Università di Pisa

Corso di laurea in Ingegneria Informatica

Progetto Basi di dati 2016/2017

TAVOLA DEI VOLUMI

Area Gestione

Concetto	Tipo	Volume
Centro	E	10
Sala	E	80
Cliente	E	800000
Documento	E	801500
CaratteristicheFisiche	E	640000
Personale	E	1500
Istruttore	E	500
Medico	E	500
Segreteria	E	500
Responsabilità	R	1350
Contratto	E	6400000
PotenziamentoMuscolare	E	25600000
Rateizzazione	E	6400000
Rata	E	24000000
AbbonamentoStandard	E	3
AutorizzazionePersonalizzata	E	960000
AutorizzazioneCentro	R	960000

STIME :

-Sala (in media 8 sale a palestra)

-Cliente (in media 80000 clienti a palestra)

-Documento(è pari al numero dei clienti e del personale , relazione 1:1)

-CaratteristicheFisiche(pari al numero dei clienti ABBONATI,80 % dei clienti totali, relazione 1:1)

-Personale (1500 , padre delle tre figlie Istruttore,Medico,Segreteria ognuna composta da 500)

-Responsabilità (se si considerano 150 responsabili , si avranno 1350 occorrenze in quanto ogni cliente ha uno e uno solo responsabile e i responsabili ovviamente non ne hanno)

-Contratti(vengono mantenuti anche quelli passati, quindi si considera che in media ogni cliente abbonato dalla sua prima sottoscrizione a oggi abbia stipulato 10 contratti)

-PotenziamentoMuscolare(in media si considera di ogni contratto, corrente o passato, 4 gruppi muscolari scelti)

-Rateizzazione (1:1 con contratto)

-Rata(in media 30 rate per ogni cliente abbonato)

-AbbonamentoStandard (3: silver,gold,platinum)

-AutorizzazionePersonalizzata (si consideri che il 50% degli abbonati fa il personalizzato e che il 10% di questi fa un multisede con tre palestre)

-Lo stesso vale per la standard.

Area Servizi

Concetto	Tipo	Volume
Scheda Alimentazione	E	3200000
Dieta	E	10000
Visita	E	6400000
Misurazione	E	5120000
AccessoCentro	E	30000000
AccessoSala	E	47500000
Spogliatoio	E	40
Armadietto	E	2400
OrarioAperturaCentro	E	65
Turnazione	E	7500
Corso	E	24000
TipologiaCorso_Sala	E	240
IscrizioneCorsi	E	480000
CalendarioLezioni	E	72000
AccessoPagato	E	7500000
Log_Accesso	E	2000
Log_Sala	E	1600
Log_SalaMedica	E	100
Log_AccessoPagato	E	500

Stime:

-Le schede di alimentazione sono mantenute nel database anche una volta cambiate per cui il carico sarà piuttosto alto , si considera in media ogni cliente abbonato (che sono 640000) abbia avuto 5 schede di alimentazione.

-Per quanto riguarda le diete , un medico spesso assegna a soggetti simili delle diete identiche per cui il carico sarà nettamente inferiore alle schede.

-Per le visite stessa considerazione delle schede, ma si consideri che dopo una visita non sempre si cambierà la scheda di alimentazione, per cui il numero di visite è maggiore.

-Le misurazioni sono di numero inferiore alle visite perché esse possono essere effettuate solo durante una visita, ma con l'implementazione delle sfide l'utilizzo delle misurazioni per le performance sportiva è molto frequente.

-Gli accessi hanno un carico importante, un cliente che accede a un centro può poi accedere a più sale , per cui AccessoSala>AccessoCentro.AccessoCentro inoltre considera solo gli accessi degli abbonati.

-Si considerano 4 spogliatoi per ogni centro: 2 per il personale e 2 per i clienti, e 60 armadietti a spogliatoio.

-Non tutti i centri terranno aperto la domenica per cui il numero di occorrenze sarà minore di 70 (7 giorni per 10 centri).

-Turnazione : in media il personale lavorerà 5 giorni a settimana (1500 * 5).

-TipologiaCorso_Sala (in media ogni sala tiene 3 corsi diversi)

-Corsi : in media ogni sala 3 corsi , ma considerando che si tiene conto anche di corsi passati magari per effettuare dei confronti si considera un moltiplicatore come 100 (240*100*3).

-IscrizioneCorsi: Ogni corso avrà avuto in media 20 iscritti quindi si consideri il numero di occorrenze uguale a quello dei Corsi*20.

-CalendarioLezioni: Con una media di tre volte a settimane il numero di occorrenze del calendario sarà uguale a quello dei Corsi*3.

-Le log tengono conto degli accessi fino al momento dell'uscita del cliente.

Monitoraggio allenamento

Concetto	Tipo	Volume
SchedaAllenamento	E	3000000
Esercizio	E	10000
EsercizioScheda	R	24000000
Attrezzatura	E	15000
Esercizio_Attrezzatura	R	10000
Esercizio_Configurazione	E	20000
EsercizioSvolto	E	96000000
EsercizioSvolto_Configurazione	E	120000000
MV_Performance	E	500000

Stime :

-SchedaAllenamento contiene non solo le schede nuove ma anche quelle passate per cui il carico è notevole come in generale le tabelle di quest' area.

-Si considera una gamma di 10000 possibili esercizi e 15000 possibili attrezzature.

- Ogni scheda in media si articola in tre giorni diversi e ogni giorno è composto in media da 6 esercizi da svolgere. Si considera quindi 3000000*18.

-Non tutti gli esercizi richiedono attrezzatura quindi consideriamo in media un rapporto 1:1 tra l'attrezzatura e gli esercizi.

-A sua volta non tutti gli attrezzi hanno una configurazione , ma molto più spesso ne hanno anche più di una si considera stavolta invece un rapporto 2:1 tra configurazione degli attrezzi e l'attrezzatura stessa.

-Gli esercizi svolti hanno un carico notevole e ovviamente maggiore delle schede erogate dal centro , così come a sua volta la configurazione avrà un peso anche maggiore della prima.

-Da questo carico nasce l'esigenza di un Materialized View che permetta l'analisi delle performance senza dover far troppe operazioni di lettura. Si consideri quindi una tabella che verrà svuotata mensilmente per l'analisi delle performance dell'atleta.Per cui considerando 500000 occorrenze di esercizi , per ogni centro se ne avranno 50000 , ogni scheda per giorno ha 6 esercizi per cui facendo 50000/(30*6) = 277 circa, cioè ogni giorno in una palestra vengono svolte 277 schede d'allenamento (poi ovviamente ci saranno anche esercizi sbagliati ma come stima in eccesso è sufficiente).

Aree Allestibili

Concetto	Tipo	Volume
AreaAllestibile	E	40
TariffeAreeAllestibili	E	240
Prenotazione	E	1600000
Partecipante	R	4800000
PrenotazioneAlternativa	E	240000
SaldoAreeAllestibiliCliente	R	480000
SaldiAreeAllestibiliDaPagare	E	1920000

Stime :

-Si considera 4 aree per centro

-Per ogni area in media 4 tipi diversi di allestimento

-In media per ogni area 100000 prenotazioni memorizzate nel database (16 * 100000)

-In media ogni prenotazione prende 3 partecipanti(3*160000)

-Consideriamo che il 5% delle prenotazioni abbia come stato ‘alternativa’ e quindi ad essa corrisponda tre prenotazioni alternative (quindi 3*80000=240000)

-Si consideri che il 60% dei clienti usufruisca delle aree allestibili e quindi abbia il proprio saldo cliente.

-Ogni cliente che usufruisce del servizio ha in media 4 saldi da pagare (una volta pagato il saldo si cancella , quindi 4 * 480000

Integratori

Concetto	Tipo	Volume
Fornitore	E	30
Integratore	E	4500
Ordine	E	3500
OrdiniEvasi	E	3000
Acquisto	E	35000
Magazzino	E	10
MerceMagazzino	E	420000
InventarioMagazzino	R	350
Vendite	E	500000
Log_VenditeIntegratori	E	7000
ReportIntegratori	E	10

Stime :

-Gli integratori sono 4500 perché si considera in media una gamma di 150 diversi integratori per fornitore (magari gli stessi però con prezzi diversi).

-Gli ordini sono in numero maggiore degli ordini evasi perché viene comunque tenuta traccia dell'ordine evaso fino alla sua conclusione(buona o meno che sia) nella tabella Ordine, dove vi sono anche gli Ordini "incompleti", cioè quelli non ancora evasi.

-Il numero degli acquisti è stato calcolato come il numero degli ordini in giacenza nel database (evasi e non) quindi 3500 per il moltiplicatore 10 perché ogni ordine ovviamente compie diversi acquisti.

-I magazzini sono uno per centro quindi 10.

-La merce magazzino è ovviamente in numero maggiore a quello ordinata , considerando il ricambio di merce tra quella entrante e venduta consideriamo approssimativamente che essa è in numero 4 volte più grande di quella ordinata (gli ordini solitamente sono mensili e in genere la merce dura in media 3 o 4 mesi).

-InventarioMagazzino tiene conto della quantità di ogni tipo di integratore in magazzino, quindi si considera il numero di occorrenze pari al prodotto del numero di centri(ognuno ha un solo inventario) per 35, gamma di integratori.

-Vendite tiene traccia delle vendite vecchie e nuove quindi è una tabella molto carica di occorrenze (per questo ha il Log).

-Il Log ovviamente ha un carico nettamente inferiore , su di esso viene fatto un reporting mensile , quindi si svuota ogni mese.

Area Social

Concetto	Tipo	Volume
ProfiloSocial	E	641500
AreaForum	E	40
PostPrincipale	E	20000000
PostRisposta	E	160000000
RichiestaAmicizia	R	20000000
Amicizia	R	10000000
Interesse	E	3849000
Cerchia	E	2566000
Consigliati	R	51320000
ComposizioneCerchia	R	38490000
Giudizio	R	320000000
Sfida	E	2566000
VincitoriSfida	R	2952540
AderisciSfida	R	15396000
ConcludiSfida	R	12830000
SchedaAlimentazioneModificata	E	800000
EsercizioModificato	E	13000000
EsercizioModificatoConfigurazione	E	13000000

STIME :

-Ogni cliente abbonato ha un profilo social così come i tutor per cui si considera un rapporto 1:1 tra il numero di account e quello dei clienti più i tutor.

-I post principale si considera un numero elevato come 20000000 e una media di 8 post di risposta a post principale.

-Anche per le richieste d'amicizia si considera sempre indicativamente un numero come 20000000 e un rapporto 2:1 con le amicizie, cioè solo la metà siano andate a buon fine.

-Si consideri 6 interessi a profilo social e 4 cerchie ogni profilo.

-Per i consigliati stimando un numero medio di 150 amici a persona (all' incirca amicizia/profilosocial) si stima una lista di 80 consigliati per le proprie cerchie per ogni profilo social.

-Si consideri 15 partecipanti a cerchia (che essendo in media 4 a profilo social) implicando così un numero di record di composizione cerchia pari a 60* NumeroProfiliSocial.

-Si stima 2 giudizi ogni post di risposta.

-In media ogni profilo social ha lanciato 4 sfide .

-Per il calcolo del numero di occorrenze di vincitori si considera per gli ex aequo un molitplicatore pari a 1,2 (quindi per ogni sfida non c'è un solo vincitore) e considerando che non tutte le sfide sono ancora concluse (diciamo un 5%) l'operazione da fare è moltiplicare il numero di sfide *0,95*1,2.

-AderisciSfida considera 6 aderenti alla sfida per ognuna di esse.

-ConcludiSfida considera come moltiplicatore per sfida 5 invece che 6 in quanto non tutti i partecipanti di una sfida possono ancora aver concluso.

-Gli esercizi sfida sono modificati da una scheda ma non tutte le schede sono poi state modificate nelle sfide, per cui si prende il numero di schede /3 * un moltiplicatore come 13(in quanto non tutti e 18 gli esercizi di una scheda devono essere necessariamente modificati ma solo alcuni).

-Per semplicità le configurazioni modificate sono pari a quelle degli esercizi perché in alcuni esercizi non ci sta configurazione mentre in altri anche più di una.

In media si considera che un quarto delle schede di alimentazione siano state modificate.

TAVOLA DELLE OPERAZIONI

Operazione	Tipo	Frequenza
InserisciAutorizzazioneStandard	I	50 al giorno
EffettuaAccessoCentro	I	2000 al giorno
InserisciCorso	I	20 al mese
AnalisiPerformanceSportiva	I	16500 al giorno
AggiungiPartecipante	I	8000 a settimana
ConcludiPrenotazione	I	1000 a settimana
AccettaRifiutaRichiesta	I	1000 a settimana
CaricaSaldoDaPagare	I	1000 a settimana
AggiungiAlCarrello	I	1000 al mese
InviaOrdine	I	1000 al mese
VendiIntegratore	I	100 al giorno
DecrementaMagazzino	I	100 al mese
EsprimiGiudizioPost	I	1000 al giorno

InserisciAutorizzazioneStandard

Descrizione

L'operazione nella sua semplicità mostra a pieno titolo le semplici operazioni che vengono svolte su una parte della gestione della palestra.

Essa è effettuata da una stored procedure che permette l'inserimento di un' autorizzazione standard(quindi relativa a contratti di tipo standard , cioè gold,platinum e silver) al centro dopo aver controllato che il numero di autorizzazioni a centri non sia maggiore o uguale di 3 e che il contratto passato alla procedure non sia di tipo personalizzato.

La procedure effettua quindi le seguenti operazioni:

-ricava la tipologia del contratto dalla tabella Contratto e il prezzo corrente da pagare al mese per quell'abbonamento(corrente perché poi ovviamente aggiungendo una nuova autorizzazione il prezzo verrà incrementato).

- prende il prezzo di quel tipo di abbonamento dalla tabella AbbonamentoStandard così da poter poi vedere tramite una semplice divisione (il prezzo del contratto / il prezzo dell'abbonamento) il numero di autorizzazioni a centri già registrate per quel contratto per verificare che non si sia già raggiunto il numero massimo di autorizzazioni per contratto.

-Effettua l'inserimento in AutorizzazioneCentro

-Se questo va a buon fine aggiorna il prezzo del contratto.

INPUT : Contratto,Centro

OTPUT: -

PORZIONE DELLA TAVOLA DEI VOLUMI INTERESSATA :

Concetto	Tipo	Volume
Cliente	E	800000
Contratto	E	6400000
AbbonamentoStandard	E	240
AutorizzazioneCentro	R	800000

TAVOLA DEGLI ACCESSI :

Tipo	Costrutto	Accessi	Nome	Breve Descrizione
L	E	1	Contratto	Legge la tipologia del contratto e calcola l'importo mensile da pagare.
L	E	1	AbbonamentoStandard	Lettura per ricavare il prezzo singolo dell'abbonamento.
S	R	1	AutorizzazioneCentro	Inserimento nel caso sia possibile.
S	E	1	Contratto	Update sull'importo pagato per l'abbonamento

EffettuaAccessoCentro

Descrizione

Dopo aver visto come si autorizza l'accesso di un utente a un determinato centro vediamo poi come l'accesso viene effettivamente registrato nel database.

L'azione viene effettuata attraverso una stored procedure chiamata RegistraAccesso_Centro.

La procedure ha il compito principale di registrare l'accesso al centro del cliente non appena supera il tornello di ingresso. Per prima cosa si trova il primo armadietto libero da assegnare al cliente, cosicchè possa depositare i suoi beni. Se nessun armadietto è disponibile allora il cliente ne rimarrà senza, altrimenti si entra in un LOOP, che randomizza una password creando una per volta gli 8 valori casuali che la compongono.

Finito il ciclo, si inserisce dentro il Log_Accesso e ci rimarrà finchè il cliente non esce.

Il tutto però innesca un BEFORE trigger ControllaAccessiCentro che controlla prima dell'inserimento dell'accesso che esso sia effettivamente valido.

INPUT : Cliente,Centro,DataAccesso,OrarioAccesso.

Criteri di validità e possibili output di errore:

-controlla che ci sia un'autorizzazione per quel centro → " Il cliente non ha il permesso di entrare nel centro!"

-controlla che prima dell'accesso sia stata effettuata una visita nel periodo compreso tra la data della sottoscrizione del contratto e la sua data di scadenza , nessun cliente se non preventivamente visitato può accedere alla palestra →

"Il cliente non ha ancora effettuato alcuna visita"

-l'esistenza di un tornello a parte per la sala medica obbliga un controllo sul fatto che il cliente potrebbe essere nel centro ma nella sala medica → "Errore di sistema,il cliente si trova nella sala medica"

-controlla il numero di accessi settimanali → "Numero accessi settimanali superati !

PORZIONE DELLA TAVOLA DEI VOLUMI INTERESSATA :

Concetto	Tipo	Volume
Cliente	E	800000
Contratto	E	6400000
AbbonamentoStandard	E	240
AutorizzazioneCentro	R	800000
AutorizzazionePersonalizzata	E	960000
Armadietto	E	2400
Spogliatoio	E	40
Log_Accesso	E	2000
OrarioAperturaCentro	E	65
Visita	E	6400000
Log_SalaMedica	E	100
AccessoCentro	E	30000000

TAVOLA DEGLI ACCESSI :

Tipo	Costrutto	Accessi	Nome	Breve Descrizione
L	E	1	Cliente	Sesso del cliente
L	E	1	Armadietto	INNER JOIN con Spogliatoio per il codice dell'armadietto. L'occorrenza è una sola per il LIMIT 1.
L	E	1	Spogliatoio	-
S	E	1	Log_Accesso	Inserimento
S	E	1	Armadietto	Setta l'attributo occupato=1(solo in caso l'armadietto sia disponibile).
L	E	11	Contratto	Sceglie l'ultimo contratto stipulato del cliente e ancora valido(media dei contratti stipulati a cliente 10 , 1 lettura è della query esterna).
L	R/E	1	AutorizzazioneCentro/ AutorizzazionePersonalizzata	Controlla che ci sia un'autorizzazione per quel centro.
L	E	1	Visita	Controllo sulla visita
L	E	1	Log_SalaMedica	Controllo sala medica
L	E	1	AbbonamentoStandard/ AutorizzazionePersonalizzata	Prende il numero di ingressi settimanali.
L	E	7	AccessoCentro	Conta il numero di volte che già ha fatto accessi quella settimana. INNER JOIN con Contratto per il controllo sul numero di accessi

				settimanali. Caso peggiore , 7 accessi a settimana con un contratto personalizzato.

InserimentoCorso

Descrizione

I corsi vengono inseriti in maniera meno complessa, ma ciò non toglie che anch'essi debbano passare dei selettivi controlli prima di essere inseriti nel database.

L'inserimento effettuato mediante un semplice INSERT aziona un Trigger ControllaCorsi1 che effettua tutto quello che è necessario affinchè un corso registrato sia consistente con il resto del sistema.

In primis verifica che la sala del corso sia attinente sia alla tipologia del corso sia al tutor scelto a dirigere il corso.In caso almeno una delle due condizioni non siano verificate viene lanciato un messaggio d'errore : "Disciplina non attinente al tipo sala o non esistente!"

In secondo luogo il trigger controlla che non sia ancora in corso un attività identica a quella che si vuole inserire.

Ma quando due corsi possono considerarsi identici?Quando essi hanno in comune :

-lo stesso tutor

-la stessa sala

-la stessa disciplina

-lo stesso livello di insegnamento

Esempio : possono quindi essere introdotti dei corsi di una certa disciplina dalla stessa palestra ma magari di un livello avanzato per persone più esperte.

Nel caso l'inserimento venga bloccato il segnale d'errore è : " Corso già esistente!"

L'ultimo controllo riguarda il tutor : esso controlla che il tutor lavori effettivamente nel centro , e ciò è essenziale per stabilire poi tramite altre operazioni il calendario del corso che si basa sui turni del tutor che lavora per quel centro.

In quest'ultimo caso l'eventuale messaggio d'errore è : "Il tutor non lavora nel centro!".

PORZIONE DELLA TAVOLA DEI VOLUMI INTERESSATA :

Concetto	Tipo	Volume
TipologiaCorso_Sala	E	240
Corso	E	24000
Sala	E	80
Turnazione	E	7500

TAVOLA DEGLI ACCESSI :

Tipo	Costrutto	Accessi	Nome	Breve Descrizione
L	E	1	TipologiaCorso_Sala	Controlla che la disciplina esista e sia attinente alla sala.
L	E	1	Corso	Controlla che non esista un corso (ancora in corso)nella stessa sala,con lo stesso nome, con lo stesso istruttore,con lo stesso livello di insegnamento.
L	E	1	Sala	Prende il centro.
L	E	3	Turnazione	Controlla che il tutor lavori almeno una volta nel centro. 3 occorrenze perché in media un tutor svolge tre turni per centro.
S	E	1	Corso	Inserimento se tutte le condizioni vengono superate.

AnalisiPerformanceSportiva (Funzionalità Analytics)

Descrizione

Le seguenti stored procedure sono volte ad analizzare le criticità di una scheda di allenamento , ovvero a stabilire se gli esercizi in essa contenuti siano o meno troppo pesanti per il cliente intestatario della scheda, e se vadano o meno ricalibrati ,mentre il come viene lasciato alle competenze tecniche del tutor per cui ci si focalizza invece su come dare al tutor tutte le informazioni possibili per stabilire quali punti vadano sistemati e quali invece no.

Quali sono i possibili fattori che vanno a degradare l'efficacia di un allenamento?

-Tempi di recupero molto lunghi.

-Tempi di recupero troppo corti (che rischiano di mandare in affanno il cliente poi nell'esercizio).

-Configurazione errata degli esercizi proposti, cioè un errore nel regolare la macchina utilizzata, o addirittura l'utilizzo di un'attrezzatura diversa da quella indicata dal tutor.

-Eccessiva intensità di un allenamento che quindi non può essere portato a termine (o che magari viene portato a termine con eccesive pause o pause molto lunghe per via della stanchezza che vanno comunque a intaccare il risultato della performance).

O al contrario si deve segnalare se invece l'esercizio è troppo semplice per quel cliente che magari ha svolto non solo quello che doveva fare ma lo ha compiuto con un intensità maggiore o prolungata rispetto a quanto richiesto.

Le seguenti analisi sono possibili grazie ai particolari sensori di cui sono dotati sia le attrezzature della palestra sia le tecnologie, fornite dalla palestra ai clienti, che permettono al cliente di sapere istantaneamente il risultato della propria performance e di poter fare quindi nella maniera più semplice e possibile un'autoanalisi dell'esercizio appena svolto andando a chiamare queste stored permettendo sia a lui che al tutor di vedere dove migliorare.

Detto ciò bisogna stabilire degli intervalli entro cui muoversi per far sì che la performance venga ritenuta buona , ottima o se invece sia da segnalare perché da correggere.

Le considerazioni alla base di queste stored, create nella versione Aerobico e Anaerobico , sono le seguenti :

il margine di tolleranza per la durata e il numero totale di ripetizioni(cioè il numero delle ripetizioni moltiplicate per quello delle serie) così come il tempo di recupero possono discostare di un valore pari al

15% del valore ottimale garantendo un discreto gap di errore per quanto riguarda la performance sportiva; meno tolleranza invece si ha per la configurazione degli attrezzi , in quanto essa viene ritenuta solo giusta o sbagliata (cioè non si discrimina tra un errore più grave come lo sbagliare il tipo di configurazione e un errore meno pesante ma comunque non da sottovalutare come sbagliarne il valore).

La rigidità su questo ultimo parametro è dovuto al fatto che esso sta alla base dell'esercizio .

Facciamo un esempio: l'esercizio sulla panca va a colpire differenti gruppi muscolari a seconda della sua inclinazione per cui sarebbe altamente deleterio lavorare su un muscolo che non si doveva allenare in quel giorno della scheda.

Passiamo ora al lato pratico , l'implementazione è la seguente :

-in primis va detto che le stored vanno a riempire una log con dei giudizi sulla performance che vengono visualizzati dal tutor o dal cliente quando necessario.

- i parametri dell'esercizio svolto sono passati alla funzione direttamente come argomenti.

-i parametri dell'esercizio ottimale vengono estrapolati con un ciclo per la configurazione in cui si controlla una a una se tutte le configurazioni richieste sono state utilizzate nell'esercizio svolto e in caso affermativo si controlla se sono state utilizzate bene. Nel ciclo non si entra in caso l'esercizio non richieda attrezzatura.

-La durata, le serie , gli intervalli e il recupero si risolvono con un semplice CASE statement settando prima una variabile che prende i valori delle durate, serie , intervalli e recupero di riferimento e poi facendo il confronto per vedere se si rientra o meno nei margini di tolleranza illustrati prima.

- L'azione conclusiva è quella di riempire la tabella MV_Performance con delle stringhe che rappresentano dei giudizi espressi sulla base dei dati raccolti riguardo tre ambiti : tempi di recupero, configurazione e svolgimento dell'esercizio(se è stato portato a termine o meno, o se solo in parte , o se si è fatto anche più del necessario).Un caso particolare è se l'esercizio non rispetta la scheda di allenamento in quel caso i tre giudizi sono settati a null e un altro attributo evidenzia l'incompatibilità tra la scheda del giorno e quell'esercizio.

INPUT :

Cliente,Esercizio,IstanteInizio,SchedaAllenamento,Durata,Recupero,GiornoScheda.

OUTPUT :

Possibili segnali di errori:

-"Attenzione,inserito esercizio non valutabile" quando l'esercizio non è compatibile con quel determinato giorno di allenamento della scheda.

Giudizi allenamento (come verrà riempita la log table):

CONFIGURAZIONE

-"Configurazione con attrezzi corretta" se o non si è scelta la giusta attrezzatura o la giusta configurazione dell'attrezzo.

-"Configurazione con attrezzi errata" se si è sia scelta la corretta attrezzattura che la corretta configurazione

-"Esercizio senza attrezzi" se l'esercizio non richiede attrezzi e quindi non richiede un giudizio in questo caso su questa parte dell'allenamento.

DURATA (solo per esercizio aerobico)

-"Ok,tempo relativamente giusto" se il tempo di scarto dall'ottimale rientra nei margini di tolleranza.

-"Attenzione,tempo inferiore all'ottimale"se l'esercizio è durato troppo poco rispetto all'ottimale.

-"Ottimo,tempo superiore all'ottimale" se l'atleta è riuscito a protrarre nel tempo l'esercizio ancor di più dell'ottimale.

SERIE DI RIPETIZIONI (solo per esercizio anaerobico)

-"Ok,numero ripetizioni adeguato" se il gap tra il numero di ripetizioni fatte e quello ottimale rientra nei margini di tolleranza.

-"Attenzione,numero ripetizioni inferiore al richiesto" se il numero di ripetizioni fatte è tanto inferiore a quello richiesto dell 'esercizio da uscire dal gap di tolleranza.

-"Ottimo,numero ripetizioni superiore al richiesto" se si è riuscito a fare più ripetizioni del dovuto.

TEMPO DI RECUPERO

-"Ok,tempo relativamente giusto" se il tempo di recupero non si discosta troppo da quello indicato dal tutor.

-"Con calma,tempo inferiore all'ottimale" se è stato troppo corto il tempo di riposo.

-"Attenzione,tempo superiore all'ottimale" se è stato troppo lungo il tempo di recupero.

INSERIMENTO

Cosa va a finire nella LogPerformance?

-IstanteInizio (passato come argomento)

-Cliente (passato come argomento)

-Esercizio (passato come argomento)

-commentorecupero (valore stringa della variabile creata nella stored)

-commentoserie (valore stringa della variabile creata nella stored)

-commentoconfigurazione (valore stringa della variabile creata nella stored)

In caso di incompatibilità tra esercizio e giornoscheda l'inserimento è il seguente:

(_IstanteInizio,_cliente,"Esercizio non compatibile con la scheda" ,NULL,NULL,NULL).

PORZIONE DELLA TAVOLA DEI VOLUMI INTERESSATA :

Concetto	Tipo	Volume
Esercizio_Configurazione	E	200000
EsercizioScheda	R	24000000
EsercizioSvolto_Configurazione	E	120000000
Esercizio	E	100000
MV_Performance	E	500000

TAVOLA DEGLI ACCESSI

Tipo	Costrutto	Accessi	Nome	Breve Descrizione
L	R	6	EsercizioScheda	Controllo compatibilità esercizio- scheda

S	E	1	*MV_Performance	Se tra i 6 esercizi della scheda non c'è quello svolto allora si fa subito l'inserimento e si esce dalla procedure.
L	E	2	Esercizio_Configurazione	CURSOR: Prende record per record dalla tabella nell'ordine l'Attrezzatura,il TipoConfigurazione e infine ValoreConfigurazione.
L	E	2	Esercizio_Configurazione	Prima di entrare nel LOOP si vede se è necessario giudicare o meno la configurazione controllando che l'esercizio abbia o meno attrezzatura(se non ha configurazione basterà semplicemente che l'attrezzatura sia corretta, ma per fare ciò si entrerà nel LOOP)
L	E	2	EsercizioSvolto_Configurazione	Controllo nel LOOP che ogni attrezzatura ci sia e abbia la configurazione giusta.
S	E	1	MV_Performance	Inserimento dei giudizi nella LogTable

AggiungiPartecipante,ConcludiPrenotazione, AccettaRifiutaRichiesta (AreeAllestibili)

Descrizione

L'operazione di aggiungere un partecipante a un'attività in un'area allestibile avviene mentre la prenotazione è nello stato 'Gruppo in composizione'. Se la prenotazione è stata inviata(cioè 'Da confermare'*) non si possono più aggiungere partecipanti.

Essa viene effettuata mediante una stored procedure (AggiungiPartecipante) che prende in ingresso il cliente e il codice di prenotazione ed effettua diversi controlli prima di effettuare l'effettivo inserimento.

Per l'efficienza delle tre operazioni studiate si sono create due ridondanze in Prenotazione: la prima tiene il conto del numero dei partecipanti(NumPartecipanti), la seconda tiene conto del punteggio del gruppo(che verrà poi spiegato).

NumPartecipanti è coinvolta in AggiungiPartecipante(per il numero max di partecipanti) e ConcludiPrenotazione(in quanto è usata per vedere se si è raggiunto il numero minimo di partecipanti), mentre PunteggioGruppo in AggiungiPartecipante e AccettaRifiutaRichiesta.

*L’unico modo per far sì che la prenotazione sia “Da confermare” è che la prenotazione sia stata effettivamente inoltrata e questo avviene tramite la stored procedure ConcludiPrenotazione.

AggiungiPartecipante

PRIMO CONTROLLO : La prenotazione è già stata inoltrata?

SECONDO CONTROLLO: Il cliente che si vuole aggiungere siamo sicuri non sia già quello che ha prenotato?(che è quindi già un partecipante ed ha quindi già contribuito al punteggio totale del gruppo).

TERZO CONTROLLO : Si è già raggiunto il tetto massimo di partecipanti per quell’area?(La ridondanza quindi evita di contare uno ad uno il numero di occorrenze di partecipanti all’attività)

Effettuati i controlli si procede all’inserimento , ma prima viene chiamata una funzione che calcola il tipo del contratto che ha attualmente il cliente.

La function è TipologiaContrattoCliente e prende in ingresso il Cliente e la Sede e ovviamente è NOT DETERMINISTIC(in istanti temporali diversi il Cliente avrà un diverso contratto con quella sede).

L’implementazione funziona nel seguente modo : due variabili , una per i contratti standard e una per quelli personalizzati, “pescano” rispettivamente su AutorizzazioneCentro e su AutorizzazionePersonalizzata andando quindi ad esaminare se esiste nell’una o nell’altra che l’ultimo contratto fatto sia per quel centro e in caso affermativo anche che esso sia ancora valido.

Se il responso è negativo per entrambe ,allora vengono settate tutte e due a NULL e la stringa ritornata è “senza contratto” , altrimenti ,visto che il caso che entrambe le variabili siano NOT NULL non esiste ,con un semplice ELSE si sceglie quale strada prendere (standard o personalizzato) e in ogni caso la funzione ritorna il tipo di contratto.

La funzione può restituire 5 diverse stringhe a seconda del contratto:

-‘senza contratto’

-‘silver’

-‘gold’

-‘platinum’

-‘personalizzato’

Determinare il contratto è fondamentale per **l’attribuzione del punteggio di gruppo** a sua volta indispensabile per decidere a chi assegnare prima l’area allestibile in quella fascia oraria(si guarda il punteggio di gruppo e in caso di pari merito il timestamp di prenotazione).

Una volta aggiunti tutti i partecipanti e conclusa la prenotazione bisogna vedere se essa verrà accettata o meno.

CRITERIO DI SCELTA (applicato dalla stored procedure AccettaRifiutaRichiesta)

Viene servito prima chi ha il punteggio gruppo più alto di tutti per quell’area , ottenuto sommando il punteggio gruppo del richiedente con quello di tutti i partecipanti al gruppo a partire dal momento dell’inoltro (perché prima era possibile anche disdire e quindi decrementare il punteggio) della prenotazione. In caso di pari merito si guarda l’ordine cronologico , considerando il timestamp di inserimento della prenotazione nel database.

Il criterio soddisfa due politiche di marketing :

-assegnare 5 punti ai senza contratto (e non 0), serve a incentivare le persone a formare gruppi più grandi possibili anche se questo vuol dire aggiungere persone senza contratto o comunque con una priorità bassa dal punto di vista del punteggio , insomma .per quanto bassi siano i punteggi ,essi fanno comunque numero.

-assegnare 20 punti ai personalizzati e 25 ai platinum (che hanno tra l'altro anche degli sconti maggiori) invoglia il cliente a sottoscrivere contratti sempre migliori e quindi le aree allestibili diventano una delle tante spinte per raggiungere questo scopo.

In caso contrario , cioè se la richiesta non viene approvata lo stato della prenotazione viene settato ad "alternativa" perché tanto l'area richiesta è già stata assegnata a qualcun altro**(tutto ciò accade mediante una stored procedure che tiene di tutti questi criteri che è ottimizzata dalla ridondanza PunteggioGruppo in quanto essa evita di dover per ogni prenotazione andare a calcolare la somma del punteggio gruppo del richiedente più quello di ogni singolo partecipante all'attività**).

Determinata la tipologia del contratto si procede all'aggiornamento .

INPUT : Cliente,CodPrenotazione

OUTPUT:

Possibili messaggi di errore

-"La prenotazione è già stata inoltrata"

-"Chi prenota è già un partecipante"

-"L'area non può più ospitare partecipanti"

PORZIONE DELLA TAVOLA DEI VOLUMI INTERESSATA :

Concetto	Tipo	Volume
Contratto	E	6400000
AutorizzazioneCentro	R	320000
AutorizzazionePersonalizzata	E	960000
AreaAllestibile	E	40
Prenotazione	E	1600000
Partecipante	R	4800000

TAVOLA DEGLI ACCESSI AggiungiPartecipante(con ridondanze)

Tipo	Costrutto	Accessi	Nome	Breve Descrizione
L	E	1	Prenotazione	JOIN con AreaAllestibile della Prenotazione. Si prende il numero di partecipanti, lo stato e il clienterichiedente.
L	E	1	AreaAllestibile	Si prende la sede e il tetto massimo di partecipanti per il controllo sul numero di partecipanti.
S	R	1	Partecipante	INSERT
S	E	1	Prenotazione	Update numero partecipanti
L	E	6	Contratto	(FUNCTION) Consideriamo in media che ogni cliente abbia avuto 10 contratti (per semplicità 5 standard e 5

				personalizzati). JOIN con AutorizzazioneCentro sulla query esterna . 1 lettura è sulla query esterna. 5 sulla subquery.
L	R	1	AutorizzazioneCentro	JOIN con Contratto
L	E	6	Contratto	JOIN con AutorizzazionePersonalizzata sulla query esterna. 1 lettura è sulla query esterna. 5 sulla subquery.
L	E	1	AutorizzazionePersonalizzata	-
S	E	1	Prenotazione	UPDATE del PunteggioGruppo.

TAVOLA DEGLI ACCESSI AggiungiPartecipante (senza ridondanze)

Tipo	Costrutto	Accessi	Nome	Breve Descrizione
L	E	1	Prenotazione	JOIN con AreaAllestibile della Prenotazione. Si prende il numero di partecipanti, lo stato e il clienterichiedente.
L	E	1	AreaAllestibile	Si prende la sede e il tetto massimo di partecipanti per il controllo sul numero di partecipanti.
L	E	18	Partecipante	Consideriamo in media che le aree ospitino massimo 20 persone. Consideriamo il caso peggiore cioè che i partecipanti siano 19(18 partecipanti + il cliente richiedente).
S	R	1	Partecipante	INSERT
L	E	6	Contratto	(FUNCTION) Consideriamo in media che ogni cliente abbia avuto 10 contratti (per semplicità 5 standard e 5 personalizzati). JOIN con AutorizzazioneCentro sulla query esterna . 1 lettura è sulla query esterna. 5 sulla subquery.
L	R	1	AutorizzazioneCentro	JOIN con Contratto
L	E	6	Contratto	JOIN con AutorizzazionePersonalizzata sulla query esterna. 1 lettura è sulla query esterna. 5 sulla subquery.
L	E	1	AutorizzazionePersonalizzata	-

TAVOLA DEGLI ACCESSI ConcludiPrenotazione(con ridondanza NumPartecipanti)

Tipo	Costrutto	Accessi	Nome	Breve Descrizione
L	E	1	Prenotazione	JOIN con AreaAllestibile della Prenotazione. Si prende il numero di partecipanti.
L	E	1	AreaAllestibile	Si prende il numero minimo di partecipanti.
S	E	1	Prenotazione	Update stato -> “Da confermare”

TAVOLA DEGLI ACCESSI ConcludiPrenotazione(senza ridondanza NumPartecipanti)

Tipo	Costrutto	Accessi	Nome	Breve Descrizione
L	E	1	AreaAllestibile	Si prende il numero minimo di partecipanti.
L	R	18	Partecipante	Conta il numero di partecipanti.
S	E	1	Prenotazione	Update stato -> “Da confermare”

TAVOLA DEGLI ACCESSI AccettaRifiutaPrenotazione(con ridondanza PunteggioGruppo)

Tipo	Costrutto	Accessi	Nome	Breve Descrizione
L	E	21	Prenotazione	Considerato che in media un’area ha 10 prenotazioni da confermare per essa. 1 per la query esterna. 10 per la subquery che sceglie il massimo tra i punteggi dei gruppi. 10 per la subquery per vedere possibili accavvallamenti temporali con le altre prenotazioni.
L	-	X	-	Chiamata alla funzione ControllaDisponibilitàArea.
S	E	1	Prenotazione	Update stato -> “alternativa ”o “approvata”

TAVOLA DEGLI ACCESSI AccettaRifiutaPrenotazione(senza ridondanza PunteggioGruppo)

Tipo	Costrutto	Accessi	Nome	Breve Descrizione
L	E	21	Prenotazione	Considerato che in media un’area ha 10 prenotazioni da confermare per essa. 1 per la query esterna. 10 per la subquery che sceglie il massimo tra i punteggi dei gruppi.

				10 per la subquery per vedere possibili accavvallamenti temporali con le altre prenotazioni.
L	R	180	Partecipante	18 per ogni prenotazione.Per ogni partecipante viene chiamata la TipologiaContrattoCliente.
L	E	2160 (190*12)	Contratto	190 perché va calcolato il punteggio non solo dei partecipanti ma anche del cliente richiedente.
L	R	190	AutorizzazioneCentro	-
L	E	190	AutorizzazionePersonalizzata	-
L	-	X	-	Chiamata alla funzione ControllaDisponibilitàArea(non presente la ridondanza per cui al fine dello studio di essa non ci interessa).
S	E	1	Prenotazione	Update stato -> “alternativa ”o “approvata”

CaricaSaldoDaPagare

Descrizione

Dopo aver visto il meccanismo della prenotazione e relative ridondanze vediamo invece il pagamento delle aree allestibili.

Le agevolazioni relative al contratto non si limitano a degli enormi vantaggi sulle prenotazioni ma anche sul loro effettivo costo attraverso una FUNCTION (stavolta DETERMINISTIC) che a seconda del contratto ne stabilisce lo sconto sul prezzo finale.

SCONTI

-0% se non si possiede contratto

-20% per i contratti silver

-40% per i contratti gold

-60% per i contratti personalizzati

-80% per i contratti platinum

Quindi sulla linea delle prenotazioni i vantaggi più sostanziosi sono per i contratti personalizzati e platinum.

La procedure funziona in maniera molto semplice :

-dapprima carica il saldo del cliente

-in secondo luogo carica il saldo uno alla volta di tutti i partecipanti con un LOOP

Nel calcolo del saldo si considera la tariffa oraria , la tariffa attrezzatura dell’area , se o meno includere nel prezzo quest’ultima e gli sconti dei contratti ,portando così a **un’alta flessibilità** dei prezzi .

Per tariffa attrezzatura si intende quell'equipaggiamento necessario per svolgere attività particolari come ad esempio il softair dove c'è bisogno di armi , protezioni che possono essere fornite dalla palestra o che il cliente se ne è già in possesso può anche non richiedere.

Il saldo viene inserito con stato "da pagare" .Una volta pagato andrà eliminato dal database tanto il saldo totale e mensile poi è salvato in un'altra tabella , SaldoAreeAllestibiliCliente , che è l'account relativo all'utilizzo delle aree allestibili del cliente.

INPUT : CodPrenotazione

PORZIONE DEL DIAGRAMMA E-R INTERESSATO:

Contratto,AutorizzazioneStandard,AutorizzazionePersonalizzata,AreaAllestibile,Prenotazione,Partecipante,TariffeAree Allestibili,SaldiAreeAllestibiliDaPagare

PORZIONE DELLA TAVOLA DEI VOLUMI INTERESSATA :

Concetto	Tipo	Volume
Contratto	E	6400000
AutorizzazioneCentro	R	320000
AutorizzazionePersonalizzata	E	960000
AreaAllestibile	E	40
Prenotazione	E	1600000
Partecipante	R	4800000
TariffeAreeAllestibili	E	1920000

TAVOLA DEGLI ACCESSI

Tipo	Costrutto	Accessi	Nome	Breve Descrizione
L	E	1	Prenotazione	NATURAL JOIN con TariffeAreaAllestibile e INNER JOIN con AreaAllestibile. Prende clienterichiedente, dataattivita, inizioattivita, fineattivita.
L	E	1	TariffeAreeAllestibili	Prende tariffaoraria e tariffaattrezzatura.
L	E	1	AreaAllestibile	Prende la sede.
L	E	6	Contratto	(FUNCTION TipologiaContrattoCliente sul clienterichiedente) per poi valutarne lo sconto

L	R	1	AutorizzazioneCentro	-
L	E	6	Contratto	-
L	E	1	AutorizzazionePersonalizzata	-
S	E	1	SaldiAreeAllestibiliDaPagare	Viene caricato il saldo di chi ha prenotato.
S	E	1	SaldoAreeAllestibiliCliente	UPDATE sul SaldoMese,SaldoTotale
L	R	3	Partecipante	CURSOR Per ogni partecipante ne ricava il contratto e carica il saldo
L	E	18	Contratto	(FUNCTION TipologiaContrattoCliente su ognuno dei clienti)
L	R	3	AutorizzazioneCentro	-
L	E	18	Contratto	-
L	E	3	AutorizzazionePersonalizzata	-
S	E	3	SaldiAreeAllestibiliDaPagare	Carica i saldi di ogni cliente
S	E	3	SaldoAreeAllestibiliCliente	UPDATE sul SaldoMese,SaldoTotale

InviaOrdine,VenditaIntegratori,DecrementaMagazzino

Descrizione InviaOrdine

La stored procedure riceve come input il codice dell'ordine e la data di consegna preferita ed effettua sostanzialmente due azioni :

-setta lo stato dell'ordine da 'incompleto' a 'evaso'

-inserisce un record in ordinievasi

Per farlo controlla che l'ordine sia ovviamente esistente o che non sia già stato precedentemente inviato.

Il controllo più corposo però è quello che riguarda la capienza del magazzino : il magazzino presenta una ridondanza , CapienzaAttualeVirtuale , che tiene il conto degli stock in magazzino e di quelli ordinati , e tramite essa quindi si controlla che l'ordine venga inviato solo nel caso la merce ordinata possa essere contenuta nel magazzino qual'ora anche nel caso peggiore tutti gli ordini inviati vadano a buon fine e quindi le merci ordinate vadano tutte a magazzino.

Per fare ciò la ridondanza viene aggiornata tramite tre operazioni :

-InviaOrdine , che la incrementa della somma delle quantità dei singoli stock di merce ordinata.

-VenditaIntegratori, che decrementa del valore della quantità di roba venduta.

-DecrementaMagazzino, che effettua un decremento pari alla somma delle singole quantità dei singoli stock di merce ordinata di un ordine fallito passato come input.

Il tutto evita al controllo della InviaOrdine di effettuare accessi in lettura per contare tutta la merce di quel centro in MerceMagazzino e tutta la merce ordinata in Acquisto.

INPUT codordine,dataconsegnapreferita

POSSIBILI OUTPUT D'ERRORE

-"Ordine inesistente"

-“Ordine già inviato”

-“La merce richiesta non entra in magazzino”

Descrizione Vendita Integratori

La vendita degli integratori è realizzata mediante una stored procedure che opera un’azione molto semplice ma con effetti collaterali sulle altre tabelle:

-in primis controlla che in magazzino ci sia abbastanza merce da vendere e prende da MerceMagazzino l’attributo Rank della merce per decretarne poi l’eventuale sconto sul prezzo originale, preso da inventario magazzino,mediante una function ScontoRank che applica uno sconto del 10% se il rank è ‘low’, 25% se ‘medium’, 50% se ‘high’ (Il rank viene aggiornato mediante Event CalcolaRankMerce).

-opera un decremento sulla capienza del magazzino e sullo stock di merce.In caso il decremento sia uguale alla quantita originale opera una DELETE su mercemagazzino.

-inserisce un record nel Log, per l’analytics

-infine inserisce il record nelle Vendite.

INPUT : CodVendita,Cliente,Centro,Integratore,DataVendita,Quantita,CodProdotto

OUTPUT:

Possibili output di errore

"Al momento quello stock non dispone della quantità richiesta"

Descrizione DecrementaMagazzino

Non tutti gli ordini vanno a buon fine(ciò accade se si supera la data di consegna preferita) , questa procedure chiamata da un’altra procedure EliminaOrdiniFalliti, si occupa di operare un semplice decremento della capienzaattualevirtuale del magazzino pari alla somma delle quantità delle singoli stock di merce. Dopo averla chiamata, EliminaOrdiniFalliti cancella l’ordine e così in cascata tutti gli acquisti.

PORZIONE DELLA TAVOLA DEI VOLUMI INTERESSATA :

Concetto	Tipo	Volume
Ordine	E	3500
Acquisto	E	35000
Magazzino	E	10
InventarioMagazzino	R	350
Integratore	E	4500
MerceMagazzino	E	420000
Vendite	E	500000
Log_VenditeIntegratori	E	7000

TAVOLA DEGLI ACCESSI INVIAORDINE (con ridondanza)

Tipo	Costrutto	Accessi	Nome	Breve Descrizione
L	E	1	Ordine	Prendo stato e il centro.
L	E	1	Magazzino	Prendo la capienza massima e la capienzaattualevirtuale.
L	E	10	Acquisto	Legge le singola quantità di merce.
S	E	1	Magazzino	Incrementa

				CapienzaAttualeVirtuale
S	E	1	Ordine	Stato diventa ‘evaso’
S	E	1	OrdiniEvasi	Inserimento

TAVOLA DEGLI ACCESSI INVIAORDINE (senza ridondanza)

Tipo	Costrutto	Accessi	Nome	Breve Descrizione
L	E	1	Ordine	Prendo stato e il centro.
L	E	1	Magazzino	Prendo la capienza massima e la capienzaattualevirtuale.
L	E	10	Acquisto	Legge le singola quantità di merce dell’ordine
L	E	3500	Acquisto	Somma le singole quantità di merce di tutti gli ordini.
L	E	42000	MerceMagazzino	Somma le singole quantità di tutte le merci in magazzino.
S	E	1	Ordine	Stato diventa ‘evaso’
S	E	1	OrdiniEvasi	Inserimento

TAVOLA DEGLI ACCESSI VENDITAINTEGRATORI (con ridondanza)

Tipo	Costrutto	Accessi	Nome	Breve Descrizione
L	E	1	Magazzino	Setta una variabile del valore CodMagazzino
L	R	1	InventarioMagazzino	Prende nota del costo dell’integratore.
L	E	1	MerceMagazzino	Setta una variabile del valore della quantità di quel prodotto in quello stock in Magazzino e prende nota anche del rank.
S	E	1	Vendite	Inserimento
S	E	1	Magazzino	Decrementa CapienzaAttualeVirtuale
S	E	1	MerceMagazzino	Decrementa Quantita o opera DELETE
S	E	1	Log_VenditeIntegratori	Inserimento

TAVOLA DEGLI ACCESSI VENDITAINTEGRATORI (senza ridondanza)

Tipo	Costrutto	Accessi	Nome	Breve Descrizione
L	E	1	Magazzino	Setta una variabile del valore CodMagazzino
L	R	1	InventarioMagazzino	Prende nota del costo dell’integratore.
L	E	1	MerceMagazzino	Setta una variabile del valore della quantità di quel prodotto in quello stock in Magazzino e prende nota anche del rank.
S	E	1	Vendite	Inserimento
S	E	1	MerceMagazzino	Decrementa Quantita o opera

				DELETE
S	E	1	Log_VenditeIntegratori	Inserimento

TAVOLA DEGLI ACCESSI DecrementaMagazzino (con ridondanza)

Tipo	Costrutto	Accessi	Nome	Breve Descrizione
L	E	10	Acquisto	Per ogni acquisto somma le singole quantità di merce.
S	E	1	Magazzino	Decrementa CapienzaAttualeVirtuale

TAVOLA DEGLI ACCESSI DecrementaMagazzino (senza ridondanza)

Tipo	Costrutto	Accessi	Nome	Breve Descrizione
-	-	-	-	Non avrebbe senso di esistere. È chiamata solo per aggiornarla.

EsprimiGiudizioPost

Descrizione

L'azione può essere svolta con un semplice inserimento e comporta l'attribuire un TOT numero di stelle a un certo commento.

Un BEFORE TRIGGER controlla che il numero di stelle sia tra 0 e 5 mentre un AFTER TRIGGER fa sì che questa azione aggiorni il numero di stelle totali del post e del proprietario di tale profilo.

Le ridondanze vanno necessariamente aggiornate in quanto ogni profilo in base al numero di post pubblicati e alle stelle guadagnate deve essergli attribuita una certa popolarità sul social della palestra.

Ricalcolare ogni volta il numero di record su Giudizio che è una tabella con un gran numero di occorrenze implicherebbe un numero elevatissimo di accessi in lettura ogni volta che viene espresso un giudizio per vedere se le stelle assegnate permettono in qualche modo all'utente di scalare in alto nel grado di popolarità del social .

Più in generale si permette così all'utente di visualizzare il proprio numero di stelle generale o relativo a un dato post a costo di un semplice accesso in lettura evitando dispendiose operazioni di conteggio (SUM) sui record di tale tabelle.

Gradi sbloccati in base al numero di stelle:

- **"Nuovo Arrivato"** per chi supera le 30 stelle

-"Conosciuto" per chi supera le 200 stelle

-"Popolare" per chi supera le 500 stelle

-"Vip" per chi supera le 1000 stelle

Inserimento : bisogna dare al sistema il codice del post da commentare , il numero intero che esprime il voto e il codice del proprio account .

PORZIONE DEL DIAGRAMMA E-R INTERESSATO:

Giudizio,ProfiloSocial,PostRisposta

PORZIONE DELLA TAVOLA DEI VOLUMI INTERESSATA :

Concetto	Tipo	Volume
ProfiloSocial	E	6415000
PostRisposta	E	160000000
Giudizio	R	320000000

TAVOLA DEGLI ACCESSI (con ridondanza)

Tipo	Costrutto	Accessi	Nome	Breve Descrizione
S	R	1	Giudizio	Inserimento del giudizio da parte dell'utente("innesca" il trigger)
L	E	1	ProfiloSocial	INNER JOIN con PostRisposta. -una lettura per prendere il numero di stelle dell'utente che ha messo il post giudicato prima del voto dell'altro utente.
L	E	1	PostRisposta	Attraverso il codice del post permette di risalire all'utente che ha messo il post sul social.
S	E	1	PostRisposta	Aggiorna il numero totale di stelle del post.
S	E	1	ProfiloSocial	Aggiorna il numero totale di stelle dell'utente che ha postato.
S	E	1	*ProfiloSocial	*Se si verificano certe condizioni si fa l'update per aggiornare il grado di popolarità dell'utente.

TAVOLA DEGLI ACCESSI (senza ridondanza)

Tipo	Costrutto	Accessi	Nome	Breve Descrizione
S	R	1	Giudizio	Inserimento del giudizio da parte dell'utente("innesca" il trigger)
L	R	50	Giudizio	Per vedere il numero di stelle
L	E	1	PostRisposta	Attraverso il codice del post permette di risalire all'utente che ha messo il post sul social.
S	E	1	PostRisposta	Aggiorna il numero totale di stelle del post.
S	E	1	ProfiloSocial	Aggiorna il numero totale di stelle dell'utente che ha postato.

S	E	1	*ProfiloSocial	*Se si verificano certe condizioni si fa l'update per aggiornare il grado di popolarità dell'utente.

Codici

Area Gestione

Tabelle

```
CREATE TABLE IF NOT EXISTS Centro (
CodCentro char(10) not null,
Citta char(80) not null,
Indirizzo char(80) not null,
Telefono char(12) not null,
Dimensionemetriquadri int,
Maxclientiospitabili int,
Dirigente char(16) not null,
PRIMARY KEY(CodCentro),
CONSTRAINT Dirigente_Centro
FOREIGN KEY (Dirigente)
REFERENCES Personale(CodFiscale)
ON DELETE NO ACTION
ON UPDATE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi:
CodCentro, Citta, Indirizzo, Telefono, Dimensionemetriquadri, Maxclientiospitabili, Dirigente
Chiave primaria: CodCentro.
Vincoli Integrita Referenziale: Dirigente_Centro
Dipendenze Funzionali: CodCentro→Citta, Indirizzo, Telefono, DimensioneMetriQuadri, MaxClientiOspitabili, Dirigente
Note:
Il dirigente è un membro del personale.

```
CREATE TABLE IF NOT EXISTS Sala(
CodSala char(20) not null,
NomeSala char(30) not null,
TipoSala char(80) not null,
AbbonamentoMinimo int not null,
Responsabile char(16) not null,
Centro char(20) not null,
Interno bool not null,
TariffaAccessoSingolo double not null,
PRIMARY KEY (CodSala),
CONSTRAINT ResponsabilitaSala
FOREIGN KEY (Responsabile)
REFERENCES Istruttore(CodFiscale)
ON UPDATE CASCADE,
CONSTRAINT CentroAppartenenza
FOREIGN KEY (Centro)
REFERENCES Centro(CodCentro)
```

```
ON DELETE CASCADE
ON UPDATE NO ACTION,
CONSTRAINT Tipologia_Sala
FOREIGN KEY (TipoSala)
REFERENCES TipologiaCorso_Sala(TipoSala)
ON DELETE NO ACTION
ON UPDATE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: CodSala, NomeSala, TipoSala, AbbonamentoMinimo, Responsabile, Centro, Interno, TariffaAccessoSingolo

Chiave primaria: CodSala
Vincoli Integrita Referenziale: ResponsabilitaSala, CentroAppartenenza, Tipologia_Sala

Dipendenze Funzionali:
CodSala→NomeSala, TipoSala, AbbonamentoMinimo, Responsabile, Centro, Interno, TariffaAccessoSingolo

Note:
L'Attributo “Interno” permette di specificare se la sala si trovi internamente o esternamente al centro. L'attributo AbbonamentoMinimo serve principalmente per controllare gli accessi alle sale, poiché alcune sale sono riservate solo ad abbonamenti che hanno “Priorità” più alta. TipoSala ci permette di selezionare i possibili corsi che possono essere tenuti nella sala. I tipi della sala e i possibili corsi abbinabili alla sala possiamo notarli nella tabella successiva, TipologiaCorso_Sala. Con TarriffaAccessoSingolo si va a specificare quanto costa all'ora entrare per un singolo accesso nel centro(Ovviamente questo vale per i senza contratto e per chi ha un contratto con Priorita minore della sala).

```
CREATE TABLE IF NOT EXISTS TipologiaCorso_Sala (
 TipoSala CHAR(80) not null,
 Disciplina CHAR(80) not null,
PRIMARY KEY(TipoSala,Disciplina)
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: TipoSala, Disciplina.
Chiave primaria: TipoSala,Disciplina.
Vincoli Integrita Referenziale: -
Dipendenze Funzionali:
“TipoSala” e “Disciplina” sono chiave primaria.
Note:
Come già spiegato nelle note della tabella Sala, TipologiaCorso_Sala serve solo per stabilire la compatibilità di un corso con la sala in cui si vorrebbe far svolgere.

```
CREATE TABLE IF NOT EXISTS Cliente(
CodFiscale char(16) not null,
Nome char(80) not null,
Cognome char(80) not null,
```

```
DataNascita date not null,
Sesso char(1) not null,
Residenza char(80) not null,
Indirizzo char(80) not null,
Telefono char(12) not null,
CodiceDocumentoRiconoscimento char(10) not null,
PRIMARY KEY (CodFiscale),
CONSTRAINT Tutoraggio
FOREIGN KEY (TutorAttuale)
REFERENCES  Istruttore(CodFiscale)
ON DELETE SET NULL
ON UPDATE NO ACTION,
CONSTRAINT RiconoscimentoC
FOREIGN KEY (CodiceDocumentoRiconoscimento)
REFERENCES Documento(CodDocumento)
ON DELETE NO ACTION
ON UPDATE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi:CodFiscale, Nome, Cognome, DataNascita, Sesso, Residenza, Indirizzo, Telefono, CodiceDocumentoRiconoscimento.

Chiave primaria: CodFiscale
Vincoli Integrita Referenziale: RiconoscimentoC.
Dipendenze Funzionali:
CodFiscale→Nome, Cognome, DataNascita, Sesso, Residenza, Indirizzo, Telefono, CodiceDocumentoRiconoscimento.
Ridondanze: Abbonato.
Note:
CodiceDocumentoRiconoscimento è una chiave esterna per la tabella Documento che sarà elencata di seguito a questa. TutorAttuale contiene l'istruttore che tutt'ora sta seguendo il cliente. Il tutor viene preso a caso tra i tutor con meno clienti da seguire dalla stored procedure Aggiungi_Contratto().

```
CREATE TABLE IF NOT EXISTS Documento(
 CodDocumento char(10) not null,
 Prefettura char(80) not null,
 PRIMARY KEY (CodDocumento)
 ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi:CodDocumento, Prefettura.
Chiave primaria: CodDocumento
Vincoli Integrita Referenziale: -
Dipendenze Funzionali:
CodDocumento→Prefettura.
Note: -

```
CREATE TABLE IF NOT EXISTS CaratteristicheFisiche(
 Cliente char(16) not null,
 Altezza double not null,
```

```
 Peso double not null,
 PercentualeMassaGrassa double not null,
 PercentualeMassaMagra double not null,
 AcquaTotale double not null,
 StatoAttuale char(30) not null,
PRIMARY KEY(Cliente),
CONSTRAINT Caratteristiche
FOREIGN KEY (Cliente)
REFERENCES Cliente(CodFiscale)
ON DELETE CASCADE
ON UPDATE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi:Cliente, Altezza, Peso, PercentualeMassaGrassa, PercentualeMassaMagra, AcquaTotale, StatoAttuale
Chiave primaria: Cliente
Vincoli Integrita Referenziale: Caratteristiche
Dipendenze Funzionali:
Cliente→Altezza, Peso, PercentualeMassaGrassa, PercentualeMassaMagra, AcquaTotale, StatoAttuale
Note:
Per l'attributo StatoAttuale valgono solo 3 valori (Sottopeso, Sovrappeso o Normopeso) di cui ne verrà scelto uno dalla stored procedure InserimentoCaratteristiche(), che valuterà principalmente la percentuale di massa grassa del cliente.

```
CREATE TABLE IF NOT EXISTS Personale (
 CodFiscale char(16) not null,
 Nome char(80) not null,
 Cognome char(80) not null,
 DataNascita date not null,
 Sesso char(1) not null,
 Residenza char(80) not null,
 Indirizzo char(80) not null,
 DocumentoRiconoscimento char(10),
 Telefono char(12) not null,
 PRIMARY KEY(CodFiscale),
 CONSTRAINT RiconoscimentoP
 FOREIGN KEY (DocumentoRiconoscimento)
 REFERENCES Documento(CodDocumento)
 ON DELETE SET NULL
 ON UPDATE CASCADE
 ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: CodFiscale, Nome, Cognome, DataNascita, Sesso, Residenza, Indirizzo, DocumentoRiconoscimento, Telefono
Chiave primaria: CodFiscale
Vincoli Integrita Referenziale: RiconoscimentoP
Dipendenze Funzionali:

CodFiscale→Nome, Cognome, DataNascita, Sesso, Residenza, Indirizzo, DocumentoRiconoscimento, Telefono

Note:
La tabella “Personale” ha gli stessi attibuti della tabella “Cliente” .

```
CREATE TABLE IF NOT EXISTS Istruttore (
  CodFiscale char(16) not null,
  Nome char(80) not null,
  Cognome char(80) not null,
  DataNascita date not null,
  Sesso char(1) not null,
  Residenza char(80) not null,
  Indirizzo char(80) not null,
  DocumentoRiconoscimento char(10),
  Telefono char(12) not null,
 PRIMARY KEY(CodFiscale),
 CONSTRAINT RiconoscimentoIstrut
 FOREIGN KEY (DocumentoRiconoscimento)
 REFERENCES Documento(CodDocumento)
 ON DELETE SET NULL
 ON UPDATE CASCADE,
 CONSTRAINT Personale_Istruttore
 FOREIGN KEY (CodFiscale)
 REFERENCES Personale(CodFiscale)
 ON DELETE CASCADE
 ON UPDATE CASCADE
 ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: CodFiscale, Nome, Cognome, DataNascita, Sesso, Residenza, Indirizzo, DocumentoRiconoscimento, Telefono
Chiave primaria: CodFiscale
Vincoli Integrita Referenziale: RiconoscimentoIstrut, Personale_Istruttore

Dipendenze Funzionali:
CodFiscale→Nome, Cognome, DataNascita, Sesso, Residenza, Indirizzo, DocumentoRiconoscimento, Telefono

Note: -

```
CREATE TABLE IF NOT EXISTS Medico (
 CodFiscale char(16) not null,
 Nome char(80) not null,
 Cognome char(80) not null,
```

```
 DataNascita date not null,
 Sesso char(1) not null,
 Residenza char(80) not null,
 Indirizzo char(80) not null,
 DocumentoRiconoscimento char(10),
 Telefono char(12) not null,
 PRIMARY KEY(CodFiscale),
 CONSTRAINT RiconoscimentoMed
 FOREIGN KEY (DocumentoRiconoscimento)
 REFERENCES Documento(CodDocumento)
 ON DELETE SET NULL
 ON UPDATE CASCADE,
 CONSTRAINT Personale_Medico
 FOREIGN KEY (CodFiscale)
 REFERENCES Personale(CodFiscale)
 ON DELETE CASCADE
 ON UPDATE CASCADE
 ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: CodFiscale, Nome, Cognome, DataNascita, Sesso, Residenza, Indirizzo, DocumentoRiconoscimento, Telefono
Chiave primaria: CodFiscale
Vincoli Integrita Referenziale: RiconoscimentoMed, Personale_Medico
Dipendenze Funzionali:
CodFiscale→Nome, Cognome, DataNascita, Sesso, Residenza, Indirizzo, DocumentoRiconoscimento, Telefono
Note: -

```
CREATE TABLE IF NOT EXISTS Segreteria (
 CodFiscale char(16) not null,
 Nome char(80) not null,
 Cognome char(80) not null,
 DataNascita date not null,
 Sesso char(1) not null,
 Residenza char(80) not null,
 Indirizzo char(80) not null,
 DocumentoRiconoscimento char(10),
 Telefono char(12) not null,
 PRIMARY KEY(CodFiscale),
 CONSTRAINT RiconoscimentoSegr
 FOREIGN KEY (DocumentoRiconoscimento)
 REFERENCES Documento(CodDocumento)
 ON DELETE SET NULL
 ON UPDATE CASCADE,
 CONSTRAINT Personale_Segreteria
 FOREIGN KEY (CodFiscale)
 REFERENCES Personale(CodFiscale)
 ON DELETE CASCADE
 ON UPDATE CASCADE
 ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: CodFiscale, Nome, Cognome, DataNascita, Sesso, Residenza, Indirizzo, DocumentoRiconoscimento, Telefono
Chiave primaria: CodFiscale
Vincoli Integrita Referenziale: RiconoscimentoSegr, Personale_Segreteria

Dipendenze Funzionali:
CodFiscale→Nome, Cognome, DataNascita, Sesso, Residenza, Indirizzo, DocumentoRiconoscimento, Telefono

Note: -

```
CREATE TABLE IF NOT EXISTS Responsabilita (
 Dipendente char(16) not null,
 Responsabile char(30) not null DEFAULT '',
 PRIMARY KEY(Dipendente),
 CONSTRAINT Dipendente_Responsabilita
 FOREIGN KEY(Dipendente)
 REFERENCES Personale(CodFiscale)
 ON DELETE CASCADE
 ON UPDATE CASCADE,
 CONSTRAINT Responsabile_Responsabilita
 FOREIGN KEY (Responsabile)
 REFERENCES Personale(CodFiscale)
 ON DELETE CASCADE
 ON UPDATE CASCADE
 )ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: Dipendente, Responsabile
Chiave primaria: Responsabile
Vincoli Integrita Referenziale: Dipendente_Responsabilita, Responsabile_Responsabilita
Dipendenze Funzionali:
Dipendente→Responsabile
Note:
La tabella ha l'onere di gestire tutti i vincoli di responsabilita dei componenti del personale.

```
CREATE TABLE IF NOT EXISTS Contratto(
CodContratto char(10) not null,
Consulente char(16) not null,
DuratainMesi int not null,
ModPagamento char(12) not null,
Tipologia char(15) not null,
SedeSottoscrizione char(10) not null,
Scopo char(40) not null,
DataSottoscrizione date not null,
Cliente char(16) not null,
ImportoTotale int DEFAULT 0 not null,
PRIMARY KEY(CodContratto),
```

```
CONSTRAINT Consulenza
FOREIGN KEY (Consulente)
REFERENCES Segreteria(CodFiscale)
ON DELETE NO ACTION
ON UPDATE CASCADE,
CONSTRAINT Sottoscrizione
FOREIGN KEY (SedeSottoscrizione)
REFERENCES Centro(CodCentro)
ON DELETE NO ACTION
ON UPDATE CASCADE,
CONSTRAINT Clientela
FOREIGN KEY (Cliente)
REFERENCES Cliente(CodFiscale)
ON DELETE NO ACTION
ON UPDATE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: CodContratto, Consulente, DuratainMesi, ModPagamento, Tipologia, SedeSottoscrizione, Cliente,
Scopo, DataSottoscrizione, ImportoTotale.
Chiave primaria: CodContratto
Vincoli Integrita Referenziale: Sottoscrizione, Clientela, Consulenza.
Dipendenze Funzionali:
CodContratto→Consulente, Cliente, DuratainMesi, ModPagamento, Tipologia, SedeSottoscrizione, DataSottoscrizione, Scopo, ImportoTotale.
Note:
Il campo ModPagamento ammette solo due valori (Dilazione e Immediato). Quando il contratto è dilazionato c'è tutta la gestione delle rate mediante tabelle Rateizzazione e Rata. Sul consulente, che appartiene alla segreteria, sono effettuati controlli sul suo turno di lavoro, per verificare se effettivamente il consulente ha potuto stipulare il contratto. Lo "Scopo" può variare tra potenziamento muscolare, dimagrimento e attività ricreativa.

```
CREATE TABLE IF NOT EXISTS PotenziamentoMuscolare (
 Contratto char(10) not null,
 Muscolo char(50) not null,
 LivelloPotenziamento char(15) not null,
 PRIMARY KEY(Contratto,Muscolo),
 CONSTRAINT PotenziamentoCorrente
 FOREIGN KEY (Contratto)
 REFERENCES Contratto(CodContratto)
 ON DELETE CASCADE
 ON UPDATE CASCADE
 )ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: Contratto, Muscolo, LivelloPotenziamento.
Chiave primaria: Contratto, Muscolo
Vincoli Integrita Referenziale: PotenziamentoCorrente.
Dipendenze Funzionali:
Contratto, Muscolo→LivelloPotenziamento.

Note:

Su questa tabella non c'è molto da dire, semplicemente se il cliente ha scelto come scopo del contratto il potenziamento muscolare, per ogni muscolo che il cliente vuole potenziare viene aggiunta una tupla con il contratto, il muscolo che il cliente vorrebbe potenziare e il livello di potenziamento del muscolo.

```
CREATE TABLE IF NOT EXISTS Rateizzazione (
 CodRateizzazione char(10) not null,
 Contratto char(10) not null,
 TassoInteressePercentuale double not null,
 NumeroRate int not null,
 IstitutoFinanziario char(80) not null,
 PRIMARY KEY(CodRateizzazione),
 CONSTRAINT Pagamento
 FOREIGN KEY (Contratto)
 REFERENCES Contratto(CodContratto)
 ON UPDATE CASCADE
 ON DELETE CASCADE
 )ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: CodRateizzazione, Contratto, TassoInteressePercentuale, NumeroRate, IstitutoFinanziario.
Chiave primaria: CodRateizzazione.
Vincoli Integrita Referenziale: Pagamento.
Dipendenze Funzionali:
CodRateizzazione→ Contratto, TassoInteressePercentuale, NumeroRate, IstitutoFinanziario.
Note:
La tabella memorizza l'istituto finanziario che gestirà la rateizzazione, l'interesse che applicherà e il numero rate totali che il cliente ha scelto.

```
CREATE TABLE IF NOT EXISTS Rata (
CodRata char(15) not null,
Rateizzazione char(10) not null,
Importo double not null,
StatoPagamento char(25) not null,
DataScadenza date not null,
PRIMARY KEY(CodRata),
CONSTRAINT Dilazione
FOREIGN KEY (Rateizzazione)
REFERENCES Rateizzazione(CodRateizzazione)
ON UPDATE CASCADE
ON DELETE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: CodRata, Rateizzazione, Importo, StatoPagamento, DataScadenza.
Chiave primaria: CodRata.
Vincoli Integrita Referenziale: Dilazione.
Dipendenze Funzionali:
CodRata→ Rateizzazione, Importo, StatoPagamento, DataScadenza.
Note:

La tabella serve a memorizzare tutte le singole rate nate dalla rateizzazione del contratto. Sono gestite mediante un trigger che mentre inserisce la rata, controlla che esista la rateizzazione collegata e calcola l'importo maggiorato dell'interesse.

```
CREATE TABLE IF NOT EXISTS AbbonamentoStandard (
NomeAbbonamento char(15) not null,
Priorita int not null,
Prezzo int not null,
MaxNumeroIngressiSettimanali int not null,
AccessoPiscine bool,
NumeroMaxIngressoPiscineMese int not null,
PossibilitaFrequentazioneCorsi bool,
PRIMARY KEY(NomeAbbonamento)
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: NomeAbbonamento, Priorita, Prezzo, MaxNumeroIngressiSettimanali, AccessoPiscine, NumeroMaxAccessoPiscineMese, PossibilitaFrequentazioneCorsi

Chiave primaria: NomeAbbonamento
Vincoli Integrita Referenziale: -

Dipendenze Funzionali:
NomeAbbonamento→Priorita, Prezzo, MaxNumeroIngressiSettimanali, AccessoPiscine, NumeroMaxAccessoPiscineMese, PossibilitaFrequentazioneCorsi

Note:
La tabella ha il semplice scopo di memorizzare tutti i tipi di contratti standard (Silver, Gold e Platinum) e tutti i servizi che offrono. Priorita permete di capire a quali sale può entrare il cliente e va da un minimo di 1 del silver a un massimo di 3 del platinum. MaxNumeroIngressiSettimanali specifica quante volte il cliente può entrare nel centro. Ovviamente chi vuole un numero illimitato pagherà per il contratto personalizzato.

```
CREATE TABLE IF NOT EXISTS AutorizzazionePersonalizzata(
Contratto char(10) not null,
Centro char(10) not null,
Priorita INT not null,
MaxNumeroIngressiSettimanali int not null,
AccessoPiscine bool,
NumeroMaxIngressoPiscineMese int not null,
PossibilitaFrequentazioneCorsi bool,
PRIMARY KEY (Contratto,Centro),
CONSTRAINT AutorizzaContrattoPersonalizzato
FOREIGN KEY (Contratto)
REFERENCES Contratto(CodContratto)
ON DELETE CASCADE
ON UPDATE CASCADE,
CONSTRAINT CentroAutorizzazionePersonalizzata
FOREIGN KEY (Centro)
REFERENCES Centro(CodCentro)
```

```
ON DELETE CASCADE
ON UPDATE CASCADE
)Engine=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: Contratto, Centro, Priorita, Prezzo, MaxNumeroIngressiSettimanali, AccessoPiscine, NumeroMaxAccessoPiscineMese, PossibilitaFrequentazioneCorsi
Chiave primaria: Contratto, Centro
Vincoli Integrita Referenziale: AutorizzaContrattoPersonalizzato, CentroAutorizzazionePersonalizzata

Dipendenze Funzionali:
Contratto, Centro→Priorita, MaxNumeroIngressiSettimanali, AccessoPiscine, NumeroMaxAccessoPiscineMese, PossibilitaFrequentazioneCorsi

Note: A differenza del semplice contratto standard, quello personalizzato permette di scegliere i servizi inclusi. La tabella permette quindi di gestire contratti personalizzati multi-sede grazie alla chiave primaria composta dal contratto e dal centro. Il prezzo verrà calcolato da una procedure che permetterà anche l'inserimento.

```
CREATE TABLE IF NOT EXISTS AutorizzazioneCentro(
Contratto char(10) not null,
Centro char(10) not null,
PRIMARY KEY (Contratto,Centro),
CONSTRAINT AutorizzaContratto
FOREIGN KEY (Contratto)
REFERENCES Contratto(CodContratto)
ON DELETE CASCADE
ON UPDATE CASCADE,
CONSTRAINT CentroAutorizzazione
FOREIGN KEY (Centro)
REFERENCES Centro(CodCentro)
ON DELETE CASCADE
ON UPDATE CASCADE
)Engine=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: Contratto, Centro.
Chiave primaria: Contratto, Centro.
Vincoli Integrita Referenziale: AutorizzaContratto, CentroAutorizzazione.

Dipendenze Funzionali:
Contratto e Centro sono chiave.

Note: La logica dietro a questa tabella è simile a quella dei contratti personalizzati, infatti permettono ai contratti standard di essere dei multi-sede ma non contengono i campi con i vari servizi, visto che i contratti standard li hanno di default.

Stored Procedures

Le stored procedure analizzate in questo principalmente inseriscono tuple nelle tabelle descritte sopra e effettuano controlli sulla validità delle informazioni date in ingresso come parametri. La stored principale risulta essere *Inserisci_Contratto()* visto che i controlli che si vedranno successivamente riguarderanno i clienti con i loro contratti, per verificare se il loro contratto è valido o ha determinate caratteristiche. Di tutto ciò si parlerà principalmente nella gestione degli accessi dei clienti, gestita da 2 trigger principali: *ControllaAccessoCentro* e *ControllaAccessoSala.*

Aggiungi_Contratto()

```
DROP PROCEDURE IF EXISTS Aggiungi_Contratto;
DELIMITER $$
CREATE PROCEDURE Aggiungi_Contratto(IN _CodContratto VARCHAR(10), IN _Consulente VARCHAR(50), IN
_DuratainMesi INT,
                        IN _ModPagamento VARCHAR(12),  IN _Tipologia VARCHAR(15), IN _SedeSottoscrizione
VARCHAR(10),
                        IN _Scopo VARCHAR(40), IN _DataSottoscrizione DATE, IN _Cliente VARCHAR(50))

BEGIN


  IF (_Tipologia NOT IN (SELECT NomeAbbonamento FROM AbbonamentoStandard) AND
_Tipologia<>'Personalizzato') THEN


      BEGIN
       SET @Errore=CONCAT('Nome abbonamento ',_Tipologia,' non esistente!');

       SIGNAL SQLSTATE '45000'
       SET MESSAGE_TEXT=@Errore;

      END;
   ELSE
    BEGIN
   IF (_Scopo<>'Dimagrimento'  AND _Scopo<>'Potenziamento Muscolare' AND _Scopo<>'Attività ricreativa' ) THEN
     SIGNAL SQLSTATE '45000'
     SET MESSAGE_TEXT='Scopo non valido';
   END IF;


  IF (_DataSottoscrizione='' OR _DataSottoscrizione IS NULL) THEN

    SET _DataSottoscrizione=CURRENT_DATE();

  END IF;
     INSERT INTO Contratto
     VALUES
(_CodContratto,_Consulente,_DuratainMesi,_ModPagamento,_Tipologia,_SedeSottoscrizione,_Scopo,_DataSottoscrizi
one,_Cliente,
            0);
 END ;
```

```
 END IF;
END $$
DELIMITER ;
```

Descrizione: La procedure riceve in ingresso tutti i parametri essenziali che andranno a essere inseriti nel contratto. Per prima cosa la stored procedure controlla se la tipologia del contratto è valida, ovvero se è una tra quelle esistenti nel database o è un contratto personalizzato e se non valida si restituisce un errore e si annulla l'inserimento. Altro controllo svolto è sullo scopo, quindi si verifica se è uno dei tre valori ammissibili. Si controlla se la data non sia vuota o NULL, in questo caso si assegna un Current_Date() al parametro inviato in precedenza.

InserisciAutorizzazioneStandard()

```
DROP PROCEDURE IF EXISTS InserisciAutorizzazioneStandard;
 DELIMITER $$
 CREATE PROCEDURE InserisciAutorizzazioneStandard(IN _Contratto VARCHAR(20),
                                                  IN _Centro VARCHAR(25))
  BEGIN
   DECLARE _TipologiaContratto VARCHAR(20) DEFAULT '';
   DECLARE PrezzoAbbonamento INT DEFAULT 0;
   DECLARE PrezzoAbbonamentoMese INT DEFAULT 0;


   SELECT Tipologia,ImportoTotale/DurataInMesi INTO _TipologiaContratto, PrezzoAbbonamentoMese
   FROM Contratto
   WHERE CodContratto=_Contratto;

    IF _TipologiaContratto='Personalizzato'THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT='Contratto standard non esistente';
     END IF;

   SELECT Prezzo INTO PrezzoAbbonamento
         FROM AbbonamentoStandard
   WHERE NomeAbbonamento=_TipologiaContratto;


   IF (PrezzoAbbonamentoMese/PrezzoAbbonamento)>=3 THEN
     SIGNAL SQLSTATE '45000'
     SET MESSAGE_TEXT='Numero massimo autorizzazioni raggiunto';
         END IF;


    INSERT INTO AutorizzazioneCentro
     VALUES (_Contratto,_Centro);

    UPDATE Contratto
          SET ImportoTotale=((ImportoTotale/DuratainMesi) + (PrezzoAbbonamento)) * DuratainMesi
          WHERE CodContratto=_Contratto;

   END $$
DELIMITER ;
```

Descrizione: La procedure, come dice anche il nome, permette l'inserimento di un autorizzazione standard al centro dopo aver controllato che il numero di autorizzazioni a centri non sia maggiore o uguale di 3 e controlla se il contratto esiste ed è attivo. Come prima cosa si ricava la tipologia del contratto e il prezzo di quel tipo di abbonamento poi aggiorna l'importo del contratto, aggiungendo il prezzo per la nuova autorizzazione.

AggiungiAutorizzazionePersonalizzata()

```
DROP PROCEDURE IF EXISTS AggiungiAutorizzazionePersonalizzata;
DELIMITER $$
CREATE PROCEDURE AggiungiAutorizzazionePersonalizzata (IN _Contratto VARCHAR(20),
                                                       IN _Centro VARCHAR(20),
                                                       IN _Priorita INT ,
                                                       IN _MaxNumeroIngressiSettimanali INT,
                                                       IN _AccessoPiscine BOOL,
                                                       IN _NumeroMaxIngressoPiscineMese INT,
                                                       IN _PossibilitaFrequentazioneCorsi BOOL)

BEGIN

   DECLARE Durata INT DEFAULT 0;

   IF (NOT EXISTS (SELECT * FROM Contratto
                           WHERE CodContratto=_Contratto
                                   AND Tipologia='Personalizzato')) THEN
     SIGNAL SQLSTATE '45000'
     SET MESSAGE_TEXT='Contratto non esistente';
    END IF;

   IF (SELECT COUNT(*) FROM AutorizzazionePersonalizzata WHERE Contratto=_Contratto)>=3 THEN
     SIGNAL SQLSTATE '45000'
     SET MESSAGE_TEXT='Numero massimo autorizzazioni raggiunto';
         END IF;

   INSERT INTO AutorizzazionePersonalizzata
   VALUES (_Contratto, _Centro,
_Priorita,_MaxNumeroIngressiSettimanali,_AccessoPiscine,_NumeroMaxIngressoPiscineMese,
                           _PossibilitaFrequentazioneCorsi);


   SELECT DuratainMesi INTO Durata
   FROM Contratto
   WHERE CodContratto=_Contratto;

 UPDATE Contratto
 SET ImportoTotale=((10 *_MaxNumeroIngressiSettimanali)+(2 *_AccessoPiscine)+(1
*_NumeroMaxIngressoPiscineMese)+
        (2.5 *_Priorita)+
                           (5 *_PossibilitaFrequentazioneCorsi) + 8 + (ImportoTotale/Durata))* Durata
 WHERE CodContratto=_Contratto;


END $$
DELIMITER ;
```

Descrizione: La procedure fa lo stesso lavoro della precedente, le uniche differenze stanno nel calcolo dell'importo e nel primo controllo, dove ovviamente deve verificare l'esistenza del contratto.

InserisciPersonale()

```
DROP PROCEDURE IF EXISTS InserisciPersonale;
DELIMITER $$
CREATE PROCEDURE InserisciPersonale(IN _CodFiscale VARCHAR(16), IN _Nome VARCHAR(80),
                                    IN  _Cognome VARCHAR(80), IN _DataNascita DATE,
                                    IN _Sesso VARCHAR(1), IN _Residenza VARCHAR(80),
                                    IN _Indirizzo VARCHAR(80),
                                    IN _DocumentoRiconoscimento VARCHAR(10),
                                    IN _Telefono VARCHAR(12), IN _Ruolo VARCHAR(20))
BEGIN

 IF (_Ruolo='Istruttore' OR _Ruolo='Medico' OR _Ruolo='Segreteria') THEN
 BEGIN
  INSERT INTO Personale
  VALUES
(_CodFiscale,_Nome,_Cognome,_DataNascita,_Sesso,_Residenza,_Indirizzo,_DocumentoRiconoscimento,_Telefono);

  IF (_Ruolo='Istruttore') THEN

    INSERT INTO Istruttore
    VALUES
(_CodFiscale,_Nome,_Cognome,_DataNascita,_Sesso,_Residenza,_Indirizzo,_DocumentoRiconoscimento,_Telefono);

  ELSE IF (_Ruolo='Medico') THEN

      INSERT INTO Medico
      VALUES
(_CodFiscale,_Nome,_Cognome,_DataNascita,_Sesso,_Residenza,_Indirizzo,_DocumentoRiconoscimento,_Telefono);

                ELSE IF (_Ruolo='Segreteria') THEN

        INSERT INTO Segreteria
        VALUES
(_CodFiscale,_Nome,_Cognome,_DataNascita,_Sesso,_Residenza,_Indirizzo,_DocumentoRiconoscimento,_Telefono);

       END IF;
                END IF;
       END IF;

 END;

 ELSE

  SIGNAL SQLSTATE '45000'
  SET MESSAGE_TEXT='Ruolo non esistente';

 END IF;

END $$
DELIMITER ;
```

Descrizione: La procedure permette l'inserimento del personale, controllando nello stesso momento in quale entità figlia di Personale va inserita la nuova tupla. Tutto ciò è possibile grazie al parametro di ingresso _Ruolo, che identifica la tabella che lo conterrà. Viene valutata anche la possibilità che il ruolo inserito non esista.

InserimentoCaratteristiche()

```
DROP PROCEDURE IF EXISTS InserimentoCaratteristiche;
 DELIMITER $$
 CREATE PROCEDURE InserimentoCaratteristiche(IN _Cliente Varchar(16), IN _Altezza double, IN _Peso double,
IN _PercentualeGrasso double, IN _PercentualeMagro double, IN _Acqua double)
 BEGIN
  DECLARE Stato Varchar(15) DEFAULT '';
  DECLARE _Sesso Varchar(1) DEFAULT '';
  SELECT Sesso INTO _Sesso FROM Cliente WHERE CodFiscale=_Cliente;
  IF (_Altezza<=0 OR _Peso<=0 OR _PercentualeGrasso<=0 OR _PercentualeMagro<=0 OR _Acqua<=0) THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT='Valori non validi';
  END IF;


  IF (_Sesso='M') THEN
  BEGIN
    IF _PercentualeGrasso<=10 THEN
      SET Stato='Sottopeso';
    ELSEIF (_PercentualeGrasso>10 AND _PercentualeGrasso<=20) THEN
      SET  Stato='Normopeso';
    ELSE
      SET Stato='Sovrappeso';
           END IF;
  END;
  ELSE
  BEGIN
    IF _PercentualeGrasso<=20 THEN
      SET Stato='Sottopeso';
    ELSEIF (_PercentualeGrasso>20 AND _PercentualeGrasso<=30) THEN
      SET  Stato='Normopeso';
    ELSE
      SET Stato='Sovrappeso';
           END IF;
  END;
  END IF;
  INSERT INTO CaratteristicheFisiche
  VALUES (_Cliente,_Altezza,_Peso,_PercentualeGrasso,_PercentualeMagro,_Acqua,Stato);
  END $$
  DELIMITER ;
```

Descrizione: La procedure permette l'inserimento delle caratteristiche fisiche di un cliente, distinguendo i casi da uomo a donna, usando parametri di valutazione dello stato differenti (lo stato ammette 3 valori: Sottopeso, Normopeso, Sovrappeso). Alla fine della valutazione inserisce la tupla nella tabella CaratteristicheFisiche. Il primo controllo verifica che i parametri inviati contengano valori validi.

InserisciPotenziamentoMuscolare()

```
DROP PROCEDURE IF EXISTS InserisciPotenziamentoMuscolare;
 DELIMITER $$
 CREATE PROCEDURE InserisciPotenziamentoMuscolare (IN _Contratto VARCHAR(25), IN _Muscolo
VARCHAR(50), IN _Livello VARCHAR(70))
 BEGIN

  IF (_Livello<>'Lieve' AND _Livello<>'Moderato' AND _Livello<>'Elevato') THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT='Valore non valido';
  END IF;

  IF NOT EXISTS (SELECT * FROM Contratto WHERE CodContratto=_Contratto) THEN

     SIGNAL SQLSTATE '45000'
     SET MESSAGE_TEXT='Contratto non esistente';
  END IF;

  INSERT INTO PotenziamentoMuscolare
  VALUES (_Contratto, _Muscolo, _Livello);

END $$
DELIMITER ;
```

Descrizione: La procedure permette l'inserimento del potenziamento muscolare del cliente. Come già detto quando si è parlato della tabella PotenziamentoMuscolare, il livello del potenziamento ammette solo 3 valori: Lieve, Moderato, Elevato. Se viene inserito un valore diverso, la stored restituirà un errore. Lo stesso succede se il contratto non esiste (il controllo è da considerarsi inutile, vista la presenza di vincolo di integrità tra Contratto e PotenziamentoMuscolare).

Triggers

```
DROP TRIGGER IF EXISTS CheckRuoloRespIN;
 DELIMITER $$
 CREATE TRIGGER CheckRuoloRespIN
 BEFORE INSERT ON Responsabilita
 FOR EACH ROW
 BEGIN

 IF (NEW.Dipendente IN (SELECT CodFiscale FROM Istruttore)) THEN

     SET @Ruolo='Istruttore';

 ELSE IF (NEW.Dipendente IN (SELECT CodFiscale FROM Medico)) THEN

         SET @Ruolo='Medico';
```

```
    ELSE IF (NEW.Dipendente IN (SELECT CodFiscale FROM Segreteria )) THEN

        SET @Ruolo='Segreteria';

                    ELSE
                            SIGNAL SQLSTATE '45000'
         SET MESSAGE_TEXT='Persona non esistente';
                    END IF;
            END IF;
 END IF;


 IF (NEW.Responsabile IN (SELECT CodFiscale FROM Istruttore)) THEN

    SET @RuoloResp='Istruttore';

 ELSE IF (NEW.Responsabile IN (SELECT CodFiscale FROM Medico)) THEN

        SET @RuoloResp='Medico';

    ELSE IF (NEW.Responsabile IN (SELECT CodFiscale FROM Segreteria )) THEN

         SET @RuoloResp='Segreteria';

                    ELSE
                            SIGNAL SQLSTATE '45000'
         SET MESSAGE_TEXT='Persona non esistente';
                    END IF;
            END IF;
 END IF;

 IF @RuoloResp<>@Ruolo THEN

   SIGNAL SQLSTATE '45000'
   SET MESSAGE_TEXT='Il ruolo del dipendente è diverso da quello del responsabile';
 END IF;
END $$
DELIMITER ;



DROP TRIGGER IF EXISTS CheckRuoloRespUP;
 DELIMITER $$
 CREATE TRIGGER CheckRuoloRespUP
 BEFORE UPDATE ON Responsabilita
 FOR EACH ROW
 BEGIN

 IF (NEW.Dipendente IN (SELECT CodFiscale FROM Istruttore)) THEN

    SET @Ruolo='Istruttore';

 ELSE IF (NEW.Dipendente IN (SELECT CodFiscale FROM Medico)) THEN

        SET @Ruolo='Medico';

    ELSE IF (NEW.Dipendente IN (SELECT CodFiscale FROM Segreteria )) THEN

         SET @Ruolo='Segreteria';
```

```
                    ELSE
                            SIGNAL SQLSTATE '45000'
          SET MESSAGE_TEXT='Persona non esistente';
                    END IF;
               END IF;
  END IF;


  IF (NEW.Responsabile IN (SELECT CodFiscale FROM Istruttore)) THEN

     SET @RuoloResp='Istruttore';

 ELSE IF (NEW.Responsabile IN (SELECT CodFiscale FROM Medico)) THEN

         SET @RuoloResp='Medico';

     ELSE IF (NEW.Responsabile IN (SELECT CodFiscale FROM Segreteria )) THEN

          SET @RuoloResp='Segreteria';

                    ELSE
                            SIGNAL SQLSTATE '45000'
          SET MESSAGE_TEXT='Persona non esistente';
                    END IF;
               END IF;
  END IF;

  IF @RuoloResp<>@Ruolo THEN

    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT='Il ruolo del dipendente è diverso da quello del responsabile';
  END IF;
END $$
DELIMITER ;
```

Derscrizione: I trigger appena mostrati hanno il compito di controlare che il responsabile e il dipendente che stiamo inserendo siano dello stesso ruolo, perchè non può esistere un medico che ha come responsabile un instruttore.

ControlloRateizzazione

```
  DROP TRIGGER IF EXISTS ControlloRateizzazione;
  DELIMITER $$
  CREATE TRIGGER ControlloRateizzazione
  BEFORE INSERT ON Rateizzazione
  FOR EACH ROW
  BEGIN

  IF new.Contratto IN (SELECT CodContratto FROM Contratto WHERE CodContratto=new.Contratto
AND ModPagamento='Dilazionato') THEN
    BEGIN
     IF new.Contratto IN (SELECT Contratto FROM Rateizzazione) THEN
       SIGNAL SQLSTATE '45000'
```

```
            SET MESSAGE_TEXT ='Rateizzazione del contratto gia presente nel database!';
           END IF;
          END;
 ELSE
  SIGNAL SQLSTATE '45000'
  SET MESSAGE_TEXT ='Contratto con pagamento non dilazionato!';
 END IF;

 END $$
 DELIMITER ;
```

Descrizione: Il trigger effettua il controllo nell'inserimento della rateizzazione. Per prima cosa controlla che quel contratto esista e abbia il pagamento dilazionato e inolte controlla se la rateizzazione per quel contratto già esiste.

ControlloRate

```
DROP TRIGGER IF EXISTS ControllaRate;
 DELIMITER $$
 CREATE TRIGGER ControllaRate
 BEFORE INSERT ON Rata
 FOR EACH ROW
 BEGIN
  DECLARE ContoRate INT DEFAULT 0;
  DECLARE RateTotali INT DEFAULT 0;
  DECLARE DataScadenzaUltimaRata DATE;
  DECLARE Interesse INT DEFAULT 0;
  DECLARE ImportoContratto INT DEFAULT 0;
  DECLARE Contratto_Cliente VARCHAR(50) DEFAULT '';

  SELECT COUNT(*) INTO ContoRate
  FROM Rata
  WHERE Rateizzazione=new.Rateizzazione;

  SELECT NumeroRate,TassoInteressePercentuale,Contratto INTO RateTotali,Interesse,Contratto_Cliente
  FROM Rateizzazione
  WHERE CodRateizzazione=new.Rateizzazione;

  IF ContoRate>=RateTotali THEN
   SIGNAL SQLSTATE '45000'
   SET MESSAGE_TEXT ='Numero di rate massimo raggiunto per quel contratto';
  END IF;

 IF (RateTotali<>0) THEN
   BEGIN
   SELECT MAX(R.DataScadenza) INTO DataScadenzaUltimaRata
   FROM Rata R
   WHERE R.Rateizzazione=new.Rateizzazione;



   IF (period_diff(date_format(new.DataScadenza,'%Y%m'),
      date_format(DataScadenzaUltimaRata%m')) <> 12/RateTotali) THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT ='Periodo tra le due date non valido! ';
    END IF;
```

```
    END;
  END IF;

    IF (new.StatoPagamento<>'Eseguito' AND new.StatoPagamento<>'Non ancora dovuto' AND
new.StatoPagamento<>'Scaduto') THEN
     SIGNAL SQLSTATE '45000'
     SET MESSAGE_TEXT ='Valore non valido nel campo StatoPagamento! ';
     END IF;


   SELECT ImportoTotale INTO ImportoContratto
   FROM Contratto
   WHERE CodContratto=Contratto_Cliente;

   SET NEW.Importo=(ImportoContratto/RateTotali)*((100 + Interesse)/100);

   END $$
   DELIMITER ;
```

Descrizione: Il trigger per prima cosa controlla il numero di rate inserite fino ad ora e verifica che non si sia arrivati al numero massimo stabilito dalla rateizzazione. Se la rata che si sta inserendo non è la prima, si prende la data dell'ultima rata inserita. Successivamente si controlla che la data inserita sia valida controllando che sia passato dall'ultima rata, il periodo di tempo da una scadenza rata ad un'altra. L'ultimo controllo verifica semplicemtente che il valore che si sta inserendo nel campo StatoPagamento sia uno dei tre ammissibili, ovvero Eseguito, Non ancora dovuto o Scaduto.

Events

ControlloSottoscrizioneeRate

```
 DROP EVENT IF EXISTS ControlloSottoscrizioneeRate;
DELIMITER $$
CREATE EVENT ControlloSottoscrizioneeRate
ON SCHEDULE EVERY 1 DAY
DO
 BEGIN
   UPDATE Cliente C1 NATURAL JOIN (SELECT Co.CodContratto,C2.CodFiscale AS CodFiscale
                          FROM Contratto Co INNER JOIN Cliente C2
                          ON Co.Cliente=C2.CodFiscale
                          WHERE Date_add(Co.DataSottoscrizione, INTERVAL Co.DuratainMesi
                                        MONTH)=current_date()) As D
   SET C1.Abbonato=false,
```

```
    C1.TutorAttuale=NULL;


  UPDATE Rata
  SET StatoPagamento='Scaduto'
  WHERE DataScadenza<=Current_date()
    AND StatoPagamento='Non ancora dovuto';

END $$
DELIMITER ;
```

Descrizione: L'event viene eseguito una volta giorno cosicchè possa controllare la scadenza dei contratti e la scadenza delle singole rate che non sono state pagate dai clienti.

Area Servizi

Tabelle

```
CREATE TABLE IF NOT EXISTS SchedaAlimentazione (
CodSchedaAlim Char(15) not null,
Cliente char(16) not null,
Obiettivo char(100) not null,
DataEmissione date not null,
DataInizio date not null,
DataFine date not null,
Medico char(16) not null,
Dieta char(25) not null,
IntervalloVisite_Settimane int not null,
PRIMARY KEY(CodSchedaAlim),
CONSTRAINT DietaDaSeguire
FOREIGN KEY (Dieta)
REFERENCES Dieta(CodDieta)
ON DELETE NO ACTION
ON UPDATE CASCADE,
CONSTRAINT MedicoDiRiferimento
FOREIGN KEY (Medico)
REFERENCES Medico(CodFiscale)
ON DELETE NO ACTION
ON UPDATE CASCADE,
CONSTRAINT ClienteTarget
FOREIGN KEY (Cliente)
REFERENCES Cliente(CodFiscale)
ON UPDATE CASCADE
ON DELETE NO ACTION
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: CodSchedaAlim, Cliente, Obiettivo, DataEmissione, DataInizio, DataFine, Medico, Dieta, IntervalloVisite_Settimane**.**

Chiave primaria: CodSchedaAlim**.**
Vincoli Integrita Referenziale: MedicoDiRiferimento, ClienteTarget**.**
Dipendenze Funzionali:
CodSchedaAlim→Cliente, Obiettivo, DataEmissione, DataInizio, DataFine, Medico, Dieta, IntervalloVisite_Settimane
Note:
IntervalloVisite_Settimane indica ogni quanto il cliente dovrà effettuare la visita successiva.

```
CREATE TABLE IF NOT EXISTS Dieta(
CodDieta char(10) not null,
ApportoCaloricoGiornalieto int not null,
NumeroPasti int not null,
ComposizionePasto text not null,
PRIMARY KEY(CodDieta)
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: CodDieta, ApportoCaloricoGiornalieto, NumeroPasti, ComposizionePasto.
Chiave primaria: CodDieta.
Vincoli Integrita Referenziale: -
Dipendenze Funzionali:
CodDieta→ApportoCaloricoGiornaliero, NumeroPasti, ComposizionePasto.
Note:
IntervalloVisite_Settimane indica ogni quanto il cliente dovrà effettuare la visita successiva.

```
CREATE TABLE IF NOT EXISTS Visita(
CodVisita char(20) not null,
Medico char(16) not null,
Cliente char(16) not null,
DataVisita date not null,
OraVisita time not null,
ValutazioneFisica text not null,
PRIMARY KEY(CodVisita),
CONSTRAINT MedicoVisitante
FOREIGN KEY (Medico)
REFERENCES Medico(CodFiscale)
ON DELETE CASCADE
ON UPDATE CASCADE,
CONSTRAINT ClienteVisitato
FOREIGN KEY (Cliente)
REFERENCES Cliente(CodFiscale)
ON DELETE CASCADE
ON UPDATE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: CodVisita, Medico, Cliente, DataVisita, OraVisita, ValutazioneFisica.
Chiave primaria:CodVisita
Vincoli Integrita Referenziale: MedicoVisitante, ClienteVisitato.
Dipendenze Funzionali:
CodVisita→Medico, Cliente, DataVisita, OraVisita, ValutazioneFisica.

Note:
La ValutazioneFisica contiene una descrizione dello stato fisico, al momento della visita, del cliente.

```
CREATE TABLE IF NOT EXISTS Misurazione(
Peso int not null,
IndiceMassaMagra int not null,
IndiceMassaGrassa int not null,
IndiceAcqua int not null,
Visita char(20) not null,
PRIMARY KEY (Visita),
CONSTRAINT Visita_Misurazione
FOREIGN KEY (Visita)
REFERENCES Visita(CodVisita)
ON DELETE NO ACTION
ON UPDATE NO ACTION
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: Peso, IndiceMasseMagra, IndiceMassaGrassa, IndiceAcqua, Visita
Chiave primaria:Visita
Vincoli Integrita Referenziale: Visita_Misurazione
Dipendenze Funzionali:
Visita→Peso, IndiceMasseMagra,IndiceMassaGrassa, IndiceAcqua

Note:
La Visita è sia chiave interna che esterna, poiché durante una visita possono essere effettuate delle misurazioni per controllare più nel dettaglio lo stato fisico del cliente.

```
CREATE TABLE IF NOT EXISTS AccessoCentro(
Cliente char(16) not null,
Centro char(20) not null,
DataAccesso date not null,
OrarioAccesso time not null,
ArmadiettoAssegnato char(20),
PasswordArmadietto char(8),
OrarioUscita time not null,
PRIMARY KEY(Cliente,Centro,DataAccesso,OrarioAccesso),
CONSTRAINT ClienteAccesso
FOREIGN KEY (Cliente)
REFERENCES Cliente(CodFiscale)
ON DELETE NO ACTION
ON UPDATE CASCADE,
CONSTRAINT CentroAccesso
```

```
FOREIGN KEY (Centro)
REFERENCES Centro(CodCentro)
ON DELETE NO ACTION
ON UPDATE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: Cliente, Centro, DataAccesso, OrarioAccesso, ArmadiettoAssegnato, PasswordArmadietto, OrarioUscita
Chiave primaria:Cliente, DataAccesso, OrarioAccesso
Vincoli Integrita Referenziale: ClienteAccesso, CentroAccesso
Dipendenze Funzionali:
Cliente,DataAccesso,OrarioAccesso→ArmadiettoAssegnato, PasswordArmadietto, OrarioUscita, Centro
Note:
La chiave esterna è composta anche da OrarioAccesso, cosicchè si possa tener conto anche dei clienti che effettuano più di un accesso al giorno. L'armadietto viene scelto a caso tra quelli disponibili in quel momento. La password numerica viene generata randomicamente

```
CREATE TABLE IF NOT EXISTS Log_Accesso (
Cliente char(16) not null,
Centro char(20) not null,
DataAccesso date not null,
OrarioAccesso time not null,
ArmadiettoAssegnato char(20),
PasswordArmadietto char(8),
PRIMARY KEY(Cliente)
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: Cliente, Centro, DataAccesso, OrarioAccesso, ArmadiettoAssegnato, PasswordArmadietto.

Chiave primaria:Cliente.
Vincoli Integrita Referenziale: -
Dipendenze Funzionali:
Cliente.→Centro, DataAccesso, OrarioAccesso, ArmadiettoAssegnato, PasswordArmadietto.
Note:
La tabella Log_Accesso è utile per salvarsi momentaneamente l'accesso fino a che il cliente non decide di uscire, cosi da effettuare controlli migliori e più veloci per gli accessi al centro, da cui poi dipendono anche gli accessi alle singole sale.

```
CREATE TABLE IF NOT EXISTS AccessoSala(
Cliente char(16) not null,
Sala char(20) not null,
DataAccesso date not null,
OrarioAccesso time not null,
OrarioUscita time not null,
PRIMARY KEY(Cliente,DataAccesso,OrarioAccesso),
CONSTRAINT ClienteAccessoSala
FOREIGN KEY (Cliente)
```

```
REFERENCES Cliente(CodFiscale)
ON DELETE NO ACTION
ON UPDATE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: Cliente, Sala, DataAccesso, OrarioAccesso, OrarioUscita.

Chiave primaria:Cliente, DataAccesso, OrarioAccesso.
Vincoli Integrita Referenziale: -
Dipendenze Funzionali:
Cliente,DataAccesso,OrarioAccesso→OrarioUscita**,** Sala
Note:
Come già detto nelle note della tabella AccessoCentro, l'accesso alla sala dipende dal centro in cui è acceduto effettivamente il cliente.

```
CREATE TABLE IF NOT EXISTS Log_Sala(
Cliente char(16) not null,
Sala char(20) not null,
DataAccesso date not null,
OrarioAccesso time not null,
PRIMARY KEY(Cliente),
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: Cliente, Sala, DataAccesso, OrarioAccesso.
Chiave primaria:Cliente.
Vincoli Integrita Referenziale: -
Dipendenze Funzionali:
Cliente→Sala, DataAccesso, OrarioAccesso.
Note:
Log_Sala è stata creata per lo stesso motivo di Log_Accesso.

```
CREATE TABLE IF NOT EXISTS Spogliatoio(
CodSpogliatoio char(10) not null,
Capienza int not null,
Centro char(10) not null,
NumeroPostiDisponibili int not null,
SessoAccedenti char(1) not null,
PRIMARY KEY(CodSpogliatoio),
CONSTRAINT CentroContenente
FOREIGN KEY (Centro)
REFERENCES Centro(CodCentro)
ON DELETE CASCADE
ON UPDATE CASCADE
```

)ENGINE=InnoDB DEFAULT CHARSET=latin1;
Attributi: CodSpogliatoio, Capienza, Centro, NumeroPostiDisponibili,SessoAccedenti
Chiave primaria: CodSpogliatoio
Vincoli Integrita Referenziale: CentroContenente
Dipendenze Funzionali:
CodSpogliatoio→Capienza, Centro, NumeroPostiDisponibili,SessoAccedenti
Note:
Ho deciso di descrivere qui la tabella Spogliatoio solo per la presenza degli Armadietti che sono oggetto degli accessi al centro.

```
CREATE TABLE IF NOT EXISTS Armadietto(
CodArmadietto char(10) not null,
IdSpogliatoio char(10) not null,
Occupato bool not null,
PRIMARY KEY(CodArmadietto),
CONSTRAINT Locazione
FOREIGN KEY(IdSpogliatoio)
REFERENCES Spogliatoio(CodSpogliatoio)
ON DELETE CASCADE
ON UPDATE NO ACTION
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: CodArmadietto, IdSpogliatoio, Occupato.

Chiave primaria: CodArmadietto
Vincoli Integrita Referenziale: Locazione
Dipendenze Funzionali:
CodArmadietto→IdSpogliatoio, Occupato
Ridondanze: Occupato
Note:
Nella tabella è presente una ridondanza nel campo Occupato che settato a 0 quando non è stato assegnato a nessuno e settato a 1 non appena viene assegnato ad un cliente che ha appena effettuato l'accesso al centro.

```
CREATE TABLE IF NOT EXISTS OrarioAperturaCentro (
Centro char(10) not null,
GiornoSettimana char(15) not null,
OrarioApertura time not null,
OrarioChiusura time not null,
PRIMARY KEY (Centro,GiornoSettimana),
CONSTRAINT AperturaCentro
FOREIGN KEY (Centro)
REFERENCES Centro(CodCentro)
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: Centro, GiornoSettimana, OrarioApertura, OrarioChiusura

Chiave primaria: Centro, GiornoSettimana
Vincoli Integrita Referenziale: AperturaCentro
Dipendenze Funzionali:

Centro, GiornoSettimana→OrarioApertura, OrarioChiusura
Note:-

```
CREATE TABLE IF NOT EXISTS Turnazione(
Dipendente char(16) not null,
Centro char(10) not null,
GiornoSettimana char(15) not null,
InizioTurno time not null,
FineTurno time not null,
PRIMARY KEY (Dipendente,InizioTurno,GiornoSettimana),
CONSTRAINT TurnoDipendente
FOREIGN KEY (Dipendente)
REFERENCES Personale(CodFiscale)
ON DELETE NO ACTION
ON UPDATE NO ACTION,
CONSTRAINT TurnoCentro
FOREIGN KEY (Centro)
REFERENCES Centro(CodCentro)
ON UPDATE NO ACTION
ON DELETE NO ACTION
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: Dipendente, Centro, GiornoSettimana, InizioTurno, FineTurno
Chiave primaria: Centro, GiornoSettimana, Dipendente, InizioTurno
Vincoli Integrita Referenziale: TurnoDipendente, TurnoCentro
Dipendenze Funzionali:
Dipendente, GiornoSettimana, InizioTurno→FineTurno, Centro
Note:
La tabella Turnazione è il fulcro di tutta la gestione dei turni e delle lezioni dei corsi che verrà spiegata successivamente.

```
CREATE TABLE IF NOT EXISTS Corso(
CodCorso char(10) not null,
NomeDisciplina char(80) not null,
LivelloInsegnamento char(20) not null,
InizioCorso date not null,
FineCorso date not null,
Sala char(10) not null,
NumeroMaxPartecipanti int not null,
Istruttore char(16) not null,
PRIMARY KEY (CodCorso),
CONSTRAINT Istruzione
FOREIGN KEY (Istruttore)
REFERENCES Istruttore(CodFiscale)
ON DELETE NO ACTION
ON UPDATE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: CodCorso, NomeDisciplina, LivelloInsegnamento, InizioCorso, FineCorso, Sala, NumeroMaxPartecipanti, Istruttore.
Chiave primaria: CodCorso

Vincoli Integrita Referenziale: Istruzione, Sala_Lezione
Dipendenze Funzionali:
CodCorso→NomeDisciplina, LivelloInsegnamento, InizioCorso, FineCorso, Sala, NumeroMaxPartecipanti, Istruttore.
Note:
Sull'attributo Sala, lavorerà un trigger che verificherà che la sala sia nella tabella Sala o Piscina.

```
CREATE TABLE IF NOT EXISTS IscrizioneCorsi(
Cliente char(50) not null,
Corso char(20) not null,
PRIMARY KEY(Cliente,Corso),
CONSTRAINT IscrizioneCorso
FOREIGN KEY (Corso)
REFERENCES Corso(CodCorso)
ON DELETE CASCADE
ON UPDATE NO ACTION,
CONSTRAINT IscrizioneCliente
FOREIGN KEY (Cliente)
REFERENCES Cliente(CodFiscale)
ON DELETE CASCADE
ON UPDATE NO ACTION
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: Cliente, CodCorso

Chiave primaria: Cliente, CodCorso
Vincoli Integrita Referenziale: IscrizioneCorso, IscrizioneCliente
Dipendenze Funzionali:
Cliente, CodCorso sono chiave.
Note:
La tabella memorizza tutte le iscrizioni dei clienti ai corsi.

```
CREATE TABLE IF NOT EXISTS CalendarioLezioni (
CodCorso char(10) not null,
GiornoSettimana char(15) not null,
OrarioInizio time not null,
OrarioFine time not null,
PRIMARY KEY (CodCorso,GiornoSettimana,OrarioInizio),
CONSTRAINT CorsoRiferimento
FOREIGN KEY (CodCorso)
REFERENCES Corso(CodCorso)
ON DELETE CASCADE
ON UPDATE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: Cliente, CodCorso
Chiave primaria: CodCorso, GiornoSettimana, OrarioInizio, OrarioFine
Vincoli Integrita Referenziale: CorsoRiferimento
Dipendenze Funzionali:

CodCorso, GiornoSettimana, OrarioInizio→ OrarioFine
Note:
L'inserimento in CalendarioLezioni verrà disciplinato dal trigger ControllaCorsi2, che verificherà la disponibilità del tutor nel giorno della settimana e nella fascia oraria prevista dal calendario.

```
CREATE TABLE IF NOT EXISTS AccessoSala_Medica (
Cliente CHAR(16) not null,
Centro CHAR(50) not null,
DataAccesso DATE not null,
OrarioAccesso TIME not null,
OrarioUscita TIME not null,
PRIMARY KEY (Cliente,DataAccesso,OrarioAccesso),
CONSTRAINT ClienteAccesso_SalaMedica
FOREIGN KEY (Cliente)
REFERENCES Cliente(CodFiscale)
ON DELETE CASCADE
ON UPDATE CASCADE,
CONSTRAINT CentroAccesso_SalaMedica
FOREIGN KEY (Centro)
REFERENCES Centro(CodCentro)
ON DELETE NO ACTION
ON UPDATE NO ACTION
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: Cliente, Centro, DataAccesso, OrarioAccesso, OrarioUscita
Chiave primaria: Cliente, DataAccesso, OrarioAccesso
Vincoli Integrita Referenziale: CentroAccesso_SalaMedica, ClienteAccesso_SalaMedica
Dipendenze Funzionali:
Cliente, DataAccesso, OrarioAccesso→ OrarioUscita, Centro
Note:
La tabella ha il compito di registrare tutti gli accessi dei clienti alla sala medica del centro. Da non confondere con accesso centro, si trovano nello stesso edificio ma hanno due ingressi differenti.

```
CREATE TABLE IF NOT EXISTS Log_SalaMedica (
Cliente CHAR(16) not null,
Centro CHAR(50) not null,
DataAccesso DATE not null,
OrarioAccesso TIME not null,
PRIMARY KEY (Cliente)
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: Cliente, Centro, DataAccesso, OrarioAccesso, OrarioUscita
Chiave primaria: Cliente
Vincoli Integrita Referenziale: -
Dipendenze Funzionali:
Cliente→ Centro, DataAccesso, OrarioAccesso

Note:
Come le precendenti Log, serve a memorizzare gli accessi giornalieri.

```
CREATE TABLE IF NOT EXISTS Accesso_Pagato (
Cliente CHAR(16) not null,
Sala CHAR(20) not null,
Prezzo DOUBLE not null,
DataAccesso DATE not null,
OrarioIngresso TIME not null,
OrarioUscita TIME,
PRIMARY KEY(Cliente,DataAccesso,OrarioIngresso),
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: Cliente, DataAccesso, OrarioIngresso, OrarioUscita, Sala

Chiave primaria: Cliente, Sala, DataAccesso, OrarioIngresso
Vincoli Integrita Referenziale: Cliente_AccessoPagato, Sala_AccessoPagato
Dipendenze Funzionali:
Cliente, Sala, DataAccesso, OrarioIngresso→ OrarioUscita
Note:
La tabella memorizza tutti gli accessi fatti al centro, in una sala specifica, sotto compenso. Coloro che usufruiscono di questo tipo di accessi sono di solito i senza contratto e coloro che hanno un contratto con priorità più bassa della sala stessa o, se si tratta di una piscina, non ha il permesso per accedervi da contratto.

```
CREATE TABLE IF NOT EXISTS Log_AccessoPagato (
Cliente char(16) not null,
Sala char(20) not null,
DataAccesso date not null,
OrarioAccesso time not null,
PRIMARY KEY(Cliente)
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: Cliente
Chiave primaria: Cliente,
Vincoli Integrita Referenziale: -
Dipendenze Funzionali:
Cliente→ Centro, DataAccesso, OrarioIngresso.
Note: -

Stored Procedures

Le stored procedures dell'area servizi sono incentrate prevalentemente sulla registrazione degli accessi ai centri/sale. Il lavoro maggiore comunque viene fatto dai due trigger ControllaAccessoCentro e ControllaAccessoSala di cui parleremo nell'apposita area

RegistraAccesso_Centro()

```
DROP PROCEDURE IF EXISTS RegistraAccesso_Centro;
DELIMITER $$
CREATE PROCEDURE RegistraAccesso_Centro (IN _Cliente VARCHAR(50) ,IN _Centro VARCHAR(50),
                                                    IN _DataAccesso DATE , IN _OrarioAccesso TIME)
BEGIN

  DECLARE Armadietto_Accesso VARCHAR(20) DEFAULT '';
  DECLARE Contatore INT DEFAULT 0;
  DECLARE Password_Armadietto VARCHAR(8) DEFAULT '';
  DECLARE ElementoPass INT DEFAULT 0;
  DECLARE _Sesso VARCHAR(1) DEFAULT '';

  SELECT Sesso INTO _Sesso
  FROM Cliente
  WHERE CodFiscale=_Cliente;

  SELECT A.CodArmadietto INTO Armadietto_Accesso
  FROM Armadietto A INNER JOIN Spogliatoio S
   ON A.IDSpogliatoio=S.CodSpogliatoio
  WHERE A.Occupato=0
   AND S.Centro=_Centro
   AND S.SessoAccedenti=_Sesso
  LIMIT 1;

  IF (Armadietto_Accesso<>'' AND Armadietto_Accesso IS NOT NULL) THEN
    BEGIN
    WHILE Contatore<8 DO
      BEGIN
              SELECT FLOOR(Rand()*10) INTO ElementoPass;
       SET Password_Armadietto=CONCAT(Password_Armadietto,ElementoPass);
       SET Contatore=Contatore+1;

      END ;
            END WHILE;
        INSERT INTO Log_Accesso
       (Cliente,Centro,DataAccesso,OrarioAccesso,ArmadiettoAssegnato,PasswordArmadietto);

    UPDATE Armadietto
    SET Occupato=1
    WHERE CodArmadietto=Armadietto_Accesso;

    END ;

  ELSE

  INSERT INTO Log_Accesso
(Cliente,Centro,DataAccesso,OrarioAccesso,ArmadiettoAssegnato,PasswordArmadietto)
  VALUES (_Cliente,_Centro,_DataAccesso,_OrarioAccesso,NULL,NULL);
END IF;

END $$
DELIMITER ;
```

Descrizione: La procedure ha il compito principale di registrare l'accesso al centro del cliente non appena supera il tornello di ingresso. Per prima cosa si trova il primo armadietto libero da assegnare al cliente,

cosicchè possa depositare i suoi beni. Se il sistema rileva un armadietto libero, si entra in un LOOP che randomizza una password valore per valore finchè non ne totalizza 8. Finito il ciclo, si inserisce dentro il Log_Accesso e ci rimarrà finchè il cliente non esce. Se invece il sistema non rileva alcun armadietto libero, allora si prosegue direttamente con l'inserimento nel Log e con il conseguente controllo dell'accesso.

RegistraUscita_Centro()

```
DROP PROCEDURE IF EXISTS RegistraUscita_Centro;
DELIMITER $$
CREATE PROCEDURE RegistraUscita_Centro(IN _Cliente VARCHAR(50), IN _OrarioUscita TIME )
BEGIN

  DECLARE _OrarioAccesso TIME;
  DECLARE _DataAccesso DATE;
  DECLARE _ArmadiettoAssegnato VARCHAR(50) DEFAULT '';
  DECLARE _PasswordArmadietto VARCHAR(8) DEFAULT '';
  DECLARE _Centro VARCHAR(20) DEFAULT '';



  SELECT OrarioAccesso,DataAccesso,ArmadiettoAssegnato,PasswordArmadietto,Centro INTO
  _OrarioAccesso,_DataAccesso, _ArmadiettoAssegnato, _PasswordArmadietto,_Centro
  FROM Log_Accesso
  WHERE Cliente=_Cliente;


  INSERT INTO AccessoCentro
  VALUES
(_Cliente,_Centro,_DataAccesso,_OrarioAccesso,_ArmadiettoAssegnato,_PasswordArmadietto,_OrarioUscita);

  UPDATE Armadietto
  SET Occupato=0
  WHERE CodArmadietto=_ArmadiettoAssegnato;


  DELETE
  FROM Log_Accesso
  WHERE Cliente=_Cliente;


END $$
DELIMITER ;
```

Descrizione: RegistraUscita_Centro() ovviamente finisce ciò che RegistraAccesso_Centro() aveva iniziato. Appena il cliente supera il tornello per uscire, la stored verrà chiamata dallo smartwatch e verrà svuotato il Log_Accesso e si inserirà il suo contenuto nella tabella permanente AccessoCentro. Ovviamente all'armadietto assegnato verrà reimpostato il valore 0 nel campo Occupato.

RegistraAccesso_Sala()

```
DROP PROCEDURE IF EXISTS RegistraAccesso_Sala;
```

```
DELIMITER $$
CREATE PROCEDURE RegistraAccesso_Sala (IN _Cliente VARCHAR(50), IN _Sala VARCHAR(20),
                                       IN _DataAccesso DATE, IN _OrarioAccesso TIME)
BEGIN

  INSERT INTO Log_Sala (Cliente,Sala,DataAccesso,OrarioAccesso)
  VALUES (_Cliente,_Sala,_DataAccesso,_OrarioAccesso);

END $$
DELIMITER ;
```

Descrizione: La procedure è molto semplice perchè il resto del lavoro è svolto dal trigger ControllaAccesso_Sala.

RegistraUscita_Sala()

```
DROP PROCEDURE IF EXISTS RegistraUscita_Sala;
DELIMITER $$
CREATE PROCEDURE RegistraUscita_Sala (IN _Cliente VARCHAR(50),IN _OrarioUscita TIME)
BEGIN

  DECLARE _OrarioAccesso TIME;
  DECLARE _DataAccesso DATE;
  DECLARE _Sala VARCHAR(20) DEFAULT '';
  DECLARE _Esiste INT DEFAULT 0;
  DECLARE _EsisteAccesso INT DEFAULT 0;


  SELECT OrarioAccesso,DataAccesso,Sala INTO _OrarioAccesso,_DataAccesso,_Sala
  FROM Log_Sala
  WHERE Cliente=_Cliente;


  IF (_Sala IS NULL OR _Sala=' ') THEN
    SIGNAL SQLSTATE'45000'
    SET MESSAGE_TEXT='Il cliente non può uscire se non è mai stato nella sala!';
  END IF;


  SELECT COUNT(*) INTO _Esiste
  FROM AccessoSala
  WHERE DataAccesso=_DataAccesso
   AND OrarioAccesso=(SELECT MAX(AC.OrarioAccesso)
                      FROM AccessoSala AC
                      WHERE AC.Cliente=_Cliente
                        AND AC.DataAccesso=_DataAccesso)
   AND _OrarioUscita BETWEEN OrarioAccesso AND OrarioUscita
   AND Cliente=_Cliente;

   IF (_Esiste>0) THEN
     SIGNAL SQLSTATE'45000'
     SET MESSAGE_TEXT='Si sta inserendo un uscita non lecita!';
         END IF;

  INSERT INTO AccessoSala
```

```
  VALUES (_Cliente,_Sala,_DataAccesso,_OrarioAccesso,_OrarioUscita);

  DELETE
  FROM Log_Sala
  WHERE Cliente=_Cliente;

END $$
DELIMITER ;
```

Descrizione: RegistraUscita_Sala come prima cosa prende i valori dell'accesso del cliente dal Log_Sala. Ovviamente se il cliente non ha effettuato l'accesso nella sala, i valori restituiti saranno NULL. Se cosi fosse la procedure restituirebbe un errore. Se invece l'accesso si sta inserendo in un tempo futuro rispetto a quando è avvenuto effettivamente l'accesso, si controlla se esistono accessi in quella data e in quella sala nella tabella AccessoSala. Se cosi fosse si andrebbe incontro ad un errore. Il funzionamento corretto porterebbe all'inserimento dell'accesso nella tabella permanente e il cancellamento del cliente e del suo accesso dal Log_Sala.

InserisciIngresso_Pagato()

```
DROP PROCEDURE IF EXISTS InserisciIngresso_Pagato;
DELIMITER $$
CREATE PROCEDURE InserisciIngresso_Pagato (IN _Cliente VARCHAR(50), IN _DataAccesso DATE,
                                                    IN _OrarioAccesso TIMEIN _Sala VARCHAR(20))
BEGIN

  DECLARE _Tipologia VARCHAR(25) DEFAULT '';
  DECLARE _Priorita INT DEFAULT 0;
  DECLARE _PrioritaSala INT DEFAULT 0;
  DECLARE _AccessoPiscina BOOL DEFAULT 0;
  DECLARE _Contratto VARCHAR(25) DEFAULT '';
  DECLARE Centro_Sala VARCHAR(20) DEFAULT '';
  DECLARE MaxIngressi INT DEFAULT 0;

 SELECT CodContratto,Tipologia INTO _Contratto,_Tipologia
 FROM Contratto
 WHERE Cliente=_Cliente
  AND  _DataAccesso BETWEEN DataSottoscrizione AND( DataSottoscrizione + INTERVAL DuratainMesi
MONTH );

 IF (_Contratto IS NOT NULL OR _Contratto<>'') THEN

   BEGIN

     IF (_Tipologia='Personalizzato') THEN
       BEGIN
                    IF (_Sala IN (SELECT CodSala FROM Sala WHERE TipoSala<>'Piscina')) THEN
         BEGIN

          SELECT Centro,AbbonamentoMinimo INTO Centro_Sala,_PrioritaSala
          FROM Sala
          WHERE CodSala=_Sala;

          SELECT Priorita INTO _Priorita
          FROM AbbonamentoPersonalizzato
          WHERE Centro=Centro_Sala
```

```
        AND Contratto=_Contratto;

                        IF (_Priorita<_PrioritaSala) THEN

            INSERT INTO Log_AccessoPagato
            VALUES (_Cliente,_Sala,_DataAccesso,_OrarioAccesso);

                        ELSE
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT='Il cliente ha già il permesso per entrare';
                        END IF;
        END ;
                    ELSE
                     BEGIN
         SELECT Centro INTO Centro_Sala
                            FROM Sala
         WHERE CodSala=_Sala;

         SELECT AccessoPiscine, NumeroMaxIngressoPiscineMese INTO _AccessoPiscina,MaxIngressi
         FROM AbbonamentoPersonalizzato
         WHERE Centro=Centro_Sala
                            AND Contratto=_Contratto;

                        IF (_AccessoPiscina=0 OR (_AccessoPiscina=1 AND MaxIngressi<(
                                                SELECT COUNT(*) FROM AccessoSala
                                                WHERE Sala=_Sala
                                                AND YEAR(_DataAccesso)=YEAR(DataAccesso)
                                                AND MONTH(_DataAccesso)=MONTH(DataAccesso))))
THEN

            INSERT INTO Log_AccessoPagato
            VALUES (_Cliente,_Sala,_DataAccesso,_OrarioAccesso);

           ELSE

            SIGNAL SQLSTATE '45000'
           SET MESSAGE_TEXT='Il cliente ha gia il permesso per entrare in piscina';

           END IF;

                        END;
                END IF;
                    END;
            ELSE
      BEGIN
                IF (_Sala IN (SELECT CodSala FROM Sala WHERE TipoSala<>'Piscina')) THEN
        BEGIN
         SELECT AbbonamentoMinimo INTO _PrioritaSala
                        FROM Sala
        WHERE CodSala=_Sala;

        SELECT Priorita INTO _Priorita
        FROM AbbonamentoStandard
        WHERE NomeAbbonamento=_Tipologia;

                        IF (_Priorita<_PrioritaSala) THEN

           INSERT INTO Log_AccessoPagato
           VALUES (_Cliente,_Sala,_DataAccesso,_OrarioAccesso);
```

```
                        ELSE
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT='Il cliente ha già il permesso per entrare';
                         END IF;
          END ;
                      ELSE
         BEGIN
          SELECT Centro INTO Centro_Sala
          FROM Sala
          WHERE CodSala=_Sala;

          SELECT AccessoPiscine,MaxNumeroIngressiSettimanali INTO _AccessoPiscina,MaxIngressi
          FROM AbbonamentoStandard
          WHERE NomeAbbonamento=_Tipologia;

                        IF (_AccessoPiscina=0  OR (_AccessoPiscina=1 AND MaxIngressi<
                                (SELECT COUNT(*) FROM AccessoSala
                                 WHERE Sala=_Sala
                                 AND YEAR(_DataAccesso)=YEAR(DataAccesso)
                                 AND MONTH(_DataAccesso)=MONTH(DataAccesso)))) THEN

            INSERT INTO Log_AccessoPagato
            VALUES (_Cliente,_Sala,_DataAccesso,_OrarioAccesso);

           ELSE

            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT='Il cliente ha gia il permesso per entrare in piscina';

           END IF;

                        END;
                  END IF;
                END ;
           END IF;
          END ;
          ELSE
    INSERT INTO Log_AccessoPagato
           VALUES (_Cliente,_Sala,_DataAccesso,_OrarioAccesso);
         END IF;
END $$
DELIMITER ;
```

Descrizione: La procedure ha il compito di valutare diversi casi. Per prima cosa controlla se la persona che vuole accedere ha un contratto attivo. Dopodichè controlla il tipo di contratto e successivamente controlla se la sala è una sala standard o è una piscina. Di conseguenza la procedure estrapolerà da AutorizzazionePersonalizzata tutto ciò che le servirà per l'occasione. Lo stesso avverrà con i contratti standard. L'unica eccezione è per i senza contratto, che possono accedere senza problemi.

InserisciUscita_Pagato()

```
DROP PROCEDURE IF EXISTS InserisciUscita_Pagato;
```

```
DELIMITER $$
CREATE PROCEDURE InserisciUscita_Pagato (IN _Cliente VARCHAR(16), IN _Sala VARCHAR(20),IN
_OrarioUscita TIME)
BEGIN

DECLARE _DataAccesso DATE;
DECLARE _OrarioIngresso TIME ;


IF (EXISTS (SELECT *
          FROM Log_AccessoPagato
          WHERE Cliente=_Cliente
           AND Sala=_Sala))     THEN
BEGIN
  IF (_Sala IN (SELECT CodSala FROM Sala)) THEN
   BEGIN
   SELECT TariffaAccessoSingolo INTO @Tariffa
   FROM Sala
   WHERE CodSala=_Sala;

   SELECT OrarioAccesso,DataAccesso INTO _OrarioIngresso,_DataAccesso
   FROM Log_AccessoPagato
   WHERE Cliente=_Cliente AND Sala=_Sala;

   SET @ImportodaPagare=@Tariffa*((TIME_TO_SEC(_OrarioUscita)-TIME_TO_SEC(_OrarioIngresso))/3600);

   INSERT INTO Accesso_Pagato
   VALUES (_Cliente,_Sala,@ImportodaPagare,_DataAccesso,_OrarioIngresso,_OrarioUscita);

    END ;
  ELSE

     SIGNAL SQLSTATE '45000'
     SET MESSAGE_TEXT='Il cliente non ha accessi da pagare';

  END IF;
  END;
ELSE

  SIGNAL SQLSTATE '45000'
  SET MESSAGE_TEXT='Il cliente non ha accessi da pagare';

END IF;
END $$
DELIMITER ;
```

Descrizione: Si controlla se nel Log degli accessi pagati è presente il cliente. Se cosi non fosse verrà restituito un errore dalla stored. Se invece il cliente è presente nel log, allora si calcola il prezzo prendendo la TariffaAccessoSingolo o la TariffaOraria rispettivamente da Sala o da Piscina. Il calcolo poi viene effettuato convertento le ore di accesso in secondi, dividendo per 3600 cosi da trovare le ore. Dopo aver ricavato le ore , si moltiplicano per la tariffa.

RegistraAccesso_SalaMedica()

```
DROP PROCEDURE IF EXISTS RegistraAccesso_SalaMedica;
 DELIMITER $$
 CREATE PROCEDURE RegistraAccesso_SalaMedica(IN _Cliente CHAR(16), IN _Centro CHAR(50),IN
_DataAccesso DATE, IN _OrarioAccesso TIME)
 BEGIN

    IF NOT EXISTS (SELECT * FROM Contratto WHERE _Cliente=Cliente
                          AND _DataAccesso BETWEEN DataSottoscrizione
                          AND DataSottoscrizione + INTERVAL DuratainMesi  MONTH) THEN
       SIGNAL SQLSTATE '45000'
       SET MESSAGE_TEXT ='Il cliente non ha contratto valido';
    END IF;

   INSERT INTO Log_SalaMedica
   VALUES (_Cliente,_Centro,_DataAccesso,_OrarioAccesso);

END $$
DELIMITER ;
```

Descrizione: La stored si limita a inserire nel log di accesso la tupla del cliente dopo aver verificato l'esistenza di un contratto valido per il cliente.

RegistraUscita_SalaMedica()

```
DROP PROCEDURE IF EXISTS RegistraUscita_SalaMedica;
DELIMITER $$
CREATE PROCEDURE RegistraUscita_SalaMedica (IN _Cliente CHAR(16), IN _OrarioUscita TIME)
BEGIN

  DECLARE _DataAccessoSala DATE;
  DECLARE _OrarioAccesso TIME;
  DECLARE _Centro CHAR(20) DEFAULT '';

  SELECT Centro,DataAccesso,OrarioAccesso INTO _Centro,_DataAccessoSala,_OrarioAccesso
  FROM Log_SalaMedica
  WHERE Cliente=_Cliente;

  INSERT INTO AccessoSala_Medica
  VALUES (_Cliente,_Centro,_DataAccessoSala,_OrarioAccesso,_OrarioUscita);

  DELETE
  FROM Log_SalaMedica
  WHERE Cliente=_Cliente;

END $$
DELIMITER ;
```

Descrizione: La stored si limita a riprendere il contenuto del log e ad inserirlo nella tabella permanente AccessoSala_Medica.

Triggers

ControllaOrario

```
DROP TRIGGER IF EXISTS ControllaOrario;
DELIMITER $$
CREATE TRIGGER ControllaOrario
BEFORE INSERT ON Turnazione
FOR EACH ROW
BEGIN
  DECLARE OraInizio Time;
  DECLARE OraFine Time;
  DECLARE TurniPrecedenti int DEFAULT 0;
  DECLARE OreGiornata INT DEFAULT 0;

  SELECT OrarioApertura,OrarioChiusura INTO OraInizio,OraFine
  FROM OrarioAperturaCentro
  WHERE new.Centro=Centro AND GiornoSettimana=new.GiornoSettimana;

  IF (new.InizioTurno>=OraInizio AND new.InizioTurno<OraFine) AND (new.FineTurno>OraInizio AND
new.FineTurno<=OraFine) THEN
   BEGIN
   SET OreGiornata=((SELECT SUM(to_seconds(FineTurno)-To_seconds(InizioTurno)) as Secondi
                          FROM Turnazione
                          WHERE Dipendente=new.Dipendente
                           AND GiornoSettimana=new.GiornoSettimana) + (to_seconds(new.FineTurno)-
                               To_seconds(new.InizioTurno)));
           IF OreGiornata>28800 THEN
     SIGNAL SQLSTATE '45000'
     SET MESSAGE_TEXT='Inserimento non valido';
     ELSE
     BEGIN


                    IF EXISTS (SELECT *
                                  FROM Turnazione
                                   WHERE (Dipendente=new.Dipendente
                                   AND GiornoSettimana=new.GiornoSettimana)
                                   AND (((new.InizioTurno >InizioTurno AND new.InizioTurno<FineTurno) OR
                                           (new.FineTurno >InizioTurno AND new.FineTurno<FineTurno))
                                   OR ((InizioTurno> new.InizioTurno AND InizioTurno< new.FineTurno) OR
                                           (FineTurno >new.InizioTurno AND FineTurno<new.FineTurno)))) THEN

       SIGNAL SQLSTATE '45000'
       SET MESSAGE_TEXT='Inserimento non valido';
                    END IF;
            END;
     END IF;
          END;
 ELSE
       SIGNAL SQLSTATE '45000'
       SET MESSAGE_TEXT='Inserimento non valido';
 END IF;
END $$
 DELIMITER ;
```

Descrizione: Il trigger verifica se il turno del dipendente che stiamo inserendo è valido e non vada in conflitto con altri turni, con l'orario di apertura del centro e il dipendente non superi li 8 ore giornaliere. Come prima cosa il trigger si ricava l'orario di apertura del centro nel giorno che stiamo inserendo, poi si calcola le ore della giornata del tutor aggiunte a quelle dei turni dello stesso giorno, gia presenti. Se si superano le 8 ore il trigger restituirà un errore. Dopo aver superato questo controllo si verifica se questo turno va in conflitto con altri già presenti nel database. Se ciò si verifica, annulla l'insert.

ControllaCorsi1

```
DROP TRIGGER IF EXISTS ControllaCorsi1;
DELIMITER $$
CREATE TRIGGER ControllaCorsi1
BEFORE INSERT ON Corso
FOR EACH ROW
BEGIN

DECLARE GiaPresente INT DEFAULT 0;
DECLARE CentroI VARCHAR(10) DEFAULT '';
DECLARE Valido INT DEFAULT 0;
DECLARE Verifica INT DEFAULT 0;


IF NEW.NomeDisciplina NOT IN (SELECT Disciplina
                                    FROM TipologiaCorso_Sala T INNER JOIN Sala S
                                    ON T.TipoSala=S.TipoSala
                                    WHERE S.CodSala=NEW.Sala) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT='Disciplina non attinente al tipo sala o non esistente!';
       END IF;


IF EXISTS(SELECT *
            FROM Corso
            WHERE Istruttore=new.Istruttore
            AND FineCorso>NEW.InizioCorso
            AND Sala=new.Sala
            AND NomeDisciplina=new.NomeDisciplina
            AND LivelloInsegnamento=new.LivelloInsegnamento) THEN

 SIGNAL SQLSTATE '45000'
 SET MESSAGE_TEXT='Corso già esistente!';
END IF;


SELECT Centro INTO CentroI
FROM  Sala
WHERE CodSala=new.Sala;



IF NOT EXISTS(SELECT *
                    FROM Turnazione
                    WHERE Dipendente=new.Istruttore
                    AND Centro=CentroI) THEN
 SIGNAL SQLSTATE '45000'
```

```
 SET MESSAGE_TEXT='Il tutor non lavora nel centro!';
END IF;
END $$
DELIMITER ;
```

Descrizione: Il trigger inizialmente mi controlla se la sala è valida per il corso che si vuole iniziare. Se cosi non fosse, il trigger annulla l'inserimento. Successivamente si controlla che non esista un corso identico a quello già inserito, tenuto dallo stesso istruttore. Infine controlla se il tutor lavora effettivamente nel centro.

ControllaCorsi2

```
DROP TRIGGER IF EXISTS ControllaCorsi2;
DELIMITER $$
CREATE TRIGGER ControllaCorsi2
BEFORE INSERT ON CalendarioLezioni
FOR EACH ROW
BEGIN
 DECLARE ContaLezioni INT DEFAULT 0;
 DECLARE CentroI VARCHAR(10) DEFAULT '';
 DECLARE IstruttoreCorso VARCHAR(16) DEFAULT '';
 DECLARE Inizio TIME DEFAULT '';
 DECLARE Fine TIME DEFAULT '';
 DECLARE Lavora INT DEFAULT 0;
 DECLARE Sala_Istruttore VARCHAR(50) DEFAULT '';
  DECLARE ContaLezioni_AltriTutor INT DEFAULT 0;

 SELECT C.Istruttore INTO IstruttoreCorso
 FROM Corso C
 WHERE CodCorso=new.CodCorso;

 SELECT S.Centro,S.CodSala INTO CentroI,Sala_Istruttore
 FROM Sala S INNER JOIN Corso C
 ON S.CodSala=C.Sala
 WHERE C.CodCorso=NEW.CodCorso;

 SELECT InizioTurno,FineTurno,Count(*) INTO Inizio,Fine,Lavora
  FROM Turnazione
  WHERE Dipendente=IstruttoreCorso
   AND GiornoSettimana=new.GiornoSettimana
   AND Centro=CentroI
   AND (InizioTurno<=new.OrarioInizio AND InizioTurno<new.OrarioFine) AND (FineTurno>=new.OrarioFine AND
FineTurno>new.OrarioInizio);

 IF (Lavora=0) THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT='Inserimento annullato, non in linea con il turno del tutor';
   END IF;



 SELECT COUNT(*) INTO ContaLezioni
 FROM CalendarioLezioni CL INNER JOIN Corso C
   ON C.CodCorso=CL.CodCorso
 WHERE (C.Sala=Sala_Istruttore
```

```
  AND CL.GiornoSettimana=new.GiornoSettimana)
  AND ((Inizio<=new.OrarioInizio AND Inizio<new.OrarioFine) AND (Fine>=new.OrarioFine AND
Fine>new.OrarioInizio))
  AND (((CL.OrarioInizio<=new.OrarioInizio AND CL.OrarioFine>new.OrarioInizio) OR
(CL.OrarioFine>=new.OrarioFine AND CL.OrarioFine>=new.OrarioInizio))
    OR (CL.OrarioInizio>=new.OrarioInizio AND CL.OrarioFine<new.OrarioFine));


  IF ContaLezioni>0 THEN
   SIGNAL SQLSTATE '45000'
   SET MESSAGE_TEXT='Inserimento annullato, non in linea con le altre lezioni dello stesso istruttore o altri tutor!';
  END IF;


END $$
DELIMITER ;
```

Descrizione: Questo trigger a differenza dell'altro, fa in modo che le lezioni non entrino in conflitto con i turni del tutor. Inoltre controlla che la lezione stessa non vada in conflitto con altre lezioni dello stesso tutor.

ControllaTutor

```
DROP TRIGGER IF EXISTS ControllaTutor;
DELIMITER $$
CREATE TRIGGER ControllaTutor
BEFORE INSERT ON SchedaAllenamento
FOR EACH ROW
BEGIN

 DECLARE CentroCliente VARCHAR(20) DEFAULT '';
 DECLARE Verifica INT DEFAULT 0;
 DECLARE OrarioAccesso_Cliente TIME;
 DECLARE GiornoEmissione_Cliente VARCHAR(20);


  SELECT Centro,DAYNAME(DataAccesso),OrarioAccesso
         INTO  CentroCliente,GiornoEmissione_Cliente,OrarioAccesso_Cliente
  FROM AccessoCentro
  WHERE Cliente=NEW.Cliente
  AND DataAccesso=NEW.DataEmissione;


  IF (OrarioAccesso_Cliente IS NULL) THEN
   SIGNAL SQLSTATE '45000'
   SET MESSAGE_TEXT='Il cliente non è acceduto nel giorno inserito';
  END IF;



  IF NOT EXISTS ( SELECT *
                  FROM Turnazione
                  WHERE Dipendente=NEW.Tutor
                  AND GiornoSettimana=GiornoEmissione_Cliente
                  AND Centro=CentroCliente
```

```
                    AND OrarioAccesso_Cliente BETWEEN InizioTurno AND FineTurno)  THEN

    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT='Il tutor non lavora quel giorno';
  END IF;

END $$
DELIMITER ;
```

Descrizione: ControllaTutor verifica che la scheda allenamento che si sta inserendo sia stata emessa durante un suo turno di lavoro.

ControllaMedico

```
DROP TRIGGER IF EXISTS ControllaMedico;
 DELIMITER $$
 CREATE TRIGGER ControllaMedico
 BEFORE INSERT ON SchedaAlimentazione
 FOR EACH ROW
 BEGIN

  DECLARE CentroCliente VARCHAR(20) DEFAULT '';
  DECLARE Verifica INT DEFAULT 0;
  DECLARE OrarioAccesso_Cliente TIME;
  DECLARE GiornoEmissione_Cliente VARCHAR(20);


    SELECT Centro,DAYNAME(DataAccesso),OrarioAccesso INTO
CentroCliente,GiornoEmissione_Cliente,OrarioAccesso_Cliente
    FROM AccessoSala_Medica
    WHERE Cliente=NEW.Cliente
      AND DataAccesso=NEW.DataEmissione;

 IF (OrarioAccesso_Cliente IS NULL ) THEN
   SIGNAL SQLSTATE '45000'
   SET MESSAGE_TEXT='Il cliente non è acceduto alla sala medica quel giorno!';
 END IF;


   IF NOT EXISTS (SELECT *
              FROM Turnazione
              WHERE Dipendente=NEW.Medico
              AND GiornoSettimana=GiornoEmissione_Cliente
              AND Centro=CentroCliente
              AND OrarioAccesso_Cliente BETWEEN InizioTurno AND FineTurno) THEN

    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT='Il medico non lavora nel centro / quel giorno!';
   END IF;

END $$
DELIMITER ;
```

Descrizione: Stessa cosa di ControllaTutor, solo che fa riferimento al medico e all'emissione della scheda alimentazione del cliente.

ControllaConsulente

```
DROP TRIGGER IF EXISTS ControllaConsulente;
 DELIMITER $$
 CREATE TRIGGER ControllaConsulente
 BEFORE INSERT ON Contratto
 FOR EACH ROW
 BEGIN
 DECLARE Lavoro INT DEFAULT 0;



 IF NOT EXISTS(SELECT *
                    FROM Turnazione T
                    WHERE T.Dipendente=new.Consulente
                    AND DAYNAME(new.DataSottoscrizione)=T.GiornoSettimana
                    AND T.Centro=new.SedeSottoscrizione) THEN
  SIGNAL SQLSTATE '45000'
  SET MESSAGE_TEXT='Il/la consulente non lavora in quel giorno/centro';
 END IF;

 END $$
 DELIMITER ;
```

Descrizione: Anche questo trigger è sulla falsa riga di quelli precendenti, solo che non è presente un controllo dell'ora e si riferisce al consulente e alla stipulazione di contratti.

ControllaAccessiCentro

```
 DROP TRIGGER IF EXISTS ControllaAccessiCentro;
 DELIMITER $$
 CREATE TRIGGER ControllaAccessiCentro
 BEFORE INSERT ON Log_Accesso
 FOR EACH ROW
 BEGIN

 DECLARE VolteSettimana INT DEFAULT 0;
 DECLARE NumeroMassimo INT DEFAULT 0;
 DECLARE Tip CHAR(15) DEFAULT '';
 DECLARE Contr CHAR(10) DEFAULT '';


 SELECT C.CodContratto,C.Tipologia INTO Contr,Tip
 FROM  Contratto C
 WHERE Cliente=new.Cliente
```

```
  AND   C.DataSottoscrizione=(SELECT MAX(CC.DataSottoscrizione) FROM Contratto CC WHERE
CC.Cliente=new.Cliente)
  AND NEW.DataAccesso BETWEEN C.DataSottoscrizione AND (C.DataSottoscrizione + INTERVAL
C.DuratainMesi MONTH);


 IF (new.Centro NOT IN (SELECT AC.Centro FROM AutorizzazioneCentro AC WHERE AC.Contratto=Contr))
       AND
      (new.Centro NOT IN (SELECT AP.Centro FROM AutorizzazionePersonalizzata AP
                                WHERE Contr=AP.Contratto))THEN
   SIGNAL SQLSTATE '45000'
   SET MESSAGE_TEXT='Il cliente non ha il permesso di entrare nel centro!';
 END IF;

 IF (NEW.Cliente NOT IN (SELECT Cliente FROM Visita WHERE DataVisita<=NEW.DataAccesso)) THEN

     SIGNAL SQLSTATE '45000'
     SET MESSAGE_TEXT='Il cliente non ha ancora effettuata alcuna visita';
 END IF;



  IF EXISTS (SELECT *
                 FROM AccessoCentro
                 WHERE Cliente=NEW.Cliente
                 AND DataAccesso=NEW.DataAccesso
                  AND NEW.OrarioAccesso BETWEEN OrarioAccesso AND OrarioUscita) THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT= 'Errore di sistema, inserimento accesso compreso in accesso precedente!';
  END IF;

  IF (Tip IN (SELECT NomeAbbonamento FROM AbbonamentoStandard)) THEN
  BEGIN

  SELECT MaxNumeroIngressiSettimanali INTO NumeroMassimo
  FROM AbbonamentoStandard
  WHERE NomeAbbonamento=Tip;
  END;

 ELSE
 BEGIN

  SELECT MaxNumeroIngressiSettimanali INTO NumeroMassimo
  FROM AutorizzazionePersonalizzata AP
  WHERE AP.Contratto=Contr
    AND AP.Centro=new.Centro;
 END;
 END IF;


 SELECT COUNT(*) INTO VolteSettimana
 FROM AccessoCentro AC INNER JOIN Contratto C
 ON AC.Cliente = C.Cliente
 WHERE C.CodContratto=Contr
 AND WEEK(DataAccesso,1)=WEEK(NEW.DataAccesso,1)
 AND YEAR(DataAccesso)=YEAR(NEW.DataAccesso);

 IF (VolteSettimana>=NumeroMassimo) THEN
   SIGNAL SQLSTATE '45000'
```

```
  SET MESSAGE_TEXT='Numero accessi settimanali superati!';
 END IF;

 END $$
 DELIMITER ;
```

Descrizione: Questo trigger insieme al prossimo, ControllaAccessiSala, sono i principali trigger di controllo degli accessi dei clienti. Prima di tutto si controlla se il giorno in cui c'è stato il presunto accesso il centro era aperto. Poi si prende il contratto a la tipologia e si verifica che il cliente abbia il permesso di accederci. Dopodichè si controlla se il cliente è stato visitato almeno una volta. L'ultimo controllo verifica che il cliente non abbia raggiunto il limite massimo di accessi alla settimana. In caso contrario, gli viene negato l'accesso.

ControllaAccessiSala

```
DROP TRIGGER IF EXISTS ControllaAccessiSala;
 DELIMITER $$
 CREATE TRIGGER ControllaAccessiSala
 BEFORE INSERT ON Log_Sala
 FOR EACH ROW
 BEGIN

  DECLARE Abbonamento_Sala INT DEFAULT 0;
  DECLARE PrioritaSala INT DEFAULT 0;
  DECLARE PrioritaCliente INT DEFAULT 0;
  DECLARE ContaAccessiPiscinaMese INT DEFAULT 0;
  DECLARE ContrattoCliente VARCHAR(50) DEFAULT '';
  DECLARE Tipologia_Contratto VARCHAR(50) DEFAULT '';
  DECLARE Accesso INT DEFAULT 0;
  DECLARE Gia_Inserito INT DEFAULT 0;
        DECLARE Gia_Inserito_Passato INT DEFAULT 0;
  DECLARE Iscritto INT DEFAULT 0;
  DECLARE Corso_Sala VARCHAR(20) DEFAULT '';
  DECLARE Nome_Corso VARCHAR(20) DEFAULT '';
  DECLARE Abilitato BOOL DEFAULT 0;
  DECLARE AccessiPiscine INT DEFAULT 0;


  SELECT C.CodContratto,C.Tipologia INTO ContrattoCliente,Tipologia_Contratto
  FROM Contratto C
  WHERE C.Cliente=NEW.Cliente
   AND C.DataSottoscrizione=(SELECT MAX(CC.DataSottoscrizione)
                             FROM Contratto CC
                             WHERE CC.Cliente=NEW.Cliente)
   AND NEW.DataAccesso BETWEEN C.DataSottoscrizione AND (C.DataSottoscrizione + INTERVAL
           C.DuratainMesi MONTH);

  IF (ContrattoCliente IS NULL OR ContrattoCliente='') THEN

    SIGNAL SQLSTATE '45000'
```

```
      SET MESSAGE_TEXT='Il cliente non ha un contratto attivo';

   END IF;

    SELECT AbbonamentoMinimo INTO PrioritaSala
           FROM Sala
           WHERE CodSala=NEW.Sala;

     IF (Tipologia_Contratto='Personalizzato') THEN
      BEGIN


                  SELECT AccessoPiscine, NumeroMaxIngressoPiscineMese,Priorita
                        INTO Abilitato,AccessiPiscine,PrioritaCliente
                  FROM AutorizzazionePersonalizzata
                  WHERE Contratto=ContrattoCliente;

             END;

      ELSE

       SELECT Priorita,AccessoPiscine,NumeroMaxIngressoPiscineMese INTO PrioritaCliente,Abilitato,AccessiPiscine
       FROM AbbonamentoStandard
       WHERE NomeAbbonamento=Tipologia_Contratto;

             END IF;


    IF  ((SELECT TipoSala FROM Sala WHERE CodSala=NEW.Sala)='Piscina' AND Abilitato=0) THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT='Il cliente non è abilitato ad entrare nella sala(piscina)';
           ELSE

      SELECT COUNT(*) INTO ContaAccessiPiscinaMese
      FROM AccessoSala A INNER JOIN Sala S
       ON S.CodSala=A.Sala
      WHERE YEAR(A.DataAccesso)=YEAR(NEW.DataAccesso)
       AND MONTH(A.DataAccesso)=MONTH(NEW.DataAccesso)
       AND A.Cliente=NEW.Cliente
       AND S.TipoSala='Piscina';

       IF (ContaAccessiPiscinaMese>AccessiPiscine) THEN
         SIGNAL SQLSTATE '45000'
         SET MESSAGE_TEXT='Limite accesso piscine raggunto!';
                  END IF;



      IF (PrioritaSala>PrioritaCliente) THEN
       SIGNAL SQLSTATE '45000'
       SET MESSAGE_TEXT='Cliente non abilitato ad entrare nella sala';
             END IF;



      SET @Errore=CONCAT('Errore di sistema, il cliente ',NEW.Cliente,' non risulta nel centro in data','
',NEW.DataAccesso);

             IF NOT EXISTS (SELECT *
```

```
                        FROM Log_Accesso
                        WHERE DataAccesso=NEW.DataAccesso) THEN
                SIGNAL SQLSTATE '45000'
                SET MESSAGE_TEXT=@Errore;
        ELSE

         BEGIN

                SET @Errore=CONCAT('Errore di sistema, il cliente ',NEW.Cliente,' si trova ancora nella sala
                 precedente!',' ',NEW.DataAccesso);

                IF EXISTS (SELECT *
                        FROM Log_Sala
                        WHERE Cliente=NEW.Cliente) THEN

      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT=@Errore;

    END IF;


    SELECT C.CodCorso,C.NomeDisciplina INTO Corso_Sala,Nome_Corso
    FROM Corso C INNER JOIN CalendarioLezioni CL
    ON C.CodCorso=CL.CodCorso
    WHERE CL.GiornoSettimana=DAYNAME(NEW.DataAccesso)
    AND NEW.DataAccesso BETWEEN C.InizioCorso AND C.FineCorso
    AND NEW.OrarioAccesso BETWEEN CL.OrarioInizio AND CL.OrarioFine;

     IF (Nome_Corso IS NOT NULL AND Nome_Corso<>'') THEN
      BEGIN

          SET @Errore=CONCAT('Utente',' ',NEW.Cliente,' non iscritto al corso di',' ',Nome_Corso);

                IF NOT EXISTS (SELECT *
                                FROM IscrizioneCorsi
                                WHERE Corso=Corso_Sala
                                AND Cliente=NEW.Cliente) THEN
       SIGNAL SQLSTATE '45000'
       SET MESSAGE_TEXT=@Errore;
      END IF;
      END;
                 END IF;
                END;
         END IF;
        END IF;
END $$
DELIMITER ;
```

Descrizione: Il trigger ha il compito di verificare che l'accesso alla sala del cliente sia lecito.Per prima cosa si ricava le informazioni essenziali del contratto valido in quel momento, altrimenti annulla l'inserimento, per il corretto controllo dell'accesso. Dopo essersi ricavato sia ila priorità del cliente che l'abbonamento minimo del centro, si verifica che la priorità del cliente sia maggiore o uguale a quella della sala, cosi da garantirgli l'accesso. Si verifica che la sala non sia una piscina, altrimenti si va a controllare se il cliente ha diritto ad accedere alle piscine e, in caso di esito positivo, controlla che non abbia superato il numero massimo di accessi alle piscine garantiti dal contratto. Successivamente si controlla se il cliente ha acceduto al centro, tramite il Log (si suppone quindi che l'inserimento non possa essere fatto dopo da terzi, ma che deve essere fatto tutto in giornata prima dell'uscita dal centro del cliente!) e che non entri in conflitto con un altro accesso ad un'altra sala. Come ultimo controllo si prende il corso che dovrebbe essere tenuto in quella fascia oraria e si verifica che il cliente sia iscritto.

IscriviaiCorsi

```
 DROP TRIGGER IF EXISTS IscriviaiCorsi;
DELIMITER $$
CREATE TRIGGER IscriviaiCorsi
BEFORE INSERT ON IscrizioneCorsi
FOR EACH ROW
BEGIN

  DECLARE Tipologia_Contratto VARCHAR(20) DEFAULT '';
  DECLARE Frequentazione BOOL DEFAULT 0;
  DECLARE Centro_Corso VARCHAR(20) DEFAULT '';
  DECLARE Contratto_Cliente VARCHAR(50) DEFAULT '';
  DECLARE MassimoIscritti INT DEFAULT 0;
  DECLARE SalaCorso VARCHAR(20) DEFAULT '';
  DECLARE Tipo_Sala VARCHAR(20) DEFAULT '';
  DECLARE PrioritaCliente INT DEFAULT 0;
  DECLARE Piscina BOOL DEFAULT 0;

 SELECT NumeroMaxPartecipanti INTO MassimoIscritti
 FROM Corso
 WHERE CodCorso=NEW.Corso;

 IF ((SELECT COUNT(*) FROM IscrizioneCorsi WHERE Corso=NEW.Corso)>=MassimoIscritti) THEN
   SIGNAL SQLSTATE '45000'
   SET MESSAGE_TEXT='Massimo numero di iscritti raggiunto';
 END IF;



  SELECT Tipologia INTO Tipologia_Contratto
  FROM Contratto
  WHERE Cliente=NEW.Cliente;

  IF (Tipologia_Contratto='Personalizzato') THEN

   BEGIN

    SELECT S.Centro,S.CodSala,S.TipoSala INTO Centro_Corso, SalaCorso,Tipo_Sala
    FROM Corso C INNER JOIN Sala S
```

```
    ON C.Sala=S.CodSala
        WHERE C.CodCorso=NEW.Corso;

   SELECT C.CodContratto INTO Contratto_Cliente
   FROM Contratto C
   WHERE C.Cliente=NEW.Cliente
    AND C.DataSottoscrizione=(SELECT MAX(C2.DataSottoscrizione)
                                                    FROM Contratto C2
                    WHERE C2.Cliente=NEW.Cliente);



   SELECT PossibilitaFrequentazioneCorsi,Priorita,AccessoPiscine INTO Frequentazione, PrioritaCliente, Piscina
   FROM AutorizzazionePersonalizzata
   WHERE Centro=Centro_Corso
    AND Contratto=Contratto_Cliente;

        IF  ((Tipo_Sala='Piscina' AND Piscina=0) AND (PrioritaCliente<(SELECT AbbonamentoMinimo FROM
Sala WHERE CodSala=SalaCorso))) THEN

      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT='Il cliente non ha la possibilità di entrare nella sala!';
          END IF;

   END;

  END IF;


 IF (Tipologia_Contratto='Silver') OR
   ( Tipologia_Contratto='Personalizzato' AND Frequentazione=0) THEN

      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT ='Il Cliente non ha il permesso di seguire corsi!';

  END IF;
END $$
DELIMITER ;
```

Descrizione: Il trigger semplicemente controlla l'iscrizione al corso di un cliente. Viene annullato in due casi: quando il cliente non prevede la frequentazione di corsi nel contratto e quando il numero di iscritti è uguale al massimo numero di partecipanti.

ControlloInserimento_Visita

```
DROP TRIGGER IF EXISTS ControlloInserimento_Visita;
DELIMITER $$
CREATE TRIGGER ControlloInserimento_Visita
BEFORE INSERT ON Visita
FOR EACH ROW
BEGIN
```

```
 DECLARE CentroCliente VARCHAR(50) DEFAULT '';

SELECT Centro INTO CentroCliente
FROM AccessoSala_Medica
WHERE DataAccesso=NEW.DataVisita
 AND NEW.OraVisita BETWEEN OrarioAccesso AND OrarioUscita
 AND Cliente=NEW.Cliente;

 IF (CentroCliente IS NULL OR CentroCliente='') THEN

  SELECT Centro INTO CentroCliente
  FROM Log_SalaMedica
  WHERE DataAccesso=NEW.DataVisita
  AND Cliente=NEW.Cliente
  AND NEW.OraVisita>=OrarioAccesso;

 END IF;


 IF (CentroCliente IS NULL OR CentroCliente='') THEN
   BEGIN
     SET @errore=CONCAT('Il cliente ',NEW.Cliente,' non ha effettuato accessi in data ',NEW.DataVisita);
     SIGNAL SQLSTATE '45000'
     SET MESSAGE_TEXT=@errore;
   END;
END IF;


IF (NOT EXISTS (SELECT * FROM Turnazione
              WHERE Dipendente=NEW.Medico
               AND DAYNAME(NEW.DataVisita)=GiornoSettimana
               AND NEW.OraVisita BETWEEN InizioTurno AND FineTurno
               AND Centro=CentroCliente)) THEN

   SIGNAL SQLSTATE '45000'
   SET MESSAGE_TEXT='Il tutor non lavora nel centro del cliente';
END IF;
END $$
DELIMITER ;
```

Descrizione: Il trigger controlla se nel giorno in cui si è svolta la visita, il medico lavorava in quel centro. Il funzionamento è simile al trigger per il controllo delle schede alimentazione.

Analytics (Area Servizi)

```
CREATE TABLE IF NOT EXISTS IntervalliTempo (
Ora1 TIME ,
Ora2 TIME ,
PRIMARY KEY ( Ora1,Ora2))
```

```
Engine=InnoDB DEFAULT CHARSET=latin1;


INSERT INTO IntervalliTempo
VALUES('00:00:00','01:00:00'),
     ('01:00:00','02:00:00'),
     ('02:00:00','03:00:00'),
     ('03:00:00','04:00:00'),
     ('04:00:00','05:00:00'),
     ('05:00:00','06:00:00'),
     ('06:00:00','07:00:00'),
     ('07:00:00','08:00:00'),
     ('08:00:00','09:00:00'),
     ('09:00:00','10:00:00'),
     ('10:00:00','11:00:00'),
     ('11:00:00','12:00:00'),
     ('12:00:00','13:00:00'),
     ('13:00:00','14:00:00'),
     ('14:00:00','15:00:00'),
     ('15:00:00','16:00:00'),
     ('16:00:00','17:00:00'),
     ('17:00:00','18:00:00'),
     ('18:00:00','19:00:00'),
     ('19:00:00','20:00:00'),
     ('20:00:00','21:00:00'),
     ('21:00:00','22:00:00'),
     ('22:00:00','23:00:00'),
     ('23:00:00','24:00:00');
```

Descrizione: Prima di tutto si crea questa tabella, per avere tutti gli intervalli di tempo possibili per calcolare quanto sono frequentate le varie fasce orarie.

```
DROP VIEW IF EXISTS Intervalli;
CREATE VIEW Intervalli as
select  distinct OAC.Centro,I.Ora1,I.Ora2
from OrarioAperturaCentro OAC INNER JOIN IntervalliTempo I ON
    (I.Ora1 >= OAC.OrarioApertura
      AND
      I.Ora1<OAC.OrarioChiusura)
      AND
            (I.Ora2>OAC.OrarioApertura
      AND
      I.Ora2<=OAC.OrarioChiusura)
order by OAC.Centro,I.Ora1,I.Ora2;
```

Descrizione: Si crea una view per separare centro per centro con i suoi rispettivi orari di apertura e chiusura.

```
DROP VIEW IF EXISTS AccessiGiorni;
CREATE VIEW AccessiGiorni AS
SELECT TrovaGiorno(DataAccesso) AS Giorno,Sala,OrarioAccesso,OrarioUscita
FROM AccessoSala ;



DROP VIEW IF EXISTS Corso_Giorni;
```

```
CREATE VIEW Corso_Giorni AS
SELECT C.CodCorso, A.Giorno
    FROM AccessiGiorni A INNER JOIN Corso C
            ON A.Sala=C.Sala
            WHERE EXISTS (  SELECT *
                            FROM CalendarioLezioni CL
                            WHERE CL.GiornoSettimana=A.Giorno
                            AND A.OrarioAccesso>=CL.OrarioInizio
                            AND CL.CodCorso=C.CodCorso);
```

Descrizione: Si creano due view che seviranno per il calcolo dei frequentanti di ciascun corso.

```
DROP PROCEDURE IF EXISTS Calcolo_Offerte;
DELIMITER $$
CREATE PROCEDURE Calcolo_Offerte()
BEGIN

 DECLARE Clienti_PiscineStandard INT DEFAULT 0;
 DECLARE Clienti_PiscinePersonalizzato INT DEFAULT 0;
 DECLARE FrequenzaPiscine DOUBLE DEFAULT 0.00;
 DECLARE ClientiAree INT DEFAULT 0;
 DECLARE FrequenzaUtilizzoAreeAllestibili DOUBLE DEFAULT 0.00;
 DECLARE MediaFrequentatori DOUBLE DEFAULT 0.00;
 DECLARE ClientiTotali INT DEFAULT 0;



 SELECT COUNT(DISTINCT AP.Cliente) INTO Clienti_PiscineStandard   /*VENGONO CONSIDERATI ANCHE I
SENZA CONTRATTO*/
 FROM Accesso_Pagato AP LEFT JOIN  Contratto C
  ON C.Cliente=AP.Cliente
   INNER JOIN AbbonamentoStandard AB
    ON C.Tipologia=AB.NomeAbbonamento
     INNER JOIN Sala S
      ON S.CodSala=AP.Sala
 WHERE (AP.DataAccesso NOT BETWEEN C.DataSottoscrizione AND (C.DataSottoscrizione + INTERVAL
C.DuratainMesi MONTH)
    OR C.Cliente IS NULL)
    AND (AB.AccessoPiscine=0 OR AB.AccessoPiscine IS NULL)
    AND S.TipoSala='Piscina';


    SELECT COUNT(DISTINCT AP.Cliente) INTO Clienti_PiscinePersonalizzato
 FROM Accesso_Pagato AP INNER JOIN  Contratto C
  ON C.Cliente=AP.Cliente
   INNER JOIN AutorizzazionePersonalizzata AB
    ON C.CodContratto=AB.Contratto
     INNER JOIN Sala S
      ON S.CodSala=AP.Sala
 WHERE (AP.DataAccesso NOT BETWEEN C.DataSottoscrizione AND (C.DataSottoscrizione + INTERVAL
C.DuratainMesi MONTH)
    OR C.Cliente IS NULL)
    AND ((AB.AccessoPiscine=0 AND AB.Centro=S.Centro) OR AB.AccessoPiscine IS NULL)
    AND S.TipoSala='Piscina';
```

```
 SELECT COUNT(*) INTO ClientiTotali
 FROM Cliente;

 SET FrequenzaPiscine=((Clienti_PiscineStandard + Clienti_PiscinePersonalizzato)/ClientiTotali) *100;



  SELECT COUNT(DISTINCT SA.Cliente) INTO ClientiAree  /* Vengono considerati anche i senza contratto */
FROM SaldoAreeAllestibiliCliente SA INNER JOIN   Contratto C
  ON SA.Cliente=C.Cliente
WHERE SA.SaldoMese>0 OR SA.SaldoTotale>0;

SET FrequenzaUtilizzoAreeAllestibili=(ClientiAree/ClientiTotali) * 100;

INSERT INTO MV_Piscine_Aree
VALUES (CURRENT_TIMESTAMP(),FrequenzaPiscine,FrequenzaUtilizzoAreeAllestibili);



INSERT INTO MV_ResocontoCorsi
SELECT CURRENT_TIMESTAMP(),D.CodCorso,AVG(D.Frequentatori)
FROM( SELECT CodCorso,Giorno,COUNT(*) AS Frequentatori
     FROM  Corso_Giorni
     GROUP BY CodCorso,Giorno) AS D
GROUP BY D.CodCorso;



INSERT INTO MV_UtilizzoAttrezzatura(
SELECT CURRENT_TIMESTAMP(),D.Centro,D.Tipologia,D.CodAttrezzatura,D.NumeroUtilizzi
FROM ( SELECT S.Centro,A.Tipologia,A.CodAttrezzatura, COUNT(*) AS NumeroUtilizzi
      FROM Attrezzatura A INNER JOIN EsercizioSvolto_Configurazione ESC
      ON ESC.Attrezzatura=A.CodAttrezzatura
       INNER JOIN Sala S
        ON S.CodSala=A.Sala
                  GROUP BY S.Centro,A.CodAttrezzatura,A.Tipologia ) AS D
ORDER BY D.Centro,D.Tipologia,D.NumeroUtilizzi);



INSERT INTO MV_FasceOrarie
select  CURRENT_TIMESTAMP(),I.Centro,I.Ora1,I.Ora2,COUNT(*) as NumeroAccessiFasciaOraria
from Intervalli I INNER JOIN AccessoCentro AC
     ON
      (I.centro=AC.Centro
       AND
      ((AC.OrarioAccesso>=I.Ora1
       AND
       AC.OrarioAccesso<I.Ora2)
       OR
       (AC.OrarioUscita>I.Ora1
       AND
       AC.OrarioUscita<=I.Ora2)))
group by I.centro,I.Ora1,I.Ora2
```

```
order by I.centro,I.Ora1,I.Ora2;


 END $$
 DELIMITER ;
```

Descrizione: La procedure farà tutti i calcoli necessari per l'analytics del Reporting. Prima di tutto mi calcola tutti i clienti (con contratto e senza) che hanno usufruito di piscine e li metto in rapporto con i clienti totali per avere una percentuale dei clienti che in media usano le piscine. Lo stesso è stato fatto con le aree allestibili. Questo potrebbe essere utile all'azienda per valutare se conviene creare nuovi tipi di contratto o nuove possibilità di personalizzazione del contratto finalizzati all'accesso alle piscine o alle aree allestibili. Come terzo risultato ho la media dei frequentatori per ciascun corso, cosi da avere un'idea sui corsi da abolire o da continuare. Il quarto risultato ci rivela l'utilizzo delle attrezzature e quali quindi potrebbero essere i centri sovradimensionati. L'ultimo risultato è la frequentazione del centro per fasce orarie. Il risultato è molto importante, perchè controllando ciò l'azienda può decidere di redistribuire i turni del personale cosi da ammortizzare il carico di lavoro del personale che lavora nelle ore di picco della settimana. Tutti i risultati sono memorizzati nelle Materialised_View sottostanti.

```
CREATE TABLE IF NOT EXISTS MV_Piscine_Aree(
DataEsecuzione TIMESTAMP not null,
PercentualeUtilizzoPiscine DOUBLE not null,
PercentualeUtilizzoAree DOUBLE not null,
PRIMARY KEY (DataEsecuzione)
)ENGINE=InnoDB DEFAULT CHARSET=latin1;


CREATE TABLE IF NOT EXISTS MV_UtilizzoAttrezzatura (
DataEsecuzione TIMESTAMP not null,
Centro CHAR(20) not null,
CodAttrezzatura CHAR(20) not null,
TipologiaAttrezzatura CHAR(80) not null,
NumeroUtilizzi INT not null,
PRIMARY KEY(CodAttrezzatura)
)ENGINE=InnoDB DEFAULT CHARSET=latin1;


CREATE TABLE IF NOT EXISTS MV_ResocontoCorsi (
DataEsecuzione TIMESTAMP not null,
Corso CHAR(20) not null,
NumeroFrequentatori INT not null,
PRIMARY KEY(Corso)
)ENGINE=InnoDB DEFAULT CHARSET=latin1;


CREATE TABLE IF NOT EXISTS MV_FasceOrarie (
DataEsecuzione TIMESTAMP not null,
Centro CHAR(20) not null,
Da TIME not null,
A TIME not null,
NumeroAccessi INT not null,
PRIMARY KEY (Centro,Da,A)
```

```
)ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE IF NOT EXISTS Log_Sfida (
Sfida CHAR(20) not null,
PRIMARY KEY (Sfida)
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Allenamenti e monitoraggi smart

Tabelle

```
CREATE TABLE IF NOT EXISTS SchedaAllenamento(
CodSchedaAllenamento Char(15) not null,
Cliente char(16) not null,
DataEmissione date not null,
DataInizio date not null,
DataFine date not null,
Tutor char(16) not null,
PRIMARY KEY(CodSchedaAllenamento),
CONSTRAINT TutorRiferimento
FOREIGN KEY (Tutor)
REFERENCES Istruttore(CodFiscale)
ON DELETE NO ACTION
ON UPDATE CASCADE,
CONSTRAINT ClienteRiferimento
FOREIGN KEY (Cliente)
REFERENCES Cliente(CodFiscale)
ON UPDATE CASCADE
ON DELETE NO ACTION
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi : CodSchedaAllenamento,Cliente DataEmissione,DataInizio,DataFine,Tutor.
C**hiave primaria** : CodSchedaAllenamento .
Vincoli di integrità referenziale : TutorRiferimento,ClienteRiferimento.
Dipendenze funzionali : CodSchedaAllenamento → Cliente DataEmissione,DataInizio,DataFine,Tutor
Note: -

```
CREATE TABLE IF NOT EXISTS Esercizio(
CodEsercizio char(10) not null,
Nome char(50) not null,
TipoEsercizio char(15) not null,
DispendioEnergeticoMedio int,
DurataInMinuti int ,
NumeroRipetizioni int ,
NumeroSerie int ,
TempodirecuperoSecondi int not null,
PRIMARY KEY(CodEsercizio)
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi:
CodEsercizio,Nome,TipoEsercizio,DispendioEnergeticoMedio,DurataInMinuti,NumeroRipetizioni,Numero Serie,TempodirecuperoSecondi.
Chiave primaria: Codesercizio
Vincoli di integrità referenziale : -
Dipendenze funzionali: CodEsercizio → Nome,TipoEsercizio,DispendioEnergeticoMedio,DurataInMinuti,NumeroRipetizioni,NumeroSerie,Tempodi recuperoSecondi.

Note:
TipoEsercizio assume solo due valori possibili : 'Aerobico' o 'Anaerobico'. Se esso è 'Aerobico' allora gli attributi NumeroRipetizioni e NumeroSerie sono impostati a NULL (con una stored procedure, descritta in seguito) mentre DurataInMinuti assume un determinato valore intero. Viceversa se esso è 'Anaerobico' allora i primi verranno settati con un valore ben specifico mentre il terzo a NULL.

```
CREATE TABLE IF NOT EXISTS EsercizioScheda(
Scheda char(15) not null,
Esercizio char(10) not null,
Giorno int not null,
PRIMARY KEY(Scheda,Esercizio,Giorno),
CONSTRAINT SchedaRiferimentoEs
FOREIGN KEY(Scheda)
REFERENCES SchedaAllenamento(CodSchedaAllenamento)
ON UPDATE CASCADE
ON DELETE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi : Scheda, Esercizio,Giorno.
Chiave primaria : Scheda,Esercizio,Giorno.
Vincoli di integrità referenziale : SchedaRiferimentoEs.
Dipendenze funzionali : attributi tutti chiave
Note :
La tabella in sintesi funge da 'ponte' tra la scheda e gli esercizi assicurando che essi ne facciano effetivamente parte.

```
CREATE TABLE IF NOT EXISTS Attrezzatura(
CodAttrezzatura char(20) not null,
Tipologia char(80) not null,
LivelloDiUsuraPercentuale int not null,
Funzionante bool not null,
Sala char(20),
PRIMARY KEY (CodAttrezzatura),
CONSTRAINT SalaAttrezzatura
FOREIGN KEY (Sala)
REFERENCES Sala(CodSala)
ON DELETE SET NULL
ON UPDATE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi : CodAttrezzatura,Tipologia,LivelloDiUsuraPercentuale,Funzionante,Sala.
Chiave primaria: CodAttrezzatura.
Vincoli di integrità referenziale : SalaAttrezzatura.
Dipendenze funzionali : (CodAttrezzatura → Tipologia,LivelloDiUsuraPercentuale,Funzionante,Sala).
Note :

Tipologia indica ovviamente se l'attrezzo è ad esempio un manubrio, un tapis roulant o una panca….
Il LivelloDiUsuraPercentuale indica lo stato di usura dell'attrezzatura e assume valori tra 0 e 100, estremi compresi, con 100 che indica quando l'attrezzo è rotto e 0 che invece stabilisce che esso è perfettamente funzionante.
Funzionante è un booleano che assume valore TRUE quando l'attrezzo è utilizzabile (cioè con usura<100), FALSE in caso contrario e quindi è una ridondanza.

```
CREATE TABLE IF NOT EXISTS Esercizio_Attrezzatura (
Esercizio CHAR(50) not null,
Attrezzatura CHAR(20) not null,
PRIMARY KEY(Esercizio,Attrezzatura),
CONSTRAINT Attrezzatura_Esercizio
FOREIGN KEY (Attrezzatura)
REFERENCES Attrezzatura(CodAttrezzatura)
ON DELETE CASCADE
ON UPDATE NO ACTION
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi : Esercizio,Attrezzatura .
Chiave Primaria : Esercizio,Attrezzatura.
Vincoli di integrità referenziale : Attrezzatura_esercizio.
Dipendenze funzionali : attributi tutti chiave
Note : Come EsercizioScheda anche questa tabella funge da tramite tra altre due, nello specifico tra esercizio e attrezzatura.

```
CREATE TABLE IF NOT EXISTS Esercizio_Configurazione(
Esercizio CHAR(20) not null,
Attrezzatura CHAR(20) not  null,
TipoConfigurazione CHAR(50) not null,
ValoreConfigurazione INT  ,
PRIMARY KEY(Esercizio,Attrezzatura,TipoConfigurazione),
CONSTRAINT Esercizio_Scheda
FOREIGN KEY (Esercizio)
REFERENCES Esercizio(CodEsercizio)
ON DELETE NO ACTION
ON UPDATE NO ACTION
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi : Esercizio,Attrezzatura,TipoConfigurazione,ValoreConfigurazione.
Chiave primaria : Esercizio,Attrezzatura,TipoConfigurazione.
Vincoli di integrità referenziale : Esercizio_scheda.
Dipendenze funzionali : (Esercizio,Attrezzatura,TipoConfigurazione → ValoreConfigurazione)
Note :
Per TipoConfigurazione si intende attributi propri dell'attrezzatura come la velocità , l'inclinazione ...
ValoreConfigurazione è un valore che può essere settato a NULL in quanto non tutti i tipi di configurazioni sono anche regolabili o comunque hanno un valore che le contraddistingue .

```
CREATE TABLE IF NOT EXISTS EsercizioSvolto(
Cliente CHAR(16) not null,
Esercizio CHAR(10) not null,
IstanteInizio TIMESTAMP not null,
SchedaAllenamento CHAR(20),
```

```
Ripetizioni int ,
NumeroSerie int,
Durata INT ,
TempodiRecupero INT not null,
GiornoScheda INT default 0,
Sfida CHAR(20),
PRIMARY KEY (Cliente,IstanteInizio),
CONSTRAINT Esercizio_Svolto
FOREIGN KEY(Esercizio)
REFERENCES Esercizio(CodEsercizio)
ON DELETE CASCADE
ON UPDATE CASCADE,
CONSTRAINT Esercizio_Svolto_Scheda
FOREIGN KEY (SchedaAllenamento)
REFERENCES SchedaAllenamento(CodSchedaAllenamento)
ON DELETE CASCADE
ON UPDATE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi:Cliente,Esercizio,IstanteInizio,SchedaAllenamento,Ripetizioni,NumeroSerie,Durata,TempodiRecupero,GiornoScheda,Sfida
Chiave primaria : Cliente,Esercizio,IstanteInizio.
Vincoli di integrità referenziale : Esercizio_Svolto,Esercizio_Svolto_Scheda
Dipendenze funzionali : Cliente,IstanteInizio → SchedaAllenamento,Ripetizioni,NumeroSerie,Durata,TempodiRecupero,GiornoScheda,Sfida, Esercizio
Note :
Esercizio_Svolto è un po' l'alter ego fisico dell'astratto Esercizio . Per esso infatti vale lo stesso discorso di Esercizio per quanto riguarda 'Aerobico' o 'Anaerobico', ma essendo calato nella realtà a differenza della sua controparte non si può certo evitare controlli sull' effettivo accesso del cliente in quel dato orario in quella sala così come va guardato se la sala è dotata di quell 'attrezzatura e se essa è funzionante o non.

```
CREATE TABLE EsercizioSvolto_Configurazione(
Cliente CHAR(16) not null,
Esercizio CHAR(10) not null,
IstanteInizio TIMESTAMP not null,
Attrezzatura CHAR(50) not null,
TipoConfigurazione CHAR(80) ,
ValoreConfigurazione INT,
PRIMARY KEY(Cliente,IstanteInizio,Attrezzatura,TipoConfigurazione),
CONSTRAINT Eserciziosvolto_Target
FOREIGN KEY (Cliente,IstanteInizio)
REFERENCES EsercizioSvolto(Cliente,IstanteInizio)
ON DELETE CASCADE
ON UPDATE NO ACTION,
CONSTRAINT Esercizio_Attrezzatura
FOREIGN KEY (Attrezzatura)
REFERENCES Attrezzatura(CodAttrezzatura)
ON DELETE NO ACTION
ON UPDATE NO ACTION
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi : Cliente, Esercizio,IstanteInizio,Attrezzatura,TipoConfigurazione,ValoreConfigurazione.
Chiave primaria : Cliente,Esercizio,IstanteInizio,Attrezzatura,TipoConfigurazione.
Vincoli di integrità referenziale: EsercizioSvolto_Target,Esercizio_Attrezzatura.

Dipendenze funzionali :
(Cliente,Esercizio,IstanteInizioAttrezzatura,TipoConfigurazione→ValoreConfigurazione)
Note :
Stesso discorso per EsercizioSvolto con la differenza che come anticipato prima si necessita un controllo sull'effetiva disponibilità nelle sale dell'attrezzatura richiesta.

LogPerformance

```
CREATE TABLE LogPerformance(
DataAnalisi TIMESTAMP not null,
Cliente CHAR(16) not null,
EsercizioScheda char(50),
TempoRecupero  char (80) ,
SvolgimentoEsercizio char(80) ,
FedeltaAttrezzature char(80) ,
PRIMARY KEY(DataAnalisi,Cliente))
Engine=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: DataAnalisi,Cliente,EsercizioScheda,TempoRecupero,SvolgimentoEsercizio,FedeltaAttrezzature.
Chiave primaria : DataAnalisi,Cliente.
Vincoli di integrità referenziali : - (risparmiati perché è un log)
Dipendenze funzionali : (DataAnalisi,Cliente→
EsercizioScheda,TempoRecupero,SvolgimentoEsercizio,FedeltaAttrezzature)
Note :
In quanto log esso viene utilizzato per le funzionalità analytics , in particolare nell'analisi della performance sportiva che verrà illustrata meglio nella parte di questo documento dedicata alle operazioni .

Stored Procedures

L'implementazione di queste due stored procedure è volta a evitare fastidiosi inserimenti in cui dover utilizzare come parametri valori NULL e conseguenti TRIGGER che verifichino la compatibilità tra il tipo di esercizio , ovvero ' Aerobico ' e 'Anaerobico', e quali parametri siano stati settati a NULL o meno.
Molto più economico effettuare invece queste operazioni con funzionalità simili lato client che a seconda del tipo di esercizio assicurano la consistenza del dato inserito.
Le due stored operano in maniera speculare l'una all'altra : AggiungiEsercizioAerobico effettua l'inserimento in Esercizio settando Tipologia con la stringa 'Aerobico' e lasciando a NULL le ripetizioni e le serie mentre AggiungiEsercizioAnaerobico setta Tipologia con la stringa 'Anaerobico' e mette a NULL la durata.

AggiungiEsercizioAerobico

```
DROP PROCEDURE IF EXISTS AggiungiEsercizioAerobico;
```

```
DELIMITER $$
CREATE PROCEDURE AggiungiEsercizioAerobico ( IN _codesercizio CHAR(10) , IN _Nome CHAR(50),
IN _dispendioenergeticomedio INT, IN _Durata INT , IN _TempoRecupero INT )
BEGIN
  INSERT INTO Esercizio
  VALUES(_codesercizio,_nome,'aerobico',_dispendioenergeticomedio,_durata,NULL,NULL,_temporecupero);
END $$
DELIMITER ;
```

AggiungiEsercizioAnaerobico

```
DROP PROCEDURE IF EXISTS AggiungiEsercizioAnaerobico;
DELIMITER $$
CREATE PROCEDURE AggiungiEsercizioAnaerobico ( IN _codesercizio CHAR(10) , IN _Nome CHAR(50),
IN _dispendioenergeticomedio INT, IN _NumeroRipetizioni INT , IN _NumeroSerie INT, IN _TempoRecupero INT )
BEGIN
  INSERT INTO Esercizio

VALUES(_codesercizio,_nome,'anaerobico',_dispendioenergeticomedio,NULL,_NumeroRipetizioni,_NumeroSerie,_te
mporecupero);
END $$
DELIMITER ;
```

AggiungiEsercizio_Svolto e AggiungiConfigurazione_EsercizioSvolto

Le procedure inseriscono rispettivamente in EsercizioSvolto e in Eserciziosvolto_configurazione.
La prima controlla che il cliente abbia effettivamente effettuato l'accesso al centro e alla sala.
La seconda che la sala dell'attrezzatura coincida con quella dell'allenamento.

```
DROP PROCEDURE IF EXISTS AggiungiEsercizio_Svolto;
DELIMITER $$
CREATE PROCEDURE AggiungiEsercizio_Svolto(IN _Cliente VARCHAR(16), IN _Esercizio VARCHAR(20),IN
_Istante TIMESTAMP,
                              IN _SchedaAllenamento VARCHAR(20), IN _Ripetizioni INT ,IN _NumeroSerie INT,
                              IN _Durata INT, IN _TempodiRecupero INT,IN _GiornoScheda INT,
                              IN _Sfida VARCHAR(20))
BEGIN

  IF (NOT EXISTS (SELECT * FROM AccessoSala
              WHERE DataAccesso=DATE(_Istante)
               AND Cliente=_Cliente
               AND TIME(_Istante) BETWEEN OrarioAccesso AND OrarioUscita) AND
             NOT EXISTS (SELECT * FROM Log_Sala
              WHERE Cliente=_Cliente
               AND OrarioAccesso<=TIME(_Istante))) THEN

    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT='Errore, il cliente non ha eseguito accesso alla sala!';

  END IF;

  INSERT INTO EsercizioSvolto
  VALUES
(_Cliente,_Esercizio,_Istante,_SchedaAllenamento,_Ripetizioni,_NumeroSerie,_Durata,_TempodiRecupero,_GiornoSc
heda, _Sfida);
```

```
END $$
DELIMITER ;


DROP PROCEDURE IF EXISTS AggiungiConfigurazione_EsercizioSvolto;
DELIMITER $$
CREATE PROCEDURE AggiungiConfigurazione_EsercizioSvolto(IN _Cliente VARCHAR(16),
                                                        IN _Esercizio VARCHAR(20),
                                                        IN _Istante TIMESTAMP,
                                                        IN _Attrezzatura VARCHAR(20),
                                                        IN _TipoConfigurazione VARCHAR(80) ,
                                                        IN _Valore INT)
BEGIN

  SET @SalaAttrezzatura='';

SELECT Sala INTO @SalaAttrezzatura
FROM Attrezzatura
WHERE CodAttrezzatura=_Attrezzatura;

IF NOT EXISTS (SELECT * FROM EsercizioSvolto
            WHERE Cliente=_Cliente
             AND IstanteInizio=_Istante
             AND Esercizio=_Esercizio
             AND Sala=@SalaAttrezzatura) THEN
 BEGIN
           SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT='Errore di sistema, la sala non coincide';


 END ;
END IF;

INSERT INTO EsercizioSvolto_configurazione
VALUES (_Cliente,_Esercizio,_Istante,_Attrezzatura,_TipoConfigurazione,_Valore);

END $$
DELIMITER ;
```

EVENTS

ControllaAttrezzatura

Aggiorna lo stato di usura dell'attrezzatura aumentandone il livello di usura.
Se l'usura arriva a 100 l'attrezzo non è più funzionante.

```
DROP EVENT IF EXISTS ControllaAttrezzatura;
DELIMITER $$
CREATE EVENT ControllaAttrezzatura
ON SCHEDULE EVERY 2 MONTH
STARTS '2017-11-17 00:00:00'
DO
 BEGIN
```

```
        UPDATE Attrezzatura
    SET LivelloDiUsuraPercentuale=LivelloDiUsuraPercentuale+10;

    UPDATE Attrezzatura
    SET Funzionante=0
    WHERE LivelloDiUsuraPercentuale>=100;

 END $$
DELIMITER ;
```

Analisi della performance sportiva(Area Servizi)

```
DROP PROCEDURE IF EXISTS AnalisiPerformanceEsercizioAerobico ;
DELIMITER $$
CREATE PROCEDURE AnalisiPerformanceEsercizioAerobico (IN _cliente CHAR(16), IN _esercizio CHAR(10),IN
_IstanteInizio TIMESTAMP,IN _schedaallenamento CHAR(20), IN _Durata INT , IN _Recupero INT ,IN
_GiornoScheda INT)
BEGIN
    declare _finito INT default 0;
    declare _durataOtt INT default 0;
    declare _recuperoOtt INT default 0;
    declare _attrezzatura CHAR(50) default ' ';
    declare _configurazioneExSvolto CHAR(80) default ' ';
    declare _valoreconfigurazione INT default 0;
    declare _valutazioneconfigurazione BOOL default TRUE;
    declare _valutazionetempi DOUBLE default 0.00;
    declare _valutazionerecupero DOUBLE default 0.00;
    declare _commentoconfigurazione CHAR(50) default ' ';
    declare _commentotempi CHAR(50) default ' ';
    declare _commentorecupero CHAR(50) default ' ';


/* configurazione */
    declare ControllaConfigurazione cursor for
    (SELECT Attrezzatura,TipoConfigurazione,ValoreConfigurazione
     FROM esercizio_configurazione
     WHERE esercizio=_esercizio);

          declare continue handler for not found set _finito=1;

    IF NOT EXISTS (SELECT*
                        FROM esercizioscheda
                        WHERE esercizio=_esercizio
                        AND
                        giorno=_giornoscheda
                        AND
                        scheda=_schedaallenamento) THEN
     BEGIN
       INSERT INTO logperformance
       VALUES(current_timestamp,_cliente,"Esercizio non compatibile con la scheda" ,NULL,NULL,NULL);
       SIGNAL SQLSTATE '45000'
       SET MESSAGE_TEXT="Attenzione,inserito esercizio non valutabile";
                END;
          END IF;

 IF EXISTS (SELECT *
              FROM esercizio_configurazione
             WHERE
```

```
            esercizio=_esercizio)          THEN

   open ControllaConfigurazione;


   Controlla : LOOP
  BEGIN
    FETCH ControllaConfigurazione INTO _attrezzatura,_configurazioneexsvolto,_valoreconfigurazione;
    IF _finito=1 THEN
     LEAVE Controlla;
          END IF;
    IF NOT EXISTS (SELECT *
                        FROM eserciziosvolto_configurazione
                        WHERE esercizio=_esercizio
                        AND
                        attrezzatura=_attrezzatura
                        AND
                        cliente=_cliente
                        AND
                        istanteinizio=_istanteinizio
                        AND
                        tipoconfigurazione=_configurazioneexsvolto
                          AND
                       valoreconfigurazione=_valoreconfigurazione) THEN
          BEGIN
            set _valutazioneconfigurazione=FALSE;
             LEAVE Controlla;
           END ;
     END IF;
   END;
          END LOOP;

  close ControllaConfigurazione;

  IF (_valutazioneconfigurazione is TRUE) THEN
   set _commentoconfigurazione="Configurazione con attrezzi corretta";
  END IF;
  IF _valutazioneconfigurazione is FALSE THEN
   set _commentoconfigurazione="Configurazione con attrezzi errata";
  END IF;

ELSE
 set _commentoconfigurazione="Esercizio senza attrezzi";
END IF;



 /* durata */

  set _durataOtt=(select DurataInMinuti
            from esercizio
            where codesercizio=_esercizio);

   CASE

  WHEN _durata BETWEEN (0.85*_durataOtt) AND (1.15*_durataOtt) THEN
   set _commentotempi="Ok,tempo relativamente giusto";
  WHEN _durata<(0.85*_durataOtt) THEN
   set _commentotempi="Attenzione,tempo inferiore all'ottimale";
```

```
   WHEN _durata>(1.15*_durataOtt) THEN
    set _commentotempi="Ottimo,tempo superiore all'ottimale";

   END CASE ;

/* recupero */

  set _recuperoOtt=(select TempodirecuperoSecondi
                from esercizio
                where codesercizio=_esercizio);

  CASE

   WHEN _recupero BETWEEN (0.85*_recuperoOtt) AND (1.15*_recuperoOtt) THEN
    set _commentorecupero="Ok,tempo relativamente giusto";
   WHEN _recupero<(0.85*_recuperoOtt) THEN
    set _commentorecupero="Con calma,tempo inferiore all'ottimale";
   WHEN _recupero>(1.15*_recuperoOtt) THEN
    set _commentorecupero="Attenzione,tempo superiore all'ottimale";

   END CASE ;


   INSERT INTO LogPerformance
   VALUES (_istanteinizio,_cliente,_esercizio,_commentorecupero,_commentotempi,_commentoconfigurazione);


END $$
DELIMITER ;
```

AnalisiPerformanceEsercizioAnaerobico

```
DROP PROCEDURE IF EXISTS AnalisiPerformanceEsercizioAnaerobico ;
DELIMITER $$
CREATE PROCEDURE AnalisiPerformanceEsercizioAnaerobico (IN _cliente CHAR(16), IN _esercizio CHAR(10),
IN _IstanteInizio TIMESTAMP,IN _schedaallenamento CHAR(20), IN _ripetizioni INT ,IN _serie INT,
 IN _Recupero INT ,IN _GiornoScheda INT)
BEGIN
    declare _finito INT default 0;
    declare _ripetizioniOtt INT default 0;
    declare _serieOtt INT default 0;
    declare _recuperoOtt INT default 0;
    declare _attrezzatura CHAR(50) default ' ';
    declare _configurazioneExSvolto CHAR(80) default ' ';
    declare _valoreconfigurazione INT default 0;
    declare _valutazioneconfigurazione BOOL default TRUE;
    declare _valutazionerecupero DOUBLE default 0.00;
    declare _commentoconfigurazione CHAR(80) default ' ';
    declare _commentoserie char(80) default 0;
    declare _commentorecupero CHAR(80) default ' ';
    declare _ripetizionitot INT default 0;
    declare _ripetizioniOttTOT int default 0;
/* configurazione */
```

```
 declare ControllaConfigurazione cursor for
(select Attrezzatura,TipoConfigurazione,ValoreConfigurazione
        from esercizio_configurazione
 where esercizio=_esercizio);

        declare continue handler for not found set _finito=1;

IF NOT EXISTS (select *
            from esercizioscheda
            where esercizio=_esercizio
                AND
                giorno=_giornoscheda
                AND
                scheda=_schedaallenamento) THEN
                  BEGIN
    insert into logperformance
    VALUES(_IstanteInizio,_cliente,"Esercizio non compatibile con la scheda" ,NULL,NULL,NULL);
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT="Attenzione,inserito esercizio non valutabile";
                END;
      END IF;

IF EXISTS (select * from esercizio_configurazione
              where esercizio=_esercizio)        THEN

open ControllaConfigurazione;


Controlla : LOOP
        BEGIN
 FETCH ControllaConfigurazione INTO _attrezzatura,_configurazioneexsvolto,_valoreconfigurazione;
 IF _finito=1 THEN
  LEAVE Controlla;
       END IF;
 IF NOT EXISTS (select *
        from eserciziosvolto_configurazione
        where esercizio=_esercizio
            AND
            attrezzatura=_attrezzatura
            AND
            cliente=_cliente
            AND
            istanteinizio=_istanteinizio
            AND
            tipoconfigurazione=_configurazioneexsvolto
            AND
            valoreconfigurazione=_valoreconfigurazione) THEN
        BEGIN
                        set _valutazioneconfigurazione=FALSE;
    LEAVE Controlla;
        END ;
            END IF;
END;
        END LOOP;

close ControllaConfigurazione;

IF (_valutazioneconfigurazione is TRUE) THEN
 set _commentoconfigurazione="Configurazione con attrezzi corretta";
```

```
  END IF;
  IF _valutazioneconfigurazione is FALSE THEN
    set _commentoconfigurazione="Configurazione con attrezzi errata";
  END IF;

ELSE
  set _commentoconfigurazione="Esercizio senza attrezzi";
END IF;


 /* serie */

  set _serieOtt=(select numeroserie
                 from esercizio
                 where codesercizio=_esercizio);

   set _ripetizioniOtt =(select numeroripetizioni
                         from esercizio
                         where codesercizio=_esercizio);

  set _ripetizionitot=_ripetizioni*_serie;
  set _ripetizioniOtttot=_serieOtt*_ripetizioniOtt;

  CASE

  WHEN _ripetizionitot BETWEEN (0.85*_ripetizioniOttTot) AND (1.15*_ripetizioniOttTot) THEN
    set _commentoserie="Ok,numero ripetizioni adeguato";
  WHEN _ripetizionitot<(0.85*_ripetizioniOttTot) THEN
    set _commentoserie="Attenzione,numero ripetizioni inferiore al richiesto";
  WHEN _ripetizionitot>(1.15*_ripetizioniOttTot) THEN
    set _commentoserie="Ottimo,numero ripetizioni superiore al richiesto";

  END CASE ;


/* recupero */

  set _recuperoOtt=(select TempodirecuperoSecondi
                    from esercizio
                    where codesercizio=_esercizio);

  CASE

  WHEN _recupero BETWEEN (0.85*_recuperoOtt) AND (1.15*_recuperoOtt) THEN
    set _commentorecupero="Ok,tempo relativamente giusto";
  WHEN _recupero<(0.85*_recuperoOtt) THEN
    set _commentorecupero="Con calma,tempo inferiore all'ottimale";
  WHEN _recupero>(1.15*_recuperoOtt) THEN
    set _commentorecupero="Attenzione,tempo superiore all'ottimale";

  END CASE ;


  INSERT INTO LogPerformance
   VALUES (_istanteinizio,_cliente,_esercizio,_commentorecupero,_commentoserie,_commentoconfigurazione);

 END $$
DELIMITER ;
```

Aree allestibili

Tabelle

```
CREATE TABLE IF NOT EXISTS AreaAllestibile(
CodArea char(20) not null,
Sede char(20) not null,
Locazione char(10) CHECK (Locazione='interno' or Locazione = 'esterno'),
MaxNumeroPersone int not null,
MinNumeroPersone int not null,
PRIMARY KEY(CodArea),
CONSTRAINT FK_Sede
FOREIGN KEY (Sede)
REFERENCES Centro(CodCentro)
ON DELETE CASCADE
ON UPDATE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: CodArea,Sede,Locazione,MaxNumeroPersone,MinNumeroPersone.
Chiave primaria :CodArea.
Vincoli di integrità referenziali: FK_S
Dipendenze funzionali: CodArea → Sede,Locazione,MaxNumeroPersone,MinNumeroPersone
Note: -

```
CREATE TABLE IF NOT EXISTS TariffeAreeAllestibili(
TipoArea char(30) not null,
CodArea char(20) not null,
TariffaOrariaPerPersona double not null,
TariffaAttrezzattura double default 0.00,
PRIMARY KEY (TipoArea,CodArea),
CONSTRAINT AreaRif
FOREIGN KEY (CodArea)
REFERENCES AreaAllestibile(CodArea)
ON DELETE CASCADE
ON UPDATE CASCADE
)Engine=InnoDB DEFAULT CHARSET=latin1;
```

Attributi:TipoArea,CodArea,TariffaOrariaPerPersona,TariffaAttrezzatura
Chiave primaria : TipoArea,CodArea
Vincoli di integrità referenziali : AreaRif
Dipendenze funzionali : TipoArea,CodArea→ TariffaOrariaPerPersona,TariffaAttrezzatura
Note :
TariffaAttrezzatura è stata inserita per quelle attività, come ad esempio il softair ,in cui sono richieste armi e un vestiario adatto a questi tipo di giochi per il quale si paga un costo aggiuntivo in caso si voglia utilizzare del materiale prontamente fornito dalla palestra nell'eventualità i partecipanti non ne dispongano.
L'importanza di questa tabella non è solo sul piano tariffario , ma serve anche da tramite tra un area e la corrispondente attività, cioè essa ne verifica la **compatibilità**. La mancata esistenza di un record in cui è presente un determinato CodArea e una specifica tipologia di area significa infatti che quell'area non può essere adibita a quell'attività.

```
CREATE TABLE IF NOT EXISTS Prenotazione(
CodPrenotazione char(30) not null,
Area char(20) not null,
TipoArea char(30) not null,
AttrezzaturaParticolareRichiesta bool,
ClienteRichiedente char(50) not null,
DataInvioPrenotazione timestamp not null,
DataAttivita date not null,
InizioAttivita time not null,
FineAttivita time not null,
Stato char(30) not null,
PunteggioGruppo int not null,
NumPartecipanti int not null,
PRIMARY KEY (CodPrenotazione),
CONSTRAINT Prenota_Cliente
FOREIGN KEY (ClienteRichiedente)
REFERENCES Cliente(CodFiscale)
ON DELETE CASCADE
ON UPDATE CASCADE,
CONSTRAINT AreaRiferimento
FOREIGN KEY (TipoArea,Area)
REFERENCES AreaAllestibile(TipoArea,CodArea)
ON DELETE CASCADE
ON UPDATE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi:Codprenotazione,Area,TipoArea,AttrezzaturaParticolareRichiesta,ClienteRichiedente,DataInvioPrenotazione,DataAttivita,InizioAttivita,FineAttivita,Stato,PunteggioGruppo,NumPartecipanti.
Chiave primaria: Codprenotazione
Vincoli di integrità referenziali : Prenota_Cliente,gAreaRiferimento.
Dipendenze funzionali : CodPrenotazione → Area,TipoArea,AttrezzaturaParticolareRichiesta,ClienteRichiedente,DataInvioPrenotazione,DataAttivita,InizioAttivita,FineAttivita,Stato,PunteggioGruppo,NumPartecipanti.
Ridondanze : Numpartecipanti,PunteggioGruppo
Note :
AttrezzaturaParticolareRichiesta è il parametro booleano il cui valore è a TRUE se si vuole disporre delle attrezzature della palestra per una determinata attività, mentre a FALSE indica che i partecipanti dispongono già del necessario.
PunteggioGruppo è un attributo fondamentale per stabilire l'assegnazione delle aree allestibili nelle prenotazioni , la politica seguita verrà meglio illustrata nelle parte delle operazioni , ma può essere sintetizzata dicendo che essa si basa sul numero dei partecipanti e sui contratti da essi posseduti (5 punti i senza contratto,10 i silver , 15 i gold , 20 i personalizzati , 25 i platinum).
Stato può assumere ben 4 valori :
-"gruppo in composizione" , è quando la procedura di prenotazione non è stata ancora inviata e in cui si possono aggiungere dei partecipanti .
-"da confermare" si intende che la prenotazione è stata inviata e la direzione sceglierà poi se accettarla o meno.In questo stato non si possono più aggiungere partecipanti né toglierli.
-" alternativa" indica che la richiesta non è stata accettata per cui sono state fatte tre proposte alternative . Lo stato passa a "approvata" se il richiedente approva via web una delle tre alternative.
-"approvata" ovviamente indica che la richiesta è andata a buon fine (o per scelta immediata della direzione o perché è stata successivamente accettata dal cliente come alternativa).

```
CREATE TABLE IF NOT EXISTS Partecipante(
Cliente char(16) not null,
CodPrenotazione char(30) not null,
PRIMARY KEY(CodPrenotazione,Cliente),
CONSTRAINT Partecipante_cliente
FOREIGN KEY (Cliente)
REFERENCES Cliente(CodFiscale)
ON DELETE CASCADE
ON UPDATE CASCADE,
CONSTRAINT Partecipante_Prenotazione
FOREIGN KEY (CodPrenotazione)
REFERENCES Prenotazione(CodPrenotazione)
ON DELETE CASCADE
ON UPDATE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi:Cliente,CodPrenotazione.
Chiave primaria:Cliente,CodPrenotazione.
Vincoli di integrità referenziali : Partecipante_Cliente,Partecipante_Prenotazione
Dipendenze funzionali : attributi tutti chiave.
Note: -

```
CREATE TABLE IF NOT EXISTS PrenotazioneAlternativa (
CodPrenotazione char(30) not null,
DataAlternativa date not null,
OrarioInizioAlternativo time not null,
OrarioFineAlternativo time not null,
DataInoltroPrenotazione timestamp not null,
PRIMARY KEY(codprenotazione,dataalternativa,orarioinizioalternativo,
                orariofinealternativo),
CONSTRAINT PrenotazioneDiRiferimento
FOREIGN KEY (CodPrenotazione)
REFERENCES Prenotazione(CodPrenotazione)
ON DELETE CASCADE
ON UPDATE CASCADE
)Engine=InnoDB DEFAULT CHARSET=latin1;
```

Attributi : Codprenotazione,dataalternativa,orarioinizioalternativo,orariofinealternativo.
Chiave primaria : Codprenotazione,dataalternativa,orarioinizioalternativo,orariofinealternativo.
Vincoli di integrità referenziale : PrenotazioneDiRiferimento
Dipendenze funzionali :attributi tutti chiave
Note :
Le proposte alternative ad una prenotazione sono tre per scelta della direzione. Esse rimangono attive per 48 ore ,e, una volta passate , esse vengono eliminate insieme alla prenotazione di riferimento per mezzo di un EVENT se entro questo gap temporale il cliente non ha espresso ancora il proprio giudizio.L'eliminazione avviene semplicemente attraverso una delete sul record in Prenotazione visto che poi essa si propaga (CASCADE).
La proposta se accettata dal cliente funge nel seguente modo : la prenotazione di riferimento (quella in Prenotazione appunto) passa da stato "alternativa" a stato " approvata" e i valori relativi alla data e all'ora dell'attività assumono gli stessi della prenotazione alternativa che verrà poi cancellata insieme alle altre due.Tutto ciò avviene con una Stored Procedure.

```
CREATE TABLE IF NOT EXISTS SaldoAreeAllestibiliCliente(
Cliente char(16) not null,
SaldoMese double default 0,
SaldoTotale double default 0 ,
Centro char(20) not null,
PRIMARY KEY(Cliente,Centro),
CONSTRAINT ClientePagamento
FOREIGN KEY (Cliente)
REFERENCES Cliente(CodFiscale)
ON DELETE NO ACTION
ON UPDATE CASCADE,
CONSTRAINT REF_Sede
FOREIGN KEY (Centro)
REFERENCES Centro(CodCentro)
)Engine=InnoDB DEFAULT CHARSET=latin1;
```

Attributi :Cliente,SaldoMese,SaldoTotale,Centro.
Chiave primaria: Cliente,Centro.
Vincoli di integrità referenziale :ClientePagamento,REF_Sede.
Dipendenze funzionali : Cliente,Centro → SaldoMese,SaldoTotale
Ridondanze : SaldoMese,SaldoTotale (è la tabella stessa "ridondante")
Note :
La tabella in questione rappresenta il profilo del cliente riguardo questa particolare funzionalità del centro messa a disposizione del cliente , che è appunto quella delle "Aree Allestibili".
Il profilo tiene conto di quanto il cliente ha effettivamente utilizzato quest'opzione tenendo conto del suo saldo mese e del suo saldo totale(una ridondanza magari utile per future promozioni …) che vengono aggiornati automaticamente mediante le varie stored procedure che effettuano le principali operazioni.

```
CREATE TABLE IF NOT EXISTS SaldiAreeAllestibiliDaPagare(
Cliente char(16) not null,
Saldo double not null,
Mese char(15) not null,
Anno int not null,
StatoSaldo char(15) not null,
Centro char(20) not null,
PRIMARY KEY(Cliente,Mese,Anno,Centro),
CONSTRAINT SedeSaldo
FOREIGN KEY (Centro)
REFERENCES Centro(CodCentro)
ON DELETE NO ACTION
ON UPDATE CASCADE,
CONSTRAINT ClienteArea
FOREIGN KEY (Cliente)
REFERENCES Cliente(CodFiscale)
ON DELETE NO ACTION
ON UPDATE CASCADE
) Engine=InnoDB DEFAULT CHARSET=latin1;
```

Attributi : Cliente,Saldo,Mese,Anno,StatoSaldo,Centro.
Chiave primaria Cliente,Mese,Anno,Centro
Vincoli di integrità referenziale : SedeSaldo,ClienteArea.
Dipendenze funzionali : Cliente,Mese,Anno,Centro→Saldo,StatoSaldo
Note:

Il pagamento dell'area allestibile non viene effettuato singolarmente ma è cumulativo , cioè si paga a fine mese per tutte le volte che in quel mese si è utilizzata l'area.
Per questo la chiave primaria è cliente,mese,anno,centro perché in un solo mese di quell'anno per quell'opportuno centro si avrà uno e uno solo pagamento.

Stored Procedure

Iscrizione CentroAreeAllestibili

Per sfruttare le aree allestibili il cliente deve prima crearsi un profilo dove essere costantemente aggiornato sulle proprie spese relative a quest'area.
La procedure consiste in un controllo e un insert in SaldoAreeAllestibiliCliente.

```
DROP PROCEDURE IF EXISTS IscrizioneCentroAreeAllestibili;
DELIMITER $$
CREATE PROCEDURE IscrizioneCentroAreeAllestibili(IN _Cliente CHAR(16),IN _Centro CHAR(20))
BEGIN
  IF EXISTS (select *
          from SaldoAreeAllestibiliCliente
          where Cliente=_Cliente
              AND
              Centro=_Centro) THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT="Hai già sincronizzato l' account con questo centro";
  END IF;
  INSERT INTO SaldoAreeAllestibiliCliente
  VALUES(_Cliente,0,0,_Centro);
END $$
DELIMITER ;
```

PrenotazioneAreaAllestibile

Da qui incominciano le vere e proprie operazioni.
La prenotazione viene implementata in primis con due controlli fondamentali :
-compatibilità dell'area e dell'attività
-disponibilità dell'area per quella fascia oraria.
A seguire c'è un CASE statement che assegna un PunteggioGruppo a seconda del tipo di contratto posseduto dal cliente richiedente.
A questo punteggio si andrà poi a sommare quello dei partecipanti (il come verrà mostrato in seguito).

```
DROP PROCEDURE IF EXISTS PrenotazioneAreaAllestibile;
DELIMITER $$
CREATE PROCEDURE PrenotazioneAreaAllestibile (IN _CodPrenotazione CHAR(30) , IN _Area CHAR(20),
IN _TipoArea CHAR(30), IN _AttrezzaturaRichiesta BOOL, IN _Cliente CHAR(50), IN _DataAttivita DATE,
IN _InizioAttivita TIME , IN _FineAttivita TIME )
BEGIN
  declare _tipologiacontratto CHAR(30) default ' ';
  declare _sede              CHAR(20) default ' ';

  set _sede = (select sede
            from areaallestibile
            where codarea=_area);
```

```
  IF NOT EXISTS  (select *
                from tariffeareeallestibili
                where codarea=_area
                        AND
                       tipoarea=_tipoarea) THEN
              SIGNAL SQLSTATE '45000'
     SET MESSAGE_TEXT="Attività non compatibile con l'area richiesta";
  END IF;

/* compatibilità area */

  IF (ControlloDisponibilitàArea(_Area,_DataAttivita,_InizioAttivita,_FineAttivita) = "occupata") THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT="L'area è già occupata in quella fascia oraria";
  END IF;

/* disponibilità area */

  set _tipologiacontratto=TipologiaContrattoCliente (_Cliente,_sede);

  CASE

    WHEN _tipologiacontratto='Silver' THEN
    INSERT INTO prenotazione
    values (_Codprenotazione,_Area,_TipoArea,_AttrezzaturaRichiesta,_Cliente,current_timestamp,
        _DataAttivita,_InizioAttivita,_FineAttivita,"gruppo in composizione",10,1);

            WHEN _tipologiacontratto='Gold' THEN
    INSERT INTO prenotazione
    values (_Codprenotazione,_Area,_TipoArea,_AttrezzaturaRichiesta,_Cliente,current_timestamp,
        _DataAttivita,_InizioAttivita,_FineAttivita,"gruppo in composizione",15,1);

    WHEN _tipologiacontratto='Platinum' THEN
    INSERT INTO prenotazione
    values (_Codprenotazione,_Area,_TipoArea,_AttrezzaturaRichiesta,_Cliente,current_timestamp,
        _DataAttivita,_InizioAttivita,_FineAttivita,"gruppo in composizione",25,1);

     WHEN _tipologiacontratto='Personalizzato' THEN
    INSERT INTO prenotazione
    values (_Codprenotazione,_Area,_TipoArea,_AttrezzaturaRichiesta,_Cliente,current_timestamp,
        _DataAttivita,_InizioAttivita,_FineAttivita,"gruppo in composizione",20,1);

     WHEN _tipologiacontratto='senza contratto' THEN
     INSERT INTO prenotazione
    values (_Codprenotazione,_Area,_TipoArea,_AttrezzaturaRichiesta,_Cliente,current_timestamp,
        _DataAttivita,_InizioAttivita,_FineAttivita,"gruppo in composizione",5,1);

  END CASE;

END $$
DELIMITER ;
```

AggiungiPartecipante

L’implementazione presenta dapprima tre filtri , ovvero l’operazione fallise se :
-la prenotazione è già stata inoltrata

-chi prenota è il richiedente
-il numero di partecipanti è già quello massimo per quell'area.
Di seguito, viene inserito il partecipante, incrementato il numero dei partecipanti e il punteggio gruppo sempre a seconda del tipo di contratto.

```
DROP PROCEDURE IF EXISTS AggiungiPartecipante;
DELIMITER $$
CREATE PROCEDURE AggiungiPartecipante(IN _Cliente CHAR(30),IN  _CodPrenotazione CHAR(30))
BEGIN
 declare _tipologiacontratto CHAR(30) default ' ';
 declare _clienterichiedente CHAR(30) default ' ';
 declare _sede CHAR(20) default ' ';
 declare _stato CHAR(20) default ' ';
 declare _maxpartecipanti INT default 0;
 declare _numtotpartecipanti INT default 0;

 SELECT A.maxnumeropersone,A.Sede,P.NumPartecipanti,P.Stato,P.ClienteRichiedente
 INTO _maxpartecipanti,_sede,_numtotpartecipanti,_stato,_clienterichiedente
 FROM prenotazione P inner join areaallestibile A on P.area=A.codarea
 WHERE P.codprenotazione=_codprenotazione;


 IF (_stato= "da confermare") THEN

    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT =" La prenotazione è già stata inoltrata";
 END IF;


 IF  _clienterichiedente=_cliente THEN

      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT="Chi prenota è già un partecipante";
END IF;

IF _numtotpartecipanti+1>_maxpartecipanti THEN
  SIGNAL SQLSTATE '45000'
  SET MESSAGE_TEXT="L'area non può ospitare più partecipanti";
END IF;

INSERT INTO Partecipante
VALUES (_Cliente,_CodPrenotazione);

UPDATE Prenotazione
SET NumPartecipanti=NumPartecipanti+1
WHERE codprenotazione=_codprenotazione;

SET _tipologiacontratto=TipologiaContrattoCliente(_Cliente,_sede);

CASE

   WHEN _tipologiacontratto='silver' THEN

   UPDATE prenotazione
   SET PunteggioGruppo=PunteggioGruppo+10
   WHERE CodPrenotazione=_codprenotazione;
```

```
        WHEN _tipologiacontratto='gold' THEN

    UPDATE prenotazione
    SET PunteggioGruppo=PunteggioGruppo+15
    WHERE CodPrenotazione=_codprenotazione;


    WHEN _tipologiacontratto='platinum' THEN

        UPDATE prenotazione
    SET PunteggioGruppo=PunteggioGruppo+25
    WHERE CodPrenotazione=_codprenotazione;

     WHEN _tipologiacontratto='personalizzato' THEN

        UPDATE prenotazione
    SET PunteggioGruppo=PunteggioGruppo+20
    WHERE CodPrenotazione=_codprenotazione;



        WHEN _tipologiacontratto='senza contratto' THEN

        UPDATE prenotazione
        SET PunteggioGruppo=PunteggioGruppo+5
        WHERE CodPrenotazione=_codprenotazione;

    END CASE;

END $$
DELIMITER ;
```

DisdiciPartecipazione

Speculare alla precedente, effettua l'azione contraria. Da notare che il richiedente non può disiscriversi e che una volta inviata la richiesta di prenotazione non ci si può più disiscrivere (così come iscriversi).

```
DROP PROCEDURE IF EXISTS DisdiciPartecipazione;
DELIMITER $$
CREATE PROCEDURE DisdiciPartecipazione (IN _Cliente CHAR(30),IN  _CodPrenotazione CHAR(30))
BEGIN
  declare _tipologiacontratto CHAR(30) default ' ';
  declare _sede CHAR(20) default ' ';

  IF (SELECT Stato
      FROM Prenotazione
      WHERE CodPrenotazione=_codprenotazione) = "da confermare" THEN

      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT =" La prenotazione è già stata inoltrata";
  END IF;

  IF _Cliente = (SELECT clienterichiedente
                 FROM prenotazione
                 WHERE codprenotazione=_codprenotazione) THEN
                   SIGNAL SQLSTATE '45000'
       SET MESSAGE_TEXT="Il richiedente non può disdire";
```

```
END IF;

IF NOT EXISTS ( SELECT *
                    FROM Partecipante
                    WHERE codprenotazione=_codprenotazione
                    AND cliente=_cliente ) THEN


      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = "Cancellazione fallita, non sei iscritto";
END IF;

DELETE  FROM Partecipante
WHERE Cliente=_cliente
 AND codprenotazione=_codprenotazione;

UPDATE Prenotazione
SET NumPartecipanti=NumPartecipanti-1
WHERE codprenotazione=_codprenotazione;

SET _tipologiacontratto=TipologiaContrattoCliente(_Cliente,_sede);

CASE

  WHEN _tipologiacontratto='silver' THEN

   UPDATE prenotazione
   SET PunteggioGruppo=PunteggioGruppo-10
   WHERE CodPrenotazione=_codprenotazione;

 WHEN _tipologiacontratto='gold' THEN

   UPDATE prenotazione
   SET PunteggioGruppo=PunteggioGruppo-15
   WHERE CodPrenotazione=_codprenotazione;


 WHEN _tipologiacontratto='platinum' THEN

   UPDATE prenotazione
   SET PunteggioGruppo=PunteggioGruppo-25
   WHERE CodPrenotazione=_codprenotazione;

 WHEN _tipologiacontratto='personalizzato' THEN

   UPDATE prenotazione
   SET PunteggioGruppo=PunteggioGruppo-20
   WHERE CodPrenotazione=_codprenotazione;

  WHEN _tipologiacontratto='senza contratto' THEN

   UPDATE prenotazione
   SET PunteggioGruppo=PunteggioGruppo-5
   WHERE CodPrenotazione=_codprenotazione;

   END CASE;

END $$
DELIMITER ;
```

ConcludiPrenotazione

ConcludiPrenotazione sancisce l'inoltro effettivo della procedura di prenotazione, che era già presente in Prenotazione ma con stato "Gruppo in formazione" (quindi come fosse silente) mentre ora viene resa nota mettendo stato a "Da confermare". Se non si è raggiunto un minimo di partecipanti la richiesta non può essere inoltrata.

```
DROP PROCEDURE IF EXISTS ConcludiPrenotazione;
DELIMITER $$
CREATE PROCEDURE ConcludiPrenotazione (IN _Codprenotazione CHAR(30))
BEGIN
  declare _numpartecipanti int default 0;
  declare _minpartecipanti int default 0;

   SELECT P.numpartecipanti,A.MinNumeroPersone INTO  _numpartecipanti,_minpartecipanti
   FROM Prenotazione P inner join AreaAllestibile A on P.area = A.codarea
   WHERE P.codprenotazione=_codprenotazione;

  IF _numpartecipanti >= _minpartecipanti THEN

  UPDATE Prenotazione
  SET Stato="Da confermare"
  WHERE codprenotazione=_codprenotazione;

ELSE

  SIGNAL SQLSTATE '45000'
  SET MESSAGE_TEXT="Numero persone insufficienti per usufruire dell'area";

  END IF;

END $$
DELIMITER ;
```

AccettaRifiutaPrenotazione

Come abbiamo visto in precedenza le varie operazioni hanno degli effetti collaterali , cioè oltre a svolgere le loro mansioni incrementano o decrementano il punteggio gruppo.
Questa procedure rappresenta il criterio di scelta di chi servire e chi no. Viene servito prima chi ha il punteggio gruppo più alto di tutti per quell'area , ottenuto sommando il punteggio gruppo del richiedente con quello di tutti i partecipanti al gruppo a partire dal momento dell'inoltro (perché prima era possibile anche disdire e quindi decrementare il punteggio) della prenotazione. In caso di pari merito si guarda l'ordine cronologico , considerando la data di inserimento della prenotazione nel database.
Il criterio soddisfa due politiche di marketing :
-assegnare 5 punti ai senza contratto (e non 0), serve a incentivare le persone a formare gruppi più grandi possibili anche se questo vuol dire aggiungere persone senza contratto o comunque con una priorità bassa dal punto di vista del punteggio , insomma .per quanto bassi siano i punteggi ,essi fanno comunque numero.
-assegnare 20 punti ai personalizzati e 25 ai platinum (che hanno tra l'altro anche degli sconti maggiori) invoglia il cliente a sottoscrivere contratti sempre migliori e quindi le aree allestibili diventano una delle tante spinte per raggiungere questo scopo.
In caso contrario , cioè se la richiesta non viene approvata lo stato della prenotazione viene settato ad "alternativa" perché tanto l'area richiesta è già stata assegnata a qualcun altro.

```
DROP PROCEDURE IF EXISTS AccettaRifiutaPrenotazione;
DELIMITER $$
CREATE PROCEDURE AccettaRifiutaPrenotazione(IN _Codprenotazione CHAR(30))
BEGIN
  declare _Area CHAR(20) default ' ';
  declare _DataAttivita DATE;
  declare _InizioAttivita TIME;
  declare _FineAttivita TIME ;
  declare _StatoArea CHAR(20) default ' ';
  declare _PunteggioGruppo INT default 0;
  declare _DataInvioPrenotazione timestamp;

  SELECT Area,DataInvioPrenotazione,DataAttivita,InizioAttivita,FineAttivita,PunteggioGruppo into
       _Area,_DataInvioPrenotazione,_DataAttivita,_InizioAttivita,_FineAttivita,_PunteggioGruppo
  FROM Prenotazione
  WHERE codprenotazione=_codprenotazione;

  set _StatoArea=ControlloDisponibilitàArea(_Area,_DataAttivita,_InizioAttivita,_FineAttivita);

  IF _StatoArea="libera" THEN

    IF _punteggiogruppo=(SELECT MAX(PunteggioGruppo)
                         FROM prenotazione
                         WHERE area=_area) THEN

          IF NOT EXISTS (SELECT *
                         FROM prenotazione
                         WHERE Area=_area
                         AND
                         punteggiogruppo=_punteggiogruppo
                         AND
                         DataInvioPrenotazione<_DataInvioPrenotazione
                         AND
                         ((InizioAttivita<=_InizioAttivita
                         AND
                           FineAttivita>=_FineAttivita)
                          OR
                           (InizioAttivita>_InizioAttivita
                         AND
                           FineAttivita<=_FineAttivita)
                          OR
                          (InizioAttivita>=_InizioAttivita
                         AND
                           FineAttivita<_FineAttivita)
                          OR
                          (InizioAttivita>=_InizioAttivita
                         AND
                           FineAttivita<=_FineAttivita))) THEN
            UPDATE prenotazione
            SET stato="approvata"
            WHERE codprenotazione=_codprenotazione;
           END IF;
    ELSE
          UPDATE prenotazione
           SET stato="alternativa"
           WHERE codprenotazione=_codprenotazione;
    END IF;
```

```
ELSE
        UPDATE prenotazione
        SET stato="alternativa"
        WHERE codprenotazione=_codprenotazione;
END IF ;
END $$
DELIMITER ;
```

EsaminaRichiesteCentro

La procedure si appoggia a quella di prima nel senso che a meno che non ci siano prenotazioni da confermare nel database essa realizza un ciclo in cui ogni prenotazione viene esaminata chiamando la AccettaRifiuraPrenotazione su di essa.

```
DROP PROCEDURE IF EXISTS EsaminaRichiesteCentro;
DELIMITER $$
CREATE PROCEDURE EsaminaRichiesteCentro(IN _Centro CHAR(20) )
BEGIN
  declare _finito int default 0;
  declare _codprenotazione CHAR(30) default ' ';

  declare PrenotazioniTarget cursor for
  (SELECT codprenotazione
   FROM prenotazione
   WHERE stato="Da confermare");

   declare continue handler for not found set _finito=1;

  IF NOT EXISTS (SELECT *
                        FROM Prenotazione P inner join AreaAllestibile A on P.Area=A.CodARea
                        WHERE A.Sede=_Centro
                        AND
                        P.Stato="Da confermare") THEN

    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT="Nessuna richiesta di prenotazione da esaminare";
 END IF;


  open PrenotazioniTarget;

  Esamina: LOOP
    BEGIN
     FETCH PrenotazioniTarget INTO _codprenotazione;
     IF _finito=1 THEN
       LEAVE Esamina;
     END IF;
       CALL AccettaRifiutaPrenotazione(_codprenotazione);
    END ;
   END LOOP;

  close PrenotazioniTarget;
END $$
DELIMITER ;
```

CaricaSaldoDaPagare

La procedure funziona in maniera molto semplice :
-dapprima carica il saldo del cliente
-in secondo luogo carica il saldo uno alla volta di tutti i partecipanti con un LOOP
Nel calcolo del saldo si considera la tariffa oraria , la tariffa attrezzatura dell'area , se o meno includere nel prezzo quest'ultima e gli sconti dei contratti ,portando così a **un'alta flessibilità** dei prezzi .
Il saldo viene inserito con stato "da pagare" .Una volta pagato andrà poi modificato con una semplice operazione di UPDATE .

```
DROP PROCEDURE IF EXISTS CaricaSaldoDaPagare;
DELIMITER $$
CREATE PROCEDURE CaricaSaldoDaPagare (IN _codprenotazione CHAR(30) )
BEGIN
  declare _finito int default 0;
  declare _cliente CHAR(16) default ' ';
  declare _contratto CHAR(30) default ' ';
  declare _tariffaoraria int default 0;
  declare _tariffaattrezzatura int default 0;
  declare _dataattivita date ;
  declare _inizioattivita time;
  declare _fineattivita time;
  declare _mese char(15) default ' ';
  declare _anno int;
  declare _centro CHAR(20) default ' ';
  declare _saldo DOUBLE ;

  declare Partecipanti cursor for
  (SELECT cliente
   FROM partecipante
   WHERE codprenotazione=_codprenotazione);

  declare continue handler for not found set _finito=1;

  SELECT P.clienterichiedente,P.dataattivita,P.inizioattivita,P.fineattivita,
                    T.TariffaOrariaPerPersona,T.TariffaAttrezzattura,A.Sede
  INTO   _cliente, _dataattivita,_inizioattivita,_fineattivita,
                    _tariffaoraria,_tariffaattrezzatura,_centro
  FROM prenotazione P NATURAL JOIN TariffeAreeAllestibili T INNER JOIN AreaAllestibile A
                   on P.Area=A.CodArea
  WHERE codprenotazione=_codprenotazione;

  set _mese=MONTHNAME(_dataattivita);

   set _anno=YEAR(_dataattivita);

   set _contratto = TipologiaContrattoCliente(_cliente,_centro);

    set _saldo =(time_to_sec(TIMEDIFF(_fineattivita,_inizioattivita)) / 3600 )  * (_tariffaoraria +
            _tariffaattrezzatura) * ScontoContratto(_contratto);

   INSERT INTO saldiareeallestibilidapagare
   VALUES ( _cliente,_saldo,_mese,_anno,"da pagare",_centro);

    UPDATE SaldoAreeAllestibiliCliente
    SET SaldoMese=SaldoMese+ _saldo, SaldoTotale=SaldoTotale+_saldo
    WHERE Cliente=_cliente;
```

```
        open Partecipanti;

  CaricoDebito: LOOP
  BEGIN
    FETCH Partecipanti into _cliente;
    IF _finito=1 THEN
     LEAVE CaricoDebito;
    END IF;
    set _contratto=TipologiaContrattoCliente(_cliente,_centro);
          set _saldo =(time_to_sec(TIMEDIFF(_fineattivita,_inizioattivita)) / 3600 )  * (_tariffaoraria +
            _tariffaattrezzatura) * ScontoContratto(_contratto);
          INSERT INTO saldiareeallestibilidapagare
    VALUES ( _cliente,_saldo,_mese,_anno,"da pagare",_centro);
    UPDATE SaldoAreeAllestibiliCliente
    SET SaldoMese=SaldoMese+ _saldo, SaldoTotale=SaldoTotale+_saldo
    WHERE Cliente=_cliente;

          END ;
   END LOOP;

   close Partecipanti ;


END $$
DELIMITER ;
```

ProponiAlternativa , ProponiTreAlternative, ConfermaAlternativa

Le tre procedure descrivono il ciclo di prenotazioni nel caso di prenotazione alternativa.
ProponiTreAlternative controlla che la prenotazione data sia effettivamente settata come “alternativa” e poi controlla la correttezza di tutte e tre le alternative chiamando la function di utilità che controlla che l’area sia effettivamente libera in quella fascia oraria . Se anche solo una delle proposte non è corretta cade la proposta anche delle altre, viene lanciato un segnale di errore che indica quale delle tre è sbagliata . Le proposte vengono lanciate in contemporanea per evitare conflitti di orario l’una con l’altra.
L’inserimento avviene chiamando tre volte ProponiAlternativa.
ConfermaAlternativa pone lo stato della prenotazione in Prenotazione a “approvata” ed elimina tutte le alternative.

```
DROP PROCEDURE IF EXISTS ProponiAlternativa;
DELIMITER $$
CREATE PROCEDURE ProponiAlternativa ( IN _Codprenotazione CHAR(30),IN _DataAlternativa DATE,
IN _OrarioInizioAlternativo TIME , IN _OrarioFineAlternativo TIME )
BEGIN

    INSERT INTO prenotazionealternativa

VALUES(_codprenotazione,_dataalternativa,_orarioinizioalternativo,_orariofinealternativo,current_timesta
mp);
END $$
DELIMITER ;
```

```
DROP PROCEDURE IF EXISTS ProponiTreAlternative;
DELIMITER $$
CREATE PROCEDURE ProponiTreAlternative (  IN _Codprenotazione CHAR(30),
        IN _DataAlternativa1 DATE,IN _OrarioInizioAlternativo1 TIME , IN _OrarioFineAlternativo1 TIME
,
        IN _DataAlternativa2 DATE,IN _OrarioInizioAlternativo2 TIME , IN _OrarioFineAlternativo2 TIME
,
        IN _DataAlternativa3 DATE,IN _OrarioInizioAlternativo3 TIME , IN _OrarioFineAlternativo3 TIME
)
BEGIN
  declare _stato char(30) default ' ';
  declare _controllo1 char(30) default ' ';
  declare _controllo2 char(30) default ' ';
  declare _controllo3 char(30) default ' ';
  declare _area char(20) default ' ';

   set _stato=(SELECT stato
                FROM prenotazione
               WHERE codprenotazione=_codprenotazione);

    IF _stato<>"alternativa" THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT="Non puoi proporre alternative a questa prenotazione";
           END IF;

    set _area=(SELECT area
                 FROM prenotazione
                 WHERE codprenotazione=_codprenotazione);

    set _controllo1 = ControlloDisponibilitàArea(_area,_dataalternativa1,_orarioinizioalternativo1,
                                                 _orariofinealternativo1);

    IF _controllo1="occupata" THEN
    BEGIN

      DELETE FROM prenotazionealternativa
      WHERE codprenotazione=_codprenotazione;

      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT=" Alternativa 1 non valida,riprova";
           END ;
           END IF;


     set _controllo2 = ControlloDisponibilitàArea(_area,_dataalternativa2,_orarioinizioalternativo2,
                                  _orariofinealternativo2);

    IF _controllo2="occupata" THEN
    BEGIN
      DELETE FROM prenotazionealternativa
      WHERE codprenotazione=_codprenotazione;
```

```
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT=" Alternativa 2 non valida,riprova";
           END ;
           END IF;

    set _controllo3 = ControlloDisponibilitàArea(_area,_dataalternativa3,_orarioinizioalternativo3,
                                    _orariofinealternativo3);

    IF _controllo3="occupata" THEN
    BEGIN
      delete from prenotazionealternativa
      where codprenotazione=_codprenotazione;

      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT=" Alternativa 3 non valida,riprova";
           END ;
           END IF;

     CALL ProponiAlternativa(_codprenotazione,_dataalternativa1,
                                    _orarioinizioalternativo1,_orariofinealternativo1);
     CALL ProponiAlternativa(_codprenotazione,_dataalternativa2,
                                    _orarioinizioalternativo2,_orariofinealternativo2);
     CALL ProponiAlternativa(_codprenotazione,_dataalternativa3,
                                    _orarioinizioalternativo3,_orariofinealternativo3);
END $$
DELIMITER ;



DROP PROCEDURE IF EXISTS ConfermaAlternativa;
DELIMITER $$
CREATE PROCEDURE ConfermaAlternativa(IN _Codprenotazione CHAR(30),IN _DataAlternativa
DATE,
IN _OrarioInizioAlternativo TIME , IN _OrarioFineAlternativo TIME )
BEGIN
  IF NOT EXISTS ( SELECT *
                          FROM prenotazionealternativa
                          WHERE CodPrenotazione=_codprenotazione
                          AND
                          DataAlternativa=_dataalternativa
                          AND
                         OrarioInizioAlternativo=_orarioinizioalternativo
                          AND
                         OrarioFineAlternativo=_orariofinealternativo) THEN

    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT="Nessuna delle alternative corrisponde";
 END IF;


  UPDATE prenotazione
  SET Dataattivita=_Dataalternativa,InizioAttivita=_OrarioInizioalternativo,
     FineAttivita=_orariofinealternativo,stato="approvata"
```

```
  WHERE CodPrenotazione=_codprenotazione;

  delete from prenotazionealternativa
  where codprenotazione=_codprenotazione;
END $$
DELIMITER ;
```

EVENT

L'event tramite cursor controlla che ,per ogni record su Prenotazione con "alternativa" settata , le sue alternative non siano state nel database per più di due giorni.
Se si , viene eliminato il record su Prenotazione e i tre su PrenotazioneAlternativa per propagazione.

```
DROP EVENT IF EXISTS EliminaDopo2Giorni;
DELIMITER $$
CREATE EVENT EliminaDopo2Giorni
ON SCHEDULE EVERY 1 MINUTE STARTS '2017-11-18 00:00:00'
DO
  BEGIN

    declare _giornipassati INT default 0 ;
    declare _datainoltro TIMESTAMP;
    declare _codprenotazione CHAR(30) default ' ';
    declare _finito INT default 0 ;

    declare ControllaProposteAlternative cursor for

    (SELECT codprenotazione
     FROM prenotazione
     WHERE stato="alternativa");

     declare continue handler for not found set _finito=1;

     open ControllaProposteAlternative;

     Controllo: LOOP
       BEGIN

        FETCH ControllaProposteAlternative INTO _codprenotazione;
        IF _finito=1 THEN
          LEAVE Controllo;
                    END IF;
        set _datainoltro=(SELECT DISTINCT DataInoltroPrenotazione
                          FROM prenotazionealternativa
                          WHERE codprenotazione=_codprenotazione);

                    set _giornipassati=TIMESTAMPDIFF(DAY,_datainoltro,current_timestamp);
        IF _giornipassati >=2 THEN
          DELETE FROM prenotazione
          WHERE codprenotazione = _codprenotazione;
                    END IF;
```

```
                    END;
    END LOOP;
END $$
DELIMITER ;
```

SvuotaSaldoMese

Event banale , ogni mese setta il saldo mese a 0 nel profilo aree allestibili del cliente .

```
DROP EVENT IF EXISTS SvuotaSaldoMese;
DELIMITER $$
CREATE EVENT SvuotaSaldoMese
ON SCHEDULE EVERY 1 MONTH STARTS '2017-11-17 00:00:00'
DO
BEGIN
  UPDATE saldoareeallestibilicliente
  SET SaldoMese=0;
END $$
DELIMITER ;
```

Functions

Funzioni di utilità

Le funzioni che andrò di seguito a descrivere si può dire che non hanno "vita propria" nel senso che la loro vera utilità verrà chiarita andando poi ad esaminare le vere operazioni, quelle più corpose, ma sono comunque fondamentali perché essendo spesso utilizzate mi permettono di risparmiare ogni volta righe e righe di codice.

TipologiaContrattoCliente

Come accennato prima , le aree allestibili per gestire l'ordine delle prenotazioni non seguono un criterio del tutto cronologico ma gran parte della scelta dipende dal tipo del contratto del cliente che possiede
-per quel determinato centro
-in quel determinato momento(essendo i contratti passati scaduti registrati nel database) → per questo la funzione ha la proprietà di essere NON DETERMINISTICA.
Questa function dunque presi in ingresso il codice identificativo del cliente e del centro mi ridà una stringa che identifica il tipo di contratto posseduto.
L'implementazione funziona nel seguente modo : due variabili , una per i contratti standard e una per quelli personalizzati, "pescano" rispettivamente su AutorizzazioneCentro e su AutorizzazionePersonalizzata andando quindi ad esaminare se esiste nell'una o nell'altra che l'ultimo contratto fatto sia per quel centro e in caso affermativo anche che esso sia ancora valido.
Se il responso è negativo per entrambe ,allora vengono settate tutte e due a NULL e la stringa ritornata è "senza contratto" , altrimenti ,visto che il caso che entrambe le variabili siano NOT NULL non esiste ,con un semplice ELSE si sceglie quale strada prendere (standard o personalizzato) e in ogni caso la funzione ritorna il tipo di contratto.

```
DROP FUNCTION IF EXISTS TipologiaContrattoCliente;
DELIMITER $$
CREATE FUNCTION TipologiaContrattoCliente (  _Cliente CHAR(50) ,_Sede CHAR(50)) RETURNS CHAR(30)
NOT DETERMINISTIC
BEGIN
  declare _tipologiacontrattostandard CHAR(30) default ' ';
  declare _tipologiacontrattopersonalizzato CHAR(30) default ' ';
```

```
 SET _tipologiacontrattostandard= (SELECT C.Tipologia
                                   FROM Contratto C INNER JOIN AutorizzazioneCentro AC
                                        ON C.CodContratto=AC.Contratto
                                   WHERE C.DataSottoscrizione=(SELECT Max(C2.DataSottoscrizione)
                                                               FROM Contratto C2
                                                               WHERE C.Cliente=C2.Cliente)
                                   AND C.DataSottoscrizione + INTERVAL C.DuratainMesi
MONTH>=Current_Date()
                                   AND AC.Centro=_Sede
                                   AND C.Cliente=_Cliente);

SET _tipologiacontrattopersonalizzato= (SELECT C.Tipologia
                                        FROM Contratto C INNER JOIN AutorizzazionePersonalizzata AP
                                             ON C.CodContratto=AP.Contratto
                                        WHERE C.DataSottoscrizione=(SELECT Max(C2.DataSottoscrizione)
                                                                    FROM Contratto C2
                                                                    WHERE C.Cliente=C2.Cliente)
                                        AND C.DataSottoscrizione + INTERVAL C.DuratainMesi
                                                                    MONTH>=Current_Date()
                                        AND AP.Centro=_Sede
                                        AND C.Cliente=_Cliente);

IF (_tipologiacontrattopersonalizzato IS NULL AND _tipologiacontrattostandard IS NULL) THEN
    RETURN 'senza contratto';
ELSE IF _tipologiacontrattostandard IS NOT NULL THEN
       RETURN _tipologiacontrattostandard;
              ELSE
        RETURN _tipologiacontrattopersonalizzato;
                    END IF;
          END IF;
END $$
DELIMITER ;
```

ControlloDisponibilitàArea

Un area ovviamente prima di essere assegnata o di essere prenotata va controllato se essa sia effetivamente libera per quell'orario e in quel giorno.
Per questo si controlla sia nelle prenotazioni "principali" (solo le prenotazioni "approvate" occupano l'area) sia in quelle alternative (per le quali si occupa momentaneamente l'area , cioè se a un cliente viene offerta in un determinato orario una data sala come opzione alternativa alla sua prenotazione essa non potrà né essere offerta a un altro cliente né momentaneamente essere prenotata da altri).
L'implementazione, abbastanza macchinosa ,considera tutti i possibili accavallamenti temporali tra un un 'attività e un ' altra.

```
DROP FUNCTION IF EXISTS ControlloDisponibilitàArea;
DELIMITER $$
CREATE FUNCTION ControlloDisponibilitàArea (_Area CHAR(20),_DataAttivita DATE , _InizioAttivita TIME,
                              _FineAttivita TIME)
RETURNS CHAR(20) NOT DETERMINISTIC
BEGIN
   declare _controllo char(20) default ' ';
   IF EXISTS (select codprenotazione
         from prenotazione
```

```
        where stato='approvata'
            AND
            Area=_Area
            AND
            DataAttivita=_DataAttivita
            AND
            (
            (InizioAttivita<=_InizioAttivita
             AND
             FineAttivita>=_FineAttivita)
             OR
             (InizioAttivita>=_InizioAttivita
             AND
             FineAttivita<=_FineAttivita)
             OR
            (InizioAttivita<_FineAttivita
             AND
             FineAttivita>=_FineAttivita)
             OR
            (InizioAttivita<=_InizioAttivita
             AND
             FineAttivita>_InizioAttivita)))
             /* controllo disponibilità area */

             OR

             EXISTS
                ( SELECT *
                  FROM Prenotazionealternativa PA NATURAL JOIN Prenotazione P
                  WHERE P.area=_area
                   AND
                   PA.DataAlternativa=_DataAttivita
                  AND
                 ( (PA.OrarioInizioAlternativo<=_InizioAttivita
                   AND
                    PA.OrarioFineAlternativo>=_FineAttivita)
                   OR
                   (PA.OrarioInizioAlternativo>=_InizioAttivita
                   AND
                    PA.OrarioFineAlternativo<=_FineAttivita)
                   OR
                  (PA.OrarioInizioAlternativo<_FineAttivita
                   AND
                   PA.OrarioFineAlternativo>=_FineAttivita)
                   OR
                   (PA.OrarioInizioAlternativo<=_InizioAttivita
                   AND
                    PA.OrarioFineAlternativo>_InizioAttivita)))

    THEN

   SET _controllo="occupata";

ELSE

   SET _controllo="libera";

END IF;
```

```
RETURN _controllo;

END $$
DELIMITER ;
```

ScontoContratto

Il tipo di contratto non gioca un ruolo chiave solo nell'assegnazione dell'area ma anche nel suo prezzo(tariffaoraria+ tariffaattrrezzatura) ,che viene moltiplicato per un coefficiente che coincide proprio con il ritorno di questa funzione, sempre per incentivare i clienti ad acquistare contratti più sostanziosi.
Da notare che i senza contratto possono utilizzare le aree ma a prezzo pieno (coefficiente 1).
L'implementazione è un banale case che ritorna un valore double diverso a secondo dell'argomento della funzione.
È quasi evidente la completa sinergia tra questa e la precedente funzione TipologiaContrattoCliente.

```
DROP FUNCTION IF EXISTS ScontoContratto;
DELIMITER $$
CREATE FUNCTION ScontoContratto(_tipocontratto CHAR(30))
RETURNS DOUBLE DETERMINISTIC
BEGIN
  CASE
    WHEN _tipocontratto="senza contratto" THEN
      RETURN 1;
          WHEN _tipocontratto="silver" THEN
      RETURN 0.8;
          WHEN _tipocontratto="gold" THEN
      RETURN 0.6;
          WHEN _tipocontratto="personalizzato" THEN
      RETURN 0.4;
          WHEN _tipocontratto="platinum" THEN
      RETURN 0.2;
  END CASE;
END $$
DELIMITER ;
```

Integratori

Tabelle

```
CREATE TABLE IF NOT EXISTS Fornitore(
NomeAzienda char(80) not null,
FormaSocietaria char(30) not null,
PartitaIva char(25) not null,
Indirizzo char(30) not null,
Citta char(80) not null,
Telefono char(12) not null,
PRIMARY KEY(PartitaIVA)
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi : NomeAzienda,FormaSocietaria,PartitaIva,Indirizzo,Citta,Telefono.
Chiave primaria : PartitaIVA.
Vincoli di integrità referenziale:-
Dipendenze funzionali : PartitaIVA→ NomeAzienda,FormaSocietaria,Indirizzo,Citta,Telefono

```
CREATE TABLE IF NOT EXISTS Integratore(
NomeCommerciale char(30) not null,
Tipologia char(25) not null,
SostanzaContenuta char(80) not null,
NumeroPezziConfezione int not null,
QuantitaPerPezzo int not null,
Fornitore char(80) not null,
PRIMARY KEY(NomeCommerciale,Fornitore),
CONSTRAINT ChiFornisce
FOREIGN KEY (Fornitore)
REFERENCES Fornitore(PartitaIva)
ON DELETE CASCADE
ON UPDATE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi:NomeCommerciale,Tipologia,SostanzaContenuta,NumeroPezziConfezione,QuantitaPerPezzo,Fornitore
Chiave primaria : NomeCommerciale,Fornitore
Vincoli di integrità referenziale:ChiFornisce.
Dipendenze funzionali : NomeCommerciale,Fornitore→ Tipologia,SostanzaContenuta,NumeroPezziConfezione,QuantitaPerPezzo
Note:

```
CREATE TABLE IF NOT EXISTS Ordine(
CodOrdine char(30) not null,
Fornitore char(25) not null,
Centro char(20) not null,
Stato char(20) not null,
MetodoPagamento char(25) not null,
PRIMARY KEY (CodOrdine),
CONSTRAINT Fornitura
FOREIGN KEY (Fornitore)
REFERENCES Fornitore(PartitaIVA)
ON DELETE NO ACTION
ON UPDATE CASCADE,
CONSTRAINT CentroInvioOrdine
FOREIGN KEY (Centro)
REFERENCES Centro(CodCentro)
ON DELETE NO ACTION
ON UPDATE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: CodOrdine,Fornitore,Centro,Stato,MetodoPagamento
Chiave primaria : CodOrdine
Vincoli di integrità referenziale:Fornitura,CentroInvioOrdine.
Dipendenze funzionali : CodOrdine → Fornitore,Centro,Stato,MetodoPagamento
Note :
Stato assume come valori “incompleto” o “evaso” : incompleto è lo stato in cui si possono acquistare prodotti da aggiungere al carrello spesa ; evaso è lo stato in cui l’ordine è già stato inviato e quindi non si possono aggiungere prodotti al carrello.

```
CREATE TABLE IF NOT EXISTS OrdiniEvasi(
CodOrdine char(30) not null,
DataInvioOrdine date not null,
DataConsegnaPreferita date not null,
Stato char(20) not null,
PRIMARY KEY(CodOrdine),
CONSTRAINT OrdineRiferito
FOREIGN KEY (CodOrdine)
REFERENCES Ordine(CodOrdine)
ON DELETE CASCADE
ON UPDATE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi :CodOrdine,DataInvioOrdine,DataConsegnaPreferita,Stato.
Chiave primaria : CodOrdine.
Vincoli di integrità referenziale : OrdineRiferito.
Dipendenze funzionali :CodOrdine→ DataInvioOrdine,DataConsegnaPreferita,Stato
Note :
Stato assume 4 valori :
-‘in attesa’ indica che ancora la merce non è stata inviata dal fornitore.
-‘merce inviata’ indica che il fornitore ha inviato la merce
-‘merce arrivata’ indica che la merce è arrivata in magazzino
-‘fallito’ indica che data di consegna è passata , cioè la merce desiderata non è arrivata entro quel termine e quindi l’ordine è decaduto.

```
CREATE TABLE IF NOT EXISTS Acquisto(
CodOrdine char(30) not null,
CodProdotto char(30) not null,
NomeIntegratore char(60) not null,
Quantita int not null,
PRIMARY KEY(CodProdotto),
CONSTRAINT OrdineRiferimento
FOREIGN KEY (CodOrdine)
REFERENCES Ordine(CodOrdine)
ON DELETE CASCADE
ON UPDATE CASCADE,
CONSTRAINT IntegratoreRiferimento
FOREIGN KEY (NomeIntegratore)
REFERENCES Integratore(NomeCommerciale)
ON DELETE CASCADE
ON UPDATE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi :CodOrdine,CodProdotto,NomeIntegratore,Quantita.
Chiave primaria: CodProdotto.
Vincoli di integrità :OrdineRiferimento,IntegratoreRiferimento.
Dipendenze funzionali : CodProdotto → CodOrdine, NomeIntegratore,Quantita
Note:
Acquisto altro non è che i prodotti ordinati al fornitore.

```
CREATE TABLE IF NOT EXISTS Magazzino(
```

CodMagazzino char(25) not null,
Centro char(20) not null,
CapienzaMassima int not null,
PRIMARY KEY(CodMagazzino),
CONSTRAINT CentroMagazzino
FOREIGN KEY (Centro)
REFERENCES Centro(CodCentro)
ON UPDATE CASCADE
ON DELETE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
Attributi : CodMagazzino,Centro,CapienzaMassima,CapienzaAttualeVirtuale
Chiave primaria: CodMagazzino
Vincoli di integrità referenziale :CentroMagazzino
Dipendenze funzionali : CodMagazzino→ Centro,CapienzaMassima
NOTE :
RIDONDANZE:CapienzaAttualeVirtuale
CapienzaAttualeVirtuale è la capienza del magazzino considerata anche la merce degli ordini.

CREATE TABLE IF NOT EXISTS MerceMagazzino(
CodProdotto char(30) not null,
Integratore char(60) not null,
Magazzino char(25) not null,
Quantita int not null,
DataScadenza date not null,
Rank char(10) ,
PRIMARY KEY(CodProdotto),
CONSTRAINT IntegratoreContenuto
FOREIGN KEY(Integratore)
REFERENCES Integratore(NomeCommerciale)
ON DELETE CASCADE
ON UPDATE CASCADE,
CONSTRAINT MagazzinoContenimento
FOREIGN KEY (Magazzino)
REFERENCES Magazzino(CodMagazzino)
ON DELETE CASCADE
ON UPDATE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
Attributi : CodProdotto,Integratore,Magazzino,Quantita,DataScadenza,Rank
Chiave primaria : CodProdotto
Vincoli di integrità referenziale : IntegratoreContenuto,MagazzinoContenimento
Dipendenze funzionali :CodProdotto → Integratore,Magazzino,Quantita,DataScadenza,Rank
NOTE :
Rank è un attributo connesso a una funzionalità analytics di controllo sulle date di scadenza della merce in magazzino:
-a 30 giorni dalla data di scadenza la merce ha priorità ‘ high’;
-a 60 giorni dalla data di scadenza la merce ha priorità ‘medium’;
-a 90 giorni dalla data di scadenza la merce ha priorità ‘low’;
Il Rank influisce sul prezzo di vendità dei prodotti.

CREATE TABLE IF NOT EXISTS InventarioMagazzino(
Magazzino char(25) not null,

```
Integratore char(60) not null,
Costo INT not null,
PRIMARY KEY(Magazzino,Integratore),
CONSTRAINT EsistenzaMagazzino
FOREIGN KEY(Magazzino)
REFERENCES Magazzino(CodMagazzino)
ON DELETE CASCADE
ON UPDATE CASCADE)
ENGINE=InnoDB  DEFAULT CHARSET=latin1;
```

Attributi : Magazzino,Integratore,Costo
Chiave primaria :Magazzino,Integratore.
Vincoli di integrità referenziale : EsistenzaMagazzino.
Dipendenze funzionali :Magazzino,Integratore → Costo
NOTE :
L'inventario del magazzino serve per fissare per ogni centro il prezzo di vendita di un determinato integratore.

```
CREATE TABLE IF NOT EXISTS Vendite(
CodVendita char(25) not null,
Cliente char(16) not null,
Centro char(20) not null,
Integratore char(30) not null,
DataVendita date not null,
Quantita int not null,
PRIMARY KEY (CodVendita),
CONSTRAINT Vendita_Centro
FOREIGN KEY(Centro)
REFERENCES Centro(CodCentro)
ON DELETE CASCADE
ON UPDATE CASCADE,
CONSTRAINT Vendita_Integratore
FOREIGN KEY(Integratore)
REFERENCES Integratore(NomeCommerciale)
ON DELETE CASCADE
ON UPDATE CASCADE,
CONSTRAINT Vendita_CLiente
FOREIGN KEY(Cliente)
REFERENCES Cliente(CodFiscale)
ON DELETE CASCADE
ON UPDATE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi :CodVendita,Cliente,Centro,Integratore,DataVendita,Quantita
Chiave primaria:CodVendita
Vincoli di integrità referenziali :Vendita_Centro,Vendita_Integratore,Vendita_Cliente
Dipendenze funzionali : CodVendita → Cliente,Centro,Integratore,DataVendita,Quantita

Stored procedure

CreaOrdine

È la funzionalità principale di quest'area .
Prima di creare l'ordine controlla che nessun altro dello stesso centro abbia creato un ordine con stato 'incompleto' , ovvero non ancora inviato,cioè il segnale avverte l'utente della palestra che invece di aprire un altro ordine può aggiungere i prodotti richiesti al carrello spesa dell'altro ordine.Se tutto va bene l'ordine viene creato.

```
DROP PROCEDURE IF EXISTS CreaOrdine;
DELIMITER $$
CREATE PROCEDURE CreaOrdine (IN _CodOrdine CHAR(30), IN _Fornitore CHAR(25), IN _Centro CHAR(20) ,
                    IN _MetodoPagamento CHAR (25))
BEGIN

 IF EXISTS ( SELECT *
                  FROM Ordine
                 WHERE Fornitore=_Fornitore
                        AND
                       Stato='incompleto'
                        AND
                       Centro=_Centro) THEN
          SIGNAL SQLSTATE '45000'
          SET MESSAGE_TEXT="Il centro ha già aperto una procedura di acquisto per questo venditore";
 END IF;

 INSERT INTO Ordine
 VALUES(_CodOrdine,_Fornitore,_Centro,'incompleto',_MetodoPagamento);

 END $$
 DELIMITER ;
```

AggiungiProdottoInventario

Questa funzionalità che permette di aggiungere un prodotto all'inventario(**azione obbligatoria prima di fare l'ordine di quello specifico prodotto)** dà uno specifico segnale di errore in caso l'integratore inserito non esista.

```
 DROP PROCEDURE IF EXISTS AggiungiProdottoInventario;
DELIMITER $$
CREATE PROCEDURE AggiungiProdottoInventario(IN _CodIntegratore char(60), IN _Magazzino char(25),
                                                    IN _Costo INT)
BEGIN

 IF _Codintegratore NOT IN (SELECT NomeCommerciale
                                  FROM Integratore ) THEN
           SIGNAL SQLSTATE '45000'
   SET MESSAGE_TEXT=" L'integratore non esiste ";
 END IF;

 INSERT INTO InventarioMagazzino
 VALUES (_Magazzino,_CodIntegratore,_Costo);

END $$
DELIMITER ;
```

InviaOrdine

La procedure si articola nel seguente modo :

- 2 controlli, ordine inesistente e ordine già inviato .
- Attraverso un cursor si estrae uno a uno i prodotti del carrello in un LOOP.
- Per ogni record si fa un UPDATE : l'update della CapienzaVirtuale del magazzino.
- Dopo il loop si aggiorna lo stato dell'ordine a 'evaso'.
- Si inserisce l'ordine in ordini evasi con la data di consegna passata come argomento.

```
DROP PROCEDURE IF EXISTS InviaOrdine;
DELIMITER $$
CREATE PROCEDURE InviaOrdine (IN _CodOrdine CHAR(30),IN _DataConsegnaPreferita DATE)
BEGIN
 declare _centro char(20) default '';
 declare _nomeintegratore char(60) default ' ';
 declare _quantita int default 0;
 declare _finito int default 0;
 declare _capienzavirtuale INT default 0;
 declare _capienzamassima INT default 0;
 declare _quantitaTot INT default 0;

 declare ProdottiOrdine cursor for
 (SELECT NomeIntegratore,Quantita
  FROM Acquisto
  WHERE CodOrdine=_CodOrdine);

  declare continue handler for not found
  set _finito=1;


 IF NOT EXISTS (SELECT *
                FROM Ordine
                WHERE CodOrdine=_CodOrdine) THEN

       SIGNAL SQLSTATE '45000'
       SET MESSAGE_TEXT = "Ordine inesistente";
 ELSE
   IF EXISTS (SELECT*
              FROM Ordine
              WHERE CodOrdine=_CodOrdine
              AND
              Stato='evaso') THEN
                        SIGNAL SQLSTATE '45000'
         SET MESSAGE_TEXT="Ordine già inviato";
 END IF;
 END IF;

SELECT O.centro,M.capienzamassima,M.capienzaattualevirtuale into _centro,_capienzamassima,_capienzavirtuale
 FROM Ordine O natural join Magazzino M
 WHERE O.CodOrdine=_CodOrdine;



 OPEN ProdottiOrdine;

 Conta : LOOP
  BEGIN
   FETCH ProdottiOrdine INTO _nomeintegratore,_quantita;

   IF _finito=1 THEN
```

```
      LEAVE Conta;
           END IF;

    set _quantitaTot=_quantitaTot+_quantita;

          END;
          END LOOP;

  CLOSE ProdottiOrdine;

IF _capienzavirtuale+_quantitaTot>_capienzamassima THEN
   SIGNAL SQLSTATE '45000'
   SET MESSAGE_TEXT = CONCAT( "Merce in sovrannumero: levare almeno",':',
(_capienzavirtuale+quantitaTot)-capienzamassima);
END IF;

    UPDATE Magazzino
    SET CapienzaAttualeVirtuale=CapienzaAttualeVirtuale+_quantitaTot
    Where Centro=_centro;

  UPDATE Ordine
  SET Stato='evaso'
  WHERE CodOrdine=_CodOrdine;

  INSERT INTO OrdiniEvasi
  VALUES(_CodOrdine,current_date,_DataConsegnaPreferita,'in attesa');

END $$
DELIMITER ;
```

AggiungiAlCarrello / DecrementaCarrello

L'idea di queste procedure è quella di un carrello spesa da cui mettere e togliere prodotti, questo è quello che accade una volta inserito un ordine in stato 'incompleto'.
In entrambe si controlla che l'ordine non sia stato inviato , altrimenti non si potrebbe applicare alcuna modifica al carrello.
AggiungiAlCarrello controlla pure che gli ordini non superino la capienza massima del magazzino mentre la sua controparte controlla che la quantità del decremento non superi quella originale e in caso positivo aggiunge un record su acquisto.
La DecrementaCarrello inoltre si differenzia in due casi : se il decremento è uguale alla quantità originale allora opera una DELETE , altrimenti opera un semplice UPDATE.

```
DROP PROCEDURE IF EXISTS AggiungiAlCarrello;
DELIMITER $$
CREATE PROCEDURE AggiungiAlCarrello (IN _CodOrdine CHAR(30),IN _CodProdotto CHAR(30), IN
_NomeIntegratore CHAR(30) ,
                          IN _Quantità INT)

BEGIN
  declare _Capienza INT DEFAULT 0;
  declare _CapienzaMax INT DEFAULT 0;
  declare _fornitore CHAR(80) DEFAULT ' ';
  declare _magazzino CHAR(25) default ' ';

  set _magazzino=(SELECT codmagazzino
                  FROM ordine natural join magazzino
                  WHERE codordine=_codordine);
```

```
 IF _nomeintegratore NOT IN (SELECT integratore
                                FROM inventariomagazzino
                                WHERE magazzino=_magazzino) THEN
     SIGNAL SQLSTATE '45000'
     SET MESSAGE_TEXT="Prima aggiungere il prodotto nell'inventario !";

    END IF;



 IF NOT EXISTS (SELECT *
                FROM Ordine
                WHERE CodOrdine=_CodOrdine) THEN
          SIGNAL SQLSTATE '45000'
          SET MESSAGE_TEXT = "Ordine inesistente";
 ELSE
  IF EXISTS (SELECT *
               FROM Ordine
               WHERE CodOrdine=_CodOrdine
               AND
               Stato='evaso') THEN
       SIGNAL SQLSTATE '45000'
       SET MESSAGE_TEXT="Ordine già inviato";
 END IF;
 END IF;

 SET _fornitore = (SELECT fornitore
                FROM Ordine
                WHERE CodOrdine=_CodOrdine);

 IF _NomeIntegratore NOT IN (SELECT NomeCommerciale
                                FROM Integratore
                                WHERE Fornitore=_fornitore) THEN
       SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT='Il fornitore non vende il prodotto richiesto';
 END IF;

 INSERT INTO Acquisto
 VALUES (_CodOrdine,_Codprodotto,_NomeIntegratore,_Quantita);

END $$
DELIMITER ;




DROP PROCEDURE IF EXISTS DecrementaCarrello;
DELIMITER $$
CREATE PROCEDURE DecrementaCarrello (IN _CodProdotto CHAR(30),
                        IN _Decremento INT)

BEGIN
 declare _quantita INT default 0;
 declare _codordine CHAR(30) default ' ';
 declare _stato CHAR(10)  default ' ';

 set _stato=(SELECT O.stato
              FROM ordine O NATURAL JOIN acquisto A
```

```
            WHERE codprodotto=_codprodotto);

 set _codordine =(SELECT codordine
                  FROM acquisto
                  WHERE codprodotto=_codprodotto);

 IF _stato='evaso' THEN
  SIGNAL SQLSTATE '45000'
  SET MESSAGE_TEXT="L'ordine è stato inviato";
 END IF;


 set _quantita = (SELECT quantita
                  FROM acquisto
                  WHERE codprodotto=_codprodotto);

 IF _Decremento > _quantita THEN
  SIGNAL SQLSTATE '45000'
  SET MESSAGE_TEXT=" Lo slot non contiene una quantità simile";
 END IF;

 IF _Decremento = _quantita THEN

 DELETE FROM Acquisto
 WHERE CodProdotto=_codprodotto;

 END IF;

 IF _Decremento < _quantita THEN

 UPDATE acquisto
 SET quantita=quantita-_Decremento
 WHERE codprodotto=_codprodotto;
 END IF;

END $$
DELIMITER ;
```

StatoMerceInviata / StatoMerceArrivata

Aggiorna lo stato della merce .
Se la merce è arrivata la direzione invia una notifica per far sì che qualcuno stipi la merce.

```
DROP PROCEDURE IF EXISTS StatoMerceArrivata;
DELIMITER $$
CREATE PROCEDURE StatoMerceArrivata (IN _codordine CHAR(30) )
BEGIN
  declare _statoMerce char(20) default ' ';

  SET _statoMerce=(select stato
            from ordinievasi
            where CodOrdine=_codOrdine);

  IF (_statoMerce <> 'merce inviata' ) AND (_statoMerce <> 'merce arrivata') THEN
   SIGNAL SQLSTATE '45000'
   SET MESSAGE_TEXT="La merce non è stata inviata";
  END IF;
```

```
  IF (_statoMerce = 'merce arrivata') THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT="La merce è già arrivata ed è in attesa di essere stipata";
  END IF;


  UPDATE OrdiniEvasi
  SET stato='merce arrivata'
  WHERE codordine=_codordine;

END $$
DELIMITER ;


DROP PROCEDURE IF EXISTS StatoMerceInviata;
DELIMITER $$
CREATE PROCEDURE StatoMerceInviata (IN _codordine CHAR(30) )
BEGIN
  declare _statoMerce char(20) default ' ';

  SET _statoMerce=(select stato
                from ordinievasi
                where CodOrdine=_codOrdine);

  IF (_statoMerce = 'merce arrivata') THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT="La merce è già arrivata ed è in attesa di essere stipata";
  END IF;
  IF (_statoMerce = 'merce inviata') THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT="La merce è già stata inviata";
  END IF;

  UPDATE OrdiniEvasi
  SET stato='merce inviata'
  WHERE codordine=_codordine;

END $$
DELIMITER ;
```

StipaMerce

La procedure realizza l'azione del caricamento della merce in magazzino.
Implementazione :
-controlla che la merce sia arrivata (stato = merce arrivata su ordinievasi)
-LOOP che per ogni stock (ogni record di acquisto) carica la merce in MerceMagazzino.
-Dopodichè cancella l'ordine(e per propagazione gli acquisti).

```
  DROP PROCEDURE IF EXISTS StipaMerce;
DELIMITER $$
CREATE PROCEDURE StipaMerce (IN _CodOrdine CHAR(30), IN _DataScadenza DATE )
BEGIN


  declare _finito int default 0;
  declare _nomeintegratore char(60) default ' ';
  declare _quantita int default 0;
  declare _magazzino char(25) default ' ';
```

```
 declare _stato char(20) default ' ';
 declare _codprodotto char(30) default ' ';
 declare _centro char(20) default ' ';


 declare ProdottiOrdine cursor for
 (select CodProdotto,NomeIntegratore,Quantita
  from Acquisto
  where CodOrdine=_CodOrdine);

  declare continue handler for not found set _finito=1;

  set _centro = (select centro
                 from ordine
                 where codordine=_codordine);

  set _magazzino=( select M.CodMagazzino
                   from Magazzino M NATURAL JOIN Ordine O
                   where O.CodOrdine=_CodOrdine);

  set _stato=(select stato
              from ordinievasi
              where CodOrdine=_CodOrdine);

   IF _stato <> 'merce arrivata' THEN
     SIGNAL SQLSTATE '45000'
     SET MESSAGE_TEXT="La merce non è ancora arrivata";
           END IF;

  open ProdottiOrdine;

  AggiornaDeposito : LOOP
   BEGIN
    FETCH ProdottiOrdine INTO _codprodotto,_nomeintegratore,_quantita;

    IF _finito=1 THEN
      LEAVE AggiornaDeposito;
            END IF;

    INSERT INTO MerceMagazzino
    VALUES(_codprodotto,_nomeintegratore,_magazzino,_quantita,_DataScadenza,NULL);

           END;
           END LOOP;

   close ProdottiOrdine;

   delete from Ordine where CodOrdine=_CodOrdine;
END $$
DELIMITER ;
```

VenditaIntegratori

Dopo aver trattato la parte degli acquisti passiamo alla parte delle vendite.
In primis però definiamo questa funzione: essa stabilisce uno sconto a seconda della data di scadenza del prodotto da cui dipende lo stato del rank.
-high: prodotto messo a metà prezzo (prezzo d' acquisto);
-medium: 25% di sconto;
-low: 10% di sconto.

In pratica questi sconti servono a vendere il prima possibile i prodotti che stanno per scadere , in primis quelli con rank “high” per cercare di non andare in perdita.

ScontoRank

```
DROP FUNCTION IF EXISTS ScontoRank;
DELIMITER $$
CREATE FUNCTION ScontoRank(_rank CHAR(10))
RETURNS DOUBLE DETERMINISTIC
BEGIN
  CASE
    WHEN _rank="high" THEN
    RETURN 0.5;
    WHEN _rank="medium" THEN
    RETURN 0.75;
    WHEN _rank="low" THEN
    RETURN 0.9;
    WHEN _rank is NULL THEN
    RETURN 1;
  END CASE;
END $$
DELIMITER ;
```

Ma lo stato chi lo aggiorna ?

CalcolaRankMerce

```
DROP EVENT IF EXISTS CalcolaRankMerce;
DELIMITER $$
CREATE EVENT CalcolaRankMerce
ON SCHEDULE EVERY 1 DAY STARTS '2017-11-19 00:00:00'
DO
  BEGIN
    declare _finito INT default 0;
    declare _codprodotto char(30) default ' ';
    declare _datascadenza DATE;

    declare ProdottiMagazzino cursor for
    (select codprodotto,datascadenza
     from mercemagazzino
     where DATEDIFF(datascadenza,current_date)<=90);

     declare continue handler for not found
     set _finito=1;

    open ProdottiMagazzino;

    Ranking : LOOP
    BEGIN
    FETCH ProdottiMagazzino INTO _codprodotto,_datascadenza;
    IF _finito=1 THEN
      LEAVE Ranking;
            END IF;
```

```
   CASE

    WHEN DATEDIFF(_datascadenza,current_date)<=30 THEN

      UPDATE MerceMagazzino
      SET Rank='high'
      WHERE codprodotto=_codprodotto;

          WHEN DATEDIFF(_datascadenza,current_date)<=60
        AND DATEDIFF(_datascadenza,current_date)>30 THEN

                  UPDATE Mercemagazzino
      SET Rank='medium'
      WHERE codprodotto=_codprodotto;

          WHEN DATEDIFF(_datascadenza,current_date)<=90
                    AND DATEDIFF(_datascadenza,current_date)>60 THEN

                  UPDATE MerceMagazzino
      SET Rank='low'
      WHERE codprodotto=_codprodotto;

         END CASE;
   END ;
   END LOOP;

   close ProdottiMagazzino;

END $$
DELIMITER ;
```

Ecco infine la procedure vera e propria, essa opera un'azione molto semplice ma con alcuni effetti collaterali:
-in primis controlla che in magazzino ci sia abbastanza merce da vendere.
-opera un decremento sulla capienza del magazzino e sullo stock di merce.In caso il decremento sia uguale alla quantita originale opera una DELETE su mercemagazzino.
-inserisce un record nel Log, per l'analytics
-infine inserisce il record nelle Vendite.

```
DROP PROCEDURE IF EXISTS VenditaIntegratori;
DELIMITER $$
CREATE PROCEDURE VenditaIntegratori (IN _codvendita CHAR(25),IN _cliente CHAR(16), IN _centro
CHAR(20),
                          IN _integratore CHAR(30),IN _datavendita DATE,IN _quantita INT,
                          IN _codprodotto CHAR(30))

BEGIN
    declare quantitadepositoprodotto int default 0;
    declare _costo int default 0;
    declare _magazzino CHAR(25) default ' ';
    declare _rank char(10) default ' ';

    set _magazzino=(select codmagazzino
                        from magazzino
                where centro=_centro);

        set _costo=(select costo
              from inventariomagazzino
```

```
            where magazzino=_magazzino
                and
                integratore=_integratore);



    select quantita,rank into quantitadepositoprodotto,_rank
    from MerceMagazzino
    where codprodotto=_codprodotto;

    IF quantitadepositoprodotto<_quantita THEN
         SIGNAL SQLSTATE '45000'
         SET MESSAGE_TEXT="Al momento quello stock non dispone della quantità richiesta";
           END IF;

    INSERT INTO vendite
    VALUES (_codvendita,_cliente,_centro,_integratore,_datavendita,_quantita);

    UPDATE Magazzino
    SET CapienzaAttualeVirtuale=CapienzaAttualeVirtuale-_quantita
    WHERE Centro=_centro;

          IF quantitadepositoprodotto-_quantita=0 THEN

          DELETE FROM mercemagazzino
          WHERE codprodotto=_codprodotto;

    ELSE

          UPDATE MerceMagazzino
    SET quantita=quantita-_quantita
    WHERE CodProdotto=_codprodotto;

    END IF;

    INSERT INTO Log_VenditeIntegratori
    VALUES(_codvendita,_integratore,_quantita,_quantita*_Costo*ScontoRank(_rank),_centro);


END $$
DELIMITER ;
```

E se gli ordini falliscono ?

Una procedure , la **EliminaOrdiniFalliti** controlla tutti i record su Ordinievasi che hanno lo stato ‘fallito’ e su di essi chiama un’altra procedure, la **DecrementaMerceMagazzino** che si occupa di eliminarli**.**
Con un LOOP quest’ultima elimina uno a uno tutti i record di acquisto connessi a quell’ordine andando ad intaccare così la CapienzaVirtuale del magazzino.

```
DROP PROCEDURE IF EXISTS DecrementaMagazzino;
DELIMITER $$
CREATE PROCEDURE DecrementaMagazzino( IN _centro char(20), IN _codordine char(30))
BEGIN
  declare _finito INT default 0;
```

```
 declare _nomeintegratore char(60) default ' ';
 declare _quantita INT default 0;
 declare _quantitatot INT default 0;

 declare ProdottiDaEliminare cursor for
 (select nomeintegratore,quantita
  from acquisto
  where codordine=_codordine);

  declare continue handler for not found set _finito=1;

  open ProdottiDaEliminare;

  Decrementa: LOOP
    BEGIN
      FETCH ProdottiDaEliminare INTO _nomeintegratore,_quantita;
      IF _finito=1 THEN
        LEAVE Decrementa;
            END IF;

      SET _quantitatot=_quantitatot+_quantita;
          END;
   END LOOP;
  close ProdottiDaEliminare;


    UPDATE Magazzino
     SET  CapienzaAttualeVirtuale=CapienzaAttualeVirtuale-_quantitatot
     WHERE Centro=_centro;

END $$
DELIMITER ;

DROP PROCEDURE IF EXISTS EliminaOrdiniFalliti;
DELIMITER $$
CREATE PROCEDURE EliminaOrdiniFalliti (IN _centro char(20))
BEGIN
 declare finito INT default 0;
 declare _codordine char(30) default ' ';

 declare OrdiniFalliti cursor for
 (select O.CodOrdine
  from Ordinievasi OE inner join Ordine O on OE.CodOrdine=O.CodOrdine
  where OE.stato='fallito'
                    AND
       O.centro=_centro);


 declare continue handler for not found set finito=1;

 IF NOT EXISTS (select *
            from OrdiniEvasi OE inner join Ordine O on OE.CodOrdine=O.CodOrdine
            where OE.stato='fallito'
                                              AND
                                             O.centro=_centro) THEN
   SIGNAL SQLSTATE '45000'
   SET MESSAGE_TEXT ="Nessun ordine fallito";
END IF;
```

```
 open OrdiniFalliti;

 Elimina: LOOP
   BEGIN
     FETCH OrdiniFalliti INTO _codordine;
     IF finito=1 THEN
       LEAVE Elimina;
             END IF;
     CALL DecrementaMagazzino(_centro,_codordine);
     delete from ordine where codordine=_codordine;
   END;
   END LOOP;

close OrdiniFalliti;

END $$

DELIMITER ;
```

Chi controlla che gli ordini sono falliti?

```
DROP EVENT IF EXISTS OrdineFallito;
DELIMITER $$
CREATE EVENT OrdineFallito

ON SCHEDULE EVERY 1 MINUTE
STARTS '2017-11-17 00:00:00'
DO
 BEGIN
   declare _finito INT default 0;
   declare _dataconsegna date ;
   declare _codordine char(30) default ' ';

   declare ControllaOrdini cursor for
   (select CodOrdine,dataconsegnapreferita
    from ordinievasi);

    declare continue handler for not found set _finito=1;

    open ControllaOrdini;

    SCAN: LOOP
      BEGIN
        FETCH ControllaOrdini INTO _codordine,_dataconsegna;
        IF _finito=1 THEN
          LEAVE SCAN;
                    END IF ;
        IF _dataconsegna<current_date THEN
          UPDATE Ordinievasi
          SET stato='fallito'
          WHERE codordine=_codordine;
                    END IF;
              END ;
      END LOOP;

      close ControllaOrdini;
END $$
DELIMITER ;
```

ANALYTICS SULLE VENDITE SUGLI INTEGRATORI
L'AggiornaReport permette di mettere in evidenza grazie al Log_VenditeIntegratori(aggiornato mediante la procedure delle vendite) per ogni centro :
-la quantità venduta di ogni tipo di integratore (la tipologia serve per possibili raggruppamenti per tipologia di integratore , ad esempio per bevande e non solo per nome)
-Il guadagno totale ricavato dalla vendita di quei singoli integratori , sconti di rank inclusi.
-il lotto piu vicino alla scadenza (in caso di pari merito più lotti concatenati nella stringa divisi da un trattino alto) così che possano essere tempestivamente venduti.
-giorni alla scadenza di quei lotti.
Per gli ultimi due punti utilizza un'altra stored procedure : la CalcolaScadenzaPiuProssima .

```
CREATE TABLE Log_VenditeIntegratori(
codvendita char(25) not null,
integratore char(60) not null,
quantita INT not null,
guadagno DOUBLE not null,
centro char(20) not null,
PRIMARY KEY(codvendita))
Engine=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE ReportIntegratori(
Centro char(20) not null,
Tipologia char(25) not null,
Integratore char(60) not null,
QuantitaVenduta INT ,
GuadagnoTotale DOUBLE,
LottoPiuVicinoAllaScadenza CHAR(200) not null,
GiorniAllaScadenza INT ,
PRIMARY KEY(Centro,Integratore))
Engine=InnoDB DEFAULT CHARSET=latin1;
```

```
DROP PROCEDURE IF EXISTS CalcoloScadenzaPiuProssima ;
DELIMITER $$
CREATE PROCEDURE CalcoloScadenzaPiuProssima (IN _Centro CHAR(20),IN _integratore CHAR(30),
                                OUT _Lotto CHAR(200) , OUT _Giorni INT)

BEGIN
    declare _finito INT default 0;
    declare _codprodotto char(30) default ' ';
    declare _datascadenza DATE;
    declare _magazzino char(20) default ' ';


     declare LottiScadenza cursor for
     (SELECT M1.codprodotto,M1.datascadenza
     FROM MerceMagazzino M1
     WHERE  NOT EXISTS (select *
                                  from MerceMagazzino M2
                                   where M2.DataScadenza<M1.DataScadenza
                                  AND magazzino=_magazzino
```

```
                                    AND integratore=_integratore)
            AND magazzino=_magazzino
           AND integratore=_integratore);

    declare continue handler for not found set _finito=1;

    set _Lotto='';
    set _Giorni=0;

    set _magazzino=(select codmagazzino
                from magazzino
                where centro=_centro);

            open LottiScadenza ;

    Concatena : LOOP
        BEGIN
         FETCH LottiScadenza INTO _codprodotto,_datascadenza;
         IF _finito=1 THEN
          LEAVE Concatena;
                                    END IF;
         SET _Lotto=CONCAT(_Lotto,' -  ',_codprodotto);
         SET _Giorni=DATEDIFF(_datascadenza,current_date);
                                   END ;
        END LOOP;

    close LottiScadenza;

END $$
DELIMITER ;
DROP PROCEDURE IF EXISTS AggiornaReport ;
DELIMITER $$
CREATE PROCEDURE AggiornaReport (IN _Centro CHAR(20))
BEGIN
    declare _finito INT default 0;
    declare _tipologia CHAR(25) default ' ';
    declare _integratore CHAR(30) default ' ';
    declare _quantita INT default 0;
    declare _magazzino char(25) default ' ';
    declare _guadagnototale DOUBLE default 0.00;


    declare EsaminaVendite cursor for
    (select distinct integratore
     from inventariomagazzino
     where magazzino=_magazzino
     order by integratore);

    declare continue handler for not found set _finito=1;

    set _magazzino=(select codmagazzino
                            from magazzino
                        where centro=_centro);

    set @LottiScadenza='';
    set @giorniallascadenza=0;
```

```
    open EsaminaVendite;

    CreaReport : LOOP
    BEGIN
      FETCH EsaminaVendite INTO _integratore;
      IF _finito=1 THEN
        LEAVE CreaReport;
                        END IF;
      set _quantita=(select SUM(quantita)
                  from log_venditeintegratori
                  where integratore=_integratore
                      AND
                      centro=_centro);

      if _quantita IS NULL THEN
        set _quantita=0;
                        end if;

      set _guadagnototale=(select SUM(guadagno)
                        from log_venditeintegratori
                        where integratore=_integratore
                            AND
                            centro=_centro);

      set _tipologia=(select tipologia
                    from integratore
                    where NomeCommerciale=_integratore);
                        CALL
CalcoloScadenzaPiuProssima(_centro,_integratore,@LottiScadenza,@giorniallascadenza);
      INSERT INTO ReportIntegratori
      VALUES
(_centro,_tipologia,_integratore,_quantita,_guadagnototale,@LottiScadenza,@giorniallascadenza);
                END ;
    END LOOP ;

    close EsaminaVendite ;

END $$
DELIMITER ;
```

Come rendere possibile un reporting mensile che parta in automatico ?

```
DROP EVENT IF EXISTS DeferredRefresh;
DELIMITER $$
CREATE EVENT DeferredRefresh
ON SCHEDULE EVERY 1 MONTH STARTS '2017-11-19 00:00:00'
DO
  BEGIN
     declare _finito INT default 0;
    declare _centro CHAR(20) default ' ';

    declare CentriFitness cursor for
    (select codcentro
     from centro);
```

```
    declare continue handler for not found set _finito=1;

    open CentriFitness;

    TRUNCATE TABLE reportintegratori;

    Aggiorna : LOOP
      BEGIN
        FETCH CentriFitness INTO _centro;
        IF _finito=1 THEN
          LEAVE Aggiorna;
                        END IF;
        CALL AggiornaReport(_centro);
              END;
      END LOOP;

      close CentriFitness;

   TRUNCATE TABLE Log_VenditeIntegratori;


END $$
DELIMITER ;
```

AreaSocial

Tabelle

```
CREATE TABLE IF NOT EXISTS ProfiloSocial (
Username char(60) not null,
Pass_word char(32) not null,
Proprietario char(16) not null,
NumeroStelleTotali int default 0,
NumeroPostPubblicati int DEFAULT 0,
SfideVinte int DEFAULT 0,
SfidePartecipate int DEFAULT 0,
Popolarita char(20) DEFAULT 'Sconosciuto',
PRIMARY KEY (Username),
UNIQUE(Proprietario)
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi:Username,Pass_word,Proprietario,NumeroStelleTotali,NumeroPostPubblicati,SfideVinte,Popolarita
Chiave primaria :Username
Vincoli di integrità referenziale :- Vincolo di unique, perché si suppone che il proprietario del profilo sia unico
Dipendenze funzionali Username → Pass_word,Proprietario,NumeroStelleTotali,NumeroPostPubblicati,SfideVinte,Popolarita.
Ridondanze : NumeroStelleTotali,NumeroPostPubblicati,SfideVinte.
Note :
L'idea di avere tutte queste ridondanze è proprio quella di ricreare le tipiche statistiche che sono visibili nei principali social ma anche app in generale .

Le stelle sono connesse ai giudizi , un maggior numero di stelle vuol dire un apprezzamento più alto dagli altri amici del social.
La popolarita è connessa sia alle stelle sia alla partecipazione nel mondo social .

```
CREATE TABLE IF NOT EXISTS AreaForum(
NomeArea char (50) not null,
DataUltimoPost date,
UtenteUltimoPost char(60),
UltimoPost char(30),
PRIMARY KEY (NomeArea),
CONSTRAINT UltimoPostUtente
FOREIGN KEY (UtenteUltimoPost)
REFERENCES ProfiloSocial(Username)
ON UPDATE CASCADE
ON DELETE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: NomeArea, DataUltimoPost,UtenteUltimoPost,UltimoPost
Chiave primaria: NomeArea
Vincoli di integrità referenziale : UltimoPostUtente
Dipendenze funzionali : Nome Area → DataUltimoPOst,UtenteUltimoPost,UltimoPost;
Ridondanze : DataUltimoPost,UtenteUltimoPost,UltimoPost
Note: -

```
CREATE TABLE IF NOT EXISTS PostPrincipale(
CodPost char(30) not null,
TitoloPost char(100) not null,
TimestampPubblicazione Timestamp not null,
Username char(60) not null,
Testo text not null,
StringaIndirizzoWeb char(120) default ' ',
AreaForum char(50) not null,
PRIMARY KEY(CodPost),
CONSTRAINT AutorePrincipale
FOREIGN KEY (Username)
REFERENCES ProfiloSocial(Username)
ON DELETE CASCADE
ON UPDATE CASCADE,
CONSTRAINT AreaAppartenenzaPrincipale
FOREIGN KEY (AreaForum)
REFERENCES AreaForum(NomeArea)
ON DELETE CASCADE
ON UPDATE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: Codpost,Titolopost,TimestampPubblicazione,Username,Testo,StringaIndirizzoWeb,AreaForum
Chiave primaria: Codpost
Vincoli di integrità referenziale : AutorePrincipale,AreaAppartenenzaPrincipale
Dipendenze funzionali : Codpost →
Titolopost,TimestampPubblicazione,Username,Testo,StringaIndirizzoWeb,AreaForum;

```
CREATE TABLE IF NOT EXISTS PostRisposta(
CodPost char(30) not null,
TitoloPost char(100) not null,
```

```
TimestampPubblicazione timestamp not null,
Username char(60) not null,
Testo text not null,
StringaIndirizzoWeb char(120),
NumeroStelleTotali int DEFAULT 0,
PostPrincipale char(30) not null,
PRIMARY KEY(CodPost),
CONSTRAINT AutoreRisposta
FOREIGN KEY (Username)
REFERENCES ProfiloSocial(Username)
ON DELETE CASCADE
ON UPDATE CASCADE,
CONSTRAINT Risposta
FOREIGN KEY (PostPrincipale)
REFERENCES PostPrincipale(CodPost)
ON DELETE CASCADE
ON UPDATE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi:
Codpost,Titolopost,TimestampPubblicazione,Username,Testo,StringaIndirizzoWeb,NumeroStelleTotali,Post Principale
Chiave primaria: Codpost
Vincoli di integrità referenziale : AutoreRisposta,Risposta
Dipendenze funzionali : (Codpost → Titolopost,TimestampPubblicazione,Username,Testo,StringaIndirizzoWeb,NumeroStelleTotali,PostPrincipal e);
Ridondanze : NumeroStelleTotali
Note:-

```
CREATE TABLE IF NOT EXISTS Giudizio (
Username char(60) not null,
Codpost char(30)  not null,
VotoStelle int default 0,
PRIMARY KEY(Username,codpost),
CONSTRAINT UsernameEsistente
FOREIGN KEY (username)
REFERENCES Profilosocial(username)
ON DELETE NO ACTION
ON UPDATE CASCADE,
CONSTRAINT PostEsistente
FOREIGN KEY (codpost)
REFERENCES postrisposta(codpost)
ON DELETE CASCADE
ON UPDATE CASCADE
)Engine=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: Username,Codpost,VotoStelle
Chiave primaria:Username,Codpost
Vincoli di integrità referenziale : UsernameEsistente,PostEsistente
Dipendenze funzionali : (Username,Codpost→ VotoStelle);
NOTE :-

```
CREATE TABLE IF NOT EXISTS RichiestaAmicizia(
UtenteRichiedente char(60) not null,
UtenteDestinatario char(60) not null,
```

```
Stato char(15) not null,
PRIMARY KEY(UtenteRichiedente,UtenteDestinatario),
CONSTRAINT InvioRichiesta
FOREIGN KEY(UtenteRichiedente)
REFERENCES ProfiloSocial(Username)
ON DELETE CASCADE
ON UPDATE CASCADE,
CONSTRAINT RiceveRichiesta
FOREIGN KEY(UtenteDestinatario)
REFERENCES ProfiloSocial(Username)
ON DELETE CASCADE
ON UPDATE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: UtenteRichiedente,UtenteDestinatario,Stato
Chiave primaria:UtenteRichiedente,UtenteDestinatario
Vincoli di integrità referenziale : InvioRichiesta,RiceveRichiesta.
Dipendenze funzionali : (UtenteRichiedente,UtenteDestinatario→ Stato);
NOTE :
Quando viene inserita la richiesta di amicizia ha stato : "da confermare".
Una procedure cambia lo stato da quello a "accettata" o a "rifiutata" a seconda dell'azione che si vuole realizzare (in seguito a questo UPDATE partirà un trigger).

```
CREATE TABLE IF NOT EXISTS Amicizia(
Utente1 char(60) not null,
Utente2 char(60) not null,
DataInizioAmicizia date not null,
PRIMARY KEY(Utente1,Utente2,DataInizioAmicizia),
CONSTRAINT Amicizia1
FOREIGN KEY(Utente1)
REFERENCES ProfiloSocial(Username)
ON DELETE CASCADE
ON UPDATE CASCADE,
CONSTRAINT Amicizia2
FOREIGN KEY(Utente2)
REFERENCES ProfiloSocial(Username)
ON DELETE CASCADE
ON UPDATE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi : Utente1,Utente2,DataInizioAmicizia.
Chiave primaria : Utente1,Utente2,DataInizioAmicizia
Vincoli di integrità referenziale :Amicizia1,Amicizia2
Dipendenze funzionali : tutti attributi chiave

```
CREATE TABLE IF NOT EXISTS Interesse (
Username char(60) not null,
Interesse char(50) not null,
PRIMARY KEY(Username,Interesse),
CONSTRAINT InteresseUtente
FOREIGN KEY (Username)
REFERENCES ProfiloSocial(Username)
ON DELETE CASCADE
ON UPDATE CASCADE
```

```
)Engine=InnoDB DEFAULT CHARSET=latin1;
```
Attributi : Username,Interesse
Chiave primaria : Username,Interesse
Vincoli di integrità referenziale : InteresseUtente
Dipendenze funzionali : tutti attributi chiave

```
CREATE TABLE IF NOT EXISTS Cerchia(
CodCerchia char(20) not null,
NomeCerchia char(100) not null,
Utente char(60) not null,
Interesse1 char(50) default null,
Interesse2 char(50) default null,
Interesse3 char(50) default null,
PRIMARY KEY(CodCerchia),
CONSTRAINT AmministratoreCerchia
FOREIGN KEY(Utente)
REFERENCES ProfiloSocial(Username)
ON UPDATE CASCADE
ON DELETE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi : CodCerchia,NomeCerchia,Utente,Interesse1,Interesse2,Interesse3
Chiave primaria : CodCerchia
Vincoli di integrità referenziale : AmministratoreCerchia
Dipendenze funzionali : CodCerchia→NomeCerchia,Utente,Interesse1,Interesse2,Interesse3
Note: -

```
CREATE TABLE IF NOT EXISTS Consigliati (
UtenteConsigliato char(20) not null,
UtenteCerchia char(20) not null,
Cerchia char(20) not null,
NumeroInteressiInComune int default 0,
PRIMARY KEY(Cerchia,UtenteConsigliato),
CONSTRAINT CerchiaRiferimento
FOREIGN KEY (Cerchia)
REFERENCES Cerchia(CodCerchia)
ON DELETE CASCADE
ON UPDATE CASCADE,
CONSTRAINT UtenteCerchiaRiferimento
FOREIGN KEY (UtenteCerchia)
REFERENCES ProfiloSocial(Username)
ON DELETE CASCADE
ON UPDATE CASCADE,
CONSTRAINT UtenteConsigliatoRiferimento
FOREIGN KEY (UtenteConsigliato)
REFERENCES ProfiloSocial(Username)
ON DELETE CASCADE
ON UPDATE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi : UtenteConsigliato,UtenteCerchia,Cerchia,NumeroInteressiInComune
Chiave primaria: UtenteConsigliato,UtenteCerchia,Cerchia

Vincoli di integrità referenziale :
CerchiaRiferimento,UtenteConsigliatoRiferimento,UtenteCerchiaRiferimento
Dipendenze funzionali : (UtenteCerchia,UtenteConsigliato,Cerchia→NumeroInteressiInComune
Ridondanze : NumeroInteressiInComune(fondamentali per i consigliati)
Note : -

```
CREATE TABLE IF NOT EXISTS ComposizioneCerchia(
Cerchia char(50) not null,
Utente char(50) not null,
PRIMARY KEY (Cerchia,Utente),
CONSTRAINT CerchiaPartecipazione
FOREIGN KEY (Cerchia)
REFERENCES Cerchia(CodCerchia)
ON DELETE CASCADE
ON UPDATE NO ACTION,
CONSTRAINT PartecipazioneCerchia
FOREIGN KEY(Utente)
REFERENCES ProfiloSocial(Username)
ON DELETE CASCADE
ON UPDATE NO ACTION
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi : Cerchia,Utente
Chiave primaria :Cerchia,Utente
Vincoli di integrità referenziale : CerchiaPartecipazione,PartecipazioneCerchia
Dipendenze funzionali :attributi tutti chiave
Note: -

Richieste d'amicizia

L'inserimento di una richiesta di amicizia è una semplice operazione di INSERT che fa scattare un **BEFORE Trigger** che valuta che il valore assegnato a stato sia uguale a "Da confermare".

ControllaStato1

```
drop trigger if exists ControllaStato1;
delimiter $$
create trigger ControllaStato1
before insert on RichiestaAmicizia
for each row
begin
  if new.stato <> 'Da confermare' then
    signal sqlstate '45000'
    set message_text = 'Inserimento non valido : il valore assegnato a Stato non è valido' ;
  end if;
end $$
delimiter ;
```

La stored procedure prende come argomento tre parametri :
-il primo è l'utente che invia la richiesta
-il secondo è colui che la riceve
-il terzo è una stringa che determina se la richiesta d'amicizia debba andare a buon fine o meno.

“Accettata” in caso affermativo, “Rifiutata” in caso contrario. Un valore diverso genererebbe un segnale d’errore.

```
drop procedure if exists RispondiRichiestaAmicizia;
delimiter $$
create procedure RispondiRichiestaAmicizia(in UtenteA char(60), in UtenteB char(60), in esito char(20))
begin
  update richiestaamicizia
  set stato=esito
  where UtenteRichiedente=UtenteA
        and
        Utentedestinatario=UtenteB;
end $$
delimiter ;
```

E il segnale d’errore ??

Anche qui entra in gioco un **BEFORE trigger** .

ControllaStato2

```
drop trigger if exists  ControllaStato2;
delimiter $$
create trigger ControllaStato2
before update on RichiestaAmicizia
for each row
begin
  if old.stato = 'Da confermare'
    and (new.stato not in ('Rifiutata','Confermata')) then
      signal sqlstate '45000'
      set message_text='La richiesta va confermata o rifiutata,altri valori non sono ammessi';
  end if;
end $$
delimiter ;
```

Una volta effettuato l’update , come reazione a catena attraverso **un’AFTER Trigger** viene fatto l’inserimento in Amicizia in caso il nuovo stato assunto sia “Accettata” altrimenti il database tiene solo traccia del rifiuto della richiesta senza effettuare alcuna insert.

RichiestaConfermata

```
drop trigger if exists RichiestaConfermata;
delimiter $$
create trigger RichiestaConfermata
after update on RichiestaAmicizia
for each row
begin
  if old.stato='Da confermare'
    and new.stato='Confermata' then
            insert into Amicizia
      values(new.UtenteRichiedente,new.UtenteDestinatario,current_date);
  end if;
end $$
delimiter ;
```

Post

L’inserimento di un post genera dei semplici effetti collaterali gestiti tramite AFTER TRIGGER .
In primis vengono settati i tre attributi ridondanti dell’area forum , cioè la data dell’ultimo post, chi lo ha postato e l’ultimo post stesso .

AggiornaAreaForum1

```
DROP TRIGGER IF EXISTS AggiornaAreaForum1;
DELIMITER $$
CREATE TRIGGER AggiornaAreaForum1
AFTER INSERT ON postprincipale
FOR EACH ROW
BEGIN
  UPDATE AreaForum
  SET    Dataultimopost=current_date,
        UtenteUltimoPost=new.username,
        UltimoPost=new.Codpost
  WHERE NomeArea=new.areaforum;
END $$
delimiter ;
```

In secondo luogo l’inserimento di un post principale così come un post di risposta fanno incrementare il valore dell’attributo ridondante **NumeroPostPubblicati** nel profilo social. Il post di risposta non attiva invece il trigger precedente : l’idea è che il forum tenga traccia dell’ultimo “Topic” , mentre il post di risposta è considerato più come una specie di commento. Essendo tue tabelle diverse si necessitano due trigger che svolgono la medesima azione ma su tabelle diverse.

AggiornaNumeroPostPubblicati1

```
drop trigger if exists AggiornaNumeroPostPubblicati1;
delimiter $$
create trigger AggiornaNumeroPostPubblicati1
after insert on postprincipale
for each row
begin
  update profilosocial
  set NumeroPostPubblicati=NumeroPostPubblicati + 1
  where username=new.username;
end $$
delimiter ;
```

AggiornaNumeroPostPubblicati2

```
drop trigger if exists AggiornaNumeroPostPubblicati2;
delimiter $$
create trigger AggiornaNumeroPostPubblicati2
after insert on postrisposta
for each row
begin
  update profilosocial
  set NumeroPostPubblicati=NumeroPostPubblicati+ 1
  where username=new.username;
end $$
delimiter ;
```

Giudizi

Oltre a permettere di effettuare richieste d’amicizia, creazione di post principali e post di risposta l’implementazione del database permette di mettere un altro fondamentale tassello nelle fondamenta dell’apparato social di questa palestra, ovveri i giudizi.

Il funzionamento di questi è simile a quello dei like su Facebook : dato un argomento , si dibatte su di esso e gli utenti esprimono preferenze o meno sui vari commenti .
Tradotto per questo database i giudizi non sono altro che l'opinione dei clienti sui post di risposta di altri utenti.
La scala dei valori dei giudizi viene espressa mediante delle "stelle" con un range che va da 0 a 5 stelle.
Un trigger controlla che gli utenti inseriscano dei valori corretti.

ControllaInserimentoGiudizio

```
DROP TRIGGER IF EXISTS ControllaInserimentoGiudizio;
delimiter $$
CREATE TRIGGER ControllaInserimentoGiudizio
BEFORE INSERT ON Giudizio
FOR EACH ROW
BEGIN
  IF new.VotoStelle <0 OR new.VotoStelle >5 THEN
          signal sqlstate '45000'
   set message_text="Inserimento giudizio non valido: il voto assegnato non è valido" ;
   END IF ;
END $$
delimiter ;
```

Essi sono fondamentali nel determinare il grado di popolarità "virtuale" di un utente che mano a mano che guadagna stelle (e quindi diciamo consensi) non fa altro che sbloccare degli "achievements" , dei **gradi di popolarità** .
Per sbloccare gradi l'utente dovrà essere più attivo possibile nel social , pubblicando diversi post , ma allo stesso tempo dovrà cercare di accaparrarsi ogni volta il maggior numero di stelle possibile.
Gradi sbloccati in base al numero di stelle:
- **"Nuovo Arrivato"** per chi supera le 30 stelle
-"Conosciuto" per chi supera le 200 stelle
-"Popolare" per chi supera le 500 stelle
-"Vip" per chi supera le 1000 stelle

Il tutto anch'esso gestito da un trigger :

```
DROP TRIGGER IF EXISTS AggiornaNumeroStelle;
delimiter $$
CREATE TRIGGER AggiornaNumeroStelle
AFTER INSERT ON Giudizio
FOR EACH ROW
BEGIN
   set @StellePrimaDelGiudizio=(select PS.NumeroStelleTotali
                                from profilosocial PS
                                inner join
                                postrisposta PR
                                on
                                PS.username=PR.username
                               where PR.codpost=new.codpost);
   update profilosocial PS
        inner join
        postrisposta PR
        on
        PS.username=PR.username
          set PS.NumeroStelleTotali=PS.NumeroStelleTotali+new.VotoStelle,
      PR.NumeroStelleTotali=PR.NumeroStelleTotali+new.VotoStelle
   where codpost=new.codpost;
  CASE
   WHEN @stelleprimadelgiudizio<30 and
       (@stelleprimadelgiudizio + new.votostelle) >=30 then
```

```
        begin
    update profilosocial PS
         inner join
         postrisposta PR on
         PS.username=PR.username
    set PS.popolarita='Nuovoarrivato'
    where PR.codpost=new.codpost;
        end;
        WHEN @stelleprimadelgiudizio<200 and
    (@stelleprimadelgiudizio + new.votostelle) >=200 then
        begin
    update profilosocial PS
         inner join
         postrisposta PR on
         PS.username=PR.username
    set PS.popolarita='Conosciuto'
    where PR.codpost=new.codpost;
        end;
        WHEN @stelleprimadelgiudizio<500 and
    (@stelleprimadelgiudizio + new.votostelle) >=500 then
        begin
                update profilosocial PS
         inner join
         postrisposta PR on
         PS.username=PR.username
    set PS.popolarita='Popolare'
    where PR.codpost=new.codpost;
        end;
        WHEN @stelleprimadelgiudizio<1000 and
   (@stelleprimadelgiudizio + new.votostelle) >=1000 then
        begin
    update profilosocial PS
         inner join
         postrisposta PR on
         PS.username=PR.username
    set PS.popolarita='Vip'
    where PR.codpost=new.codpost;
        end;
  ELSE BEGIN END;
  END CASE ;

END $$
delimiter ;
```

Interessi e Consigliati

Il sistema dei consigliati permette agli utenti di vedere quali utenti sono più affini per interessi comuni alle proprie cerchie in quanto la creazione di una Cerchia avviene sulla base degli interessi del creatore della cerchia : esso infatti non può creare una cerchia con interessi non congrui con i suoi.
Quindi quando un utente aggiunge un amico o leva l'amicizia, o aggiunge un interesse o lo leva, il database si deve adattare ai cambiamenti cambiando la lista di amici consigliata all'utente da aggiungere alla cerchia.
Partendo da queste regole l'implementazione è la seguente:

ControllaCerchia (interessi cerchia compatibili con quelli del creatore cerchia)

```
DROP TRIGGER IF EXISTS ControllaCerchia;
delimiter $$
CREATE TRIGGER ControllaCerchia
BEFORE INSERT ON Cerchia
FOR EACH ROW
```

```
BEGIN
  IF new.interesse1 not in (select I.Interesse
                            from interesse
                            where I.username=new.utente) THEN
  signal sqlstate '45000'
  set message_text="Creazione cerchia negata: il primo interesse deve essere tra gli interessi
  del creatore della cerchia!";
  end if;
  IF new.interesse2 not in (select I.Interesse
                             from interesse I
                            where I.username=new.utente) THEN
  signal sqlstate '45000'
  set message_text="Creazione cerchia negata: il secondo interesse deve essere tra gli interessi
  del creatore della cerchia!";
  end if;
  IF new.interesse3 not in (select I.Interesse
                            from interesse I
                             where I.username=new.utente) THEN
  signal sqlstate '45000'
  set message_text="Creazione cerchia negata: il terzo interesse deve essere tra gli interessi
del creatore della cerchia!";
  end if;
END $$
delimiter ;
```

Suggerimento Cerchie (4 trigger) :
-2 su Amicizia , uno per quando si stringe amicizia , che quindi suggerisce le cerchie interessanti del nuovo amico e un altro che invece dopo aver tolto l'amicizia con l'utente ne leva anche le cerchie precedentemente suggerite.
-2 su Interesse , in quanto gli interessi di un utente potendo evolversi nel tempo (ovvero si possono acquisire nuovi interessi così come perderne) generano un cambiamento anche nelle cerchie consigliate dal sistema.
Il tutto viene realizzato con dei LOOP gestiti mediante CURSOR e HANDLER.
I controlli si appoggiano su una funzione che presi in ingresso un utente e una cerchia ne determina il numero di interessi in comune . La funzione è non deterministica proprio perché gli interessi di un utente possono variare nel tempo.La funzione estrae uno a uno i record di Interesse relativi all'utente e li confronta con i tre interessi della Cerchia , se almeno uno coincide allora si incrementa una variabile che verrà poi ritornata dalla funzione e così si va avanti finchè non si sono estratti tutti i record target di Interesse .

InteressiInComuneCerchia

```
DROP FUNCTION IF EXISTS InteressiInComuneCerchia;
delimiter $$
CREATE FUNCTION InteressiInComuneCerchia (_utente char(60), _cerchia char(20))
RETURNS INT not deterministic
BEGIN
  declare _containteressi int default 0;
  declare _interesse char(50) default ' ';
  declare _finito int default 0;
  declare _interessecerchia1 char(50) default ' ';
  declare _interessecerchia2 char(50) default ' ';
  declare _interessecerchia3 char(50) default ' ';

  declare InteressiTarget cursor for
  (select I.interesse
   from Interesse I
   where I.username=_utente);
```

```
  declare continue handler for not found set _finito=1;

       select C.interesse1,C.interesse2,C.interesse3  into    _interessecerchia1,_interessecerchia2,_interessecerchia3
       from Cerchia C
       where C.codcerchia=_cerchia);

  open InteressiTarget;

  Conta : LOOP
   fetch InteressiTarget into _interesse;
   if _finito=1 then
    leave Conta;
          end if;
   IF (_interessecerchia1=_interesse)
     OR
     (_interessecerchia2=_interesse)
     OR
     (_interessecerchia3=_interesse) THEN

   set _containteressi=_containteressi+1;
   end if;
   END LOOP;
   return(_containteressi);
END $$
delimiter ;
```

GestisciConsigliati

```
DROP TRIGGER IF EXISTS GestisciConsigliati1;
delimiter $$
CREATE TRIGGER GestisciConsigliati1
AFTER INSERT ON Amicizia
FOR EACH ROW
BEGIN
  CALL ConsigliaCerchia ( new.utente1,new.utente2);
  CALL ConsigliaCerchia (new.utente2,new.utente1);
END $$
DELIMITER ;



DROP PROCEDURE IF EXISTS ConsigliaCerchia;
DELIMITER $$
CREATE PROCEDURE ConsigliaCerchia ( IN utenteA CHAR(60),IN utenteB CHAR(60))
BEGIN
  declare CerchiaTarget char(50) default ' ';
  declare finito int default 0;

  declare CerchieTarget cursor for
  (select Distinct(C.Codcerchia)
   from Cerchia C
       inner join
       Interesse I
       on
       (C.Interesse1=I.Interesse
        OR
        C.Interesse2=I.Interesse
```

```
        OR
        C.Interesse3=I.Interesse)

   where I.username=utenteB
        and
        C.utente=utenteA);

   declare continue handler for not found set finito=1;

   open CerchieTarget ;

   InserisciConsigliati:  LOOP
     fetch CerchieTarget into CerchiaTarget;
     if finito=1 then
       leave inserisciconsigliati;
             end if;
     insert into Consigliati
     values(utenteB,Cerchiatarget,InteressiInComuneCerchia(utenteB,cerchiatarget));
     end loop;

   close CerchieTarget;

END $$
delimiter ;


DROP TRIGGER IF EXISTS GestisciConsigliati2;
delimiter $$
CREATE TRIGGER GestisciConsigliati2
AFTER INSERT ON Interesse
FOR EACH ROW
BEGIN
  declare utente char(20) default ' ';
  declare cerchiatarget char(50) default ' ';
  declare finito int default 0;

  declare utenticerchietarget cursor for
  (select C.utente,C.codcerchia
   from Cerchia C
        inner join
        Amicizia A
        on
        C.utente=A.utente2
   where A.utente1=new.username
        and
                    (C.interesse1=new.interesse
                     OR
        C.interesse2=new.interesse
        OR
        C.interesse3=new.interesse));

  declare continue handler for not found set finito=1;

  open utenticerchietarget;

  InserisciConsigliati : LOOP
    fetch utenticerchietarget into utente,cerchiatarget;
    if finito=1 then
      leave inserisciconsigliati;
```

```
            end if;
    insert into Consigliati
    values(new.username,utente,cerchiatarget,InteressiInComuneCerchia(new.username,cerchiatarget));
    end loop;
    close utenticerchietarget;
end $$
delimiter ;


DROP TRIGGER IF EXISTS GestisciConsigliati3;
DELIMITER $$
CREATE TRIGGER GestisciConsigliati3
AFTER DELETE ON Amicizia
FOR EACH ROW
BEGIN

 CALL ConsigliaCerchia2 (old.Utente1,old.Utente2);
 CALL ConsigliaCerchia2 (old.Utente2,old.Utente1);

END $$
DELIMITER ;

DROP PROCEDURE IF EXISTS ConsigliaCerchia2 ;
delimiter $$
CREATE PROCEDURE ConsigliaCerchia2 (IN _Utente1 VARCHAR(60), IN _Utente2 VARCHAR(60))
BEGIN
  declare CerchiaTarget char(50) default ' ';
  declare finito int default 0;

  declare CerchieTarget cursor for
  (select C.Codcerchia
   from Cerchia C
        inner join
        Interesse I
        on
        (C.Interesse1=I.interesse
         OR
         C.Interesse2=I.interesse
         OR
         C.Interesse3=I.interesse)
   where I.username=_utente2
         and
         C.utente=_utente1);

   declare continue handler for not found set finito=1;

   open CerchieTarget ;

   RimuoviConsigliati : LOOP
     fetch CerchieTarget into CerchiaTarget;
     if finito=1 then
       leave Rimuoviconsigliati;
             end if;
     delete from  Consigliati
     where utenteconsigliato=_utente2
           and
           cerchia=cerchiatarget;
     end loop;
     close CerchieTarget;
```

```
end $$
DELIMITER ;


DROP TRIGGER IF EXISTS GestisciConsigliati4;
DELIMITER ;
CREATE TRIGGER GestisciConsigliati4
AFTER DELETE ON Interesse
FOR EACH ROW
BEGIN
  declare utente char(20) default ' ';
  declare cerchiatarget char(50) default ' ';
  declare controllo int default 0;
  declare finito int default 0;

  declare utenticerchietarget cursor for
  (select C.utente,C.codcerchia
   from Cerchia C
        inner join
        Amicizia A
        on
        C.utente=A.utente2
   where A.utente1=old.username
         and
         C.interesse1=old.interesse
         or
         C.interesse2=old.interesse
         or
         C.interesse3=old.interesse);

  declare continue handler for not found set finito=1;

  open utenticerchietarget;

  RimuoviConsigliati : LOOP
    fetch utenticerchietarget into utente,cerchiatarget;
    if finito=1 then
      leave rimuoviconsigliati;
            end if;
    set controllo=InteressiInComuneCerchia(old.username,cerchiatarget);
    IF controllo=0 then
    delete from Consigliati
    where utenteconsigliato=old.username
          and
          cerchia=cerchiatarget;
             else
    update Consigliati
    set NumeroInteressiInComune=NumeroInteressiInComune - 1
    where cerchia=cerchiatarget
          and
          utenteconsigliato=old.username;
    end if;

    end loop;
    close utenticerchietarget;
end $$
DELIMITER ;
```

Aggiungi alla cerchia !

```
DROP  PROCEDURE IF EXISTS AggiungiAllaCerchia;
DELIMITER $$
CREATE PROCEDURE AggiungiAllaCerchia (IN _Utente2 CHAR(30), IN _Cerchia CHAR(20))
BEGIN
  declare _utente1 CHAR(30) default ' ';

  set _utente1=(select utente
                  from cerchia
                 where codcerchia=_Cerchia);
 IF NOT EXISTS (select *
         from consigliati
         where utenteconsigliato=_utente2
              AND
              utentecerchia=_utente1
              AND
              cerchia=_cerchia) THEN
                    SIGNAL SQLSTATE '45000'
     SET MESSAGE_TEXT ="L'utente non può essere aggiunto alla cerchia";
 END IF;
 INSERT INTO composizionecerchia
 VALUES (_Cerchia,Utente2);
 END $$
 DELIMITER ;
```

Sfide (Area Social)

Tabelle

Sfida

```
 CREATE TABLE IF NOT EXISTS Sfida(
CodSfida char(20) not null,
TitoloSfida char(80) not null,
UtenteProponente char(60) not null,
DataLancioSfida date not null,
DataInizioSfida date not null,
DataFineSfida date not null,
ScopoDaRaggiungere char(80) not null,
ValoreScopo int not null,
SchedaAllenamento char(50) ,
SchedaAlimentazione char(50) ,
CodPost char(60) not null,
PRIMARY KEY(CodSfida),
CONSTRAINT Proponente
FOREIGN KEY(UtenteProponente)
REFERENCES ProfiloSocial(Username)
```

```
ON UPDATE CASCADE
ON DELETE NO ACTION,
CONSTRAINT CodPostSfida
FOREIGN KEY(CodPost)
REFERENCES Postprincipale(Codpost)
ON UPDATE CASCADE
ON DELETE CASCADE,
CONSTRAINT SchedaAlimentazioneSfida
FOREIGN KEY(SchedaAlimentazione)
REFERENCES SchedaAlimentazione(CodSchedaAlim)
ON UPDATE CASCADE
ON DELETE SET NULL,
CONSTRAINT SchedaAllenamentoSfida
FOREIGN KEY(SchedaAllenamento)
REFERENCES SchedaAllenamento(CodSchedaAllenamento)
ON UPDATE CASCADE
ON DELETE SET NULL
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: CodSfida, TitoloSfida, UtenteProponente, DataLancioSfida, DataInizioSfida, DataFineSfida, ScopoDaRaggiungere, ValoreScopo, SchedaAllenamento, SchedaAlimentazione, CodPost.
Chiave primaria: CodSfida
Vincoli Integrita Referenziale: Proponente, CodPostSfida, SchedaAlimentazioneSfida, SchedaAllenamentoSfida.
Dipendenze Funzionali:
CodSfida→TitoloSfida, UtenteProponente, DataLancioSfida, DataInizioSfida, DataFineSfida, ScopoDaRaggiungere, ValoreScopo, SchedaAllenamento, SchedaAlimentazione, CodPost.

Note:
La sfida contiene una scheda alimentazione e scheda allenamento di base, poi modificabili.

Sforzo

```
CREATE TABLE IF NOT EXISTS Sforzo(
CodSforzo char(20) not null,
CodSfida char(20) not null,
Username char(60) not null,
ValoreSforzo int not null,
DataSforzo date not null,
PRIMARY KEY(CodSforzo),
CONSTRAINT UserSforzo
FOREIGN KEY (Username)
REFERENCES ProfiloSocial(Username)
ON UPDATE CASCADE
ON DELETE CASCADE,
CONSTRAINT SfidaSforzo
FOREIGN KEY (CodSfida)
REFERENCES Sfida(CodSfida)
ON UPDATE CASCADE
ON DELETE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: CodSforzo,CodSfida, Username, ValoreSforzo, DataSforzo
Chiave primaria:CodSforzo.

Vincoli Integrita Referenziale: UserSforzo, SfidaSforzo.
Dipendenze Funzionali:
CodSforzo→CodSfida, Username, ValoreSforzo, DataSforzo.

Note:
Lo sforzo contiene una valutazione complessiva degli esercizi che un cliente sfolge in un giorno. Servirà poi per il calcolo del vincitore della sfida.

SfidaConclusa

```
CREATE TABLE IF NOT EXISTS SfidaConclusa(
CodSfida char(20) not null,
Username char(60) not null,
DataConclusione date not null,
VotoSfida double not null,
PRIMARY KEY(CodSfida,Username),
CONSTRAINT UserConclude
FOREIGN KEY(Username)
REFERENCES ProfiloSocial(Username)
ON UPDATE CASCADE
ON DELETE CASCADE,
CONSTRAINT SfidaConclude
FOREIGN KEY(CodSfida)
REFERENCES Sfida(CodSfida)
ON UPDATE CASCADE
ON DELETE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: CodSfida, Username, DataConclusione, VotoSfida
Chiave primaria:CodSfida, Username
Vincoli Integrita Referenziale: UserSforzo, SfidaSforzo.
Dipendenze Funzionali:
CodSfida, Username→DataConclusione, VotoSfida.
Note:
La tabella contiene tutti coloro che hanno concluso volutamente o forzatamente la sfida.

AderisciSfida

```
CREATE TABLE IF NOT EXISTS AderisciSfida(
Utente char(60) not null,
CodSfida char(20) not null,
PRIMARY KEY(Utente,CodSfida),
CONSTRAINT UtentePartecipante
FOREIGN KEY(Utente)
REFERENCES ProfiloSocial(Username)
```

```
ON DELETE CASCADE
ON UPDATE CASCADE,
CONSTRAINT AdesioneSfida
FOREIGN KEY(CodSfida)
REFERENCES Sfida(CodSfida)
ON DELETE CASCADE
ON UPDATE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: Utente, CodSfida.

Chiave primaria:Utente, CodSfida.
Vincoli Integrita Referenziale: UtentePartecipante, AdesioneSfida

Dipendenze Funzionali:
CodSfida, Utente è chiave primaria

Note:
AderisciSfida contiene colore che si sono iscritti alla sfida.

MisurazioneSfida

```
CREATE TABLE IF NOT EXISTS MisurazioneSfida (
Sfida CHAR(20) not null,
ValoreMisurazione INT not null,
Visita CHAR(20) not null,
PRIMARY KEY(Visita),
CONSTRAINT Misurazione_Sfida
FOREIGN KEY (Sfida)
REFERENCES Sfida(CodSfida)
ON DELETE CASCADE
ON UPDATE CASCADE,
CONSTRAINT VisitaMisurazione_Sfida
FOREIGN KEY (Visita)
REFERENCES Visita(CodVisita)
ON DELETE NO ACTION
ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET= latin1;
```

Attributi: Sfida, ValoreMisurazione,Visita.
Chiave primaria:Visita.
Vincoli Integrita Referenziale: Misurazione_Sfida, VisitaMisurazione_Sfida.
Dipendenze Funzionali:
Visita→ Sfida, ValoreMisurazione.

Note:
MisurazioneSfida contiene tutte le misurazioni fatte dai clienti durante la sfida, per misurare quanto sono vicini al tragurdo.

Vincitore_Sfida

```
CREATE TABLE IF NOT EXISTS Vincitore_Sfida (
Sfida CHAR(20) not null,
Vincitore CHAR(60) not null,
```

```
PRIMARY KEY(Sfida,Vincitore),
CONSTRAINT Sfida_Vinta
FOREIGN KEY(Sfida)
REFERENCES Sfida(CodSfida)
ON DELETE CASCADE
ON UPDATE NO ACTION,
CONSTRAINT Vincitore_Sfida
FOREIGN KEY(Vincitore)
REFERENCES ProfiloSocial(Username)
ON DELETE CASCADE
ON UPDATE NO ACTION
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: Sfida, Vincitore
Chiave primaria:Sfida, Vincitore
Vincoli Integrita Referenziale: Sfida_Vinta, Vincitore_Sfida.
Dipendenze Funzionali:
Sfida, Vincitore è chiave primaria.

Note:
La tabella terrà conto di tutti i vincitori delle sfide.

Log Tables

```
CREATE TABLE IF NOT EXISTS Log_Sfide (
Sfida CHAR(20) not null,
PRIMARY KEY (Sfida)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
CREATE TABLE IF NOT EXISTS Log_Conclusioni (
Sfida CHAR(20) not null,
Username CHAR(60) not null,
PRIMARY KEY (Sfida,Username)
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Note:
Le due tabelle mi serviranno per calcolare e vincitori e per forzare la fine della sfida per coloro che non l'hanno ancora conclusa.

Esercizio_Modifica

```
CREATE TABLE IF NOT EXISTS Esercizio_Modifica(
Esercizio char(20) not null,
Sfida CHAR(20) not null,
Ripetizioni int ,
NumeroSerie int,
Durata INT ,
TempodiRecupero INT not null,          /*Espresso in secondi */
GiornoScheda INT default 0,
PRIMARY KEY(Esercizio,Sfida,GiornoScheda),
```

```
CONSTRAINT EsercizioScheda_Modifica
FOREIGN KEY (Esercizio)
REFERENCES Esercizio(CodEsercizio)
ON DELETE NO ACTION
ON UPDATE NO ACTION,
CONSTRAINT EsercizioperSfida
FOREIGN KEY(Sfida)
REFERENCES Sfida(CodSfida)
ON DELETE CASCADE
ON UPDATE NO ACTION
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: Esercizio, Sfida, Ripetizioni, NumeroSerie, Durata, TempodiRecupero, GiornoScheda.
Chiave primaria:Esercizio, Sfida, GiornoScheda.
Vincoli Integrita Referenziale: EsercizioScheda_Modifica, EseciziooperSfida.
Dipendenze Funzionali:
Esercizio, Sfida, GiorniScheda→ Ripetizioni, NumeroSerie, Durata, TempodiRecupero.
Note:
All'atto di creazione della sfida, tutti gli esercizi della scheda allenamento scelta verranno inviati in questa tabella, cosicchè li si possa modificare senza problemi.

```
CREATE TABLE IF NOT EXISTS Esercizio_Modifica_Config(
Esercizio char(20) not null,
Sfida CHAR(20) not null,
GiornoScheda INT not null,
Attrezzatura CHAR(80) not null,
TipologiaConfigurazione CHAR(60) not null,
ValoreConfigurazione INT not null,
PRIMARY KEY(Esercizio,Sfida,Attrezzatura,TipologiaConfigurazione),
CONSTRAINT EsercizioScheda_ModificaConfig
FOREIGN KEY (Esercizio,Sfida,GiornoScheda)
REFERENCES Esercizio_Modifica(Esercizio,Sfida,GiornoScheda)
ON DELETE NO ACTION
ON UPDATE NO ACTION
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: Esercizio, Sfida, Attrezzatura, TipologiaConfigurazione,ValoreConfigurazione, GiornoScheda.
Chiave primaria:Esercizio, Sfida, Attrezzatura, TipologiaConfigurazione.
Vincoli Integrita Referenziale: EsercizioScheda_ModificaConfig.
Dipendenze Funzionali:
Esercizio, Sfida, Attrezzatura, TipologiaConfigurazione→ ValoreConfigurazione.

Note:
Stesso criterio della tabella precedente, a differenza dell'altra prende le configurazioni.

```
CREATE TABLE IF NOT EXISTS Modifiche_SchedaAlimentazione (
Sfida CHAR(20) not null,
ApportoCaloricoGiornaliero INT not null,
NumeroPasti INT not null,
ComposizionePasti TEXT not null,
PRIMARY KEY(Sfida),
CONSTRAINT ModificaAlimentazione_Sfida
```

```
FOREIGN KEY (Sfida)
REFERENCES Sfida(CodSfida)
ON DELETE CASCADE
ON UPDATE NO ACTION
)ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Attributi: Sfida, ApportoCaloricoGiornaliero, NumeroPasti, ComposizionePasti.
Chiave primaria:Sfida
Vincoli Integrita Referenziale: ModificaAlimentazione_Sfida
Dipendenze Funzionali:
Sfida→ ApportoCaloricoGiornaliero, NumeroPasti, ComposizionePasti.
Note:
Verrà riempita sempre nello stesso modo.

Stored Procedures

InserisciSfida()

```
DROP PROCEDURE IF EXISTS InserisciSfida;
DELIMITER $$
CREATE PROCEDURE InserisciSfida (IN _TitoloSfida VARCHAR(50), IN _Username VARCHAR(50),
                                 IN _Testo TEXT, IN _IndirizzoWeb VARCHAR(60),
                                 IN _DataLancioSfida DATE,  IN _DataInizioSfida DATE,
                                 IN _DataFineSfida DATE, IN _Scopo VARCHAR(50), IN _ValoreScopo INT,
                                 IN _SchedaAlimentazione VARCHAR (50),
                                 IN _SchedaAllenamento VARCHAR(50), IN _CodiceSfida VARCHAR(50),
                                 IN _CodPost VARCHAR(50))
 BEGIN

  IF (_Username NOT IN (SELECT Username FROM ProfiloSocial )) THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT='Username inesistente';
  END IF;

  INSERT INTO PostPrincipale (CodPost,TitoloPost,Username,Testo,StringaIndirizzoWeb,AreaForum)
   VALUES (_CodPost,_TitoloSfida,_Username,_Testo,_IndirizzoWeb,'Sfide');

  INSERT INTO Sfida (CodSfida,TitoloSfida,UtenteProponente,DataLancioSfida,DataInizioSfida,DataFineSfida,
              ScopoDaRaggiungere,ValoreScopo,SchedaAllenamento,SchedaAlimentazione,Codpost)
          VALUES
(_CodiceSfida,_TitoloSfida,_Username,_DataLancioSfida,_DataInizioSfida,_DataFineSfida,_Scopo,
        _ValoreScopo,_SchedaAllenamento,_SchedaAlimentazione,_Codpost);
  INSERT INTO AderisciSfida
  VALUES (_Username,_CodiceSfida);

  INSERT INTO Esercizio_Modifica
  SELECT
E.CodEsercizio,_CodiceSfida,E.NumeroRipetizioni,E.NumeroSerie,E.DuratainMinuti,E.TempodiRecuperoSecondi,ES.
Giorno
  FROM Esercizio E INNER JOIN EsercizioScheda ES
```

```
   ON E.CodEsercizio=ES.Esercizio
  WHERE ES.Scheda=_SchedaAllenamento;

  INSERT INTO Esercizio_Modifica_Config
  SELECT
EC.Esercizio,_CodiceSfida,EM.GiornoScheda,EC.Attrezzatura,EC.TipoConfigurazione,EC.ValoreConfigurazione
  FROM Esercizio_Configurazione EC INNER JOIN Esercizio_Modifica EM
   ON EM.Esercizio=EC.Esercizio
  WHERE EM.Sfida=_CodiceSfida;
   END $$
DELIMITER ;
```

Descrizione: La procedure permette di inserire una sfida con scheda alimentazione e scheda allenamento. Dopo aver inserito con successo la sfida e il post di creazione nell'area forum 'Sfide', inserisce dentro le tabelle modifica tutti i valori delle sfide e delle schede.

AderisciSfida()

```
DROP PROCEDURE IF EXISTS AderisciSfida;
DELIMITER $$
CREATE PROCEDURE AderisciSfida (IN _Username VARCHAR(60), IN _Sfida VARCHAR(20),IN _DataIscrizione
DATE)
BEGIN

 DECLARE ProponenteSfida VARCHAR(60) DEFAULT '';

 SELECT UtenteProponente INTO ProponenteSfida
 FROM Sfida
 WHERE CodSfida=_Sfida;

 IF (NOT EXISTS (SELECT *
                  FROM Amicizia
                  WHERE (Utente1=_Username AND Utente2=ProponenteSfida)
                  OR (Utente2=_Username AND Utente1=ProponenteSfida))) THEN
     SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT='Il proponente non è tra i tuoi amici';
 END IF;

 IF NOT EXISTS (SELECT CodSfida
                  FROM Sfida
                  WHERE _Sfida=CodSfida
                  AND _DataIscrizione>=DataLancioSfida
                  AND _DataIscrizione<DataInizioSfida) THEN

  SIGNAL SQLSTATE '45000'
  SET MESSAGE_TEXT='Tempo di iscrizione per la sfida esaurito';
 END IF;

 INSERT INTO AderisciSfida
 VALUES(_Username,_Sfida);


END $$
DELIMITER ;
```

Descrizione: La procedure permette al cliente di partecipare alle sfide solo se lui e il proponente sono amici e la sfida non è iniziata.

CalcoloSforzoGiornaliero()

```
DROP PROCEDURE IF EXISTS CalcoloSforzoGiornaliero;
DELIMITER $$
CREATE PROCEDURE CalcoloSforzoGiornaliero (IN _Username VARCHAR(50), IN _Data DATE, IN _Sfida
VARCHAR(50), IN _Sforzo VARCHAR(20))
BEGIN

 DECLARE Conteggio INT DEFAULT 0;
 DECLARE Effettivo INT DEFAULT 0;
 DECLARE EffettivoConfig INT DEFAULT 0;
 DECLARE OttimaleConfig INT DEFAULT 0;
 DECLARE Ottimale INT DEFAULT 0;
 DECLARE CodEx VARCHAR(10) DEFAULT '';
 DECLARE GiornoSchd INT DEFAULT 0;
 DECLARE Percentuale DOUBLE DEFAULT 0.00;
 DECLARE PercentualeConfig DOUBLE DEFAULT 0.00;
 DECLARE ValSforzo INT DEFAULT 0;
 DECLARE Scheda_Allenamento VARCHAR(25) DEFAULT '';
 DECLARE NumConfig INT DEFAULT 0;
 DECLARE TotConfig INT DEFAULT 0;

 DECLARE Finito BOOL DEFAULT 0;

 DECLARE EserciziSvolti CURSOR FOR
  SELECT Esercizio,(IF(Ripetizioni=0 OR Ripetizioni IS NULL ,Durata,Ripetizioni) *
                IF (NumeroSerie=0 OR NumeroSerie IS NULL,1,NumeroSerie)),GiornoScheda
  FROM EsercizioSvolto
  WHERE Cliente=(SELECT PS.Proprietario
                FROM ProfiloSocial PS
                WHERE PS.Username=_Username)
  AND _Data=DATE(IstanteInizio)
  AND Sfida=_Sfida;


 DECLARE CONTINUE HANDLER FOR NOT FOUND
  SET Finito=1;


  OPEN EserciziSvolti;

  Cursore: LOOP
   BEGIN
    FETCH EserciziSvolti INTO CodEx,Effettivo,GiornoSchd;

    IF Finito=1 THEN
      LEAVE Cursore;
           END IF;
```

```
    SELECT IF(EM.Ripetizioni=0 OR EM.Ripetizioni IS NULL,EM.Durata,EM.Ripetizioni) *
          IF (EM.NumeroSerie=0 OR EM.NumeroSerie IS NULL,1,EM.NumeroSerie) INTO Ottimale
    FROM Esercizio_Modifica EM
    WHERE Esercizio=CodEx
     AND EM.GiornoScheda=GiornoSchd
     AND EM.Sfida=_Sfida;

    SELECT SUM(ValoreConfigurazione) INTO OttimaleConfig
    FROM Esercizio_Modifica_Config
    WHERE Esercizio=CodEx
     AND Sfida=_Sfida
     AND GiornoScheda=GiornoSchd;

     SELECT SUM(ValoreConfigurazione),COUNT(*) INTO EffettivoConfig, NumConfig
     FROM EsercizioSvolto_Configurazione ESC INNER JOIN EsercizioSvolto ES
      ON ESC.Esercizio=ES.Esercizio
      AND ESC.Cliente=ES.Cliente
      AND ES.IstanteInizio=ESC.IstanteInizio
     WHERE ES.Esercizio=CodEx
      AND ES.GiornoScheda=GiornoSchd
      AND ES.Sfida=_Sfida
      AND DATE(ES.IstanteInizio)=_Data
      AND ESC.Cliente=(SELECT PS.Proprietario
                FROM ProfiloSocial PS
                WHERE PS.Username=_Username);

  SET Conteggio=Conteggio + 1;
  SET Percentuale=Percentuale+((Effettivo/Ottimale)*100);
  SET PercentualeConfig=PercentualeConfig+((EffettivoConfig/OttimaleConfig)*100);
  SET TotConfig=TotConfig + NumConfig;
  END;
  END LOOP;
   CLOSE EserciziSvolti;

  IF Conteggio=0 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT='Non hai svolto esercizi della sfida oggi';
  END IF;

   IF (TotConfig>0) THEN
    SET ValSforzo=(AssegnaSforzo(Percentuale/Conteggio) + AssegnaSforzo(PercentualeConfig/TotConfig))/2;
   ELSE
    SET ValSforzo=AssegnaSforzo(Percentuale/Conteggio);
    END IF;
  INSERT INTO Sforzo
  VALUES (_Sforzo,_Sfida,_Username,ValSforzo,_Data);

END $$
DELIMITER ;
```

Descrizione: La procedure permette ai clienti di calcolare il proprio sforzo giornaliero in proporzione agli esercizi svolti e le configurazioni usate. Prima di tutto con un CURSOR si estraggono tutte le sfide svolte e dentro il ciclo si estrapolerà anche la configurazione effetiva e si paragonerà con quella attuale. A questo punto partirà una function (descritta più avanti) che restituisce un risultato in base al rapporto esercizio svolto/ esercizio ottimale.

Set_Finito()

```
DROP PROCEDURE IF EXISTS Set_Finito;
DELIMITER $$
CREATE PROCEDURE Set_Finito()
BEGIN

  DECLARE _Sfida VARCHAR(20) DEFAULT '';
  DECLARE Finito INT DEFAULT 0;

  DECLARE CercaSfide CURSOR FOR
   SELECT CodSfida
   FROM Sfida S LEFT JOIN Vincitore_Sfida VS
    ON S.CodSfida=VS.Sfida
   WHERE VS.Sfida IS NULL
    AND DataFineSfida<=Current_date();

  DECLARE CONTINUE HANDLER FOR NOT FOUND
   SET Finito=1;

   OPEN CercaSfide;
   Scansione: LOOP
    BEGIN
    FETCH CercaSfide INTO _Sfida;

    IF (Finito=1) THEN
      LEAVE Scansione;
           END IF;

    INSERT INTO Log_Sfida
    VALUES (_Sfida);

    INSERT INTO Log_Conclusioni
    SELECT A.CodSfida,A.Utente
    FROM AderisciSfida A
    WHERE A.Utente NOT IN (SELECT S.Username
                                    FROM SfidaConclusa S
                                    WHERE S.CodSfida=_Sfida);

    END;
    END LOOP;


    END $$
    DELIMITER ;
```

Descrizione: Set_finito() è una delle 3 procedure che verranno poi richiamate dall'event ControlloFineSfide. Ha il semplice scopo di terminare forzatamente le sfide terminate a coloro che ancora non hanno concuso.

CalcoloPunteggio_Event()

```
DROP PROCEDURE IF EXISTS CalcoloPunteggio_Event;
DELIMITER $$
```

```
CREATE PROCEDURE CalcoloPunteggio_Event()
 BEGIN


  DECLARE Finito INT DEFAULT 0;
  DECLARE Utente VARCHAR(60) DEFAULT '';
  DECLARE _Sfida VARCHAR(20) DEFAULT '';
  DECLARE MediaSforzo DOUBLE DEFAULT 0;
  DECLARE VotoTempo INT DEFAULT 0;
  DECLARE VotoObiettivo INT DEFAULT 0;
  DECLARE DataF DATE;
  DECLARE DataI DATE;
  DECLARE VotoFinale DOUBLE DEFAULT 0.00;
  DECLARE MisurazioneFinale DOUBLE DEFAULT 0;

  DECLARE Conclusioni CURSOR FOR
          SELECT Username,Sfida
          FROM log_Conclusioni;


   DECLARE CONTINUE HANDLER FOR NOT FOUND
    SET Finito=1;

   OPEN Conclusioni;

   Cursore: LOOP
    BEGIN
     FETCH Conclusioni INTO Utente,_Sfida;

     IF Finito=1 THEN
       LEAVE Cursore;
            END IF;



     SELECT AVG(S.ValoreSforzo) INTO MediaSforzo
     FROM Sforzo S
     WHERE S.Username=Utente
     AND S.CodSfida=_Sfida;

     SELECT DataInizioSfida,DataFineSfida INTO DataI,DataF
     FROM Sfida
     WHERE CodSfida=_Sfida;


     SET VotoTempo=VotoGiorni(DataI,DataF,DataF);       */

    SELECT MS.ValoreMisurazione INTO MisurazioneFinale
  FROM MisurazioneSfida MS
  WHERE MS.Visita=(SELECT V.Codvisita
                                              FROM Visita V
                 WHERE V.Cliente=(SELECT PS.Proprietario
                                       FROM ProfiloSocial PS
                                       WHERE PS.Username=Utente)
                                    AND V.DataVisita=(SELECT MAX(V2.DataVisita)
                                                      FROM Visita V2
                                                      WHERE Cliente=V.Cliente))
           AND MS.Sfida=_Sfida;
```

```
    SET VotoObiettivo=VotoMisurazioni(MisurazioneFinale,(SELECT ValoreScopo FROM Sfida WHERE
CodSfida=_Sfida));

    SET VotoFinale=(MediaSforzo+VotoObiettivo+VotoTempo)/4;

    INSERT INTO SfidaConclusa
    VALUES (_Sfida,Utente,DataF,VotoFinale);   /

    END;
        END LOOP;

        CLOSE Conclusioni;


END $$
DELIMITER ;
```

Descrizione: Questa è la seconda funzione dell'event e ha il compito di calcolare il punteggio finale dei partecipanti che hanno concluso forzatamente.

Rank_Sfida()

```
DROP PROCEDURE IF EXISTS Rank_Sfida;
DELIMITER $$
CREATE PROCEDURE Rank_Sfida()
BEGIN

INSERT INTO Vincitore_Sfida
SELECT SC.CodSfida,SC.Username
FROM SfidaConclusa SC
WHERE SC.VotoSfida=(SELECT MAX(SC2.VotoSfida)
                            FROM SfidaConclusa SC2
                           WHERE SC2.CodSfida=SC.CodSfida);

END $$
DELIMITER ;
```

Descrizione: Questa è l'ultima procedure dell'event e ha il compito di calcolare i vincitori. I pari merito verranno inseriti lo stesso come vincitori. Alla fine di tutto, viene aggiornato il profilo social dei vincitori con una sfida vinta in più.

CalcoloPunteggio()

```
DROP PROCEDURE IF EXISTS CalcoloPunteggio;          /* Non la si deve scambiare con la procedure dell'event */
DELIMITER $$
CREATE PROCEDURE CalcoloPunteggio(IN _Sfida VARCHAR(20), IN _Utente VARCHAR(60),IN
_DataConclusione DATE)
```

```
 BEGIN

   DECLARE MediaSforzo DOUBLE DEFAULT 0;
   DECLARE VotoTempo INT DEFAULT 0;
   DECLARE VotoObiettivo INT DEFAULT 0;
   DECLARE DataF DATE;
   DECLARE DataI DATE;
   DECLARE VotoFinale DOUBLE DEFAULT 0.00;
   DECLARE MisurazioneFinale INT DEFAULT 0;


   SELECT AVG(S.ValoreSforzo) INTO MediaSforzo    /* CALCOLO LO SFORZO MEDIO */
   FROM Sforzo S
   WHERE S.Username=_Utente
   AND S.CodSfida=_Sfida
   GROUP BY S.Username;



   SELECT DataInizioSfida,DataFineSfida INTO DataI,DataF
   FROM Sfida
   WHERE CodSfida=_Sfida;


   SET VotoTempo=VotoGiorni(DataI,DataF,_DataConclusione);       /* Calcolo quanti giorni ha impiegato per
concludere la sfida */



   SELECT MS.ValoreMisurazione INTO MisurazioneFinale      /* PRENDO L'ULTIMA MISURAZIONE DELLA
SFIDA */
   FROM MisurazioneSfida MS
   WHERE MS.Visita=(SELECT V.Codvisita
                                          FROM Visita V
                WHERE V.Cliente=(SELECT PS.Proprietario
                                                                  FROM ProfiloSocial PS
                        WHERE PS.Username=_Utente)
                                              AND V.DataVisita=(SELECT MAX(V2.DataVisita)
                        FROM Visita V2
                        WHERE Cliente=V.Cliente))
  AND MS.Sfida=_Sfida;

   SET VotoObiettivo=VotoMisurazioni(MisurazioneFinale,(SELECT ValoreScopo FROM Sfida WHERE
CodSfida=_Sfida)); /* Calcolo la prestazione del cliente */


   SET VotoFinale=(MediaSforzo+VotoObiettivo+VotoTempo)/4;



   INSERT INTO SfidaConclusa
   VALUES (_Sfida,_Utente,_DataConclusione,VotoFinale);   /* Inserisco il tutto */



END $$
```

Descrizione: Questa stored è una versione alternativa a quella dell'event. Quando un cliente vuole concludere la sfida, viene chiamata questa procedure che stabilisce il voto finale del cliente e lo inserisce nella tabella SfidaConclusa.

Functions

```
DROP FUNCTION IF EXISTS AssegnaSforzo;
 DELIMITER $$
 CREATE FUNCTION AssegnaSforzo (_Percentuale DOUBLE)
 RETURNS INT DETERMINISTIC

 BEGIN
  DECLARE Voto INT DEFAULT 0;

 CASE
     WHEN (_Percentuale=0) THEN
       SET Voto=0;

     WHEN (_Percentuale>0 AND _Percentuale<20) THEN
       SET Voto=5;

     WHEN (_Percentuale>=20 AND _Percentuale<40) THEN
       SET Voto=4;

           WHEN (_Percentuale>=40 AND _Percentuale<60) THEN
       SET Voto=3;

           WHEN (_Percentuale>=60 AND _Percentuale<80) THEN
                   SET Voto=2;

           WHEN (_Percentuale>=80 AND _Percentuale<=100) THEN
       SET Voto=1;
  ELSE
   BEGIN
   END;

  END CASE;

  RETURN Voto;

 END $$

 DELIMITER ;
```

Descrizione: Prima funzione per il calcolo del voto.

```
 DROP FUNCTION IF EXISTS VotoGiorni;
 DELIMITER $$
 CREATE FUNCTION VotoGiorni(_DataInizio DATE, _DataFine DATE, _DataConclusione DATE)
 RETURNS INT DETERMINISTIC
 BEGIN

  DECLARE GiorniFine INT DEFAULT 0;
```

```
  DECLARE GiorniTotali INT DEFAULT 0;
  DECLARE Percentuale DOUBLE DEFAULT 0.00;
  DECLARE Voto INT DEFAULT 0;

  SET GiorniFine=DATEDIFF(_DataConclusione,_DataInizio);
  SET GiorniTotali=DATEDIFF(_DataFine,_DataInizio);
  SET Percentuale=(GiorniFine/GiorniTotali)*100;

  CASE
      WHEN (Percentuale=0) THEN
        SET Voto=0;

     WHEN (Percentuale>=0 AND Percentuale<20) THEN
        SET Voto=5;

     WHEN (Percentuale>=20 AND Percentuale<40) THEN
        SET Voto=4;

            WHEN (Percentuale>=40 AND Percentuale<60) THEN
        SET Voto=3;

            WHEN (Percentuale>=60 AND Percentuale<80) THEN
                    SET Voto=2;

            WHEN (Percentuale>=80 AND Percentuale<=100) THEN
        SET Voto=1;
    ELSE
   BEGIN
   END;
  END CASE;

  RETURN Voto;

END $$
DELIMITER ;
```

Descrizione: Seconda funzione per il calcolo del voto.

```
DROP FUNCTION IF EXISTS VotoMisurazioni;
 DELIMITER $$
 CREATE FUNCTION VotoMisurazioni(_ScopoRaggiunto INT, _ScopoOttimale INT )
 RETURNS INT DETERMINISTIC
 BEGIN

  DECLARE _Percentuale DOUBLE DEFAULT 0.00;
  DECLARE Voto INT DEFAULT 0;

  SET _Percentuale=(_ScopoRaggiunto/_ScopoOttimale)*100;

  CASE
     WHEN (_Percentuale=0) THEN
        SET Voto=0;

     WHEN (_Percentuale>0 AND _Percentuale<10) THEN
        SET Voto=1;

     WHEN (_Percentuale>=10 AND _Percentuale<20) THEN
```

```
        SET Voto=2;

            WHEN (_Percentuale>=20 AND _Percentuale<30) THEN
        SET Voto=3;

            WHEN (_Percentuale>=30 AND _Percentuale<40) THEN
                SET Voto=4;

            WHEN (_Percentuale>=40 AND _Percentuale<50) THEN
        SET Voto=5;

            WHEN (_Percentuale>=50 AND _Percentuale<60) THEN
        SET Voto=6;

      WHEN (_Percentuale>=60 AND _Percentuale<70) THEN
        SET Voto=7;

            WHEN (_Percentuale>=70 AND _Percentuale<80) THEN
        SET Voto=8;

            WHEN (_Percentuale>=80 AND _Percentuale<90) THEN
                SET Voto=9;

            WHEN (_Percentuale>=90 AND _Percentuale<=100) THEN
        SET Voto=10;
    ELSE
    BEGIN
    END;

   END CASE;

   RETURN Voto;

END $$
DELIMITER ;
```

Descrizione: Terza funzione per il calcolo del voto.

Triggers

```
DROP TRIGGER IF EXISTS AggiornaVittorie;
DELIMITER $$
CREATE TRIGGER AggiornaVittorie
AFTER INSERT ON Vincitore_Sfida
FOR EACH ROW
BEGIN

 UPDATE ProfiloSocial
```

```
 SET SfideVinte=SfideVinte+1
 WHERE Username=NEW.Vincitore;

END $$
DELIMITER ;


DROP TRIGGER IF EXISTS AggiornaVittorie;
DELIMITER $$
CREATE TRIGGER AggiornaVittorie
AFTER DELETE ON Vincitore_Sfida
FOR EACH ROW
BEGIN

 UPDATE ProfiloSocial
 SET SfideVinte=SfideVinte-1
 WHERE Username=OLD.Vincitore;

END $$
DELIMITER ;
```

Descrizione: Questi due trigger hanno la funzione di gestire la ridondanza SfideVinte della tabella ProfiloSocial. Una in insert e una in delete.

Events

```
DROP EVENT IF EXISTS ControlloFineSfide;
DELIMITER $$
CREATE EVENT ControlloFineSfide
ON SCHEDULE EVERY 1 DAY
DO
BEGIN
  CALL Set_Finito();

  CALL CalcoloPunteggio_Event();

  CALL Rank_Sfida();

  DELETE FROM Log_Conclusioni;
  DELETE FROM Log_Sfida;
END $$
DELIMITER ;
```

Descrizione: Questo è l'event di cui si è parlato sopra. Si limita a chiamare le stored e a svuotare i log alla fine del processo.