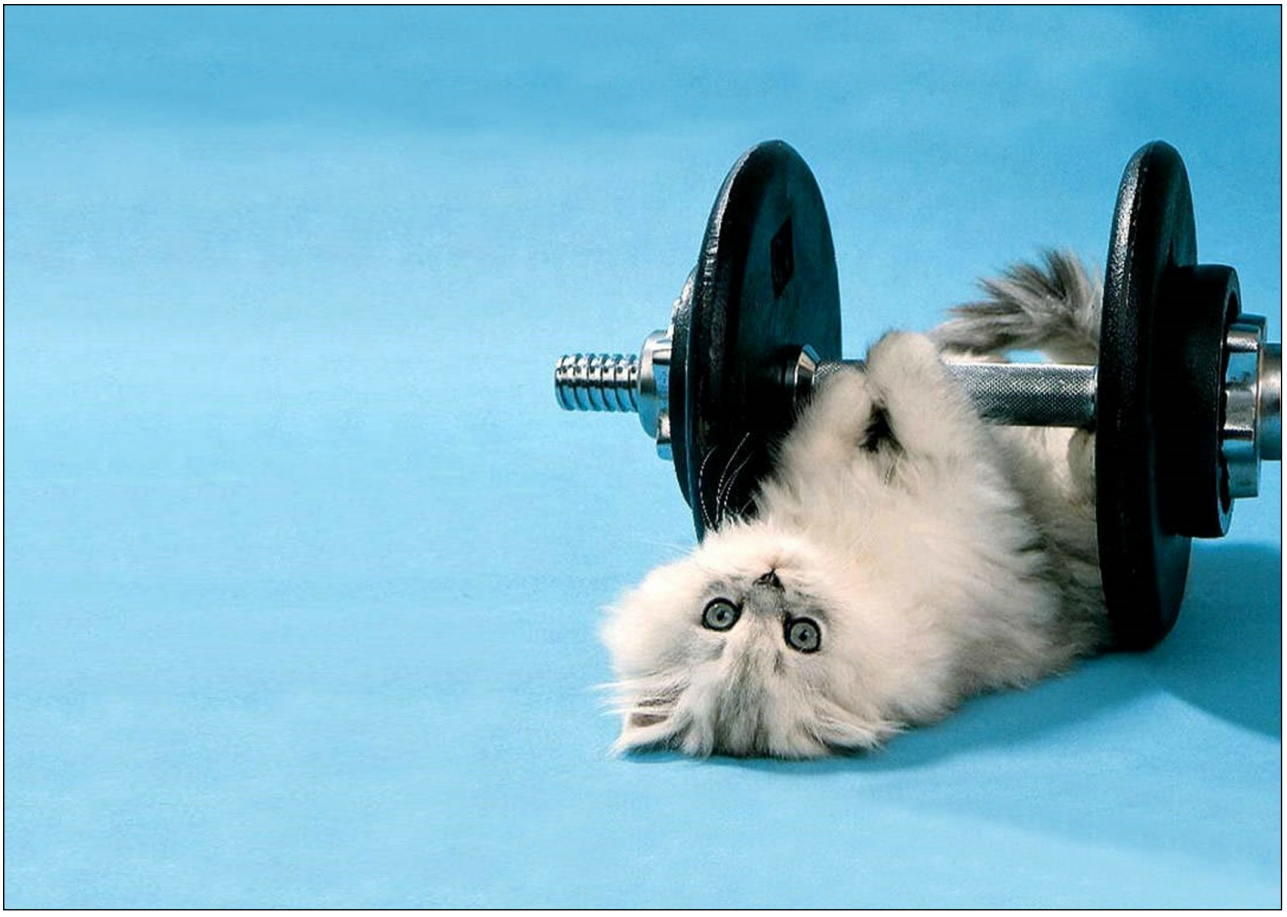
PRESENTAZIONE
Progetto di Basi di Dati

Gestione di una Palestra

Munaro Michael - 1049522

Prelaz Marco - 1047343



Abstract

Tale progetto consiste nella realizzazione di una base di dati per la gestione di una palestra.

La clientela avrà la possibilità di svolgere tra i più svariati tipi di corsi, che saranno tenuti da istruttori specializzati nelle discipline richieste. Gli utenti potranno sottoscrivere un abbonamento, che darà loro il diritto di seguire uno qualsiasi di queste attività, oppure potranno usufruire dei servizi offerti in giornata (senza quindi abbonarsi). Qualsiasi cliente, abbonato o occasionale che sia, potrà entrare in palestra semplicemente per allenarsi senza alcun obbligo di iscrizione ad un corso e inoltre se vorrà potrà essere seguito da un personal trainer.

Le operazioni tipiche sono aggiunta e modifica dei clienti ed eventualmente del personale, inserimento di nuovi corsi e la gestione delle iscrizioni a questi. Infine sarà possibile modificare i prezzi giornalieri o degli abbonamenti, che potranno essere rinnovati o eliminati una volta scaduti.

Analisi Dei Requisiti

Si vuole realizzare una base di dati che modelli alcune classi riguardanti la gestione di una palestra. Ora, indicheremo le entità progettuali e le relative informazioni necessarie:

I **corsi** saranno identificati da un codice e di questi ci interessa sapere il giorno della settimana in cui sono svolti, le date di inizio e fine corso, il tipo di corso (Zumba, Fitness, Yoga, Kung Fu, ecc...) e i partecipanti minimi richiesti per farlo partire.

Dei **clienti,** che potranno iscriversi alle varie attività sopra citate, ci interessa invece il codice fiscale, il nome, il cognome, la data di nascita, il luogo di nascita, l'indirizzo, il sesso e il telefono. I clienti possono essere **occasionali** oppure **abbonati**, nel primo caso avremo anche la data e l'orario di entrata, nel secondo invece non serviranno informazioni aggiuntive poiché questi saranno titolari di un **badge** contenente la data di inizio dell'abbonamento. Tutti i **tipi di abbonamento** sono descritti in una classe contenente la durata e il prezzo, mentre per le entrate occasionali avremo una classe **prezzo entrata** contenente il tipo di giorno (feriale/festivo) e appunto il prezzo.

Tutto questo sarà gestito dal **personale**, del quale ci interessa codice fiscale, nome, cognome, data e luogo di nascita, indirizzo, sesso, telefono e stipendio.

Il personale viene distinto in **segretari** del quale ci interessa la password (utilizzata per accedere alla base di dati che solo loro potranno gestire tramite un'interfaccia web) e in **personal trainer**, incaricati di seguire i clienti che non partecipano ai corsi, o **istruttore**. Quest'ultimo infine avrà delle **specialità** che identificano le discipline che potrà insegnare nei vari corsi.

Progettazione Concettuale

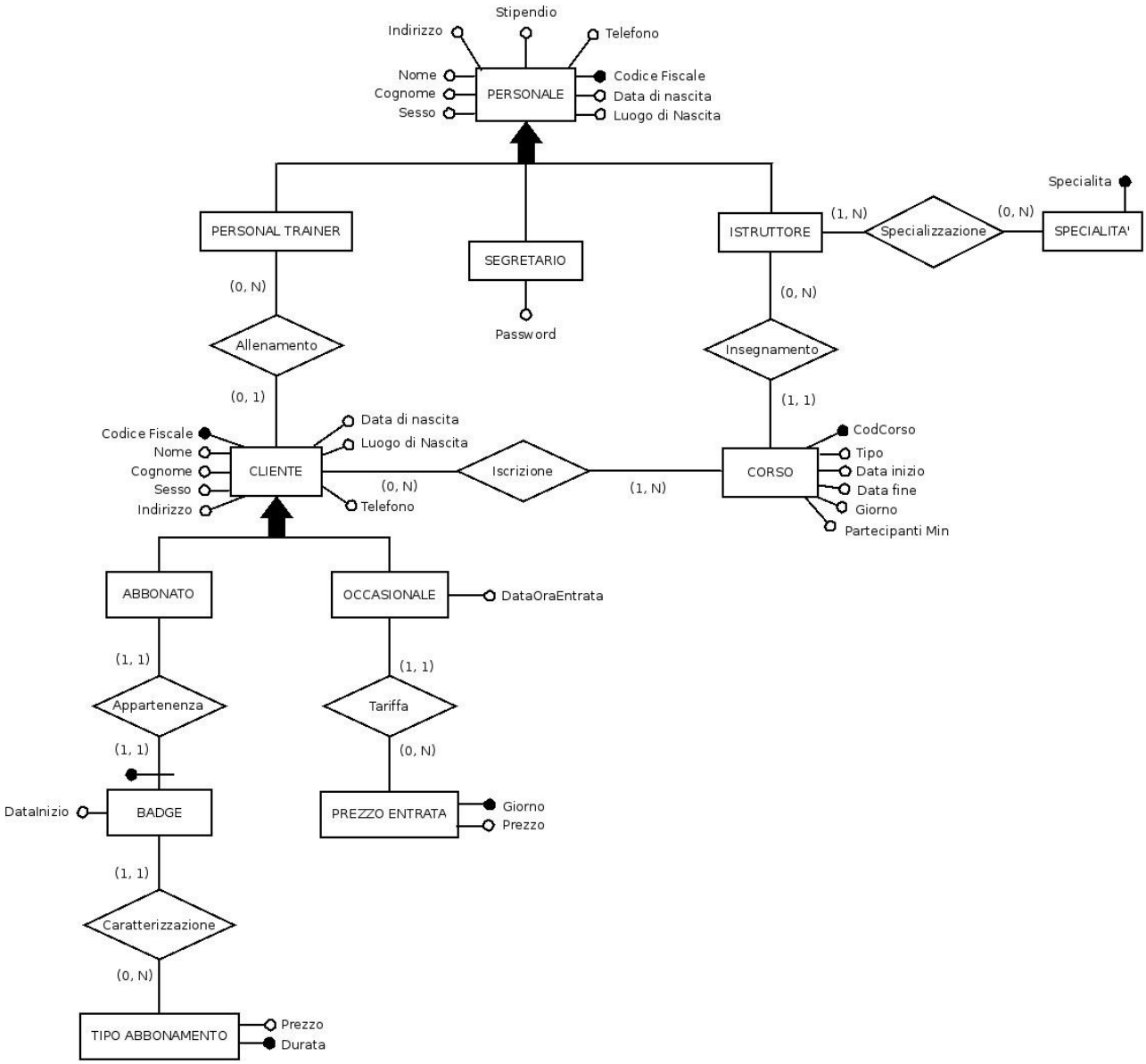
Lista Delle Classi

CLASSE	ATTRIBUTI
PERSONALE	CodiceFiscale: string Nome: string Cognome: string DataNascita: date LuogoNascita: string Indirizzo: string Sesso: enum('M,'F') Telefono: int Stipendio: real
Sottoclasse: ISTRUTTORE	
Sottoclasse: PERSONAL TRAINER	
Sottoclasse: SEGRETARIO	Password: string
CLIENTE	CodiceFiscale: string Nome: string Cognome: string DataNascita: date LuogoNascita: string Indirizzo: string Sesso: enum('M,'F') Telefono: int
Sottoclasse: ABBONATO	
Sottoclasse: OCCASIONALE	DataOraEntrata: date
BADGE	DataInizio: date
TIPO ABBONAMENTO	Prezzo: real Durata: enum

PREZZO ENTRATA	Giorno: enum Prezzo: real
CORSO	Giorno: enum Tipo: string PartecipantiMinimi: int DataInizio: date DataFine: date
SPECIALITA	Specialita: string

Lista Delle Associazioni

ASSOCIAZIONE	MOLTEPLICITA'
CLIENTE-CORSO: ***Iscrizione***	- ogni cliente può iscriversi a 1 o più corsi - ogni corso è frequentato da 0 o più clienti - molteplicità: N, N
PERSONAL TRAINER-CLIENTE: ***Allenamento***	- ogni Personal Trainer possiede 0 o più clienti - ogni cliente possiede 0 o 1 personal trainer - molteplicità: N, 1
ISTRUTTORE-CORSO: ***Insegnamento***	- ogni istruttore si occupa di 1 o più corsi - ogni corso è tenuto da 1 e un solo istruttore - molteplicità: N, 1
ISTRUTTORE-SPECIALITA: ***Specializzazione***	- ogni istruttore è specializzato in più specialità - ogni specialità caratterizza 0 o più istruttori - molteplicità: N, N
ABBONATO-BADGE: ***Appartenenza***	- ogni abbonato possiede 1 e un solo badge - ogni badge appartiene ad 1 e un solo abbonato - molteplicità: 1, 1
BADGE-TIPO ABBONAMENTO: ***Caratterizzazione***	- ogni badge è caratterizzato da un solo tipo - ogni tipo caratterizza 0 o più badge - molteplicità: 1, N
OCCASIONALE-PREZZO ENTRATA: ***Tariffa***	- ogni occasionale è caratterizzato da 1 sola tariffa - ogni prezzo entrata caratterizza 0 o più clienti occasionali


Stipendio
Indirizzo
Telefono
Nome
Cognome
Sesso
PERSONALE
Codice Fiscale
Data di nascita
Luogo di Nascita
Specialita
PERSONAL TRAINER
SEGRETARIO
ISTRUTTORE
(1, N)
Specializzazione
(0, N)
SPECIALITA'
(0, N)
Allenamento
Password
(0, N)
Insegnamento
(0, 1)
(1, 1)
Codice Fiscale
Nome
Cognome
Sesso
Indirizzo
CLIENTE
Data di nascita
Luogo di Nascita
Telefono
(0, N)
Iscrizione
(1, N)
CORSO
CodCorso
Tipo
Data inizio
Data fine
Giorno
Partecipanti Min
ABBONATO
OCCASIONALE
DataOraEntrata
(1, 1)
Appartenenza
(1, 1)
(1, 1)
Tariffa
(0, N)
DataInizio
BADGE
PREZZO ENTRATA
Giorno
Prezzo
(1, 1)
Caratterizzazione
(0, N)
TIPO ABBONAMENTO
Prezzo
Durata

Progettazione Logica

Chiavi Sintetiche

Si è scelto di aggiungere e utilizzare una *chiave sintetica* per una sola delle classi che si sono modellate.

In particolare, alla classe **Corso** è stato aggiunto l'attributo *CodCorso* per facilitarne non solo la comprensione, ma soprattutto per poter essere utilizzata come chiave primaria ed esterna relazioni che collegano questa classe ad altre.

Per le altre classi si è scelto di non introdurre delle particolari chiavi sintetiche in quanto possedevano già degli attributi utilizzabili agevolmente come chiavi (vedi codice fiscale).

Implementazioni Classi

Di seguito vengono elencate le implementazioni scelte per le classi e per le relazioni che intercorrono tra di esse:

- ***PERSONALE:***
  - *CodiceFiscale: string, PRIMARY KEY*
  - *Nome: string, NOT NULL*
  - *Cognome: string, NOT NULL*
  - *DataNascita: date*
  - *LuogoNascita: string*
  - *Indirizzo: string*
  - *Sesso: enum('M','F')*
  - *Telefono: int*
  - *Stipendio: real*

- ***SEGRETARIO:***
  - *CodiceFiscale: string, PRIMARY KEY, FOREIGN KEY(Personale)*
  - *Password: string NOT NULL*

- ***PERSONAL TRAINER:***
  - *CodiceFiscale: string, PRIMARY KEY, FOREIGN KEY(Personale)*

- ***ISTRUTTORE:***
  - *CodiceFiscale: string, PRIMARY KEY, FOREIGN KEY(Personale)*

- ***CLIENTE:***
  - *CodiceFiscale: string, PRIMARY KEY*
  - *Nome: string, NOT NULL*
  - *Cognome: string, NOT NULL*

- *DataNascita: date*
- *LuogoNascita: string*
- *Indirizzo: string*
- *Sesso: enum('M','F')*
- *Telefono: int*
- *CodPersonalTrainer: string, FOREIGN KEY(PersonalTrainer)*

- ***ABBONATO:***
  - *CodiceFiscale: string, PRIMARY KEY, FOREIGN KEY(Cliente)*

- ***OCCASIONALE:***
  - *CodiceFiscale: string, FOREIGN KEY(Cliente)*
  - *DataOraEntrata: datetime*
  - *Giorno: enum('Feriale', 'Festivo') FOREIGN KEY(PrezzoEntrata), NOT NULL*
  - *PRIMARY KEY (CodiceFiscale, DataOraEntrata)*

- ***SPECIALITA:***
  - *Specialita: string, PRIMARY KEY*

- ***CORSO:***
  - *CodCorso: string, PRIMARY KEY*
  - *Giorno: enum('Monday', 'Tuesday', 'Wednesday', …) NOT NULL*
  - *Tipo: string, NOT NULL*
  - *PartecipantiMinimi: int, NOT NULL*
  - *DataInizio: date, NOT NULL*
  - *DataFine: date, NOT NULL*
  - *CodIstruttore, FOREIGN KEY(Istruttore)*

- ***PREZZO ENTRATA:***
  - *Prezzo: real, NOT NULL*
  - *Giorno: enum('Feriale', 'Festivo') PRIMARY KEY*

- ***TIPO ABBONAMENTO:***
  - *Prezzo: real, NOT NULL*
  - *Durata: enum('Mensile', 'Trimestrale', 'Semestrale', 'Annuale') PRIMARY KEY*

- ***BADGE:***
  - *CodAbbonato: string, PRIMARY KEY, FOREIGN KEY(Abbonato)*
  - *DataInizio: date, NOT NULL*
  - *Durata: enum, NOT NULL, FOREIGN KEY (TipoAbbonamento)*

- ***SPECIALIZZAZIONE:***
  - *CodIstruttore: string, FOREIGN KEY(Istruttore)*

- *Specialita: string, FOREIGN KEY(Istruttore)*
- *PRIMARY KEY (CodIstruttore, Specialita)*

- ***ISCRIZIONE:***
  - *CodCliente: string, FOREIGN KEY(Cliente)*
  - *CodCorso: string, FOREIGN KEY(Corso)*
  - *PRIMARY KEY (CodCliente, CodCorso)*

Implementazione della Base di Dati

Di seguito viene riportato il codice *SQL* che è stato utilizzato per implementare le classi, le relazioni e le gerarchie del DataBase in base a quanto è stato descritto fino ad ora:

```
SET FOREIGN_KEY_CHECKS=0;

DROP TABLE IF EXISTS Personale;
DROP TABLE IF EXISTS Segretario;
DROP TABLE IF EXISTS Istruttore;
DROP TABLE IF EXISTS Specialita;
DROP TABLE IF EXISTS Specializzazione;
DROP TABLE IF EXISTS PersonalTrainer;
DROP TABLE IF EXISTS Corso;
DROP TABLE IF EXISTS Cliente;
DROP TABLE IF EXISTS Iscrizione;
DROP TABLE IF EXISTS Abbonato;
DROP TABLE IF EXISTS PrezzoEntrata;
DROP TABLE IF EXISTS Occasionale;
DROP TABLE IF EXISTS TipoAbbonamento;
DROP TABLE IF EXISTS Badge;


CREATE TABLE Personale (
    CodiceFiscale  CHAR(16)  PRIMARY KEY,
    Nome          CHAR(15)  NOT NULL,
    Cognome       CHAR(15)  NOT NULL,
    DataNascita    DATE,
    LuogoNascita   CHAR(20),
    Indirizzo      CHAR(30),
    Sesso         ENUM('M','F'),
    Telefono       INT(11),
    Stipendio      FLOAT(8)
    ) ENGINE=InnoDB;

CREATE TABLE Segretario (
    CodiceFiscale  CHAR(16)  PRIMARY KEY,
    Password       CHAR(15)  NOT NULL,
```

```
    FOREIGN KEY (CodiceFiscale) REFERENCES Personale(CodiceFiscale)
                        ON DELETE CASCADE
                        ON UPDATE CASCADE
    ) ENGINE=InnoDB;

CREATE TABLE PersonalTrainer (
    CodiceFiscale       CHAR(16)  PRIMARY KEY,
    FOREIGN KEY (CodiceFiscale) REFERENCES Personale(CodiceFiscale)
                        ON DELETE CASCADE
                        ON UPDATE CASCADE
    ) ENGINE=InnoDB;

CREATE TABLE Istruttore (
    CodiceFiscale       CHAR(16)  PRIMARY KEY,
    FOREIGN KEY (CodiceFiscale) REFERENCES Personale(CodiceFiscale)
                        ON DELETE CASCADE
                        ON UPDATE CASCADE
    ) ENGINE=InnoDB;

CREATE TABLE Specialita (
    Specialita          CHAR(25) PRIMARY KEY
    ) ENGINE=InnoDB;

CREATE TABLE Specializzazione (
    CodIstruttore       CHAR(16),
    Specialita          CHAR(25),
    PRIMARY KEY(CodIstruttore, Specialita),
    FOREIGN KEY (CodIstruttore) REFERENCES Istruttore(CodiceFiscale)
                                        ON DELETE CASCADE
                        ON UPDATE CASCADE,
    FOREIGN KEY (Specialita) REFERENCES Specialita(Specialita)
                                        ON DELETE CASCADE
                        ON UPDATE CASCADE
    ) ENGINE=InnoDB;

CREATE TABLE Corso (
    CodCorso            CHAR(28) PRIMARY KEY,
    Giorno              ENUM('Monday','Tuesday','Wednesday','Thursday',
                                'Friday','Saturday','Sunday') COLLATE utf8_general_ci NOT NULL,
    Tipo                CHAR(25) NOT NULL,
    PartecipantiMinimi  INT(3) NOT NULL,
    DataInizio          DATE NOT NULL,
    DataFine                    DATE NOT NULL,
    CodIstruttore       CHAR(16),
    FOREIGN KEY(CodIstruttore) REFERENCES Istruttore(CodiceFiscale)
                                        ON DELETE SET NULL
                        ON UPDATE CASCADE
    ) ENGINE=InnoDB;

CREATE TABLE Cliente (
    CodiceFiscale       CHAR(16)  PRIMARY KEY,
    Nome                CHAR(15)  NOT NULL,
    Cognome             CHAR(15)  NOT NULL,
```

```
    DataNascita          DATE,
    LuogoNascita         CHAR(20),
    Indirizzo            CHAR(30),
    Sesso                ENUM('M','F'),
    Telefono             INT(11),
    CodPersonalTrainer   CHAR(16),
    FOREIGN KEY(CodPersonalTrainer) REFERENCES PersonalTrainer(CodiceFiscale)
                                        ON DELETE SET NULL
                         ON UPDATE CASCADE
    ) ENGINE=InnoDB;

CREATE TABLE Iscrizione (
    CodCliente           CHAR(16),
    CodCorso             CHAR(28),
    PRIMARY KEY (CodCliente, CodCorso),
    FOREIGN KEY (CodCliente) REFERENCES Cliente(CodiceFiscale)
                                        ON DELETE CASCADE
                         ON UPDATE CASCADE,
    FOREIGN KEY (CodCorso) REFERENCES Corso(CodCorso)
                                        ON DELETE CASCADE
                         ON UPDATE CASCADE
    ) ENGINE=InnoDB;

CREATE TABLE Abbonato (
    CodiceFiscale        CHAR(16)   PRIMARY KEY,
    FOREIGN KEY (CodiceFiscale) REFERENCES Cliente(CodiceFiscale)
                          ON DELETE CASCADE
                          ON UPDATE CASCADE
    ) ENGINE=InnoDB;

CREATE TABLE PrezzoEntrata (
    Giorno               ENUM('Feriale','Festivo') PRIMARY KEY,
    Prezzo               FLOAT(8) NOT NULL
    ) ENGINE=InnoDB;

CREATE TABLE Occasionale (
    CodiceFiscale        CHAR(16),
    DataOraEntrata       DATETIME,
    Giorno               ENUM('Feriale','Festivo') NOT NULL,
    PRIMARY KEY(CodiceFiscale, DataOraEntrata),
    FOREIGN KEY (CodiceFiscale) REFERENCES Cliente(CodiceFiscale)
                          ON DELETE CASCADE
                          ON UPDATE CASCADE,
    FOREIGN KEY (Giorno) REFERENCES PrezzoEntrata(Giorno)
                               ON DELETE NO ACTION
                               ON UPDATE NO ACTION
    ) ENGINE=InnoDB;

CREATE TABLE TipoAbbonamento (
    Durata               ENUM('Mensile','Trimestrale','Semestrale','Annuale') PRIMARY KEY,
    Prezzo               FLOAT(8)   NOT NULL
    ) ENGINE=InnoDB;
```

```
CREATE TABLE Badge (
    CodAbbonato     CHAR(16) PRIMARY KEY,
    DataInizio      DATE NOT NULL,
    Durata          ENUM('Mensile', 'Trimestrale', 'Semestrale','Annuale') NOT NULL,
    FOREIGN KEY (CodAbbonato) REFERENCES Abbonato(CodiceFiscale)
                        ON DELETE CASCADE
                        ON UPDATE CASCADE,
    FOREIGN KEY (Durata) REFERENCES TipoAbbonamento(Durata)
                            ON DELETE NO ACTION
                            ON UPDATE NO ACTION
    ) ENGINE=InnoDB;

SET FOREIGN_KEY_CHECKS=0;
```

Query

1- Trovare i corsi che si tengono nel giorno corrente:

```
SELECT CodCorso, Tipo
FROM Corso
WHERE DAYNAME(CURRENT_DATE())=Giorno AND DataInizio<=CURRENT_DATE()
```

OUTPUT:

CodCorso	Tipo
TuesdayCrossinFit	Crossing Fit
TuesdayZumba	Zumba

2- Determinare il numero di clienti seguiti da ogni Personal Trainer

```
SELECT p.CodiceFiscale, p.Nome, p.Cognome, COUNT(c.CodiceFiscale) AS ClientiSeguiti
FROM (PersonalTrainer NATURAL JOIN Personale p) LEFT JOIN Cliente c ON
p.CodiceFiscale=c.CodPersonalTrainer
GROUP BY p.CodiceFiscale, p.Nome, p.Cognome
```

OUTPUT:

CodiceFiscale	Nome	Cognome	ClientiSeguiti
BRNMHL87A20A757Q	Michael	Bernardi	3
FNTFRC84E62L407F	Federica	Fontana	2
MLLSML84D12G224I	Samuele	Melli	4

3- Determinare il numero di abbonati per ogni tipo di abbonamento

```
SELECT Durata AS TipoAbbonamento, COUNT(CodAbbonato) AS NumeroAbbonati
FROM Badge NATURAL RIGHT JOIN TipoAbbonamento
GROUP BY Durata
```

OUTPUT:

TipoAbbonamento	NumeroAbbonati
Mensile	10
Trimestrale	4
Semestrale	4
Annuale	5

4- Determinare il numero di clienti occasionali che entrano nei giorni festivi

```
SELECT COUNT(DISTINCT CodiceFiscale) AS OccasionaliNeiGiorniFestivi
FROM Occasionale NATURAL JOIN PrezzoEntrata
WHERE Giorno='Festivo'
```

OUTPUT:

OccasionaliNeiGiorniFestivi
5

5- Trovare il/i corsi con il maggior numero di partecipanti

```
SELECT CodCorso, Tipo, COUNT(*) AS NumeroPartecipanti
FROM Iscrizione NATURAL JOIN Corso
GROUP BY CodCorso, Tipo
HAVING COUNT(*)>=ALL(SELECT COUNT(*)
                     FROM Iscrizione NATURAL JOIN Corso
                     GROUP BY CodCorso, Tipo)
```

OUTPUT:

CodCorso	Tipo	NumeroPartecipanti
SaturdayPilates	Pilates	26
MondayCapoeira	Capoeira	26

6- Determinare il numero di clienti occasionali e data dei giorni in cui sono entrati almeno 5 clienti occasionali, diversi, quest'anno

```
SELECT COUNT(DISTINCT CodiceFiscale) AS NumeroOccasionali, DATE(DataOraEntrata) AS
Data
FROM Occasionale
WHERE DATE(DataOraEntrata) IN (SELECT DATE(DataOraEntrata)
                               FROM Occasionale
                               WHERE EXTRACT(YEAR FROM DataOraEntrata)=EXTRACT(YEAR
FROM CURRENT_DATE())
                               GROUP BY DATE(DataOraEntrata)
                               HAVING COUNT(DISTINCT CodiceFiscale)>=5)
GROUP BY DATE(DataOraEntrata)
```

OUTPUT:

NumeroOccasionali	Data
5	2015-09-08

7- Restituire i badge dei clienti abbonati che scadono in giornata
(Nel nostro database, abbiamo scelto di aggiornare/cancellare un badge nel giorno della sua scadenza. Il segretario interrogherà questa query e si occuperà di cancellare o fare l'update del badge in base alle scelte del cliente contattato precedentemente per via telefonica)

```
SELECT CodiceFiscale, Nome, Cognome, Telefono
FROM Abbonato NATURAL JOIN Cliente JOIN Badge ON CodiceFiscale=CodAbbonato
WHERE ScadenzaBadge(CodiceFiscale)=CURRENT_DATE()
ORDER BY CodiceFiscale
```

OUTPUT:

CodiceFiscale	Nome	Cognome	Telefono
PPESMN87R01L736A	Simone	Pepe	34037654109

8- Eliminare il/i corsi che inizino oggi e non hanno raggiunto il numero di iscritti sufficiente

```
DELETE FROM Corso WHERE DataInizio=CURRENT_DATE() AND
NumeroIscrittiSuff(CodCorso)=0
```

Funzioni

1- Funzione che ritorna l'istruttore che tiene meno corsi e che è specializzato in un certo tipo di corso dando la priorità a chi non tiene corsi. Se più istruttori tengono lo stesso numero di corsi e questo numero è il minimo tra tutti, allora ne restituisco uno a caso (in realtà chi ha il codice fiscale minore)

```
DROP FUNCTION IF EXISTS IstruttoreMenoImpegnato;
DELIMITER |
CREATE FUNCTION IstruttoreMenoImpegnato (SpecializRichiesta CHAR(25)) RETURNS
CHAR(16)
BEGIN
DECLARE NuovoIstruttore CHAR(16);
SELECT CodiceFiscale INTO NuovoIstruttore
FROM (Istruttore LEFT JOIN Corso ON CodiceFiscale=Corso.CodIstruttore) JOIN
Specializzazione ON CodiceFiscale=Specializzazione.CodIstruttore
WHERE Specialita=SpecializRichiesta AND CodCorso IS NULL AND
CodiceFiscale<=ALL(SELECT CodiceFiscale
        FROM (Istruttore LEFT JOIN Corso ON CodiceFiscale=Corso.CodIstruttore) JOIN
Specializzazione ON CodiceFiscale=Specializzazione.CodIstruttore
        WHERE Specialita=SpecializRichiesta AND CodCorso IS NULL);
IF(NuovoIstruttore IS NOT NULL) THEN
RETURN NuovoIstruttore;
ELSE
SELECT DISTINCT CodiceFiscale INTO NuovoIstruttore
FROM (Istruttore JOIN Corso ON CodiceFiscale=Corso.CodIstruttore) JOIN Specializzazione ON
CodiceFiscale=Specializzazione.CodIstruttore
WHERE Specialita=SpecializRichiesta AND CodiceFiscale=ANY(SELECT CodiceFiscale
        FROM (Istruttore JOIN Corso ON CodiceFiscale=Corso.CodIstruttore) JOIN
Specializzazione ON CodiceFiscale=Specializzazione.CodIstruttore
        WHERE Specialita=SpecializRichiesta
        GROUP BY CodiceFiscale
        HAVING COUNT(*)<=ALL(SELECT COUNT(*)
                FROM (Istruttore JOIN Corso ON CodiceFiscale=Corso.CodIstruttore) JOIN
Specializzazione ON CodiceFiscale=Specializzazione.CodIstruttore
                WHERE Specialita=SpecializRichiesta
                GROUP BY CodiceFiscale))
                AND CodiceFiscale<=ALL(SELECT CodiceFiscale
                FROM (Istruttore JOIN Corso ON CodiceFiscale=Corso.CodIstruttore) JOIN
Specializzazione ON CodiceFiscale=Specializzazione.CodIstruttore
                        WHERE Specialita=SpecializRichiesta
                        GROUP BY CodiceFiscale
```

```
                    HAVING COUNT(*)<=ALL(SELECT COUNT(*)
                    FROM (Istruttore JOIN Corso ON CodiceFiscale=Corso.CodIstruttore) JOIN
Specializzazione ON CodiceFiscale=Specializzazione.CodIstruttore
                            WHERE Specialita=SpecializRichiesta
                            GROUP BY CodiceFiscale));
RETURN NuovoIstruttore;
END IF;
END |
DELIMITER ;
```

2- Funzione che verifica se un certo corso ha raggiunto il numero di iscritti sufficienti per farlo partire, ritorna vero o falso

```
DROP FUNCTION IF EXISTS NumeroIscrittiSuff;
DELIMITER |
CREATE FUNCTION NumeroIscrittiSuff (CodiceCorso CHAR(28)) RETURNS BOOL
BEGIN
DECLARE IscrittiMinimini INT;
DECLARE IscrittiTotali INT;
DECLARE Sufficiente BOOL;
SELECT PartecipantiMinimi, COUNT(*) INTO IscrittiMinimini,IscrittiTotali
FROM Iscrizione NATURAL JOIN Corso
WHERE CodCorso=CodiceCorso;
IF(IscrittiTotali >= IscrittiMinimini) THEN
SET Sufficiente = 1;
ELSE
SET Sufficiente = 0;
END IF;
RETURN Sufficiente;
END |
DELIMITER ;
```

3- Funzione che calcola la scadenza di un Badge in base al tipo di abbonamento (durata)

```
DROP FUNCTION IF EXISTS ScadenzaBadge;
DELIMITER |
CREATE FUNCTION ScadenzaBadge (CodiceAbbonato CHAR(16)) RETURNS DATE
BEGIN
DECLARE DataInizioAbb DATE;
DECLARE TipoAbbonamento ENUM('Mensile', 'Trimestrale', 'Semestrale','Annuale');
DECLARE DataFineAbb DATE;
SELECT DataInizio, Durata INTO DataInizioAbb,TipoAbbonamento
FROM Badge
WHERE CodAbbonato=CodiceAbbonato;
IF(TipoAbbonamento='Mensile') THEN
SET DataFineAbb = DATE_ADD(DataInizioAbb,INTERVAL 1 MONTH);
ELSEIF(TipoAbbonamento='Trimestrale') THEN
SET DataFineAbb = DATE_ADD(DataInizioAbb,INTERVAL 3 MONTH);
ELSEIF(TipoAbbonamento='Semestrale') THEN
SET DataFineAbb = DATE_ADD(DataInizioAbb,INTERVAL 6 MONTH);
ELSEIF(TipoAbbonamento='Annuale') THEN
SET DataFineAbb = DATE_ADD(DataInizioAbb,INTERVAL 1 YEAR);
END IF;
RETURN DataFineAbb;
END |
DELIMITER ;
```

Trigger

1- Trigger che controlla se un occasionale può iscriversi ad un corso, altrimenti genera un errore

```
DROP TRIGGER IF EXISTS ControlloGiorno1;
DELIMITER |
CREATE TRIGGER ControlloGiorno1
BEFORE INSERT ON Iscrizione
FOR EACH ROW
BEGIN
IF (New.CodCliente NOT IN (SELECT CodiceFiscale
                           FROM Abbonato JOIN Badge ON CodiceFiscale=CodAbbonato
                           WHERE CodiceFiscale=New.CodCliente)) AND
  (SELECT COUNT(*)
   FROM Occasionale, Corso
   WHERE CodiceFiscale=New.CodCliente AND CodCorso=New.CodCorso AND
DAYNAME(DataOraEntrata)=Corso.Giorno AND
   DATE(DataOraEntrata)=CURRENT_DATE() AND DATE(DataOraEntrata)>=DataInizio AND
DATE(DataOraEntrata)<DataFine
   )=0 THEN
INSERT INTO Iscrizione SELECT * FROM Iscrizione LIMIT 1;
END IF;
END |
DELIMITER ;
```

2- Trigger che:
1) cambia istruttore ad un corso se a questo viene assegnato un istruttore che non ha una specializzazione su quel tipo di corso. Viene scelto l'istruttore, ovviamente che ha quella specializzazione e che tiene meno corsi
2) controlla che il giorno in cui si svolge il corso sia giusto sia per la data di inizio che di fine e che la data d'inizio non venga prima di quella della fine, altrimenti genera un errore

```
DROP TRIGGER IF EXISTS ControllaGiornoESpecializ;
DELIMITER |
CREATE TRIGGER ControllaGiornoESpecializ
BEFORE INSERT ON Corso
FOR EACH ROW
BEGIN
IF (SELECT COUNT(*)
   FROM Specializzazione
```

```
   WHERE Specialita=New.Tipo AND (CodIstruttore=New.CodIstruttore OR
CodIstruttore=NULL)
  )=0 THEN
SET New.CodIstruttore=IstruttoreMenoImpegnato(New.Tipo);
END IF;
IF (DAYNAME(New.DataInizio)<>New.Giorno OR DAYNAME(New.DataFine)<>New.Giorno OR
New.DataInizio>New.DataFine) THEN
INSERT INTO Corso SELECT * FROM Corso LIMIT 1;
END IF;
END |
DELIMITER ;
```

3- Trigger che cambia personal trainer ad un cliente quando quello che lo seguiva viene licenziato (eliminato dal DB). Viene scelto il personal trainer che segue meno clienti dando la priorità a chi non ne segue. Se più personal trainer seguono lo stesso numero di clienti e questo numero è il minimo tra tutti, allora ne restituisco uno a caso (in realtà chi ha il codice fiscale minore)

```
DROP TRIGGER IF EXISTS CambiaPersonalTrainer;
DELIMITER |
CREATE TRIGGER CambiaPersonalTrainer
BEFORE DELETE ON PersonalTrainer
FOR EACH ROW
BEGIN
DECLARE NuovoPersonalTrainer CHAR(16);
SELECT PersonalTrainer.CodiceFiscale INTO NuovoPersonalTrainer
FROM PersonalTrainer LEFT JOIN Cliente ON PersonalTrainer.CodiceFiscale=CodPersonalTrainer
WHERE Cliente.CodiceFiscale IS NULL AND PersonalTrainer.CodiceFiscale<=ALL(SELECT
PersonalTrainer.CodiceFiscale
FROM PersonalTrainer LEFT JOIN Cliente ON PersonalTrainer.CodiceFiscale=CodPersonalTrainer
        WHERE Cliente.CodiceFiscale IS NULL);
IF(NuovoPersonalTrainer IS NOT NULL) THEN
UPDATE Cliente SET CodPersonalTrainer=NuovoPersonalTrainer WHERE
CodPersonalTrainer=Old.CodiceFiscale;
ELSE
SELECT DISTINCT PersonalTrainer.CodiceFiscale INTO NuovoPersonalTrainer
FROM PersonalTrainer JOIN Cliente ON PersonalTrainer.CodiceFiscale=CodPersonalTrainer
WHERE PersonalTrainer.CodiceFiscale=ANY(SELECT PersonalTrainer.CodiceFiscale
                        FROM PersonalTrainer JOIN Cliente ON
PersonalTrainer.CodiceFiscale=CodPersonalTrainer
                        GROUP BY PersonalTrainer.CodiceFiscale
                        HAVING COUNT(*)<=ALL(SELECT COUNT(*)
```

```
                                FROM PersonalTrainer JOIN Cliente ON
PersonalTrainer.CodiceFiscale=CodPersonalTrainer
                                GROUP BY PersonalTrainer.CodiceFiscale))
  AND PersonalTrainer.CodiceFiscale<=ALL(SELECT PersonalTrainer.CodiceFiscale
                                 FROM PersonalTrainer JOIN Cliente ON
PersonalTrainer.CodiceFiscale=CodPersonalTrainer
                                 GROUP BY PersonalTrainer.CodiceFiscale
                                 HAVING COUNT(*)<=ALL(SELECT COUNT(*)
                                 FROM PersonalTrainer JOIN Cliente ON
PersonalTrainer.CodiceFiscale=CodPersonalTrainer
                                GROUP BY PersonalTrainer.CodiceFiscale));
UPDATE Cliente SET CodPersonalTrainer=NuovoPersonalTrainer WHERE
CodPersonalTrainer=Old.CodiceFiscale;
END IF;
END |
DELIMITER ;
```

Interfaccia Web

L'interfaccia web consente di eseguire le operazioni fondamentali sulla base di dati. Queste operazioni sono: *Inserimento* (personale, clienti, corsi, specializzazioni e iscrizioni), *Aggiornamento* (tutte le tabelle tranne specializzazione), *Cancellazione* (personale, clienti e corsi) e *Visualizzazione* (tutte le tabelle).

La connessione alla base di dati avviene mediante l'inclusione del seguente file php che ha il compito di autenticarsi e selezionare il database che si vuole utilizzare.

Connessione_db.php:

```
<?php
        $connessione=mysql_connect( basidati, mmunaro', 'password_mysql' )
        or die($_SERVER['PHP_SELF'] . "Connessione fallita!");
        mysql_select_db('mmunaro-PR', $connessione);
?>
```

La HomePage del progetto ha il compito di ricevere tramite form i campi dati 'Utente' e 'Password' per l'accesso alla base di dati. Gli utenti che hanno il permesso alla login sono solamente i segretari (tabella SQL) dato che sono unicamente loro che utilizzano questo gestionale. Per comodità è stato inserito un utente 'admin' 'password'.

index.xhtml:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/
DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="it" lang="it">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <title>Palestra The Gym</title>
  <meta name="title" content="Palestra The Gym" />
  <meta name="description" content="Home Page del sito di The Gym" />
  <meta name="language" content="ïtalian it" />
  <meta name="Author" content="Munaro Michael, Prelaz Marco" />
  <link rel="stylesheet" type="text/css" href="style.css"/>
</head>
<body>
  <div id="logo">
        <img src="images/test.png" alt="logo The Gym" />
  </div>
```

```
        <div id="login">
    <form method="post" action="login.php" id="form">
      <p>
        <label for="username"></label>
        <input type="text" name="username" id="username" placeholder="Username"/>
        <label for="password"></label>
        <input type="password" name="password" id="password" placeholder="Password"/>
       <button type="submit" id="logButton"><span xml:lang="en">Login</span></button>
      </p>
    </form>
  </div>
</body>
</html>
```

La pagina Login ha il compito di controllare se i dati inseriti corrispondano con i dati presenti nella tabella Segretari (o se corrispondono a 'admin' 'password') e se questi corrispondono viene creata la sessione che è mantenuta per tutta la durata della navigazione (ogni pagina ha bisogno dell'autenticazione per essere utilizzata)

login.php:

```
<?php
//includo i file necessari a collegarmi al db con relativo script di accesso
include("connessione_db.php");

//variabili POST con anti sql Injection

$username=$_POST["username"]; //faccio l'escape dei caratteri dannosi
$password=$_POST["password"]; //sha1 cifra la password anche qui in questo modo corrisponde
con quella del db

 $query = "SELECT * FROM Segretario WHERE CodiceFiscale = \"$username\" AND
Password = \"$password\" ";
 $ris = mysql_query($query, $connessione) or die (mysql_error());
 $riga=mysql_fetch_array($ris);

/*Prelevo l'identificativo dell'utente */
$login=$riga['CodiceFiscale'];
/* Effettuo il controllo */
if ($login == NULL) $trovato = 0 ;
else $trovato = 1;
```

```
/* Username e password corrette */
if($trovato == 1 || ($username=='admin' && $password=='password') ) {
  if($username=='admin' && $password=='password') {
          $login='username';
          $password='password';
   }
 /*Registro la sessione*/
  session_start();

  /*Registro il codice dell'utente*/
  $_SESSION['login'] = $login;
  Header('Location: HomeAmministrazione.php'); }
else
 Header('Location: index.xhtml');
?>
```

Subito dopo l'autenticazione, si viene rimandati alla HomeAmministrazione che è la pagina principale dalla quale si può scegliere l'operazione da effettuare

HomeAmministrazione.php:

```
<?
/* attiva la sessione */
session_start();
/* verifica se la variabile 'login' e` settata */
$login=$_SESSION['login'];
if (empty($login)) {
 /* non passato per il login: accesso non autorizzato ! */
 echo "Impossibile accedere. Prima effettuare il login.";
} else {
?>

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="it" lang="it">
<head>
   <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
   <title>Palestra The Gym</title>
   <meta name="title" content="Palestra The Gym" />
   <meta name="description" content="Home Page del sito di The Gym" />
   <meta name="language" content="ïtalian it" />
   <meta name="Author" content="Munaro Michael, Prelaz Marco" />
   <link rel="stylesheet" type="text/css" href="style.css"/>
```

```
</head>
<body>
<h1> Amministrazione </h1>
<h2> Seleziona Un'Operazione </h2>
<table width="100%" CELLSPACING="50" align="center">
   <tr>
     <td align="center" ><h2><a href="Inserimento.php">Inserimento</a></h2></td>
   </tr>
   <tr>
     <td align="center" ><h2><a href="Aggiornamento.php">Aggiornamento</a></h2></td>
   </tr>
   <tr>
     <td align="center" ><h2><a href="Cancellazione.php">Cancellazione</a></h2></td>
   </tr>
   <tr>
     <td align="center" ><h2><a href="Visualizzazione.php">Visualizzazione</a></h2></td>
   </tr>
</table>
</font>

<h2><a href="logout.php">LogOut</a></h2>
</body>
</html>
<?
}
?>
```

Una delle scelte previste dalla HomeAmministrazione è quella di LOGOUT che rimanda alla pagina *logout* la quale distrugge la sessione corrente

logout.php:

```
<?php
session_start();
session_destroy();
?>

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="it" lang="it">
<head>
   <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
   <title>Palestra The Gym</title>
```

```
    <meta name="title" content="Palestra The Gym" />
    <meta name="description" content="Home Page del sito di The Gym" />
    <meta name="language" content="ïtalian it" />
    <meta name="Author" content="Munaro Michael, Prelaz Marco" />
    <link rel="stylesheet" type="text/css" href="style.css"/>

</head>
<body id="logoutPage">
<h2>Logout effettuato!
<br />
<a href="index.xhtml"> Torna alla pagina di Login </a></h2>
</body>
</html>
```

L'organizzazione delle restanti pagine php in linea generale è la seguente:

• INSERIMENTO: Ad ogni tipo di inserimento corrisponde una pagina con una form, nella quale inserire i dati che verranno *salvati* in variabili php da un'altra pagina (le pagine che sono state chiamate di 'recupero'). Queste, oltre a memorizzare le variabili, hanno il compito di effettuare l'inserimento (in questo caso) tramite una query da inviare al database mediante l'apposita istruzione. Se ci fossero degli errori durante l'esecuzione della query, questi verrebbero mostrati in output.

•AGGIORNAMENTO: Ad ogni aggiornamento corrisponde una pagina nella quale si seleziona chi/cosa modificare (Corso, Istruttore,…). Dopo di che viene raggiunta una pagina nella quale inserire i dati da modificare (mediante apposite form) e che esegue la modifica vera e propria mediante query da mandare al database (sempre con l'apposito comando) . Se ci fossero degli errori durante l'esecuzione della query, questi verrebbero mostrati in output.

•CANCELLAZIONE: Ad ogni cancellazione corrisponde una pagina nella quale scegliere chi/cosa cancellare (Personale, Clienti, …) dalla base di dati. Anche qui se ci fossero degli errori durante l'esecuzione della query, questi verrebbero mostrati in output.

•VISUALIZZAZIONE: Per ogni tabella della base di dati corrisponde una pagina php che ha il compito di interrogare la base di dati mediante una query che restituisce tutti i valori della tabella interessata.