if new.tipo='release' then
        if dataReleasePrec is not null and new.data_Versione < dataReleasePrec then
            SIGNAL sqlstate '45001' SET message_text = 'La release della versione precedente deve essere più vecchia della release';
        elseif dataAlfa is not null and NEW.data_Versione < dataAlfa then
            SIGNAL sqlstate '45001' SET message_text = 'La alfa deve essere più vecchia della release della stessa versione';
        elseif dataBeta is not null and new.data_Versione < dataBeta then
            SIGNAL sqlstate '45001' SET message_text = 'La beta deve essere più vecchia della release della stessa versione';
        end if;
    end if;
end$$
delimiter ;

```

Questo Trigger serve per rispettare i vincoli sulle versioni.

Es: La alfa 1.00 deve avere una data di uscita precedente alla beta 1.00 (se esiste) o della release 1.00.

La beta 2.00 deve avere una data di uscita precedente alla release 2.00 (se esiste) ma deve avere una data successiva della alfa 2.00, della release 1.00 e così via.