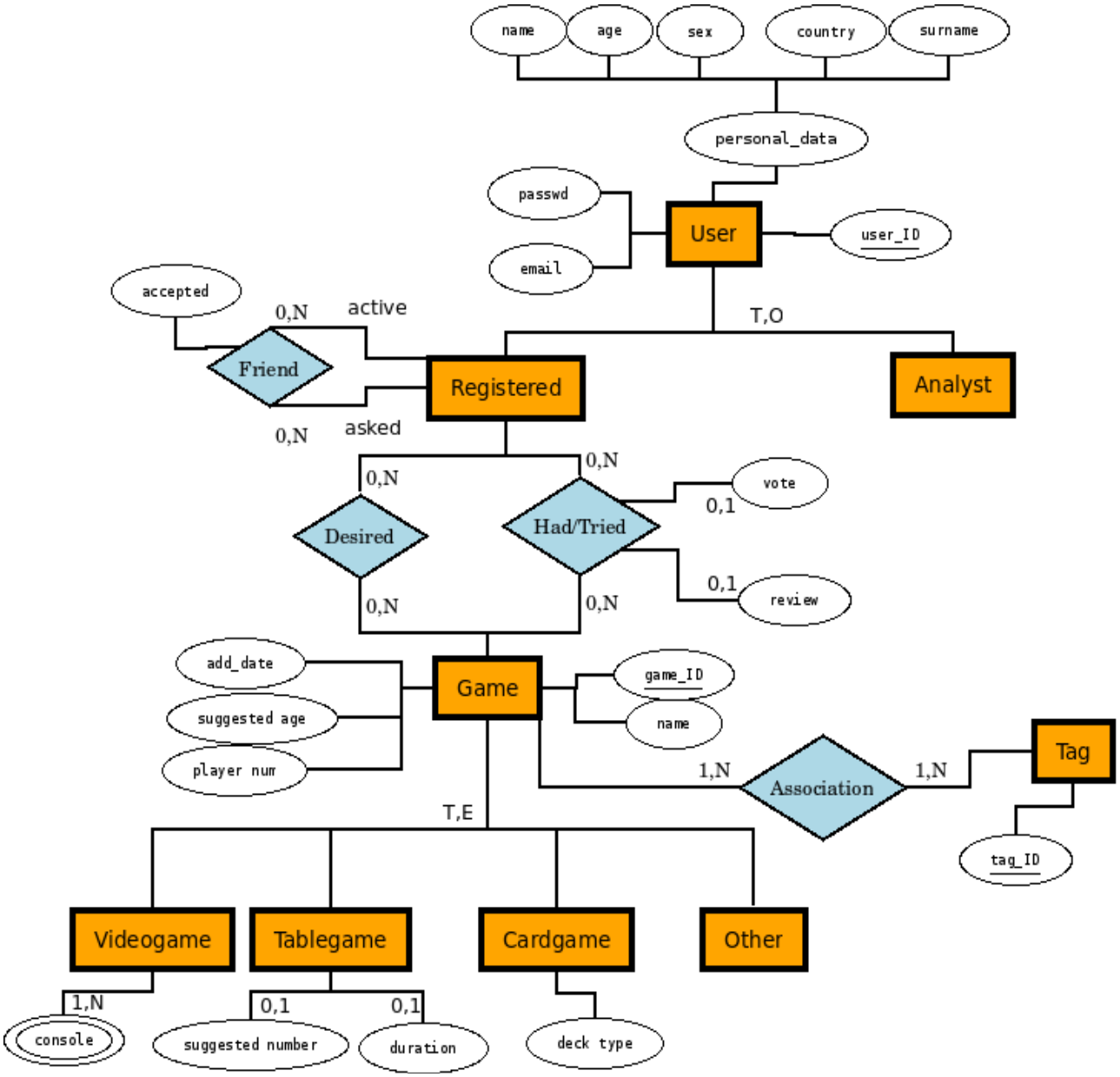
Progetto di Laboratorio di Basi di Dati

RELAZIONE TECNICA di “GIOGU”

Di Michele Corrias, 741863

1 Progettazione Concettuale

1.1 Schema ER e giustificazioni



Premessa: su ogni entità importante è stato scelto di utilizzare un ID numerico come chiave primaria, piuttosto che una stringa di lunghezza variabile. Ad esempio, per identificare univocamente un utente sarebbe stata sufficiente la propria mail, ma sicuramente mantenere in memoria un integer è più performante di un varchar.

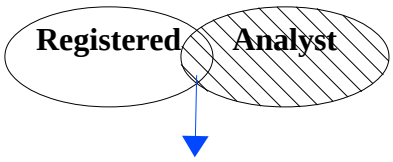
User è una grossa tabella che contiene i dati minimi che deve avere ogni utente dell'applicazione.

L'insieme degli utenti è esaurito da **Registered**, **Analyst** e **Admin**: per questo abbiamo sicuramente una gerarchia totale.

E' previsto che i Registered non possano utilizzare le funzioni di statistica e monitoraggio della comunità, e viceversa gli Analyst non possano richiedere amicizie o interagire con i giochi dell'applicazione. Quindi si potrebbe supporre una gerarchia esclusiva.

Tuttavia ci sono anche gli Admin, utenti con privilegi maggiori: modificano e/o cancellano i dati inseriti dagli altri. In più possono essere anche gli altri utenti generici contemporaneamente: a loro verranno quindi concessi i privilegi di entrambi iscritti e analisti.

Questa analisi porta alla conclusione di utilizzare una gerarchia totale overlapping (sovrapposta): ci sarà una parte di utenti solo iscritti, una parte di solo analisti, e infine un sottoinsieme generato dall'intersezione dei due insiemi precedenti, che individua l'insieme degli amministratori.



Admin

Anche se, a livello ER, non si evince il fatto che gli admin siano utenti con privilegi maggiori di utenti iscritti e analisti, questi poteri verranno concessi in futuro.

Lo schema ER è limitato da vincoli rigidi, pertanto non può modellare la realtà in tutta la sua complessità.

Friend è un'associazione ricorsiva, di cardinalità 0,N da entrambi i lati, poiché un iscritto può non avere amici come averne molti.

Naturalmente è un'associazione simmetrica: se A è amico di B, B è amico di A. Per logiche di minimizzazione di spazio, viene mantenuta solo una direzione della relazione.

accepted è un attributo che indica se una relazione è approvata o in fase di approvazione [2.7 per dettagli].

Le associazioni tra le entità *Registered* e *Game* sono state ridotte a due soltanto: **Desired** e **Had/Tried**.

Desired indica che un utente desidera provare e/o possedere un gioco: non ha senso, infatti, prevedere tutte le opzioni. Ai fini dell'applicazione non è importante distinguere il “desiderare provare” dal “desiderare possedere”: l'utente segnala i giochi genericamente desiderati, e basta.

Infatti nella realtà una persona non desidera possedere un gioco e poi non provarlo. Come del resto, non desidera provare un gioco e poi non possederlo, perché se desidera provarlo si presume che al soggetto piaccia minimamente il gioco desiderato.

Infine, è logicamente scontato che una persona desideri possedere e provare contemporaneamente un gioco. Il “desiderare di possedere” sussume in sé il “desiderare di provare”, e per questo abbiamo un'unica associazione “desiderare”, che contiene in sé il desiderare possedere.

Had/Tried indica che un utente ha provato e/o possiede un gioco: come sopra, non ha senso, infatti, prevedere associazioni diverse per tutte le opzioni. Ai fini dell'applicazione Web non è importante distinguere il “provare” dal “possedere” un gioco: l'utente segnala i giochi genericamente in una lista di provati/posseduti, perché così li può recensire e valutare, e basta.

E' quindi importante solo il distinguere le due liste, cioè, la lista dei desiderati (che sono solo desiderati, e quindi non possono essere recensiti o valutati) dalla lista dei posseduti/provati.

A tal proposito, i due attributi r*eview* e v*ote* hanno ragione di esistere solo sulla associazione di Had/Tried .

Analizzando le cardinalità delle associazioni, entrambe sono N:N (molti-a-molti), poiché più utenti possono essere in relazione con più giochi:

- uno stesso utente può desiderare o possedere/provare più giochi;
- lo stesso gioco può essere desiderato o posseduto/provato da più utenti;
- la cardinalità minima per entrambe è 0, opzionale, perché un utente può non desiderare o possedere/provare giochi, e viceversa.

I due attributi r*eview* e *vote* hanno una cardinalità opzionale, cioè 0,1, perchè "per ogni gioco posseduto/provato, ogni utente può mettere a disposizione una recensione e una valutazione " (cit. specifiche).

Game è l'altra entità principale dello schema ER, insieme a User. Possiede gli attributi generici di ogni gioco.

L'attributo *add_date* di Game serve ai fini dell'applicazione, per eliminare i giochi inseriti ma non segnalati da un numero minimo di utenti entro un certo tempo, così da evitare un numero eccessivo di giochi inesistenti [2.12 Autocancellazione per dettagli].

La tipologia dei giochi è stata prevista, fin da subito, come una gerarchia di *Game*, perché un gioco si suddivide in più varianti: **Videogame**, **Tablegame**, **Cardgame**, **Other**.

C'era la necessità di esprimere le diverse tipologie come entità indipendenti, poiché ognuna ha i propri attributi specifici: ad esempio videogioco ha una piattaforma di riferimento, il che non c'entra nulla con un gioco da tavolo.

Videogame ha un attributo multivalore *console*, perchè "i videogiochi hanno necessariamente una o più piattaforme di riferimento " (cit. specifiche).

Tablegame possiede gli attributi *suggested_number* e *duration*, entrambi con cardinalità opzionale, cioè 0,1: questo perché "i giochi da tavola possono avere un numero di giocatori suggerito e una durata prevista" (cit. specifiche).

Cardgame ha un attributo *deck*, che specifica quale tipo di mazzo dovrebbe essere usato per il gioco da carte previsto.

Grazie all'introduzione della tipologia **Other** è stato possibile classificare la gerarchia come Totale, in quanto, un gioco da inserire che non rientra in una particolar tipologia delle sopra elencate, allora trova qui il suo posto. Naturalmente tale gerarchia è esclusiva, perché un gioco è di un unico tipo. Quindi T,E.

Un vincolo che non si evince dallo schema ER, ma comunque presente nell'implementazione, è che un gioco è considerato unico in base alla coppia (nome,tipologia): questo è assicurato da un vincolo row-level [2.2.1 per dettagli].

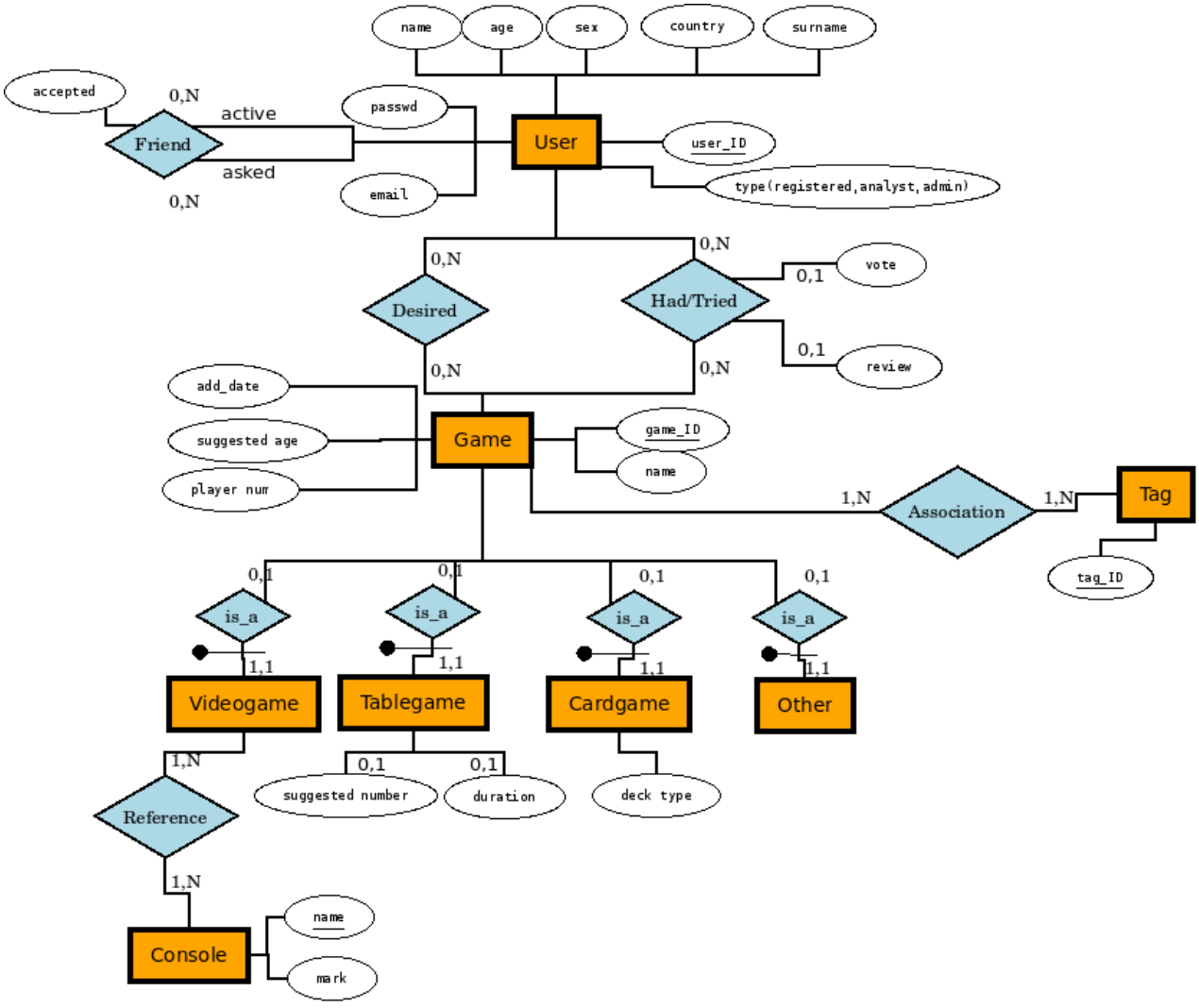
Il vincolo ha lo scopo di evitare il proliferare di giochi doppi: infatti

- due giochi sono simili se hanno nome simile;
- due giochi sono uguali se hanno stesso nome e stessa tipologia, indipendentemente dagli altri dati

Questo criterio è stato utilizzato nell'inserimento guidato [4.6 Inserimento guidato].

Tag è un'entità associata a *Game* dall'entità *Associazione*: questa relazione esplicita il fatto che un gioco può essere taggato con più tags.

1.2 Ristrutturazione ER e giustificazioni



1.2.1 User

Per la ristrutturazione della gerarchia su User, si è scelto il metodo del mantenimento della sola superclasse, aggiungendo un attributo *type* per distinguere le varie categorie di utenza.

Il motivo della scelta è dettato dal fatto che la ristrutturazione dello schema ER dipende da logiche di efficienza. Mentre lo schema ER cerca di rispettare tutti i vincoli delle specifiche di progetto, la ristrutturazione punta sull'efficienza di implementazione: siamo tutti consci del fatto che:

- un'analista non può richiedere amicizie o segnalare giochi, ma iscritti e amministratori sì.
- un normale utente iscritto non ha accesso al monitoraggio della comunità, ma analisti e amministratori sì.

Questo concetto si perde un po' con la ristrutturazione, in quanto tutte le relazioni proprie dell'utente *Registered* vengono genericamente collegate ad utente. Nello schema ER originale, però, questo vincolo è ben chiaro, quindi sarà bene fare un check a livello implementativo.

La scelta di mantenere sia superclasse che sottoclassi non è efficiente: è meglio tenere un'unica grande tabella di utenti, e associare subito l'utente alla sua categoria, piuttosto che mantenere tabelle separate per ogni categoria di utente. Questo sarebbe stato infatti, uno spreco di spazio e di prestazioni, perché ogni volta sarebbe stato indispensabile un join tra la superclasse e la sottoclasse di utenza specifica, per recuperare tutti i dati di un utente appartenente a una categoria (es: un analista).

La scelta di mantenere solo le sottoclassi è un po' illogica: non ha senso replicare praticamente tutti gli attributi di User su ogni sottoclasse, i quali tra l'altro sono molti. Meglio tenerli raggruppati ad un livello superiore.

1.2.2 Game

Nella ristrutturazione è stato scelto di mantenere sia la superclasse Game che le sottoclassi che rappresentano le varie tipologie, indicandole come entità deboli. Il motivo di questa soluzione è che Game è un'entità importante, non collassabile nelle sottoclassi a meno di sprechi di spazio incredibile, dettati dalla replica di tutti gli attributi di Game su ogni sottoclasse indicante una tipologia.

Le varie tipologie hanno invece degli attributi specifici, e *Videogame* partecipa addirittura

all'associazione *Reference* con *Console*. Quindi per la logica di minimizzazione dei valori nulli si è scelto di non mantenere solo la superclasse, in quanto per ogni tipologia gli attributi specifici delle altre tipologie sarebbero stati NULL.

L'attributo *type* su Game è stato aggiunto solo per scopi d'efficienza di implementazione ai fini dell'applicazione, per questo non compare nell'ER. Il motivo della sua aggiunta è dettato dal fatto che, per recuperare la tipologia di un gioco, sarebbe necessario scorrere tutte le tabelle delle diverse tipologie (nel caso peggiore, addirittura occorre scorrere tutte le quattro tabelle delle varie tipologie, prima di desumere che un gioco è della tipologia Other).

Pertanto, si è scelto di aggiungere un'informazione ridondante a livello superiore, per motivi di efficienza. Immediatamente ricavata la tipologia di un gioco, si può poi andare nella tabella corrispondente e ricavare tutti i valori dei suoi attributi.

1.2.3 Videogame & Console

Nella ristrutturazione l'attributo multivalore *console* di Videogame si converte in:

- l'entità **Console**, con i suoi attributi specifici: *mark* e *name*, che indicano rispettivamente marca e nome di una piattaforma;
- l'associazione **Reference**, N:N (molti a molti) tra Videogame e Console, in quanto un videogame può avere più piattaforme di riferimento, e viceversa una piattaforma è di riferimento a più videogames.

1.2.4 Personal data

L'attributo composto p*ersonal data* su Users è ristrutturato spostando i dati specifici direttamente sulla entità User, come suoi attributi.

2 Progettazione Logica

2.1 Schema Relazionale

Il DB creato per questo progetto si chiama “**giogu**”.

[0_db.sql]

Le entità sono state tradotte in relazioni definite sui loro stessi attributi .

Le associazioni N:N (molti-a-molti) sono state tradotte in relazioni a sé stanti con i propri attributi e come chiave primaria l'unione delle chiavi primarie delle associazioni coinvolte: è questo il caso di **Friend**, **Desired**, **HadTried**, **Association**, **Reference**.

Tutte le restanti associazioni, del tipo *is-a* 1:1, sono state tradotte inserendo la chiave primaria di **Game** come chiave esterna in ogni sottoclasse di tipologia. Queste sono anche entità deboli quindi la precedente chiave fa parte della loro chiave primaria (per la precisione è la chiave primaria).

Nello schema relazione sono presenti le principali tabelle, più tabelle “ausiliarie”, domini, viste, sequenze, funzioni, triggers...

2.1.1 Tabelle principali:

- Users(**uid**, email**, passwd, name, surname, age, sex, *country*, type (registered, analyst, admin))
- Friend(***uid_active*, *uid_asked***, accepted)
- Desired(***uid*, *gid***)
- HadTried(***uid*, *gid***, review*, vote*)
- Game(**gid**, name, sugg_age, n_player, add_date, type)
- VideoGame(***gid***)
- TableGame(***gid***, sugg_num*, duration*)
- CardGame(***gid***, *deck*)
- Other(***gid***)
- Console(**name**, mark)
- Reference(***gid*, *console***)
- Tag(**tid**)
- Association(***tid*, *gid***)

[LEGENDA: **chiave primaria***, chiave esterna, NULL*, UNIQUE**]*

- Users => tabella Utenti
  - uid = id del singolo utente. E' di tipo SERIAL, quindi il DBMS costruisce e gestisce autonomamente una sequenza seq_users_uid
  - mail, passwd, name, surname, age, sex, country = dati personali. In

particolare, `country` è chiave esterna su `Expected_Zone.abbr` (si veda 2.1.2 per dettagli)

- `type` = ruolo dell'utente. Implementato come smallint
  - 0 = registered (utente iscritto)
  - 1 = analyst (utente analista)
  - 2 = admin (utente amministratore)

- `Friend` = richiesta di amicizia
  - `uid_active, uid_asked` = rispettivamente chi richiede e chi riceve la richiesta. Entrambi sono chiave esterna su `Users.uid`
- `Desired` = lista di giochi desiderati
  - `uid` – `gid` = l'utente identificato con uid desidera il gioco identificato con gid
- `HadTried` = lista di giochi posseduti e/o provati
  - `review` = recensione
  - `vote` = votazione
- `Tag`
  - `tid` = nome di un singolo tag (es.: di strategia, di abilità...)
- `Association` = associazione tra il `gid` del gioco e il `tid` del tag
- `Game` = Gioco
  - `gid` = id del singolo gioco. E' di tipo SERIAL, quindi il DBMS costruisce e gestisce autonomamente una sequenza `seq_game_gid`.
  - `name` = nome del gioco
  - `sugg_age` = età suggerita
  - `n_player` = numero di giocatori richiesto
  - `add_date` = data di inserimento del gioco nel database [2.12 Autocancellazione per dettagli]
  - `type` = tipologia di gioco. Implementato come smallint

    - 0 = videogame
    - 1 = gioco da tavolo
    - 2 = gioco di carte
    - 3 = altra tipologia di gioco
- `Videogame`, `Tablegame` `Cardgame`, `Other` = tipologie
    - `sugg_num` = numero di giocatori suggerito nei giochi da tavola
    - `duration` = durata prevista nei giochi da tavola
    - `deck` = tipo di mazzo nei giochi di carte (es.: tirolesi, napoletane...). E' chiave esterna su `Expected_Card.type` (2.1.2 per dettagli)
- `Console`
    - `name` = nome della console (es.: Wii, Xbox, Play Station...)
    - `mark` = marca della console (es.: Nintendo, Microsoft, Sony...)
- `Reference` = il videogame gid fa riferimento alla piattaforma console

2.1.2 Tabelle ausiliarie

- `Expected_Card(`**`type`**`)`
- `Expectd_Zone(`**`abbr`**`, country**)`
- `Stat_Users(`***`uid`***`, n_desired, n_played)`
- `Stat_Users_Avg(`**`avg_desired, avg_played`**`)`
- `Stat_Game(`***`gid`***`, n_desiring,n_playing)`
- `Stat_Game_Age(`***`gid`***`, avg_vote, age)`
- `Stat_Game_Sex(`***`gid`***`, avg_vote, sex)`
- `Stat_Game_Country(`***`gid`***`, avg_vote, `*`country`*`)`

[3_pgsql_schema.sql]

`Expected_Card` ed `Expected_Zone` sono due tabelle previste per rappresentare due insiemi finiti di elementi che dovrebbero rispecchiare la realtà.

L'idea sarebbe che:

- **Expected_Card** dovrebbe contenere tutti i tipi di mazzo esistenti (l'attributo

deck di Cardgame è una chiave esterna su Expected_Card.type);

- **Expected_Zone** dovrebbe contenere tutte le zone geografiche di provenienza degli utenti (l'attributo country di Users è chiave esterna su Expected_Card.abbr).

In questo modo è possibile realizzare un menù "a tendina" attraverso il quale l'utente seleziona l'informazione desiderata, piuttosto che scriverla lui direttamente: da una parte è un'agevolazione per l'utente, e dall'altra si uniformano le informazioni in input, nel senso che si evitano spiacevoli situazioni del tipo:

1. User 1: … provenienza: Milano
2. User 2: … provenienza: Lombardia

Queste situazioni sono assolutamente da evitare, ai fini delle statistiche.

Allo stesso modo è stata pensata anche la tabella Console*:* essa viene riempita con un set di console, e l'utente può scegliere in fase di input quali selezionare.

Queste tabelle vengono riempite, in fase di allestimento del DB, caricando il file project/sorgenti_db/7_input.sql.

Per semplicità e per ovvi motivi, data la non rilevanza ai fini del progetto, non sono state inserite tutte le zone geografiche del mondo, così come non sono stati inseriti tutti i tipi di mazzi di carte.

Nel caso in cui il social network cominciasse a rilevare una certa importanza, sarebbe compito dell'admin aggiornare queste tabelle (via console con psql, poiché non è stata realizzata un'interfaccia Web con questo compito).

Le Stats_* sono tutte le tabelle che contengono informazioni ai fini di monitoraggio della comunità, e quindi di competenza dell'utente analista e amministratore.

- **Stat_Users**: contiene, per ogni utente (identificato da uid chiave esterna su Users.uid), il numero di giochi desiderati e posseduti;
- **Stat_Users_Avg**: contiene il numero medio di giochi posseduti e desiderati dagli utenti. E' stato scelto di creare una tabella a parte, anche se per una sola tupla, per evitare ridondanza ripetendo inutilmente in Stat_Users, per ogni utente, il numero

medio di giochi desiderati e posseduti.

- **Stat_Game**: contiene, per ogni gioco (identificato da `gid` chiave esterna su `Game.gid`), il numero di utenti che desiderano e posseggono quel gioco;
- **Stat_Game_Sex**, **Stat_Game_Age**, **Stat_Game_Country** contengono per ogni gioco, il voto medio ricevuto per sesso, età, provenienza.

Tutte le tabelle `Stats_*` sono aggiornate dagli opportuni triggers che richiamano le corrispondenti funzioni PlpgSQL, ma analizzeremo nel dettaglio in seguito [2.11 per dettagli].

2.2 Vincoli & Domini

2.2.1 Vincoli row-level

I vincoli NOT NULL e UNIQUE sono già stati specificati nello schema relazionale [2.1.1]

- `Users.age` deve essere uno smallint maggiore di zero;
- `Users.type` deve essere uno smallint compreso tra {0,1,2};
- `Friend.accepted` è un boolean, settato di default a FALSE perché una richiesta di amicizia è sempre prima soggetta a conferma;
- `Friend.self_friendship` è un constraint che impedisce amicizie con se stessi
- `Game.sugg_age` e `Game.n_player` devono essere maggiori di zero;
- `Game.add_date` è un dato di tipo date, settato di default alla data del giorno corrente;
- `Game.type` deve essere uno smallint tra {0,1,2,3};
- `Game.name` e `Game.type`, assieme, sono in un vincolo di UNIQUE: questo perché un gioco è considerato unico all'interno della sua tipologia, mentre possono coesistere più giochi con lo stesso nome ma tipologie diverse (es.: Solitario videogame e Solitario delle carte);
- `HadTried.vote` deve essere uno smallint compreso tra 1 e 10, o nullo;
- `TableGame.sugg_num` deve essere uno smallint maggiore di zero, settato di default a 2;

- `Console.mark` e `Console.name` sono soggetti ad un vincolo di CHECK di consistenza, in cui ad ogni marca di console corrispondono i propri modelli.
    1. *mark = Sony → name = PlayStation, PlayStation2, PlayStation3, PSP*
    2. *mark = Microsoft → name = Xbox, XBox 360*
    3. *mark = Nintendo → name = Wii, GameBoy, DS*
    4. *mark = Sega → name = Master System, Mega Drive*
    5. *mark = Other → name = PC, Other*
- `Stat_Users.n_desired` e `Stat_Users.n_player` devono essere interi positivi;
- `Stat_Users_Avg.avg_desired` e `Stat_Users_Avg.avg_played` devono essere smallint positivi compresi tra 0 e 10;
- `Stat_Game.n_desiring` e `Stat_Game.n_playing` devono essere interi positivi;
- `Stat_Game_Age.avg_vote`, `Stat_Game_Sex.avg_vote` e `Stat_Game_Country.avg_vote` devono essere smallint compresi tra 0 e 10;
- `Stat_Game_Age.age` deve essere uno smallint maggiore di zero.

[3_plpgsql_schema.sql]

2.2.2 Domini

- `Users.sex` e `Stat_Game_Sex.sex` appartengono al dominio **Gender**, che può assumere valori *M* o *F*
- `Users.country`, `Expected_Zone.abbr`, `Stat_Game_Country.country` appartengono al dominio **Initials**, che può assumere le sigle delle regioni d'Italia. Ad ogni sigla di `Expected_Zone.abbr` corrisponde il nome della regione. Mentre `Users.country` e `Stat_Game_Country.country` sono chiave esterna su `Expected_Zone.abbr`, in modo tale da non usare direttamente i nomi e quindi lavorare con varchar troppo lunghi nelle operazioni di statistica.
    1. *VDA → Valle d'Aosta PIE → Piemonte*

2. *LOM → Lombardia* *LIG → Liguria*
3. *TAD → Trentino Alto Adige* *VEN → Veneto*
4. *FVG → Friuli Venezia Giulia*
5. *TOS → Toscana* *EMR → Emilia Romagna*
6. *MAR → Marche* *UMB → Umbria*
7. *LAZ → Lazio* *CAM → Campania*
8. *MOL → Molise* *BAS → Basilicata*
9. *ABR → Abruzzo* *PUG → Puglia*
10. *CAL → Calabria* *SIC → Sicilia*
11. *SAR → Sardegna* *XXX → Estero*

- `Expected_Card.type` e `TableGame.deck` appartengono al dominio **TypeDeck**, che contiene (dovrebbe) tutti i tipi di carte esistenti (*francesi, bergamasche, bolognesi, bresciane, genovesi, lombarde, siciliane, nuoresi, piacentine, piemontesi, romagnole, romane, sarde, toscane_fiorentine, trentine, trevisane, triestine, viterbesi, spagnole, tedesche_austriache, svizzere, tarocchi, collezione*);
- `Console.name` e `Reference.console` appartengono al dominio **Console_model**, che contiene (dovrebbe) tutti i modelli di console esistenti (*Play Station, Play Station 2, Play Station 3, PSP, XBox, XBox 360, Wii, GameBoy, DS, Master System, Mega Drive, PC, Other*);
- `Console.mark` appartiene al dominio **Console_mark** che contiene (dovrebbe) tutti i marchi di console esistenti (*Sony, Microsoft, Nintendo, Sega, Other*).

[2_domains.sql]

2.3 Viste

- Viste utilizzate nel **Profilo utente** (`includes/profile_user.inc.php`)
  1. `Profiles`: seleziona solo i campi da mostrare nel proprio profilo tra tutti i dati personali, referenziando anche il nome della regione di provenienza (`Users.country → Expected_Zone.abbr`);

2. `DesiredGame`: referenzia il nome e la tipologia dei giochi desiderati `(Desired.gid → Game.gid)`;
3. `HadTriedGame`: referenzia il nome e la tipologia dei giochi posseduti, con eventuale recensione e votazione `(HadTried.gid → Game.gid)`;
4. `Friendship`: seleziona solo le amicizie certe `(accepted` = TRUE`)`;
5. `PendingFriendship`: referenzia id, nome e cognome del richiedente un'amicizia fra le amicizie pendenti `(accepted` = FALSE`)`;
6. `FriendsList`: compone la lista di amici di tutti gli utenti, referenziando nome e cognome sia di colui che ha richiesto l'amicizia, sia di colui che l'ha confermata;

- Viste utilizzate nella **Ricerca utente** (`search_user.php`)
  1. `UserDesired`: referenzia nome, cognome di un utente e il nome dei giochi da lui desiderati `(Desired.uid → Users.gid)`;
  2. `UserHadTried`: referenzia nome, cognome di un utente e il nome dei giochi da lui posseduti `(HadTried.uid → Users.gid)`
- Viste utilizzate nella **Ricerca giochi** e **Tag** (`search_game.php, tag.php`)
  1. `GamesType`: seleziona solo id, nome e tipologia dei giochi;
- Viste utilizzate nel **Profilo gioco** (`profile_game.php`)
  1. `Videogm`: referenzia il nome di un videogioco e i suoi dati generici, più i dati specifici di ogni videogioco;
  2. `Tablegm`: referenzia il nome di un gioco da tavolo e i suoi dati generici, più i dati specifici di ogni gioco da tavolo;
  3. `Cardgm`: referenzia il nome di un gioco di carte e i suoi dati generici, più i dati specifici di ogni gioco di carte;
  4. `Othergm`: referenzia il nome di un gioco di altra categoria e i suoi dati generici (costruita per lo più per coerenza);
- Viste utilizzate nelle **Classifiche giochi suddivise per tipologia** (`includes/charts.inc.php`)

1. `ChartVideo`: seleziona dinamicamente nome e voto medio di ogni videogioco (approssimando la media a due cifre decimali), basandosi sui voti degli utenti che posseggono tale videogioco, ordinando la classifica decrescentemente ;
2. `ChartTable`: seleziona dinamicamente nome e voto medio di ogni gioco da tavolo (approssimando la media a due cifre decimali), basandosi sui voti degli utenti che posseggono tale gioco, ordinando la classifica decrescentemente ;
3. `ChartCard`: seleziona dinamicamente nome e voto medio di ogni gioco di carte (approssimando la media a due cifre decimali), basandosi sui voti degli utenti che posseggono tale gioco, ordinando la classifica decrescentemente ;
4. `ChartOther`: seleziona dinamicamente nome e voto medio di ogni altro gioco (approssimando la media a due cifre decimali), basandosi sui voti degli utenti che posseggono tale gioco, ordinando la classifica decrescentemente ;

- Viste utilizzate nelle **Autodelete** (`autodelete.php`)

  1. `Autodelete`: si appoggia alla tabella *Stats_Game*. Seleziona gli id dei giochi inseriti più di 5 giorni fa, segnalati da meno di 3 persone.

- Viste utilizzate nelle **Statistiche** (`includes/analyst/stat_*.php`)

  1. `StatDesGames, StatPossGames`: utilizzate nelle funzioni sql per le Statistiche Utenti (`6_functions/3_stat_user.sql`). Responsabili di contare il numero di giochi rispettivamente desiderati e posseduti/provati per ogni utente;
  2. `StatsUsersRefer`: referenzia nome e cognome di ogni utente in *Stats_Users* coninvolto nelle statistiche;
  3. `StatsGameRefer`: referenzia il nome di ogni gioco in *Stats_Game* coninvolto nelle statistiche;
  4. `StatsGameSexRefer,StatsGameAgeRefer,StatsGameCountryRefer`: referenzia il nome di ogni gioco in rispettivamente *Stats_Game_Sex*,

Stats_Game_Age, Stats_Game_Country.

[3_view.sql]

2.4 Ruoli e Privilegi

Coerentemente con le specifiche, il DB "**giogu**" ha 3 ruoli, e altrettanti privilegi, diversificati per ogni ruolo.

Il file `sorgenti_sito/includes/file_config_db.php` contiene le credenziali di log-in per accedere dall'interfaccia direttamente al DB. Alla prima connessione l'utente accede a giogu come utente registrato, colui che ha minori privilegi di tutti. Una volta connesso l'applicazione associa il corretto ruolo dell'utente, andando a recuperare l'attributo `type` in `Users` [4.1 Log-in per dettagli].

- admin: è Superuser, e ha tutti i privilegi su **giogu** with GRANT OPTION . Può usufruire sia delle normali operazioni dell'utente registrato che dell'utente analista. Pertanto, a livello applicazione, può accedere all'interfaccia di entrambi.

  Essendo poi amministratore, ha il potere di:

  - cancellare utenti, giochi, tag;
  - modificare recensioni e votazioni di utenti;
  - modificare dati generici e/o specifici di un gioco appartenente a una certa categoria;
  - liberare un gioco da un tag specifico o da tutti i suoi tags.

- analyst: è l'utente analista. Ha funzioni di monitoraggio della comunità. A livello applicazione ha un'interfaccia propria, che gli consente di:

  - accedere alle statistiche utenti [2.11.1 per dettagli];
  - accedere alle statistiche giochi [2.11.2 per dettagli];
  - accedere alle classifiche di gioco per tipologia [2.8 Classifiche giochi];
  - cercare un gioco per nome o per tag, e accedere al profilo del gioco, visualizzando i suoi dati generici e specifici di categoria, i suoi tags e i suoi dati statistici specifici;
  - modificare i propri dati (nome, cognome, password);

Ha i permessi di SELECT su ogni tabella che in qualche modo è coinvolta nelle statistiche, e ha completi privilegi nelle tabelle delle statistiche `Stat_*`

1. `SELECT, UPDATE(passwd,name,surname,age,sex,country) ON TABLE Users;`
2. `SELECT ON TABLE Game, Videogame, Tablegame, Cardgame, Other, Console, Reference, Tag, Association;`
3. `SELECT ON TABLE Desired, HadTried;`
4. `SELECT, INSERT, UPDATE, DELETE ON TABLE Stat_Users, Stat_Users_Avg, Stat_Game, Stat_Game_Age, Stat_Game_Sex, Stat_Game_Country;`
5. `SELECT ON TABLE Expected_Zone, Expected_Card;`
6. `USAGE ON SEQUENCE game_gid_seq, users_uid_seq;`
7. `SELECT ON VIEW Profiles, DesiredGame, HadTriedGame, UserDesired, UserHadTried, GamesType, Videogm, Tablegm, Cardgm, Othergm, ChartVideo, ChartTable, ChartCard, ChartOther, StatDesGames, StatPossGames, StatsUsersRefer, StatsGameRefer, StatsGameSexRefer, StatsGameAgeRefer, StatsGameCountryRefer` (tutte le viste, meno quelle coinvolte nella gestione dell'amicizia e quelle usate nella cancellazione automatica).

- registered: è l'utente iscritto a **giogu**. E' il normale utente per cui è pensata l'applicazione.

  A livello applicazione ha un'interfaccia propria, che gli consente di:

  - accedere al proprio profilo [4.2 per dettagli];
  - cercare utenti per e-mail, o per Nome e/o Cognome e/o giochi segnalati da tale utente (indifferentemente desiderati o posseduti) [4.3 per dettagli];
  - accedere a una lista dei 10 utenti più simili [2.10 Similarità per dettagli];
  - cercare un gioco per nome o per tag, e accedere al profilo del gioco [4.4 per dettagli];
  - inserire un gioco, accedendo ad un menù diversificato per ogni categoria [4.6

per dettagli];

- accedere alle classifiche di gioco per tipologia [2.8 Classifiche giochi per dettagli];
- inserire tags e taggare giochi [4.7 per dettagli];
- modificare i propri dati personali;

1. `SELECT, UPDATE(passwd,name,surname,age,sex,country) ON TABLE Users;`
2. `SELECT, INSERT, UPDATE, DELETE ON TABLE Friend;`
3. `SELECT, INSERT ON TABLE Game, Videogame, Tablegame, Cardgame, Other, Console, Reference, Tag, Association;`
4. `SELECT, INSERT, UPDATE, DELETE ON TABLE Desired, HadTried;`
5. `SELECT ON TABLE Expected_Zone, Expected_Card;`
6. `SELECT, INSERT, UPDATE, DELETE ON TABLE Stat_Users, Stat_Users_Avg, Stat_Game, Stat_Game_Age, Stat_Game_Sex, Stat_Game_Country;`
7. `USAGE ON SEQUENCE game_gid_seq, users_uid_seq;`
8. `SELECT ON TABLE Profiles, DesiredGame, HadTriedGame, UserDesired, UserHadTried, Friendship, PendingFriendship, FriendsList, GamesType, Videogm, Tablegm, Cardgm, Othergm, ChartVideo, ChartTable, ChartCard, ChartOther, StatDesGames, StatPossGames, StatsUsersRefer, StatsGameRefer, StatsGameSexRefer, StatsGameAgeRefer, StatsGameCountryRefer` (tutte le viste, meno quelle usate nella cancellazione automatica).

Da notare che entrambi gli utenti analista e iscritto possono sì modificare il proprio profilo, ma non completamente: id e ruolo rimangono inviolati.

Inoltre, entrambi non hanno in generale privilegi di DELETE (l'admin sì) e hanno solo privilegi di SELECT su `Expected_Zone, Expected_Card`, a prova del fatto che queste tabelle dovrebbero essere mantenute dall'admin.

[1_roles.sql & 5_grants.sql]

2.5 Giochi e consistenza con le tipologie

La tabella `Game` contiene tutti i giochi inseriti nel database, più gli attributi generici di ogni gioco. E' subito possibile ricavare la tipologia di gioco dal campo *type*.

Per mantenere consistenza tra la superclasse e le sottoclassi specifiche per ogni tipologia, sono stati necessari alcuni vincoli aggiuntivi:

- ogni id nelle sottotabelle di tipologia (`Videogame...`) identifica univocamente un gioco, e referenzia la chiave prima *gid* in `Game`. Pertanto è presente un vincolo di integrità referenziale tra la chiave esterna gid (che è anche primaria nelle sottotabelle) e la chiave primaria `Game.gid`. E quindi un gioco non può esistere in una sottotabella di tipologia senza che ci sia anche nella tabella principale `Game`;
- se un gioco viene cancellato da Game, sparisce anche dalla sottotabella di tipologia corrispondente, per il vincolo di integrità referenziale; ma il contrario non è assicurato. E' necessario pertanto un trigger, che attivi una funzione che impedisca la cancellazione di un gioco da una tipologia (`Videogame...`) se questo è ancora presente in `Game.`

  Di questo si occupano i triggers `consistence_specdel_*_trig` (4, uno per ogni tabella di tipologia), attivati BEFORE DELETE, che richiamano la funzione `specdel_game()` che impedisce il caso di cui sopra sollevando un'eccezione.
- occorre prevenire la possibilità che uno stesso gioco (e quindi uno stesso *gid*) coesista in più tabelle di tipologia (es.: *gid*=1 sia in `Videogame` che `Cardgame`). Questo check è svolto dai triggers `consistence_*_trig` (4, uno per ogni tabella di tipologia), attivati BEFORE INSERT, UPDATE, che richiamano la funzione `check_game_type()`, che verifica che un gioco sia inserito nella corretta tabella di tipologia, controllando l'attributo *type* di `Game`.

  In particolare, la funzione verificherà se il tipo del gioco corrisponde alla tipologia corretta: se sto inserendo (o modificando) un videogame con *gid*=X in `Videogame`, allora la funzione verificherà prima se l'attributo *type* di X sia 0 (perchè 0 corrisponde a `Videogame`). In caso contrario solleverà un'eccezione.

- per mantenere coerenza con il trigger precedente, il trigger `consistence_game_type_noupd_trig` attivato BEFORE UPDATE sull'attributo *type* di `Game`, impedisce la modifica di una tipologia di un certo gioco, richiamando la funzione `game_type_noupd()` che solleva un'eccezione.

[6_functions/7_consistence_trigger.sql]

2.6 Liste giochi desiderati – posseduti/provati

I giochi desiderati si trovano in `Desired`, mentre quelli posseduti o provati si trovano in `HadTried`.

Come già detto, è importante distinguere una lista dall'altra: un gioco è AUT desiderato AUT posseduto/provato (non è importante, invece, distinguere se posseduto o provato, piuttosto che se desiderato di possedere o desiderato di provare [1.1 giustificazioni]).

Se un gioco è desiderato, non può essere né recensito né votato; al contrario, un gioco posseduto o provato è soggetto a votazione e recensione.

Pertanto lo stesso gioco non può stare contemporaneamente nelle due liste. Questo check è implementato dai triggers `consistence_des_trig`, `consistence_poss_trig`, BEFORE INSERT sulle corrispondenti tabelle, che richiamano la funzione `desired_or_hadtried()`, la quale controlla che se sto inserendo un gioco come desiderato, allora non deve essere già posseduto e viceversa.

[6_functions/7_consistence_trigger.sql]

2.7 Amicizia

Friend memorizza le amicizie. Nella logica di'implementazione, quando un'amicizia viene richiesta, viene settata una tupla con id richiedente e id richiesto, e un flag *accepted* a FALSE: se l'amicizia viene accettata, *accepted* è settato a TRUE, altrimenti la tupla viene direttamente cancellata (per logiche di spazio).

Pertanto le tuple con il flag a FALSE sono le richieste d'amicizia pendenti, ancora in fase di approvazione.

Come già detto precedentemente, l'amicizia è un'associazione simmetrica: se A è amico di B, B è amico di A, ma per minimizzazione di spazio, viene mantenuta solo una direzione della relazione. Il trigger `consistence_friendship_trig` BEFORE INSERT richiama la

funzione `check_friendship()`, la quale verifica che:

1. il flag *accepted* sia inizialmente a FALSE (perchè un'amicizia deve essere confermata prima di essere settata);
2. garantisce l'unidirezionalità: se B ha richiesto l'amicizia ad A, verifica prima che A non abbia già richiesto l'amicizia a B;
3. uno dei due casi cotrari solleva un'eccezione.

N.B.: grazie al vincolo di riga `Friend.self_friendship` un utente iscritto non può essere amico di se stesso [2.2.1 per dettagli].

[6_functions/7_consistence_trigger.sql]

2.8 Classifiche di giochi per tipologia

Le classifiche sono visionabili da tutte le categorie di utenti: sia da registered che da analyst (oltre che ovviamente da admin). Il motivo di ciò sta nel fatto che:

- gli utenti iscritti possono conoscere nuovi giochi interessanti;

- gli utenti analisti monitorano quali giochi, tra le varie tipologie, vanno più in voga tra quelli votati: si tratta pur sempre di una forma di statistica.

Ogni classifica di giochi per tipologia, basata sulle votazioni degli utenti per i giochi posseduti, è realizzata da una corrispondente vista [2.3 per dettagli].

2.9 Funzioni PlpgSQL d'appoggio

Alcune funzioni generiche sono d'appoggio nel calcolo della similarità tra utenti, e nel calcolo delle statistiche:

- `count_game_des(uid_in integer)`: conta i giochi desiderati di uid_in;
- `count_game_poss(uid_in integer)`: conta i giochi posseduti/provati di uid_in;
- `count_user_des(gid_in integer)`: conta gli utenti che desiderano il gioco gid_in;
- `count_user_poss(gid_in integer)`: conta gli utenti che posseggono il gioco gid_in.

[6_functions/1_generalFunctions.sql]

2.10 Similarità

Per la similarità tra utenti, sono state realizzate due funzioni PlpgSQL:

- `similarity_calculate(uid_a integer, uid_b integer)`: questa funzione resituisce un valore intero da 1 a 100, rappresentante il tasso di similarità tra due utenti. L'algoritmo utilizzato è il seguente:

$$\mathbf{Similarity_{A,B} = 0.7 * X + 0.3 * Y}$$

dove X rappresenta il numero di giochi in comune tra A e B (indifferentemente se segnalati come desiderati o posseduti/provati);

$$\mathbf{X = 100 * (2 * \#game_{A,B}) / (\#game_A + \#game_B)}$$

Y misura la somiglianza dei voti sui giochi posseduti da entrambi.

Il grado di affinità Y è calcolato come la media degli scarti tra ogni coppia di voti $\mathbf{(v_A(g), v_B(g))}$ dove $v_A(g)$ e $v_B(g)$ sono i voti espressi da A e B sullo stesso gioco g, su una scala da 1 a 10 (HadTried.vote [2.1.1 per dettagli]). Quindi l'affinità su un singolo gioco è : $\mathbf{100 - 10 \cdot |v_A(g) - v_B(g)|}$.

$$\mathbf{Y = \frac{\sum_{g \in G=\{g1,\ldots,gn\}} (100 - 10 \cdot |v_A(g) - v_B(g)|)}{n}}$$

dove G è l'insieme dei giochi in comune tra A e B, di cardinalità n.

- `similarity(uid_in integer)`: calcola, per tutti gli utenti iscritti e amministratori (non analisti, in quanto non possono segnalare giochi) la similarità con l'utente `uid_in`.

[6_functions/2_similarity.sql]

2.11 Statistiche

2.11.1 Statistiche Utenti

Le statistiche utenti sono memorizzate nella tabella `Stat_User`: ogni tupla memorizza

- l'id di un utente;
- il numero di giochi da lui desiderati;
- il numero di giochi da lui posseduto/provato.

In aggiunta, la tabella `Stat_User_Avg` è composta da una sola riga, e memorizza

- un intero rappresentante il numero medio di giochi desiderati;
- un intero rappresentante il numero medio di giochi posseduti.

La scelta di implementare a parte il numero medio dei giochi des./poss. Piuttosto che memorizzarli nella tabella `Stat_User` è per risparmiare spazio ed evitare ridondanze.

Le funzioni PlpgSQL coinvolte nelle statistiche utenti sono:

- `stat_user_fill()`: riempie la tabella `Stat_User`, calcolando per tutti gli *uid* in `Users` il numero di giochi desiderati (`count_game_des(uid_in integer)`) e il numero di giochi posseduti (`count_game_pos(uid_in integer)`) [2.9 per dettagli]
- `stat_user_update()`: aggiorna la tabella `Stat_Users` su un utente specifico, cancellando la tupla corrispondente e ricalcolandone i nuovi valori;
- `avg_game_des()`: calcola il numero medio dei giochi desiderati, ottenendo la somma di tutti i giochi desiderati dalla vista `StatDesGames` [2.3 per dettagli] e dividendo per numero di utenti che possono desiderare giochi (escludendo quindi gli analyst);
- `avg_game_poss()`: calcola il numero medio dei giochi posseduti, ottenendo la somma di tutti i giochi desiderati dalla vista `StatPossGames` [2.3 per dettagli] e dividendo per numero di utenti che possono possedere giochi (escludendo quindi gli analyst);
- `stat_user_avg_fill()`: riempie la tabella `Stat_Users_Avg`, richiamando le due precendenti funzioni;
- `stat_user_avg_update()`: semplicemente svuota `Stat_Users_Avg` e richiama `stat_user_avg_fill()`.

`Stat_User` e `Stat_User_Avg` devono essere aggiornate nel caso di:

1. inserimento/cancellazione di un utente;
2. segnalazione/cancellazione di un gioco desiderato/provato.

Pertanto sono definiti i seguenti trigger:

1. `stat_user_users_t`: richiama `stat_user_update()` AFTER INSERT su `Users` (notare che la DELETE è superflua, in quanto questa scatenerà la DELETE su `Desired`, tab. dei giochi desiderati, o `HadTried`, tab. dei giochi provati, per i vincoli di integrità referenziale);
2. `stat_user_des_t`, `stat_user_poss_t`: richiamano `stat_user_update()` AFTER INSERT, DELETE su rispettivamente `Desired` e `Hadtried`.
3. `stat_user_avg_users_t`, `stat_user_avg_des_t`, `stat_user_avg_poss_t`: richiamano `stat_user_avg_update()` AFTER INSERT, DELETE su rispettivamente `Users`, `Desired` e `Hadtried` (qui DELETE su `Users` non è superflua, in quanto un utente che non ha desiderato né provato giochi, influisce comunque sul numero medio di giochi des/poss).

[6_functions/3_stat_user.sql & 6_functions/4_trigger_stat_user.sql]

2.11.2 Statistiche Giochi

Le statistiche giochi sono memorizzate nella tabella `Stat_Game`: ogni tupla memorizza

- l'id di un gioco;
- il numero di utenti che lo desiderano;
- il numero di utenti che lo possiedono o hanno provato.

In aggiunta, ci sono le tabelle `Stat_Game_Sex`, `Stat_Game_Age` e `Stat_Game_Country` che memorizzano

- l'id del gioco;
- la categoria;
- il voto medio ricevuto da quella categoria.

N.B.: per ogni gioco può esistere più di una tupla: una per ogni categoria risultante (es. `Stat_Game_Age` avrà, per ogni gioco, una tupla per ogni età che ha dato almeno un voto a quel gioco. Se nessun ventenne avrà dato un voto al gioco X, allora mancherà la tupla X → 20 anni → voto medio).

Le funzioni PlpgSQL coinvolte nelle statistiche giochi sono:

- `stat_game_fill()`: riempie la tabella `Stat_Game`, calcolando per tutti gli

gid in `Game` il numero di utenti desideranti (`count_users_des(uid_in integer)`) e il numero di utenti possedenti (`count_users_pos(uid_in integer)`) [2.9 per dettagli]

- `stat_game_update()`: aggiorna la tabella `Stat_Game` su un gioco specifico, cancellando la tupla corrispondente e ricalcolandone i nuovi valori;
- `stat_game_sex(gid_in integer)`, `stat_game_age(gid_in integer)`, `stat_game_country(gid_in integer)`: dato un gioco, calcola il voto medio per rispettivamente sesso, età e provenienza;
- `stat_game_sex_fill()`, `stat_game_age_fill()`, `stat_game_country_fill()`: riempiono le tabelle Stat_Game_Sex, Stat_Game_Age, Stat_Game_Country, richiamando le precedenti 3 funzioni per ogni gioco;
- `stat_game_sex_update()`, `stat_game_age_update()`, `stat_game_country_update()`: aggiorna selettivamente su un certo gioco rispettivamente le tabelle `Stat_Game_Sex`, `Stat_Game_Age` e `Stat_Game_Country`;
- `stat_game_sex_userupdate()`, `stat_game_age_userupdate()`, `stat_game_country_userupdate()`: a seguito di una modifica profilo da parte di un utente, cambiano anche le valutazioni medie sui giochi che possiede; quindi queste funzioni cancellano ogni gioco posseduto dalle tabelle `Stat_Game_Sex`, `Stat_Game_Age` e `Stat_Game_Country` e ricalcolano il valore medio per categorie di utenza.

Le tabelle sulle statistiche sui giochi devono essere aggiornate nel caso di:

1. inserimento/cancellazione di un gioco;
2. segnalazione/cancellazione di un gioco desiderato, e anche modifica di voto nel caso di posseduto/provato.
3. modifica dei dati `age`, `sex`, `country` su `Users`.

Pertanto sono definiti i seguenti trigger:

1. `stat_game_insgm_t`: richiama `stat_game_update()` AFTER INSERT su `Game` (notare che la DELETE è superflua, in quanto questa scatenerà la DELETE su `Desired`, tab. dei giochi desiderati, o `HadTried`, tab. dei giochi provati, per i vincoli di integrità referenziale su *gid*);

2. `stat_game_des_t`, `stat_game_poss_t`: richiamano `stat_game_update()` AFTER INSERT, DELETE su rispettivamente `Desired` e `Hadtried`.

3. `stat_game_sex_t`, `stat_game_age_t`, `stat_game_country_t`: richiamano rispettivamente `stat_game_sex_update()`, `stat_game_age_update()`, `stat_game_country_update()` AFTER INSERT, UPDATE di HadTried.vote o DELETE per aggiornare le medie;

4. `stat_game_sex_userupd_t`, `stat_game_age_userupd_t`, e `stat_game_country_userupd_t`: richiamano rispettivamente `stat_game_sex_userupdate()`, `stat_game_age_userupdate()`, `stat_game_country_userupdate()` AFTER UPDATE di `Users.sex`, `Users.age`, `Users.country` per aggiornare le medie.

[6_functions/5_stat_game.sql & 6_functions/6_trigger_stat_game.sql]

2.12 Autocancellazione

Ogni utente (registrato o amministratore) può inserire giochi di qualsiasi tipologia.

Entro però un periodo di tempo, pari a **5 giorni**, il gioco deve essere segnalato da almeno **3 persone**: infatti l'applicazione controlla i giochi più vecchi di 5 giorni dalla data corrente,e verifica quanti hanno referenziato il gioco (indifferentemente desiderato o posseduto/provato).

Ecco che qui entra in gioco l'attributo *add_date* della tabella `Game`: questo indica quando è stato inserito il gioco, quindi diventa molto semplice trovare i giochi inseriti da almeno 6 giorni.

Questa operazione si appoggia alla vista `Autodelete,` che esegue la ricerca di cui sopra, sfruttando la tabella `Stat_Game`: per ogni gioco, infatti, nella tabella è memorizzato un record con due attributi, e cioè quanti desiderano e quanti possiedono il gioco

Se la somma dei due attributi è inferiore a 3 (utenti) e il gioco è vecchio di più di 5 giorni (CURRENT_DATE-*add_date*>5) allora la vista seleziona il *gid* del gioco.

Naturalmente solo l'admin possiede i privilegi sulla vista di `Autodelete`.

L'automatica cancellazione è realizzata grazie ad uno script ***autodelete.sh***, che esegue uno script PHP `autodelete.php,` responsabile di eseguire una query sulla vista `Autodelete` e cancellare i relativi giochi.

autodelete.sh è così composto:

- **/home/.../php/bin/php** */home/.../apache2/htdocs/sorgenti_sito/autodelete.php*
- cioè: **path_interprete_php/php** *path_scriptphp*
- **path_interprete_php** è la directory dove si trova l'interprete del linguaggio php;
- *path_scriptphp* è la directory dove si trova `autodelete.php,` che deve essere eseguito periodicamente.

N.B.: essendo le DELETE in generale concesse solo all'admin, in `autodelete.php` la connessione è statica come utente amministratore, importanto il file `file_config_db.php` e usando i parametri di admin.

La periodica esecuzione dello script .sh che richiama lo script .php è realizzata:

- in una macchina UNIX sfruttando il processo di sistema **cron**, che è time-based. E' sufficiente collocare lo script .sh nella directory `/etc/cron.daily` affinchè lo script venga eseguito giornalmente dal sistema;
- in alternativa, ad opera dell'amministratore, che esegue quotidianamente lo script ***autodelete.sh***.

3 Tecnologie

3.1 Prodotti Software

Per la realizzazione di questo progetto sono stati utilizzati i seguenti prodotti software:

- PostgreSQL 9.0.1 (DBMS)
- phpPgAdmin 5.0.1
- Apache 2.2.17 HTTP Server
- PHP 5.2.15
- OS Linux Ubuntu 10.10
- Editor di testo gedit - Version 2.30.3
- Mozilla Firefox 3.6.15
- OpenOffice.org 3.2.1
- Dia versione 0.97.1 - editor di diagrammi, grafici, diagrammi di flusso, ecc.

3.2 Linguaggi di programmazione

Per rendere possibile questo progetto e consegnarlo all'utente finale sono stati utilizzati:

- SQL
- PlpgSQL
- PHP
- X HTML
- CSS

4 Applicazione Web e cenni d'implementazione

In generale, in tutto il codice PHP prodotto, si è cercato di:

- controllare che ad ogni pagina l'utente fosse prima loggato, e altrimenti redirigerlo al login;
- prevenire l'SQL-injection con la funzione pg_escape_string(input) con input proveniente dall'esterno (mail, password, nomi di giochi e dati, tags...).

4.0 Class db

La suddetta è una classe PHP scritta appositamente per gestire tutte le attività in cui è coinvolto il database: connessione, disconnessione e query.

- `connection()`: richiede il file di configurazione degli utenti del database `file_config_db.php`, che contiene i parametri di connessione a **giogu** per ciascun ruolo. Se la variabile ruolo in sessione è già settata ($_SESSION['type']), recupera le credenziali dell'utente verificando se è admin (type = 2), analyst (type = 1) o registered; altrimenti vuol dire che questa è la prima connessione, che serve per il Log-in, quindi connette l'utente come un utente generico;
- `disconnection()` : chiude la connessione col DB;
- `query($sql)`: esegue la query $sql e ritorna il risultato.

[Class_db.php]

4.1 Log-in & Log-out

Il log-in avviene con la funzione `login($email,$pass)`.

Questa verifica, con una query su `Users`, se c'è un utente con la mail `$email` e password `$pass`: la prima connessione avverrà come utente registered, non sapendo ancora l'applicazione di che tipo è l'utente che si sta connettendo!

Se viene correttamente ritornato un solo risultato, allora vengono settate $_SESSION['uid'] e $_SESSION['type'] con l'id e il ruolo dell'utente.

In fase di logout, nella corrispondente pagina, viene fatto l'unset delle variabili settate in $_SESSION, e successivamente viene invocata la funzione `session_unset()` che distrugge la sessione corrente.

[check_login.php & logout.php]

4.2 Profilo Utente

Il profilo utente permette di accedere a tutte le informazioni su quell'utente:

- i suoi dati pubblici (mail, nome, cognome, età, sesso, provenienza);
- i suoi giochi desiderati (e accedere al profilo di ognuno cliccandone il nome);
- i suoi giochi posseduti, ed eventuali recensioni/votazioni che ha assegnato (e accedere al profilo di ognuno cliccandone il nome);
- la lista dei suoi amici (nel caso di visualizzazione del proprio profilo, permette anche di revocarne l'amicizia) e accedere al profilo di ognuno cliccandone il nome.

Se due utenti iscritti sono amici, allora possono accedere al profilo dell'altro; l'admin può visualizzare il profilo di chiunque. Per gli utenti analisti non è quindi previsto un profilo, in quanto non fanno tutte le cose precedenti

Questa implementazione fa sì che uno può accedere alle recensioni e votazioni degli altri utenti solo se amici, mentre l'admin può visionare tutto nel caso debba moderare qualcosa.

I casi previsti nel profilo utente sono:

1. l'utente non è loggato → redirigilo al login;
2. $_GET['uid'] = $_SESSION['uid'] → l'utente registrato sta visualizzando il proprio profilo, quindi ha pieno accesso;

3. l'utente sta visualizzando un profilo diverso dal suo:
    1. $_SESSION['type'] = 2 → l'utente loggato è un admin che vuole vedere il profilo di qualcuno, e può accedervi; se poi non sono amici, può anche richiederne l'amicizia;
    2. $_SESSION['type'] = 0 → l'utente loggato è un normale iscritto che vuole vedere il profilo di qualcuno: controlla se sono amici
        1. se sono amici, l'utente loggato può accedere;
        2. altrimenti l'utente loggato propone una richiesta di amicizia (con gli opportuni controlli).

[includes/profile_user.inc.php]

4.3 Ricerca Utente

La ricerca di un utente può essere fatta solo da utenti iscritti o amministratori. Avviene:

- per e-mail, e quindi ha un unico risultato;
- per nome e/o cognome e/o giochi segnalati (desiderati o posseduti): questa ricerca può produrre più risultati.

[search_user.php]

4.4 Profilo Gioco

Il profilo gioco permette, a tutti gli utenti, di visualizzare:

1. i dati sia generici che specifici di del suddetto gioco;
2. tutti i tags con cui il gioco è stato taggato;

inoltre, c'è una versione “diversificata” del gioco in base al tipo di utente connesso:

3. se l'utente è analista ($_SESSION['type'] = 1), allora può accedere a tutti i statistici specifici per quel gioco (quanti lo desiderano e lo possiedono, il voto medio per sesso, età, regione);
4. se l'utente è iscritto ($_SESSION['type'] = 0), allora può:
    1. se non ha ancora segnalato il gioco né come desiderato né come posseduto/provato, segnalarlo come desiderato o posseduto (e in questo caso recensirlo e valutarlo);

2. se ha già desiderato il gioco, allora può cancellarlo dai desiderati o farlo passare dalla lista dei desiderati ai posseduti/provati (e in questo caso recensirlo e valutarlo);
3. se l'utente possiede il gioco, può modificare la propria recensione e/o la valutazione.

E' da notare il fatto che un utente può cancellare un gioco dalla lista dei giochi desiderati, ma non dalla lista dei posseduti/provati: questo perchè `HadTried` non distingue il possedere dall'aver provato.

Coerentemente con la realtà, è possibile non desiderare più un gioco che credevamo ci piacesse, poiché dopo averlo provato si è cambiata idea su quel gioco. Mentre un gioco, una volta provato, rimane comunque provato per sempre.

L'utente admin accede ad entrambe le versioni del gioco.

[profile_game.php]

4.5 Ricerca Gioco

La ricerca gioco può essere fatta da tutti i tipi di utenti. Avviene:

- per nome del gioco;
- per tag.

In entrambi i casi la ricerca può produrre più risultati, e cliccando sul nome del gioco apparso si può accedere al profilo.

[search_game.php]

4.6 Inserimento Guidato

Un gioco può essere inserito sia da utenti iscritti che amministratori, tramite la voce 'Insert game' della propria interfaccia. Questa visualizza un menù che permette di scegliere quale tipologia di gioco si vuole inserire, e di conseguenza seleziona i dati specifici da introdurre per il gioco della tipologia selezionata. Oltre a tutti i dati generici di ogni gioco (tab. `Game`):

- un videogioco ha anche una o più piattaforme di riferimento;
- un gioco da tavolo può avere anche numero di giocatori suggerito e durata;
- un gioco di carte una un solo mazzo di riferimento.

Se si introduce un gioco con un nome simile ad uno già presente nell'applicazione, allora prima dell'inserimento vengono suggeriti i giochi già presenti, e viene chiesta l'intenzione dell'utente se inserire o meno il gioco.

Per nome simile si intende case-insensitive e/o con un nome contenuto nel nome di un gioco già presente nel DB (`name ILIKE '%$name%'`). es.: se si vuole inserire il gioco 'MARIO' ed è già presente nell'applicazione 'Super Mario', allora prima di procedere con l'inserimento l'app. suggerisce all'utente la presenza di 'Super Mario' e chiede conferma se vuole inserirlo o meno.

Come già detto nelle giustificazioni dello schema ER [1.2 per dettagli], un vincolo presente nell'implementazione è che un gioco è considerato unico in base alla coppia (nome, tipologia). Il vincolo ha lo scopo di evitare il proliferare di giochi doppi.

Quindi se si vuole inserire un gioco ed è già presente nel DB un altro gioco con stessa tipologia e stesso nome, l'inserimento viene negato dal vincolo row-level `UNIQUE(name,type)` [2.2.1 per dettagli].

[includes/insert_game_form.inc.php &
includes/games_form/insert_*gm_form.inc.php & insert_*.php]

4.7 Tag

Un utente iscritto o amministratore può taggare uno o più giochi accedendo all'area 'Tags' dalla sua interfaccia.

In particolare l'utente può:

- visionare tutti i tags già inseriti nell'applicazione, i quali vengono automaticamente stampati per evitare l'inserimento di tags molto simili;
- inserire un nuovo tag nell'applicazione;
- associare un tag già presente a un gioco di tipologia specifica, introducendo nome e tipo;
- inserire un nuovo tag nell'applicazione e associarlo subito a un gioco di tipologia specifica;

I tag associati a un gioco sono visualizzabili nel profilo del gioco stesso [4.4 Profilo Gioco per dettagli].

[includes/tag_form.inc.php & tag.php]

4.8 Interfaccia esclusiva dell'amministratore

L'amministratore ha un'interfaccia 'Moderate' esclusiva, dalla quale può moderare la comunità, cancellando o modificando i dati inseriti dagli altri utenti.

Selezionandola, infatti, appare un menù dal quale può scegliere di:

- cancellare utenti, giochi, tag;
- modificare recensioni e votazioni di utenti;
- modificare dati generici e/o specifici di un gioco appartenente a una certa categoria;
- liberare un gioco da un tag specifico o da tutti i suoi tags.

[includes/admin/moder_form.inc.php & admin/*.php]

5 Possibili miglioramenti

1. Per scelte dovute a mancanza di tempo, le statistiche per categorie di utenza rispetto all'età non sono raggruppate in fasce d'età (es.:10-20, 20-30, 30-40 …), ma semplicemente in anni singoli;
2. sempre per mancanza di tempo, non è stato realizzato un form di registrazione per gli utenti: gli unici solo quelli presenti nel file `7_input.sql`;
3. come già detto all'inizio, nelle giustificazioni [1.1] `Expected_Zone` e `Expected_Card` dovrebbero essere più “piene” di quanto non siano; specificatamente, soprattutto per `Expected_Zone`, utenti da tutto il mondo dovrebbero essere in grado di iscriversi selezionando la loro provenienza, che dovrebbe già essere presente in tabella;
4. per i due punti precedenti, **giogu** al momento è strettamente dipendente dal file di input precompilato, che riempie le tabelle `Users`, `Expected_Zone` e `Expected_Card`;
5. **giogu** non è realizzato in più lingue purtroppo, ma solo in uno (scarno) inglese;
6. nella realizzazione del progetto non si è tenuto conto della grafica: è molto scarna ed essenziale (sono stati usati gli i-frame HTML in sostituzione di

Javascript e Ajax);

7. l'utente non può caricare alcun avatar;
8. non è stata realizzata nessuna forma di comunicazione (chat, PersonalMessage) tra utenti o tra utenti e admin: soprattutto la seconda, però, sarebbe caldamente consigliata in quanto gli utenti potrebbero aiutare l'admin nell'attività di moderatore (come avviene nei forum).