Documentazione Finale Progetto

Sistema Gestionale Campionato Rugby

1. Titolo e Gruppo

- Titolo Progetto: Sistema Gestionale Campionato Rugby
- Nome Gruppo: Rugby Trento
- Corso: Programmazione Avanzata
- Anno Accademico: 2025/2026

2. Data e Versione

- Versione: 1.0
- Data: 17 Dicembre 2025

3. Componenti del Gruppo

1. Tommaso Santus
    a. Matricola: 226666
    b. Email: tommaso.santus@studenti.unitn.it
2. Francesco Martella
    a. Matricola: 234474
    b. Email: francesco.martella@studenti.unitn.it

4. Indice

Sommario

5. Descrizione del Progetto

Il Sistema Gestionale Campionato Rugby è un'applicazione C++11 completa per la gestione di campionati di rugby. Il sistema permette di amministrare stagioni complete con squadre, giocatori, staff tecnico e partite, fornendo strumenti avanzati per analisi statistiche e generazione di classifiche.

Obiettivi Principali:

- Gestire multiple stagioni sportive con persistenza su file CSV
- Registrare squadre con roster completo (giocatori e staff)
- Tracciare partite con statistiche dettagliate e aggiornamento automatico classifiche
- Generare report statistici avanzati utilizzando algoritmi STL
- Implementare salvataggio parallelo dei dati mediante threading

Ambito Applicativo:

Il sistema è progettato per federazioni sportive, società di rugby e organizzatori di campionati che necessitano di uno strumento per:

- Monitorare performance individuali e di squadra
- Analizzare trend statistici (top performer, distribuzione ruoli, medie)
- Gestire calendari e risultati delle partite

Caratteristiche Distintive:

- Architettura Object-Oriented con ereditarietà (Persona->Giocatore/Staff)
- Move Semantics per ottimizzazione trasferimento dati
- Smart Pointers (*std::unique_ptr*) per gestione automatica memoria
- Algoritmi STL per operazioni ad alta efficienza
- Template Metaprogramming per calcoli generici
- Threading per salvataggio parallelo CSV

6. Elenco e Descrizione dei Requisiti

Requisiti Funzionali

RF1 – Gestione Stagioni

- Creazione di nuove stagioni sportive identificate per anno
- Caricamento stagioni esistenti da database CSV
- Modifica dati stagione (aggiunta squadre, partite)
- Visualizzazione completa di tutte le informazioni

RF2 – Gestione Squadre

- Creazione squadra con dati anagrafici (nome, indirizzo, ID univoco)
- Aggiunta/rimozione giocatori al roster
- Aggiunta/rimozione membri staff tecnico
- Tracking statistiche aggregate (mete, placcaggi, possesso palla)
- Gestione con move semantics

RF3 – Gestione Giocatori

- Registrazione giocatori con dati personali (nome, cognome, età, ruolo)
- Tracciamento statistiche individuali (14 metriche)
- Supporto 15 ruoli standard rugby
- Gestione con move semantics

RF4 – Gestione Staff Tecnico

- Registrazione membri staff (allenatore, aiutante, direttore sportivo e segreteria)
- Associazione staff a squadre specifiche
- Gestione con move semantics

RF5 – Gestione Partite

- Registrazione partita con data, squadre coinvolte, risultato
- Calcolo automatico punti classifica secondo regolamento rugby:
    - 4 punti vittoria, 2 punti pareggio e 0 punti sconfitta
    - +1 punto bonus se sconfitta entro 7 punti
    - +1 punto bonus se segnate più di 4 mete
- Tracking statistiche partita (possesso, cartellini, mete)

RF6 – Classifiche e Statistiche

- Generazione classifica completa ordinata per punteggio
- Calcolo Top N squadre (ottimizzato con *std::partial_sort*)
- Individuazione squadra mediana (*std::nth_element*)
- Calcolo medie, deviazioni standard, min/max statistiche
- Identificazione top performer (Meta Man, Miglior Placcatore e Ball Carrier)

RF7 – Persistenza Dati

- Salvataggio automatico su 5 file CSV (stagioni, squadre, giocatori, staff, partite)
- Caricamento all'avvio
- Salvataggio parallelo con threading per performance

RF8 – Interazione Utente

- Menu testuale interattivo gerarchico
- Validazione input con gestione eccezioni
- Messaggi di feedback operazioni
- Visualizzazione report formattati

Requisiti Non Funzionali

RNF1 – Efficienza Algoritmica

- Utilizzo algoritmi STL con complessità ottimale:
  - O(n) per ricerche lineari (*std::max_element* , *std::count_if*)
  - O(n log n) per ordinamenti completi (*std::sort*)
  - O(n log k) per ordinamenti parziali (*std::partial_sort*)
  - O(n) per mediana (*std::nth_element*)
- Pre-allocazione memoria con *reserve()* per vector

RNF2 – Gestione Memoria

- Smart Pointers (*std::unique_ptr*) per ownership automatico
- Move semantics per ridurre copie inutili
- Assenza memory leak (verificato)

RNF3 – Manutentibilità

- Separazione responsabilità (SRP): ogni classe ha un compito specifico
- Template functions per codice riutilizzabile
- Naming convention chiaro e consistente
- Codice commentato seguendo documentzione Doxygen

RNF4 – Scalabilità

- Supporto stagioni multiple simultanee
- Gestione efficiente dataset grandi (300+ giocatori, 60+ partite)
- Threading per operazioni I/O parallele

RNF5 – Robustezza

- Gestione eccezioni completa (*try-catch-throw*)
- Validazione input utente
- Controllo esistenza file prima apertura
- Fallback su errori non critici

RNF6 – Standard C++11

- Conformità completa allo standard C++11
- Utilizzo feature moderne:
    - Lambda expressions con capture
    - Rvalue references (*&&*)
    - *Std::move, std::forward*
    - *Dafault, delete* per metodi speciali
    - *Noexcept* per garanzie no-throw

7. Attività svolte per la Realizzazione

Fase di Analisi

- Studio requisiti dominio rugby (regole punteggi, ruoli, statistiche)
- Definizione entità principali (Persona, Giocatore, Staff, Squadra, Partita, Stagione)
- Progettazione gerarchia classi con ereditarietà
- Identificazione relazioni tra entità

Fase di Progettazione

- Diagramma UML delle classi con cardinalità
- Schema database CSV

Fase di Implementazione

- Iterazione 1: Classi Base (Persona, Giocatore, Staff)
- Iterazione 2: Contenitori (Squadra e Partita con vector, Stagione con unique_ptr)
- Iterazione 3: Gestionale e persistenza CSV
- Iterazione 4: Algoritmi statistiche STL
- Iterazione 5: Move semantics e ottimizzazioni
- Iterazione 6: Threading salvataggio parallelo

Fase di Test

- Test unitari funzionalità di base
- Test di integrazione con dataset reale
- Verifica memory leak
- Test concorrenza threading

Metodologia di Sviluppo

- Version Control: Git/GitHub repository condiviso
- Branching: Sviluppo su branch *main* unico
- Collaborazione: Pair programming remoto
- Build System: Makefile per compilazione automatizzata

8. Attività di Implementazione

Object-Oriented Design (OOD)

Classe *Persona* (Astratta)

- Descrizione: Classe base astratta per entità con dati anagrafici
- Attributi:
  - *std::string nome*: Nome della persona
  - *std::string cognome*: Cognome della persona
  - *int eta*: Età in anni
- Metodi:
  - *Persona(nome, cognome, eta)*: Costruttore
  - *Virtual ~Persona()*: Distruttore
  - *std::string getNome()*: Ritorna nome
  - *std::string getCognome()*: Ritorna cognome
  - *int getEta() const*: Ritorna età
  - *Friend operator<<*: Stampa dati (implementato nelle derivate)
- Relazioni: Classe base per *Giocatore* e *Staff*

Classe *Giocatore* (Deriva da Persona)

- Descrizione: Rappresenta un giocatore con statistiche di gioco
- Attributi aggiuntivi:
  - *Int id*: Identificatore univoco
  - *std::string ruolo*: Ruolo in campo (15 possibili)
  - *int placcaggi, metriCorsi, mete, calciPiazzati, falliCommessi, offload, minutiGiocati, partiteGiocate*: Statistiche
- Metodi Principali:
  - Getter/Setter per tutte le statistiche
  - *void aggiornaStatistiche(...)*: Aggiorna valori dopo partita
- Relazioni: Contenuto in *Squadra* (composizione 1:molti)

Classe *Staff* (Deriva da Persona)

- Descrizione: Membro staff tecnico
- Attributi Aggiuntivi:
  - *RuoloStaff ruolo*: Enum (ALLENATORE, AIUTANTE_ALLENATORE, DS, SEGRETERIA)
- Metodi:
  - *getRuolo() const*: Ritorna ruolo enum
  - *static std::string ruoloToString(RuoloStaff)*: Conversione per output
- Relazioni: Contenuto in *Squadra* (composizione 1:molti)

Classe *Squadra*

- Descrizione: Entità principale con roster completo

- Attributi:
    - *Int id*: Identificatore univoco
    - *Std::string nome, indirizzo*: Dati anagrafici
    - *Std::vector<Giocatore> giocatori*: Roster giocatori
    - *Std::vector<Staff> staffTecnico*: membri dello staff
    - *Int punteggioClassifica, metetotali*: Dati classifica
    - *Int possessoPalla, placcaggiTotali, metriGuadagnatiTotali, …*: 10+ Statistiche
- Metodi Principali:
    - *addGiocatore(const Giocatore&)*: Aggiunge giocatore (copia)
    - *addStaff(const Staff&)*: Aggiunge Staff (copia)
    - *addStaff(Staff&&)*: Aggiunge Staff (move)
    - *aggiornaStatstiche()*: Ricalcola totali da giocatori
    - *reserve()* nel costruttore per pre-allocazione
- Relazioni
    - Contenuta in *Stagione* (1:N)
    - Riferita da *Partita* (2 per partita)

Classe *Partita*

- Descrizione: Rappresenta una partita disputata
- Attributi:
    - *Int id, data*: Identificatore (data formato YYYYMMDD)
    - *Squadra& locali, & ospiti*: Reference a squadre coinvolte
    - *Int ptLocali, ptOspiti, meteLocali, meteOspiti*: Risultato
    - *Int cartellinoRosso/Giallo Loc/Osp*: Disciplina
    - *Double possessoLoc/Osp*: Percentile possesso palla
- Metodi:
    - *setRisultato(ptLoc, ptOsp)*: Imposta risultato
    - *calcolaPuntiClassifica()*: Applica regolamento rugby
- Relazioni: Contenuta in *Stagione* (1:N), riferisce 2 *Squadra*

Classe *Stagione*

- Descrizione: Container stagione completa
- Attributi:
    - *Int anno*: Anno stagione
    - *Std::vector<std::unique_ptr<Squadra>> squadre*: Ownership esclusivo
    - *Std::vector<Partita> partite*: Calendario
- Metodi Principali:
    - *addSquadra(std::unique_ptr<Squadra>)*: Trasferisce ownership
    - *classificaSquadre() const*: STL sort con lambda
    - *topSquadre(n) const*: STL partial_sort
    - *mediaPunteggioSquadre() const*: STL accumulate
    - Template *calcoloMedia<T>(getter)*: Pointer-to-member function
- Copy/Move Semantics:

    - Copy constructor: deep copy con *new* per unique_ptr
    - Move constructor: trasferisce vector con *std::move*
- Relazioni: Contenitore principale, gestito da *Gestionale*

Classe *Gestionale*

- Descrizione: Controller applicazione, gestisce I/O e business logic
- Attributi:
    - *Std::vector<std::unique_ptr<Stagione>> stagioni*: Tutte le stagioni
    - *Std::string path**: Persorsi file CSV (5 path)
- Metodi Principali:
    - *creaStagione()*: Wizard creazione interattiva
    - *selezionaStagione()*: Carica da CSV
    - *modificaStagione(Stagione&)* : Menu modifica con 6 opzioni
    - *salvaParallel(Stagione&)*:Threading con 3 std::thread
    - *fetchSquadre/Giocatori/Staff/Partite()*: Parsing CSV
    - *salvaSquadra/Giocatori/Staff/Partite()*: Scrittura CSV
- Relazioni:
    - Gestisce (1:N) *Stagione*
        - Ownership esclusivo tramite vector di unique_ptr
    - Utilizza *CSVManager*
        - Dipendenza per tutte le operazioni I/O
    - Coordina *Squadra*, *Giocatore*, *Staff* e *Partita*
        - Non possiede direttamente
    - Delega Statistiche

Classe *CSVManager* (Utility Statica Header-Only)

- Descrizione: Libreria Header-Only per gestione I/O file CSV con pattern RAII
- Struttura Interna:
    - Classe FileGuard
        - Descrizione: Wrapper per gestione automatica apertura/chiusura file
        - Attributi privati:
            - std::fstream& file: Reference al file gestito
        - Metodi:
            - *FileGuard(std::fstream& f)*: Costruttore che acquisisce ownership
            - *~FileGuard( )*: Distruttore
            - *FileGuard(const FileGuard&) = delete*: Previene copie
            - *FileGuard& operator=(const FileGuard&) = delete*: Previene assegnazioni
        - Pattern: Resource Acquisition Is Initialization (RAII)
        - Garanzie: Chiusura file anche in caso di eccezioni
    - Namespace CSVHelper
        - Descrizione: Funzioni template generiche per operazioni CSV
        - Template Functions:

        - *Template<typename T> std::vector<T> caricaRighe(const std::string& path)*
            - Legge file CSV e restituisce vector di tipo T
        - *Template<typename T> void salvaRighe(const std::string& path, const std::vector<T>& dati)*
            - Scrive vector di tipo T su file CSV
    - Relazioni:
        - Usata da *Gestionale* per tutte le operazioni I/O
        - FileGuard incapsulato in CSVHelper
        - Nessuna dipendenza da altre classi del progetto

Classe *Statistiche* (Utility Statica)

- Descrizione: Algoritmi STL per analisi dati
- Metodi Principali (tutti *static*):
    - *getMigliorPlaccatore(Squadra&)*: *std::max_element* + lambda
    - *getMetaMan(Squadra&)*: *std::max_element*
    - *contaGiocatoriPerRuolo()*: *std::count_if* + lambda
    - *giocatoriConMeteSopra(soglia)* : *std::copy_if* + lambda
    - *classificaCompleta(Stagione&)*: *std::sort* + lambda comparator
    - *topNSquadre(n)* : *std::partial_sort* (O(n log k))
    - *squadraMediana()*: *std::nth_element* (O(n))
    - *mediaMetrica(getter)*: *std::accumulate* + pointer-to-member
    - *squadreBilanciate()*: Filtro su medie multiple
    - *stampaReportStagione()*: Output formattato completo
- Relazioni:
    - Analizza *Stagione*
        - Accesso read-only tramite const reference
    - Elabora *Squadra*
        - Estrae metriche aggregate
    - Esamina *Giocatore*
        - Accesso tramite roster della squadra
    - Invocata da *Gestionale*

Descrizione Use Case

UC1: Creazione Nuova Stagione

- Attore: Utente amministratore
- Precondizioni: Applicazione avviata, menu principale visualizzato
- Flusso Principale
    - Utente seleziona “Crea stagione” (Opzione 2)
    - Sistema richiede anno stagione
    - Utente inserisce anno
    - Sistema verifica univocità anno con *trovaStagione()*
    - Sistema crea *std::unique_ptr<Stagione>* e lo aggiunge a *stagioni*

    - Sistema invoca *modificaStagione()* per configurazione iniziale
- Postcondizioni: Stagione creata in memoria, pronta per aggiungere squadre
- Oggetti Utilizzati:
    - *Gestionale::creaStagione()* – business logic
    - *Stagione(anno)* – costruttore
    - *Gestionale::trovaStagione()* – validazione con *std::find_if* + lambd

UC2: Aggiunta Squadra a Stagione

- Attore: Utente Amministratore
- Precondizioni: Stagione selezionata, menu modifica visualizzato
- Flusso Principale:
    - Utente seleziona "Aggiungi squadra" (opzione 1 del menu modifica)
    - Sistema invoca *Gestionale::aggiungiSquadra()*
    - Sistema richiede nome e indirizzo squadra
    - Sistema genera ID univoco con *getMaxSquadraId() + 1*
    - Sistema crea *std::unique_ptr<Squadra>*
    - Sistema richiede se aggiungere staff tecnico
    - Per ogni membro staff: input nome, cognome, età, ruolo -> *addStaff(Staff&&)* (move)
    - Sistema richiede se aggiungere giocatori
    - Per ogni giocatore: input dato -> addGiocatore(Giocatore)
    - Sistema trasferisce ownership con *Stagione::addSquadra(std::move(squadraPtr))*
- Postcondizioni: Squadra aggiunta con roster completo
- Oggetti Utilizzati:
    - Gestionale::aggiungiSquadra() - creazione squadra
    - Squadra::addStaff(Staff&&) – move semantics
    - Squadra::addGiocatore() – composizione
    - Stagione::addSquadra() – trasferimento ownership

UC3: Registrazione Partita

- Attore: U.A.
- Precondizioni: Stagione con almeno 2 squadre
- Flusso Principale:
    - Utente seleziona "Aggiungi partita" (opzione 2 nel menu modifica)
    - Sistema visualizza elenco squadre disponibili
    - Utente inserisce data (YYYMMDD), ID squadra locale, ID squadra ospite
    - Sistema recupera puntatori con *Stagione::trovaSquadrePerId()*
    - Utente inserisce punti locali e ospiti
    - Sistema crea *Partita(id, data, locali, ospiti)*
    - Sistema invoca *Partita::setRisultato(ptLoc, ptOsp)*
    - Sistema calcola punti classifica secondo regolamento
    - Sistema aggiorna *Squadra::aggiungiPuntiClassifica()*
    - Sistema aggiunge partita con *Stagione::addPartita()*
- Postcondizioni: Partita registrata, classifiche aggiornate

- Oggetti Utilizzati:
    - *Gestionale::aggiungiPartita()* – wizard input
    - *Partita::setRisultato()* – calcolo punteggi
    - *Squadra::aggiungiPuntiClassifica()* – aggiornamento stato

UC4: Statistiche Avanzate STL

- Attore: Utente Analista
- Precondizioni: Stagione con dati completi
- Flusso Principale:
    - Utente seleziona "Statistiche Avanzate" (opzione 4 del menu modifica)
    - Sistema presenta sottomenu:
        - Statistiche per Squadra (giocatori)
        - Report Completo Stagione
    - Se opzione 1:
        - Per ogni squadra: *Statistiche::stampaReportSquadra()*
        - Miglior placcatore *std::max_element(giocatori, lambda)* con capture
        - Meta Man: *std::max_elemnt* su mete
        - Media mete: *std::accumulate(giocatori, 0, lambda)/size*
        - Distribuzione ruoli: *std::count_if* per ruolo standard
        - Giocatori con 5+ mete: *std::copy_if(giocatori, risultato, lambda)*
    - Se opzione 2:
        - Sistema invoca *Statistiche::stampaReportStagione()*
        - Classifica completa: *std::sort* + lambda
        - Podio: *std::partial_sort* primi 3
        - Squadra mediana: *std::nth_element* a metà vector
        - Media/dev.std: *std::accumulate* + calcolo varianza
        - Squadre bilanciate: filtro custom su medie multiple
        - Efficienza: calcolo ratio + *std::sort* su coppie
- Postcondizioni: Report dettagliati visualizzati
- Oggetti Utilizzati:
    - *Statistiche::getMigliorPlaccatore()* – max_element
    - *Statistiche::contaGiocatoriPerRuolo()* – count_if
    - *Statistiche::giocatoriConMeteSopra()* – copy_if
    - *Statistiche::classificaCompeta()* – sort
    - *Statistiche::topNSquadre()* – partial_sort
    - *Statistiche::squadraMediana()* – nth_element
    - *Statistiche::mediaMetrica()* – accumulate + pointer-to-member

UC5: Salvataggio Dati (Parallelo)

- Attore: Sistema(Automatico)
- Precondizioni: Modifiche effettuate a stagione
- Flusso Principale:
    - Utente conferma uscita da menu modifica
    - Sistema invoca Gestionale::salvaParallel(stagione)

    - Sistema crea 5 thread paralleli:
        - *Std::thread t1([this, &stagioine]() {salvaStagioni(stagione); })*
        - *Std::thread t2([this, &stagioine]() {salvaSquadra(stagione); })*
        - *Std::thread t3([this, &stagioine]() {salvaPartite(stagione); }*
        - *Std::thread t4([this, &stagioine]() {salvaGiocatori(stagione); })*
        - *Std::thread t5([this, &stagioine]() {salvaStaff(stagione); })*
    - Thread t1:
        - Carica anni esistenti con *CSVManager::CSVHelper::caricaRighe<int>()*
        - Aggiunge anno corrente se mancante
        - Salva con *CSVManager::CSVHelper::salvaRighe()*
    - Thread t2:
        - Per ogni squadra: serializza in *SquadraData* struct
        - Converte in CSV con *toCSV()* method
        - Scrive con *CSVManager::FileGuard* RAII
    - Thread t3:
        - Serializza partite in *PartitaData*
        - Salva in formato CSV
    - Thread t4:
        - Serializza giocatori in *GiocatoriData*
        - Salva in formato CSV
    - Thread t5:
        - Serializza staff in *StaffData*
        - Salva in formato CSV
    - Sistema attende completamento: *t1.join(); t2.join(); t3.join(); t4.join(); t5.join();*
- Postcondizioni: Tutti i file CSV aggionrati su disco
- Oggetti Utilizzati:
    - *Gestionale::salvaParallel()* – coordinatore threading
    - *Std::thread* – parallelismo
    - *CSVManager::CSVHelper::FileGuard* – RAII file management
    - *CSVManager::CSVHelper::salvaRighe<T>()* – template generico

UC6: Caricamento Stagione Esistente

- Attore: Utente Amministratore
- Precondizioni: File CSV esistenti
- Flusso Principale:
    - Utente seleziona “Carica Stagione” (opzione 1)
    - Sistema invoca *Gestionale::fetchStagioni(pathStagioni)*
    - Sistema legge *stagioni.csv* e visualizza elenco numerato
    - Utente seleziona numero stagioine
    - Sistema recupera anno con *recuperaStagione(path, numeroRiga)*
    - Sistema crea *std::unique_ptr<Stagione>(new Stagione(anno))*
    - Sistema carica squadre:
        - *fetchSquadre(*stagione)* – parsing CSV con splitCSVLine()
        - Per ogni riga matching anno: crea *Squadra* e popola attributi

        - Aggiunge con *stagione->addSquadra(std::move(squadraPtr))*
    - Sistema carica dipendenze per ogni squadra:
        - *fetchGiocatori(*squadra)* – parsing giocatori.csv
        - *fetchStaff(*squadra)* – parsing staff.csv con move semantics
    - Sistema carica partite:
        - *fetchPartite(*stagione)* – parsing partite.csv
        - Recupera reference squadre con *trovaSquadraPerId()*
        - Ricalcola punteggi classifica
    - Sistema invoca *modificaStagione()* per gestione interattiva
- Postcondizioni: Stagione competamente ricostruita in memoria
- Oggetti Utilizzati:
    - *Gestionale::fetchStagioni/Squadre/Giocatori/Staff/Partite()*
    - *Gestionale::splitCSVLine()* – tokenizzazione
    - *Stagione::trovaSquadraPerId()* – ricerca con std::find_if

Database CSV – Struttura Dettagliata

File: *stagioni.csv*

Campo: anno

Tipo: integer (YYYY)

Chiave: primaria

Descrizione: Anno identificativo stagione sportiva

Esempio:

anno

2023

2024

File: *squadre.csv*

Campi:

1. Squadra_id: integer, chiave primaria
2. Stagione_anno: integer, foreign key -> stagioni.anno
3. Nome: string
4. Indirizzo: string
5. Possesso_palla: int, % medio possesso
6. Territorio: double, % controllo territorio
7. Placcaggi_totali: integer
8. Metri_guadagnati_totali: integer, metri avanzamento
9. Mete_totali: integer
10. Falli_totali: integer
11. Mischie_vinte: integer

12. Mischie_perse: integer
13. Touche_vinte: integer
14. Touche_perse: integer
15. Punteggio_classifica: integer

Esempio:

11,2024,RugbyTrento,ViaRovigo15Trento,54,56.2,668,3080,23,72,47,11,41,14,19

File: *giocatori.csv*

Campi:

1. Giocatore_id: integer, chiave primaria
2. Squadra_id: integer, foreign key -> squadre.squadra_id
3. Nome: string
4. Cognome: string
5. Eta: integer
6. Ruolo: Enum
7. Placcaggi: integer
8. Metri_corsi: integer
9. Mete: integer
10. Calci_piazzati: integer
11. Falli commessi: integer
12. Offload: integer
13. Minuti_giocati: integer
14. Partite_giocate: integer

Esempio

1,1,Giovanni,Rossi,28,Pilone,45,95,1,0,8,3,720,9

File: *staff.csv*

Campi:

1. Staff_id: integer, chiave primaria
2. Squadra_id: integer, foreign key -> squadre.squadra_id
3. Nome: string
4. Cognome: string
5. Eta: integer
6. Ruolo: Enum

Esempio

1,1,Marco,Rossi,45,ALLENATORE

File: *partite.csv*

Campi:

1. Partita_id: integer, chiave primaria
2. Stagione_anno: integer, foreign key ->Stagioni.anno
3. Data: integer
4. Id_locali: integer, foreign key -> squadre.squadra_id
5. Id_ospiti: integer, foreign key -> squadre.squadra_id
6. Pt_locali: integer
7. Pt_ospiti: integer
8. Mete_locali: integer
9. Mete_ospiti: integer
10. Cartellino_rosso_loc: integer
11. Cartellino_rosso_osp: integer
12. Cartellino_giallo_loc: integer
13. Cartellnino_giallo_osp: integer
14. Possesso_loc: double
15. Possesso_osp

Esempio:

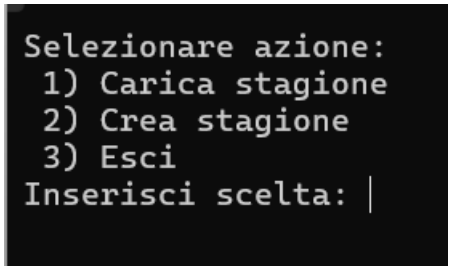
31,2024,20240908,11,12,30,24,4,3,0,0,1,2,53.8,46.2

Vincoli di Integrità Referenziale:

- *Squadre.stagione_anno -> stagioni.anno*
- *Giocatori.squadra_id -> squadre.squadra_id*
- *Staff.squadra_id -> squadre.squadra_id*
- *Partite.stagione_anno -> stagioni.anno*
- *Partite.id_locali/id_ospiti -> squadre.squadra_id*

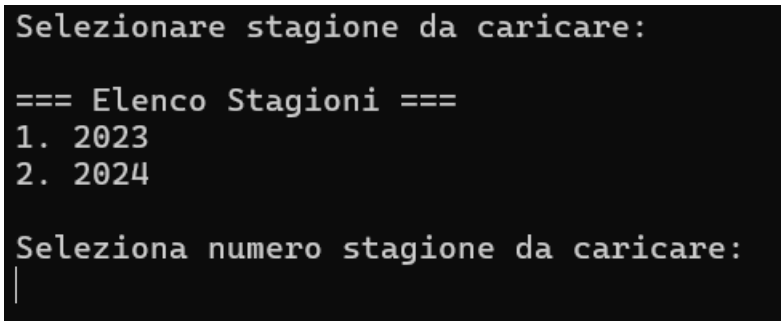
Modalità di Interazione

Menu Principale:



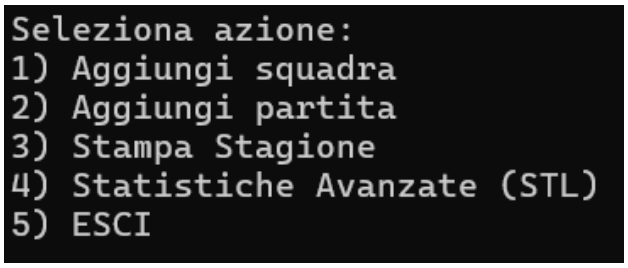
Appena avviato il programma, ti chiede se si vuole lavorare su una stagione già esistente o se si vuole crearne una nuova.

Menu Carica Stagione:



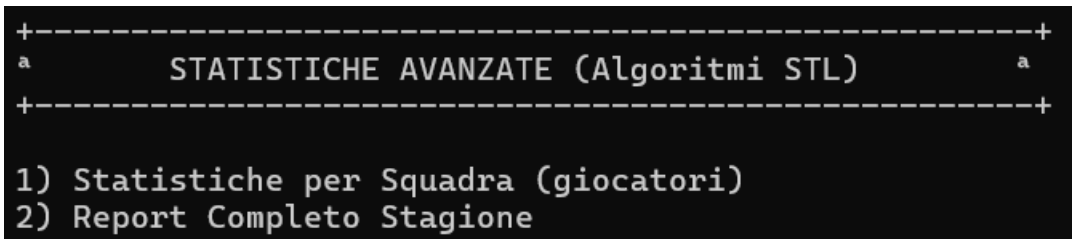
Ti propone l'elenco delle stagioni che si possono recuperare dal database.

Menu Seleziona Operazioni:



Elenco di operazioni che si possono eseguire con la stagione selezionata.

Menu Statistiche Avanzate:



Elenco delle statistiche proposte, per squadra o per stagione.

Menu Secondari:

Eventuali menu non riportati, sono quelli di inserimento dati per Stagione, Squadra, Partita, Giocatore e Staff. Chiedono all'utente di scrivere in input le informazioni necessarie alla creazione degli oggetti.

Menu Scelta Azione:

Viene richiesto se si vuole continuare a lavorare sulla stagione. Presenti menu simili per l'aggiunta di Squadre, Giocatori, Staff e Partite, con l'opzione di aggiungerne ancora.

9. Indicazioni Aggiuntive

Componenti Interni Sviluppati

CSVManager.h:

- Tipo: Header-Only library
- Contenuto:
    - Classe FileGuard
    - Namespace CSVHelper con template functions
- Utilizzo: Gestione completa persistenza CSV
- Pattern: RAII, Template Metaprogramming

Librerie Standard C++11 Utilizzate:

- Input/Output
    - <iostream> - I/O stream console
    - <fstream> - File stream per CSV
    - <iomanip> - Formattazione output (setw, setprecision)
- Contenitori
    - <vector> - Contenitore dinamico principale
    - <map> - Distribuzioni e lookup
    - <string> - Gestione stringhe
- Memory Management
    - <memory> - Smart pointers
    - <utility> - move (e forward)
- Algoritmi e Numerici
    - <algorithm> - Algoritmi STL
    - <numeric> - accumulate per calcoli aggregati
    - <cmath> - per deviazione standard
- Concorrenza
    - <thread> - nel salvataggio parallelo
    - <mutex> - Sincronizzazione (dichiarato ma non utilizzzato a causa di bug)
- Gestione Errori
    - <stdexcept> - Eccezioni standard

NOTA: Non sono state utilizzate librerie esterne di terze parti. Tutto il codice si basa esclusivamente sulla Standard Library C++11.

Algoritmi Utilizzati

1. Ordinamento Completo (std::sort)
2. Ordinamento Parziale (std::partial_sort)
3. Selezione N-esimo (std::nth_elementh)
4. Ricerca Massimo (std::max_element)
5. Conteggio Condizionale (std::count_if)
6. Copia Condizionale (std::copy_if)
7. Accumulo (std::accumulate)
8. Trasformazione (std::trasform)
9. Rimozione Condizionale (std::remove_if)
10. Ricerca Lineare con Predicato (std::find_if)

10. Bibliografia

Testi di Riferimento

1. **Bjarne Stroustrup**, *The C++ Programming Language (4th Edition)*, Addison-Wesley, 2013
2. **Scott Meyers**, *Effective Modern C++ (C++11/14)*, O'Reilly Media, 2014
3. **Nicolai M. Josuttis**, *The C++ Standard Library (2nd Edition)*, Addison-Wesley, 2012

Documentazione Online

1. **cppreference.com** - C++ Reference Documentation
2. **cplusplus.com** - C++ Resources Network
3. **ISO C++** - Standard C++ Foundation
4. **Herb Sutter**, *GotW (Guru of the Week)* - Exception Safety
5. **Stack Overflow** - Q&A Community (STL algorithms, move semantics, threading)

Repository Progetto

https://github.com/tommso00/PrAva-RugbyTrento