Università degli Studi di Trieste - Dipartimento di Ingegneria e Architettura

Animal Café

Progetto di Basi di Dati

IN0501184
Anna Guccione
Maggio 2024

Sommario

1. Presentazione del progetto

1.1. Introduzione

Si intende realizzare un database destinato a supportare "Animal Café", un'ipotetica applicazione mobile di gioco di tipo simulazione in cui gli utenti sono proprietari di un bar frequentato da animali antropomorfi, ispirati al famoso gioco "Animal Crossing".

Contesto

Durante la partita, l'utente deve versare gli ingredienti dell'ordine effettuato dal cliente uno a uno, cercando di raggiungere precisamente la quantità richiesta dal cliente. Per ogni ordine vengono calcolate le stelle e il prezzo totale della bevanda. Le stelle corrispondono al numero di ingredienti di cui si è rispettata precisamente la percentuale richiesta dal cliente. Il prezzo totale della singola bevanda viene calcolato a partire dal prezzo base della bevanda, aggiungendo la mancia del cliente per ogni stella ottenuta.

Alla fine della partita vengono calcolati il punteggio e il guadagno totale. Il punteggio corrisponde alla somma delle stelle ottenute. Il guadagno corrisponde alla somma dei prezzi di ogni bevanda venduta a cui viene sottratta la somma delle spese sulle materie prime.

Il punteggio dell'utente viene aggiornato ad ogni fine partita, sommandoci quello ottenuto. Raggiunto un certo punteggio si aumenta di livello.

Requisiti del database

Ogni **utente** del gioco possiede un profilo, caratterizzato da un nome, un avatar, il livello di esperienza e un portafoglio virtuale. Gli utenti possono accedere con un nickname e una password e stringere amicizia con altri utenti.

I giocatori possono avviare una **partita** all'interno del gioco, ambientata nel proprio bar, ciascuna caratterizzata da una durata, dal denaro guadagnato e da un punteggio totale.

Ogni **cliente** del bar è un animale, identificato da un nome, da una specie e da una personalità, che influisce sul suo dialogo di interazione e sulla mancia che può dare. Durante le partite, ciascun cliente effettua un **ordine**, che è caratterizzato dal tipo di bevanda, dal prezzo finale e dalle stelle che l'utente ottiene durante il gioco.

Il menù del bar comprende una varietà di **bevande**, ognuna caratterizzata da un tipo, da un prezzo e da una difficoltà. La bevanda è composta da un certo numero di **ingredienti** che possono essere presenti in percentuali variabili e che sono caratterizzati da un nome e da una categoria.

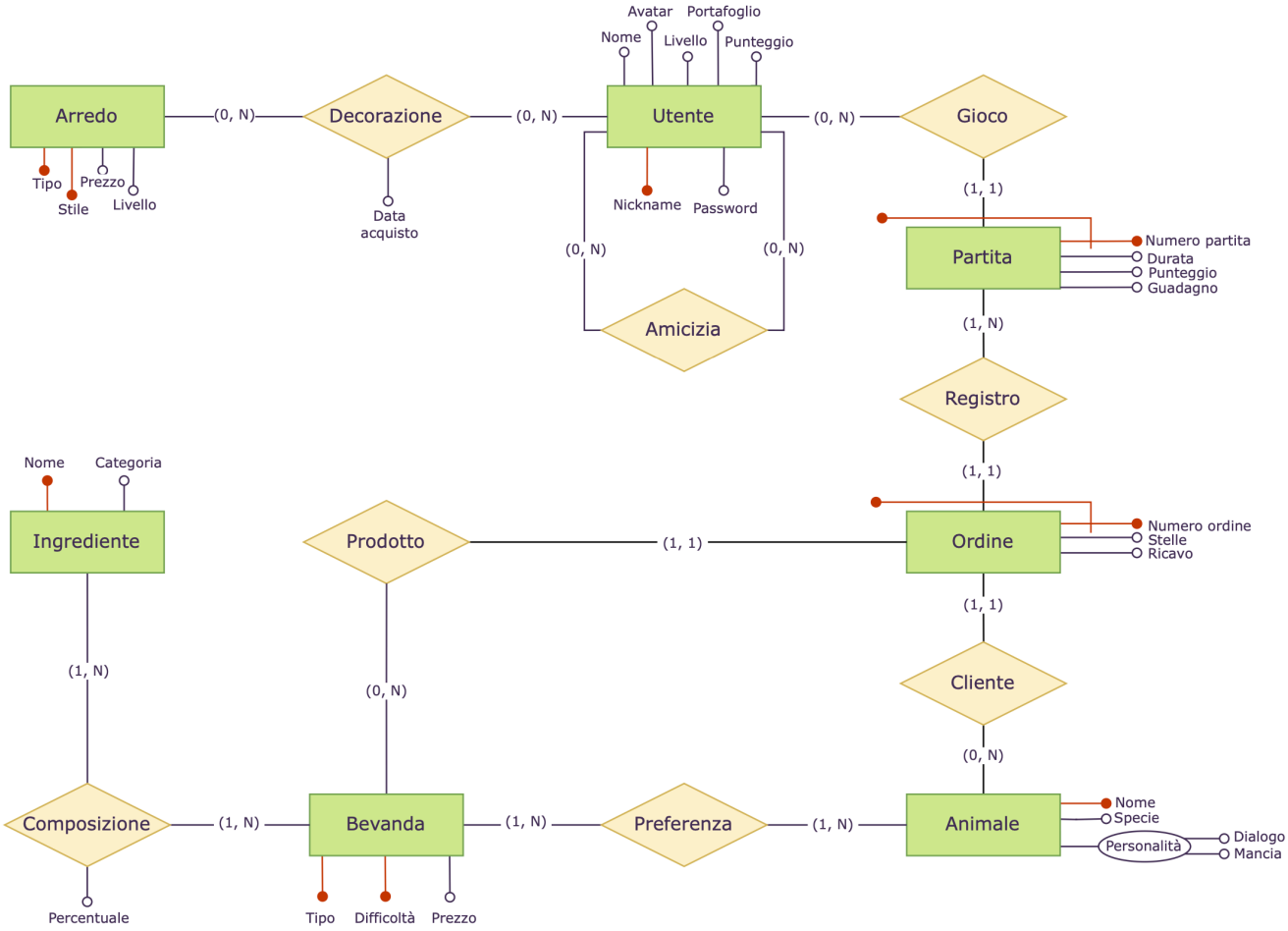
Il livello dell'utente aumenta in base al **punteggio**. Aumentare di livello determina l'accesso ad acquisti nello shop di nuovi arredi per il bar, ognuno caratterizzato da un tipo, uno stile, un prezzo e dal livello di accesso.

1.2. Glossario dei termini

Termine	Definizione	Sinonimi	Collegamento
Utente	Account della persona che si iscrive nel gioco	Giocatore	Partita, Punteggio, Portafoglio
Bar	Luogo in cui è ambientata la partita		Partita, Arredo
Punteggio	Esperienza dell'utente ottenuta dalle partite		Utente, Partita, Stelle
Portafoglio	Bilancio dell'utente		Utente
Partita	Giornata di gioco nel bar		Utente, Bar, Ordine, Punteggio
Animale	Cliente che ordina una bevanda al bar	Cliente	Bevande, Ordine
Ingrediente	Elemento che compone la bevanda		Bevande
Bevanda	Prodotta descritti nel menù	Prodotto	Animale, Ingredienti, Ordine
Ordine	Ordinazione della bevanda effettuata dal cliente		Partita, Animale, Bevande, Stelle
Stelle	Punteggio dell'utente su ogni ordine	Punti	Punteggio, Ordine
Arredo	Decorazioni per il bar che si possono comprare		Bar

2. Schema concettuale

2.1. Diagramma Entity-Relationship



2.2. Dizionario dei dati

Entità

Nome	Descrizione	Attributi	Identificatore
Utente	Account della persona che si iscrive nel gioco	Nome, Avatar, Livello, Portafoglio, Punteggio, Nickname, Password	Nome
Arredo	Prodotto decorativo per il bar	Tipo, Stile, Prezzo, Livello	Tipo, Stile
Partita	Giornata di gioco nel bar	Numero partita, Durata, Punteggio, Guadagno	Numero partita, Gioco (R)

Ordine	Singola richiesta di una bevanda da parte di un cliente	Numero ordine, Stelle, Ricavo	Numero ordine, Registro (R)
Animale	Cliente che ordina una bevanda al bar	Nome, Specie, Personalità	Nome
Bevanda	Prodotto descritto nel menù	Tipo, Difficoltà, Prezzo	Tipo, Difficoltà
Ingrediente	Elemento che compone la bevanda	Nome, Categoria	Nome

Relationship

Nome	Descrizione	Componenti	Attributi
Amicizia	Rapporto tra due account	Utente	
Decorazione	L'utente può decorare il proprio bar con degli arredi	Utente, Arredo	Data acquisto
Gioco	L'utente gioca a una partita	Utente, Partita	
Registro	In una partita si registra una sequenza di ordini	Partita, Ordine	
Cliente	Il cliente dell'ordine è un animale	Ordine, Animale	
Preferenza	L'animale ha delle bevande di preferenza	Animale, Bevanda	
Prodotto	Il prodotto richiesto in un ordine è una bevanda	Ordine, Bevanda	
Composizione	La bevanda è composta da più ingredienti	Bevanda, Ingrediente	Percentuale

2.3. Vincoli non espirimibili graficamente

- Il rapporto di amicizia tra due account dev'essere simmetrico.
- In una sequenza d'ordine di una partita, uno stesso cliente compare solo una volta.
- Il livello del giocatore viene aumentato al raggiungimento di un certo punteggio.
- Il bar può essere decorato da al massimo un arredo per tipo.
- Il portafoglio dell'utente non può avere un bilancio negativo.

3. Progettazione logica

3.1. Operazioni d'interesse

Azione	Tipo	Frequenza
Generazione di una partita	Interattiva	3/giorno
Gestione degli ordini di una partita	Interattiva	15/giorno
Decorazione del bar con un arredo acquistato	Interattiva	1/settimana
Visualizzazione della classifica amici	Interattiva	1/settimana
Registrazione di una partita	Batch	3/giorno
Pulizia dei dati storici di partite e ordini	Batch	1/settimana
Report mensile degli account che utilizzano l'app	Batch	1/mese

3.2. Tavola dei volumi

Concetto	Tipo	Volume
Utente	E	2.000
Amicizia	R	3.000
Decorazione	R	10
Arredo	E	100
Gioco	R	4.000
Partita	E	4.000
Registro	R	20.000
Ordine	E	20.000
Prodotto	R	20.000
Cliente	R	20.000
Bevanda	E	30
Animale	E	400
Preferenza	R	1.600
Composizione	R	60
Ingredienti	E	10

3.3. Ristrutturazione

Analisi delle ridondanze

- **Analisi dell'attributo Punteggio in Partita**

  Si osserva che l'attributo punteggio in *Partita* è la somma delle stelle degli *Ordini* di una stessa partita e inoltre che l'attributo *Punteggio* di *Utente* è la somma dei punteggi delle partite. Si decide di eliminare l'attributo *Punteggio* in *Partita* in quanto calcolabile e poiché, considerando l'operazioni "Registrazione di una partita", l'accesso aggiuntivo di scrittura in *Partita* risulta superfluo.

  Tuttavia si decide di mantenerlo in *Utente* per ottimizzare l'operazione di aumento di livello automatico.

- **Analisi dell'attributo Guadagno in Partita**

  Per lo stesso ragionamento, si osserva che l'attributo *Guadagno* di *Partita* è ridondante in quanto somma dei ricavi dei singoli ordini. Quindi si decide di toglierlo per evitare l'accesso aggiuntivo di scrittura in *Partita*.

- **Analisi dei cicli**
  Si vede che c'è un ciclo *Ordine-Cliente-Animale-Preferenza-Bevanda-Prodotto-Ordine*, per cui si decide di eliminare la relazione n-aria *Preferenza* in quanto facilmente ricavabile dalle altre relazioni e possibile appesantimento del sistema.

Partizionamento di entità o relazioni

- **Partizionamento verticale di entità**

  Ci si accorge che gli attributi di *Utente* sono divisibili in due categorie: quelli riferiti all'utente in partita e quelli riferiti al suo account online. Si decide di creare una seconda entità *Account* e di unirle con la relazione uno a uno *Online*. Gli attributi di online saranno *Nickname*, *Password* e *Livello*; ad Account sarà inoltre associata la relazione ricorsiva *Amicizia* per ottimizzare l'operazione di "Visualizzazione della classifica amici", che dipende da *Livello*.

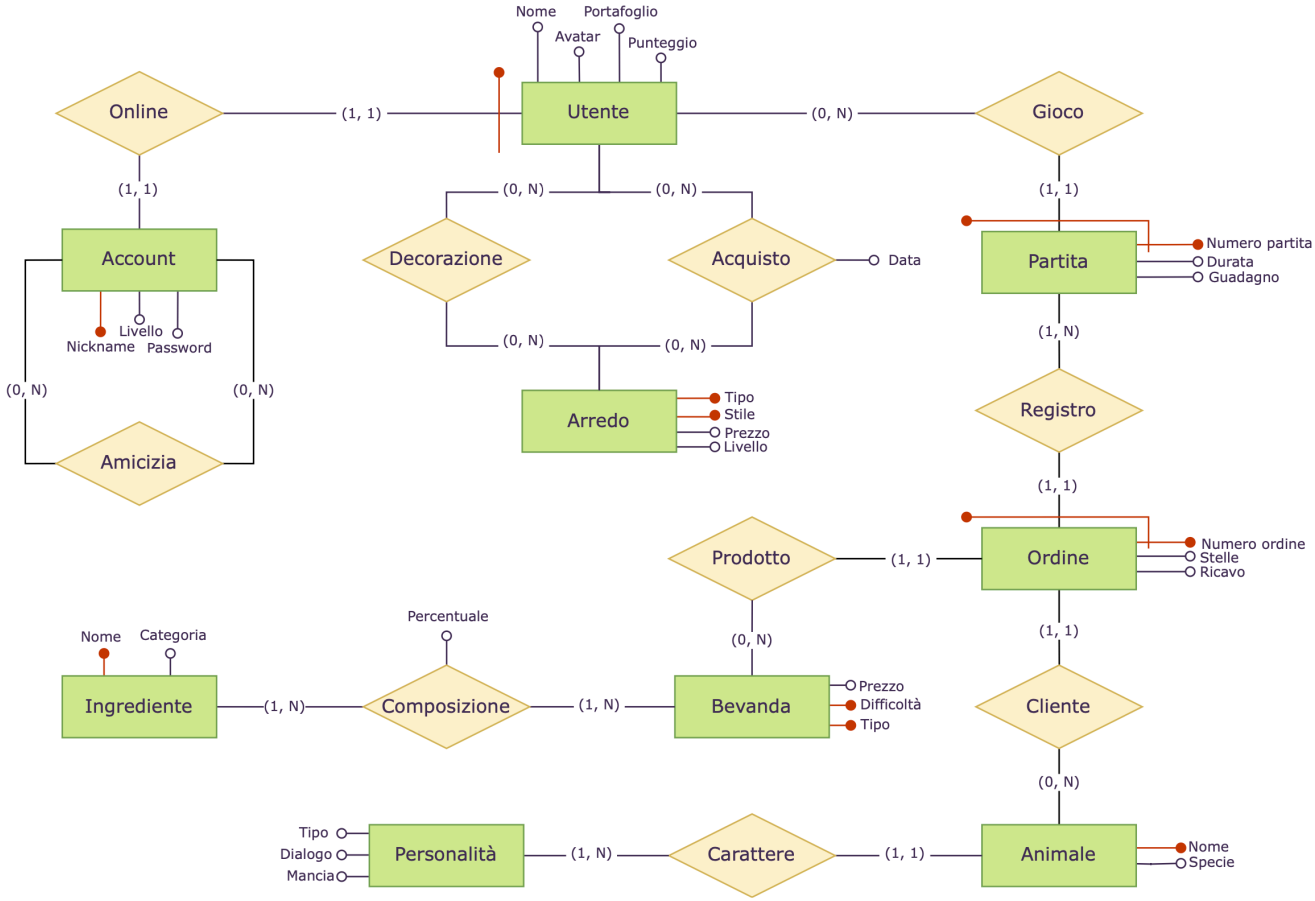
- **Eliminazione di attributi multivalore**

  Si vuole eliminare l'attributo multivalore *Personalità* di *Animale*, in quanto non supportata nello schema logico relazionale. Per questo si crea un'entità separata con relazione *Tipo* e attributi *Dialogo* e *Mancia*.

- **Partizionamento orizzontale di relazione**

  Si nota che nella relazione *Decorazione* la *Data acquisto* potrebbe essere nulla. Si sceglie quindi di separare la relazione in due *Acquisto* e *Decorazione*, una che rappresenta l'acquisto, l'altra la decorazione del bar. Questo anche in previsione del fatto che queste operazioni avvengono separatamente.

3.4. Diagramma Entity-Relationship ristrutturato



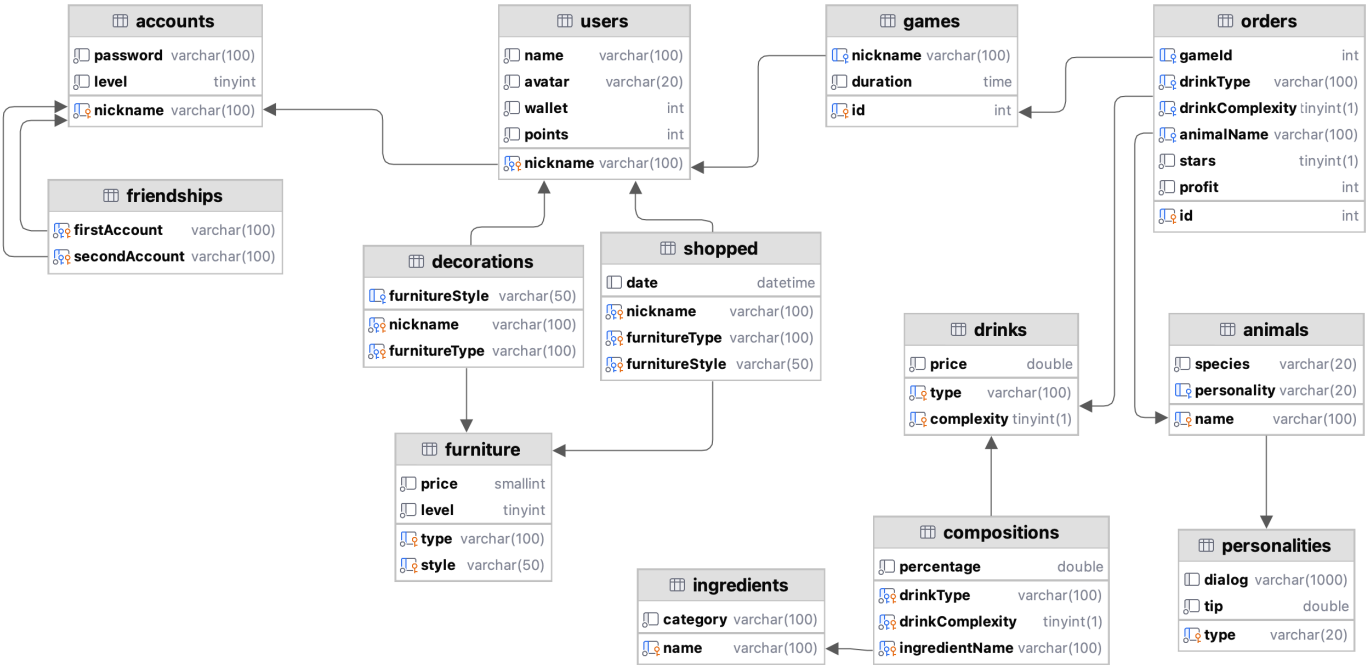
4. Schema logico

4.1. Passaggio al modello relazionale

Si individuano le seguenti relazioni, identificatori e attributi per lo schema logico con il modello relazionale:

- **Account** → (Nickname, Password, Livello)
- **Amicizie** → (*Nickname*, *Nickname*)
- **Utenti** → (*Nickname*, Nome, Avatar, Portafoglio, Punteggio)
- **Acquisti** → (*Nickname*, *Tipo arredo*, *Stile*, Data)
- **Decorazioni** → (*Nickname*, *Tipo*, *Stile*)
- **Arredi** → (Tipo, Stile, Prezzo, Livello)
- **Partite** → (Numero, *Nickname*, Durata, Guadagno)
- **Ordini** → (Numero, *Numero partita*, *Tipo bevanda*, *Difficoltà bevanda*, *Nome animale*, Stelle, Ricavo)
- **Bevande** → (Tipo, Difficoltà, Prezzo)
- **Composizione** → (*Tipo bevanda*, *Difficoltà bevanda*, *Nome ingrediente*, Percentuale)
- **Ingredienti** → (Nome, Categoria)
- **Animali** → (Nome, Specie, *Tipo personalità*)
- **Personalità** → (Tipo, Dialogo, Mancia)

4.2. Schema logico



4.3. Normalizzazione

1. Si osserva che il database è già in prima forma normale, in quanto tutte le colonne sono atomiche.
2. Si osserva che il database è già in seconda forma normale, in quanto ciascuna colonna dipende interamente dalla primary key.
3. Si osserva che nella tabella Ordini non è rispettata la terza forma normale, perché la colonna Ricavo è un campo calcolato che risulta dipendere dalla Bevanda e dalle Stelle: infatti il prezzo viene calcolato a partire dal prezzo base della bevanda a cui viene aggiunta la mancia in base a quante stelle si guadagna. La soluzione potrebbe consistere nel creare una tabella separata che tenga conto di tutte le possibili mance e quindi di tutti i possibili prezzi per ogni bevanda, il che risulta una complicazione eccessiva per il database che non porta vantaggi. Si decide quindi di matenere il campo calcolato per semplificare la logica e le operazioni.

5. Codice SQL

5.1. Operazioni

Operazione 1: generazione della partita

Stored procedure che genera una partita vuota.

```
CREATE PROCEDURE createGame(IN n varchar(100), OUT gid INT)
BEGIN
    INSERT INTO games (nickname) VALUE (n);
    SELECT id INTO gid FROM games WHERE nickname = n ORDER BY id DESC LIMIT 1;
END $$
```

Stored procedure che genera casualmente il cliente e la bevanda che viene ordinata nell'ordine. Si osserva che si è aggiunto un controllo che il cliente generato non sia già comparso nella stessa partita per rispettare il vincolo non esprimibile.

```
CREATE PROCEDURE generateOrder(IN gid INT, OUT oid INT)
BEGIN
    DECLARE an varchar(100);
    DECLARE dt varchar(100);
    DECLARE dc INT;
    SELECT name INTO an FROM animals WHERE name NOT IN
    (SELECT animalName FROM orders WHERE gameId = gid)
    ORDER BY RAND() LIMIT 1;
    SELECT type, complexity INTO dt, dc FROM drinks ORDER BY RAND() LIMIT 1;
    INSERT INTO orders (gameId, drinkType, drinkComplexity, animalName) VALUE
(gid, dt, dc, an);
    SELECT id INTO oid FROM orders WHERE gameId = gid ORDER BY id DESC LIMIT 1;
END $$
```

Operazione 2: gestione degli ordini

Stored procedure che aggiorna l'ordine in base al risultato del giocatore.

```
CREATE PROCEDURE updateOrder(IN oid INT, IN s INT)
BEGIN
    UPDATE orders SET stars = s, profit = (getPrice(oid) + getTip(oid) * s) WHERE
id = oid;
END $$
```

Funzioni per semplificare la sintassi.

```
-- Trova il prezzo della bevanda ordinata
CREATE FUNCTION getPrice(oid INT)
RETURNS INT
DETERMINISTIC
BEGIN
    DECLARE p INT;
    SELECT price INTO p FROM orders
    INNER JOIN drinks ON drinkType = type AND drinkComplexity = complexity
    WHERE id = oid;
    RETURN p;
END;

-- Trova la mancia del cliente
CREATE FUNCTION getTip(oid INT)
RETURNS INT
DETERMINISTIC
BEGIN
    DECLARE t INT;
    SELECT tip INTO t FROM orders
    INNER JOIN animals ON animalName = name
    INNER JOIN personalities ON personality = type
    WHERE orders.id = oid;
    RETURN t;
END;
```

Operazione 3: registrazione della partita

Stored procedure che conclude la partita stampando i risultati e aggiornando i dati del giocatore.

```
CREATE PROCEDURE endGame(IN gid INT, IN d TIME)
BEGIN
    UPDATE games SET duration = d WHERE id = gid;
    SELECT * FROM GameDetails WHERE id = gid;
    UPDATE users SET points = points + gamePoints(gid), wallet = wallet +
gameProfits(gid)
    WHERE nickname = (SELECT nickname FROM games WHERE id = gid);
END $$
```

Si osserva che qui si è utilizzata una vista per i dettagli delle partite.

```
CREATE VIEW GameDetails AS
SELECT id, nickname, duration, gamePoints(id) AS points, gameProfits(id) AS profit
FROM games;
```

Funzioni per calcolare il punteggio e i profitti della partita.

```
-- Trova i punti della partita
CREATE FUNCTION gamePoints(gid INT)
RETURNS INT
DETERMINISTIC
BEGIN
    DECLARE p INT;
    SELECT SUM(stars) INTO p FROM orders WHERE gameId = gid GROUP BY gameId;
    RETURN p;
END;

-- Trova i profitti della partita
CREATE FUNCTION gameProfits(gid INT)
RETURNS INT
DETERMINISTIC
BEGIN
    DECLARE p INT;
    SELECT SUM(profit) INTO p FROM orders WHERE gameId = gid GROUP BY gameId;
    RETURN p;
END;
```

Operazione 4: gestione delle decorazioni

Stored procedure per decorare il bar, che controlla che il mobile sia prima acquistato.

```
CREATE PROCEDURE decorateBar(IN n varchar(100),IN t varchar(100),IN s varchar(50))
BEGIN
    DECLARE control varchar(50);
    SELECT s INTO control FROM shopped
        WHERE nickname = s AND furnitureType = t AND furnitureStyle = s;
    IF control IS NULL THEN
        SIGNAL SQLSTATE '45001' SET message_text = 'Item not bought.';
    END IF;
    UPDATE decorations SET furnitureStyle = s WHERE nickname = n AND furnitureType
= t;
END $$
```

Operazione 5: visualizzazione delle classifiche amici

Stored procedure che visualizza la classifica degli amici dell'utente fornito.

```
CREATE PROCEDURE viewRankings(IN n varchar(100))
BEGIN
    SELECT DISTINCT nickname, level, rank() over (order by level desc) as ranking
from
    (SELECT DISTINCT a.nickname, a.level FROM accounts a INNER JOIN friendships
     ON a.nickname = firstAccount OR a.nickname = secondAccount WHERE firstAccount
= n
     OR secondAccount = n) as rankedFriends ORDER BY level DESC;
END $$
```

Operazione 6: pulizia dei dati storici

Stored procedure che elimina le partite (e quindi gli ordini) tranne le ultime due per ogni utente.

```
CREATE PROCEDURE cleanHistory()
BEGIN
DELETE FROM games WHERE (nickname, id) NOT IN (select nickname, id from
(SELECT g.* FROM games g WHERE (SELECT count(*) FROM games WHERE nickname =
g.nickname AND id >= g.id) <= 2) AS last2games);
END $$
```

Operazione 7: report degli utenti

Stored procedure che stampa il numero di utenti che si sono iscritti all'app.

```
CREATE PROCEDURE usersReport()
BEGIN
    SELECT count(*) AS 'users count' FROM users;
END $$
```

Operazione 8: aggiornamento del livello (per il trigger)

Stored procedure per modificare il livello di un utente e funzione che determina per quali punti effettuare il passaggio di livello.

```
CREATE PROCEDURE updateLevel(IN n VARCHAR(100))
BEGIN
    UPDATE accounts a JOIN users u USING (nickname)
    SET level = getLevel(points) WHERE a.nickname = n;
END $$

-- Definisce i passaggi di livello
CREATE FUNCTION getLevel(p INT)
RETURNS INT
DETERMINISTIC
BEGIN
    DECLARE lvl INT;
    IF p >= 500 THEN SET lvl = 10;
    ELSEIF p >= 400 THEN SET lvl = 9;
    ELSEIF p >= 300 THEN SET lvl = 8;
    ELSEIF p >= 250 THEN SET lvl = 7;
    ELSEIF p >= 200 THEN SET lvl = 6;
    ELSEIF p >= 150 THEN SET lvl = 5;
    ELSEIF p >= 100 THEN SET lvl = 4;
    ELSEIF p >= 70 THEN SET lvl = 3;
    ELSEIF p >= 40 THEN SET lvl = 2;
    ELSEIF p >= 10 THEN SET lvl = 1;
    ELSE SET lvl = 0;
    END IF;
    RETURN lvl;
END;
```

5.2. Trigger

Trigger 1: amicizie simmetriche

Controllo prima di un inserimento nella tabella delle amicizie perché l'ordine sia alfabetico, in modo che non ci siano duplicati.

```
CREATE TRIGGER beforeInsertFriendships
BEFORE INSERT ON friendships
FOR EACH ROW
BEGIN
    IF NEW.firstAccount > NEW.secondAccount THEN
        SET @temp = NEW.firstAccount;
        SET NEW.firstAccount = NEW.secondAccount;
        SET NEW.secondAccount = @temp;
    END IF;
END $$
```

Trigger 2: aggiornamento del livello

Controllo che viene fatto ad ogni modifica dei punteggio degli utenti per controllare che si debba fare il passaggio di livello.

```
CREATE TRIGGER afterUpdatePoints
AFTER UPDATE ON users
FOR EACH ROW
BEGIN
    IF NEW.points != OLD.points THEN
        CALL updateLevel(NEW.nickname);
    END IF;
END $$
```

Trigger 3: aggiornamento portafoglio

Trigger che viene eseguito a seguito di un acquisto per modificare il portafoglio dell'utente.

```
CREATE TRIGGER afterInsertShopped
AFTER INSERT ON shopped
FOR EACH ROW
BEGIN
    DECLARE itemPrice INT;
    SELECT price INTO itemPrice FROM furniture
        WHERE type = NEW.furnitureType AND style = NEW.furnitureStyle;
    UPDATE users SET wallet = wallet - itemPrice
        WHERE nickname = NEW.nickname;
END $$
```