Elaborato per il corso di Basi Di Dati A.A 2022/2023

Ettore Farinelli
ettore.fainelli@studio.unibo.it
0001019995

26 agosto 2023

Indice

Capitolo 1

Analisi dei requisiti

Si pone l'obbiettivo di realizzare un database capace di gestire un organizzazione di arti marziali come può essere, per esempio, L'Ultimate Fighting Championship, (UFC). La base di dati dovrà quindi essere capace di registrare nuovi **lottatori** e in caso rimuoverli(squalifica, infortuneo, ritiro). Inoltre sarà possibile registrare **eventi**, dove i combattenti si scontreranno aggiornando (dopo gli **scontri**), gli **score** dei partecipanti e in caso le **classifiche**.

1.1 Intervista

A seguito di una prima intervista si sono ottenute le seguenti richieste:

Per ogni partecipante alla lega bisogna tenere traccia del nome, cognome, codice fiscale, data di nascita, peso e **arte marziale** in cui vuole lottare. Alla iscrizione di un nuovo lottatore esso verrà inserito all'ultimo posto nella classifica della propria categoria. Ci sarà la possibilità di registrare i **team** dei combattenti tenendo traccia di: nome, amministratore e origine. Saranno presenti diverse classifiche per ogni tipo di categoria dove i lottatori saranno ordinati in base ai loro **record** (V, P, S), dove le vittorie assegnano 3 punti, i pareggi 1 e le sconfitte 0.

Le categorie in cui verranno suddivisi i membri della lega sono: *Peso Piuma* (fino a 65kg), *Welterweight* (65kg - 77kg), *Peso Medio* (77kg - 84kg) e *Pesi Massimi* (da 84kg in poi). Inoltre non ci sarà una vera e propria divisione in arti marziali in quanto combattenti praticanti discipline diverse potranno scontrarsi tra loro, le arti marziali sono le seguanti: *MMA* (Mixed Martial Arts), *BJJ* (Brazial Jiu Jitsu) e infine *Muay Thai*. Per ogni partecipante dovrà essere inserita la disciplina di competenza possibilmente modificabile in futuro (sarà gestito in maniera analoga il peso). Inoltre,

partecipanti appartenenti a una determinata categoria potranno scontrarsi solo con altri membri della stessa. Gli eventi saranno constituiti da almeno 2 combattimenti ciascuno, bisognerà tener traccia del: nome dello stadio, luogo (indirizzo), costo noleggio stadio, spesa staff, data, orario inizio, orario fine, biglietti standard venduti, biglietti premium venduti, introiti netti, inoltre sarà necessario poter associare a un evento degli sponsor selezionabili tra quelli disponibili che hanno effettuato un contratto con la lega. Ogni lottatore partecipante riceverà un pagamento extra e verrà calcolata una quantità di guadagni tramite pubblicità, tutto in base al numero di biglietti venduti per l'evento, così da poter calcolare gli introiti dell'evento, il quale verrà aggiunto in una HISTORY dove saranno immagazzinati tutti gli eventi passati.

1.2 Definizioni

- **Lottatore**: partecipante alla lega.
- **Organizzatore**: amministratore che ha l'accesso al database e le autorizzazioni per gestirlo.
- **Evento**: un insieme di scontri avente un luogo e una data.
- **Scontro**: un incontro tra due lottatori della stessa categoria.
- **Classifica**: lista numerata in ordine dal partecipante migliore al peggiore in base ai record personali.
- **Arte marziale**: disciplina frequentante da un partecipante.
- **Team**: associazione a cui possono far parte 1 o piu combattenti, ha lo scopo di seguirli durante gli scontri e allenarli.
- **Record**: terna di vittorie, pareggi, sconfitte (V, P, S), ogni prtecipante ha la sua che definisce la sua posizione in classifica.

1.2.1 Operazioni Amministratore

- Registrare un nuovo lottatore.
- Rimuovere un lottatore.
- Registrare un nuovo team.
- Rimuovere un team.

- Aggiungere/Rimuovere uno sponsor.
- Registrare uno evento.
- Visualizzare le classifiche.
- Modificare i dati dei partecipanti.

Capitolo 2

Progettazione Concettuale

2.1 Amministratore

L'amministratore è colui che utilizza effettivamente a livello di applicazione il database, quindi sarà lui che organizzerà tutte le altri parti principali del database come i lottatori, gli eventi, i team e le sponsorizzazioni.

2.2 Lottatore

I lottatori sono il cuore dell'intero database, per questo parte tutto dall'entità "lottatore" a cui sarà associata un'entità "record" creata in modo tale da poter tener traccia dello score di ogni combattente, inoltre i lottatore potranno o meno far parte di un team precedentemente registrato. Infine, ogni lottatore potrà partecipare a uno scontro per ogni evento, in questo caso non sono riuscito a gestire (a livello di schema concettuale) il vincolo per il quale due lottatori di categorie diverse non possono combattere.

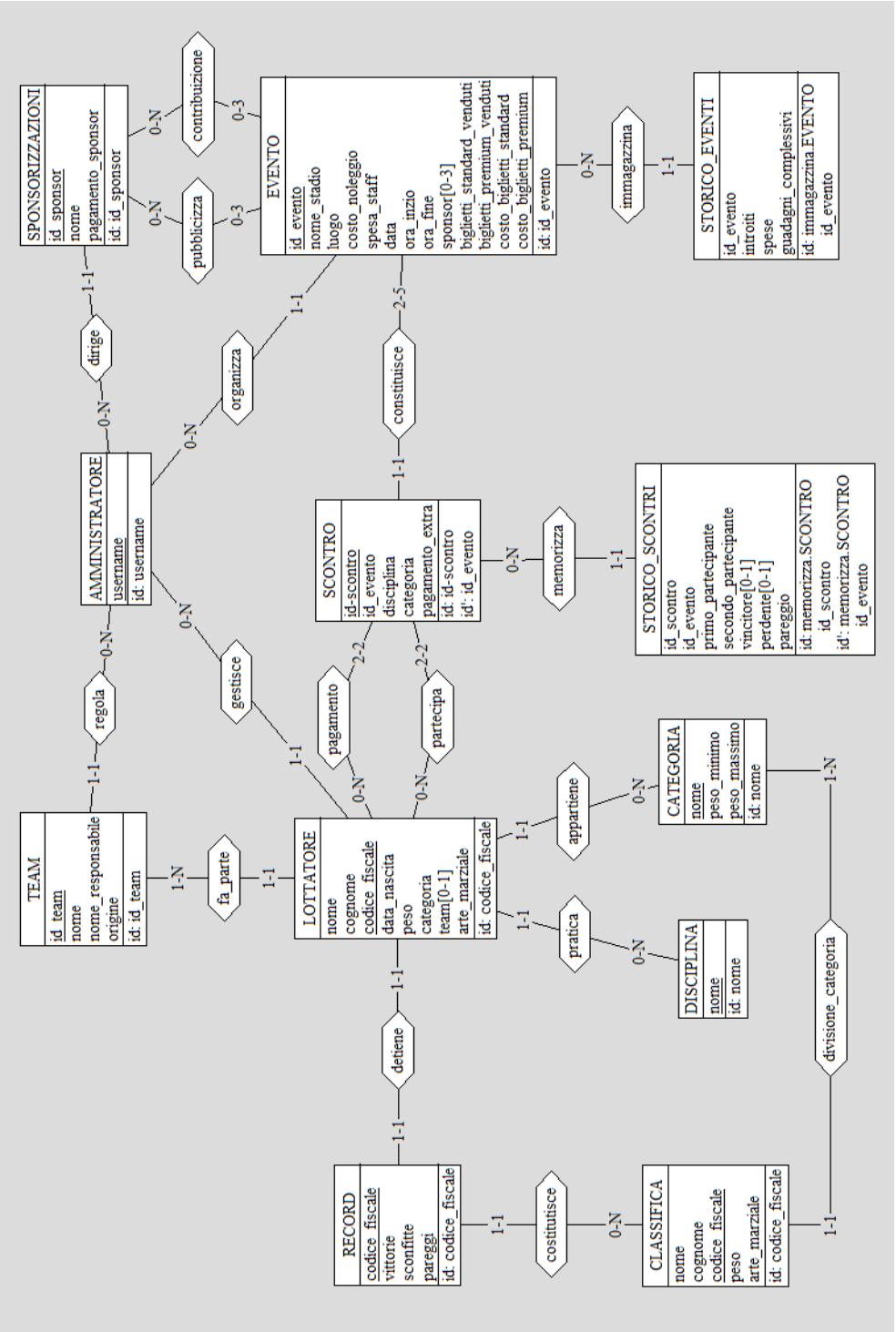
2.3 Evento

Gli eventi sono il principale elemento della lega dal quale proviene il profitto di quest'ultima. Quindi, è importante tenerne traccia con un entità "Storico_eventi" per poi realizzare a livello di frontend un grafico mostrante i guadagni generali, inoltre ogni evento potrà essere sponsorizzato da uno o più sponsor registrati nella lega.

2.4 Scontro

Gli scontri constituiscono gli eveti e sono formati da due lottatori della stessa categoria, è importante anche che i due partecipanti a uno scontro ricevano un pagamento in base alle volontà dell'amministratore.

2.5 Schema concettuale finale



Capitolo 3

Progettazione logica

3.1 Stima del volume dei dati

- AMMINISTRATORE: E, 1
- LOTTATORE: E, 250
- CATEGORIA: E, 4
- DISCIPLINA: E, 3
- RECORD: E, 250
- CLASSIFICA: E, 250
- SCONTRO: E, 2000
- STORICO_SCONTRI: E, 2000
- EVENTO: E, 700
- STORICO_EVENTI: E, 700
- SPONSORIZZAZIONI: E, 15
- TEAM: E, 150
- DIRIGE: A, 20
- GESTISCE: A, 350
- REGOLA: A, 200
- ORGANIZZA: A, 700

- DETIENE: A, 250
- CONSTITUISCE: A, 250
- MEMORIZZA: A, 2000
- DIVISIONE_CATEGORIA: A, 4
- APPARTIENE: A, 250
- PRATICA: A, 375
- FA_PARTE: A, 200
- PARTECIPA: A, 4000
- PAGAMENTO: A, 4000
- CONSTITUISCE: A, 2000
- PUBBLICIZZA: A, 3000
- CONTRIBUIZIONE: A, 3000
- IMMAGAZINA: A, 700

3.2 Descrizione delle operazioni principali e stima della loro frequenza

- Registrazione nuovo lottatore: 1/10 g
- Rimuovere un lottatore: 1/15 g
- Registrazione nuovo team: 1/12 g
- Rimuovere un team: 1/18 g
- Registrare un nuovo sponsor: 1/120 g
- Rimuovere uno sponsor: 1/210 g
- Registrare un evento: 1/20 g
- Visualizzare la classifica: 3/ g
- Modificare dati di un partecipante: 1/5 g

3.3 Schemi di navigazione e tabelle degli accessi

3.3.1 Registrazione nuovo lottatore

Tavola degli accessi

Concetto	Costrutto	Accessi	Tipo
AMMINISTRATORE	E	1	R
LOTTATORE	E	1	W
GESTISCE	A	1	R
TOTALE		4	

3.3.2 Rimuovere un lottatore

Tavola degli accessi

Concetto	Costrutto	Accessi	Tipo
AMMINISTRATORE	E	1	R
LOTTATORE	E	1	W
GESTISCE	A	1	R
TOTALE		4	

3.3.3 Registrazione/Rimozione nuovo team

Tavola degli accessi

Concetto	Costrutto	Accessi	Tipo
AMMINISTRATORE	E	1	R
TEAM	E	1	W
REGOLA	A	1	R
TOTALE		4	

3.3.4 Registrare/Rimuovere un nuovo sponsor

Tavola degli accessi

Concetto	Costrutto	Accessi	Tipo
AMMINISTRATORE	E	1	R
SPONSORIZZAZIONI	E	1	W
DIRIGE	A	1	R
TOTALE		4	

3.3.5 Registrare un evento

Tavola degli accessi

Concetto	Costrutto	Accessi	Tipo
AMMINISTRATORE	E	1	R
EVENTO	E	1	W
SCONTRO	E	3	W
SPONSORIZZAZIONI	E	3	R
STORICO_SCONTRI	E	3	W
STORICO_EVENTI	E	1	W
LOTTATORE	E	6	R
CATEGORIA	E	6	R
PUBBLICIZZA	A	3	R
PARTECIPA	A	6	R
CONSTITUISCE	A	3	R
APPARTIENE	A	6	R
CONTRIBUIZIONE	A	3	R
ORGANIZZA	A	1	R
IMMAGAZINA	A	1	R
STORICO_SCONTRI	A	3	R
PAGAMENTO	A	6	R
TOTALE		64	

3.3.6 Visualizzare la classifica

Tavola degli accessi

Concetto	Costrutto	Accessi	Tipo
CLASSIFICA	E	1	R
CATEGORIA	E	1	R
DIVISIONE_CATEGORIA	A	1	R
TOTALE		3	

3.3.7 Modificare dati di un partecipante

Tavola degli accessi

Concetto	Costrutto	Accessi	Tipo
AMMINISTRATORE	E	1	R
LOTTATORE	E	1	R
LOTTATORE	E	1	W
RECORD	E	1	R
RECORD	E	1	W
CATEGORIA	E	1	R
CATEGORIA	E	1	W
DISCIPLINA	E	1	R
DISCIPLINA	E	1	W
TEAM	E	1	R
TEAM	E	1	W
FA_PARTE	A	1	R
PRATICA	A	1	R
GESTISCE	A	1	R
APPARTIENE	A	1	R
DETIENE	A	1	R
TOTALE		21	

3.4 Analisi delle ridondanze

3.4.1 Campo Categoria nell'entità Lottatore

Il campo categoria all'interno di lottatore mi permette di non controllare la categoria di quest'ultimo attraverso il peso nella fase di creazione di uno scontro.

Caso senza ridondanza

Tavola degli accessi

Concetto	Costrutto	Accessi	Tipo
AMMINISTRATORE	E	1	R
EVENTO	E	1	W
SCONTRO	E	3	W
SPONSORIZZAZIONI	E	3	R
STORICO_SCONTRI	E	3	W
STORICO_EVENTI	E	1	W
LOTTATORE	E	6	R
CATEGORIA	E	6	R
PUBBLICIZZA	A	3	R
PARTECIPA	A	6	R
CONSTITUISCE	A	3	R
APPARTIENE	A	6	R
CONTRIBUIZIONE	A	3	R
ORGANIZZA	A	1	R
IMMAGAZINA	A	1	R
STORICO_SCONTRI	A	3	R
PAGAMENTO	A	6	R
TOTALE		64	

$$\frac{1}{20g} \times 64R = 3,2\frac{R}{g} \tag{3.1}$$

Caso con ridondanza

Tavola degli accessi

Concetto	Costrutto	Accessi	Tipo
AMMINISTRATORE	E	1	R
EVENTO	E	1	W
SCONTRO	E	3	W
SPONSORIZZAZIONI	E	3	R
STORICO_SCONTRI	E	3	W
STORICO_EVENTI	E	1	W
LOTTATORE	E	6	R
PUBBLICIZZA	A	3	R
PARTECIPA	A	6	R
CONSTITUISCE	A	3	R
CONTRIBUIZIONE	A	3	R
ORGANIZZA	A	1	R
IMMAGAZINA	A	1	R
STORICO_SCONTRI	A	3	R
PAGAMENTO	A	6	R
TOTALE		52	

$$\frac{1}{20g} \times 52R = 2,6\frac{R}{g} \tag{3.2}$$

Concludo quindi che devo preservare la ridondanza.

3.5 Traduzione di entità e associazioni in relazioni

Lottatore(codice_fiscale, nome, cognome, data_nascita, peso, team*, arteMarziale, id_team*, categoria, usernameAmm)
fk: arteMarziale references Disciplina
fk: usernameAmm references Amministratore
fk: id_team references Team
fk: categoria references Categoria

Team(id_team, nome, nome_responsabile, origine, usernameAmm)
fk: usernameAmm references Amministratore

Amministratore(username)

Sponsorizzazioni(id_sponsor, nome, pagamento_sponsor, usernameAmm)
fk: usernameAmm references Amministratore

Evento(id_evento, id_sponsorPubbl, id_sponsorContr, nome_stadio, luogo, costo_noleggio, spesa_staff, data, ora_inizio, ora_fine, sponsor1*, sponsor2*, sponsor3*, biglietti_standard_venduti, biglietti_premium_venduti, costo_biglietti_standard, costo_biglietti_premium, usernameAmm)
fk: usernameAmm references Amministratore
fk: (id_sponsorPubbl, id_sponsorContr) references Sponsorizzazioni

Storico_Eventi(id_storicoEventi, id_evento, introiti, spese, guadagni_conplessivi)
fk: id_evento references Evento

Scontro(id_scontro, id_evento, disciplina, categoria, pagamento_extra, codice_fiscalePart, codice_fiscalePaga)
fk: id_evento references Evento
fk: (codice_fiscalePart, codice_fiscalePaga) references Lottatore

Storico_Scontri(id_storicoScontri, id_scontro, id_evento, primo_partecipante, secondo_partecipante, vincitore*, perdente*, pareggio)
fk: (id_evento, id_scontro) references Scontro

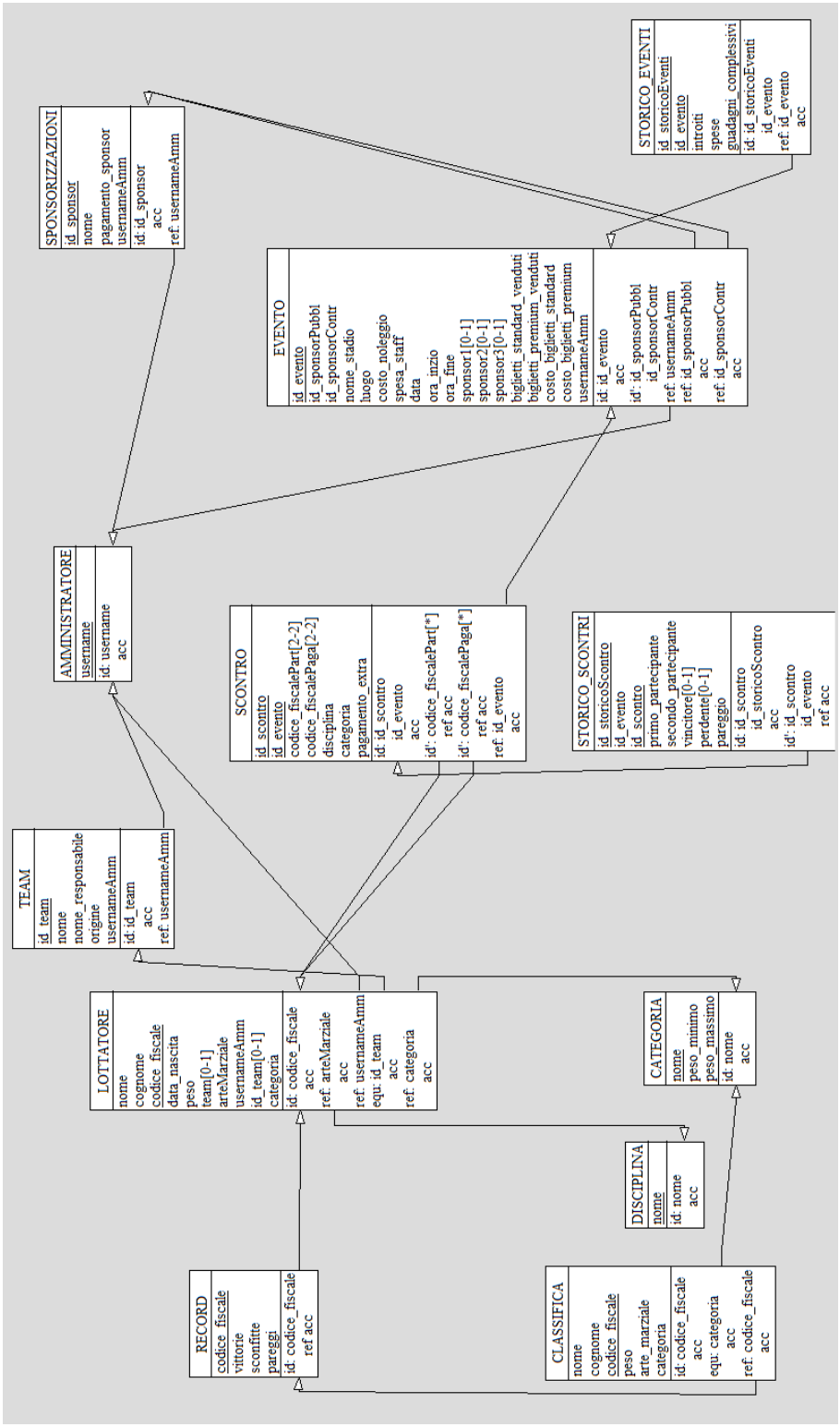
Categoria(nome, peso_minimo, peso_massimo)

Disciplina(nome)

Classifica(nome, cognome, codice_fiscale, peso, arte_marziale, categoria)
fk: categoria references Categoria
fk: codice_fiscale references Record

Record(codice_fiscale, vittorie, sconfitte, pareggi)
fk: codice_fiscale references Lottatore

3.6 Schema relazionale finale



3.7 Traduzione operazioni in query SQL

3.7.1 Creazione tabelle

```
CREATE TABLE AMMINISTRATORE (
    username varchar(30) NOT NULL PRIMARY KEY
);

CREATE TABLE LOTTATORE (
    codiceFiscale varchar(50) NOT NULL,
    nome varchar(20) NOT NULL,
    cognome varchar(30) NOT NULL,
    dataNascita date NOT NULL,
    team varchar(40),
    peso float NOT NULL,
    categoria varchar(20) NOT NULL,
    arteMarziale ENUM('BJJ', 'MMA', 'MuayThai') NOT NULL,
    PRIMARY KEY (codiceFiscale)
);

CREATE TABLE TEAM (
    idTeam integer NOT NULL,
    nome varchar(50) NOT NULL,
    nome_responsabile varchar(20),
    origine varchar(40),
    PRIMARY KEY (idTeam)
);

CREATE TABLE RECORD (
    codiceFiscale varchar(50) NOT NULL,
    vittorie integer,
    sconfitte integer,
    pareggi integer,
    PRIMARY KEY (codiceFiscale)
);

CREATE TABLE CLASSIFICA_PIUMA (
    codiceFiscale varchar(50) NOT NULL,
    nome varchar(20) NOT NULL,
    cognome varchar(30) NOT NULL,
    peso float NOT NULL,
```

```
    arteMarziale ENUM('BJJ', 'MMA', 'MuayThai') NOT NULL,
    PRIMARY KEY (codiceFiscale)
);

CREATE TABLE CLASSIFICA_WELTERWEIGHT (
    codiceFiscale varchar(50) NOT NULL,
    nome varchar(20) NOT NULL,
    cognome varchar(30) NOT NULL,
    peso float NOT NULL,
    arteMarziale ENUM('BJJ', 'MMA', 'MuayThai') NOT NULL,
    PRIMARY KEY (codiceFiscale)
);

CREATE TABLE CLASSIFICA_MEDIO (
    codiceFiscale varchar(50) NOT NULL,
    nome varchar(20) NOT NULL,
    cognome varchar(30) NOT NULL,
    peso float NOT NULL,
    arteMarziale ENUM('BJJ', 'MMA', 'MuayThai') NOT NULL,
    PRIMARY KEY (codiceFiscale)
);

CREATE TABLE CLASSIFICA_MASSIMI (
    codiceFiscale varchar(50) NOT NULL,
    nome varchar(20) NOT NULL,
    cognome varchar(30) NOT NULL,
    peso float NOT NULL,
    arteMarziale ENUM('BJJ', 'MMA', 'MuayThai') NOT NULL,
    PRIMARY KEY (codiceFiscale)
);

CREATE TABLE DISCIPLINA (
    nome ENUM('BJJ','MMA','MuayThai') PRIMARY KEY
);

CREATE TABLE CATEGORIA (
    nome ENUM('PesoPiuma','Welterweight',
            'PesoMedio','PesiMassimi') PRIMARY KEY,
    pesoMinimo integer,
    pesiMassimi integer
);
```

```
CREATE TABLE SCONTRO (
    idEvento integer NOT NULL,
    idScontro integer NOT NULL,
    disciplina ENUM('BJJ','MMA','MuayThai'),
    categoria ENUM('PesoPiuma','Welterweight',
                   'PesoMedio','PesiMassimi'),
    pagamentoExtra float,
    PRIMARY KEY (idEvento, idScontro)
);

CREATE TABLE STORICO_SCONTRI (
    idEvento integer NOT NULL,
    idScontro integer NOT NULL,
    primoPartecipante varchar(50) NOT NULL,
    secondoPartecipante varchar(50) NOT NULL,
    vincitore varchar(50),
    perdente varchar(50),
    pareggio bool,
    PRIMARY KEY (idScontro, idEvento),
    CONSTRAINT PAREGGIO CHECK
        ((pareggio IS TRUE AND vincitore IS NULL
        AND perdente IS NULL) OR
        (pareggio IS FALSE AND vincitore IS NOT NULL
        AND perdente IS NOT NULL))
);

CREATE TABLE EVENTO (
    idEvento integer NOT NULL,
    nomeStadio varchar(40) NOT NULL,
    luogo varchar (50) NOT NULL,
    costoNoleggio float,
    spesaStaff float,
    dataEvento date NOT NULL,
    oraInizio varchar(10) NOT NULL,
    oraFine varchar(10) NOT NULL,
    bigliettiStandardVenduti integer,
    bigliettiPremiumVenduti integer,
    costoBigliettiPremium integer,
    costoBigliettiStandard integer,
    sponsor JSON,
```

```
    PRIMARY KEY (idEvento),
    CONSTRAINT ORARIO CHECK (oraInizio <= oraFine)
);

CREATE TABLE SPONSORIZZAZIONI (
    idSponsor integer NOT NULL,
    nome varchar(40) NOT NULL,
    pagamentoSponsor float,
    PRIMARY KEY (idSponsor)
);

CREATE TABLE STORICO_EVENTI (
    idEvento integer NOT NULL,
    introiti float,
    spese float,
    guadagniComplessivi float,
    PRIMARY KEY (idEvento)
);
```

3.7.2 Operazioni amministratore

Aggiungere lottatore

```
INSERT INTO LOTTATORE (nome, cognome, codiceFiscale,
    dataNascita, team, peso, categoria, arteMarziale)
VALUES (?, ?, ?, ?, ?, ?, ?, ?);

-- Aggiungere Record --

INSERT INTO RECORD (idRecord, vittorie, sconfitte, pareggi)
VALUES (?, 0, 0, 0);

-- Aggiungere classifica --

INSERT INTO CLASSIFICA (nome, cognome, codiceFiscale, peso,
    arteMarziale)
VALUES (?, ?, ?, ?, ?);
```

P.S: In questo caso inserisco il lottatore nella tabella 'classifica' che non esiste per comodità, nell'applicazione invece sarà presente un controllo che farà in

modo che il partecipante venga inserito nella classifica giusta in base al suo peso.

Rimuovere lottatore

```
DELETE FROM LOTTATORE
WHERE codiceFiscale = ?

-- Elimina il record corrispondente --

DELETE FROM RECORD
WHERE codiceFiscale = ?

-- Elimina l'elemento in classifica

DELETE FROM CLASSIFICA
WHERE codiceFiscale = ?
```

P.S: vale la stessa cosa che ho detto prima per l'aggiunta classifica.

Aggiungere team

```
INSERT INTO TEAM (idTeam, nome, nome_responsabile, origine)
VALUES(?, ?, ?, ?);
```

Rimuovere team

```
DELETE FROM TEAM
WHERE idTeam = ?
```

Aggiungere sponsor

```
INSERT INTO SPONSORIZZAZIONI(idSponsor, nome, pagamentoSponsor)
VALUES (?, ?, ?);
```

Rimuovere sponsor

```
DELETE FROM SPONSORIZZAZIONI
WHERE idSponsor = ?
```

Registrare evento

```
INSERT INTO EVENTO (idEVento, nomeStadio, luogo, costoNoleggio,
    spesaStaff, dataEvento, oraInizio, oraFine,
    bigliettiStandardVenduti, bigliettiPremiumVenduti,
    costoBigliettiStandard, costoBigliettiPremium, sponsor)
VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?);

-- Inserire evento in storicoEventi --

INSERT INTO STORICO_EVENTI (idEvento, introiti, spese,
    guadagniComplessivi)
VALUES (?, ?, ?, ?);

-- Aggiungere Scontro (2-5 volte, in base al tipo di evento) --

INSERT INTO SCONTRO (idEvento, idScontro, disciplina,
    categoria, pagamentoExtra)
VALUES (?, ?, ?, ?, ?);

-- Inserire Scontro nello storicoScontri --

INSERT INTO STORICO_SCONTRI (idEvento, idScontro,
    primoPartecipante, secondoPartecipante, vincitore,
    perdente, pareggio)
VALUES (?, ?, ?, ?, ?, ?, ?);
```

Modificare lottatore

```
UPDATE LOTTATORE
SET nome = ?, cognome = ?,
    dataNascita = ?, team = ?,
    peso = ?, categoria = ?,
    arteMarziale = ?
WHERE codiceFiscale = ?

UPDATE RECORD
SET vittore = ?,
    sconfitte = ?,
    pareggi = ?
```

```
WHERE codiceFiscale = ?
```

Visualizzare classifiche

```
SELECT c.codicFiscale
FROM CLASSIFICA c
JOIN RECORD r ON c.codiceFiscale = r.codiceFiscale
ORDER BY (r.vittoria * 3 + r.pareggio) DESC;
```

Capitolo 4

Progettazione dell'applicazione

4.1 Descrizione dell'architettura dell'applicazione realizzata

L'applicazione che permette di gestire il database è stata creata utilizzando Typescript in combinazione con il framework *React*, mentre per quanto riguarda il database, esso risiede in locale ed è stato sviluppato usando *MySQL*. L'applicazione è un'applicazione web che si appoggia a un server locale. Essa consente quindi la possibilità di connettere frontend e backend utilizzando il protocollo **HTTP**, e quindi attraverso delle **API REST**. Grazie a un server locale, il backend avrà la possibilità di ricevere richieste dal frontend, portando con sé parametri e restituendo valori, il tutto evitando una connessione diretta tramite codice. Questo design permette all'applicazione di avere una scalabilità maggiore, eliminando eventuali dipendenze tra frontend e backend. Inoltre, per la gestione del database da parte del backend, si utilizza la libreria *TypeORM* di Typescript. Libreria che permette di eseguire operazioni sui dati mediante codice.