UNIVERSITÀ DEGLI STUDI DI MILANO
ANNO 2019

PROGETTO DI BASI DI DATI

Documentazione Tecnica

Marco Cutecchia

Matricola 909938

Indice

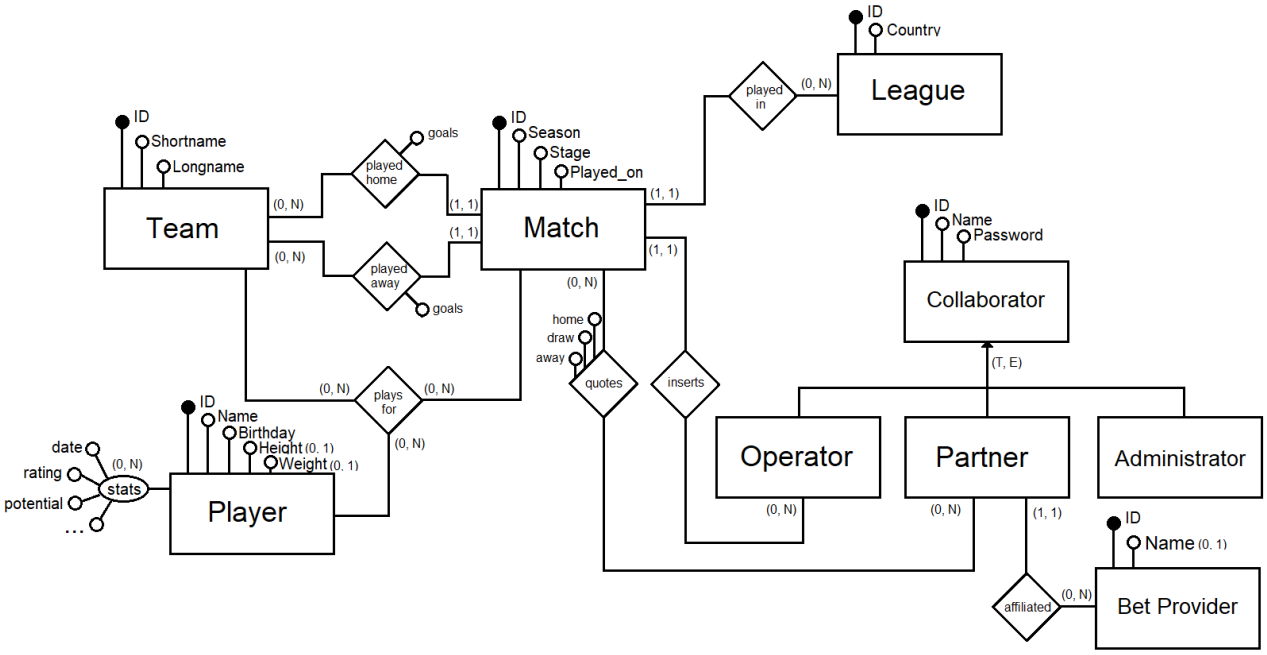
1 Tecnologie Usate

Nel realizzare questo progetto sono state utilizzate le seguenti tecnologie:

- PHP 7.3.0 per il lato server
- PostgreSQL 10.4 come database
- Apache 2.4 come web server
- Bulma 0.7.4 come framework CSS

La mia scelta riguardo a queste tecnologie è stata dettata semplicemente da un motivo di familiarità con esse. Riguardo a Bulma, l'ho preferito rispetto ad UIKit, che abbiamo usato nel corso, anche perché non richiede file Javascript extra, oltre ad un motivo estetico.

2 Schema ER



3 Schema Logico

Team(ID, Shortname, Longname)
Player(ID, Name, Birthday*, Height*, Weight*)
League(ID, Name, Country)
BetProvider(ID, Name*)
Collaborator(ID, Name, Password, Role, Affiliation*)
Stats(Player, AttributeDate, OverallRating, Potential*, PreferredFoot*, ...)
Match(ID, League, Season, Stage, PlayedOn, Hometeam, Awayteam, HometeamGoals, AwayteamGoals, CreatedBy)
Played(Player, Match, Team)
Quote(Match, BetProvider, HomeQuote, DrawQuote, AwayQuote, CreatedBy)

4 Scelte di progettazione del database

4.1 I giocatori

Nella tabella 'Player' ci sono alcuni attributi che possono essere NULL. Questa scelta viene semplicemente perché nei file dati da importare non per tutti i giocatori sono presenti queste informazioni. Più interessante è il come vengono salvate le statistiche riguardanti i giocatori. Ogni giocatore ha una grande quantità di statistiche da salvare(circa 40), però non tutte sono sempre disponibili: per esempio, un attaccante non ha informazioni riguardo la sua abilità di parare. Solamente una statistica è sempre presente, OverallRating, che per questo motivo non ammette valori NULL. La soluzione che ho adottato è quella più semplice: una grande tabella con tutte le statistiche possibili ma tutte che possono avere valori NULL. Un'altra soluzione che ho inizialmente considerato era quella di rendere ogni singola statistica una riga, ed avere una tabella del tipo

Stats(Player, AttributeDate, StatName, Value)

Questa soluzione ha il vantaggio che permette di non avere valori NULL riguardo le statistiche di un giocatore e di aggiungere nuovi tipi di statistiche senza modificare lo schema, ma ha un problema: non tutte le statistiche hanno dominio uguale. Per fare un esempio: il piede usato di un giocatore non è un valore numerico come il OverallRating. Si potrebbe codificare tutto in stringhe, ma si perderebbero tutte le possibilità di fare query facendo operazioni numeriche(a meno di riconvertire ogni volta in numero).

4.2 Importazione dati

La struttura dei file di dati CSV ha influenzato pesantemente la forma della base di dati. Molti vincoli presenti sono stati inseriti proprio per poter accettare i dati provenienti da essi. I file CSV spesso contenevano dati mancanti, fallati ed in un caso addirittura in contrasto con altri. Nell'importazione dei dati ho scelto semplicemente di ignorare questi dati fallati, mettendo NULL al loro posto quando possibile o addirittura scartando l'intera riga se questa avrebbe causato problemi di inconsistenza. L'applicazione web accetta solamente dei file CSV con lo stesso formato di quelli dati come esempio.

4.3 Le partite

La tabella 'Match' è piuttosto standard, però lo schema logico non mostra come viene prevenuto l'inserimento di match uguali: in SQL è presente anche un vincolo di UNIQUE tra League, Stage, PlayedOn, Hometeam e Awayteam. Avrei potuto rendere questi campi una chiave primaria, ma ho preferito tenere l'ID come chiave primaria perché oltre ad essere dato dai file CSV è anche una chiave più piccola da utilizzare sia nelle altre tabelle sia nel lato PHP.
Nella tabella 'Played', usata per tenere traccia di quali giocatori hanno giocato in quali partite e per quali team, si può notare quello che sembra essere un'errore della traduzione da ER a Logico: nonostante questa relazione sia una ternaria in ER qui la chiave primaria è formata solamente da Player e Match. Il motivo è per impedire che un giocatore possa aver partecipato in un match ad entrambe le squadre. Non sono riuscito a trovare una rappresentazione soddisfacente di questo tipo di relazione in ER. Nell'importazione dei dati questa scelta mi ha permesso di scovare una riga dove un giocatore giocava per entrambe le squadre in una partita.

4.4 La classifica del campionato

Per mantenere la classifica delle squadre per ogni campionato utilizzo una vista materializzata contenente per ogni tripla (Campionato, Stagione, Team) il numero di vittorie. Inoltre, è presente un trigger che ad ogni inserimento o update nella tabella delle partite aggiorna la vista materializzata. In un sito come questo ci si aspetta che la classifica venga visualizzata spesso ma aggiornata solamente le giornate delle partite.

4.5 Gli utenti e i dati inseriti

I dati degli utenti sono nella tabella 'Collaborator'. Il nome della tabella può suonare strano, ma è stato scelto solamente perché 'User' è una parola chiave in SQL. Il campo 'Password' contiene un hash della password scelta alla creazione dell'utente, in questo modo è possibile autenticare gli utenti senza salvare dati sensibili.

4.6 Il controllo dei permessi

Tutte le operazioni di inserimento, cancellazione o modifica vengono effettuate attraverso delle funzioni pl/pgSQL che si occupano anche del controllo dei permessi. Queste funzioni, oltre ai dati necessari per l'operazione, prendono anche l'id dell'utente che chiede di eseguire l'operazione e controllano se ne ha i permessi. Alla fine restituiscono un tipo di dato che ho chiamato QueryResult, che contiene la riga modificata e delle informazioni extra per la gestione degli errori, effettuata lato web.

```
db=# select * from insert_player(3, 39, 'Test Player', '1999-1-1', 180.0, 180.0);
 id | name | ... | success | error_code |                message
----+------+-----+---------+------------+---------------------------------------
    |      | ... | f       |         -1 | User is not allowed to insert players
(1 row)
```

Un esempio di QueryResult con errore di permesso negato. In questo caso l'utente con ID 3, che non ha ruolo Administrator ha provato ad inserire un giocatore.

4.7 Tutte le funzioni pl/pgSQL realizzate

Quella che segue è una lista di tutte le funzioni presenti nel database insieme ad una loro breve descrizione.

- `most_recent_stats(date)` : Restituisce una tabella contenente per tutti i giocatori solamente la riga della tabella `stats` più aggiornata rispetto alla data passata come argomento.
- `get_match_mvps(match, team)` : Restituisce le righe della tabella `player` che contengono il/i giocatore/i con la statistica OverallRating più alta tra chi ha partecipato al match passato come argomento nella squadra passata come argomento.
- `insert_*(userid, ...)` : Ci sono 8 versioni diverse di questa funzione, una per ogni tabella. Prendono l'ID dell'utente che vuole effettuare un inserimento, ne controllano i permessi e, se è tutto corretto, effettuano l'inserimento vero e proprio restituendo la riga inserita.
- `edit_*(userid, ...)` : Proprio come le funzioni `insert_*` ci sono 8 versioni di questa funzione. Prendono, oltre all'ID dell'utente che vuole effettuare una modifica, anche la chiave primaria della tabella ed i nuovi valori per la riga.
- `delete_*(user, ...)` : Ancora come prima, ma cancella una riga data la sua chiave primaria.

5 Scelte di progettazione nell'applicazione web

5.1 Autenticazione degli utenti

L'autenticazione degli utenti avviene attraverso una semplice sessione, dove vengono salvati le informazioni base dell'utente come l'id, il nome ed il ruolo. Nel database non viene salvata la password degli utenti ma solamente un salted hash creato dalle funzioni standard PHP `password_hash`. In questo modo, anche in caso di intrusioni nel server, le password degli utenti sono al sicuro.

5.2 Paginazione degli elementi

In più pagine del sito mi sono trovato a dover mostrare un numero elevato di elementi all'utente, tanto da rallentare pesantemente il browser. Per ovviare a questo problema ho implementato una visualizzazione a pagine degli elementi: solo 25 elementi vengono mostrati alla volta, per visualizzare gli altri bisogna scegliere la pagina apposta attraverso un indice in fondo alla pagina. Essendo la paginazione utilizzata in molti punti del sito ho creato un componente riutilizzabile che data una query e delle opzioni di formattazione permette di visualizzare qualunque risultato di query diviso in pagine.

5.3 Form multipagina

Nel pannello di controllo è capitato spesso di avere form dove è necessario scegliere degli oggetti da una lista che però deve essere divisa in pagine per il problema descritto sopra. Oppure la lista di questi oggetti varia in base a cosa si è scelto in un campo precedente del form. Una possibilità sarebbe potuta essere usare Javascript per fare richieste AJAX al server e caricare i contenuti dinamicamente. Questo

però avrebbe anche implicato l'implementare un API, oltre a dover includere anche un framework JS a meno di fare tutto a mano, con tanta fatica. Invece ho deciso di optare per una soluzione più semplice, leggermente meno efficiente dal lato server, ma molto più facile da implementare: ogni form che non sta in una pagina singola mantiene anche una variabile nel URL che indica a quale "stato" si trova la compilazione del form insieme a tutte le scelte fatte negli stati precedenti. All'ultimo passo viene creato un form contenente anche tutte le scelte precedenti come input nascosti.

```
state=select_player -> state=select_date&player=123 -> player=123&date=2008/08/30
```

C'è però una pagina soltanto dove ho optato per l'altra soluzione: la pagina dove si inseriscono i giocatori che hanno partecipato ad una partita. Se avessi optato per l'altra soluzione un utente avrebbe dovuto scegliere un giocatore dalla lista, scegliere la squadra e confermarlo per ogni singolo giocatore partecipante. Con 22 giocatori per ogni partita sarebbe una pessima esperienza utente. In questo caso ho fatto proprio quello che ho spiegato sopra: ho fatto in modo che la pagina restituisca i risultati di una ricerca per nome sulla tabella `Player` in formato JSON se alcuni parametri GET fossero impostati e con un po' di JS.

5.4 Error Handling lato Web

Volevo gestire gli errori lato web ma l'unica informazione sugli errori che potevo ricevere era una stringa. Avrei potuto semplicemente cercare delle parole chiave in essa, ma essendo la stringa localizzata questo metodo si sarebbe rotto appena l'applicazione fosse girata in un sistema con una lingua diversa. Inoltre, ci sono alcuni errori che volevo fossero mostrati sulla pagina ma altri che potevo gestire da solo (come tentativi di inserimento di chiavi duplicate). Per questo motivo ho aggiunto delle informazioni extra sugli errori alle funzioni SQL per le operazioni. In questo modo dal lato PHP leggendo quelle informazioni posso convertirle in eccezioni, più facili da gestire nel mio codice poi. Nel fare ciò ho costruito una piccola libreria che mi permette di fare le query più comuni su tutte le tabelle senza dovermi occupare di queste cose. Questo è un esempio, preso dal codice che gestisce il form per l'inserimento di un nuovo campionato, che mostra come vengono gestiti gli errori.

```
if( isset($_POST['name']) && isset($_POST['country']) ){
    try{
        League::prepare($db);
        League::insert($db, $_POST['name'], $_POST['country']);
        $success = true;
    }catch(PermissionDeniedException $e){
        $error = "You are not allowed to insert new leagues";
    }catch(DuplicateDataException $e){
        $error = "There is already a league with that id";
    }catch(DBException $e){
        $error = "An unknown error occurred[" . $e->getMessage() . "]";
    }
}
```

5.5 La pagina di setup

Al primo avvio dell'applicazione web, se non si ha caricato un dump del database, non si può continuare fino a quando non vengano create tutte le tabelle ed un utente amministratore. Visto che sono tanti file .sql da caricare ed in un ordine particolare ho creato una pagina di setup che in automatico carica i file .sql e li esegue, inoltre permette (solamente quando non ci sono utenti nel database) di creare il primo utente amministratore.