Universitá degli Studi di Brescia

CORSO DI LAUREA IN INGEGNERIA DELL'AUTOMAZIONE INDUSTRIALE

SPORT DATA

Progetto finale del corso di basi di dati

Studente:
Riccardo Valtorta
Matricola:
731579

Professoressa:
Anisa Rula

AA (2022/2023)

Indice

1 Obbiettivo del progetto

L'obbiettivo del progetto è quello di sviluppare un'applicazione che possa agevolare l'organizzazione e la gestione del capitale umano e materiale per le piccole associazioni sportive. Il problema della gestione delle risorse sorge dal fatto che questo tipo di organizzazioni tipicamente si trova a gestire gruppi relativamente numerosi di persone (dalle decine al centinaio di individui) e le complicate relazioni tra esse, disponendo di mezzi limitati a causa dell'elevato costo di strumenti più sofisticati. Bisogna inoltre tenere conto che chi si trova ad amministrare tali associazioni, lo fa come forma di volontariato, pertanto non dispone di conoscenze tecniche specifiche e non è propenso ad investire in questo genere di tecnologie, spesso sottovalutate e ritenute superflue. La sfida è quindi creare un tool che semplifichi la gestione di associazioni sportive in modo intuitivo ed economico.

1.1 Strumenti utilizzati

Per la realizzazione di un'applicazione web che rispondesse alle esigenze del progetto sono stati impiegati:

- DBMS: SQLite
- Linguaggi di programmazione: Python con framework Flask, HTML con Bootstrap
- DB Browser for SQLite come interfaccia grafica

2 Requisiti della base di dati

2.1 Raccolta dei requisiti

I requisiti della base di dati sono stati raccolti sulla base dell'esperienza personale, avendo osservato il funzionamento di una simile organizzazione. Per dettagliare meglio alcuni aspetti, tuttavia, sono state effettuate interviste che hanno permesso di definire in modo più preciso le informazioni rilevanti da conoscere per un atleta.

2.2 Requisiti

Di fondamentale importanza per un'associazione sportiva sono, ovviamente, gli sportivi stessi. Possiamo distinguere due principali categorie di *atleta*: gli atleti seniores sono coloro che hanno raggiunto l'età adatta a giocare nella prima squadra, mentre gli atleti appartenenti alle categorie giovanili saranno chiamati *juniores*. Le due categorie di atleti hanno caratteristiche comuni come numero di cartellino (che identifica univocamente un giocatore), nome, cognome, numero di telefono, indirizzo e-mail, residenza, dati biometrici, ruolo e risultati ottenuti durante i test fisici. Per gli atleti juniores, inoltre, è importante indicare la categoria di appartenenza (under 18, under 10...), mentre i giocatori seniores possono essere in possesso di un contratto e quindi di uno stipendio da giocatore.
Il programma deve tenere traccia dei certificati di idoneità alla pratica sportiva(visite mediche) e del loro stato (valido o scaduto, data di scadenza, i certificati hanno validità annuale), inoltre, per semplificare la gestione degli infortuni, è importante segnalare l'eventuale stato di infortunio, in modo che il medico della squadra possa essere messo tempestivamente a conoscenza dello stato di salute degli atleti.
Oltre agli atleti ci sono altri membri dello staff di fondamentale importanza, è stato già citato il medico, ogni squadra poi ha un allenatore, anch'esso identificato dal numero di tessera, e di cui interessa sapere nome, cognome, numero di telefono, indirizzo e-mail e salario e la categoria allenata. Infine abbiamo volontari e accompagnatori che supportano le altre figure del club, la differenza tra i due è che gli accompagnatori devono essere in possesso di un numero di tesseramento che li identifica, mentre i volontari no, i volontari saranno quindi identificati tramite codice fiscale, di entrambi poi si vuole sapere nome, cognome, numero di telefono e indirizzo e-mail.
L'attività delle associazioni si svolge attraverso eventi, che avranno un numero identificativo, una data, un'ora e un luogo, potranno essere di tre tipi: allenamento, partita o eventi sociali e l'applicazione dovrebbe permettere di registrare le presenze dei membri dell'associazione all'evento.
Infine l'associazione interagisce con altre aziende attraverso le sponsorizzazioni, ciascuna azienda sponsor sarà identificata dalla relativa partita IVA, nome, recapiti telefonici e

e-mail, andrà registrata la data di inizio e di fine della sponsorizzazione.

3 Progettazione concettuale

Dai requisiti esposti nella sezione precedente è stato ricavato lo schema concettuale riportato in figura 1.
La progettazione dello schema concettuale si è sviluppata secondo un approccio *inside-outside* dove, a partire dai requisiti fondamentali, si è proceduto a definire, dapprima le entità fondamentali e i relativi attributi e relazioni, procedendo via via con la definizione di entità minori e di ulteriori dettagli.

3.1 Scelte progettuali

Per rappresentare le entità *atleta juniores* e *atleta seniores* si è deciso di adottare una *generalizzazione totale ed esclusiva* poichè è esclusa la possibilità che un atleta sia contemporaneamente seniores e juniores, così come tutti gli atleti ricadono in una di queste categorie, inoltre è da notare che si tratta di una generalizzazione *completa*, in quanto l'unione di atleti juniores e seniores forma l'insieme degli atleti.
É rilevante la presenza di attributi composti per l'entità atleta. Per rappresentare al meglio il concetto dei certificati medici e la relativa scadenza si è scelto di utilizzare un'entità apposita, con un solo attributo che rappresenta la data di rilascio che, convenzionalmente coincide con quella di scadenza, avendo solitamente validità annuale. Ciascun atleta può essere in possesso di al più un certificato e un certificato deve essere necessariamente posseduto da un giocatore, infatti il certificato è un'*entità debole*, in quanto il suo identificatore è necessariamente esterno, ed è costituito da data e giocatore proprietario.
Per fare in modo che il contratto fosse entità debole, è stato introdotto il numero di protocollo, un progressivo unico per ogni contratto, infatti in questo caso sarebbe stato complesso ricorrere ad identificatori esterni in quanto i contratti sono coinvolti in due relazioni differenti, e avrebbero dovuto avere identificatori differenti a seconda della relazione in cui erano interessati.
Per migliorare la gestione delle transazioni è stato introdotto, sotto forma di entità, il concetto di pagamento, anche se non era esplicitamente menzionato all'interno dei requisiti, da notare che la partecipazione di pagamenti in tutte le relazioni è facoltativa, questo perché idealmente una società potrebbe fare affidamento unicamente sul proprio capitale, senza bisogno di ricevere finanziamenti e avendo staff di soli volontari.
La relazione di sponsorizzazione necessita di attributi che ne definiscano la durata temporale.
Infine è stata introdotta l'entità "società", necessaria per congiungere alcuni concetti tra loro.

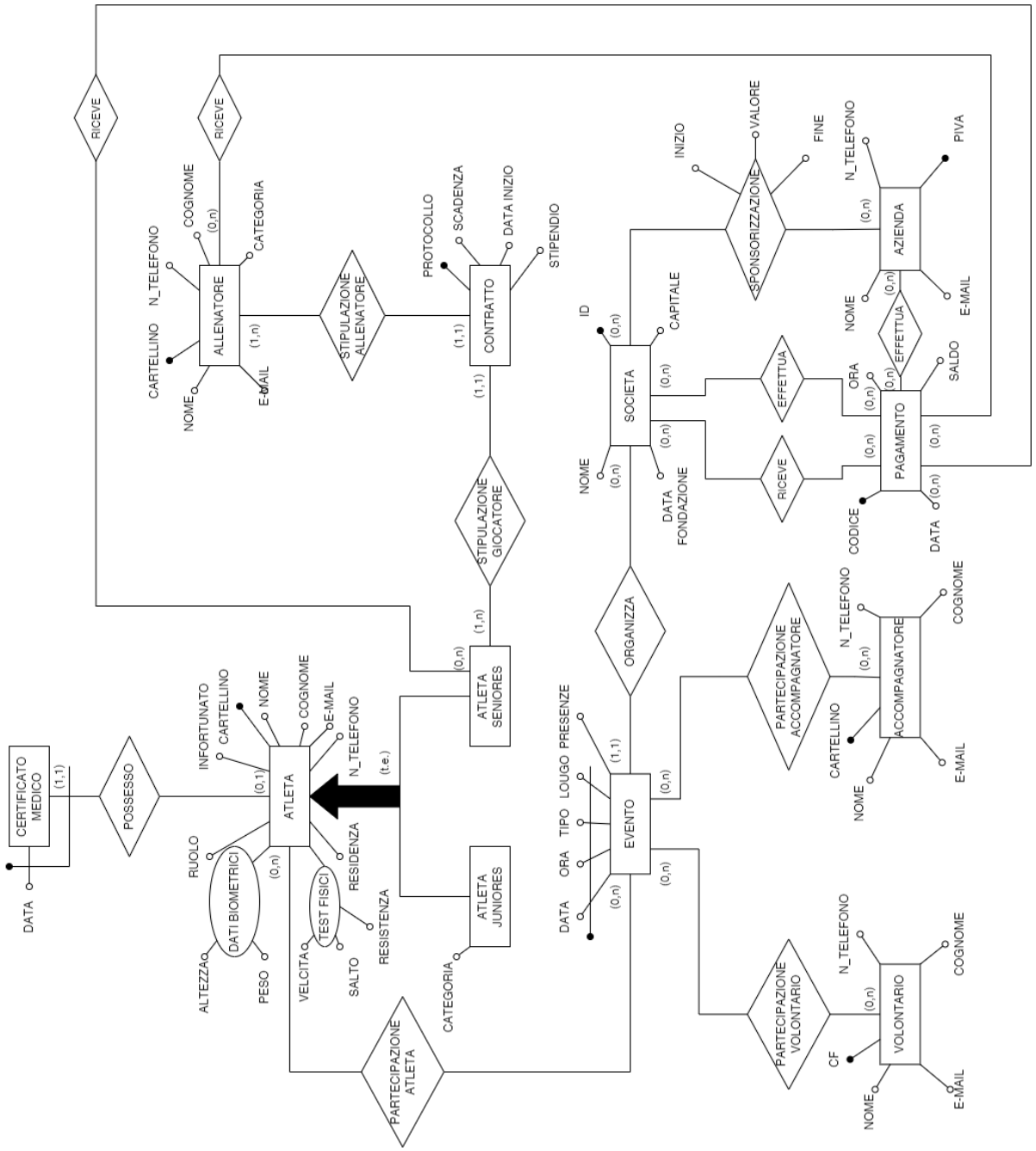

Figura 1: Schema Entità-Relazione della base di dati.

3.2 Ristrutturazione dello schema

Prima di poter passare alla progettazione dello schema logico è necessario preparare lo schema concettuale, in particolare gli obbiettivi della ristrutturazione saranno:

- ristrutturazione degli attributi multi-valore presenti in *atleta*,
- eliminazione della generalizzazione *atleta*,

In seguito bisognerà provvedere al trasporto delle chiavi primarie sulle entità che possiedono identificatori esterni.

3.3 Eliminazione di attributi multi-valore

Per prima cosa è necessario eliminare gli attributi multi-valore, ovvero "dati biometrici" e "test fisici". Per questi attributi sono state adottate strategie diverse: per quanto riguarda dati biometrici, trattandosi di attributi semplici e strettamente correlati ad un giocatore, e considerando che non possono esserci giocatori privi di tali attributi, la soluzione adottata è stata di aggiungere gli attributi che componevano i dati biometrici come attributi semplici a "atleta". Invece, per l'attributo "test fisici" la soluzione è stata quella di aggiungere un'apposita entità.

3.4 Eliminazione della generalizzazione

Al fine di poter tradurre correttamente lo schema concettuale in uno schema logico era necessario eliminare la generalizzazione "atleta". Per fare ciò è stato effettuato un *collasso verso l'alto*, preferito perché avrebbe consentito di rappresentare ugualmente tutte le entità, essendo la generalizzazione *totale*, senza introdurre ridondanze, poiché la generalizzazione era esclusiva, inoltre il collasso verso l'alto è stato preferito dal momento che le due entità figlie "atleta juniores" e "atleta seniores" possiedono pochi attributi di specializzazione ("categoria" per gli atleti juniores) e di conseguenza le ridondanze risultano contenute. Per identificare l'appartenenza all'una o all'altra sotto-entità verrà

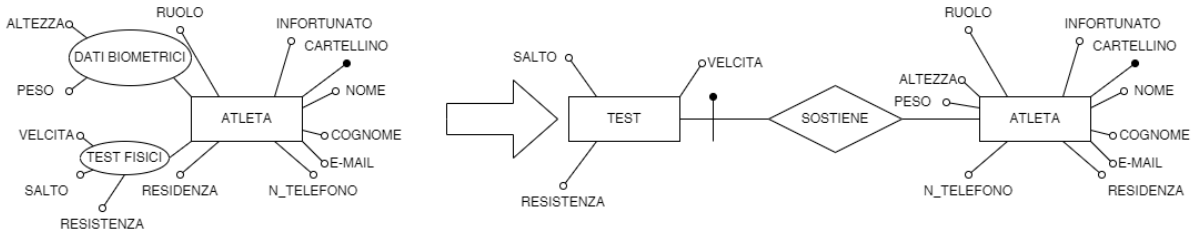

Figura 2: Eliminazione degli attributi multi-valore.

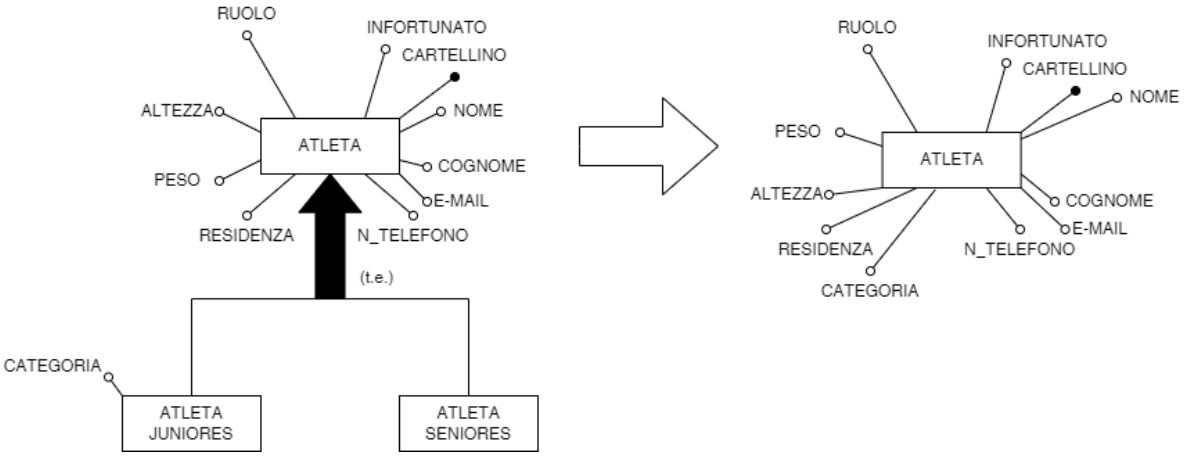

Figura 3: Collasso verso l'alto della generalizzazione "atleta".

aggiunto il valore "seniores" all'attributo categoria. Inoltre la partecipazione obbligatoria di "atleta seniores" alla relazione "stipula" è stata ora resa opzionale, perché gli atleti juniores non devono necessariamente parteciparvi.

3.5 Ristrutturazione dell'entità "pagamento"

Durante la ristrutturazione dello schema concettuale, è stata riscontrata una lacuna nello schema di partenza: l'entità pagamento non permetteva di tenere traccia di mittenti e destinatari dei pagamenti. Essendo coinvolta in molte relazioni, destinatari e mittenti di un pagamento erano inoltre entità eterogenee, pertanto la soluzione adottata è stata quella di eliminare l'entità pagamento, sostituendola con tre relazioni, una per ciascun tipo di pagamento, dotate di appositi attributi per indicare la data, l'ora e l'ammontare della somma pagata.
In questo modo, ci saranno diverse tipologie di pagamenti, a seconda delle entità coinvolte, destinatario e mittente figureranno nella futura relazione, assieme alle altre proprietà rilevanti.

3.6 Schema ristrutturato

Per concludere la ristrutturazione è stata effettuata una verifica delle ridondanze, inoltre, per semplificare la traduzione dello schema, gli identificatori esterni sono stati trasportati alle entità che li possedevano. Lo schema finale successivo alla ristrutturazione è riportato in figura 4.

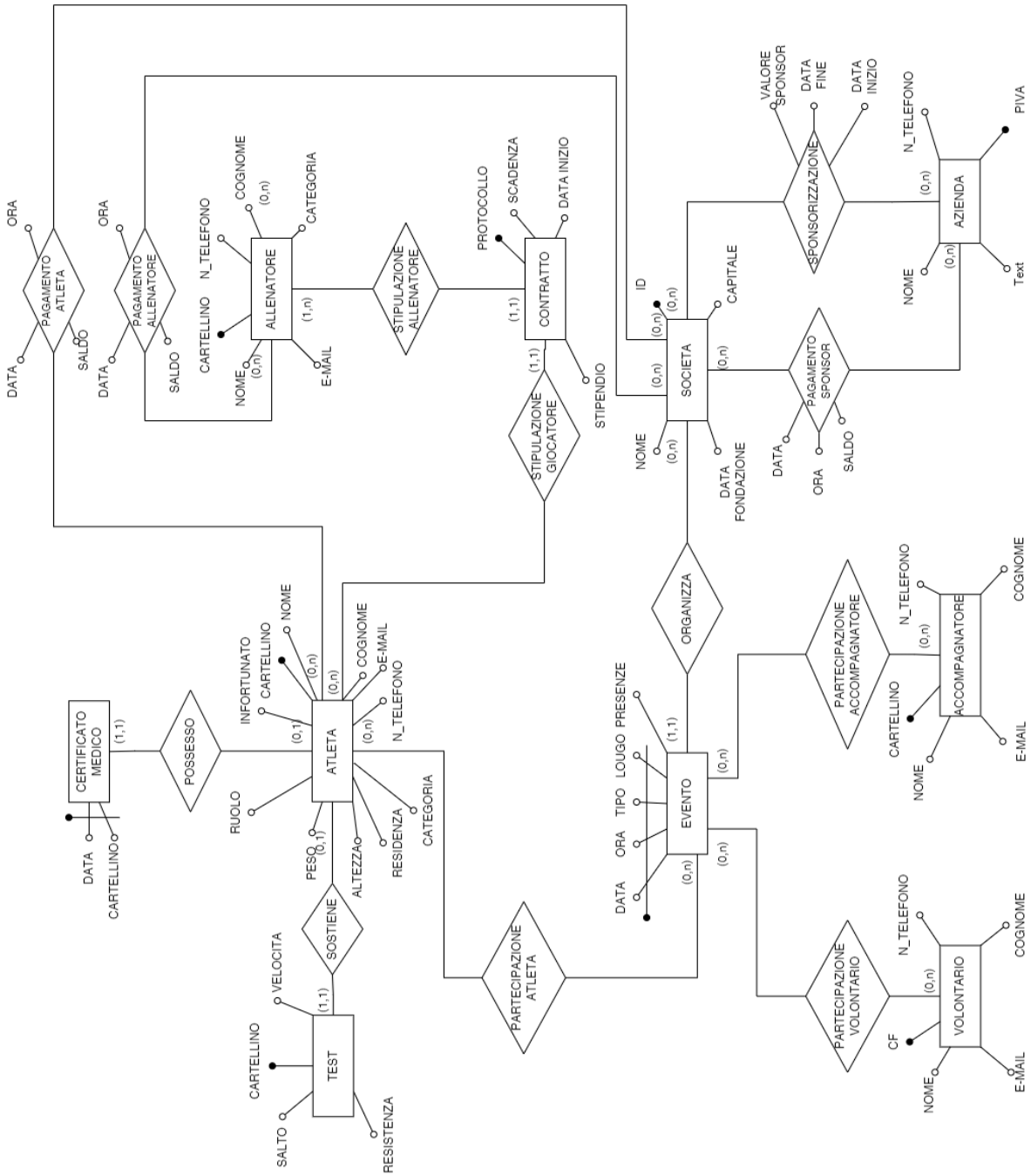

Figura 4: Schema concettuale ristrutturato

4 Progettazione logica

Una volta elaborato uno schema concettuale soddisfacente per la base di dati è necessario tradurre tale schema in uno schema logico. Il modello logico prescelto è quello *relazionale*. La progettazione logica si svilupperà secondo le fasi:

1. ristrutturazione dello schema logico (già svolta),
2. traduzione delle entità,
3. traduzione delle relazioni *molti a molti*,
4. traduzione delle relazioni *uno a molti* e *uno a uno*.

4.1 Traduzione delle entità

```
ATLETA (CARTELLINO,NOME,CONGOME,EMAIL,N_TELEFONO,CATEGORIA,
        RESIDENZA,ALTEZZA,PESO,RUOLO,INFORTUNATO)
CERTIFICATO MEDICO (CARTELLINO,DATA)
TEST (CARTELLINO,SALTO,VELOCITA,RESISTENZA)
ALLENATORE (CARTELLINO,NOME,CONGOME,EMAIL,N_TELEFONO,CATEGORIA)
CONTRATTO (PROTOCOLLO,DATA INIZIO,SCADENZA,STIPENDIO)
VOLONTARIO (CF,NOME,CONGOME,EMAIL,N_TELEFONO)
ACCOMPAGNATORE (CARTELLINO,NOME,CONGOME,EMAIL,N_TELEFONO)
EVENTO (DATA,ORA,TIPO,LUOGO,PRESENZE)
SOCIETA (ID,NOME,DATA FONDAZIONE,CAPITALE)
AZIENDA (P.IVA,NOME,N_TELEFONO,E-MAIL)
```

4.2 Traduzione delle relazioni *molti a molti*

```
PARTECIPAZIONE ATLETA (ATLETA,DATA,ORA,TIPO,LUOGO)
PARTECIPAZIONE ACCOMPAGNATORE (ACCOMPAGNATORE,DATA,ORA,TIPO,LUOGO)
PARTECIPAZIONE VOLONTARIO (VOLONTARIO,DATA,ORA,TIPO,LUOGO)
PAGAMENTO SPONSOR(SOCIETA,P.IVA,DATA,ORA,SALDO)
PAGAMENTO ATLETA(SOCIETA,ATLETA,DATA,ORA,SALDO)
PAGAMENTO ALLENATORE(SOCIETA,ALLENATORE,DATA,ORA,SALDO)
SPONSORIZZAZIONE(SOCIETA,P.IVA,DATA INIZIO,DATA FINE,VALORE SPONSOR)
```

4.3 Traduzione delle relazioni *uno a uno* e *uno a molti*

La relazione di possesso del certificato medico è già stata rappresentata intrinsecamente, infatti, introducendo il vincolo di integrità referenziale tra cartellino di visita medica e cartellino di atleta possiamo definire in modo completo la relazione. Lo stesso vale per la relazione di sostenimento dei test fisici.
Le traduzioni delle relazioni rimanenti sono definite in seguito.

```
ORGANIZZAZIONE[1] (ID,DATA,ORA,TIPO,LUOGO)
STIPULAZIONE GIOCATORE (ATLETA,CONTRATTO)
STIPULAZIONE ALLENATORE (ALLENATORE,CONTRATTO)
```

4.4 Schema logico finale con vincoli di integrità

```
ATLETA (CARTELLINO,NOME,CONGOME,EMAIL,N_TELEFONO,CATEGORIA,
        RESIDENZA,ALTEZZA,PESO,RUOLO,INFORTUNATO)
CERTIFICATO MEDICO (CARTELLINO,DATA)
TEST (CARTELLINO,SALTO,VELOCITA,RESISTENZA)
ALLENATORE (CARTELLINO,NOME,CONGOME,EMAIL,N_TELEFONO,CATEGORIA)
CONTRATTO (PROTOCOLLO,DATA INIZIO,SCADENZA,STIPENDIO)
VOLONTARIO (CF,NOME,CONGOME,EMAIL,N_TELEFONO)
ACCOMPAGNATORE (CARTELLINO,NOME,CONGOME,EMAIL,N_TELEFONO)
EVENTO (DATA,ORA,TIPO,LUOGO,PRESENZE)
SOCIETA (ID,NOME,DATA FONDAZIONE,CAPITALE)
AZIENDA (P.IVA,NOME,N_TELEFONO,E-MAIL)
PARTECIPAZIONE ATLETA (ATLETA,DATA,ORA,TIPO,LUOGO)
PARTECIPAZIONE ACCOMPAGNATORE (ACCOMPAGNATORE,DATA,ORA,TIPO,LUOGO)
PARTECIPAZIONE VOLONTARIO (VOLONTARIO,DATA,ORA,TIPO,LUOGO)
PAGAMENTO SPONSOR(SOCIETA,P.IVA,DATA,ORA,SALDO)
PAGAMENTO ATLETA(SOCIETA,ATLETA,DATA,ORA,SALDO)
PAGAMENTO ALLENATORE(SOCIETA,ALLENATORE,DATA,ORA,SALDO)
SPONSORIZZAZIONE(SOCIETA,P.IVA,DATA INIZIO,DATA FINE,VALORE SPONSOR)
ORGANIZZAZIONE (ID,DATA,ORA,TIPO,LUOGO)
STIPULAZIONE GIOCATORE (ATLETA,CONTRATTO)
STIPULAZIONE ALLENATORE (ALLENATORE,CONTRATTO)
```

In questa fase sono state riscontrate alcune criticità che non erano state individuate in precedenza, in particolare è stato necessario aggiungere alla chiave primaria delle relazioni "pagamento atleta", "pagamento giocatore" e "pagamento sponsor" gli attributi "data" e "ora". Questo perché altrimenti sarebbe stato impossibile rappresentare la possibilità che avvengano più pagamenti tra gli stessi soggetti, in quanto le chiavi primarie di due

[1] è stato usato "organizzazione" al posto di "organizza" per migliorare la leggibilità.

pagamenti che coinvolgessero gli stessi soggetti sarebbero state uguali. La situazione è analoga per la relazione "sponsorizzazione", in questo caso alla chiave primaria è stata aggiunto l'attributo "data inizio", per contemplare la possibilità che un'azienda diventi sponsor più volte a intervalli di tempo differenti.

5 Query SQL rilevanti

Una volta tradotto lo schema logico in linguaggio SQL, sono state definite alcune query di particolare rilevanza, che verranno poi impiegate per gestire l'interazione tra il front-end e il back-end.

5.1 Operazioni di modifica

Per prima cosa è stata definita la possibilità di inserire nuovi atleti all'interno del database tramite la query riportata nel listato 5.1.

```
INSERT INTO athlete (id_no,name,surname,telephone_no,email,category,
    height,weight,position,adress)
VALUES ('4392','Mario','Rossi','3318746524','ma.rossi@libero.it','
    Seniores','180','80','ala','via mazzini, 15');
```

Listato 1: Query per l'inserimento di un atleta.

Il procedimento è analogo qualora si voglia aggiungere una qualunque altra entità. Caso particolare è rappresentato dall'inserimento di nuovi pagamenti, in quanto comportano la necessità di aggiornare il capitale della società, infatti, nel caso di pagamenti verso allenatori o atleti, il capitale della società dovrà essere decrementato, mentre nel caso di pagamenti ricevuti da sponsor, il capitale andrà incrementato.

```
SELECT club.club_name, club.id_no, athlete.id_no id, athlete.name,
    athlete.surname, payment_date, payment_time, total
FROM (club INNER JOIN athlete_payment ON club.id_no = club)
INNER JOIN athlete ON athlete = athlete.id_no
```

Listato 2: Query per ottenere i dati relativi a un pagamento ad un atleta.

Da notare che l'interrogazione riportata nel listato 5.1 ha richiesto l'uso di una ridenominazione per evitare ambiguità.

```
INSERT INTO sponsor_payment (club , company , payment_date ,
    payment_time, total)
VALUES ((SELECT id_no FROM club), ?,?,?,?)
```

[2]

Listato 3: Query per l'inserimento di un nuovo pagamento.

```
UPDATE club
SET capital=((SELECT capital FROM club)-?)
```

Listato 4: Query per l'aggiornamento del capitale sociale.

La modifica degli altri pagamenti avviene allo stesso modo.
Caso particolare è rappresentato dall'inserimento di un nuovo contratto, infatti, c'è la necessità di inserire separatamente l'inserimento di contratti di atleti e di allenatori

[2] D'ora in avanti il segno "?" verrà stato utilizzato come segnaposto nelle query.

e l'inserimento di un nuovo contratto richiede contestualmente l'aggiornamento della tabella "contract" e della tabella "stipulate_atlete" o "stipulate_coach", tale azione può essere svolta facaendo uso delle query riportate nel listato 5.1

```
INSERT INTO contract (protocol_no,starting_date,expiry_date,salary)
VALUES (?,?,?,?));

INSERT INTO stipulate_athlete (contract,athlete)
VALUES (?,?);
```

Listato 5: Operazioni per l'inserimento di un nuovo contratto giocatore.

L'ultima operazione di modifica prevista per l'applicazione consiste nella possibilità di aggiornare lo stato di infortunato di un atleta. Tale modifica viene eseguita nella query indicata nel listato 5.1.

```
UPDATE athlete
SET injured = 0
WHERE (id_no=?)
```

Listato 6: Query per l'aggiornamento dell'infortunio di un giocatore.

5.2 Operazioni di interrogazione

Di grande importanza per l'applicazione sono anche le operazioni di interrogazione, che permettono di presentare all'utente le informazioni contenute nella base di dati. Per la corretta gestione della società è importante essere a conoscenza dello stato di validità dei certificati medici degli atleti. Sono state quindi approntate due query distinte rispettivamente per ottenere gli elenchi dei certificati validi e non validi, che verranno poi presentati in modo diverso all'utente.

```
SELECT athlete.id_no, athlete.name, athlete.surname, date_of_emission
FROM athlete INNER JOIN medical_cert ON athlete.id_no = medical_cert.
    id_no
WHERE (medical_cert.date_of_emission > DATE('now','-1 year'))
ORDER BY (date_of_emission)
```

Listato 7: Interrogazione per ottenere l'elenco i certificati non scaduti.

Ipotizzando un periodo di validità di un anno, per valutare la scadenza di un certificato medico è stata impiegata la funzione DATE(), di cui è riportata la sintassi.

```
DATE('timestring','modifier',...)
```

Listato 8: Uso della funzione DATE().

I *modifiers* sono stringhe che riportano intervalli o periodi di tempo in modo verboso, la funzione restituisce quindi in formato DATE la stringa, che deve essere nella forma di una data, *'timestring'*, modificata come indicato dai *modifiers*. Nel caso specifico quindi,

un certificato è valido se è stato rilasciato in una data successiva alla data odierna meno un anno. Una volta terminata l'interrogazione i certificati vengono proposti in ordine di scadenza, dalla più vicina alla più lontana. In modo analogo vengono ricavati i certificati medici scaduti.
Nel listato 5.2 è riportata l'interrogazione per ottenere i risultatati realizzati da ciascun atleta nei test fisici.

```
SELECT athlete.id_no, name, surname, jump, sprint, endurance
FROM athlete LEFT OUTER JOIN physical_test ON athlete.id_no =
    physical_test.id_no
ORDER BY athlete.id_no;
```

Listato 9: Interrogazione per ottenere i test fisici sostenuti da ciascun atleta.

Da cui poi è possibile estrarre il migliore atleta in una data categoria di test.

```
SELECT athlete.id_no, name, surname, jump, sprint, endurance
FROM athlete LEFT OUTER JOIN physical_test ON athlete.id_no =
    physical_test.id_no
WHERE (jump >= (SELECT MAX(jump) FROM physical_test));
```

Listato 10: Interrogazione per il miglior atleta in una data categoria di test.

Da requisiti di programma era richiesto di poter prendere conoscenza delle presenza agli eventi, ad esempio, se si volessero sapere le presenze degli atleti agli allenamenti si potrebbe procedere come indicato nel listato 5.2, occorre fare particolare attenzione alla ridenominazione della colonna "COUNT(*)", necessaria per poter accedere attraverso Python ai valori contenuti nella relativa colonna.

```
SELECT * FROM athlete LEFT JOIN (
    SELECT athlete_attendance.athlete, COUNT(*) AS pres
    FROM athlete_attendance
    WHERE (kind='allenamento')
    )
ON athlete = id_no
ORDER BY(id_no)
```

Listato 11: Interrogazione per ottenere le presenze agli allenamenti di un atleta

Infine potrebbe essere utile risalire alle sponsorizzazioni presenti e passate a cui ha partecipato un'azienda.

```
SELECT p_iva, company_name, telephone_no, email, start_date, end_date,
    sponsor_value
FROM company INNER JOIN sponsorship ON company.p_iva = sponsorship.
    company
```

Listato 12: Query per ottenere l'elenco delle sponsorizzazioni e la relativa azienda.

5.3 Viste

In previsione del fatto che la conoscenza degli atleti infortunati non sia di interesse della sola società, ma anche dei medici, è stata predisposta una vista apposita. La scelta di utilizzare una vista è dettata principalmente dalla volontà di mantenere un ulteriore livello di privatezza dei dati, inoltre l'utilizzo della vista permette di mettere i medici a conoscenza dei soli dati di loro interesse.

```
1 CREATE VIEW injured_athletes (id_no , name , surname , category ,
    injured) AS
2     SELECT
3         id_no, name, surname, category, injured
4     FROM
5         athlete
6     WHERE
7         (injured = TRUE);
```

Listato 13: Definzione di una vista

Tuttavia il motore utilizzato per gestire il database nell'applicazione, ovvero SQLite, non permette la modifica di viste, pertanto, per gestire lo stato di infortunio di un atleta è stato necessario agire direttamente sulla tabella athletes, come riportato nel listato 5.1.

6 Implementazione dell'interfaccia utente

Una volta progettata la base di dati, è stato possibile procedere con la realizzazione di un programma attraverso cui l'utente potesse interagire con la base di dati.

6.1 Strumenti utilizzati

Gli strumenti impiegati nella realizzazione fisica del database e della relativa applicazione sono stati scelti anche in base ai vincoli di tempo e ai limiti imposti dall'esperienza del programmatore. Il database è stato implementato attraverso SQLite, un DBMS *self-contained*, che rispetto ad altre soluzioni (la principale alternativa era mySQL) presentava il vantaggio di essere completo di tutti gli strumenti necessari. Essendo appunto self-contained, infatti, SQLite non fa uso di architetture client-server, inoltre SQLite risulta più morbido dal punto di vista della tipizzazione. I principali vantaggi che ne derivano sono l'immediatezza nell'utilizzo, la facilità di configurazione e la grande portabilità, fattori rilevanti nell'attuale caso d'uso, d'altro canto risulta essere poco scalabile e carente dal punto di vista della sicurezza, non disponendo di un sistema integrato di gestione degli utenti. Considerando che il progetto aveva come obbiettivo quello di realizzare un'applicazione dimostrativa autonoma in breve tempo e che potesse essere facilmente portabile SQLite è risultato essere il compromesso ideale. Inoltre SQLite è naturalmente supportato dal linguaggio di programmazione scelto, ovvero Python. Per la gestione del database è stato impiegato il modulo SQLite3, preferito a SQLAlchemy in quanto quest'ultimo avrebbe necessitato di un database separato con cui lavorare. Per la realizzazione del beckend è stato impiegato il framework Flask, anche in questo caso la scelta è stata dettata dalla rapidità e semplicità d'uso, mentre il frontend è stato realizzato con Bootstrap. Per l'interazione tra frontend e beckend Flask impiega Jinja2, che permette di inserire parti di codice Python all'interno dei template HTML. Infine, per testare e gestire il database in fase di sviluppo è stato impiegato il programma "DB Browser for SQLite" come GUI, una soluzione leggera e semplice.

6.2 Popolazione del database

Per eseguire i test e popolare il database con dati dimostrativi sono stati generati dati attraverso alcuni tool online sottonforma di file .csv, quindi i dati sono stati importati nel database attraverso la funzione apposita dell'applicazione DB Browser for SQLite e infine inseriti nelle tabelle del database attraverso script Python simili a quello riportato nel listato 6.2.

```
1 import sqlite3
2
3 conn=sqlite3.connect('sportdata.db')
4 conn.row_factory = sqlite3.Row
```

```
values = conn.execute('''SELECT * FROM test''').fetchall()
for v in values:
    print(v.keys())
    try:
        conn.execute('''INSERT INTO physical_test(id_no,jump,sprint,
    endurance)
                    VALUES (?,?,?,?)''',(v['ID'],v['Valore'],v['Tempo_s
    '],v['Tempo_m']))
    except: continue
    conn.commit()
conn.close()
```

Listato 14: Script Python per la popolazione del database.

6.3 Gestione del DB attraverso SQLite3

Per la gestione del database attraverso l'applicazione è stato fatto ampiamente uso dei *prepeared statement*, ovvero espressioni preimpostate e compilabili programmaticamente per gestire in sicurezza le transazioni effettuate dall'utente.

```
def pagamenti_allenatori():
    conn = sqlite3.connect(DB)
    conn.row_factory = sqlite3.Row
    allenatori = conn.execute('''SELECT club.club_name, club.id_no,
    coach.id_no id, coach.name, coach.surname, payment_date,
    payment_time,total FROM (club INNER JOIN coach_payment ON club.id_no
     = club) INNER JOIN coach ON coach = coach.id_no''').fetchall()
    if request.method == 'POST':
        conn.execute('''INSERT INTO coach_payment (club, coach,
    payment_date, payment_time, total) VALUES ((SELECT id_no FROM club),
     ?,?,?,?)''',(request.form['all'],request.form['data'], request.form
    ['ora'], request.form['total']))
        conn.commit()
        decrCapital(request.form['total'],conn)
    conn.close()
    return render_template('pagamenti_allenatori.html', allenatori=
    allenatori)
```

Listato 15: Script Python per la l'interazione con il database.

La funzione riportata nel listato 6.3, ad esempio, mostra come si possano realizzare interrogazioni attraverso la funzione *execute* della classe *Connection*, i parametri passati alla funzione sono una stringa contente l'interrogazione da eseguire, gli eventuali punti di domanda funzionano da segnaposto e in fase di esecuzione verranno sostituiti in modo ordinato con gli argomenti della lista passata come secondo argomento. Altro modulo impiegato nella realizzazione dell'app è il modulo *request*, che è stato utilizzato per gestire adeguatamente diversi tipi di richiesta e il modulo render, utilizzato per presentare all'utente le pagine HTML.

6.4 Hosting dell'applicazione

Ai fini dimostrativi l'applicazione è ospitata sul servizio *Python Anywhere* ed è accessibile a questo indirizzo utilizzando rvaltorta10 come username e BasiDati2023 come password.

7 Conclusioni

L'applicazione realizzata ha raggiunto un buon livello di completezza, permette di interagire in modo soddisfacente con il database, effettuando le operazioni di amministrazione necessarie e consultando in modo rapido tutti i dati necessari. Dato l'approccio di sviluppo funzionale rimane possibile aggiungere con una discreta facilità nuove funzionalità all'applicazione anche se rimane eveidente la necessità di un refactoring, in particolare lato frontend potrebbe essere vantaggioso introdurre l'ereditarietà tra i diversi template, una piccola miglioria lato beckend invece potrebbe essere quella di includere le funzioni di connessione al database all'interno di una funzione apposita.

```
def selecter(db: str, query: str, num='all'):
    conn = sqlite3.connect(db)
    conn.row_factory = sqlite3.Row
    if num == 'all':
        result = conn.execute(query).fetchall()
    if num == 'one':
        result = conn.execute(query).fetchone()
    if num == 'many':
        result = conn.execute(query).fetchmany()
    conn.close()
    return result
```

Listato 16: Funzione per la connessione e l'interrogazione al database.

7.1 Problemi riscontrati

Durante lo sviluppo e il testing dell'applicazione uno dei principali difetti della progettazione è emerso: salvando i numeri telefonici e le partite iva come interi si perdevano gli "0" iniziali, è stato quindi necessario modificare la definizione delle relative colonne all'interno del database, operazione che è risultata particolarmente facile attraverso l'uso di DB Browser for SQLite.

7.2 Sviluppi futuri

L'applicazione risulta essere un valido esempio di *minimum viable product* e con l'integrazione di alcune funzionalità potrebe arrivare ad essere un prodotto, magari open source, molto valido. Sicuramente uno dei punti da migliorare è la gestione della sicurezza, in particolare l'introduzione di un sistema di user login, che potrebbe differenziare le tipoliogie di utenti e i relativi privilegi, altro punto rilevante è la necessità di spostare l'applicazione su altri DBMS per avere una maggiore scalabilità. Altra funzionalità interessante che potrebbe essere introdotta è un sistema di notifiche.

Elenco delle figure

Listings

A Struttura del database SQL

```sql
CREATE SCHEMA IF NOT EXISTS sport_data;

USE sport_data;

# # # # # # # # # # # # # # # # # # # #
# relazione ATLETA
# # # # # # # # # # # # # # # # # # # #

CREATE TABLE IF NOT EXISTS `athlete` (
  `id_no` int NOT NULL, #cartellino
  `name` varchar(15) NOT NULL,
  `surname` varchar(15) NOT NULL,
  `telephone_no` int DEFAULT NULL,
  `email` varchar(30) NOT NULL,
  `category` varchar(15),
  `height` int NOT NULL,
  `weight` int NOT NULL,
  `position` varchar(20) DEFAULT NULL, #ruolo
  `adress` varchar(20) DEFAULT NULL,
  `injured` boolean DEFAULT false NOT NULL,
  PRIMARY KEY (`id_no`),
  CONSTRAINT `atleta_chk_1` CHECK ((((`category` = _utf8mb4'U8') or (`
    category` = _utf8mb4'U10') or (`category` = _utf8mb4'U12') or (`
    category` = _utf8mb4'U14') or (`category` = _utf8mb4'U16') or (`
    category` = _utf8mb4'U18') or (`category` = _utf8mb4'Seniores')))
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

# # # # # # # # # # # # # # # # # # # #
# relazione CERTIFICATO MEDICO
# # # # # # # # # # # # # # # # # # # #

CREATE TABLE IF NOT EXISTS medical_cert (
    id_no INT UNIQUE,
    date_of_emission DATE,
    PRIMARY KEY (id_no , date_of_emission),
    FOREIGN KEY (id_no)
        REFERENCES athlete (id_no)
        ON DELETE CASCADE ON UPDATE CASCADE
);

# # # # # # # # # # # # # # # # # # # #
# relazione TEST
# # # # # # # # # # # # # # # # # # # #

CREATE TABLE IF NOT EXISTS physical_test (
    id_no INT,
    jump FLOAT,
```

```
    sprint FLOAT,
    endurance FLOAT,
    CONSTRAINT PRIMARY KEY (id_no),
    FOREIGN KEY (id_no)
        REFERENCES athlete (id_no)
        ON DELETE CASCADE ON UPDATE CASCADE
);

# # # # # # # # # # # # # # # # # # #
# relazione ALLENATORE
# # # # # # # # # # # # # # # # # # #

CREATE TABLE IF NOT EXISTS `coach` (
  `id_no` int NOT NULL, #cartellino
  `name` varchar(15) NOT NULL,
  `surname` varchar(15) NOT NULL,
  `telephone_no` int DEFAULT NULL,
  `email` varchar(30) NOT NULL,
  `category` varchar(15),
  PRIMARY KEY (`id_no`),
  CONSTRAINT `coach_chk_1` CHECK ((((`category` = _utf8mb4'U8') or (`
    category` = _utf8mb4'U10') or (`category` = _utf8mb4'U12') or (`
    category` = _utf8mb4'U14') or (`category` = _utf8mb4'U16') or (`
    category` = _utf8mb4'U18') or (`category` = _utf8mb4'Seniores')))
);

# # # # # # # # # # # # # # # # # # #
# relazione CONTRATTO
# # # # # # # # # # # # # # # # # # #

CREATE TABLE IF NOT EXISTS contract (
    protocol_no INT AUTO_INCREMENT,
    starting_date DATE,
    expiry_date DATE,
    salary FLOAT,
    PRIMARY KEY (protocol_no)
);

# # # # # # # # # # # # # # # # # # #
# relazione ACCOMPAGNATORE
# # # # # # # # # # # # # # # # # # #

CREATE TABLE IF NOT EXISTS `companion` (
    `id_no` INT NOT NULL,
    `name` VARCHAR(15) NOT NULL,
    `surname` VARCHAR(15) NOT NULL,
    `telephone_no` INT DEFAULT NULL,
    `email` VARCHAR(30) NOT NULL,
    PRIMARY KEY (`id_no`)
```

```
);

# # # # # # # # # # # # # # # # # # #
# relazione VOLONTARIO
# # # # # # # # # # # # # # # # # # #

CREATE TABLE IF NOT EXISTS `volunteer` (
    `cf` VARCHAR(16),
    `name` VARCHAR(15) NOT NULL,
    `surname` VARCHAR(15) NOT NULL,
    `telephone_no` INT DEFAULT NULL,
    `email` VARCHAR(30) NOT NULL,
    PRIMARY KEY (`cf`)
);

# # # # # # # # # # # # # # # # # # #
# relazione EVENTO
# # # # # # # # # # # # # # # # # # #

CREATE TABLE IF NOT EXISTS sport_event (
    event_date DATE,
    event_time TIME,
    kind VARCHAR(15),
    place VARCHAR(30),
    attendance INT,
    PRIMARY KEY (event_date , event_time , kind , place),
    CONSTRAINT CHECK (kind = 'partita' OR kind = 'sociale'
        OR kind = 'allenamento')
);

# # # # # # # # # # # # # # # # # # #
# relazione SOCIETA
# # # # # # # # # # # # # # # # # # #

CREATE TABLE IF NOT EXISTS club (
    id_no INT,
    club_name VARCHAR(20) NOT NULL,
    foundation_year INT NOT NULL,
    capital FLOAT,
    PRIMARY KEY (id_no)
);

# # # # # # # # # # # # # # # # # # #
# relazione AZIENDA
# # # # # # # # # # # # # # # # # # #

CREATE TABLE IF NOT EXISTS company (
    p_iva INT,
    company_name VARCHAR(20),
```

```
    telephone_no INT,
    email VARCHAR(30),
    PRIMARY KEY (p_iva)
);

# # # # # # # # # # # # # # # # # # # #
# relazione PARTECIPAZIONE ATLETA
# # # # # # # # # # # # # # # # # # # #

CREATE TABLE IF NOT EXISTS athlete_attendance (
    athlete INT,
    event_date DATE,
    event_time TIME,
    kind VARCHAR(15),
    place VARCHAR(30),
    PRIMARY KEY (athlete , event_date , event_time , kind , place),
    FOREIGN KEY (athlete)
        REFERENCES athlete (id_no)
        ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (event_date , event_time , kind , place)
        REFERENCES sport_event (event_date , event_time , kind , place)
        ON DELETE CASCADE ON UPDATE CASCADE
);

# # # # # # # # # # # # # # # # # # # #
# relazione PARTECIPAZIONE ACCOMPAGNATORE
# # # # # # # # # # # # # # # # # # # #

CREATE TABLE IF NOT EXISTS companion_attendance (
    companion INT,
    event_date DATE,
    event_time TIME,
    kind VARCHAR(15),
    place VARCHAR(30),
    PRIMARY KEY (companion , event_date , event_time , kind , place),
    FOREIGN KEY (companion)
        REFERENCES companion (id_no)
        ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (event_date , event_time , kind , place)
        REFERENCES sport_event (event_date , event_time , kind , place)
        ON DELETE CASCADE ON UPDATE CASCADE
);

# # # # # # # # # # # # # # # # # # # #
# relazione PARTECIPAZIONE VOLONATRIO
# # # # # # # # # # # # # # # # # # # #

CREATE TABLE IF NOT EXISTS volunteer_attendance (
    volunteer VARCHAR(16),
```

```
    event_date DATE,
    event_time TIME,
    kind VARCHAR(15),
    place VARCHAR(30),
    PRIMARY KEY (volunteer , event_date , event_time , kind , place),
    FOREIGN KEY (volunteer)
        REFERENCES volunteer (cf)
        ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (event_date , event_time , kind , place)
        REFERENCES sport_event (event_date , event_time , kind , place)
        ON DELETE CASCADE ON UPDATE CASCADE
);

# # # # # # # # # # # # # # # # # # # #
# relazione PAGAMENTO SPONSOR
# # # # # # # # # # # # # # # # # # # #

CREATE TABLE IF NOT EXISTS sponsor_payment (
    club INT,
    company INT,
    payment_date DATE,
    payment_time TIME,
    total FLOAT NOT NULL,
    PRIMARY KEY (club , company , payment_date , payment_time),
    FOREIGN KEY (club)
        REFERENCES club (id_no)
        ON DELETE NO ACTION ON UPDATE CASCADE,
    FOREIGN KEY (company)
        REFERENCES company (p_iva)
        ON DELETE NO ACTION ON UPDATE CASCADE
);

# # # # # # # # # # # # # # # # # # # #
# relazione PAGAMENTO SPONSOR
# # # # # # # # # # # # # # # # # # # #

CREATE TABLE IF NOT EXISTS athlete_payment (
    club INT,
    athlete INT,
    payment_date DATE,
    payment_time TIME,
    total FLOAT NOT NULL,
    PRIMARY KEY (club , athlete , payment_date , payment_time),
    FOREIGN KEY (club)
        REFERENCES club (id_no)
        ON DELETE NO ACTION ON UPDATE CASCADE,
    FOREIGN KEY (athlete)
        REFERENCES athlete (id_no)
        ON DELETE NO ACTION ON UPDATE CASCADE
```

```
);

# # # # # # # # # # # # # # # # # # # #
# relazione PAGAMENTO SPONSOR
# # # # # # # # # # # # # # # # # # # #

CREATE TABLE IF NOT EXISTS coach_payment (
    club INT,
    coach INT,
    payment_date DATE,
    payment_time TIME,
    total FLOAT NOT NULL,
    PRIMARY KEY (club , coach , payment_date , payment_time),
    FOREIGN KEY (club)
        REFERENCES club (id_no)
        ON DELETE NO ACTION ON UPDATE CASCADE,
    FOREIGN KEY (coach)
        REFERENCES coach (id_no)
        ON DELETE NO ACTION ON UPDATE CASCADE
);

# # # # # # # # # # # # # # # # # # # #
# relazione SPONSORIZZAZIONE
# # # # # # # # # # # # # # # # # # # #

CREATE TABLE IF NOT EXISTS sponsorship (
    club INT,
    company INT,
    start_date DATE,
    end_date DATE,
    sponsor_value FLOAT,
    PRIMARY KEY (club , company , start_date),
    FOREIGN KEY (club)
        REFERENCES club (id_no)
        ON DELETE NO ACTION ON UPDATE CASCADE,
    FOREIGN KEY (company)
        REFERENCES company (p_iva)
        ON DELETE NO ACTION ON UPDATE CASCADE,
    CHECK (end_date > start_date)
);

# # # # # # # # # # # # # # # # # # # #
# relazione STIPULAZIONE GIOCATORE
# # # # # # # # # # # # # # # # # # # #

CREATE TABLE IF NOT EXISTS stipulate_athlete (
    athlete INT,
    contract INT,
    PRIMARY KEY (athlete , contract),
```

```
    FOREIGN KEY (athlete)
        REFERENCES athlete (id_no)
        ON DELETE NO ACTION ON UPDATE CASCADE,
    FOREIGN KEY (contract)
        REFERENCES contract (protocol_no)
        ON DELETE NO ACTION ON UPDATE CASCADE
);

# # # # # # # # # # # # # # # # # # # #
# relazione STIPULAZIONE ALLENATORE
# # # # # # # # # # # # # # # # # # # #

CREATE TABLE IF NOT EXISTS stipulate_coach (
    coach INT,
    contract INT,
    PRIMARY KEY (coach , contract),
    FOREIGN KEY (coach)
        REFERENCES coach (id_no)
        ON DELETE NO ACTION ON UPDATE CASCADE,
    FOREIGN KEY (contract)
        REFERENCES contract (protocol_no)
        ON DELETE NO ACTION ON UPDATE CASCADE
);

# # # # # # # # # # # # # # # # # # # #
# vista degli atleti infortunati
# # # # # # # # # # # # # # # # # # # #

CREATE VIEW injured_athletes (id_no , name , surname , category ,
   injured) AS
    SELECT
        id_no, name, surname, category, injured
    FROM
        athlete
    WHERE
        (injured = TRUE);
```

B Codice Python

```python
from flask import Flask, render_template, request, redirect
import sqlite3
from datetime import date, time, datetime

DB = 'sportdata.db'

def decrCapital(total, conn: sqlite3.Connection):
    conn.execute('''UPDATE club
                 SET capital=((SELECT capital FROM club)-?)''',(total,)
    )
    conn.commit()
    conn.close()

def incrCapital(total, conn: sqlite3.Connection):
    conn.execute('''UPDATE club
                 SET capital=((SELECT capital FROM club)-?)''',(total,)
    )
    conn.commit()
    conn.close()

def selecter(db: str, query: str, num='all'):
    conn = sqlite3.connect(db)
    conn.row_factory = sqlite3.Row
    if num == 'all':
        result = conn.execute(query).fetchall()
    if num == 'one':
        result = conn.execute(query).fetchone()
    if num == 'many':
        result = conn.execute(query).fetchmany()
    conn.close()
    return result

app = Flask(__name__)

@app.route('/')
def index():
    club = selecter(DB, 'SELECT * FROM club')
    return render_template('index.html', club=club)

@app.route('/atleti')
def atlteti():
    atleti = selecter(DB, '''SELECT * FROM athlete LEFT JOIN (
                        SELECT athlete_attendance.athlete, COUNT(*) AS
    pres
                        FROM athlete_attendance
                        WHERE (kind='allenamento'))
                        ON athlete = id_no
```

```
45                          ORDER BY(id_no)
46                           ''')
47      return render_template('atleti.html',atleti=atleti)
48
49  @app.route('/staff')
50  def staff():
51      return render_template('staff.html')
52
53  @app.route('/allenatori')
54  def allenatori():
55      allenatori = selecter(DB, 'SELECT * FROM coach')
56      return render_template('allenatori.html', allenatori=allenatori)
57
58  @app.route('/accompagnatori')
59  def accompagnatori():
60      accompagnatori = selecter(DB, 'SELECT * FROM companion')
61      return render_template('allenatori.html', accompagnatori=
    accompagnatori)
62
63  @app.route('/volontari')
64  def volontari():
65      volontari = selecter(DB,'SELECT * FROM volunteer')
66      return render_template('allenatori.html', volontari=volontari)
67
68  @app.route('/sponsor')
69  def sponsor():
70      sponsor = selecter(DB, '''SELECT p_iva, company_name, telephone_no,
      email, start_date, end_date, sponsor_value
71                                  FROM company INNER JOIN sponsorship ON
    company.p_iva = sponsorship.company''')
72      return render_template('sponsor.html',sponsor=sponsor)
73
74  @app.route('/pagamenti')
75  def pagamenti():
76      return render_template('pagamenti.html')
77
78  @app.route('/pagamenti_atleti', methods=('GET','POST'))
79  def pagamenti_atlteti():
80      conn = sqlite3.connect(DB)
81      conn.row_factory = sqlite3.Row
82      atleti = conn.execute('''SELECT club.club_name, club.id_no, athlete
    .id_no id, athlete.name, athlete.surname, payment_date, payment_time
    , total
83                                  FROM (club INNER JOIN athlete_payment ON
    club.id_no = club)
84                                 INNER JOIN athlete ON athlete = athlete.
    id_no''').fetchall()
85      if request.method == 'POST':
86          conn.execute('''INSERT INTO athlete_payment (club , athlete ,
```

```
    payment_date , payment_time, total)
87                      VALUES ((SELECT id_no FROM club), ?,?,?,?)''',
88                      (request.form['atleta'],request.form['data'],
    request.form['ora'], request.form['total']))
89          conn.commit()
90          decrCapital(request.form['total'],conn)
91      conn.close()
92      return render_template('pagamenti_atleti.html',atleti=atleti)
93
94
95      conn = sqlite3.connect(DB)
96      conn.row_factory = sqlite3.Row
97      atleti = conn.execute('''SELECT club.club_name, club.id_no, athlete
    .id_no id, athlete.name, athlete.surname, payment_date, payment_time
    , total
98                               FROM (club INNER JOIN athlete_payment ON
    club.id_no = club)
99                              INNER JOIN athlete ON athlete = athlete.
    id_no''').fetchall()
100     if request.method == 'POST':
101         print("E' stata fatta una richiesta POST!")
102         print(request.form['atleta'],request.form['total'],request.form
    ['data'],request.form['ora'])
103         conn.execute('''INSERT INTO athlete_payment (club , athlete ,
    payment_date , payment_time, total)
104                      VALUES ((SELECT id_no FROM club), ?,?,?,?)''',
105                      (request.form['atleta'],request.form['data'],
    request.form['ora'], request.form['total']))
106         conn.commit()
107         updateCapital(request.form['totale'],conn)
108     conn.close()
109     return render_template('pagamenti_atleti.html',atleti=atleti)
110
111 @app.route('/pagamenti_allenatori', methods=('GET','POST'))
112 def pagamenti_allenatori():
113     conn = sqlite3.connect(DB)
114     conn.row_factory = sqlite3.Row
115     allenatori = conn.execute('''SELECT club.club_name, club.id_no,
    coach.id_no id, coach.name, coach.surname, payment_date,
    payment_time, total
116                               FROM (club INNER JOIN coach_payment ON
    club.id_no = club)
117                               INNER JOIN coach ON coach = coach.id_no
    ''').fetchall()
118     if request.method == 'POST':
119         conn.execute('''INSERT INTO coach_payment (club , coach ,
    payment_date , payment_time, total)
120                      VALUES ((SELECT id_no FROM club), ?,?,?,?)''',
121                      (request.form['all'],request.form['data'], request
```

```
    .form['ora'], request.form['total']))
122         conn.commit()
123         decrCapital(request.form['total'],conn)
124     conn.close()
125     return render_template('pagamenti_allenatori.html', allenatori=
    allenatori)
126
127 @app.route('/pagamenti_sponsor', methods=('GET','POST'))
128 def pagamenti_sponsor():
129     conn = sqlite3.connect(DB)
130     conn.row_factory = sqlite3.Row
131     sponsor = conn.execute('''SELECT club.club_name, club.id_no, p_iva,
    company.company_name, payment_date, payment_time, total
132                               FROM (club INNER JOIN sponsor_payment ON
    id_no = club)
133                               INNER JOIN company ON company = p_iva''')
    .fetchall()
134     if request.method == 'POST':
135         #print("E' stata fatta una richiesta POST!")
136         #print(request.form['all'],request.form['total'],request.form['
    data'],request.form['ora'])
137         conn.execute('''INSERT INTO sponsor_payment (club , company ,
    payment_date , payment_time, total)
138                     VALUES ((SELECT id_no FROM club), ?,?,?,?)''',
139                     (request.form['az'],request.form['data'], request.
    form['ora'], request.form['total']))
140         conn.commit()
141         incrCapital(request.form['total'],conn)
142     conn.close()
143     return render_template('pagamenti_sponsor.html',sponsor=sponsor)
144
145 @app.route('/medico')
146 def medico():
147     atleti_inf = selecter(DB,'SELECT * FROM injured_athletes')
148     return render_template('medico.html', atleti=atleti_inf)
149
150 @app.route('/eventi')
151 def eventi():
152     eventi = selecter(DB, 'SELECT * FROM sport_event')
153     return render_template('eventi.html', eventi=eventi)
154
155 @app.route('/certificati', methods=('GET','POST'))
156 def certificati():
157     valid = selecter(DB,'''SELECT athlete.id_no, athlete.name, athlete.
    surname, date_of_emission
158                             FROM athlete INNER JOIN medical_cert ON
    athlete.id_no = medical_cert.id_no
159                             WHERE (medical_cert.date_of_emission > DATE
    ('now','-1 year'))
```

```
                            ORDER BY (date_of_emission)''')
    scad = selecter(DB,'''SELECT athlete.id_no, name, surname,
    date_of_emission
                            FROM athlete INNER JOIN medical_cert ON
    athlete.id_no = medical_cert.id_no
                            WHERE (medical_cert.date_of_emission <=
    DATE('now','-1 year'))
                            ORDER BY (date_of_emission);''')
    if request.method=='POST':
        id_no=request.form['id']
        date=request.form['date']
        conn = sqlite3.connect(DB)
        conn.row_factory = sqlite3.Row
        conn.execute('''REPLACE INTO medical_cert (id_no,
    date_of_emission)
                            VALUES (?,?)''',
                            (id_no,date))
        conn.commit()
    return render_template('certificati.html', valid=valid, scad=scad)

@app.route('/test')
def test():
    test = selecter(DB,'''SELECT athlete.id_no, name, surname, jump,
    sprint, endurance
                            FROM athlete LEFT OUTER JOIN physical_test
    ON athlete.id_no = physical_test.id_no
                            ORDER BY athlete.id_no;''')
    bestj = selecter(DB, '''SELECT athlete.id_no, name, surname, jump,
    sprint, endurance
                            FROM athlete LEFT OUTER JOIN physical_test
    ON athlete.id_no = physical_test.id_no
                            WHERE (jump >= (SELECT MAX(jump) FROM
    physical_test));''', num='one')
    bests = selecter(DB, '''SELECT athlete.id_no, name, surname, jump,
    sprint, endurance
                            FROM athlete LEFT OUTER JOIN physical_test
    ON athlete.id_no = physical_test.id_no
                            WHERE (sprint <= (SELECT MAX(sprint) FROM
    physical_test));''', num='one')
    beste = selecter(DB, '''SELECT athlete.id_no, name, surname, jump,
    sprint, endurance
                            FROM athlete LEFT OUTER JOIN physical_test
    ON athlete.id_no = physical_test.id_no
                            WHERE (endurance >= (SELECT MAX(endurance)
    FROM physical_test));''', num='one')
    return render_template('test.html', test=test, bestj=bestj, bests=
    bests, beste=beste)

@app.route('/aggiungi', methods=('GET','POST'))
```

```
193 def aggiungi():
194     if request.method == 'POST':
195         print(request.form)
196         id_no=request.form['id_no']
197         name=request.form['name']
198         surname=request.form['surname']
199         tel=request.form['tel']
200         email=request.form['email']
201         addr=request.form['addr']
202         cat=request.form['cat']
203         h=request.form['h']
204         w=request.form['w']
205         pos=request.form['pos']
206         conn = sqlite3.connect(DB)
207         conn.row_factory = sqlite3.Row
208         conn.execute('''INSERT INTO athlete (id_no,name,surname,
    telephone_no,email,category,height,weight,position,adress)
209                         VALUES (?,?,?,?,?,?,?,?,?,?)''',
210                         (id_no,name,surname,tel,email,cat,h,w,pos,addr)
211                         )
212         conn.commit()
213         conn.close()
214         return redirect('/atleti')
215     return render_template('aggiungi.html')
216
217 @app.route('/<int:id_no>/chiudiInf', methods=('POST',))
218 def chiudiInf(id_no):
219     conn=sqlite3.connect(DB)
220     conn.row_factory=sqlite3.Row
221     conn.execute('''UPDATE athlete
222                 SET injured = 0
223                 WHERE (id_no=?)''',(id_no,))
224     conn.commit()
225     conn.close()
226     return redirect('/medico')
227
228 @app.route('/<int:id_no>/apriInf', methods=('POST',))
229 def apriInf(id_no):
230     conn=sqlite3.connect(DB)
231     conn.row_factory=sqlite3.Row
232     conn.execute('''UPDATE athlete
233                 SET injured = 1
234                 WHERE (id_no=?)''',(id_no,))
235     conn.commit()
236     conn.close()
237     return redirect('/atleti')
238
239 @app.route('/contratti', methods=('GET','POST'))
240 def contratti():
```

```
241     conn = sqlite3.connect(DB)
242     conn.row_factory = sqlite3.Row
243     atleti=conn.execute('''SELECT protocol_no, id_no, name, surname,
    starting_date, expiry_date, salary
244                         FROM contract INNER JOIN (athlete a INNER JOIN
    stipulate_athlete s ON a.id_no=s.athlete) ON protocol_no = contract
245                         ORDER BY(expiry_date)''').fetchall()
246     allenatori=conn.execute('''SELECT protocol_no, id_no, name, surname
    , starting_date, expiry_date, salary
247                             FROM contract INNER JOIN (coach c INNER
    JOIN stipulate_coach s ON c.id_no=s.coach) ON protocol_no = contract
248                             ORDER BY(expiry_date)''').fetchall()
249     if request.method == 'POST':
250         id_no=request.form['id']
251         num=request.form['num']
252         start=request.form['start_date']
253         end=request.form['end_date']
254         valore=request.form['valore']
255         conn = sqlite3.connect(DB)
256         conn.row_factory = sqlite3.Row
257         conn.execute('''INSERT INTO contract (protocol_no,starting_date
    ,expiry_date,salary)
258                             VALUES (?,?,?,?)''',
259                             (num,start,end,valore))
260         if request.form['kind'] == 'athlete':
261             conn.execute('''INSERT INTO stipulate_athlete (contract,
    athlete)
262                             VALUES (?,?)''',
263                             (num,id_no))
264         else:
265             conn.execute('''INSERT INTO stipulate_coach (contract,coach
    )
266                             VALUES (?,?)''',
267                             (num,id_no))
268         conn.commit()
269     conn.close()
270     return render_template('contratti.html',atleti=atleti,allenatori=
    allenatori)
```