Grigoras Valentin

Progetto Basi di Dati

1. ABSTRACT

La palestra è una struttura che offre diversi corsi dal lunedì al sabato, dalle 09:00 alle 23:00, i quali si adeguano sempre di più ai bisogni e alle esigenze specifiche delle persone; tutto ciò viene garantito da personal trainer che possono seguire i clienti durante e fuori la sessione di allenamento.

Viene offerta la possibilità ad ogni persona di creare per un periodo di tempo pari a 3, 6 o 12 mesi, un proprio abbonamento scegliendo i corsi desiderati; in alternativa, potrà optare per uno tra i pacchetti offerti. Ciascun corso viene tenuto da almeno due allenatori in giorni e fasce orarie diverse, in una delle stanze della palestra; la sala pesi invece, è disponibile tutto il giorno.

I clienti che si allenano nella sala pesi, possono allenarsi da soli o decidere di essere seguiti da un personal trainer (PT). In tal caso, essi seguiranno una scheda preparata dal relativo PT per un determinato periodo, composta da almeno 10 esercizi completi di nome e descrizione relativa.

Recentemente si è constatato un forte aumento delle iscrizioni nelle palestre, questo fatto ha reso necessario l'utilizzo di basi di dati per poter memorizzare e gestire le informazioni in modo tale da garantire il corretto funzionamento del sistema.

2. ANALISI DEI REQUISITI

Si vuole realizzare una base di dati che contenga le informazioni relative alla gestione di una palestra ed in particolare si vogliono conoscere i dati relativi alle persone coinvolte e ai corsi offerti nei quali è compresa anche la sala pesi. Ciascuna persona è identificata dalle seguenti informazioni:

- un codice fiscale, che la identifica univocamente;
- un nome;

- un cognome;
- un numero di telefono;
- una e-mail;
- una data di nascita;
- un indirizzo composto da: città, via, numero civico e provincia.

Una persona è rappresentata dal cliente **e/o** dall' allenatore dotato di un ulteriore attributo booleano 'salaPesi' che è true sse tale allenatore insegna nella sala pesi. Nel caso in cui un cliente che segue il corso sala pesi decide di non essere seguito da un PT, la relativa scheda che si "auto-preparerà" non verrà memorizzata nel database. Un' allenatore invece, insegnerà in **uno solo** dei corsi offerti. Se esso si occuperà della sala pesi, seguirà degli abbonati e in tal caso, preparerà una scheda che sarà seguita da **un solo** cliente, composta da:

- un n° scheda, che la identifica univocamente;
- una data inizio;
- una data fine.

Ciascuna scheda, è a sua volta formata da **almeno 10** esercizi identificati da:

- un nome, che lo identifica univocamente;
- una descrizione per l' esercizio.

Ogni cliente stipulerà **un solo** abbonamento reale della durata di: 90 (fattore moltiplicativo pari a 1), 180 (con uno sconto del 10% => fattore moltiplicativo pari a 0.9) o 360 (con uno sconto del 30% => fattore moltiplicativo pari a 0.7) giorni. Al termine di quest'ultimo potrà essere annullato (il cliente cambierà abbonamento o si disiscriverà dalla palestra). Tale abbonamento reale fa riferimento a **uno e uno solo** abbonamento tra quelli possibili ed esso è identificato da:

- una data inizio, che lo identifica univocamente;
- una data fine, che lo identifica univocamente;
- uno sconto, che rappresenta il fattore moltiplicativo.

Ogni abbonamento verrà utilizzato per ottenere **almeno un** abbonamento reale, ed è identificato da:

- un ID, che lo identifica univocamente.

Ogni abbonamento sarà composto da **almeno un** corso identificato da:

- un nome, che lo identifica univocamente;
- un costo.

Infine, il corso sarà presente nella programmazione **almeno una volta**; essa è composta da:

- un n° stanza, che la identifica univocamente;
- un giorno, che la identifica univocamente;
- un' oraI, che la identifica univocamente;
- un' oraF.

Tale base di dati dunque, tra le varie funzionalità offrirà inserimento, modifica, rimozione e ricerca:

- delle persone e dell'eventuale abbonamento reale associato (un abbonamento e un abbonamento reale non possono essere cancellati dal database mantenendo lo storico degli abbonamenti reali);
- dei corsi offerti con le relative informazioni (stanza nella quale viene tenuto il corso, giorno/i, fasce orarie, ecc...);
- delle schede preparate (delle quali viene mantenuto lo storico e verranno cancellate solo quando l'abbonato non sarà più iscritto, oppure quando non sarà più seguito dal relativo PT);
- degli esercizi proposti dai PT.

3. PROGETTAZIONE CONCETTUALE

3.1 Descrizione delle classi

- **Persona**: modella una generica persona.

  Attributi:

  - *CF*: char(16) <<*PK*>> – rappresenta il codice fiscale della persona;
  - *nome*: varchar(255) – rappresenta il nome della persona;
  - *cognome*: varchar(255) – rappresenta il cognome della persona;
  - *dNascità*: date – rappresenta la data di nascita della persona;
  - *Email*: varchar(255) – rappresenta l'indirizzo e-mail della persona;
  - *nTel*: varchar(10) – rappresenta il recapito telefonico della persona;
  - *indirizzo* – attributo composto da:
    - *città*: varchar(255) – rappresenta la città di residenza della persona;
    - *prov*: char(2) – rappresenta la provincia di residenza della persona;
    - *via*: varchar(255) – rappresenta la via di residenza della persona;
    - *nCiv*: varchar(4) – rappresenta il n° civico di residenza della persona.

  Sono definite le seguenti sottoclassi di Persona:

  1. **Cliente**: modella un utente cliente della palestra. Gli attributi che interessano un cliente sono gli stessi che per una Persona generica.
  2. **Allenatore**: modella un utente allenatore della palestra. Gli attributi che interessano un allenatore sono gli stessi che per una Persona generica; in aggiuntiva gode del seguente attributo:
     - *salaPesi*: bool – è true <=> l'allenatore insegna nella sala pesi, è false <=> l'allenatore insegna in uno degli altri corsi.

- **Corso**: modella un corso generico della palestra.

Attributi:

- *NOME*: varchar(255) <<*PK*>> - rappresenta il nome del corso;
- *costo*: decimal(6,2) – rappresenta il costo in euro del corso.

- **Programmazione**: modella una stanza generica della palestra nella quale vengono tenuti i corsi.

  Attributi:

  - *nStanza*: int <<*PK*>> - rappresenta il numero della stanza;
  - *giorno*: enum <<*PK*>> - rappresenta un giorno nel quale sarà tenuto il corso;
  - *oraI*: time <<*PK*>> - rappresenta l' orario d' inizio corso;
  - *oraF*: time - rappresenta l' orario di fine corso.

- **Abbonamento**: modella un generico abbonamento della palestra.

  Attributi:

  - *ID*: char(3) <<*PK*>> - rappresenta l' identificativo dell'abbonamento;

- **Abbonamento reale**: modella un abbonamento stipulato.

  Attributi:

  - *ID*: char(3) <<*PK*>> <<*FK*>> - rappresenta l' identificativo dell' abbonamento;
  - *dataI*: date <<*PK*>>– rappresenta la data d' inizio dell' abbonamento reale;
  - *dataF*: date <<*PK*>> – rappresenta la data di fine abbonamento reale;

- *sconto*: decimal(2,1) – rappresenta il fattore moltiplicativo da applicare per ottenere un' eventuale sconto nell' abbonamento reale.

- **Scheda**: modella una scheda di allenamento per un cliente iscritto alla sala pesi seguito da un PT.

  Attributi:

  - *NUMERO*: varchar(5) <<*PK*>> - rappresenta il numero della scheda;
  - *cfCliente*: char(16) <<*PK*>> <<*FK*>> - rappresenta il codice fiscale della persona che segue la scheda;
  - *cfAllenatore*: char(16) <<*PK*>> <<*FK*>> - rappresenta il codice fiscale dell'allenatore che ha preparato la scheda;
  - *dInizio*: date – rappresenta la data di inizio scheda;
  - *dFine*: date – rappresenta la data di fine scheda.
- **Esercizio**: modella un esercizio generico da svolgere nella sala pesi.

  Attributi:

  - *NOME*: varchar(255) <<*PK*>> - rappresenta il nome dell'esercizio;
  - *descrizione*: varchar(255) – rappresenta la descrizione per la corretta esecuzione dell'esercizio.

3.2 Descrizione delle Associazioni

- **Cliente – Scheda**: "*Seguimento*"
  - Un cliente può seguire al massimo una scheda. Una scheda è seguita da un solo cliente;
  - Molteplicità: 1:1;

- **Scheda – Allenatore**: "*Preparazione*"

- Una scheda è preparata da un solo allenatore. Un'allenatore prepara da zero a più schede;
- Molteplicità: 1:N;

- **Scheda – Esercizio**: "*Formazione*"
  - Una scheda è composta da almeno 10 esercizi caratterizzati eventualmente da un numero serie, un numero ripetizioni, un tempo di recupero in secondi, un peso in kg e una durata in minuti. Un esercizio fa parte di zero o più schede;
  - Molteplicità: N:N;

- **Allenatore – Corso**: "*Insegnamento*"
  - Un allenatore insegna in un solo corso. Un corso viene tenuto da almeno due allenatori;
  - Molteplicità: 1:N;
- **Corso – Programmazione**: "*Presenza*"
  - Un corso sarà presente nella programmazione almeno una volta. Dato il giorno, una stanza e l'ora d' inizio corso, un corso sarà presente nella programmazione una sola volta;
  - Molteplicità: 1:N;

- **Abbonamento – Corso**: "*Composizione*"
  - Un abbonamento è composto da almeno un corso. Un corso fa parte di almeno un abbonamento;
  - Molteplicità: N:N;

- **Cliente – Abbonamento Reale**: "*Stipulazione*"
  - Un cliente stipula un solo abbonamento reale. Un abbonamento reale può essere stipulato da più clienti;

- Molteplicità: 1:N;

- **Abbonamento – Abbonamento Reale**: "*Costruzione*"
    - Un abbonamento costruisce almeno un abbonamento reale. Un abbonamento reale viene costruito da un solo abbonamento;
    - Molteplicità: 1:N;

3.3 Descrizione delle Gerarchie

Lo schema concettuale della base di dati prevede un unica gerarchia, quella con superclasse Persona e come sottoclassi Cliente ed Allenatore.
La generalizzazione della classe Persona soddisfa le seguenti proprietà:

• **Totale**, ovvero ogni occorrenza dell'entità Persona è una occorrenza di almeno una delle entità figlie (Cliente / Allenatore);
• **Sovrapposta**, ovvero ogni occorrenza dell'entità Persona è occorrenza di almeno una delle entità figlie (Cliente / Allenatore), ovvero una persone è Cliente e/o Allenatore.

Le due sottoclassi non prevedono attributi aggiuntivi, eccetto "Allenatore" che aggiunge l'attributo booleano "salaPesi" per identificare gli allenatori che si occupano della stessa.

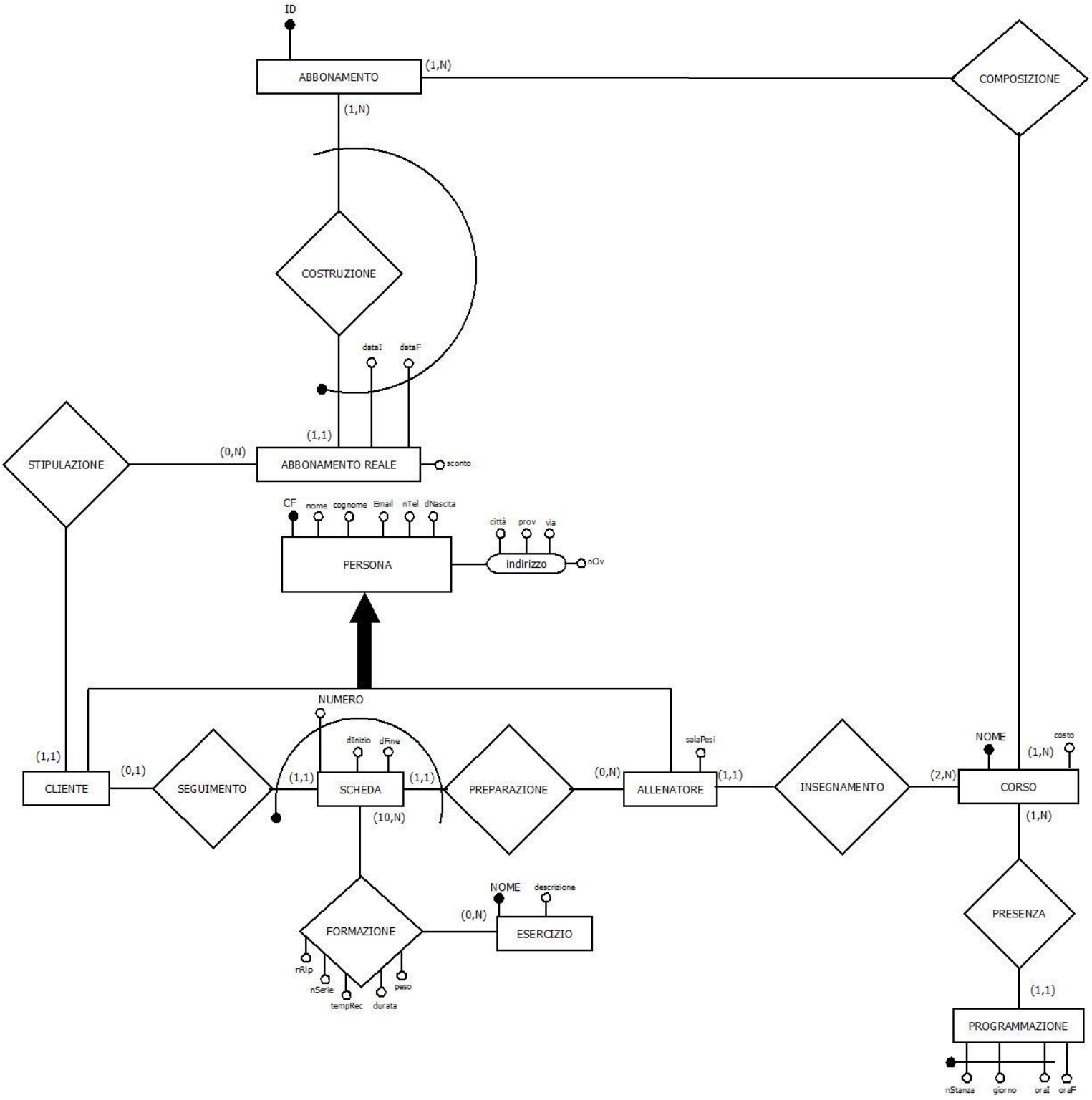
3.4 Descrizione dei vincoli d'Integrità aggiuntivi

Sono presenti alcuni vincoli di integrità semantici che non sono stati inseriti all'interno dello schema concettuale.

- *Scheda.dInizio* < *Scheda.dFine*: la data d'inizio della scheda dovrà essere antecedente alla data di fine scheda;
- *Programmazione.oraI* < Programmazione.*oraF*: l'ora d'inizio corso dovrà essere antecedente all'ora di fine corso;
- *Abbonamento reale.dataI* < *Abbonamento reale.dataF*: la data d'inizio dell ' abbonamento reale dovrà essere antecedente alla data di fine abbonamento reale;

- *Abbonamento reale.dataF - Abbonamento reale.dataI* dovrà corrispondere (con un margine di 5 giorni in difetto o eccesso) a uno dei seguenti valori: 90 (giorni), 180 (giorni) o 360 (giorni);
- *Abbonamento reale.sconto* = 1 <=> *Abbonamento reale.dataF – Abbonamento reale.dataI = 90;*
- *Abbonamento reale.sconto* = 0.9 <=> *Abbonamento reale.dataF – Abbonamento reale.dataI = 180;*
- *Abbonamento reale.sconto* = 0.7 <=> *Abbonamento reale.dataF – Abbonamento reale.dataI = 360.*

DIAGRAMMA E-R



4. PROGETTAZIONE LOGICA

4.1 Ristrutturazione dello schema E-R

4.1.1 Eliminazione delle generalizzazioni

Riguardo la gerarchia esistente nel modello concettuale avente Persona come superclasse e Cliente ed Allenatore come sottoclassi, è stato effettuato un accorpamento del genitore nelle figlie nel modello relazionale. Le ragioni di tale scelta sono state le seguenti:

- la generalizzazione è totale;
- la classe radice (Persona) non ha delle associazioni proprie, a differenza delle classi figlie (Cliente, Allenatore) aventi i medesimi attributi della classe radice e associazioni proprie. Viene dunque eliminata la classe Persona.

La gerarchia completa viene quindi ridotta alle seguenti classi:

- *Cliente*: essa eredita tutti i campi della classe Persona;
- *Allenatore*: essa eredita tutti i campi della classe Persona e Allenatore.

4.2 Traduzione verso il modello relazionale

4.2.1 Descrizione dello schema relazionale ottenuto

- **Cliente** (<u>CF</u>, IdAbbonamento, inizioA, fineA, nome, cognome, Email, nTel, dNascita, città, prov, via, nCiv);
- **Allenatore** (<u>CF</u>, corso, nome, cognome, Email, nTel, dNascita, città, prov, via, nCiv, salaPesi);
- **Scheda** (<u>cfCliente</u>, <u>cfAllenatore</u>, <u>NUMERO</u>, dInizio, dFine);
- **Esercizio** (<u>NOME</u>, descrizione);

- **Corso** (NOME, costo);
- **Programmazione** (nStanza, giorno, oraI, oraF, corso);
- **Abbonamento** (ID);
- **Abbonamento Reale** (dataI, dataF, abbonamento, sconto);
- **Formazione** (cfCliente, cfAllenatore, scheda, esercizio, nRip, nSerie, tempRec, peso, durata);
- **Composizione** (abbonamento, corso).

5. IMPLEMENTAZIONE DELLA BASE SI DATI

5.1 Creazione delle tabelle e dei trigger

Qui di seguito verrà riportato il codice relativo alla base di dati. Sono state definite le seguenti tabelle: Abbonamento, AbbonamentoReale, Cliente, Corso, Composizione, Allenatore, Esercizio, Scheda, Formazione, Programmazione.

```
DROP TABLE IF EXISTS `Abbonamento`;
DROP TABLE IF EXISTS `AbbonamentoReale`;
DROP TABLE IF EXISTS `Cliente`;
DROP TABLE IF EXISTS `Corso`;
DROP TABLE IF EXISTS `Composizione`;
DROP TABLE IF EXISTS `Allenatore`;
DROP TABLE IF EXISTS `Esercizio`;
DROP TABLE IF EXISTS `Scheda`;
DROP TABLE IF EXISTS `Formazione`;
DROP TABLE IF EXISTS `Programmazione`;

CREATE TABLE `Abbonamento` (
        `ID` char(3) PRIMARY KEY )
        ENGINE=InnoDb;

CREATE TABLE `AbbonamentoReale` (
        `abbonamento` char(3),
        `dataI` date,
        `dataF` date,
        `sconto` decimal(2,1),
        FOREIGN KEY (`abbonamento`) REFERENCES
        Abbonamento(`ID`) ON DELETE NO ACTION ON UPDATE CASCADE,
        PRIMARY KEY (`abbonamento`,`dataI`,`dataF`))
        ENGINE=InnoDb;

DROP TRIGGER IF EXISTS `Sconto`;

DELIMITER |
CREATE TRIGGER `Sconto`
BEFORE INSERT ON `AbbonamentoReale`
FOR EACH ROW
BEGIN
        IF DATEDIFF(NEW.dataF,NEW.dataI)>=85 &&
        DATEDIFF(NEW.dataF,NEW.dataI)<=95 THEN
        SET NEW.sconto=1;
        ELSEIF DATEDIFF(NEW.dataF,NEW.dataI)>95 &&
        DATEDIFF(NEW.dataF,NEW.dataI)<=185 THEN
        SET NEW.sconto=0.9;
        ELSEIF DATEDIFF(NEW.dataF,NEW.dataI)>185 &&
        DATEDIFF(NEW.dataF,NEW.dataI)<=370 THEN
        SET NEW.sconto=0.7;
        END IF;
```

```
END |

DROP TRIGGER IF EXISTS `EliminazioneAR`;

CREATE TRIGGER `EliminazioneAR`
BEFORE DELETE ON `AbbonamentoReale`
FOR EACH ROW
BEGIN
        IF OLD.abbonamento <> '' THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'This record is sacred! You are not allowed to remove it!!';
        END IF ;
END |

DROP TRIGGER IF EXISTS `CambioAbbonamentoAR`;

CREATE TRIGGER `CambioAbbonamentoAR`
BEFORE UPDATE ON `AbbonamentoReale`
FOR EACH ROW
BEGIN
IF NEW.dataI>OLD.dataI && NEW.dataI<=OLD.dataF THEN
SIGNAL SQLSTATE '45000'
SET MESSAGE_TEXT = 'This record can not be changed!!';
END IF;
END|
DELIMITER ;

CREATE TABLE `Cliente` (
        `CF` char(16) PRIMARY KEY,
        `IdAbbonamento` char(3) not null,
        `inizioA` date not null,
        `fineA` date not null,
        `nome` varchar(255) not null,
        `cognome` varchar(255) not null,
        `Email` varchar(255),
        `nTel` varchar(10) not null,
        `dNascita` date not null,
        `cittÃ ` varchar(255) not null,
        `prov` char(2) not null,
        `via` varchar(255) not null,
        `nCiv` varchar(4) not null,
        FOREIGN KEY (`IdAbbonamento`,`inizioA`,`fineA`) REFERENCES
        AbbonamentoReale(`abbonamento`,`dataI`,`dataF`)
        ON UPDATE CASCADE)
        ENGINE=InnoDb;

CREATE TABLE `Corso` (
        `NOME` varchar(255) PRIMARY KEY,
        `costo` decimal(6,2) not null)
        ENGINE=InnoDb;

CREATE TABLE `Composizione` (
        `abbonamento` char(3),
        `corso` varchar(255),
        FOREIGN KEY (`abbonamento`) REFERENCES Abbonamento(`ID`)
        ON DELETE NO ACTION ON UPDATE CASCADE,
        FOREIGN KEY (`corso`) REFERENCES Corso(`NOME`)
        ON DELETE CASCADE ON UPDATE CASCADE,
        PRIMARY KEY (`abbonamento`,`corso`))
        ENGINE=InnoDb;

CREATE TABLE `Allenatore` (
        `CF` char(16) PRIMARY KEY,
        `corso` varchar(255),
        `nome` varchar(255) not null,
        `cognome` varchar(255) not null,
        `Email` varchar(255),
```

```
	`nTel` varchar(10) not null,
	`dNascita` date not null,
	`cittÃ ` varchar(255) not null,
	`prov` char(2) not null,
	`via` varchar(255) not null,
	`nCiv` varchar(4) not null,
	`salaPesi` bool not null,
	FOREIGN KEY (`corso`) REFERENCES Corso(`NOME`)
	ON DELETE SET NULL ON UPDATE CASCADE)
	ENGINE=InnoDb;

CREATE TABLE `Esercizio` (
	`NOME` varchar(255) PRIMARY KEY,
	`descrizione` varchar(255))
	ENGINE=InnoDb;

CREATE TABLE `Scheda` (
	`NUMERO` varchar(5),
	`cfCliente` char(16),
	`cfAllenatore` char(16),
	`dInizio` date not null,
	`dFine` date not null,
	FOREIGN KEY (`cfCliente`) REFERENCES Cliente(`CF`)
	ON DELETE CASCADE ON UPDATE CASCADE,
	FOREIGN KEY (`cfAllenatore`) REFERENCES Allenatore(`CF`)
	ON DELETE CASCADE ON UPDATE CASCADE,
	PRIMARY KEY (`NUMERO`,`cfCliente`,`cfAllenatore`))
	ENGINE=InnoDb;

CREATE TABLE `Formazione` (
	`cfCliente` char(16),
	`cfAllenatore` char(16),
	`scheda` varchar(5),
	`esercizio` varchar(255),
	`nRip` int,
	`nSerie` int,
	`tempRec` int,
	`peso` int,
	`durata` int,
	FOREIGN KEY (`cfCliente`) REFERENCES Cliente(`CF`)
	ON DELETE CASCADE ON UPDATE CASCADE,
	FOREIGN KEY (`cfAllenatore`) REFERENCES Allenatore(`CF`)
	ON DELETE CASCADE ON UPDATE CASCADE,
	FOREIGN KEY (`scheda`) REFERENCES Scheda(`NUMERO`)
	ON DELETE NO ACTION ON UPDATE CASCADE,
	FOREIGN KEY (`esercizio`) REFERENCES Esercizio(`NOME`)
	ON DELETE CASCADE ON UPDATE CASCADE,
	PRIMARY KEY (`cfCliente`,`cfAllenatore`,`scheda`,`esercizio`))
	ENGINE=InnoDb;

CREATE TABLE `Programmazione` (
	`nStanza` int,
	`giorno` ENUM('Lun','Mar','Mer','Gio','Ven','Sab') not null,
	`oraI` time not null,
	`oraF` time not null,
	`corso` varchar(255),
	FOREIGN KEY (`corso`) REFERENCES Corso(`NOME`)
	ON DELETE CASCADE ON UPDATE CASCADE,
	PRIMARY KEY (`nStanza`,`giorno`,`oraI`))
	ENGINE=InnoDb;
```

Come si è potuto vedere nel codice sovrastante, sono stati creati i seguenti trigger:

- Sconto: trigger attivo che setta il fattore moltiplicativo relativo allo sconto ad ogni inserimento di un nuovo abbonamento reale;
- EliminazioneAR: trigger passivo che impedisce l' eliminazione degli abbonamenti reali;
- CambioAbbonamentoAR: trigger passivo che impedisce il cambio abbonamento per un cliente nel caso ne abbia già attivo un' altro.

5.2 *Creazione delle funzioni*

Sono state definite le seguenti funzioni:

- NumeroIscritti: dato un periodo, restituisce il numero di persone con un abbonamento attivo in quel periodo;
- CostoAbbonamento: dato un CF, restituisce il costo totale dell' abbonamento reale al quale viene applicato l' eventuale sconto.
- DurataAbbonamento: dato un periodo, restituisce la durata corrispondente dell' abbonamento reale in mesi.

```
DROP FUNCTION IF EXISTS `NumeroIscritti`;
DROP FUNCTION IF EXISTS `CostoAbbonamento`;
DROP FUNCTION IF EXISTS `DurataAbbonamento`;

DELIMITER |
CREATE FUNCTION `NumeroIscritti` (di date,df date) RETURNS int
BEGIN
        DECLARE risultato int;
        SELECT COUNT (C.CF)
        FROM Composizione AS CM JOIN AbbonamentoReale AS AR ON
        CM.abbonamento=AR.abbonamento JOIN Cliente AS C ON C.IdAbbonamento=AR.abbonamento
        && C.inizioA=AR.dataI && C.fineA=AR.dataF
        WHERE di<=AR.dataI && df>=AR.dataF
        INTO risultato;
RETURN risultato;
END |

CREATE FUNCTION `CostoAbbonamento` (cf char(16),durata int) RETURNS decimal(6,2)
BEGIN
        DECLARE risultato decimal(6,2);
        SELECT DISTINCT SUM (CS.costo*AR.sconto*(DurataAbbonamento(CL.inizioA,CL.fineA)))
        FROM Cliente AS CL JOIN AbbonamentoReale AS AR ON CL.IdAbbonamento=AR.abbonamento
        && CL.inizioA=AR.dataI && CL.fineA=AR.dataF JOIN Composizione AS CM ON
        AR.abbonamento=CM.abbonamento JOIN Corso AS CS ON CM.corso=CS.Nome
        WHERE CL.CF=cf
        INTO risultato;
RETURN risultato;
END |
```

```
CREATE FUNCTION `DurataAbbonamento` (dataI date, dataF date) RETURNS int
BEGIN
        DECLARE risultato int;
        SET risultato=0;
        IF DATEDIFF(dataF,dataI)>=85 &&
        DATEDIFF(dataF,dataI)<=95 THEN
        SET risultato=3;
        ELSEIF
        DATEDIFF(dataF,dataI)>95 &&
        DATEDIFF(dataF,dataI)<=185 THEN
        SET risultato=6;
        ELSEIF
        DATEDIFF(dataF,dataI)>185 &&
        DATEDIFF(dataF,dataI)<=370 THEN
        SET risultato=12;
        END IF;
RETURN risultato;
END |
DELIMITER ;
```

5.3 *Creazione delle procedure e delle viste*

Sono state definite le seguenti query sotto forma di viste, una procedura e altre 2 viste:

- PROCEDURA 'NumeroPartecipanti': dato 'n' in input, restituisce i corsi con almeno 'n' persone iscritte;
- VISTA 1 'QUERY1': restituisce il CF, nome e cognome dei clienti che seguono almeno 2 corsi (di cui uno sala pesi) e che sono seguiti da un PT;
- VISTA 2 'QUERY2': restituisce il CF, nome, cognome e il numero di giorni degli allenatori che sono presenti in palestra almeno 4 giorni;
- VISTA 3 'nPartecipanti': restituisce per ogni corso, il numero di persone iscritte attualmente;
- VISTA 4 'MaxPart': restituisce il maggior numero di iscritti di tutti i corsi;
- VISTA 5 'QUERY3': restituisce il CF, nome e cognome delle persone che sono iscritte nel corso più frequentato;
- VISTA 6 'QUERY4': restituisce CF, nome, cognome e il costo dell' attuale abbonamento di tutti i clienti iscritti che stanno pagando l'abbonamento più costoso di sempre;
- VISTA 7 'QUERY5': restituisce per ogni allenatore che è anche cliente il codice fiscale, nome, cognome, corso nel quale allena, numero di

partecipanti. Se allena nella sala pesi e il numero totale di ripetizioni di una scheda supera il tempo totale di recupero, tale query restituisce oltre a tali valore, anche il numero della relativa scheda. Se non allena nella sala pesi, questi 3 campi sono pari a zero.

```
DROP PROCEDURE IF EXISTS `NumeroPartecipanti`;

DELIMITER |
CREATE PROCEDURE `NumeroPartecipanti` (IN n int)
BEGIN
SELECT CM.corso, COUNT(*) AS nPartecipanti
FROM Composizione AS CM JOIN Cliente AS C ON CM.abbonamento=C.IdAbbonamento
GROUP BY CM.corso
HAVING COUNT(*)>=n;
END|
DELIMITER ;
```

OUTPUT con n=5

```
+-----------+---------------+
| corso     | nPartecipanti |
+-----------+---------------+
| Pilates   |             5 |
| Sala Pesi |             6 |
| Zumba     |             7 |
+-----------+---------------+
3 rows in set (0.00 sec)
```

```
DROP VIEW IF EXISTS `QUERY1`;

CREATE VIEW `QUERY1` (`CF`, `Nome`, `Cognome`) AS
SELECT CL.CF, CL.nome, CL.cognome
FROM AbbonamentoReale AS AR JOIN Composizione AS CM ON AR.abbonamento=CM.abbonamento
JOIN Cliente AS CL ON CL.IdAbbonamento=AR.abbonamento && CL.inizioA=AR.dataI JOIN Scheda AS S
ON S.cfCliente=CL.CF JOIN Allenatore AS A ON S.cfAllenatore=A.CF
GROUP BY CL.CF, CL.nome, CL.cognome
HAVING COUNT(*)>1
ORDER BY CL.cognome ASC;
```

OUTPUT

```
+------------------+-----------+----------+
| CF               | Nome      | Cognome  |
+------------------+-----------+----------+
| DLCVNT83P02H625U | Valentino | De Luca  |
| GNVPLA95B14B563Q | Paolo     | Genovese |
| LCCFNC96A10B524P | Francesco | Lucchese |
| TRNFRC96C12C743E | Federico  | Trentino |
+------------------+-----------+----------+
4 rows in set (0.00 sec)
```

```
DROP VIEW IF EXISTS `QUERY2`;

CREATE VIEW `QUERY2` (`CF`, `Nome`, `Cognome`, `nGiorni`) AS
SELECT DISTINCT A.CF,A.nome,A.cognome,COUNT(*) AS nGiorni
FROM  Allenatore AS A JOIN Programmazione AS P ON A.corso=P.corso
WHERE A.CF <> ALL (SELECT CF FROM Cliente)
GROUP BY A.CF,A.nome,A.cognome
HAVING COUNT(*)>3
ORDER BY A.cognome ASC;
```

OUTPUT

```
+------------------+-----------+---------+---------+
| CF               | Nome      | Cognome | nGiorni |
+------------------+-----------+---------+---------+
| BRTRRT90S05H625C | Roberto   | Bertolo |       4 |
| NOIFNC88C14G224O | Francesco | Onio    |       6 |
| ZNLFPP89L30I008X | Filippo   | Zanella |       4 |
+------------------+-----------+---------+---------+
3 rows in set (0.00 sec)
```

```
DROP VIEW IF EXISTS `nPartecipanti`;

CREATE VIEW `nPartecipanti` (`Corso`,`Numero`) AS
SELECT CM.corso, COUNT(CM.corso) AS num
FROM AbbonamentoReale AS AR JOIN Composizione AS CM ON AR.abbonamento=CM.abbonamento
JOIN Cliente AS C ON AR.abbonamento=C.IdAbbonamento && AR.dataI=C.inizioA && AR.dataF=C.fineA
GROUP BY CM.corso;

DROP VIEW IF EXISTS `MaxPart`;

CREATE VIEW `MaxPart` (`Numero`) AS
SELECT MAX(nPartecipanti.Numero)
FROM nPartecipanti;

DROP VIEW IF EXISTS `QUERY3`;

CREATE VIEW `QUERY3` (`CF`, `Nome`, `Cognome`) AS
SELECT C.CF, C.nome, C.cognome
FROM Cliente AS C
WHERE EXISTS (
        SELECT *
        FROM MaxPart,Composizione AS CM,AbbonamentoReale AS AR,nPartecipanti AS NP
        WHERE CM.corso=NP.Corso &&NP.Numero=MaxPart.Numero &&
CM.abbonamento=C.IdAbbonamento && AR.dataI=C.inizioA && AR.dataF=C.fineA &&
AR.abbonamento=C.IdAbbonamento)
ORDER BY C.cognome ASC;

OUTPUT
+------------------+-----------+----------+
| CF               | Nome      | Cognome  |
+------------------+-----------+----------+
| DLCVNT83P02H625U | Valentino | De Luca  |
| FRRVVN95S30C743Y | Viviano   | Ferrari  |
| GNVPLA95B14B563Q | Paolo     | Genovese |
| MRCFPP89S11H625L | Filippo   | Marchesi |
| PSNCLT94R67G224O | Carlotta  | Pisano   |
| RSSLCU90E06B563Q | Luca      | Russo    |
| TRNFRC96C12C743E | Federico  | Trentino |
+------------------+-----------+----------+
7 rows in set (0.00 sec)

DROP VIEW IF EXISTS `QUERY4`;

CREATE VIEW `QUERY4` (`CF`,`Nome`,`Cognome`,`CostoMax`) AS
SELECT C.CF, C.nome, C.cognome, CostoAbbonamento(C.CF) AS Costo
FROM Cliente AS C
WHERE CostoAbbonamento(C.CF)>=ALL(
        SELECT CostoAbbonamento(C.CF)
        FROM AbbonamentoReale AS AR1
        WHERE C.IdAbbonamento=AR1.abbonamento
        )
ORDER BY Costo DESC;

OUTPUT
+------------------+-----------+-----------+----------+
| CF               | Nome      | Cognome   | CostoMax |
+------------------+-----------+-----------+----------+
| GNVPLA95B14B563Q | Paolo     | Genovese  |   921.48 |
| RSSLCU90E06B563Q | Luca      | Russo     |   699.84 |
| FRRVVN95S30C743Y | Viviano   | Ferrari   |   538.38 |
| MRCFPP89S11H625L | Filippo   | Marchesi  |   502.32 |
| PSNCLT94R67G224O | Carlotta  | Pisano    |   335.16 |
| RSSLNR85M66B563U | Eleonora  | Rossi     |   335.16 |
| DLCVNT83P02H625U | Valentino | De Luca   |   269.10 |
```

```
| TRNFRC96C12C743E | Federico  | Trentino  |   209.40 |
| SBBGLI91P64G224P | Giulia    | Sabbatini |   161.46 |
| LCCFNC96A10B524P | Francesco | Lucchese  |   149.40 |
+------------------+-----------+-----------+----------+
10 rows in set (0.01 sec)
```

```
DROP VIEW IF EXISTS `QUERY5`;

CREATE VIEW `QUERY5`
(`CF`,`Nome`,`Cognome`,`Corso`,`NumeroPartecipanti`,`Scheda`,`RipetizioniTotali`,`RecuperoTotale`)
AS
SELECT A.CF, A.nome, A.cognome, NP.Corso, NP.Numero, F.scheda, SUM(F.nRip)*COUNT(F.scheda),
SUM(F.tempRec)
FROM Formazione AS F JOIN Allenatore AS A ON F.cfAllenatore=A.CF JOIN nPartecipanti AS NP ON
A.corso=NP.Corso
WHERE EXISTS(
        SELECT DISTINCT*
        FROM Cliente AS C
        WHERE A.CF=C.CF)
GROUP BY A.CF, A.nome, A.cognome, F.scheda
HAVING SUM(F.nRip)*COUNT(F.scheda) > SUM(F.tempRec)
UNION
SELECT AL.CF, AL.nome, AL.cognome, NP1.Corso, NP1.Numero, 0, 0, 0
FROM Allenatore AS AL, nPartecipanti as NP1
WHERE AL.salaPesi='false' && NP1.Corso<>'SalaPesi' && AL.corso=NP1.Corso;
```

OUTPUT

```
+------------------+-----------+------------+-----------+------
-------------+--------+-------------------+----------------+
| CF               | Nome      | Cognome    | Corso     |
NumeroPartecipanti | Scheda | RipetizioniTotali |
RecuperoTotale |
+------------------+-----------+------------+-----------+------
-------------+--------+-------------------+----------------+
| RSSLNR85M66B563U | Eleonora  | Rossi      | Sala Pesi |
6 | 3      |              1764 |            870 |
| RSSLNR85M66B563U | Eleonora  | Rossi      | Sala Pesi |
6 | 4      |              1090 |            810 |
| BRTRRT90S05H625C | Roberto   | Bertolo    | Pilates   |
5 | 0      |                 0 |              0 |
| CCCLCU89E26C743C | Luca      | Cocci      | Zumba     |
7 | 0      |                 0 |              0 |
| CNTFNC80D66L199R | Francesca | Conte      | Zumba     |
7 | 0      |                 0 |              0 |
| DLCVNT83P02H625U | Valentino | De Luca    | BodyPump  |
4 | 0      |                 0 |              0 |
| MNGCHR85A49B563D | Chiara    | Meneghetti | BodyPump  |
4 | 0      |                 0 |              0 |
| ZNLFPP89L30I008X | Filippo   | Zanella    | Pilates   |
5 | 0      |                 0 |              0 |
+------------------+-----------+------------+-----------+------
-------------+--------+-------------------+----------------+
8 rows in set, 1 warning (0.00 sec)
```

```
SET FOREIGN_KEY_CHECKS=1;
```