Università degli Studi di Napoli
Parthenope

Progetto di Basi di Dati

Fifa Ultimate Team

Valitutti Davide - 124/002013
Morelli Antonio - 124/001852
Scognamiglio Mario - 124/002032

Prof. Antonio Maratea

16 settembre 2020

Indice

Elenco delle figure

Elenco delle tabelle

Listings

Capitolo 1

Preambolo

1.1 EA

Fondata nel 1982, l'**Electronic Arts** è una società statunitense leader mondiale nel settore dell'intrattenimento digitale interattivo. EA sviluppa e fornisce giochi, contenuti e servizi online per console, dispositivi mobili e computer connessi a Internet, potendo vantare oltre trecento milioni di utenti attivi in tutto il mondo. EA è riconosciuta universalmente per il suo portfolio di blockbuster acclamati da pubblico e critica. Tra le altre, spicca con il suo successo mondiale la serie **FIFA Football**, il cui sviluppo è delegato a **EA Sports**, una divisione della società specializzata in titoli di simulazione sportiva.

1.2 FIFA

A partire dal 1993, ogni anno viene segnato dall'uscita di un nuovo titolo FIFA: tra alti e bassi, la serie si è attestata tra le pietre miliari del gaming, svolgendo il ruolo d'anello di congiunzione tra retro-gaming e casual-gaming. Il numero di giocatori all'attivo è costantemente in crescita, in particolare dal 2008, anno del rilascio della fortunata modalità di gioco **FIFA Ultimate Time** (abbreviata spesso in **FUT**).

1.3 FUT

Fine ultimo di FIFA Ultimate Team è la costruzione di squadre mediante l'utilizzo di giocatori provenienti dai campionati di tutto il mondo, dalla massima divisione messicana fino alla Serie A. Per certi versi, il sistema utilizzato ricorda quello delle figurine dei calciatori: ogni calciatore, così come ogni altro personaggio o oggetto inerente al mondo calcistico, viene rappresentato da una **carta FUT** ed è ottenibile mediante l'apertura di **pacchetti** o tramite **compravendita**. L' acquisto di pacchetti avviene mediante lo scambio di **crediti**, a loro volta ottenibili affrontando le squadre di altri utenti. Una prima versione della modalità prevedeva la presenza di sole tre tipologie di carte, le carte **bronzo**, **argento** e **oro**; col passare degli anni, ne sono state aggiunte un numero considerevole. Tra le altre, vi sono le carte **in form (IF)**, raffiguranti i giocatori che meglio hanno fatto nei rispettivi campionati, le carte **Record Breaker**, fornite a giocatori capaci di compiere primati particolari, o ancora le carte **Icons**, appartenenti alle grandi leggende del passato. Per il principio dei cassetti, ogni calciatore sarà quindi contraddistinto da un numero non definito di carte, ognuna avente le proprie statistiche dettate dalle sue prestazioni in campo. L' utente possiede il proprio **club**, ogni club le proprie carte; le carte giocatore andranno a comporre le rose del club, quest'ultime caratterizzate da due parametri: **la valutazione squadra**, che rappresenta essenzialmente una "media" dei valori dei singoli giocatori, e soprattutto **l'intesa**, indice dell' affiatamento dei giocatori in campo, dettata dalla corretta posizione in campo di un giocatore e dall'affinità tra gli stessi: un terzino non darà il suo massimo schierato come trequartista, così come un giocatore argentino avrà più affinità con giocatori conterranei, appartenenti alla stessa lega o ancora meglio alla stessa squadra. Si noti come, nel progetto, a determinare l'affinità di squadra sia solo il primo fattore. Svariate sono le modalità di gioco introdotte nel tempo: si va dai **tornei** ad eliminazione diretta alle partite **amichevoli** sino al sistema delle **divisioni**, risultante essere la modalità più giocata e per tanto utilizzata nel progetto. Il suo funzionamento è semplice: oltre ai crediti, lo svolgimento di una partita permette di ottenere e perdere **punti divisione**; l'oltrepassare o il regredire rispetto ad alcune soglie di punteggio determinano le **promozioni** e **retrocessioni** di un club. Maggiore è la divisione del club, maggiore sarà il numero di crediti ricevuti dopo ogni partita, maggiori il numero di pacchetti acquistabili e quindi le probabilità di migliorare le proprie rose; una formula tanto semplice quanto vincente, capace di attirare svariate milioni di giocatori ogni anno.

Capitolo 2

Progettazione

2.1 Sintesi dei requisiti

Si vuole realizzare una base di dati che permetta di gestire in modo quanto più fedele la modalità sopracitata. Individuiamo tre aree tematiche, le quali saranno analizzate singolarmente: Le carte, Il mercato e i pacchetti, Le partite e le divisioni.

2.2 Glossario

Termine	Definizione	Sinonimi
user_fut	utente della modalità	user, utente, giocatore, player
division rivals	le dieci divisioni entro le quali può trovarsi un club	divisioni, classifica divisioni, divisions
dr_points	punti che determinano la posizione in divisione	punti, punti divisione
credits	crediti appartenenti al club, utilizzati per comprare pacchetti o singole carte	crediti
card_code	codice univoco associato ad ogni carta appartenente ad un club	codice carta
card_id	codice associato ad ogni tipologia di carta	-
pack	pacchetto contenente le carte, vi sono diverse tipologie	pacchetto
stop	fermo nel quale può incorrere un giocatore a seguito di un infortunio	infortunio, fermo
chemistry	indice d'intesa, può essere di squadra o appartenere ad un singolo giocatore	intesa, affiatamento
contracts	numero di partite alle quali un calciatore o un allenatore possono prendere parte	contratti
fitness	indice di condizione fisica di un giocatore, più è basso più sarà alta la possibilità di andare incontro ad infortuni	forma fisica, condizione fisica
overall	media delle statistiche appartenenti ad un giocatore	valutazione
consumable	tipologia di carte utilizzabili una volta sola, il loro effetto è applicabile ad un giocatore o ad un allenatore	carte consumabili, consumabili
league_name	il reale campionato nel quale gioca un player o allena un manager	campionato, liga
team_name	la reale squadra nella quale gioca un player	squadra
squad	squadra formata dalle carte possedute da un club, compete con altre squadre	squadra, rosa, formazione, undici
manager	allenatore, apporta un bonus ai giocatori della squadra allenata provenienti dalla stessa nazione o dalla stessa lega	allenatore
player	calciatore, viene schierato in una o più squad	giocatore, calciatore
club_item	oggetti appartenenti a club reali, come stadi, palloni, uniformi e stemmi	oggetto del club
active_[1]_code	codice di un oggetto assegnato ad un club	-

Tabella 2.1: Glossario

[1] Categoria del club item

2.3 Il club

Ogni utente ha la possibilità di creare il proprio club. Scopo della base e tenere traccia delle carte appartenenti ad ogni club creato, ricordando che la loro provenienza può essere ricercata alternativamente tra le carte trovate nei pacchetti aperti o tra quelle facenti parte di una transizione, e delle partite svolte dai club, le quali determinano la posizione del club nella **classifica divisioni**, partendo dalla divisione iniziale, la decima, sino ad arrivare alla divisione più alta, la prima. Ogni club possiede al più *6* squad, e non più di *1500* carte.

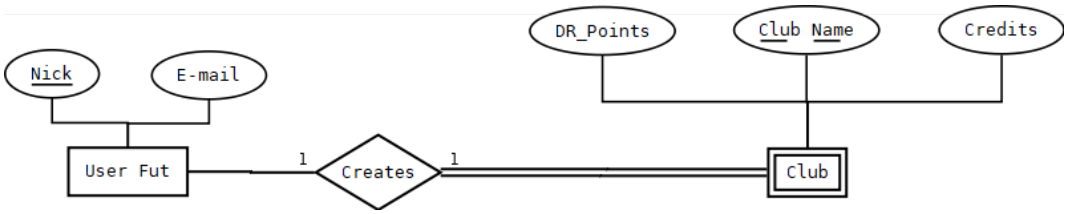

Figura 2.1: Diagramma Entità-Associazione Club

Il club possiede alcuni oggetti detti attivi quali lo **stemma**, il **pallone**, lo **stadio** e le **divise**, che non apportano particolari effetti se non dal punto di vista estetico.

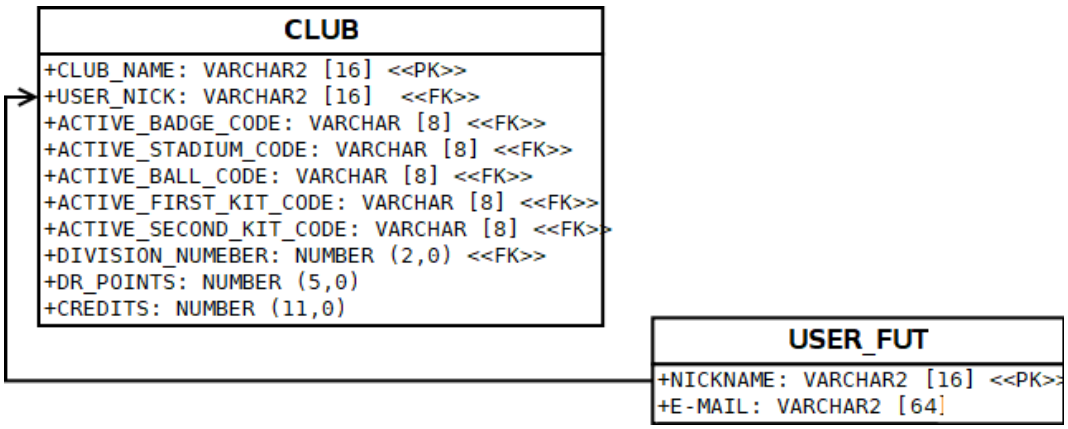

Figura 2.2: Diagramma Relazionale Club

2.4 Le carte

Sono quattro le tipologie di carte presenti nel gioco:

- Carte **player**, ovvero i calciatori che andranno a comporre le squadre che scenderanno in campo. Queste carte sono caratterizzate da un numero considerevole di indicatori, volti a determinare quale sia la valutazione della stessa: velocità, fisico, dribbling sono solo alcuni di essi. Ancora, ogni carta giocatore è contraddistinta da un ruolo e da un piede preferito, da una certa abilità con la palla e da una certa con il piede debole ed infine dal valore assunto in ognuno degli undici ruoli in cui il giocatore potrebbe essere schierato: niente, d'altronde, vieta di far difendere un attaccante, se non il buonsenso! A completare la carrellata di attributi vi sono poi dati anagrafici e soprattutto dati attivi, quali i contratti e la forma fisica;

- Carte **manager**, gli allenatori, che sedendo sulla panchina di una squadra, apportano un bonus alla valutazione dei player che provengono dalla loro stessa nazione o militano nella loro stessa lega: oltre a possedere un certo numero di contratti, sono quindi contraddistinti da una lega di provenienza, che può essere modificata a piacimento;

- Carte **consumable**, o consumabili, con un ciclo di vita pari ad un solo utilizzo. Queste carte sono di diverse tipologie, distinte mediante il campo discriminatore **category**: alcune aumentano i parametri di un giocatore per una partita, altre lo fanno per l'intera squadra, altre forniscono contratti, altre ancora diminuiscono le giornate di infortunio che un player dovrebbe attendere prima di tornare in campo. Anche queste carte sono suddivise in più categorie (bronzo, argento, oro): maggiore la rarità, maggiore l'effetto apportato;

- Carte **club item**, gli oggetti del club, elementi estetici quali divise, stadi, palloni e stemmi. A differenza dei consumabili, perdurano nel tempo, per tanto possono essere collezionati o, all'occorrenza, venduti;

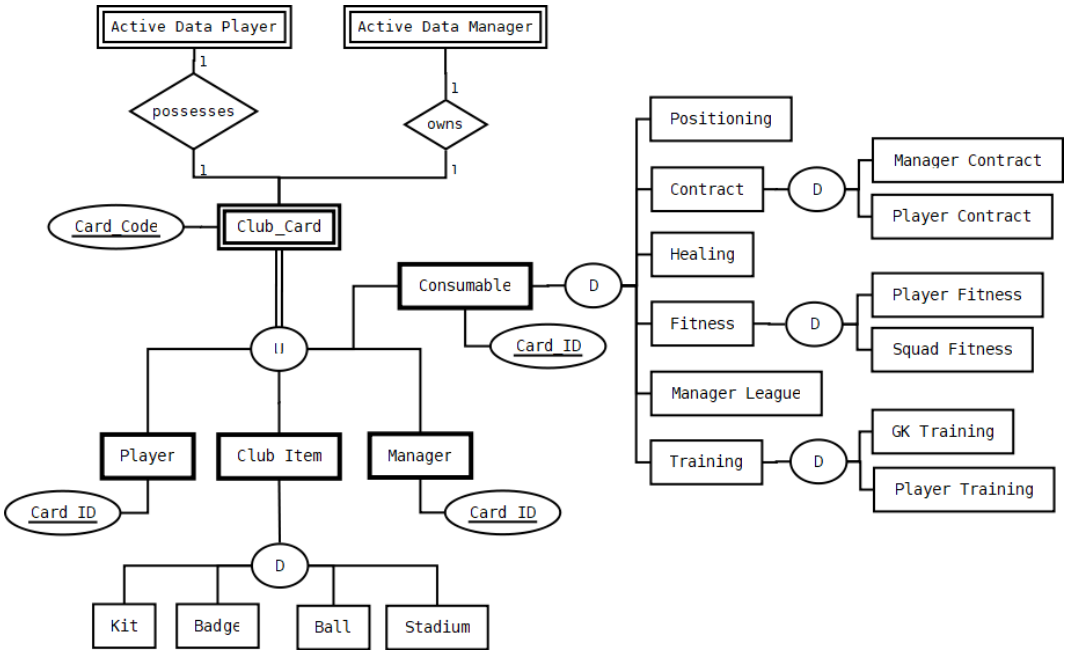

Figura 2.3: Scheletro di settore per le carte[2]

Le entità **player, manager, consumable e club item** rappresentano la nostra soluzione nel rappresentare le carte da gioco.

L'entità **club card** rappresenta invece le carte possedute dai club ed identificate dal codice univoco **card code**. Poiché diverse le associazioni condivise dalle carte, è stato scelto l'utilizzo del costrutto **unione**, il quale inoltre permette di evitare la spiacevole ridondanza che si presenterebbe memorizzando le medesime informazioni per due carte di uguale tipologia ma appartenenti a due club diversi, rendendo quindi la soluzione più apprezzabile.

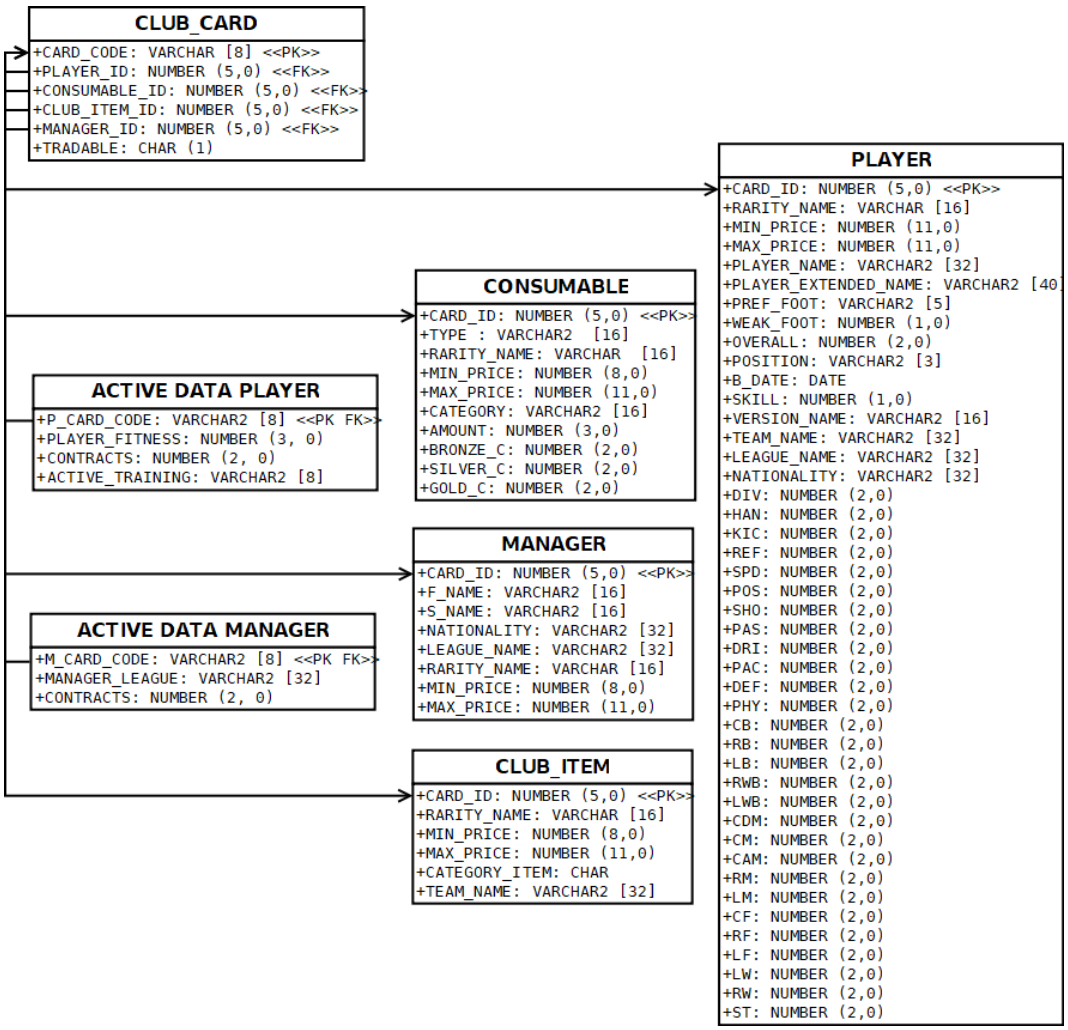

Figura 2.4: Diagramma relazionale del settore carte

Le due entità **active data player** e **active data manager** sono atte a memorizzare i dati mobili di player e manager, rispettivamente contratti, forma fisica ed eventuali consumabili di tipo training attivi per i primi, contratti e lega per i secondi; diversamente, se questi campi fossero stati memorizzati in **club card** sarebbero risultati nulli per le altre tipologie di carte.

[2]Sono stati omessi gli attributi per una migliore visione

2.5 Il mercato e i pacchetti

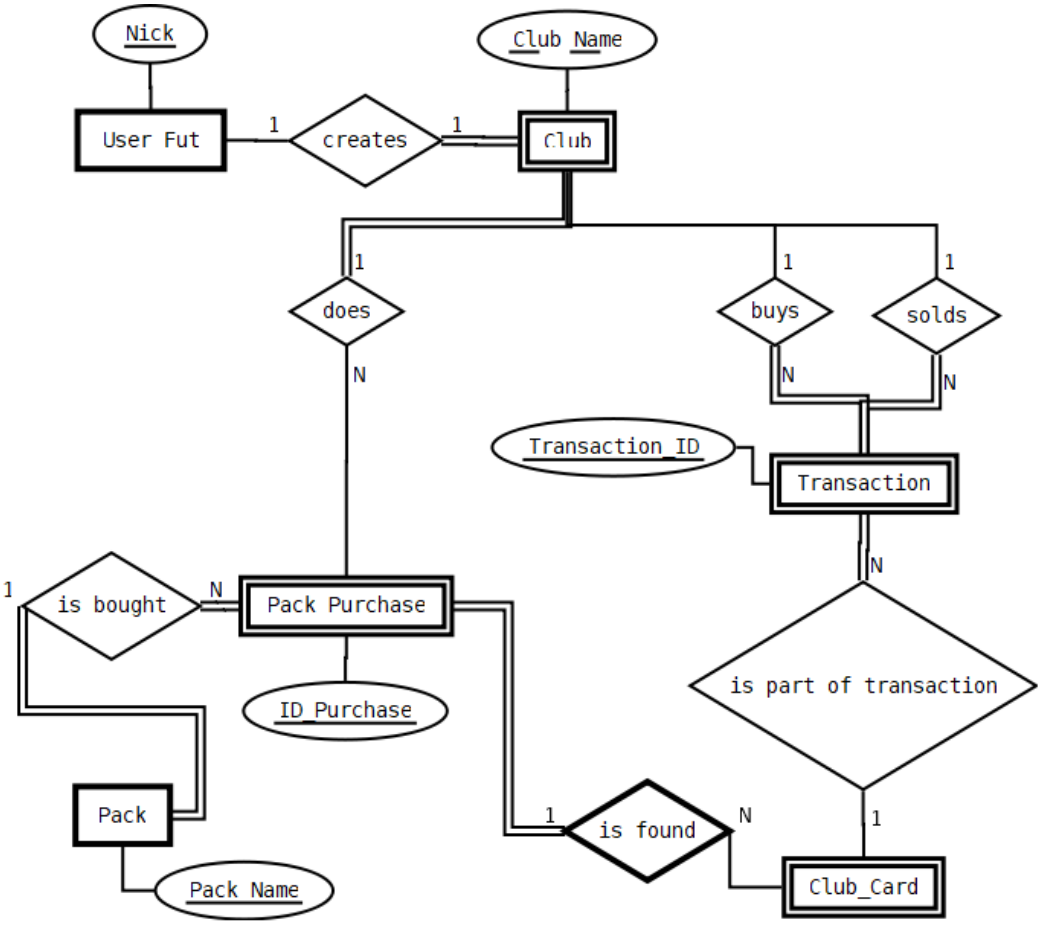

Figura 2.5: Scheletro settore per il mercato e i pacchetti

Le carte possono provenire esclusivamente da un pacchetto o essere comprate in una transizione, per tanto si è scelto di strutturare il collegamento tra **club** e **club card** nel modo esposto dal diagramma EER in figura 2.5.

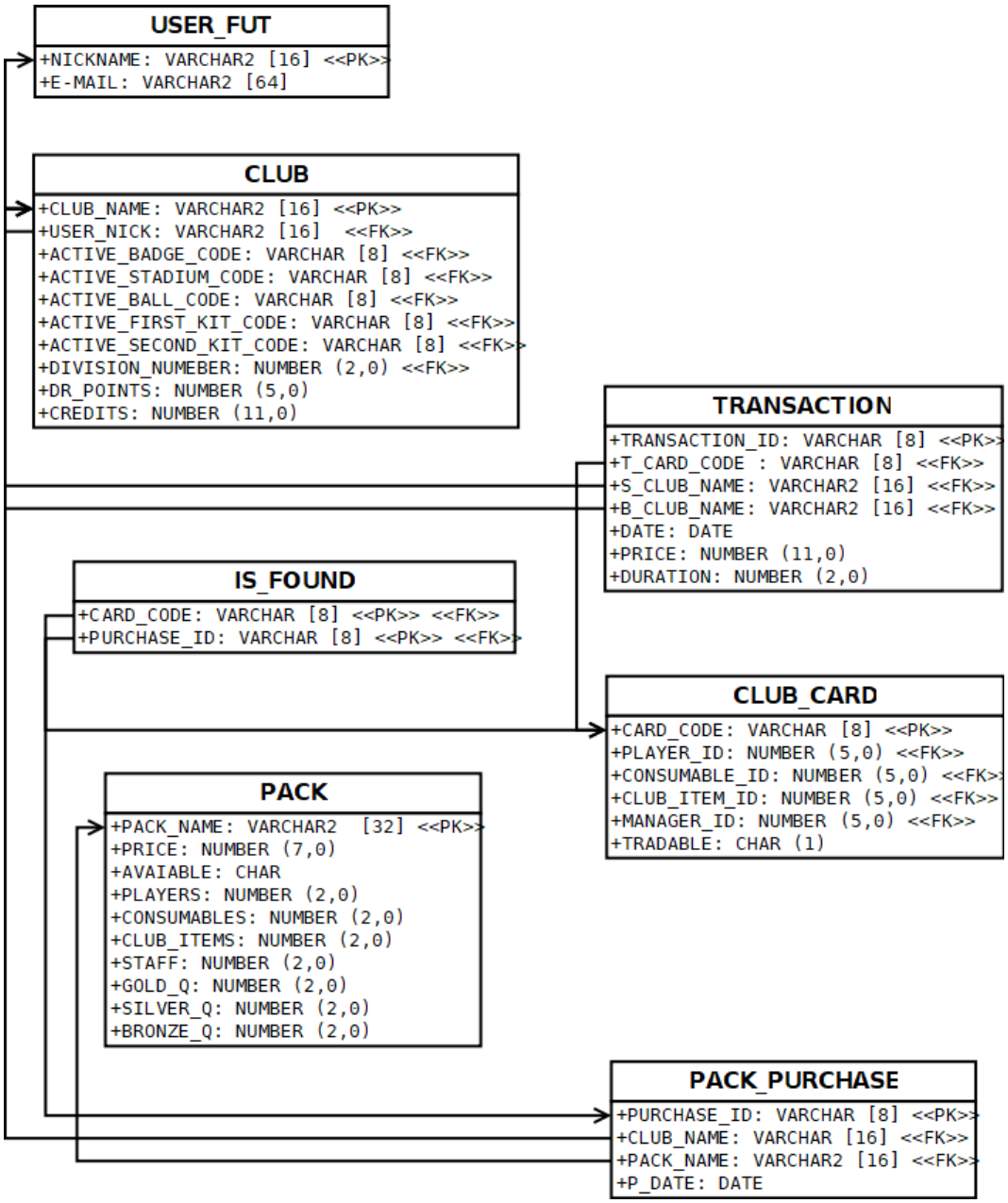

Figura 2.6: Diagramma relazionale per il mercato e i pacchetti

Si è scelto di rappresentare la relazione **is found** utilizzando una tabella di transizione. Tale decisione è stata presa in virtù del fatto che una carta ha sempre origine dall'apertura di un pacchetto ma, una volta venduta, per risalire al nuovo proprietario sarà necessario controllare la presenza di una transizione che la concerni; mantenere una foreign key in **club card** avrebbe quindi avuto l'indesiderato effetto di far sì che, a lungo andare, tale campo sarebbe risultato nullo per molte entrate della tabella.

2.6 Le partite e le divisioni

Come già anticipato, la modalità di gioco adottata nel progetto è quella delle **Division Rivals**. L'obbiettivo è di scalare una serie di divisioni; ciò non è un qualcosa di fine a sé stesso, poiché le vittorie in divisioni migliori elargiscono un maggior numero di crediti, da poter spendere nell'acquisto di ulteriori pacchetti. Inoltre, grazie all'ausilio di un job, vengono elargiti settimanalmente dei premi cui valore è direttamente proporzionale alla divisione nella quale si milita. Ogni club possiede un numero massimo di sei squadre; le

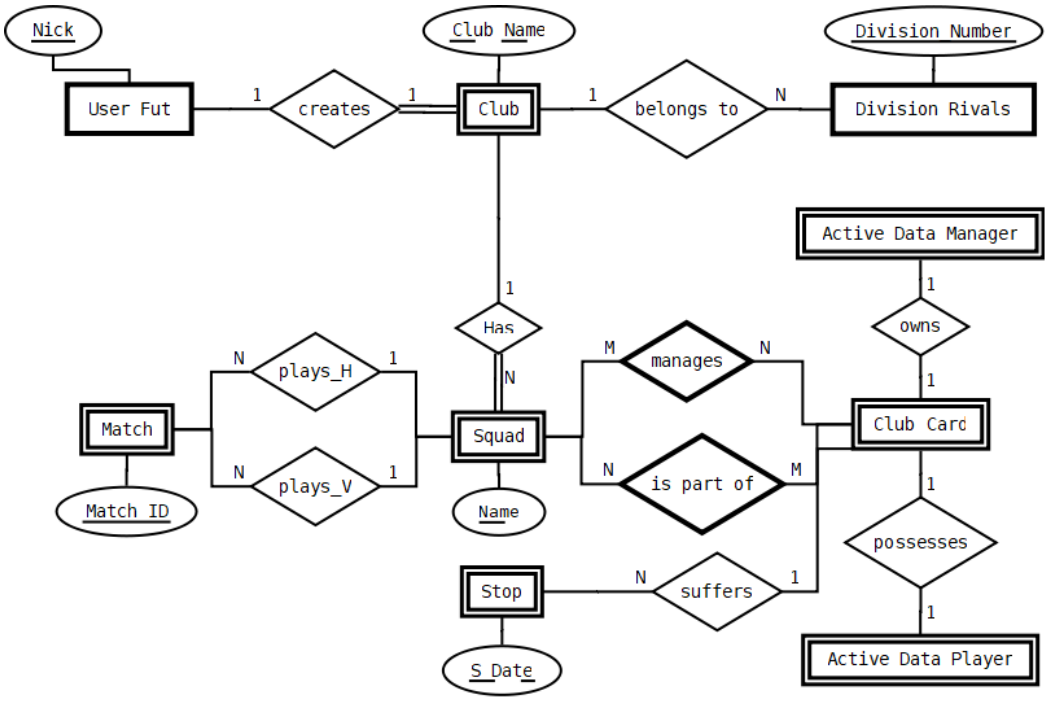

Figura 2.7: Scheletro settore per le partite e le divisioni

procedure permettono di gestire la creazione di quest'ultime, la gestione dei calciatori che andranno a comporla e dell'allenatore che siederà in panchina, memorizzati mediante le tabelle di transizione **is part of** e **manages**. La ragione che ha spinto ad utilizzare queste due tabelle sta nel fatto che gli stessi player e manager possono comporre più formazioni dello stesso club. Una squadra con almeno undici giocatori in campo, privi di infortuni e con almeno un contratto a testa, viene considerata idonea alla ricerca di un match. Trovato un altro club in attesa, viene simulato uno scontro tra le due squad, basando il risultato su alcuni parametri quali il valore della squadra, l'affinità e la forma fisica dei giocatori.

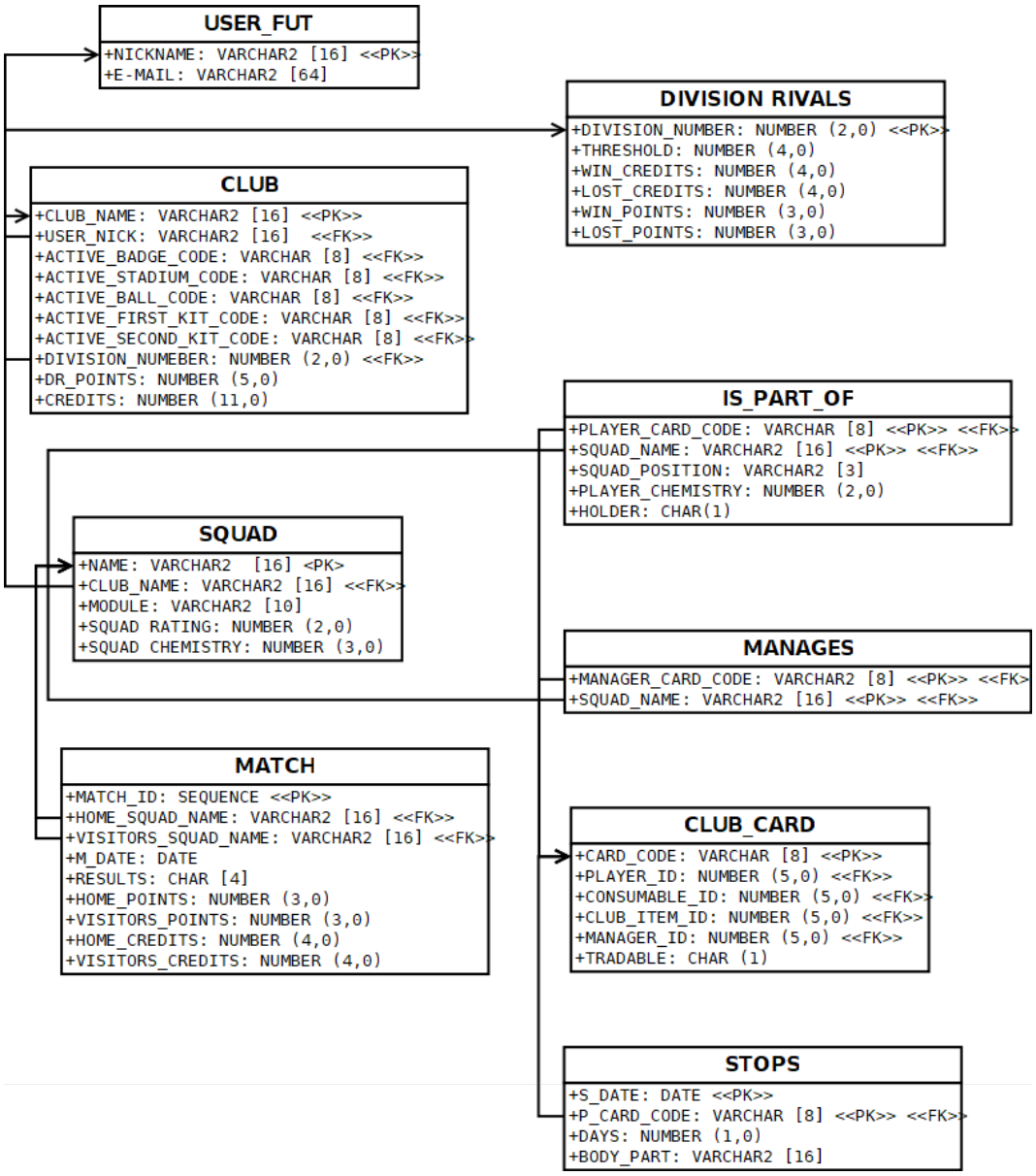

Figura 2.8: Relazionale del settore partite

Una serie di **trigger** gestiscono poi le conseguenze del dopo partita, quali l'eventualità che un giocatore si possa infortunare, l'incremento o decremento dei punti divisione e il decremento di forma fisica e contratti di player e manager.

Capitolo 3

Diagramma EER

3.1 Rappresentazione grafica

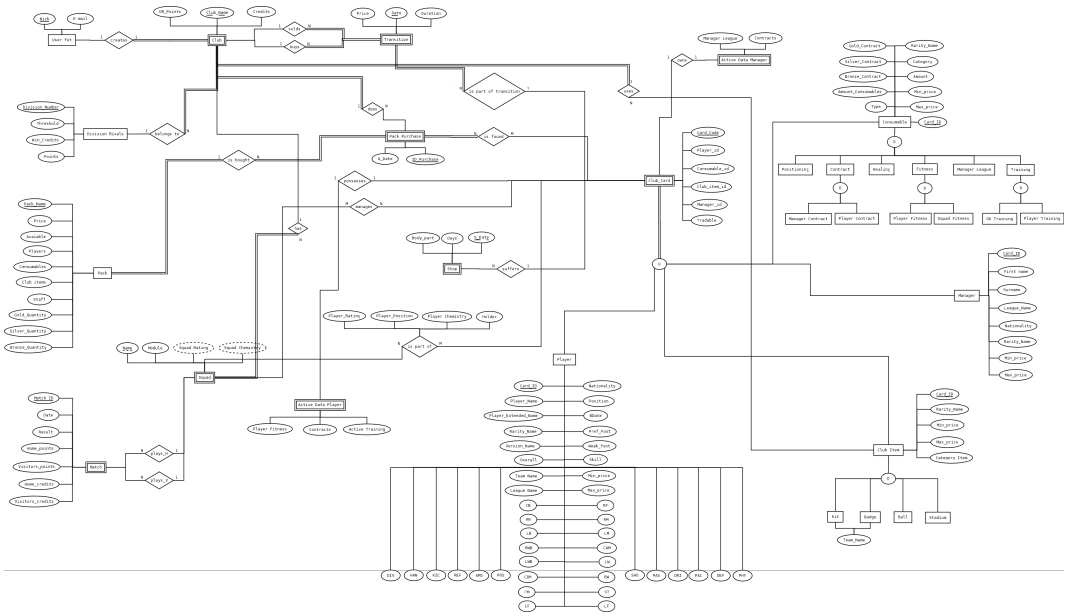

Figura 3.1: Per una migliore visione si consiglia di fare riferimento al file `ERR.png`

Capitolo 4

Diagramma Relazionale

4.1 Rappresentazione grafica

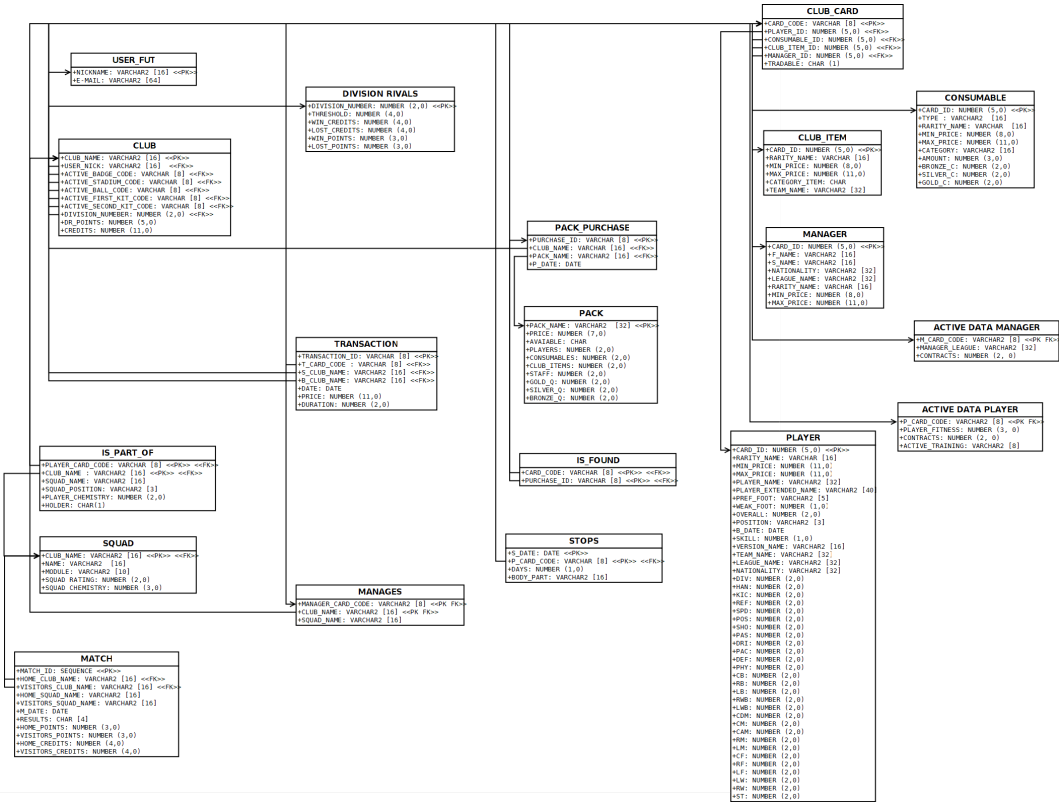

Figura 4.1: Per una migliore visione si consiglia di fare riferimento al file `REL.png`

Capitolo 5

Utenti

5.1 Categorie

Ad interagire con il database vi sono due tipologie di attori, il primo di tipo **amministratore**, libero di agire a piacimento al fine di monitorare la base, ed una seconda tipologia **comune**, che altri non rappresenta se non gli utenti che andranno ad usufruire del gioco. In particolare, sono stati creati due utenti, **PLAYER1** e **PLAYER2**, già in possesso di una squadra.

Utente	Tipo	Volume	Permessi
ADMINFUT	Amministratore	1	ALL

PLAYER	Comune	2	`SELECT ON ALL THE VIEWS;`
			`EXECUTE ON GET_SQUAD,`
			`SEARCH_CARD_FOR_SALE;`
			`EXECUTE ON ALL PROCEDURES`
			`EXECPT CLUB_CREATION;`

Tabella 5.1: Tabella utenti

5.2 Diagramma UML dei casi d'uso

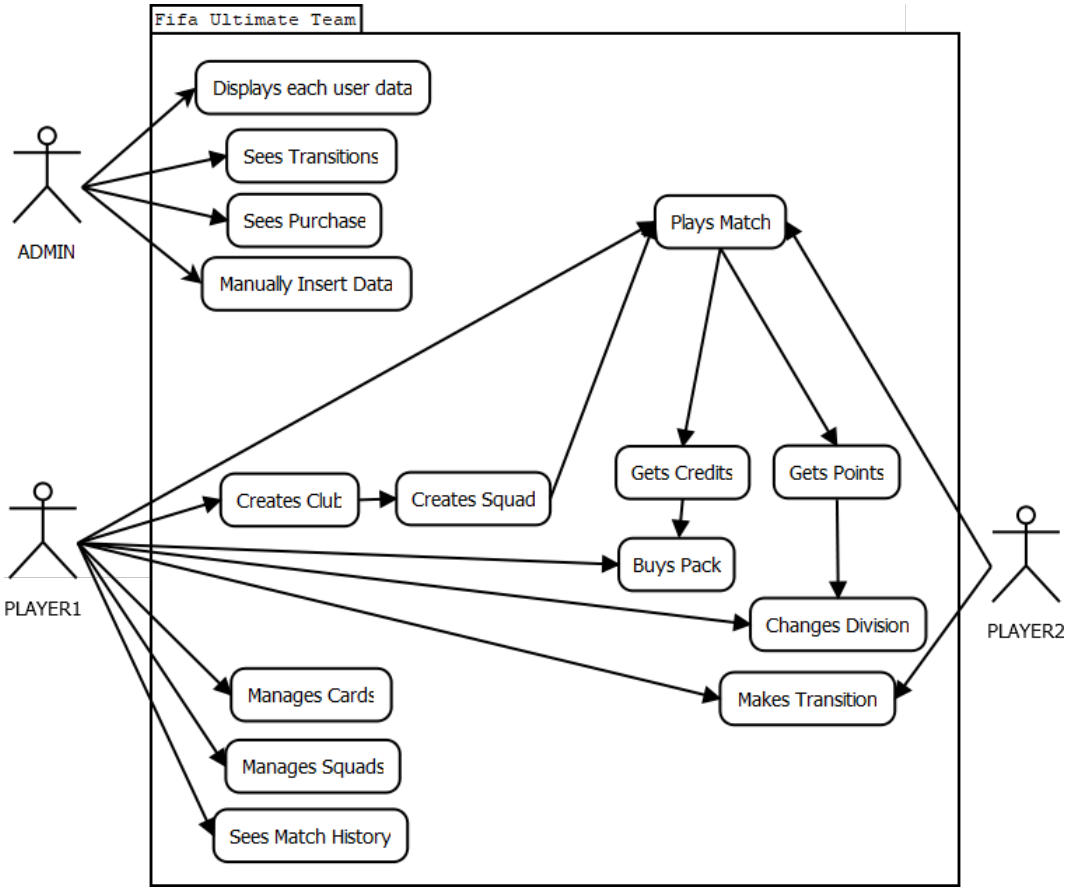

Figura 5.1: Diagramma UML dei casi d'uso

5.3 Operazioni - Utenti comuni

L'utente di tipo amministratore ha accesso completo al **database**; ciò significa che ha libertà di eseguire operazioni di inserimento, aggiornamento e modifica su qualsiasi tabella, alterando lo stato della base. Di contro, un utente comune può eseguire un set di procedure e funzioni limitato, partendo dall'acquisto di carte, sino ad arrivare alla ricerca di **partite** da affrontare.
Si noti come sia possibile anche per l'amministratore eseguire le procedure atte alla gestione di un club: se è quest'ultimo ad utilizzare la procedura, sarà necessario che venga fornito in input il nome del club sul quale si intende agire. Viceversa, un utente non dovrà necessariamente farlo, in quanto verrà utilizzato il nome del suo club; questo espediente viene realizzato mediante l'ausilio della funzione **IS_ADMIN**, che vedremo in seguito.

1. **Club Creation**
   Procedura che permette di creare un club, eseguibile solamente dall'amministratore. Crea un club associato all'utente fornito in input, il quale riceve tre pacchetti e ventimila crediti come omaggio di benvenuto. È la prima operazione da svolgere, seguita dalla creazione di una squadra.

Operazione	Club_Creation
SCOPO:	creare un club
ARGOMENTI:	nome utente, email, nome club
RISULTATO:	crea un nuovo utente con il suo club
ERRORI:	nome utente non disponibile, nome del club non disponibile
USA:	User Fut, Club
PRIMA:	si desidera creare un nuovo club
POI:	il club è stato creato

2. **Squad Creation**
   Procedura che permette di creare una squadra del club. Permette all'utente di svolgere dei **match** una volta schierati almeno **undici** giocatori titolari. Quando si crea una squadra vanno scelti nome e modulo.

Operazione	Squad_Creation
SCOPO:	creare un squadra
ARGOMENTI:	nome squadra, modulo, nome club
RISULTATO:	crea un squadra
ERRORI:	nome squadra non disponibile, modulo invalido, club inesistente, numero di squadre massimo
USA:	Squad
PRIMA:	si possiede un club e si vuole creare una squadra
POI:	la squadra è stata aggiunta

3. **Sell Card**
Procedura usata per vendere una carta posseduta dal club. Inserisce una tupla in Transaction contenente il codice carta, la data e il nome del club che intende vendere, ponendosi in attesa di un compratore.

Operazione	Sell_Card
SCOPO:	vendere una carta
ARGOMENTI:	codice carta, nome club, prezzo, durata
RISULTATO:	inserisce la carta nel mercato, pronta alla vendita
ERRORI:	codice carta errato, club inesistente, carta già in vendita
USA:	Club Card, Transaction, Player, Consumable, Club Item, Manager
PRIMA:	si possiede una carta che si intende vendere
POI:	la carta è stata posta in vendita

4. **Buy Card**
Procedura che permette di comprare una carta in vendita. Possiamo cercare una specifica carta mediante **SEARCH_CARD_FOR_SALE** o visualizzare tutti i player in vendita con **GET_PLAYERS_FOR_SALE**. Sottrae il costo della carta al bilancio del club, aggiunge la carta alle carte del club e infine la rimuove dalle carte del club che intende vendere, eliminando anche le vecchie tuple da transaction o da is_found. L'utente che compra fornisce in input un prezzo massimo oltre il quale la carta, anche se trovata, non verrà comprata.

Operazione	Buy_Card
SCOPO:	comprare una carta
ARGOMENTI:	codice carta, nome club, prezzo
RISULTATO:	il club che ha acquistato possiede la carta
ERRORI:	il club compratore è uguale al club venditore, dati errati
USA:	Transaction, Club
MODIFICA:	Transaction, Club
PRIMA:	si vuole comprare una carta in vendita
POI:	la carta è stata acquistata

5. **Pack Opening**
Procedura usata per aprire un pacchetto. Possedere più carte significa avere squadre più forti, per tanto risulta fondamentale aprire i pacchetti. Esistono pacchetti di diversa tipologia, con costi e quantitativi di carte diversi. Per conoscere il nome e il prezzo dei pacchetti disponibili si utilizza **GET_AVAILABLE_PACK**.

Operazione	Pack_Opening
SCOPO:	acquistare un pacchetto
ARGOMENTI:	nome pacchetto, nome utente
RISULTATO:	aumenta la collezione delle carte del club
ERRORI:	pacchetto non disponibile, crediti insufficienti, dati errati
USA:	Club Card, Club, Player, Consumable, Club Item, Manager, Is Found, Pack Purchase
MODIFICA:	Club Card, Club, Is Found, Pack Purchase
PRIMA:	si desidera acquistare un pacchetto
POI:	il pacchetto è stato acquistato

6. **Add Player**
Procedura che permette di schierare un player in una squad del club, dopo averla creata mediante **SQUAD_CREATION**. Un player non schierato in nessuna squadra è comunque disponibile nella propria collezione di carte. Uno degli argomenti che la procedura riceve in ingresso è la posizione nella quale si intende schierare il giocatore: ogni modulo ha le proprie posizioni, per tanto, provare a mettere un'ala in un 4-4-2 fa sì che il giocatore venga posto in panchina; ancora, tentare di schierare un giocatore con undici posizioni titolari già occupate darà il medesimo risultato. Se infine risultano essere presenti diciassette giocatori in squadra, numero massimo di slot disponibili, la procedura non andrà a buon fine.

Operazione	Add_Player
SCOPO:	aggiungere un player ad una squadra di un club
ARGOMENTI:	codice carta, nome squadra, nome del club, posizione, flag[1]
RISULTATO:	schiera il player in una squadra del club
ERRORI:	player già schierato, squadra al completo, dati errati
USA:	Club Card, Player, Is Part Of, Squad
MODIFICA:	Is Part Of
PRIMA:	si desidera schierare un player in una squadra
POI:	il player è stato schierato in squadra

[1] *Zero* il player siederà in panchina, *uno* andrà a comporre l'undici titolare

7. **Change Player**
Procedura usata per scambiare di posto due giocatori appartenenti alla stessa squadra, uno riserva, l'altro titolare.

Operazione	Change_Player
ARGOMENTI:	codice carta player a, codice carta player b, nome squadra, nome club
RISULTATO:	scambia il player a con il b
ERRORI:	dati non trovati, dati errati, a e b sono titolari, a e b sono riserve
USA:	Is Part Of
PRIMA:	si desidera sostituire un player infortunato
POI:	il player è stato sostituito

8. **Remove Player**
Procedura utilizzata al fine di rimuovere un giocatore schierato in una squadra.

Operazione	Remove_Player
ARGOMENTI:	codice carta, nome squadra, nome club
RISULTATO:	rimuove il player dalla squadra
ERRORI:	dati errati, player non schierato
USA:	Is Part Of, Squad
PRIMA:	si desidera rimuovere un player da una squadra
POI:	il player è stato rimosso dalla squadra

9. **Insert Match**
Procedura usata per cercare un avversario e disputare un match. Necessario per il club che intende disputare il match è farlo con una squadra con undici giocatori titolari, ognuno di essi con almeno un contratto disponibile e senza infortuni. Avviata la procedura, si simula l'attesa di un altro utente che faccia altrettanto; fatto ciò, un algoritmo influenzato dalla valutazione della squadra, dall' intesa di squadra e dalla forma fisica dei singoli giocatori genera un risultato in modo casuale.

Operazione	Insert_Match
SCOPO:	disputare un match
ARGOMENTI:	nome squadra, nome club
RISULTATO:	inserisce la squadra in cerca di un avversarrio
ERRORI:	dati non trovati, dati errati, giocatori in sqaudra insufficenti, squadra già in attesa, club già in attesa, player senza contratti, player titolari infortunati, manager senza contratti
USA:	Is Part Of, Squad, Match, Club
MODIFICA:	Club, Match
PRIMA:	si desidera disputare una partita
POI:	se disponibile un'avversario viene disputata la partita, altrimenti si rimane in attesa

10. **Add Training**
Procedura che permette di utilizzare le carte di tipo consumabile, categoria training, su di un player; l'effetto di tali carte ha ripercussione su una sola partita, ed è quello di incrementare la valutazione di un giocatore di due unità, modificando quindi la valutazione totale della squadra e influendo sulle sorti del match.

Operazione	Add_Training
SCOPO:	utilizzare la carta di tipo consumabile di categoria training
ARGOMENTI:	codice carta consumabile, codice carta player, nome squadra, nome club
RISULTATO:	incrementa la valutazione del player di 2 per una partita
ERRORI:	dati errati, dati non trovati
USA:	Club Card, Is Part Of, Active Data Player, Squad
MODIFICA:	Club Card, Active Data Player, Squad, Is Part Of
PRIMA:	si desidera aumentare la valutazione di un player
POI:	il training apporta un bonus al player

11. **Add Positioning**
Procedura che permette di utilizzare le carte di tipo consumabile, categoria positioning, su di un player. I consumabili di tipo positioning modificano la posizione designata di un giocatore, scegliendone una affine. Una prima punta potrà quindi diventare seconda punta o anche trequartista, ma non c'è verso di renderla un difensore centrale.

Operazione	Add_Positioning
SCOPO:	utilizzare la carta di tipo consumabile di categoria positioning
ARGOMENTI:	codice carta consumabile, codice carta player, nome club
RISULTATO:	cambia la posizione attuale in squadra di un player
ERRORI:	dati errati, dati non trovati
USA:	Club Card, Is Part Of, Consumable
MODIFICA:	Is Part Of, Club Card
PRIMA:	si desidera cambiare la posizione di un player
POI:	la posizione è stata cambiata

12. **Add Healing**
Procedura che permette di utilizzare le carte di tipo consumabile, categoria healing, su di un player. I consumabili di categoria healing permettono di diminuire le giornate di non disponibilità per un giocatore infortunato, evitando di dover svolgere quel numero di partite senza poterlo utilizzare. Ve ne sono di diverso tipo, alcune curano infortuni alla testa, altre al braccio, altre ancora qualsiasi tipo di infortunio.

Operazione	Add_Healing
SCOPO:	utilizzare la carta di tipo consumabile di categoria healing
ARGOMENTI:	codice carta consumabile, codice carta player, nome club
RISULTATO:	curare un player da un infortunio
ERRORI:	dati non trovati, player non infortunato, consumabile non compatibile con l'infortunio
USA:	Club Card, Consumable, Stop
MODIFICA:	Club Card, Stop
PRIMA:	il player ha subito un infortunio
POI:	i giorni di stop nei quali incorre il player sono stati diminuiti[2]

[2]Se pari a *zero* il player guarirà

13. **Add Contract**
Procedura che permette di utilizzare le carte di tipo consumabile, categoria contract, su di un player o un manager. I consumabili di categoria contract aumentano sino ad un massimo di novantanove i contratti posseduti da una carta giocatore o allenatore, in vero il numero di partite nelle quali può essere schierato in campo o allenare in panchina.

Operazione	Add_Contract
SCOPO:	utilizzare la carta di tipo consumabile di categoria contract
ARGOMENTI:	codice carta consumabile, codice carta player, nome del club
RISULTATO:	viene aumentato l'attributo contratti
ERRORI:	dati non trovati
USA:	Club Card, Active Data Player
PRIMA:	si desidera incrementare i contratti di un player o un manager
POI:	i contratti sono stati incrementati

14. **Add Fitness**
Procedura che permette di utilizzare le carte di tipo consumabile, categoria fitness. I consumabili di categoria fitness possono essere applicabili ad un singolo player o all'intera squadra; in questo caso, il codice carta del consumabile da applicare dovrà essere NULL. La forma fisica dei player diminuisce dopo ogni partita giocata, influendo sui risultati in campo e sulla probabilità di incorrere in infortuni.

Operazione	Add_Fitness
SCOPO:	utilizzare la carta di tipo consumabile di categoria fitness
ARGOMENTI:	nome del club, codice carta consumabile, tipo del consumabile, codice carta player[3]
RISULTATO:	aumenta la forma fisica del player o di tutta la squadra
ERRORI:	dati non trovati
USA:	Club Card, Consumable, Active Data Player
PRIMA:	si desidera incrementare la forma fisica di un player
POI:	la forma fisica è stata incrementata

[3]Se il tipo è squad, questo argomento deve essere *NULL*

15. **Add Manager League**
Procedura che permette di utilizzare le carte di tipo consumabili, categoria manager league, su di un allenatore. Questo consumabile modifica la lega attuale del manager schierato in una squadra; può essere utile affinché l'allenatore assuma la stessa lega dei giocatori schierati in campo, per potervi dare un bonus intesa.

Operazione	Add_Manager_League
SCOPO:	utilizzare la carta di tipo consumabile di categoria manager league
ARGOMENTI:	codice carta consumabile, codice carta manager, nome club
RISULTATO:	cambia la liga attuale del manager
ERRORI:	dati non trovati, dati errati
USA:	Club Card, Active Data Manager, Consumable
MODIFICA:	Club Card, Active Data Manager
PRIMA:	si desidera modificare la lega di appartenenza di un manager
POI:	la lega è stata modificata

16. **Add Club Item**
Procedura che permette di utilizzare le carte di tipo club item assegnando tale oggetto al club e rendendolo quindi attivo. Le carte club item sono di quattro tipologie, Stemma, Stadio, Pallone e Uniforme. Diversamente dai consumabili, i club item restano sempre nel club e una volta attivati non vengono eliminati. Purtroppo, l'effetto della procedura non è apprezzabile dal punto di vista grafico.

Operazione	Add_Club_Item
SCOPO:	utilizzare le carte di tipo club item
ARGOMENTI:	codice carte, nome club
RISULTATO:	rende attivo il club item
ERRORI:	carte già attiva, dati non validi, dati non trovati
USA:	Club Card, Club
MODIFICA:	Club
PRIMA:	si desidera rendere attivo un club item
POI:	il club item è stato attivato

17. **Add Manager**
Procedura che permette di designare un manager come allenatore di una squad. Un manager non schierato in nessuna squadra è comunque disponibile nella propria collezione di carte. Qualora la nazionalità o la lega dei player schierati nella stessa squadra corrispondessero a quelle del manager, questi otterrebbero un incremento d'intesa.

Operazione	Add_Manager
SCOPO:	designare un manager in una squadra del club
ARGOMENTI:	codice carta, nome squadra, nome club
RISULTATO:	schiera il manager
ERRORI:	manager già schierato, dati non trovati
USA:	Club Card, Manager, Manages, Active Data Manager, Player, Is Part Of
MODIFICA:	Manages, Active Data Manager, Is Part Of
PRIMA:	si desidera porre un manager alla guida di una squadra
POI:	il manager è stato schierato nella squadra

18. **Remove Manager**
Procedura utilizzata al fine di sollevare un manager dall'incarico di allenare una squadra.

Operazione	Remove_Manager
ARGOMENTI:	codice carta, nome squadra, nome club
RISULTATO:	rimuove il manager dalla squadra
ERRORI:	dati non trovati, squadra senza manager
USA:	Club Card, Manages, Manager, Is Part Of, Player
MODIFICA:	Manages, Is Part Of
PRIMA:	si desidera rimuovere un manager dalla guida di una squadra
POI:	il manager è stato rimosso dalla squadra

19. **Club Delete**
Procedura finalizzata al rimuovere un club con tutte le sue carte.

Operazione	Club_Delete
ARGOMENTI:	nome club
RISULTATO:	elimina il club e tutte le sue carte
ERRORI:	club inesistente
USA:	Club, Club Card, Is Found, Pack Purchase
PRIMA:	si desidera eliminare il club
POI:	il club è stato eliminato

20. **Get Squad**
Funzione utile per ricevere dettagli sui giocatori della propria squadra

Operazione	Get_Squad
ARGOMENTI:	nome club, nome squadra
RISULTATO:	visualizza i dettagli della squadra
USA:	Player, Club Card, Active Data Player, Is Part Of
PRIMA:	si desidera avere informazioni su di una squadra
POI:	vengono visualizzati i dettagli dei giocatori della squadra

21. **Search Card For Sale**
Funzione utile a cercare se una determinata carta è posta in vendita.

Operazione	Search_Card_For_Sale
ARGOMENTI:	nome club, tipologia di carta, nome della carta
RISULTATO:	visualizza i dettagli della carta ricercata se essa è in vendita
USA:	Player, Consumable, Club Item, Manager, Club Card, Transaction
PRIMA:	si desidera cercare un carta specifica tra le transazioni
POI:	vengono visualizzati i dettagli della carta ricercata con relativo prezzo

5.4 Viste

1. **Get Active Club Items**
   Vista che permette di visualizzare quali sisano le carte club item attive nel club dell'utente.
2. **Get Club Info**
   Vista che permette di visualizzare dettagli sul club dell'utente, quale divisione, numero di punti e numero di crediti.
3. **Get Club Items**
   Vista che permette di visualizzare le carte club item presenti nella collezione dell'utente.
4. **Get Consumables**
   Vista che permette di visualizzare le carte consumable presenti nella collezione dell'utente.
5. **Get Managers**
   Vista che permette di visualizzare le carte manager presenti nella collezione dell'utente.
6. **Get Manages**
   Vista che permette di visualizzare gli allenatori del club attualmente posti alla guida della panchina.
7. **Get Players**
   Vista che permette di visualizzare le carte players presenti nella collezione dell'utente.
8. **Get Available Packs**
   Vista che permette di visualizzare dettagli su quali siano i pacchetti comprabili attualmente; è discrezione di un utente amministratore il rendere o meno un pacchetto comprabile.
9. **Get Pack Purchases**
   Vista che permette di visualizzare la storia dei pacchetti acquistati dall'utente.
10. **Get Players For Sale**
    Vista che permette di visualizzare le carte player attualmente in vendita sul mercato.
11. **Get Players Stops**
    Vista che permette di visualizzare quali siano i player del club dell'utente in attesa di rientrare da uno stop.
12. **Get Match**
    Vista che permette di visualizzare le storia delle partite affrontate dal club dell'utente.

5.5 Operazioni - Amministratori

Un utente amministratore non può usufruire delle viste fornite ad un utente comune poichè non possiede un club. Per sopperire a tale problematica, sono state create diverse funzioni che permettono di avere maggiore controllo dei dati presenti nella base.

1. **Get Active Club Items F**
   Funzione che permette di visualizzare quali siano le carte club item attive nel club fornito in ingresso.
2. **Get Consumables F**
   Funzione che permette di visualizzare quali siano le carte consumable presenti nella collezione del club fornito in ingresso.
3. **Get Club Item F**
   Funzione che permette di visualizzare quali siano le carte club item presenti nella collezione del club fornito in ingresso.
4. **Get Manager F**
   Funzione che permette di visualizzare quali siano le carte manager presenti nella collezione del club fornito in ingresso.
5. **Get Manages F**
   Funzione che permette di visualizzare gli allenatori del club fornito in ingresso attualmente posti alla guida della panchina.
6. **Get Players F**
   Funzione che permette di visualizzare quali siano le carte player presenti nella collezione del club fornito in ingresso.
7. **Get Match F**
   Funzione che permette di visualizzare la storia delle partite affrontate dal club fornito in ingresso.
8. **Get Pack Purchase F**
   Funzione che permette di visualizzare la storia dei pacchetti acquistati dal club fornito in ingresso.
9. **Get Players Stops F**
   Funzione che permette di visualizzare quali siano i player del club fornito in ingresso in attesa di rientrare da un infortunio.

Capitolo 6

Jobs

Un job può esser considerato come un insieme di operazioni che devono essere eseguite in modo sequenziale e automatico. Un job può esser avviato manualmente dall'utente, o semplicemente può esser schedulato in modo adeguato per avviarlo nei giorni e negli orari desiderati.

1. **Job Transaction**
   Schedulato affinché si ripeta ogni ora, il job diminuisce la durata residua delle transizioni nello store che non hanno trovato un acquirente. Una volta arrivata a zero, la transizione viene eliminata.
2. **Job Gift**
   Schedulato affinché si ripeta settimanalmente, ogni giovedì alle ore nove del mattino, il job elargisce premi ai club consistenti in pacchetti e crediti. Il loro valore è tanto alto quanto più è alta la divisione nella quale si trova il club.
3. **Job Old Match**
   Schedulato affinchè si ripeta giornalmente, a mezzanotte, il job elimina i dati memorizzati nella tabella **TMP_MATCH**, utilizzata come supporto ai trigger **PLAYER_UPDATE_AFTER_MATCH**, **VERIFY_SQUAD** e **STOPS_RANDOMIZER**.

Capitolo 7

Vincoli d'integrità

7.1 Vincoli di integrità statici

- L'attributo **AVAILABLE** dell'entità **Pack** non può assumere valori diversi da *0* e *1*. In quanto uguale a *0* indica che il pacchetto non è disponibile all'acquisto, mentre uguale ad *1* indica che il pack è disponibile.

```
CONSTRAINT CHECK_AVAILABLE CHECK (AVAILABLE IN ('0' , '1')
    )
```

- L'attributo **Holder** dell'entità **Is Part Of** indica la titolarità o meno di un player in una squadra e per tanto assume uno tra i due valori zero ed uno, che indicano rispettivamente titolarità e non.

```
CONSTRAINT CHECK_HOLDER CHECK (HOLDER IN ('0' , '1'))
```

- L'attributo **Tradable** dell'entità **Club Card** indica la possibilità o meno di vendere una carta del club e per tanto assume uno tra i due valori zero ed uno, che indicano rispettivamente possibilità e impossibilità alla vendita.

```
CONSTRAINT CHECK_TRADABLE CHECK (TRADABLE IN ('0' , '1'))
```

- L'attributo **Rare Flag** delll'entità **Player** indica la rarità o meno di una carta del club e per tanto assume uno tra i due valori zero ed uno, che indicano rispettivamente rarità e non rarità.

```
CONSTRAINT CHECK_RARE_FLAG CHECK (RARE_FLAG IN ('0' , '1')
    )
```

- L'attributo **Modules** dell'entità **Squad** assume esclusivamente valori che esprimono un modulo secondo il quale i giocatori verranno schierati in campo.

```
CONSTRAINT CHECK_MODULES CHECK (MODULES IN ('3-1-4-2', '
    3-4-1-2', '3-4-3', '3-5-2', '4-1-2-1-2', '4-1-3-2', '
    4-1-4-1', '4-2-2-2', '4-2-3-1', '4-2-4', '4-3-1-2', '
    4-3-2-1', '4-3-3', '4-4-1-1', '4-5-1', '4-4-2', '
    5-2-1-2', '5-2-2-1', '5-3-2', '5-4-1', '5-2-1-2')
```

- L'attributo **Player Fitness** dell'entità **Active Data Player** indica la forma fisica di un giocatore ed assume valori compresi tra zero e novantanove.

```
CONSTRAINT CHECK_FITNESS CHECK (PLAYER_FITNESS BETWEEN 1
    AND 100);
```

- L'attributo **Pref Foot** dell'entità **Player** indica quale sia il piede preferito del giocatore, per tanto assume i due valori **Left** o **Right**.

```
CONSTRAINT CHECK_PREF_FOOT CHECK (PREF_FOOT IN ('Left','
    Right')
```

- L'attributo **Overall** dell'entità **Player** indica la valutazione di un giocatore ed assume valori tra uno e novantanove.

```
CONSTRAINT CHECK_OVERALL CHECK (OVERALL BETWEEN 1 AND 100)
```

- L'attributo **Position** dell'entità **Player** indica la posizione in campo del giocatore, per tanto assume uno tra i diciassette ruoli ricopribili.

```
CONSTRAINT CHECK_POSITION CHECK (POSITION IN 'GK', 'RB', '
    RWB', 'CB', 'LB', 'LWB', 'CDM', 'CM', 'CAM', 'RM', 'RW'
    , 'LM', 'LW', 'CF', 'RF', 'LF', 'ST');
```

- L'attributo **Weak Foot** dell'entità **Player** indica l'abilità di un player nell'usare il piede non preferito ed assume valori compresi tra uno e cinque.

```
CONSTRAINT CHECK_WEAK_FOOT CHECK ( WEAK_FOOT IN (1, 2, 3,
    4, 5)
```

- L'attributo **Skill** dell'entità **Player** indica l'abilità di un player nell'eseguire mosse abilità ed assume valori compresi tra uno e cinque.

```
CONSTRAINT CHECK_SKILL CHECK ( SKILL IN (1, 2, 3, 4, 5))
```

- L'attributo **Category** dell'entità **Consumable** deve essere specifico per le categorie di carte consumabili, ovvero *Contract, Healing, Fitness, GKTraining, Training, Positioning, Manager League*.

```
CONSTRAINT CHECK_CATEGORY CHECK ( CATEGORY IN ('Contract',
     'Healing', 'Fitness', 'GKTraining','Positioning', '
    Manager League'))
```

- L'attributo **Category Item** dell'entità **Club Item** deve essere specifico per le categorie di **club item**, ovvero *Kit, Ball, Badge, Stadium*.

```
CONSTRAINT CHECK_CATEGORY_ITEM CHECK (CATEGORY_ITEM IN ('
    Kit', 'Ball', 'Badge', 'Stadium'))
```

- L'attributo **Days** dell'entità **Stop** indica i giorni di infortunio nei quali incorre un giocatore ed assume valori tra uno e cinque.

```
CONSTRAINT CHECK_DAYS CHECK (DAYS BETWEEN 1 AND 6)
```

- L'attributo **Chemistry** dell'entità **Squad** indica l'affiatamento di una squadra ed assume un valore minimo di zero sino ad un massimo di cento.

```
CONSTRAINT CHECK_S_CHEMISTRY CHECK (SQUAD_CHEMISTRY
    BETWEEN 0 AND 101)
```

- L'attributo **Rating** dell'entità **Squad** indica la valutazione di una squadra ed assume un valore minimo di zero sino ad un massimo di novantanove.

- Tutti gli attributi che contraddistinguono le statistiche di un giocatore, presenti nell'entità **Player**, assumono un valore compreso tra zero e novantanove.

- Le **reti** segnate da una squadra impegnata in un match vanno da un minimo di zero ad un massimo di sette.

- **crediti** e **punti divisione** ottenuti o persi dopo un match vengono gestiti come di seguito:
  - Pareggio -> dai 100 ai 130 crediti, dai 15 ai 30 punti;
  - Vittoria -> dai 200 ai 230 crediti, dai 45 ai 55 punti;
  - Sconfitta -> dai 45 ai 55 crediti, dai -30 ai -15 punti:

7.2 Vincoli di integrità dinamici

1. Si vuole gestire la concessione dei privilegi di un nuovo utente tramite un trigger.
   Soluzione: Trigger **Insert User**.

2. Si vuole criptare l'e-mail di un nuovo utente per una maggiore sicurezza nei dati sensibili. Prima che l'e-mail venga inserita nel database verrà criptata attraverso l'algoritmo MD5.
   Soluzione: Trigger **Email2MD5**.

3. Si vuole gestire il caso in cui venga messo in vendita un giocatore titolare in una squadra; quest'ultimo dovrà essere rimosso da ogni squadra in cui è schierato, prima di poter essere inserito tra le vendite.
   Soluzione: Trigger **Player in Squad**.

4. Si vuole gestire la verifica di una squadra che vuole disputare una partita. In particolare, è necessario che vengano soddisfatti i seguenti requisiti:
   (a) Vi siano almeno undici giocatori titolari;
   (b) Ognuno dei giocatori, titolari o meno, deve possedere un contratto;
   (c) Ognuno dei giocatori, titolari o meno, non deve essere affetto da infortunio;
   (d) Se presente, il manager deve possedere un contratto;

   Soluzione: Trigger **Verify Squad**.

5. Si vuole gestire il caso in cui una squadra che sta per disputare una partita abbia giocatori con la forma fisica minore di 10, fattore che influirebbe in modo pesante sul risultato.
   Soluzione: Trigger **Check Player Fitness**.

6. Si vogliono gestire la promozione e la retrocessione di un club dopo una variazione di punti divisione. Il club otterrà una promozione se i suoi punti di abilità superano la soglia della divisione successiva a quella in cui milita; viceversa, il club verrà retrocesso se i punti abilità sono inferiori alla soglia della propria divisione e il club non si trova in divisone dieci, ultima divisione.

   (a) Divisione 1 -> 2400;

   (b) Divisione 2 -> 2200;

   (c) Divisione 3 -> 1900;

   (d) Divisione 4 -> 1600;

   (e) Divisione 5 -> 1200;

   (f) Divisione 6 -> 800;

   (g) Divisione 7 -> 500;

   (h) Divisione 8 -> 250;

   (i) Divisione 9 -> 100;

   (j) Divisione 10 -> 10;

   Soluzione: Trigger **Division Update**.

7. Si vuole gestire l'eventuale presentarsi di infortuni dopo aver disputato una partita. Un giocatore con forma fisica maggiore uguale ad ottanta non può incorrere in infortunio; viceversa, la probabilità che incomba è direttamente proporzionale al diminuire della stessa.

   Soluzione: Trigger **Stops Randomizer**.

8. Si vuole gestire l'aggiornamento dei dati attivi di player e manager dopo un match, diminuendo il numero di contratti di un'unità, eliminando l'effetto di eventuali carte consumabili training attive e simulando la diminuzione di forma fisica basandosi dal ruolo ricoperto dal giocatore. Ruoli che richiedono notevole sforzo fisico causano maggiore diminuzione di fitness, in particolare:

   - Portiere -> da 2 a 3;
   - Difensore centrale -> da 3 a 5;
   - Centrocampista centrale, centrocampista offensivo centrale, centrocampista difensivo centrale, prima punta, seconda punta, punta sinistra, punta destra -> da 5 a 8;
   - Terzino sinistro, terzino destro, esterno sinistro, esterno destro, ala sinistra, ala destra, ala arretrata sinistra, ala arretrata destra -> da 10 a 8;

   Soluzione: Trigger **Player Update After Match**.

Capitolo 8

Normalizzazione

8.1 Verifica di normalità

Una forma normale è una proprietà che, se verificata, garantisce qualità, ovvero assenza di difetti quali ridondanze di dati ed anomalie di aggiornamento, cancellazione e inserimento. La tecnica che permette di trasformare schemi non normalizzati in schemi che soddisfano una forma normale è detta normalizzazione.

8.2 Prima forma normale

La conformità alla prima forma normale è banalmente garantita dall'assenza di attributi non atomici o multivalore, adottando la convenzione che vuole che gli attributi di tipo data siano considerati atomici e non strutturati. Per tanto, la base risulta conforme alla 1NF.

8.3 Seconda forma normale

La conformità alla seconda forma normale è garantita dalla dipendenza completa di ogni attributo non primo, ovvero non appartenente a nessuna chiave, da ogni chiave composta e dalla conformità alla prima forma normale. Per tanto, la base risulta conforme alla 2NF.

8.4 Terza forma normale

La conformità alla terza forma normale è garantita dalla totale indipendenza di ogni attributo non primo da altri attributi non primi e dalla conformità alla seconda forma normale. Per tanto, la base risulta conforme alla 3NF.

8.5 Forma normale di Boys e Codd

La conformità alla forma normale di Boyce e Codd è garantita dalla conformità alla terza forma normale alla quale si aggiunge la restrizione che gli attributi debbano dipendere funzionalmente solo dalla chiave. Per tanto, si conclude affermando che la base risulta essere conforme anche alla BCNF.

Capitolo 9

Volumi e classificazione errori

9.1 Volumi

Al momento in cui è stato effettuata l'operazione di dump della base di dati, i volumi si presentavano come di seguito.

TABELLA	TIPO	VOLUME
user fut	E	2
club	ED	2
division rivals	E	10
pack	E	16
pack purchase	ED	20
is found	A	400
transaction	ED	20
club card	ED	400
club item	E	12
consumable	E	140
manager	E	20
player	E	1500
stops	ED	5
squad	ED	2
is part of	A	30
match	ED	50
manages	A	2
active data manager	ED	50
active data player	ED	200

Tabella 9.1: Tavola dei volumi. E sta per *entità*, ED per *entità debole*, A per *associazione*. Le quantità riportate sono delle stime.

9.2 Classificazione errori

Gli errori riscontrabili nel caso di utilizzo improprio di alcune procedure sono identificati da un particolare codice numerico.

NUMERO	DESCRIZIONE
-20001	dati non trovati
-20002	dati forniti in ingresso non validi
-20003	limiti massimi superati
-20004	tipo di carta, o categoria della carta non valida
-20005	oggetti uguali (carte, club, utente, etc.)
-20006	limiti minimi non soddisfatti
-20007	la verifica della squadra è fallita

Tabella 9.2: Tabella classificazione errori

Capitolo 10

Implementazione in Oracle 11g XE

10.1 Drop di oggetti già esistenti

```
--TABLE
DROP TABLE USER_FUT CASCADE CONSTRAINTS;
DROP TABLE CLUB CASCADE CONSTRAINTS;
DROP TABLE PACK CASCADE CONSTRAINTS;
DROP TABLE DIVISION_RIVALS CASCADE CONSTRAINTS;
DROP TABLE TRANSACTION CASCADE CONSTRAINTS;
DROP TABLE IS_PART_OF CASCADE CONSTRAINTS;
DROP TABLE SQUAD CASCADE CONSTRAINTS;
DROP TABLE MATCH CASCADE CONSTRAINTS;
DROP TABLE STOPS CASCADE CONSTRAINTS;
DROP TABLE CONSUMABLE CASCADE CONSTRAINTS;
DROP TABLE CLUB_ITEM CASCADE CONSTRAINTS;
DROP TABLE MANAGER CASCADE CONSTRAINTS;
DROP TABLE MANAGES CASCADE CONSTRAINTS;
DROP TABLE PLAYER CASCADE CONSTRAINTS;
DROP TABLE PACK_PURCHASE CASCADE CONSTRAINTS;
DROP TABLE IS_FOUND CASCADE CONSTRAINTS;
DROP TABLE CLUB_CARD CASCADE CONSTRAINTS;
DROP TABLE ACTIVE_DATA_PLAYER CASCADE CONSTRAINTS;
DROP TABLE ACTIVE_DATA_MANAGER CASCADE CONSTRAINTS;
DROP TABLE TMP_MATCH;

--PROCEDURE
DROP PROCEDURE ADD_CLUB_ITEM;
DROP PROCEDURE ADD_CONTRACT;
DROP PROCEDURE ADD_FITNESS;
DROP PROCEDURE ADD_HEALING;
DROP PROCEDURE ADD_MANAGER;
DROP PROCEDURE REMOVE_MANAGER;
DROP PROCEDURE ADD_MANAGER_LEAGUE;
DROP PROCEDURE ADD_PLAYER;
DROP PROCEDURE ADD_PPOSITIONING;
DROP PROCEDURE ADD_TRAINING;
DROP PROCEDURE BUY_CARD;
DROP PROCEDURE SELL_CARD;
DROP PROCEDURE CLUB_CREATION;
DROP PROCEDURE CLUB_DELETE;
DROP PROCEDURE INSERT_MATCH;
DROP PROCEDURE PACK_OPENING;
```

```
DROP PROCEDURE SQUAD_CREATION;
DROP PROCEDURE CHANGE_PLAYER;
DROP PROCEDURE CHECK_DURATION;
DROP PROCEDURE GIFT;
DROP PROCEDURE NUMBER_OF_CARDS;

--FUNCTION
DROP FUNCTION GET_CONTRACT;
DROP FUNCTION GET_SQUAD_RATING;
DROP FUNCTION IDX_POSITION;
DROP FUNCTION MANAGER_CHEMISTRY;
DROP FUNCTION MODULE_POSITIONS;
DROP FUNCTION IS_ADMIN;
DROP FUNCTION GET_SQUAD;
DROP FUNCTION SEARCH_CARD_FOR_SALE;
DROP FUNCTION CARD_IN_CLUB;
DROP FUNCTION IS_ADMIN;

--ADMIN FUNCTION
DROP FUNCTION GET_ACTIVE_CLUB_ITEM_F;
DROP FUNCTION GET_CLUB_ITEM_F;
DROP FUNCTION GET_COACH_F;
DROP FUNCTION GET_CONSUMABLES_F;
DROP FUNCTION GET_MANAGER_F;
DROP FUNCTION GET_MANAGES_F;
DROP FUNCTION GET_MATCH_F;
DROP FUNCTION GET_PACK_PURCHASE_F;
DROP FUNCTION GET_PLAYERS_F;

--TRIGGER
DROP TRIGGER CHECK_PLAYER_FITNESS;
DROP TRIGGER DIVISION_UPDATE;
DROP TRIGGER EMAIL2MD5;
DROP TRIGGER INSERT_USER;
DROP TRIGGER PLAYER_IN_SQUAD;
DROP TRIGGER PLAYER_UPDATE_AFTER_MATCH;
DROP TRIGGER VERIFY_SQUAD;
DROP TRIGGER STOPS_RANDOMIZER;

--VIEW
DROP VIEW GET_ACTIVE_CLUB_ITEMS;
DROP VIEW GET_AVAILABLE_PACKS;
DROP VIEW GET_CLUB_INFO;
DROP VIEW GET_CLUB_ITEMS;
DROP VIEW GET_CONSUMABLES;
DROP VIEW GET_MANAGERS;
DROP VIEW GET_MANAGES;
DROP VIEW GET_MATCH;
DROP VIEW GET_PACK_PURCHASES;
DROP VIEW GET_PLAYERS;
DROP VIEW GET_PLAYERS_FOR_SALE;
DROP VIEW GET_PLAYERS_STOPS;

--SEQUENCE
DROP SEQUENCE MATCH_ID_SEQ;
--TYPE
DROP TYPE ARRAY_POS_T;
```

10.2 Creazione tabelle (DDL)

```
--TABLE USER_FUT
CREATE TABLE USER_FUT (
    NICKNAME  VARCHAR2(16) PRIMARY KEY,
    EMAIL     VARCHAR2(64) NOT NULL
);

--TABLE CLUB
CREATE TABLE CLUB (
    CLUB_NAME               VARCHAR2(16) PRIMARY KEY,
    USER_NICK               VARCHAR2(16) UNIQUE NOT NULL,
    DR_POINTS               NUMBER(5, 0) DEFAULT 0,
    CREDITS                 NUMBER(11, 0) DEFAULT 0,
    DIVISION_NUMBER         NUMBER(2, 0),
    ACTIVE_BADGE_CODE       VARCHAR(8),
    ACTIVE_STADIUM_CODE     VARCHAR(8),
    ACTIVE_BALL_CODE        VARCHAR(8),
    ACTIVE_FIRST_KIT_CODE   VARCHAR(8),
    ACTIVE_SECOND_KIT_CODE  VARCHAR(8),
    CONSTRAINT FK_CREATES FOREIGN KEY ( USER_NICK )
        REFERENCES USER_FUT ( NICKNAME )
            ON DELETE CASCADE
);

--TABLE DIVISION_RIVALS
CREATE TABLE DIVISION_RIVALS (
    DIVISION_NUMBER  NUMBER(2, 0) PRIMARY KEY,
    THRESHOLD        NUMBER(4, 0),
    WIN_CREDITS      NUMBER(4, 0),
    POINTS           NUMBER(4, 0)
);

ALTER TABLE CLUB
    ADD CONSTRAINT FK_DIVISION_NUMBER FOREIGN KEY (
       DIVISION_NUMBER )
        REFERENCES DIVISION_RIVALS ( DIVISION_NUMBER )
            ON DELETE CASCADE;

--TABLE PACK
CREATE TABLE PACK (
    PACK_NAME        VARCHAR2(32) PRIMARY KEY,
    PLAYERS          NUMBER(2, 0) DEFAULT 0,
    CONSUMABLES      NUMBER(2, 0) DEFAULT 0,
    CLUB_ITEMS       NUMBER(2, 0) DEFAULT 0,
    STAFF            NUMBER(2, 0) DEFAULT 0,
    GOLD_QUANTITY    NUMBER(2, 0) DEFAULT 0,
    SILVER_QUANTITY  NUMBER(2, 0) DEFAULT 0,
    BRONZE_QUANTITY  NUMBER(2, 0) DEFAULT 0,
    AVAILABLE         CHAR(1) NOT NULL,
    PRICE            NUMBER(7, 0) NOT NULL,
    CONSTRAINT CHECK_AVAIABLE CHECK ( AVAILABLE IN ( '0', '1' )
        )
);

--TABLE PACK_PURCHASE
CREATE TABLE PACK_PURCHASE (
```

```
    PURCHASE_ID        VARCHAR(8) PRIMARY KEY,
    BUYING_CLUB_NAME   VARCHAR2(16) NOT NULL,
    BUYING_PACK_NAME   VARCHAR2(32) NOT NULL,
    P_DATE             DATE,
    CONSTRAINT FK_DOES FOREIGN KEY ( BUYING_CLUB_NAME )
        REFERENCES CLUB ( CLUB_NAME )
            ON DELETE CASCADE,
    CONSTRAINT FK_IS_BOUGHT FOREIGN KEY ( BUYING_PACK_NAME )
        REFERENCES PACK ( PACK_NAME )
            ON DELETE SET NULL
);

--TABLE CLUB_CARD
CREATE TABLE CLUB_CARD (
    CARD_CODE       VARCHAR(8) PRIMARY KEY,
    PLAYER_ID       NUMBER(5, 0),
    CONSUMABLE_ID   NUMBER(5, 0),
    CLUB_ITEM_ID    NUMBER(5, 0),
    MANAGER_ID      NUMBER(5, 0),
    COACH_ID        NUMBER(5, 0),
    TRADABLE        CHAR(1),
    CONSTRAINT CHECK_TRADABLE CHECK ( TRADABLE IN ( '0', '1' )
       )
);

--TABLE TRANSACTION
CREATE TABLE TRANSACTION (
    TRANSACTION_ID          VARCHAR(8) PRIMARY KEY,
    T_CARD_CODE             VARCHAR(8) NOT NULL, --FK
    T_DATE                  DATE,
    TRANSITION_S_CLUB_NAME  VARCHAR2(16) NOT NULL,
    TRANSITION_B_CLUB_NAME  VARCHAR2(16),
    PRICE                   NUMBER(11, 0) NOT NULL,
    DURATION                NUMBER(2, 0) NOT NULL,
    CONSTRAINT FK_SOLDS FOREIGN KEY ( TRANSITION_S_CLUB_NAME )
        REFERENCES CLUB ( CLUB_NAME )
            ON DELETE SET NULL,
    CONSTRAINT FK_BUYS FOREIGN KEY ( TRANSITION_B_CLUB_NAME )
        REFERENCES CLUB ( CLUB_NAME )
            ON DELETE SET NULL,
    CONSTRAINT FK_IS_PART_OF_T FOREIGN KEY ( T_CARD_CODE )
        REFERENCES CLUB_CARD ( CARD_CODE )
            ON DELETE CASCADE
);

--TABLE IS_FOUND
CREATE TABLE IS_FOUND (
    P_ID       VARCHAR(8), --PK FK
    CARD_CODE  VARCHAR(8), --PK FK
    CONSTRAINT FK_PURCHASE_ID FOREIGN KEY ( P_ID )
        REFERENCES PACK_PURCHASE ( PURCHASE_ID )
            ON DELETE SET NULL,
    CONSTRAINT FK_IS_FOUND FOREIGN KEY ( CARD_CODE )
        REFERENCES CLUB_CARD ( CARD_CODE )
            ON DELETE CASCADE,
    CONSTRAINT PK_IS_FOUND PRIMARY KEY ( P_ID,
                                         CARD_CODE )
```

```
);

--TABLE SQUAD
CREATE TABLE SQUAD (
    NAME              VARCHAR2(16) PRIMARY KEY,--PK,
    SQUAD_CLUB_NAME   VARCHAR2(16),-- FK
    MODULES           VARCHAR2(10) DEFAULT '4-4-2',
    SQUAD_RATING      NUMBER(2, 0) DEFAULT 0,
    SQUAD_CHEMISTRY   NUMBER(3, 0) DEFAULT 0,
    CONSTRAINT FK_HAS FOREIGN KEY ( SQUAD_CLUB_NAME )
        REFERENCES CLUB ( CLUB_NAME )
            ON DELETE CASCADE,
    CONSTRAINT CHECK_S_CHEMISTRY CHECK ( SQUAD_CHEMISTRY
       BETWEEN 0 AND 101 ),
    CONSTRAINT CHECK_MODULES CHECK ( MODULES IN ('3-1-4-2','
       3-4-1-2','3-4-3','3-5-2','4-1-2-1-2','4-1-3-2','4-1-4-1'
       ,'4-2-2-2','4-2-3-1','4-2-4','4-3-1-2','4-3-2-1','4-3-3'
       ,'4-4-1-1','4-5-1','4-4-2','5-2-1-2','5-2-2-1','5-3-2','
       5-4-1','5-2-1-2' ) )
);

--TABLE IS_PART_OF
CREATE TABLE IS_PART_OF (
   PLAYER_CARD_CODE  VARCHAR(8), --PK FK
   SQUAD_NAME        VARCHAR2(16) --PK FK,
   PLAYER_POSITION   VARCHAR2(3),
   PLAYER_CHEMISTRY  NUMBER(2, 0),
   PLAYER_RATING     NUMBER(2, 0),
   HOLDER            CHAR(1),
   CONSTRAINT PK_IS_PART_OF PRIMARY KEY ( PLAYER_CARD_CODE,
                                          SQUAD_NAME ),
   CONSTRAINT CHECK_HOLDER CHECK ( HOLDER IN ( '0', '1' ) ),
   CONSTRAINT FK_IS_PART_OF FOREIGN KEY ( PLAYER_CARD_CODE )
      REFERENCES CLUB_CARD ( CARD_CODE )
         ON DELETE CASCADE,
    CONSTRAINT FK_HAS FOREIGN KEY ( SQUAD_NAME )
        REFERENCES SQUAD ( NAME )
            ON DELETE CASCADE

);

-- MANAGER MANAGES SQUAD
CREATE TABLE MANAGES (
    MANAGER_CARD_CODE  VARCHAR(8), --PK FK
    SQUAD_NAME         VARCHAR2(16) --PK FK,
    CONSTRAINT PK_MANAGES PRIMARY KEY ( MANAGER_CARD_CODE,
                                        SQUAD_NAME ),
    CONSTRAINT FK_MANAGER_CARD_CODE FOREIGN KEY (
       MANAGER_CARD_CODE )
        REFERENCES CLUB_CARD ( CARD_CODE )
            ON DELETE CASCADE,
    CONSTRAINT FK_MANAGES_SQUAD FOREIGN KEY ( SQUAD_CLUB_NAME )
        REFERENCES SQUAD ( NAME )
        ON DELETE CASCADE

);
```

```
--TABLE ACTIVE_DATA_PLAYER
CREATE TABLE ACTIVE_DATA_PLAYER (
    P_CARD_CODE       VARCHAR(8) PRIMARY KEY,  --PK FK
    CONTRACTS         NUMBER(2, 0) DEFAULT 7,
    PLAYER_FITNESS    NUMBER(3, 0) DEFAULT 99,
    ACTIVE_TRAINING   VARCHAR(8),
    CONSTRAINT FK_CC_P FOREIGN KEY ( P_CARD_CODE )
        REFERENCES CLUB_CARD ( CARD_CODE )
            ON DELETE CASCADE,
    CONSTRAINT CHECK_FITNESS CHECK ( PLAYER_FITNESS BETWEEN 1
       AND 100 )
);

CREATE TABLE ACTIVE_DATA_MANAGER (
    M_CARD_CODE      VARCHAR(8) PRIMARY KEY,  --PK FK
    MANAGER_LEAGUE   VARCHAR2(32),
    CONTRACTS        NUMBER(2, 0) DEFAULT 7,
    CONSTRAINT FK_CC_M FOREIGN KEY ( M_CARD_CODE )
        REFERENCES CLUB_CARD ( CARD_CODE )
            ON DELETE CASCADE
);

--TABLE MATCH
CREATE TABLE MATCH (
    MATCH_ID              NUMBER(5, 0) PRIMARY KEY,
    HOME_SQUAD_NAME       VARCHAR2(16), --FK
    VISITORS_SQUAD_NAME   VARCHAR2(16), --FK
    M_DATE                DATE,
    RESULTS               CHAR(4),
    HOME_POINTS           NUMBER(3, 0),
    VISITORS_POINTS       NUMBER(3, 0),
    HOME_CREDITS          NUMBER(4, 0),
    VISITORS_CREDITS      NUMBER(4, 0),
    CONSTRAINT FK_HOME_SQUAD_NAME FOREIGN KEY ( HOME_SQUAD_NAME
        )
        REFERENCES SQUAD ( NAME )
            ON DELETE CASCADE,
    CONSTRAINT FK_VISITORS_SQUAD_NAME FOREIGN KEY (
       VISITORS_SQUAD_NAME )
        REFERENCES SQUAD ( NAME )
            ON DELETE CASCADE
);

--TABLE CONSUMABLE
CREATE TABLE CONSUMABLE (
    CARD_ID           NUMBER(5, 0) PRIMARY KEY,
    RARITY_NAME       VARCHAR(8) NOT NULL,
    TYPE              VARCHAR2(16) NOT NULL,
    CATEGORY          VARCHAR2(24) NOT NULL,
    AMOUNT            NUMBER(3, 0) DEFAULT 0,
    BRONZE_CONTRACT   NUMBER(2, 0) DEFAULT 0,
    SILVER_CONTRACT   NUMBER(2, 0) DEFAULT 0,
    GOLD_CONTRACT     NUMBER(2, 0) DEFAULT 0,
    MIN_PRICE         NUMBER(4, 0) DEFAULT 150,
    MAX_PRICE         NUMBER(5, 0) DEFAULT 5000,
    CONSTRAINT CHECK_CATEGORY CHECK ( CATEGORY IN ('Contract','
       Healing','Fitness','GKTraining','Training','Positioning'
```

```
        ,'Manager League') )
);

--FK CON CLUB_CARD
ALTER TABLE CLUB_CARD
    ADD CONSTRAINT FK_CONSUMABLE_ID FOREIGN KEY ( CONSUMABLE_ID
        )
        REFERENCES CONSUMABLE ( CARD_ID )
            ON DELETE CASCADE;

--TABLE MANAGER
CREATE TABLE MANAGER (
    CARD_ID       NUMBER(5, 0) PRIMARY KEY,
    F_NAME        VARCHAR2(16) NOT NULL,
    S_NAME        VARCHAR2(16) NOT NULL,
    NATIONALITY   VARCHAR2(16) NOT NULL,
    LEAGUE_NAME   VARCHAR2(32) NOT NULL,
    RARITY_NAME   VARCHAR(8) NOT NULL,
    MIN_PRICE     NUMBER(8, 0) DEFAULT 0,
    MAX_PRICE     NUMBER(11, 0) DEFAULT 0
);

--FK CON CLUB_CARD
ALTER TABLE CLUB_CARD
    ADD CONSTRAINT FK_MANAGER_ID FOREIGN KEY ( MANAGER_ID )
        REFERENCES MANAGER ( CARD_ID )
            ON DELETE CASCADE;

--TABLE CLUB_ITEM
CREATE TABLE CLUB_ITEM (
    CARD_ID         NUMBER(5, 0) PRIMARY KEY,
    CATEGORY_ITEM   VARCHAR2(8) NOT NULL,
    TEAM_NAME       VARCHAR2(32),
    RARITY_NAME     VARCHAR(8) NOT NULL,
    MIN_PRICE       NUMBER(8, 0) DEFAULT 0,
    MAX_PRICE       NUMBER(11, 0) DEFAULT 0,
    CONSTRAINT CHECK_CATEGORY_ITEM CHECK ( CATEGORY_ITEM IN ( '
       Kit', 'Badge', 'Ball', 'Stadium' ) )
);

--FK CON CLUB_CARD
ALTER TABLE CLUB_CARD
    ADD CONSTRAINT FK_CLUB_ITEM_ID FOREIGN KEY ( CLUB_ITEM_ID )
        REFERENCES CLUB_ITEM ( CARD_ID )
            ON DELETE CASCADE;

--FK CON CLUB
ALTER TABLE CLUB
    ADD CONSTRAINT FK_ACTIVE_BADGE_CODE FOREIGN KEY (
       ACTIVE_BADGE_CODE )
        REFERENCES CLUB_CARD ( CARD_CODE )
            ON DELETE SET NULL;

ALTER TABLE CLUB
    ADD CONSTRAINT FK_ACTIVE_STADIUM_CODE FOREIGN KEY (
       ACTIVE_STADIUM_CODE )
        REFERENCES CLUB_CARD ( CARD_CODE )
```

```
            ON DELETE SET NULL;

ALTER TABLE CLUB
    ADD CONSTRAINT FK_ACTIVE_BALL_CODE FOREIGN KEY (
       ACTIVE_BALL_CODE )
        REFERENCES CLUB_CARD ( CARD_CODE )
            ON DELETE SET NULL;

ALTER TABLE CLUB
    ADD CONSTRAINT FK_ACTIVE_FIRST_KIT_CODE FOREIGN KEY (
       ACTIVE_FIRST_KIT_CODE )
        REFERENCES CLUB_CARD ( CARD_CODE )
            ON DELETE SET NULL;

ALTER TABLE CLUB
    ADD CONSTRAINT FK_ACTIVE_SECOND_KIT_CODE FOREIGN KEY (
       ACTIVE_SECOND_KIT_CODE )
        REFERENCES CLUB_CARD ( CARD_CODE )
            ON DELETE SET NULL;

--TABLE PLAYER
CREATE TABLE PLAYER (
    CARD_ID                NUMBER(5, 0) PRIMARY KEY,
    PLAYER_NAME            VARCHAR2(32) NOT NULL,
    PLAYER_EXTENDED_NAME   VARCHAR2(56) NOT NULL,
    RARITY_NAME            VARCHAR2(32) NOT NULL,
    VERSION_NAME          VARCHAR2(32) DEFAULT 'Normal',
    OVERALL                NUMBER(2, 0) NOT NULL,
    TEAM_NAME              VARCHAR2(44) NOT NULL,
    LEAGUE_NAME            VARCHAR2(44) NOT NULL,
    NATIONALITY            VARCHAR2(32) NOT NULL,
    POSITION               VARCHAR2(3) NOT NULL,
    B_DATE                 VARCHAR2(16) NOT NULL,
    PAC                    NUMBER(2, 0),
    DRI                    NUMBER(2, 0),
    SHO                    NUMBER(2, 0),
    PAS                    NUMBER(2, 0),
    DEF                    NUMBER(2, 0),
    PHY                    NUMBER(2, 0),
    DIV                    NUMBER(2, 0),
    REF                    NUMBER(2, 0),
    HAN                    NUMBER(2, 0),
    SPD                    NUMBER(2, 0),
    KIC                    NUMBER(2, 0),
    POS                    NUMBER(2, 0),
    PREF_FOOT              VARCHAR(5) NOT NULL,
    WEAK_FOOT              NUMBER(1, 0) NOT NULL,
    SKILL                  NUMBER(1, 0),
    CB                     NUMBER(2, 0),
    RB                     NUMBER(2, 0),
    LB                     NUMBER(2, 0),
    RWB                    NUMBER(2, 0),
    LWB                    NUMBER(2, 0),
    CDM                    NUMBER(2, 0),
    CM                     NUMBER(2, 0),
    RM                     NUMBER(2, 0),
    LM                     NUMBER(2, 0),
```

```
    CAM                     NUMBER(2, 0),
    CF                      NUMBER(2, 0),
    RF                      NUMBER(2, 0),
    LF                      NUMBER(2, 0),
    RW                      NUMBER(2, 0),
    LW                      NUMBER(2, 0),
    ST                      NUMBER(2, 0),
    MIN_PRICE               NUMBER(8, 0) DEFAULT 0,
    MAX_PRICE               NUMBER(11, 0) DEFAULT 0,
    CONSTRAINT CHECK_WEAK_FOOT CHECK ( WEAK_FOOT IN ( 1, 2, 3,
       4, 5 ) ),
    CONSTRAINT CHECK_SKILL CHECK ( SKILL IN ( 1, 2, 3, 4, 5 ) )
       ,
    CONSTRAINT CHECK_OVERALL CHECK ( OVERALL BETWEEN 1 AND 99 )
       ,
    CONSTRAINT CHECK_PREF_FOOT CHECK ( PREF_FOOT IN ( 'Left', '
       Right' ) ),
    CONSTRAINT CHECK_POSITION CHECK ( POSITION IN ('GK','RB','
       RWB','CB','LB','LWB','CDM','CM','CAM','RM','RW','LM','LW
       ','CF','RF','LF','ST'))
);

--FK CON CLUB_CARD
ALTER TABLE CLUB_CARD
    ADD CONSTRAINT FK_PLAYER_ID FOREIGN KEY ( PLAYER_ID )
        REFERENCES PLAYER ( CARD_ID )
            ON DELETE CASCADE;

--TABLE STOPS
CREATE TABLE STOPS (
    S_DATE        DATE,
    BODY_PART     VARCHAR2(16),
    DAYS          NUMBER(1, 0),
    P_CARD_CODE   VARCHAR(8), --PK FK
    CONSTRAINT FK_SUFFERS FOREIGN KEY ( P_CARD_CODE )
        REFERENCES CLUB_CARD ( CARD_CODE )
            ON DELETE CASCADE,
    CONSTRAINT PK_STOPS PRIMARY KEY ( S_DATE,
                                      P_CARD_CODE ),
    CONSTRAINT CHECK_B_PART CHECK (BODY_PART IN ('Foot', 'Leg',
        'Knee', 'Arm', 'UpperBody','Head') ),
    CONSTRAINT CHECK_DAYS CHECK (DAYS BETWEEN 1 AND 6)
);

CREATE TABLE TMP_MATCH (
    MATCH_ID              NUMBER(3, 0),
    HOME_SQUAD_NAME       VARCHAR2(16),
    VISITORS_SQUAD_NAME   VARCHAR2(16),
);
```

10.3 Popolamento delle tabelle (DML)

Lo script di seguito non comprende le righe rigurdanti la tabella **Player** che sono 1500.

```
--CLUB_ITEM
INSERT INTO CLUB_ITEM VALUES (1,'Kit','Juventus','Gold'
    ,250,5000);
INSERT INTO CLUB_ITEM VALUES (2,'Kit','Crotone','Silver'
    ,350,5000);
INSERT INTO CLUB_ITEM VALUES (3,'Kit','Vibonese','Bronze'
    ,350,5000);
INSERT INTO CLUB_ITEM VALUES (4,'Ball','Cosmos','Gold'
    ,350,5000);
INSERT INTO CLUB_ITEM VALUES (5,'Ball','Adidas UCL Finale 18','
    Silver',450,5000);
INSERT INTO CLUB_ITEM VALUES (6,'Ball','Molten UEL','Bronze'
    ,450,5000);
INSERT INTO CLUB_ITEM VALUES (7,'Badge','Valencia CF','Gold'
    ,650,5000);
INSERT INTO CLUB_ITEM VALUES (8,'Badge','LOSC Lille','Silver'
    ,650,5000);
INSERT INTO CLUB_ITEM VALUES (9,'Badge','Brescia','Bronze'
    ,550,5000);
INSERT INTO CLUB_ITEM VALUES (10,'Stadium','Old Trafford','Gold
    ',150,5000);
INSERT INTO CLUB_ITEM VALUES (11,'Stadium','San Siro','Gold'
    ,150,5000);
INSERT INTO CLUB_ITEM VALUES (12,'Stadium','Anfield','Silver'
    ,150,5000);

--DIVISION_RIVALS
INSERT INTO DIVISION_RIVALS VALUES (1,2400,500,500);
INSERT INTO DIVISION_RIVALS VALUES (2,2200,400,250);
INSERT INTO DIVISION_RIVALS VALUES (3,1900,300,125);
INSERT INTO DIVISION_RIVALS VALUES (4,1600,250,80);
INSERT INTO DIVISION_RIVALS VALUES (5,1200,200,65);
INSERT INTO DIVISION_RIVALS VALUES (6,800,150,50);
INSERT INTO DIVISION_RIVALS VALUES (7,500,130,40);
INSERT INTO DIVISION_RIVALS VALUES (8,250,110,30);
INSERT INTO DIVISION_RIVALS VALUES (9,100,80,25);
INSERT INTO DIVISION_RIVALS VALUES (10,10,50,20);

--MANAGER
INSERT INTO MANAGER VALUES (1,'Jurgen','Kloop','Germany','ENG1'
    ,'Gold',250,5000);
INSERT INTO MANAGER VALUES (2,'Roy','Hodgson','Englad','ENG1','
    Gold',250,5000);
INSERT INTO MANAGER VALUES (3,'Zinedine','Zidane','France','
    ESP1','Gold',250,5000);
INSERT INTO MANAGER VALUES (4,'Massimo','Allegri','Italy','ITA1
    ','Gold',450,5000);
INSERT INTO MANAGER VALUES (5,'Jose','Mourinho','Portugal','
    ENG1','Gold',450,5000);
INSERT INTO MANAGER VALUES (6,'Ernesto','Valverde','Spain','
    ESP1','Gold',450,5000);
INSERT INTO MANAGER VALUES (7,'Gian Piero','Gasperini','Italy',
    'ITA1','Gold',450,5000);
```

```
INSERT INTO MANAGER VALUES (8,'Davide','Nicola','Italy','ITA1',
    'Silver',550,5000);
INSERT INTO MANAGER VALUES (9,'Carlo','Ancelotti','Italy','ENG1
    ','Silver',550,5000);
INSERT INTO MANAGER VALUES (10,'Gino','Lettieri','Italy','POL1'
    ,'Bronze',550,5000);
INSERT INTO MANAGER VALUES (11,'Maurizio','Sarri','Italy','ITA1
    ','Gold',550,5000);
INSERT INTO MANAGER VALUES (12,'Fabio','Cannavaro','Italy','
    DEN1','Bronze',650,5000);
INSERT INTO MANAGER VALUES (13,'Mauricio','Pochettino','
    Argentina','ENG1','Gold',650,5000);
INSERT INTO MANAGER VALUES (14,'Thomas','Tuchel','Germany','
    FRA1','Silver',150,5000);
INSERT INTO MANAGER VALUES (15,'Sergio','Conceicao','Portugal',
    'POR1','Bronze',150,5000);
INSERT INTO MANAGER VALUES (16,'Robert','Moreno','Spain','FRA1'
    ,'Bronze',150,5000);
INSERT INTO MANAGER VALUES (17,'Sergej','Semak','Russian','RUS1
    ','Bronze',150,5000);
INSERT INTO MANAGER VALUES (18,'Toshio','Matsuura','Japan','
    JAP1','Bronze',150,5000);
INSERT INTO MANAGER VALUES (19,'Takayuki','Morimoto','Japan','
    JAP1','Gold',150,5000);
INSERT INTO MANAGER VALUES (20,'Giacomo','Modica','Italy','UKR1
    ','Silver',150,5000);

--PACK
INSERT INTO PACK VALUES ('Bronze Pack',2,3,4,3,0,0,12,1,400);
INSERT INTO PACK VALUES ('Premium Bronze Pack'
    ,4,2,4,2,0,0,12,1,750);
INSERT INTO PACK VALUES ('Silver Pack',2,3,4,3,0,12,0,1,2500);
INSERT INTO PACK VALUES ('Premium Silver Pack'
    ,4,2,4,2,0,12,0,1,3750);
INSERT INTO PACK VALUES ('Gold Pack',2,3,4,3,12,0,0,1,5000);
INSERT INTO PACK VALUES ('Premium Gold Pack'
    ,4,2,4,2,12,0,0,1,7500);
INSERT INTO PACK VALUES ('Bronze Player Pack'
    ,12,0,0,0,0,0,12,1,1250);
INSERT INTO PACK VALUES ('Silver Player Pack'
    ,12,0,0,0,0,12,0,0,5000);
INSERT INTO PACK VALUES ('Gold Player Pack'
    ,12,0,0,0,12,0,0,1,12500);
INSERT INTO PACK VALUES ('Consumable Pack'
    ,0,12,0,0,12,0,0,1,3000);
INSERT INTO PACK VALUES ('Ultimate Pack'
    ,30,0,0,0,30,0,0,0,125000);
INSERT INTO PACK VALUES ('Rare Mega Pack'
    ,9,13,4,4,30,0,0,1,55000);
INSERT INTO PACK VALUES ('Jumbo Gold Pack'
    ,6,6,6,6,24,0,0,1,10000);
INSERT INTO PACK VALUES ('TOTY',30,0,0,0,20,10,0,1,125000);
INSERT INTO PACK VALUES ('Rainbow Gold Pack'
    ,10,4,2,0,10,4,2,1,10000);
INSERT INTO PACK VALUES ('Lol Silver Pack'
    ,10,4,2,0,4,10,2,1,10000);
```

```
--CONSUMABLES
INSERT INTO CONSUMABLE VALUES (1,'Bronze','Player','Contract'
    ,0,8,2,1,450,5000);
INSERT INTO CONSUMABLE VALUES (2,'Silver','Player','Contract'
    ,0,10,10,8,450,5000);
INSERT INTO CONSUMABLE VALUES (3,'Gold','Player','Contract'
    ,0,15,11,13,450,5000);
INSERT INTO CONSUMABLE VALUES (4,'Bronze','Player','Contract'
    ,0,15,6,3,450,5000);
INSERT INTO CONSUMABLE VALUES (5,'Silver','Player','Contract'
    ,0,20,24,18,450,5000);
INSERT INTO CONSUMABLE VALUES (6,'Gold', 'Player','Contract'
    ,0,28,24,28,450,5000);
INSERT INTO CONSUMABLE VALUES (7,'Bronze','Manager','Contract'
    ,0,8,2,1,450,5000);
INSERT INTO CONSUMABLE VALUES (8,'Silver','Manager','Contract'
    ,0,8,10,8,450,5000);
INSERT INTO CONSUMABLE VALUES (9,'Gold','Manager','Contract'
    ,0,11,11,13,450,5000);
INSERT INTO CONSUMABLE VALUES (10,'Bronze','Manager','Contract'
    ,0,15,6,3,450,5000);
INSERT INTO CONSUMABLE VALUES (11,'Silver','Manager','Contract'
    ,0,18,24,18,450,5000);
INSERT INTO CONSUMABLE VALUES (12,'Gold','Manager','Contract'
    ,0,24,24,28,450,5000);
INSERT INTO CONSUMABLE VALUES (13,'Bronze','Player','Fitness'
    ,20,0,0,0,450,5000);
INSERT INTO CONSUMABLE VALUES (14,'Silver','Player','Fitness'
    ,40,0,0,0,450,5000);
INSERT INTO CONSUMABLE VALUES (15,'Gold','Player','Fitness'
    ,60,0,0,0,450,5000);
INSERT INTO CONSUMABLE VALUES (16,'Bronze','Squad','Fitness'
    ,10,0,0,0,450,5000);
INSERT INTO CONSUMABLE VALUES (17,'Silver','Squad','Fitness'
    ,20,0,0,0,450,5000);
INSERT INTO CONSUMABLE VALUES (18,'Gold','Squad','Fitness'
    ,30,0,0,0,450,5000);
INSERT INTO CONSUMABLE VALUES (19,'Bronze','Head','Healing'
    ,1,0,0,0,450,5000);
INSERT INTO CONSUMABLE VALUES (20,'Silver','Head','Healing'
    ,2,0,0,0,450,5000);
INSERT INTO CONSUMABLE VALUES (21,'Gold','Head','Healing'
    ,5,0,0,0,450,5000);
INSERT INTO CONSUMABLE VALUES (22,'Bronze','UpperBody','Healing
    ',1,0,0,0,450,5000);
INSERT INTO CONSUMABLE VALUES (23,'Silver','UpperBody','Healing
    ',2,0,0,0,450,5000);
INSERT INTO CONSUMABLE VALUES (24,'Gold','UpperBody','Healing'
    ,5,0,0,0,450,5000);
INSERT INTO CONSUMABLE VALUES (25,'Bronze','Arm','Healing'
    ,1,0,0,0,450,5000);
INSERT INTO CONSUMABLE VALUES (26,'Silver','Arm','Healing'
    ,2,0,0,0,450,5000);
INSERT INTO CONSUMABLE VALUES (27,'Gold','Arm','Healing'
    ,5,0,0,0,450,5000);
INSERT INTO CONSUMABLE VALUES (28,'Bronze','Knee','Healing'
    ,1,0,0,0,450,5000);
```

```
INSERT INTO CONSUMABLE VALUES (29,'Silver','Knee','Healing'
    ,2,0,0,0,450,5000);
INSERT INTO CONSUMABLE VALUES (30,'Gold','Knee','Healing'
    ,5,0,0,0,450,5000);
INSERT INTO CONSUMABLE VALUES (31,'Bronze','Leg','Healing'
    ,1,0,0,0,450,5000);
INSERT INTO CONSUMABLE VALUES (32,'Silver','Leg','Healing'
    ,2,0,0,0,450,5000);
INSERT INTO CONSUMABLE VALUES (33,'Gold','Leg','Healing'
    ,5,0,0,0,450,5000);
INSERT INTO CONSUMABLE VALUES (34,'Bronze','Foot','Healing'
    ,1,0,0,0,450,5000);
INSERT INTO CONSUMABLE VALUES (35,'Silver','Foot','Healing'
    ,2,0,0,0,450,5000);
INSERT INTO CONSUMABLE VALUES (36,'Gold','Foot','Healing'
    ,5,0,0,0,450,5000);
INSERT INTO CONSUMABLE VALUES (37,'Bronze','All','Healing'
    ,1,0,0,0,450,5000);
INSERT INTO CONSUMABLE VALUES (38,'Silver','All','Healing'
    ,2,0,0,0,450,5000);
INSERT INTO CONSUMABLE VALUES (39,'Gold','All','Healing'
    ,4,0,0,0,450,5000);
INSERT INTO CONSUMABLE VALUES (40,'Bronze','DIV','GKTraining'
    ,5,0,0,0,350,5000);
INSERT INTO CONSUMABLE VALUES (41,'Silver','DIV','GKTraining'
    ,10,0,0,0,350,5000);
INSERT INTO CONSUMABLE VALUES (42,'Gold','DIV','GKTraining'
    ,15,0,0,0,350,5000);
INSERT INTO CONSUMABLE VALUES (43,'Bronze','HAN','GKTraining'
    ,5,0,0,0,350,5000);
INSERT INTO CONSUMABLE VALUES (44,'Silver','HAN','GKTraining'
    ,10,0,0,0,350,5000);
INSERT INTO CONSUMABLE VALUES (45,'Gold','HAN','GKTraining'
    ,15,0,0,0,350,5000);
INSERT INTO CONSUMABLE VALUES (46,'Bronze','KIC','GKTraining'
    ,5,0,0,0,350,5000);
INSERT INTO CONSUMABLE VALUES (47,'Silver','KIC','GKTraining'
    ,10,0,0,0,350,5000);
INSERT INTO CONSUMABLE VALUES (48,'Gold','KIC','GKTraining'
    ,15,0,0,0,350,5000);
INSERT INTO CONSUMABLE VALUES (49,'Bronze','SPD','GKTraining'
    ,5,0,0,0,350,5000);
INSERT INTO CONSUMABLE VALUES (50,'Silver','SPD','GKTraining'
    ,10,0,0,0,350,5000);
INSERT INTO CONSUMABLE VALUES (51,'Gold','SPD','GKTraining'
    ,15,0,0,0,350,5000);
INSERT INTO CONSUMABLE VALUES (52,'Bronze','POS','GKTraining'
    ,5,0,0,0,350,5000);
INSERT INTO CONSUMABLE VALUES (53,'Silver','POS','GKTraining'
    ,10,0,0,0,350,5000);
INSERT INTO CONSUMABLE VALUES (54,'Gold','POS','GKTraining'
    ,15,0,0,0,350,5000);
INSERT INTO CONSUMABLE VALUES (55,'Bronze','REF','GKTraining'
    ,5,0,0,0,350,5000);
INSERT INTO CONSUMABLE VALUES (56,'Silver','REF','GKTraining'
    ,10,0,0,0,350,5000);
```

```
INSERT INTO CONSUMABLE VALUES (57,'Gold','REF','GKTraining'
    ,15,0,0,0,350,5000);
INSERT INTO CONSUMABLE VALUES (58,'Bronze','ALL','GKTraining'
    ,3,0,0,0,350,5000);
INSERT INTO CONSUMABLE VALUES (59,'Silver','ALL','GKTraining'
    ,6,0,0,0,350,5000);
INSERT INTO CONSUMABLE VALUES (60,'Gold','ALL','GKTraining'
    ,10,0,0,0,350,5000);
INSERT INTO CONSUMABLE VALUES (61,'Bronze','PAC','Training'
    ,5,0,0,0,350,5000);
INSERT INTO CONSUMABLE VALUES (62,'Silver','PAC','Training'
    ,10,0,0,0,350,5000);
INSERT INTO CONSUMABLE VALUES (63,'Gold','PAC','Training'
    ,15,0,0,0,350,5000);
INSERT INTO CONSUMABLE VALUES (64,'Bronze','SHO','Training'
    ,5,0,0,0,350,5000);
INSERT INTO CONSUMABLE VALUES (65,'Silver','SHO','Training'
    ,10,0,0,0,350,5000);
INSERT INTO CONSUMABLE VALUES (66,'Gold','SHO','Training'
    ,15,0,0,0,350,5000);
INSERT INTO CONSUMABLE VALUES (67,'Bronze','PAS','Training'
    ,5,0,0,0,350,5000);
INSERT INTO CONSUMABLE VALUES (68,'Silver','PAS','Training'
    ,10,0,0,0,650,5000);
INSERT INTO CONSUMABLE VALUES (69,'Gold','PAS','Training'
    ,15,0,0,0,650,5000);
INSERT INTO CONSUMABLE VALUES (70,'Bronze','DRI','Training'
    ,5,0,0,0,650,5000);
INSERT INTO CONSUMABLE VALUES (71,'Silver','DRI','Training'
    ,10,0,0,0,650,5000);
INSERT INTO CONSUMABLE VALUES (72,'Gold','DRI','Training'
    ,15,0,0,0,650,5000);
INSERT INTO CONSUMABLE VALUES (73,'Bronze','PHY','Training'
    ,5,0,0,0,650,5000);
INSERT INTO CONSUMABLE VALUES (74,'Silver','PHY','Training'
    ,10,0,0,0,650,5000);
INSERT INTO CONSUMABLE VALUES (75,'Gold','PHY','Training'
    ,15,0,0,0,650,5000);
INSERT INTO CONSUMABLE VALUES (76,'Bronze','DEF','Training'
    ,5,0,0,0,650,5000);
INSERT INTO CONSUMABLE VALUES (77,'Silver','DEF','Training'
    ,10,0,0,0,650,5000);
INSERT INTO CONSUMABLE VALUES (78,'Gold','DEF','Training'
    ,15,0,0,0,650,5000);
INSERT INTO CONSUMABLE VALUES (79,'Bronze','ALL','Training'
    ,3,0,0,0,650,5000);
INSERT INTO CONSUMABLE VALUES (80,'Silver','ALL','Training'
    ,6,0,0,0,650,5000);
INSERT INTO CONSUMABLE VALUES (81,'Gold','ALL','Training'
    ,10,0,0,0,650,5000);
INSERT INTO CONSUMABLE VALUES (82,'Gold','LWBLB','Positioning'
    ,0,0,0,0,650,5000);
INSERT INTO CONSUMABLE VALUES (83,'Gold','LBLWB','Positioning'
    ,0,0,0,0,650,5000);
INSERT INTO CONSUMABLE VALUES (84,'Gold','RWBRB','Positioning'
    ,0,0,0,0,650,5000);
```

```
INSERT INTO CONSUMABLE VALUES (85,'Gold','RBRWB','Positioning'
    ,0,0,0,0,650,5000);
INSERT INTO CONSUMABLE VALUES (86,'Gold','LMLW','Positioning'
    ,0,0,0,0,650,5000);
INSERT INTO CONSUMABLE VALUES (87,'Gold','RMRW','Positioning'
    ,0,0,0,0,250,5000);
INSERT INTO CONSUMABLE VALUES (88,'Gold','LWLM','Positioning'
    ,0,0,0,0,250,5000);
INSERT INTO CONSUMABLE VALUES (89,'Gold','RWRM','Positioning'
    ,0,0,0,0,250,5000);
INSERT INTO CONSUMABLE VALUES (90,'Gold','LWLF','Positioning'
    ,0,0,0,0,250,5000);
INSERT INTO CONSUMABLE VALUES (91,'Gold','RWRF','Positioning'
    ,0,0,0,0,250,5000);
INSERT INTO CONSUMABLE VALUES (92,'Gold','LFLW','Positioning'
    ,0,0,0,0,250,5000);
INSERT INTO CONSUMABLE VALUES (93,'Gold','RFRW','Positioning'
    ,0,0,0,0,250,5000);
INSERT INTO CONSUMABLE VALUES (94,'Gold','CMCAM','Positioning'
    ,0,0,0,0,250,5000);
INSERT INTO CONSUMABLE VALUES (95,'Gold','CAMCM','Positioning'
    ,0,0,0,0,250,5000);
INSERT INTO CONSUMABLE VALUES (96,'Gold','CDMCM','Positioning'
    ,0,0,0,0,250,5000);
INSERT INTO CONSUMABLE VALUES (97,'Gold','CMCDM','Positioning'
    ,0,0,0,0,250,5000);
INSERT INTO CONSUMABLE VALUES (98,'Gold','CAMCF','Positioning'
    ,0,0,0,0,250,5000);
INSERT INTO CONSUMABLE VALUES (99,'Gold','CFCAM','Positioning'
    ,0,0,0,0,250,5000);
INSERT INTO CONSUMABLE VALUES (100,'Gold','CFST','Positioning'
    ,0,0,0,0,250,5000);
INSERT INTO CONSUMABLE VALUES (101,'Gold','STCF','Positioning'
    ,0,0,0,0,250,5000);
INSERT INTO CONSUMABLE VALUES (102,'Gold','DEN1','Manager
    League',1,0,0,0,250,5000);
INSERT INTO CONSUMABLE VALUES (103,'Gold','BEL1','Manager
    League',4,0,0,0,250,5000);
INSERT INTO CONSUMABLE VALUES (104,'Gold','NED1','Manager
    League',10,0,0,0,250,5000);
INSERT INTO CONSUMABLE VALUES (105,'Gold','ENG1','Manager
    League',13,0,0,0,250,5000);
INSERT INTO CONSUMABLE VALUES (106,'Gold','ENG2','Manager
    League',14,0,0,0,250,5000);
INSERT INTO CONSUMABLE VALUES (107,'Gold','FRA1','Manager
    League',16,0,0,0,250,5000);
INSERT INTO CONSUMABLE VALUES (108,'Gold','FRA2','Manager
    League',17,0,0,0,250,5000);
INSERT INTO CONSUMABLE VALUES (109,'Gold','GER1','Manager
    League',19,0,0,0,250,5000);
INSERT INTO CONSUMABLE VALUES (110,'Gold','GER2','Manager
    League',20,0,0,0,250,5000);
INSERT INTO CONSUMABLE VALUES (111,'Gold','ITA1','Manager
    League',31,0,0,0,150,5000);
INSERT INTO CONSUMABLE VALUES (112,'Gold','ITA2','Manager
    League',32,0,0,0,150,5000);
```

```
INSERT INTO CONSUMABLE VALUES (113,'Gold','MLS','Manager League
    ',39,0,0,0,150,5000);
INSERT INTO CONSUMABLE VALUES (114,'Gold','NOR1','Manager
    League',41,0,0,0,150,5000);
INSERT INTO CONSUMABLE VALUES (115,'Gold','SPFL','Manager
    League',50,0,0,0,150,5000);
INSERT INTO CONSUMABLE VALUES (116,'Gold','ESP1','Manager
    League',53,0,0,0,150,5000);
INSERT INTO CONSUMABLE VALUES (117,'Gold','ESP2','Manager
    League',54,0,0,0,150,5000);
INSERT INTO CONSUMABLE VALUES (118,'Gold','SWE1','Manager
    League',56,0,0,0,150,5000);
INSERT INTO CONSUMABLE VALUES (119,'Gold','ENG3','Manager
    League',60,0,0,0,150,5000);
INSERT INTO CONSUMABLE VALUES (120,'Gold','ENG4','Manager
    League',61,0,0,0,150,5000);
INSERT INTO CONSUMABLE VALUES (121,'Gold','GRE1','Manager
    League',63,0,0,0,150,5000);
INSERT INTO CONSUMABLE VALUES (122,'Gold','IRL1','Manager
    League',65,0,0,0,150,5000);
INSERT INTO CONSUMABLE VALUES (123,'Gold','POL1','Manager
    League',66,0,0,0,150,5000);
INSERT INTO CONSUMABLE VALUES (124,'Gold','RUS1','Manager
    League',67,0,0,0,150,5000);
INSERT INTO CONSUMABLE VALUES (125,'Gold','TUR1','Manager
    League',68,0,0,0,150,5000);
INSERT INTO CONSUMABLE VALUES (126,'Gold','AUT1','Manager
    League',80,0,0,0,150,5000);
INSERT INTO CONSUMABLE VALUES (127,'Gold','KOR1','Manager
    League',83,0,0,0,150,5000);
INSERT INTO CONSUMABLE VALUES (128,'Gold','SUI1','Manager
    League',10,0,0,0,150,5000);
INSERT INTO CONSUMABLE VALUES (129,'Gold','POR1','Manager
    League',10,0,0,0,150,5000);
INSERT INTO CONSUMABLE VALUES (130,'Gold','UKR1','Manager
    League',10,0,0,0,150,5000);
INSERT INTO CONSUMABLE VALUES (131,'Gold','CHI1','Manager
    League',10,0,0,0,150,5000);
INSERT INTO CONSUMABLE VALUES (132,'Gold','COL1','Manager
    League',10,0,0,0,150,5000);
INSERT INTO CONSUMABLE VALUES (133,'Gold','MEX1','Manager
    League',10,0,0,0,150,5000);
INSERT INTO CONSUMABLE VALUES (134,'Gold','RSA1','Manager
    League',10,0,0,0,150,5000);
INSERT INTO CONSUMABLE VALUES (135,'Gold','SAU1','Manager
    League',10,0,0,0,150,5000);
INSERT INTO CONSUMABLE VALUES (136,'Gold','AUS1','Manager
    League',10,0,0,0,150,5000);
INSERT INTO CONSUMABLE VALUES (137,'Gold','ARG1','Manager
    League',10,0,0,0,150,5000);
INSERT INTO CONSUMABLE VALUES (138,'Gold','LGD','Manager League
    ',10,0,0,0,150,5000);
INSERT INTO CONSUMABLE VALUES (139,'Gold','FIN1','Manager
    League',10,0,0,0,150,5000);
INSERT INTO CONSUMABLE VALUES (140,'Gold','JPN1','Manager
    League',10,0,0,0,150,5000);
```

10.4 Creazione utenti (DCL)

```
--Utente PLAYER 1
CREATE USER PLAYER1 IDENTIFIED BY PASS;
GRANT CONNECT, CREATE SESSION TO PLAYER1;

--VIEW
GRANT SELECT ON GET_ACTIVE_CLUB_ITEMS TO PLAYER1;
GRANT SELECT ON GET_AVAILABLE_PACKS TO PLAYER1;
GRANT SELECT ON GET_CLUB_ITEMS TO PLAYER1;
GRANT SELECT ON GET_CONSUMABLES TO PLAYER1;
GRANT SELECT ON GET_MANAGERS TO PLAYER1;
GRANT SELECT ON GET_MANAGES TO PLAYER1;
GRANT SELECT ON GET_MATCH TO PLAYER1;
GRANT SELECT ON GET_PACK_PURCHASES TO PLAYER1;
GRANT SELECT ON GET_PLAYERS_FOR_SALE TO PLAYER1;
GRANT SELECT ON GET_PLAYERS TO PLAYER1;
GRANT SELECT ON GET_CLUB_INFO TO PLAYER1;
GRANT SELECT ON GET_PLAYERS_STOPS TO PLAYER1;

--FUNCTION
GRANT EXECUTE ON GET_SQUAD TO PLAYER1;
GRANT EXECUTE ON SEARCH_CARD_FOR_SALE TO PLAYER1;

--PROCEDURE
GRANT EXECUTE ON ADD_CLUB_ITEM TO PLAYER1;
GRANT EXECUTE ON ADD_CONTRACT TO PLAYER1;
GRANT EXECUTE ON ADD_FITNESS TO PLAYER1;
GRANT EXECUTE ON ADD_HEALING TO PLAYER1;
GRANT EXECUTE ON ADD_MANAGER TO PLAYER1;
GRANT EXECUTE ON ADD_MANAGER_LEAGUE TO PLAYER1;
GRANT EXECUTE ON ADD_PLAYER TO PLAYER1;
GRANT EXECUTE ON ADD_POSITIONING TO PLAYER1;
GRANT EXECUTE ON ADD_TRAINING TO PLAYER1;
GRANT EXECUTE ON BUY_CARD TO PLAYER1;
GRANT EXECUTE ON CLUB_DELETE TO PLAYER1;
GRANT EXECUTE ON INSERT_MATCH TO PLAYER1;
GRANT EXECUTE ON PACK_OPENING TO PLAYER1;
GRANT EXECUTE ON REMOVE_MANAGER TO PLAYER1;
GRANT EXECUTE ON SELL_CARD TO PLAYER1;
GRANT EXECUTE ON SQUAD_CREATION TO PLAYER1;
GRANT EXECUTE ON CHANGE_PLAYER TO PLAYER1;

EXECUTE CLUB_CREATION('PLAYER1', 'PLAYER1@EMAIL.COM', 'club1');
```

Capitolo 11

PL-SQL

11.1 Procedure

11.1.1 Club Creation

```
-- Tabelle interessate: 2
--  -> USER_FUT, CLUB;

-- Procedure interessate: 1
--  -> PACK_OPENING;

-- Triggers interessati: 1
--  -> INSERT_USER;

-- INPUT:
--      -> usr_nick: nome utente;
--      -> usr_email: email;
--      -> usr_club_name: il nome del club;
-- OUTPUT:
--  -> Crea il club <usr_club_name> per l'utente identificato
   dal nome utente <usr_nick>.
CREATE OR REPLACE PROCEDURE CLUB_CREATION (
USR_NICK        USER_FUT.NICKNAME%TYPE,
USR_EMAIL       USER_FUT.EMAIL%TYPE,
USR_CLUB_NAME   CLUB.CLUB_NAME%TYPE
) IS

COUNTER  NUMBER(2, 0); --Contatore usato per eseguire piu'
   volte la procedura PACK_OPENING.
N1       NUMBER(1, 0); --Controlla se il nome utente scelto e'
   disponibile.
N2       NUMBER(1, 0); --Controlla se il nome del club scelto e
   ' disponibile.
USER_EXISTING EXCEPTION; --Nel caso in cui il nome utente non e
   ' disponibile.
INVALID_DATA EXCEPTION; --Nel caso in cui ci siano errori sui
   dati inseriti.

BEGIN
--Controllo se il nome utente <usr_nick> e' disponibile.
SELECT COUNT(NICKNAME)
INTO N1
```

```
FROM USER_FUT
WHERE NICKNAME = USR_NICK;
--Controllo se il nome del club <usr_club_name> e' disponibile.
SELECT COUNT(USER_NICK)
INTO N2
FROM CLUB
WHERE USER_NICK = USR_NICK;

--Se il nome utente e il nome del club sono disponibili, creo
   il nuovo utente.
IF N1 = 0
    AND N2 = 0
THEN
--Inserisco il nuovo utente.
    INSERT INTO USER_FUT
        SELECT
            USR_NICK,
            USR_EMAIL
        FROM
            DUAL;
--Inserisco il club dell'utente <usr_nick>.
    INSERT INTO CLUB (
        CLUB_NAME,
        USER_NICK,
        CREDITS,
        DIVISION_NUMBER
    )
        SELECT
            USR_CLUB_NAME,
            USR_NICK,
            3750,
            10
        FROM
            DUAL;
--Nel caso in cui o il nome ustente o il nome del club non
   siano disponibili.
ELSIF N1 > 0 OR N2 > 0 THEN
    RAISE USER_EXISTING;
ELSE
    RAISE INVALID_DATA;
END IF;

--Quando si crea un club ci sono 3 pacchetti <pack> in omaggio.
FOR COUNTER IN 1..3 LOOP
    PACK_OPENING('Bronze Player Pack', USR_NICK);
END LOOP;

COMMIT;
EXCEPTION
WHEN USER_EXISTING THEN
    RAISE_APPLICATION_ERROR('-20003', 'Il nome utente '
                                        || USR_NICK
                                        || ' non e''
                                           disponibile!');
WHEN INVALID_DATA THEN
    RAISE_APPLICATION_ERROR('-20004', 'Invalid data error');
END CLUB_CREATION;
```

11.1.2 Squad Creation

```
-- Tabelle interessate: 1
--  -> SQUAD;

-- Funzioni interessate: 1
--  -> IS_ADMIN;

-- INPUT:
--      -> s_name:  nome della squadra da creare;
--      -> s_mod:  modulo della squadra, esempio:
   ('4-3-3','4-4-2'....etc.)
--      -> cc_name: se il chiamante e' amministratore: contiene
    il nome del club su cui effettuare le operazioni;
--              se il chiamante non e' amministratore: contiene
    o NULL, oppure il nome del club dell'utente chiamante.
-- OUTPUT:
--  -> Crea una squadra con nome <s_name>.
CREATE OR REPLACE PROCEDURE SQUAD_CREATION (
SNAME    SQUAD.NAME%TYPE,
SMOD     SQUAD.MODULES%TYPE,
CC_NAME  CLUB.CLUB_NAME%TYPE
) IS
N1       NUMBER(1, 0);
C_NAME   CLUB.CLUB_NAME%TYPE;

BEGIN
--La funzione is_admin, ritorna il nome del club in base all'
   utente che chiama la funzione SQUAD_CREATION;
--Se e' l'amministratore a chiamare la funzione il nome del
   club <c_name> sara' uguale a <cc_name>
--Altrimenti il nome del club <c_name> sara' quello dell'utente
   .
--In breve l'amministratore puo' eseguire la funzione su
   qualsiasi club, l'utente solo sul suo club.
SELECT IS_ADMIN(CC_NAME, USER)
INTO C_NAME
FROM DUAL;
--Controlla se esiste una squadra con nome uguale a <s_name>.
SELECT COUNT(NAME)
INTO N1
FROM SQUAD
WHERE SNAME = NAME;
--Se <n1 != 0>, esiste una squadra con nome uguale a <s_name>.
IF N1 != 0 THEN
    DBMS_OUTPUT.PUT_LINE('Stai tentando di aggiungere una
       squadra gia'' presente!');
ELSE --Inserisco la tupla.
    INSERT INTO SQUAD ( NAME, SQUAD_CLUB_NAME, MODULES)
        SELECT SNAME, C_NAME, SMOD
        FROM DUAL;
    COMMIT;
END IF;
END SQUAD_CREATION;
/
```

11.1.3 Sell Card

```
-- Tabelle interessate: 6
--  -> CLUB_CARD, TRANSACTION, PLAYER, CONSUMABLE, CLUB_ITEM,
   MANAGER;

-- Funzioni interessate: 2
--  -> IS_ADMIN, CARD_IN_CLUB;

-- INPUT:
--      -> s_card_code:  codice della carta da vendere.
--      -> ss_club_name: se il chiamante e' amministratore:
   contiene il nome del club su cui effettuare le operazioni;
--                       se il chiamante non e' amministratore:
   contiene o NULL, oppure il nome del club dell'utente
   chiamante.
--      -> s_price:  prezzo della carta.
--  -> s_duration: durata della vendita.
-- OUTPUT:
--  -> Mette in vendita la carta <s_card_code>.
CREATE OR REPLACE PROCEDURE SELL_CARD (
S_CARD_CODE   TRANSACTION.T_CARD_CODE%TYPE,
SS_CLUB_NAME  TRANSACTION.TRANSITION_S_CLUB_NAME%TYPE,
S_PRICE       TRANSACTION.PRICE%TYPE,
S_DURATION    TRANSACTION.DURATION%TYPE
) IS
--Nome del club effettivo (is_admin).
S_CLUB_NAME  CLUB.CLUB_NAME%TYPE;
--Variabili di controllo.
N1           NUMBER(2, 0);
N2           NUMBER(2, 0);
SS_PRICE     PLAYER.MAX_PRICE%TYPE;
PRICE_MIN    PLAYER.MIN_PRICE%TYPE;
PRICE_MAX    PLAYER.MAX_PRICE%TYPE;
TR_ID        VARCHAR(8);
--Id in base alla tipologia di carta
P_ID         NUMBER(1, 0); --Id Player
C_ID         NUMBER(1, 0); --Id Consumable
CT_ID        NUMBER(1, 0); --Id Club_item
M_ID         NUMBER(1, 0); --Id Manager

NO_CARD_FOUND EXCEPTION; --Nel caso in cui la carta con codice
   carta <s_card_code> non viene trovata.

BEGIN
--La funzione is_admin, ritorna il nome del club in base all'
   utente che chiama la funzione SELL_CARD;
--Se e' l'amministratore a chiamare la funzione il nome del
   club <c_name> sara' uguale a <ss_club_name>
--Altrimenti il nome del club <s_club_name> sara' quello dell'
   utente.
--In breve l'amministratore puo' eseguire la funzione su
   qualsiasi club, l'utente solo sul suo club.
SELECT
    IS_ADMIN(SS_CLUB_NAME, USER)
INTO S_CLUB_NAME
FROM
```

```
    DUAL;

--La funzione card_in_club, controlla che il club identificato
   dal nome <s_club_name> possieda la carta
-- identificata dal codice carta <s_card_code>
-- se <n1 != 0> il club possiede la carta, altrimenti non la
   possiede.
SELECT
    CARD_IN_CLUB(S_CARD_CODE, S_CLUB_NAME)
INTO N1
FROM
    DUAL;

--Se <n1 = 0>, la carta con codice carta <s_card_code> non e'
   stata trovata.
IF N1 = 0 THEN
    RAISE NO_CARD_FOUND;
ELSE
--Vediamo di che tipologia e' la carta, per poi vedere se il
   prezzo e' nel range.
    SELECT
        COUNT(PLAYER_ID),
        COUNT(CONSUMABLE_ID),
        COUNT(CLUB_ITEM_ID),
        COUNT(MANAGER_ID)
    INTO
        P_ID,
        C_ID,
        CT_ID,
        M_ID
    FROM
        CLUB_CARD
    WHERE
        CARD_CODE = S_CARD_CODE;

--Se la carta e' di tipo <player>.
    IF P_ID != 0 THEN
        SELECT
            MIN_PRICE,
            MAX_PRICE
        INTO
            PRICE_MIN,
            PRICE_MAX
        FROM
            PLAYER
        WHERE
            CARD_ID IN (
                SELECT
                    PLAYER_ID
                FROM
                    CLUB_CARD
                WHERE
                    CARD_CODE = S_CARD_CODE
            );

--Se la carta e' di tipo <consumable>.
    ELSIF C_ID != 0 THEN
```

```
        SELECT
            MIN_PRICE,
            MAX_PRICE
        INTO
            PRICE_MIN,
            PRICE_MAX
        FROM
            CONSUMABLE
        WHERE
            CARD_ID IN (
                SELECT
                    CONSUMABLE_ID
                FROM
                    CLUB_CARD
                WHERE
                    CARD_CODE = S_CARD_CODE
            );

--Se la carta e' di tipo <club_item>.
    ELSIF CT_ID != 0 THEN
        SELECT
            MIN_PRICE,
            MAX_PRICE
        INTO
            PRICE_MIN,
            PRICE_MAX
        FROM
            CLUB_ITEM
        WHERE
            CARD_ID IN (
                SELECT
                    CLUB_ITEM_ID
                FROM
                    CLUB_CARD
                WHERE
                    CARD_CODE = S_CARD_CODE
            );

--Se la carta e' di tipo <manager>.
    ELSIF M_ID != 0 THEN
        SELECT
            MIN_PRICE,
            MAX_PRICE
        INTO
            PRICE_MIN,
            PRICE_MAX
        FROM
            MANAGER
        WHERE
            CARD_ID IN (
                SELECT
                    MANAGER_ID
                FROM
                    CLUB_CARD
                WHERE
                    CARD_CODE = S_CARD_CODE
            );
```

```
    END IF;

--Se il prezzo e minore del minimo prezzo della carta, il nuovo
    prezzo sara' uguale al minimo.
    IF S_PRICE < PRICE_MIN THEN
        SS_PRICE := PRICE_MIN;

--Se il prezzo e' maggiore del prezzo massimo, il nuovo prezzo
   sara' uguale al prezzo massimo.
    ELSIF S_PRICE > PRICE_MAX THEN
        SS_PRICE := PRICE_MAX;
    ELSE
--Altrimenti il prezzo rimane invariato.
        SS_PRICE := S_PRICE;
    END IF;

--Inserisco la carta in vendita.
    TR_ID := DBMS_RANDOM.STRING('a',8);
    INSERT INTO TRANSACTION (
        TRANSACTION_ID,
        T_DATE,
        TRANSITION_S_CLUB_NAME,
        T_CARD_CODE,
        PRICE,
        DURATION
    )
        SELECT
            TR_ID,
            TO_DATE(SYSDATE, 'DD/MM/YYYY HH24:MI:SS'),
            S_CLUB_NAME,
            S_CARD_CODE,
            SS_PRICE,
            S_DURATION
        FROM
            DUAL;

--Confermo
    COMMIT;
END IF;

EXCEPTION
WHEN NO_CARD_FOUND THEN
    RAISE_APPLICATION_ERROR('-20008', 'Selling error!');
END SELL_CARD;
/
```

11.1.4 Buy Card

```
-- Tabelle interessate: 2
--  ->TRANSACTION, CLUB;

-- Funzioni interessate: 2
--  -> IS_ADMIN;

-- INPUT:
--      -> b_card_code:  codice della carta da comprare.
--      -> bb_club_name: se il chiamante e' amministratore:
   contiene il nome del club su cui effettuare le operazioni;
--                       se il chiamante non e' amministratore:
   contiene o NULL, oppure il nome del club dell'utente
   chiamante.
--      -> b_price:  prezzo della carta.
-- OUTPUT:
--  -> Compra la carta identificata da l codice carta <
   b_card_code>.
CREATE OR REPLACE PROCEDURE BUY_CARD (
B_CARD_CODE   TRANSACTION.T_CARD_CODE%TYPE,
BB_CLUB_NAME  TRANSACTION.TRANSITION_B_CLUB_NAME%TYPE,
B_PRICE       TRANSACTION.PRICE%TYPE
) IS
--Variabili di controllo
N1            NUMBER(2, 0);
CRT           CLUB.CREDITS%TYPE;--Crediti del club, da
   aggiornare dopo aver comprato la carta <b_card_code>.
S_PRICE       TRANSACTION.PRICE%TYPE; --Prezzo della carta in
   vendita.
TR_ID         VARCHAR(8);--Id della transizione

B_CLUB_NAME   CLUB.CLUB_NAME%TYPE; --Nome effettivo del club (
   is_admin).
S_CLUB_NAME   TRANSACTION.TRANSITION_S_CLUB_NAME%TYPE; --Nome
   del club che vende la carta <b_card_code>.

SAME_CLUB EXCEPTION; --Nel caso in cui si cerca di comprare una
    carta messa in vendita dallo stesso club.
BUYING_ERROR EXCEPTION; --Nel caso in cui l'operazione non va a
    buon fine.

BEGIN
--La funzione is_admin, ritorna il nome del club in base all'
   utente che chiama la funzione BUY_CARD;
--Se e' l'amministratore a chiamare la funzione il nome del
   club <B_CLUB_NAME> sara' uguale a <BB_CLUB_NAME>
--Altrimenti il nome del club <B_CLUB_NAME> sara' quello dell'
   utente.
--In breve l'amministratore puo' eseguire la funzione su
   qualsiasi club, l'utente solo sul suo club.
SELECT
    IS_ADMIN(BB_CLUB_NAME, USER)
INTO B_CLUB_NAME
FROM
    DUAL;
```

```
--Vediamo se il club <b_club_name> ha abbastanza crediti per
   effettuare l'acquisto.
SELECT
    CLUB.CREDITS
INTO CRT
FROM
    CLUB
WHERE
    CLUB.CLUB_NAME = B_CLUB_NAME;

--Controllo se la transazione esiste.
SELECT
    COUNT(T_CARD_CODE),
    MAX(TRANSITION_S_CLUB_NAME),
    MAX(PRICE),
    MAX(TRANSACTION_ID)
INTO
    N1,
    S_CLUB_NAME,
    S_PRICE,
    TR_ID
FROM
    TRANSACTION
WHERE
        T_CARD_CODE = B_CARD_CODE
    AND TRANSITION_B_CLUB_NAME IS NULL;

--Se il club che vende e' uguale a quello che compra.
IF S_CLUB_NAME = B_CLUB_NAME THEN
    RAISE SAME_CLUB;
--Se <n1 > 0>, la transazione e' stata trovata
-- e il prezzo alla quale si vuole acquistare e minore o uguale
    al prezzo della carta in vendita.
ELSIF
    N1 > 0
    AND S_PRICE <= B_PRICE
THEN


DELETE FROM TRANSACTION
        WHERE
                T_CARD_CODE = B_CARD_CODE
            AND TRANSACTION_ID <> TR_ID;

DELETE FROM IS_FOUND
        WHERE
                CARD_CODE = B_CARD_CODE
            AND P_ID IN (
                SELECT
                    PURCHASE_ID
                FROM
                    PACK_PURCHASE
                WHERE
                    BUYING_CLUB_NAME = S_CLUB_NAME
            );
```

```
--Aggiorno la transazione impostanto il club che compra uguale
   a <b_club_name>.
    UPDATE TRANSACTION
    SET
        TRANSITION_B_CLUB_NAME = B_CLUB_NAME
    WHERE
            T_CARD_CODE = B_CARD_CODE
        AND TRANSITION_B_CLUB_NAME IS NULL
        AND TRANSITION_S_CLUB_NAME <> B_CLUB_NAME;

--Incremento i crediti del club <s_club_name> che ha venduto la
    carta.
    UPDATE CLUB
    SET
        CREDITS = CREDITS + S_PRICE
    WHERE
        CLUB_NAME = S_CLUB_NAME;

--Decremento i crediti del club <b_club_name> che ha comprato
   la carta.
    UPDATE CLUB
    SET
        CREDITS = CREDITS - S_PRICE
    WHERE
        CLUB_NAME = B_CLUB_NAME;

--Confermo
    COMMIT;
ELSE
    RAISE BUYING_ERROR;
END IF;

EXCEPTION
WHEN SAME_CLUB THEN
    RAISE_APPLICATION_ERROR('-20005', 'Stai tendando di
       comprare un carta messa in vendita da te!');
WHEN BUYING_ERROR THEN
    RAISE_APPLICATION_ERROR('-20003', 'Buying error!');
END BUY_CARD;
/
```

11.1.5 Pack Opening

```
-- Tabelle interessate: 9
--  -> CLUB_CARD, CLUB, PLAYER, CONSUMABLE, CLUB_ITEM, MANAGER,
    IS_FOUND, PACK_PURCHASE, TRANSACTION;

-- INPUT:
--      -> p_name:  nome del pacchetto che si vuole aprire;
--      -> usr_1:  nome utente;
-- OUTPUT:
--  -> Apre il pacchetto <p_name>.
CREATE OR REPLACE PROCEDURE PACK_OPENING (
P_NAME  PACK.PACK_NAME%TYPE,
USR_1   CLUB.USER_NICK%TYPE
) IS

P         PACK%ROWTYPE; --Info sul pacchetto da aprire.
CRT       CLUB.CREDITS%TYPE; --Crediti dell'utente.
USR       CLUB.USER_NICK%TYPE; --Nome utente effettivo (isAdmin
   ).
--Variabile d'appoggio per operare sul nome utente <usr_1>
   fornito in input.
USR_2     CLUB.USER_NICK%TYPE;
N1        NUMBER(4,0);
STR1      VARCHAR2(8); --Codice carta.
STR2      VARCHAR2(8);
COUNTER   NUMBER(2, 0);
M_LEAGUE  MANAGER.LEAGUE_NAME%TYPE; --lega del manager.

NO_USER_FOUND EXCEPTION; --L'utente con nome utente <usr_1> non
    e' stato trovato.
--Se ad esempio l'utente 'PLAYER1' prova ad aprire un pack con
   un altro nome utente, solo l'amministratore puo'.
USER_ERROR EXCEPTION;
BEGIN
--Salvo il nome utente <usr_1> in <usr_2>.
USR_2 := USR_1;

--Se l'utente chiamante non e' amministratore.
IF USER <> 'ADMINFUT' THEN
--Controllo se l'utente ha un club.
    SELECT
        COUNT(*)
    INTO N1
    FROM
        CLUB
    WHERE
        LOWER(USER_NICK) = LOWER(USER);

--Se <n1 = 0>, l'utente con nome utente <usr_1> non e' stato
   trovato.
    IF N1 = 0 THEN
        RAISE NO_USER_FOUND;
    END IF;

--Seleziono il nome utente chiamante in <usr>.
    SELECT
```

```
        USER
    INTO USR
    FROM
        DUAL;
--Se l'utente chiamante e' amministratore il nome utente
   corrisponde a quello dato in input.
ELSE
    USR := USR_1;
END IF;

--Se il nome utente dato in input <usr_2> non e' null
-- ed e' diverso dal nome utente che chiama la funzione, e non
   e' amministratore.
IF
    USR_2 IS NOT NULL
    AND USR <> USR_2
    AND USER <> 'ADMINFUT'
THEN
    RAISE USER_ERROR;
END IF;

--La funzione Number Of Cards, controlla il volume delle carte
   del club;
--Il quale non puo' superare 1500, se il massimo non viene
   superato varra'
--notificato all'utente quante carte ha collezionato fin ora e
-- quante ne potra' ancora collezionare.
ADMINFUT.NUMBER_OF_CARDS(USR);

--Seleziono i crediti del club dell'utente, e il prezzo del
   pacchetto.
SELECT
    CLUB.CREDITS,
    PACK.PRICE
INTO
    CRT,
    P.PRICE
FROM
    CLUB,
    PACK
WHERE
        CLUB.USER_NICK = USR
    AND PACK.PACK_NAME = P_NAME;

--Se il club non ha abbastanza crediti.
IF CRT < P.PRICE THEN
    DBMS_OUTPUT.PUT_LINE('Crediti insufficenti!');
ELSE
--Controllo se il pacchetto e' disponibile.
    SELECT
        AVAILABLE
    INTO P.AVAILABLE
    FROM
        PACK
    WHERE
        PACK_NAME = P_NAME;
```

```
--Se il pacchetto e' disponibile.
    IF P.AVAILABLE = '1' THEN

--Seleziono i dettagli del pacchetto in <p>.
        SELECT
            *
        INTO P
        FROM
            PACK
        WHERE
            PACK_NAME = P_NAME;

--Calcolo il codice carta.
        STR1 := DBMS_RANDOM.STRING('a', 8);

--Inserisco la tupla del pacchetto aquistato in <pack_purchase
   >.
        INSERT INTO PACK_PURCHASE
            SELECT
                STR1,
                CLUB_NAME,
                P_NAME,
                SYSDATE
            FROM
                CLUB
            WHERE
                USER_NICK = USR;

--Aggiorno il bilancio dei crediti sottraendo il costo del
   pacchetto.
        UPDATE CLUB
        SET
            CREDITS = CREDITS - P.PRICE
        WHERE
            USER_NICK = USR;

--Se <p.players> ovvero la quantita di <player> che possono
   uscire dal pacchetto
--  e' maggiore di 0.
        IF P.PLAYERS > 0 THEN
--Carte di tipo <player> di rarita' <gold>
        FOR COUNTER IN 1..P.PLAYERS LOOP
            IF P.GOLD_QUANTITY > 0 THEN
--Genero il codice carta.
                    STR2 := DBMS_RANDOM.STRING('a', 8);
--Inserisco la nuova carta trovata in <club_Card>.
                    INSERT
                    INTO CLUB_CARD (
                        CARD_CODE,
                        PLAYER_ID
                    )
                        SELECT
                            *
                        FROM
                            (
                                SELECT
                                    STR2,
```

```
                                    CARD_ID
                                FROM
                                    PLAYER
                                WHERE
                                    RARITY_NAME = 'Gold - Rare'
                                    OR RARITY_NAME = 'Gold -
                                       Non-Rare'
                                ORDER BY
                                    DBMS_RANDOM.RANDOM
                            )
                        WHERE
                            ROWNUM <= 1;

--Inserisco la tupla in <is_found>.
                    INSERT INTO IS_FOUND (
                        P_ID,
                        CARD_CODE
                    )
                        SELECT
                            STR1,
                            STR2
                        FROM
                            DUAL;

                    P.GOLD_QUANTITY := P.GOLD_QUANTITY - 1;

--Inserisco i dati attivi in <active_data_player>.
                    INSERT INTO ACTIVE_DATA_PLAYER (
                            P_CARD_CODE,
                            CONTRACTS,
                            PLAYER_FITNESS
                    )
                            SELECT
                                STR2,
                                7,
                                99
                            FROM
                                DUAL;

--Carta di tipo <player> di rarita' <silver>.
             ELSIF P.SILVER_QUANTITY > 0 THEN
                    STR2 := DBMS_RANDOM.STRING('a', 8);
                    INSERT INTO CLUB_CARD (
                        CARD_CODE,
                        PLAYER_ID
                    )
                        SELECT
                            *
                        FROM
                            (
                                SELECT
                                    STR2,
                                    CARD_ID
                                FROM
                                    PLAYER
                                WHERE
                                    RARITY_NAME = 'Silver -
```

```
                                        Rare'
                                    OR RARITY_NAME = 'Silver -
                                        Non-Rare'
                                ORDER BY
                                    DBMS_RANDOM.RANDOM
                            )
                        WHERE
                            ROWNUM <= 1;

                    INSERT INTO IS_FOUND (
                        P_ID,
                        CARD_CODE
                    )
                        SELECT
                            STR1,
                            STR2
                        FROM
                            DUAL;

--Inserisco i dati attivi in <active_data_player>.
                    INSERT INTO ACTIVE_DATA_PLAYER (
                            P_CARD_CODE,
                            CONTRACTS,
                            PLAYER_FITNESS
                    )
                            SELECT
                                STR2,
                                7,
                                99
                            FROM
                                DUAL;

                    P.SILVER_QUANTITY := P.SILVER_QUANTITY - 1;
--Carta di tipo <player> di rarita' <bronze>.
             ELSIF P.BRONZE_QUANTITY > 0 THEN
                    STR2 := DBMS_RANDOM.STRING('a', 8);
                    INSERT INTO CLUB_CARD (
                        CARD_CODE,
                        PLAYER_ID
                    )
                        SELECT
                            *
                        FROM
                            (
                                SELECT
                                    STR2,
                                    CARD_ID
                                FROM
                                    PLAYER
                                WHERE
                                    RARITY_NAME = 'Bronze -
                                        Rare'
                                    OR RARITY_NAME = 'Bronze -
                                        Non-Rare'
                                ORDER BY
                                    DBMS_RANDOM.RANDOM
                            )
```

```
                        WHERE
                            ROWNUM <= 1;

                    INSERT INTO IS_FOUND (
                        P_ID,
                        CARD_CODE
                    )
                        SELECT
                            STR1,
                            STR2
                        FROM
                            DUAL;

--Inserisco i dati attivi in <active_data_player>.
                    INSERT INTO ACTIVE_DATA_PLAYER (
                            P_CARD_CODE,
                            CONTRACTS,
                            PLAYER_FITNESS
                    )
                            SELECT
                                STR2,
                                7,
                                99
                            FROM
                                DUAL;

                    P.BRONZE_QUANTITY := P.BRONZE_QUANTITY - 1;
            ELSE
                DBMS_OUTPUT.PUT_LINE('Error inserimento player'
                   );
            END IF;
            END LOOP;
        END IF;

--Carta di tipo <consumable> di rarita' <gold>.
        IF P.CONSUMABLES > 0 THEN
            FOR COUNTER IN 1..P.CONSUMABLES LOOP
            IF P.GOLD_QUANTITY > 0 THEN
                    STR2 := DBMS_RANDOM.STRING('a', 8);
                    INSERT INTO CLUB_CARD (
                        CARD_CODE,
                        CONSUMABLE_ID
                    )
                        SELECT
                            *
                        FROM
                            (
                                SELECT
                                    STR2,
                                    CARD_ID
                                FROM
                                    CONSUMABLE
                                WHERE
                                    RARITY_NAME = 'Gold'
                                ORDER BY
                                    DBMS_RANDOM.RANDOM
                            )
```

```
                        WHERE
                            ROWNUM <= 1;

                    INSERT INTO IS_FOUND (
                        P_ID,
                        CARD_CODE
                    )
                        SELECT
                            STR1,
                            STR2
                        FROM
                            DUAL;


                        P.GOLD_QUANTITY := P.GOLD_QUANTITY - 1;

--Carta di tipo <consumable> di rarita' <silver>.
            ELSIF P.SILVER_QUANTITY > 0 THEN
                    STR2 := DBMS_RANDOM.STRING('a', 8);
                    INSERT INTO CLUB_CARD (
                        CARD_CODE,
                        CONSUMABLE_ID
                    )
                        SELECT
                            *
                        FROM
                            (
                                SELECT
                                    STR2,
                                    CARD_ID
                                FROM
                                    CONSUMABLE
                                WHERE
                                    RARITY_NAME = 'Silver'
                                ORDER BY
                                    DBMS_RANDOM.RANDOM
                            )
                        WHERE
                            ROWNUM <= 1;

                    INSERT INTO IS_FOUND (
                        P_ID,
                        CARD_CODE
                    )
                        SELECT
                            STR1,
                            STR2
                        FROM
                            DUAL;


                        P.SILVER_QUANTITY := P.SILVER_QUANTITY
                            - 1;

--Carta di tipo <consumable> di rarita' <bronze>.
            ELSIF P.BRONZE_QUANTITY > 0 THEN
                    STR2 := DBMS_RANDOM.STRING('a', 8);
```

```
                    INSERT INTO CLUB_CARD (
                        CARD_CODE,
                        CONSUMABLE_ID
                    )
                        SELECT
                            *
                        FROM
                            (
                                SELECT
                                    STR2,
                                    CARD_ID
                                FROM
                                    CONSUMABLE
                                WHERE
                                    RARITY_NAME = 'Bronze'
                                ORDER BY
                                    DBMS_RANDOM.RANDOM
                            )
                        WHERE
                            ROWNUM <= 1;

                    INSERT INTO IS_FOUND (
                        P_ID,
                        CARD_CODE
                    )
                        SELECT
                            STR1,
                            STR2
                        FROM
                            DUAL;

                        P.BRONZE_QUANTITY := P.BRONZE_QUANTITY
                            - 1;

            ELSE
                DBMS_OUTPUT.PUT_LINE('Error inserimento
                    consumable');
            END IF;
            END LOOP;
        END IF;

        IF P.CLUB_ITEMS > 0 THEN
        FOR COUNTER IN 1..P.CLUB_ITEMS LOOP
--Carta di tipo <club_item> di rarita' <gold>.
            IF P.GOLD_QUANTITY > 0 THEN
                    STR2 := DBMS_RANDOM.STRING('a', 8);
                    INSERT INTO CLUB_CARD (
                        CARD_CODE,
                        CLUB_ITEM_ID
                    )
                        SELECT
                            *
                        FROM
                            (
                                SELECT
                                    STR2,
                                    CARD_ID
```

```
                                FROM
                                    CLUB_ITEM
                                WHERE
                                    RARITY_NAME = 'Gold'
                                ORDER BY
                                    DBMS_RANDOM.RANDOM
                            )
                        WHERE
                            ROWNUM <= 1;

                    INSERT INTO IS_FOUND (
                        P_ID,
                        CARD_CODE
                    )
                        SELECT
                            STR1,
                            STR2
                        FROM
                            DUAL;

                        P.GOLD_QUANTITY := P.GOLD_QUANTITY - 1;
--Carta di tipo <club_item> di rarita' <silver>.
            ELSIF P.SILVER_QUANTITY > 0 THEN
                    STR2 := DBMS_RANDOM.STRING('a', 8);
                    INSERT INTO CLUB_CARD (
                        CARD_CODE,
                        CLUB_ITEM_ID
                    )
                        SELECT
                            *
                        FROM
                            (
                                SELECT
                                    STR2,
                                    CARD_ID
                                FROM
                                    CLUB_ITEM
                                WHERE
                                    RARITY_NAME = 'Silver'
                                ORDER BY
                                    DBMS_RANDOM.RANDOM
                            )
                        WHERE
                            ROWNUM <= 1;

                    INSERT INTO IS_FOUND (
                        P_ID,
                        CARD_CODE
                    )
                        SELECT
                            STR1,
                            STR2
                        FROM
                            DUAL;

                        P.SILVER_QUANTITY := P.SILVER_QUANTITY
                            - 1;
```

```
--Carta di tipo <club_item> di rarita' <bronze>.
            ELSIF P.BRONZE_QUANTITY > 0 THEN
                    STR2 := DBMS_RANDOM.STRING('a', 8);
                    INSERT INTO CLUB_CARD (
                        CARD_CODE,
                        CLUB_ITEM_ID
                    )
                        SELECT
                            *
                        FROM
                            (
                                SELECT
                                    STR2,
                                    CARD_ID
                                FROM
                                    CLUB_ITEM
                                WHERE
                                    RARITY_NAME = 'Bronze'
                                ORDER BY
                                    DBMS_RANDOM.RANDOM
                            )
                        WHERE
                            ROWNUM <= 1;

                    INSERT INTO IS_FOUND (
                        P_ID,
                        CARD_CODE
                    )
                        SELECT
                            STR1,
                            STR2
                        FROM
                            DUAL;


                        P.BRONZE_QUANTITY := P.BRONZE_QUANTITY
                           - 1;
            ELSE
                DBMS_OUTPUT.PUT_LINE('Error inserimento
                   club_items');
            END IF;

            END LOOP;

        END IF;

        IF P.STAFF > 0 THEN
        FOR COUNTER IN 1..P.STAFF LOOP
--Carta di tipo <manager> di rarita' <gold>.
            IF P.GOLD_QUANTITY > 0 THEN
                    STR2 := DBMS_RANDOM.STRING('a', 8);
                    INSERT INTO CLUB_CARD (
                        CARD_CODE,
                        MANAGER_ID
                    )
                        SELECT
                            *
```

```
                        FROM
                            (
                                SELECT
                                    STR2,
                                    CARD_ID
                                FROM
                                    MANAGER
                                WHERE
                                    RARITY_NAME = 'Gold'
                                ORDER BY
                                    DBMS_RANDOM.RANDOM
                            )
                        WHERE
                            ROWNUM <= 1;

                    INSERT INTO IS_FOUND (
                        P_ID,
                        CARD_CODE
                    )
                        SELECT
                            STR1,
                            STR2
                        FROM
                            DUAL;


                    SELECT
                            LEAGUE_NAME
                    INTO    M_LEAGUE
                    FROM
                            MANAGER M
                        JOIN CLUB_CARD C ON M.CARD_ID = C.
                           MANAGER_ID
                                        AND C.CARD_CODE =
                                           STR2;

--Inserisco i dati attivi in <active_data_manager>.
                    INSERT INTO ACTIVE_DATA_MANAGER (
                            M_CARD_CODE,
                            MANAGER_LEAGUE,
                            CONTRACTS
                    )
                            SELECT
                                STR2,
                                M_LEAGUE,
                                7
                            FROM
                                DUAL;

                        P.GOLD_QUANTITY := P.GOLD_QUANTITY - 1;
--Carta di tipo <manager> di rarita' <silver>.
             ELSIF P.SILVER_QUANTITY > 0 THEN
                    STR2 := DBMS_RANDOM.STRING('a', 8);
                    INSERT INTO CLUB_CARD (
                        CARD_CODE,
                        MANAGER_ID
                    )
```

```
                        SELECT
                            *
                        FROM
                            (
                                SELECT
                                    STR2,
                                    CARD_ID
                                FROM
                                    MANAGER
                                WHERE
                                    RARITY_NAME = 'Silver'
                                ORDER BY
                                    DBMS_RANDOM.RANDOM
                            )
                        WHERE
                            ROWNUM <= 1;

                    INSERT INTO IS_FOUND (
                        P_ID,
                        CARD_CODE
                    )
                        SELECT
                            STR1,
                            STR2
                        FROM
                            DUAL;


                    SELECT
                            LEAGUE_NAME
                    INTO    M_LEAGUE
                    FROM
                            MANAGER M
                        JOIN CLUB_CARD C ON M.CARD_ID = C.
                           MANAGER_ID
                                        AND C.CARD_CODE =
                                           STR2;

--Inserisco i dati attivi in <active_data_manager>.
                    INSERT INTO ACTIVE_DATA_MANAGER (
                            M_CARD_CODE,
                            MANAGER_LEAGUE,
                            CONTRACTS
                    )
                            SELECT
                                STR2,
                                M_LEAGUE,
                                7
                            FROM
                                DUAL;

                        P.SILVER_QUANTITY := P.SILVER_QUANTITY
                           - 1;
--Carta di tipo <manager> di rarita' <bronze>.
            ELSIF P.BRONZE_QUANTITY > 0 THEN
                    STR2 := DBMS_RANDOM.STRING('a', 8);
                    INSERT INTO CLUB_CARD (
```

```
                        CARD_CODE,
                        MANAGER_ID
                    )
                        SELECT
                            *
                        FROM
                            (
                                SELECT
                                    STR2,
                                    CARD_ID
                                FROM
                                    MANAGER
                                WHERE
                                    RARITY_NAME = 'Bronze'
                                ORDER BY
                                    DBMS_RANDOM.RANDOM
                            )
                        WHERE
                            ROWNUM <= 1;

                    INSERT INTO IS_FOUND (
                        P_ID,
                        CARD_CODE
                    )
                        SELECT
                            STR1,
                            STR2
                        FROM
                            DUAL;


                    SELECT
                            LEAGUE_NAME
                    INTO    M_LEAGUE
                    FROM
                            MANAGER M
                        JOIN CLUB_CARD C ON M.CARD_ID = C.
                           MANAGER_ID
                                        AND C.CARD_CODE =
                                           STR2;

--Inserisco i dati attivi in <active_data_manager>.
                    INSERT INTO ACTIVE_DATA_MANAGER (
                            M_CARD_CODE,
                            MANAGER_LEAGUE,
                            CONTRACTS
                    )
                            SELECT
                                STR2,
                                M_LEAGUE,
                                7
                            FROM
                                DUAL;

                        P.BRONZE_QUANTITY := P.BRONZE_QUANTITY
                           - 1;
            ELSE
```

```
                DBMS_OUTPUT.PUT_LINE('Error inserimento manager
                    ');
            END IF;

            END LOOP;

        END IF;
--Se il pacchetto che si vuole acquistare non e' disponibile.
    ELSIF P.AVAILABLE = '0' THEN
        DBMS_OUTPUT.PUT_LINE('Il pacchetto selezionato non e''
            disponibile!');
--Se viene inserito un nome di un pacchetto inesistente.
    ELSE
        DBMS_OUTPUT.PUT_LINE('Il pacchetto selezionato non
            esiste!');
    END IF;

    COMMIT;
END IF;

EXCEPTION
WHEN NO_USER_FOUND THEN
    RAISE_APPLICATION_ERROR('-20001', 'L''utente '
                                        || USER
                                        || ' non e'' stato
                                            trovato!');
WHEN USER_ERROR THEN
    RAISE_APPLICATION_ERROR('-20003', 'L''utente '
                                        || USR_1
                                        || ' non corrisponde
                                            all''utente loggato
                                            '
                                        || USER);
END PACK_OPENING;
/
```

11.1.6 Add Player

```
-- Tabelle interessate: 5
--  -> CLUB_CARD, PLAYER, IS_PART_OF, SQUAD;

-- Funzioni interessate: 4
--  -> IS_ADMIN, CARD_IN_CLUB, GET_MANAGER_CHEMISTRY,
   GET_PLAYER_CHEMISTRY

-- INPUT:
--      -> c_code:  codice carta di tipo <player>;
--      -> s_name:  nome della squadra dove schierare il <
   player>;
--      -> cc_name: se il chiamante e' amministratore: contiene
    il nome del club su cui effettuare le operazioni;
--              se il chiamante non e' amministratore: contiene
    o NULL, oppure il nome del club dell'utente chiamante.
--  -> p_position: posizione in campo del <player>;
--  -> flag: 1 = titolare, 0 = panchina;
-- OUTPUT:
--  -> Aggiunge il player <c_code> alla squadra <s_name>.

CREATE OR REPLACE PROCEDURE ADD_PLAYER (
C_CODE      CLUB_CARD.CARD_CODE%TYPE,
S_NAME      SQUAD.NAME%TYPE,
CC_NAME     CLUB.CLUB_NAME%TYPE,
P_POSITION  PLAYER.POSITION%TYPE,
FLAG        CHAR
) IS

N1     NUMBER(1, 0); --Controlla se il giocatore e' gia
   presente nella squadra.
N2     NUMBER(2, 0); --Controlla il massimo numero di giocatori
    di una squadra (max=18).
N3     NUMBER(2, 0); --Controlla se il giocatore e' presente in
    altre squadre dello stesso club.
POS    PLAYER.POSITION%TYPE; --Contiene la posizione naturale
   del <player>.
OVE    PLAYER.OVERALL%TYPE; --Valutazione <overall> del player.
C_ID   PLAYER.CARD_ID% TYPE; --Card_id del <player> che si
   vuole aggiungere.
P_NAME PLAYER.PLAYER_NAME%TYPE; --Nome del <player>.
HOLD   CHAR; --Se e' uguale a 0 il giocatore andra' in panchina
    altrimenti titolare.

--Controllo utente
C_NAME CLUB.CLUB_NAME%TYPE; --Nome effettivo del club (is_admin
   ).

--Controllo posizioni modulo scelto
A_MODULES VARCHAR2(10); --Modulo della squadra <s_name>.
P_CHEM ARRAY_POS_T; --Posizioni del modulo <a_modules>.
CNT_POS NUMBER(1, 0); --Quante posizioni uguali a quella scelta
    ci sono libere nel modulo <a_modules>.

--Aggiornamento Rating di squadra
S_RATING NUMBER(2, 0); --Valutazione della squadra.
```

```
--Aggiornamento intesa di squadra
S_CHEMISTRY NUMBER(3, 0);--Intesa della squadra.
M_CHEMISTRY NUMBER(1, 0);--Bonus intesa dato dal manager.
CHEMISTRY NUMBER(2, 0);--Intesa <chemistry> del <player>.

--Conterra' una query, nel caso in cui la posizione scelta <
   p_position> e' diversa dalla posizione naturale del <player>
-- la valutazione del giocatore <player_rating> cambia in base
   alla posizione, di conseguenza viene
-- selezionata la nuova valutazione <player_rating> in base
   alla posizione <p_position>.
PLSQL_QUERY_RATING_POS  VARCHAR(500);
EXISTING_PLAYER EXCEPTION; --Il player e' gia' presente in
   squadra.
MAX_PLAYER EXCEPTION; --Raggiunto limite massimo di <player>
   per la squadra.
INVALID_FLAG EXCEPTION; --Flag diverso da 0 o 1.

BEGIN
--La funzione is_admin, ritorna il nome del club in base all'
   utente che chiama la funzione ADD_PLAYER;
--Se e' l'amministratore a chiamare la funzione il nome del
   club <c_name> sara' uguale a <cc_name>
--Altrimenti il nome del club <c_name> sara' quello dell'utente
   .
--In breve l'amministratore puo' eseguire la funzione su
   qualsiasi club, l'utente solo sul suo club.
SELECT
    IS_ADMIN(CC_NAME, USER)
INTO C_NAME
FROM
    DUAL;

--La funzione card_in_club, controlla che il club identificato
   dal nome <c_name> possieda la carta identificata dal <c_code
   >
-- se n1 != 0 il club possiede la carta, altrimenti no.
SELECT
    CARD_IN_CLUB(C_CODE, C_NAME)
INTO N1
FROM
    DUAL;

--Se <n1 = 0>, il <player> con codice carta <c_code> non e'
   stato trovato.
IF N1 = 0 THEN
    RAISE NO_DATA_FOUND;
END IF;

--Seleziono il nome,posizione e valutazione del <player>.
SELECT
    PLAYER_NAME,
    POSITION,
    OVERALL
INTO
    P_NAME,
```

```
    POS,
    OVE
FROM
            CLUB_CARD CC
    JOIN PLAYER P ON CC.PLAYER_ID = P.CARD_ID
WHERE
    CC.CARD_CODE = C_CODE;

--Controllo se il <player> e' gia presente nella squadra <
   s_name>.
SELECT
    COUNT(PLAYER_CARD_CODE)
INTO N1
FROM
            IS_PART_OF IPF
    JOIN CLUB_CARD  CC ON IPF.PLAYER_CARD_CODE = CC.CARD_CODE
    JOIN PLAYER     P ON CC.PLAYER_ID = P.CARD_ID
WHERE
        S_NAME = SQUAD_NAME
    AND ( PLAYER_CARD_CODE = C_CODE
            OR P.PLAYER_NAME = P_NAME );

--Controlla quanti player ci sono in squadra (max=18).
SELECT
    COUNT(PLAYER_CARD_CODE)
INTO N2
FROM
    IS_PART_OF
WHERE
        S_NAME = SQUAD_NAME;

--Se <n1 != 0> il <player> e' presente nella squadra.
IF N1 != 0 THEN
    RAISE EXISTING_PLAYER;
--Raggiunto limite massimo giocatori in squadra.
ELSIF N2 > 18 THEN
    RAISE MAX_PLAYER;
ELSE
--Se la posizione scelta <p_position> e' diversa da quella
   naturale
-- e se la posizione e diversa da 'GK' cioe' portiere;
   seleziono
-- la valutazione <player_rating> del <player> in base alla
   posizione scelta.
    IF
        P_POSITION != POS
        AND P_POSITION != 'GK'
    THEN
        PLSQL_QUERY_RATING_POS := 'SELECT '
                                  || P_POSITION
                                  || ' FROM CLUB_CARD CC JOIN
                                      PLAYER P ON CC.
                                     PLAYER_ID = P.CARD_ID
                                     WHERE CC.CARD_CODE='''
                                  || C_CODE
                                  || '''';
        EXECUTE IMMEDIATE PLSQL_QUERY_RATING_POS
```

```
        INTO OVE;
    END IF;

--Se la posizione scelta <p_position> non corrisponde a quella
   naturale <pos>, ed e' uguale a 'GK'
-- si vuole schierare in porta, un giocatore non portiere, di
   conseguenza la sua valutazione <ove> cambia drasticamente.
    IF
        P_POSITION != POS
        AND P_POSITION = 'GK'
    THEN
        OVE := 20;
    END IF;

--Quanti <player> della squadra in quella posizione sono
   titolari = N2.
    SELECT
        COUNT(PLAYER_CARD_CODE)
    INTO N2
    FROM
        IS_PART_OF
    WHERE
            PLAYER_POSITION = P_POSITION
        AND HOLDER = '1'
        AND SQUAD_NAME = S_NAME;

--Seleziono il modulo della squadra <s_name>.
    SELECT
        MODULES
    INTO A_MODULES
    FROM
        SQUAD
    WHERE
            NAME = S_NAME
        AND SQUAD_CLUB_NAME = C_NAME;

--La funzione MODULE_POSITIONS, prende in input un modulo e
   ritorna le sue posizioni.
--Esempio:    <a_modules> = '4-3-3'
--  output: ('GK','LB','CB','CB','RB','CM','CM','CM','LW','ST
   ','RW');
    SELECT
        MODULE_POSITIONS(A_MODULES)
    INTO P_CHEM
    FROM
        DUAL;

--Quante posizioni ci sono nel modulo che corrispondono a
   quella scelta <p_position>.
    CNT_POS := 0;



--Se <flag != 0>, il <player> andra' titolare.
    IF FLAG != '0' THEN

--Vediamo quante posizioni ci sono nel modulo che corrispondono
```

```
    a quella scelta.
-- Perche' ad esempio nel modulo '4-3-3' ci sono 3 'CM'
   centrocampisti centrali.
        FOR I IN 1..11 LOOP
            IF P_CHEM(I) = P_POSITION THEN
                CNT_POS := CNT_POS + 1;
            END IF;
        END LOOP;

--Se <n2 = 0> non ci sono <player> titolari in quella posizione
   .
--E <cnt_pos > 0> significa che il modulo contiene la posizione
    scelta.
        IF
            N2 = 0
            AND CNT_POS > 0
        THEN
            HOLD := '1'; --Il <player> diventa titolare.

--Se <n2 > 0> ci sono <player> titolari in posizione <
   p_position>.
--E se <n2 < CNT_POS>, ci sono posizioni libere nel ruolo.
--Esempio: vogliamo aggiungere un difensore centrale, c'e' gia'
    un difensore titolare, ma se ne possono schierare 2.
        ELSIF
            N2 > 0
            AND N2 < CNT_POS
        THEN
            HOLD := '1'; --Il <player> diventa titolare.
        ELSE--Non ci sono posizioni disponibili nel modulo, il
           <player> va nelle riserve.
            HOLD := '0';
        END IF;
--Se <flag = 0>, il <player> va schierato nelle riserve.
    ELSIF FLAG = '0' THEN
        HOLD := '0';
--Se viene inserito un flag diverso da 0 o 1.
    ELSE
        RAISE INVALID_FLAG;
    END IF;

--Funzione che calcola l'intesa <chemistry> del <player>.
    SELECT
        GET_PLAYER_CHEMISTRY(P_POSITION, POS)
    INTO CHEMISTRY
    FROM
        DUAL;

--Calcolo bonus intesa del manager.
--MANAGER_CHEMISTRY ritorna 1 se il manager ha nazionalita' o
   lega uguale al player altrimenti 0;
    SELECT
        MANAGER_CHEMISTRY(C_ID, C_CODE, S_NAME, C_NAME)
    INTO M_CHEMISTRY
    FROM
        DUAL;
```

```
--Se il bonus e 1 e l'intesa <chemistry> del <player> non e' al
    massimo <10>.
    IF
        M_CHEMISTRY IS NOT NULL
        AND CHEMISTRY < 10
    THEN
--Aggiungo il bonus dato dal manager a l'intesa del <player>.
        CHEMISTRY := CHEMISTRY + M_CHEMISTRY;
    END IF;

--Inserisco il <player> in squadra.
    INSERT INTO IS_PART_OF (
        PLAYER_CARD_CODE,
        SQUAD_NAME,
        PLAYER_POSITION,
        PLAYER_CHEMISTRY,
        PLAYER_RATING,
        HOLDER
    )
        SELECT
            C_CODE,
            S_NAME,
            P_POSITION,
            CHEMISTRY,
            OVE,
            HOLD
        FROM
            DUAL;

--Calcolo della nuova valutazione <s_rating> della squadra.
    SELECT
        GET_SQUAD_RATING(S_NAME, C_NAME)
    INTO S_RATING
    FROM
        DUAL;

--Inseirsco la nuova valutazione <s_rating> della squadra.
    UPDATE SQUAD
    SET
        SQUAD_RATING = S_RATING
    WHERE
            NAME = S_NAME
        AND SQUAD_CLUB_NAME = C_NAME;

--Calcolo intesa della squadra come somma dell'intesa di ogni <
   player>.
    SELECT
        SUM(PLAYER_CHEMISTRY)
    INTO S_CHEMISTRY
    FROM
        IS_PART_OF
    WHERE
            SQUAD_NAME = S_NAME
        AND HOLDER = '1';

--Inseirsco la nuova intesa <s_chemistry> della squadra.
    UPDATE SQUAD
```

```
    SET
        SQUAD_CHEMISTRY = S_CHEMISTRY
    WHERE
            NAME = S_NAME
        AND SQUAD_CLUB_NAME = C_NAME;

--Confermo
    COMMIT;
END IF;

EXCEPTION
WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR(-20001, 'Giocatore non trovato!'
                                    || CHR(10)
                                    || SQLCODE
                                    || ' - ERROR - '
                                    || SQLERRM);
WHEN EXISTING_PLAYER THEN
    DBMS_OUTPUT.PUT_LINE('Il giocatore '
                            || P_NAME
                            || ' e'' gia'' presente in '
                            || S_NAME);
WHEN MAX_PLAYER THEN
    RAISE_APPLICATION_ERROR(-20004, 'Impossibile aggiungere
       giocatore, hai raggiunto il limite massimo!');
WHEN INVALID_FLAG THEN
    RAISE_APPLICATION_ERROR(-20003, 'Flag error!');
END ADD_PLAYER;
/
```

11.1.7 Remove Player

```
-- Tabelle interessate: 2
--  -> IS_PART_OF, SQUAD;

-- Funzioni interessate: 3
--  -> IS_ADMIN;

-- INPUT:
--      -> c_code: codice carta del player da rimuovere;
--      -> s_name: nome della squadra in cui e' schierato il
   player;
--      -> cc_name: se il chiamante e' amministratore: contiene
    il nome del club su cui effettuare le operazioni;
--              se il chiamante non e' amministratore: contiene
    o NULL, oppure il nome del club dell'utente chiamante.
-- OUTPUT:
--  -> Applica il consumabile al player.

CREATE OR REPLACE PROCEDURE REMOVE_PLAYER (
C_CODE   IN  CLUB_CARD.CARD_CODE%TYPE,
S_NAME   IN  SQUAD.NAME%TYPE,
CC_NAME  IN  SQUAD.SQUAD_CLUB_NAME%TYPE
) IS
--Contatori di controllo.
N1           NUMBER(2, 0);
N2           NUMBER(2, 0);
--Nome effettivo del club (is_admin).
C_NAME       CLUB.CLUB_NAME%TYPE;
--Aggiornamento Rating di squadra
S_RATING     NUMBER(2, 0);
--Aggiornamento intesa di squadra
S_CHEMISTRY  NUMBER(3, 0);.
NO_PLAYER EXCEPTION; --Raggiunto limite minimo di <player> per
   la squadra.

BEGIN
--La funzione is_admin, ritorna il nome del club in base all'
   utente che chiama la funzione REMOVE_PLAYER;
--Se e' l'amministratore a chiamare la funzione il nome del
   club <c_name> sara' uguale a <cc_name>
--Altrimenti il nome del club <c_name> sara' quello dell'utente
   .
--In breve l'amministratore puo' eseguire la funzione su
   qualsiasi club, l'utente solo sul suo club.
SELECT
    IS_ADMIN(CC_NAME, USER)
INTO C_NAME
FROM
    DUAL;

--Controllo che ci sia almeno un player nella squadra <s_name>
SELECT
    COUNT(PLAYER_CARD_CODE)
INTO N2
FROM
    IS_PART_OF
```

```
WHERE
        S_NAME = SQUAD_NAME;

IF N2 = 0 THEN
    RAISE NO_PLAYER;
END IF;

--Controllo che il player <c_code> giochi per la squadra <
   s_name>.
SELECT
    COUNT(*)
INTO N1
FROM
    IS_PART_OF
WHERE
        SQUAD_NAME = S_NAME
    AND PLAYER_CARD_CODE = C_CODE;

--Se <n1 = 0>, il player con codice carta <c_code> non e' stato
    trovato.
IF N1 = 0 THEN
    RAISE NO_DATA_FOUND;
ELSE
--Elimino il player dalla squadra <s_name>.
    DELETE FROM IS_PART_OF
    WHERE
            PLAYER_CARD_CODE = C_CODE
        AND SQUAD_NAME = S_NAME;

END IF;

--Calcolo della nuova valutazione <s_rating> della squadra.
SELECT
    GET_SQUAD_RATING(S_NAME, C_NAME)
INTO S_RATING
FROM
    DUAL;

--Inseirsco la nuova valutazione <s_rating> della squadra.
UPDATE SQUAD
SET
    SQUAD_RATING = S_RATING
WHERE
        NAME = S_NAME
    AND SQUAD_CLUB_NAME = C_NAME;

--Calcolo intesa della squadra come somma dell'intesa di ogni <
   player>.
SELECT
    SUM(PLAYER_CHEMISTRY)
INTO S_CHEMISTRY
FROM
    IS_PART_OF
WHERE
        SQUAD_NAME = S_NAME
    AND HOLDER = '1';
```

```
--Inseirsco la nuova intesa <s_chemistry> della squadra.
UPDATE SQUAD
SET
    SQUAD_CHEMISTRY = S_CHEMISTRY
WHERE
        NAME = S_NAME
    AND SQUAD_CLUB_NAME = C_NAME;

--Confermo
COMMIT;
EXCEPTION
WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR(-20001, 'Giocatore non trovato!'
                                    || CHR(10)
                                    || SQLCODE
                                    || ' - ERROR - '
                                    || SQLERRM);
WHEN NO_PLAYER THEN
    RAISE_APPLICATION_ERROR(-20004, 'Impossibile rimuovere
       giocatore, la squadra e'' vuota!');
END REMOVE_PLAYER;
/
```

11.1.8 Change Player

```
-- Tabelle interessate: 1
--  -> IS_PART_OF;

-- Funzioni interessate: 1
--  -> IS_ADMIN;

-- INPUT:
--      -> p_code_old:  codice carta del player;
--      -> c_code_new:  codice carta del player;
--  -> s_name: nome della squadra;
--      -> cc_name: se il chiamante e' amministratore: contiene
    il nome del club su cui effettuare le operazioni;
--              se il chiamante non e' amministratore: contiene
    o NULL, oppure il nome del club dell'utente chiamante.
-- OUTPUT:
--  -> Se il player <p_code_old> e titolare andra' nelle
   riserve
--      e il player <p_code_new> diventera titolare, o
   viceversa.

CREATE OR REPLACE PROCEDURE CHANGE_PLAYER (
P_CODE_OLD   CLUB_CARD.CARD_CODE%TYPE,
P_CODE_NEW   CLUB_CARD.CARD_CODE%TYPE,
S_NAME       CLUB.CLUB_NAME%TYPE,
CC_NAME      CLUB.CLUB_NAME%TYPE
) IS

N1       NUMBER(2, 0);
N2       NUMBER(2, 0);
C_NAME   CLUB.CLUB_NAME%TYPE; --Nome effettivo del club (
   is_admin).
SAME     NUMBER(1,0):= 0; --Se e' = 1, i player sono entrambi
   nelle riserve o entrambi titolari.

NO_DATA_FOUND_OLD EXCEPTION; --Il player con codice carta <
   p_code_old> non e' stato trovato.
NO_DATA_FOUND_NEW EXCEPTION; --Il player con codice carta <
   p_code_new> non e' stato trovato.
SAME_HOLDER_0 EXCEPTION; --Se i player sono entrambi nelle
   riserve.
SAME_HOLDER_1 EXCEPTION; --Se i player sono entrambi titolari.

BEGIN
--La funzione is_admin, ritorna il nome del club in base all'
   utente che chiama la funzione CHANGE_PLAYER;
--Se e' l'amministratore a chiamare la funzione il nome del
   club <c_name> sara' uguale a <cc_name>
--Altrimenti il nome del club <c_name> sara' quello dell'utente
   .
--In breve l'amministratore puo' eseguire la funzione su
   qualsiasi club, l'utente solo sul suo club.
SELECT
    IS_ADMIN(CC_NAME, USER)
INTO C_NAME
FROM
```

```
    DUAL;

--Controllo se <p_code_old> corrisponde a <p_code_new>.
IF P_CODE_OLD = P_CODE_NEW THEN
    SAME:= 1;
END IF;

--Controllo se il player <p_code_old> e' titolare.
SELECT
    COUNT(*)
INTO N1
FROM
    IS_PART_OF
WHERE
        SQUAD_NAME = S_NAME
    AND PLAYER_CARD_CODE = P_CODE_OLD
    AND HOLDER = '1';

--Se <n1 = 0>, il player <p_code_old> non e' titolare.
--Controllo se e' nelle riserve.
IF N1 = 0 THEN
    SELECT
        COUNT(*)
    INTO N2
    FROM
        IS_PART_OF
    WHERE
            SQUAD_NAME = S_NAME
        AND PLAYER_CARD_CODE = P_CODE_OLD
        AND HOLDER = '0';

--Se <n2 = 0>, il player con codice carta <p_code_old> non e'
   stato trovato.
    IF N2 = 0 THEN
        RAISE NO_DATA_FOUND_OLD;
    END IF;
END IF;

--Se <n1 != 0>, il player <p_code_old> e' titolare.
IF N1 <> 0 THEN
    --Se <p_code_old> corrisponde a <p_code_new> cambio il
       valore holder di <p_code_old>.
    IF SAME = 1 THEN
        UPDATE IS_PART_OF
        SET HOLDER = '0'
        WHERE PLAYER_CARD_CODE = P_CODE_OLD;
    ELSE
    --Controllo che il player con <p_code_new> e' nelle riserve
       .
        SELECT
            COUNT(*)
        INTO N1
        FROM
            IS_PART_OF
        WHERE
                SQUAD_NAME = S_NAME
            AND PLAYER_CARD_CODE = P_CODE_NEW
```

```
            AND HOLDER = '0';

        --Se <n1 = 0>, il player <p_code_new> non e' titolare.
        -- controllo se e' nelle riserve.
        IF N1 = 0 THEN
            SELECT
            COUNT(*)
            INTO N1
            FROM
                IS_PART_OF
            WHERE
                    SQUAD_NAME = S_NAME
                AND PLAYER_CARD_CODE = P_CODE_NEW
                AND HOLDER = '1';

            --Se <n1 = 0>, il player <p_code_new> non e' stato
               trovato.
            if N1 = 0 THEN
                RAISE NO_DATA_FOUND_NEW;

            --Altrimenti i player sono entrambi titolari e non
               c'e' niente da cambiare.
            ELSE
                RAISE SAME_HOLDER_1;
            END IF;

        --Se <n1 != 0>, il player <p_code_new> e' nelle riserve
            e diventera' titolare.
        ELSE
    --Imposto <holder> del player <p_code_new> ad '1', cioe'
       titolare.
            UPDATE IS_PART_OF
            SET
                HOLDER = '1'
            WHERE
                    SQUAD_NAME = S_NAME
                AND PLAYER_CARD_CODE = P_CODE_NEW;

    --Il player <p_code_old> andra' nelle riserve, imposto <
       holder> = '0'.
            UPDATE IS_PART_OF
            SET
                HOLDER = '0'
            WHERE
                    SQUAD_NAME = S_NAME
                AND PLAYER_CARD_CODE = P_CODE_OLD;

    --Confermo
            COMMIT;
        END IF;
    END IF;
--Se <n2 != 0>, il player <p_code_old> e' nelle riserve.
ELSIF N2 <> 0 THEN
    --Se <p_code_old> corrisponde a <p_code_new> cambio il
       valore holder di <p_code_old>.
    IF SAME = 1 THEN
        UPDATE IS_PART_OF
```

```
        SET HOLDER = '1'
        WHERE PLAYER_CARD_CODE = P_CODE_OLD;
    ELSE
        --Controllo se il player <p_code_new> e' titolare.
        SELECT
        COUNT(*)
        INTO N1
        FROM
            IS_PART_OF
        WHERE
                SQUAD_NAME = S_NAME
            AND PLAYER_CARD_CODE = P_CODE_NEW
            AND HOLDER = '1';

        --Il player <p_code_new> non e' titolare, controllo se
           e' nelle riserve.
        IF N1 = 0 THEN
            SELECT
            COUNT(*)
            INTO N1
            FROM
                IS_PART_OF
            WHERE
                    SQUAD_NAME = S_NAME
                AND PLAYER_CARD_CODE = P_CODE_NEW
                AND HOLDER = '0';

            --Se <n1 = 0>, il player <p_code_new> non e' stato
               trovato.
            IF N1 = 0 THEN
                RAISE NO_DATA_FOUND_NEW;

            --Se <n1 != 0>, i player sono entrambi nelle
               riserve, non c'e' niente da cambiare.
            ELSIF N1 != 0 THEN
                RAISE SAME_HOLDER_0;
            END IF;
        END IF;

        --Se <n1 != 0>, il player <p_code_new> era titolare e
           andra' nelle riserve <holder = '0'>.
        IF N1 <> 0 THEN
            UPDATE IS_PART_OF
            SET
                HOLDER = '0'
            WHERE
                    SQUAD_NAME = S_NAME
                AND PLAYER_CARD_CODE = P_CODE_NEW;

    --Il player <p_code_old> andra' titolare <holder = '1'>.
            UPDATE IS_PART_OF
            SET
                HOLDER = '1'
            WHERE
                    SQUAD_NAME = S_NAME
                AND PLAYER_CARD_CODE = P_CODE_OLD;
```

```
    --Confermo
            COMMIT;
        END IF;
    END IF;
END IF;

EXCEPTION
WHEN NO_DATA_FOUND_OLD THEN
    RAISE_APPLICATION_ERROR('-20001', 'Il giocatore con codice
       carta ('
                                        || P_CODE_OLD
                                        || ') non e'' stato
                                           trovato!');
WHEN NO_DATA_FOUND_NEW THEN
    RAISE_APPLICATION_ERROR('-20001', 'Il giocatore con codice
       carta ('
                                        || P_CODE_NEW
                                        || ') non e'' stato
                                           trovato!');
WHEN SAME_HOLDER_0 THEN
    RAISE_APPLICATION_ERROR('-20003', 'I giocatori con codici
       carta ('
                                        || P_CODE_OLD
                                        || ') - ('
                                        || P_CODE_NEW
                                        || ') sono entrambi
                                           nelle riserve!');
WHEN SAME_HOLDER_1 THEN
    RAISE_APPLICATION_ERROR('-20003', 'I giocatori con codici
       carta ('
                                        || P_CODE_OLD
                                        || ') - ('
                                        || P_CODE_NEW
                                        || ') sono entrambi
                                           titolari!');

END CHANGE_PLAYER;
/
```

11.1.9 Insert Match

```
-- Tabelle interessate: 5
--  -> CLUB, MATCH, IS_PART_OF, SQUAD;
-- Funzioni interessate: 2
--  -> IS_ADMIN;

-- INPUT:
--      -> s_name:  nome della squadra;
--      -> cc_name: se il chiamante e' amministratore: contiene
    il nome del club su cui effettuare le operazioni;
--              se il chiamante non e' amministratore: contiene
    o NULL, oppure il nome del club dell'utente chiamante.
-- OUTPUT:
--  -> Aggiunge la squadra <s_name> ad un match.
CREATE OR REPLACE PROCEDURE INSERT_MATCH (
    S_NAME   SQUAD.NAME%TYPE,
    CC_NAME  CLUB.CLUB_NAME%TYPE
) IS
--Variabili di controllo
    RAND_VALUE      NUMBER(3, 0); --Numero casuale.
    N1              NUMBER(2, 0);
    N2              NUMBER(2, 0);
    N3              NUMBER(4, 0);
    FLAG            NUMBER(1, 0);

    C_NAME          CLUB.CLUB_NAME%TYPE; --Nome del club
       effettivo (is_admin).
    USER_SQ         SQUAD%ROWTYPE;  --Tupla squadra utente.
    HOME_SQ         SQUAD%ROWTYPE;  --Tupla squadra in casa.
    AWAY_SQ         SQUAD%ROWTYPE;  --Tupla squadra fuori casa.
    HOME_W_P        NUMBER(2, 0);    --Percentuale vittoria
       squadra in casa.
    AWAY_W_P        NUMBER(2, 0);    --Percentuale vittoria
       squadra fuori casa.
    DRAW_P          NUMBER(2, 0);    --Percentuale pareggio.
    RAND_VALUE_2    NUMBER(2, 0);    --Numero casuale.
    NEW_MATCH       MATCH%ROWTYPE;   --Tupla da caricare.
    FF_HOME         NUMBER(4,0);     -- A determinare il
       risultato del match contribuisce
    FF_AWAY         NUMBER(4,0);     -- anche la forma fisica
       delle due squadre

    AWAY_NAME       SQUAD.NAME%TYPE; --Nome della squadra fuori
        casa.
    AWAY_CLUB_NAME  SQUAD.SQUAD_CLUB_NAME%TYPE; --Nome del club
        della squadra fuori casa.

    HOME_NAME       SQUAD.NAME%TYPE; --Nome della squadra in
       casa.
    HOME_CLUB_NAME  SQUAD.SQUAD_CLUB_NAME%TYPE; --Nome del club
        della squadra in casa.

--ECCEZIONI
    NO_SQUAD EXCEPTION; --Chiamata in causa se la squadra <
       s_name> non viene trovata.
    PLAYER_MIN EXCEPTION; --Per giocare una partita la squadra
```

```
        deve avere minimo 11 titolari.
    MATCH_EXISTING EXCEPTION; --La squadra <s_name> e' in
        attesa di disputare un match.
    CLUB_EXISTING EXCEPTION; --Il club ha gia' un'altra squadra
         in attesa di disputare un match.
BEGIN
    N1 := 0;
    N2 := 0;
    FLAG := 0;
    USER_SQ := NULL;

--La funzione is_admin, ritorna il nome del club in base all'
   utente che chiama la funzione INSERT_MATCH;
--Se e' l'amministratore a chiamare la funzione il nome del
   club <c_name> sara' uguale a <cc_name>
--Altrimenti il nome del club <c_name> sara' quello dell'utente
   .
--In breve l'amministratore puo' eseguire la funzione su
   qualsiasi club, l'utente solo sul suo club.
    SELECT
        IS_ADMIN(CC_NAME, USER)
    INTO C_NAME
    FROM
        DUAL;

--Controllo che la squadra <s_name> esista.
    SELECT
        COUNT(*)
    INTO N1
    FROM
        SQUAD
    WHERE
            NAME = S_NAME
        AND SQUAD_CLUB_NAME = C_NAME;

--Se <n1 = 0>, la squadra <s_name> non e' stata trovata.
    IF N1 = 0 THEN
        RAISE NO_SQUAD;
    END IF;

--Seleziono i dati relativi alla squadra che vuole disputare la
    partita in <user_sq>.
    SELECT
        *
    INTO USER_SQ
    FROM
        SQUAD
    WHERE
            NAME = S_NAME
        AND SQUAD_CLUB_NAME = C_NAME;

--Controllo quanti giocatori titolari ci sono nella squadra <
   s_name>.
    SELECT
        COUNT(*)
    INTO RAND_VALUE
    FROM
```

```
        IS_PART_OF
    WHERE
            SQUAD_NAME = S_NAME
        AND HOLDER = '1';

--Se ci sono meno di 11 giocatori titolari la squadra non puo'
   giocare la partita.
    IF RAND_VALUE < 11 THEN
        RAISE PLAYER_MIN;
    END IF;

--Controllo se ci sono match, con una squadra in casa, in
   attesa dell'avversario.
    SELECT
        COUNT(NAME)
    INTO N1
    FROM
        SQUAD
    WHERE
            SQUAD_CLUB_NAME <> C_NAME
        AND NAME IN (
            SELECT
                HOME_SQUAD_NAME
            FROM
                MATCH
            WHERE
                    HOME_SQUAD_NAME <> S_NAME
                AND VISITORS_SQUAD_NAME IS NULL);

--Se non ci sono match con una squadra in casa,
-- controllo se c'e' ne sono con una squadra fuori casa.
    IF N1 = 0 THEN
        SELECT
            COUNT(NAME)
        INTO N2
        FROM
            SQUAD
        WHERE
                SQUAD_CLUB_NAME <> C_NAME
            AND NAME IN (
                SELECT
                    VISITORS_SQUAD_NAME
                FROM
                    MATCH
                WHERE
                        VISITORS_SQUAD_NAME <> S_NAME
                    AND HOME_SQUAD_NAME IS NULL);
    END IF;

--Non ci sono squadre avversarie in attesa. La squadra <s_name>
    andra' in attesa di un avversario.
    IF
        N1 = 0
        AND N2 = 0
    THEN

--Controllo se la squadra <s_name> e' in attesa.
```

```
        SELECT
            COUNT(MATCH_ID)
        INTO N1
        FROM
            MATCH
        WHERE
            ( HOME_SQUAD_NAME = S_NAME
              AND VISITORS_SQUAD_NAME IS NULL )
            OR ( VISITORS_SQUAD_NAME = S_NAME
                 AND HOME_SQUAD_NAME IS NULL );

--Controllo se altre squadre dello stesso club sono in attesa (
   in casa).
        SELECT
            COUNT(NAME)
        INTO N2
        FROM
            SQUAD
        WHERE
                SQUAD_CLUB_NAME = C_NAME
            AND NAME IN (
                SELECT
                    HOME_SQUAD_NAME
                FROM
                    MATCH
                WHERE
                        HOME_SQUAD_NAME <> S_NAME
                    AND VISITORS_SQUAD_NAME IS NULL
            );

--Controllo se altre squadre dello stesso club sono in attesa (
   fuori casa).
        IF N2 = 0 THEN
            SELECT
                COUNT(NAME)
            INTO N2
            FROM
                SQUAD
            WHERE
                    SQUAD_CLUB_NAME = C_NAME
                AND NAME IN (
                    SELECT
                        VISITORS_SQUAD_NAME
                    FROM
                        MATCH
                    WHERE
                            VISITORS_SQUAD_NAME <> S_NAME
                        AND HOME_SQUAD_NAME IS NULL
                );

        END IF;

--Se ci sono altre squadre dello stesso club in attesa,
-- prima di iniziare una nuova partita deve terminare la
   vecchia.
        IF N2 <> 0 THEN
            RAISE CLUB_EXISTING;
```

```
        END IF;

--Se non e' in attesa la inserisco in attesa.
        IF N1 = 0 THEN

--Se <rand_value = 1> la squadra giochera' in casa altrimenti
   fuori casa.
            RAND_VALUE := DBMS_RANDOM.VALUE(LOW => 1, HIGH =>
               2);
            IF RAND_VALUE = 1 THEN
                INSERT INTO MATCH (
                    MATCH_ID,
                    HOME_SQUAD_NAME
                ) VALUES (
                    MATCH_ID_SEQ.NEXTVAL,
                    S_NAME
                );

                INSERT INTO TMP_MATCH (
                    MATCH_ID,
                    HOME_SQUAD_NAME
                ) VALUES (
                    MATCH_ID_SEQ.CURRVAL,
                    S_NAME
                );

                COMMIT;
            ELSE
                INSERT INTO MATCH (
                    MATCH_ID,
                    VISITORS_SQUAD_NAME
                ) VALUES (
                    MATCH_ID_SEQ.NEXTVAL,
                    S_NAME
                );

                INSERT INTO TMP_MATCH (
                    MATCH_ID,
                    VISITORS_SQUAD_NAME
                ) VALUES (
                    MATCH_ID_SEQ.CURRVAL,
                    S_NAME
                );

                COMMIT;
            END IF;

        ELSE-- Altrimenti viene comunicato all'utente che e' in
            attesa di un avversario.
            RAISE MATCH_EXISTING;
        END IF;
--C'e' una partita con un avversario in casa.
    ELSIF
        N1 <> 0
        AND N2 = 0
    THEN
--Mi assicuro che sia l'ultimo match quello che vado ad
```

```
   analizzare.
        SELECT
            MAX(MATCH_ID)
        INTO N3
        FROM
            MATCH;

--Aggiorno il match corrente inserendo la squadra fuori casa.
        UPDATE MATCH
        SET
            VISITORS_SQUAD_NAME = S_NAME
        WHERE
            MATCH_ID = N3;

        UPDATE TMP_MATCH
        SET
            VISITORS_SQUAD_NAME = S_NAME
        WHERE
            MATCH_ID = N3;

--Significa che sto per giocare una partita fuori casa.
        FLAG := 1;
--C'e' una partita con un avversario fuori casa.
    ELSIF
        N1 = 0
        AND N2 <> 0
    THEN
        SELECT
            MAX(MATCH_ID)
        INTO N3
        FROM
            MATCH;

--Aggiorno il match corrente inserendo la squadra in casa.
        UPDATE MATCH
        SET
            HOME_SQUAD_NAME = S_NAME
        WHERE
            MATCH_ID = N3;

        UPDATE TMP_MATCH
        SET
            HOME_SQUAD_NAME = S_NAME
        WHERE
            MATCH_ID = N3;

--Significa che sto per giocare una partita in casa.
        FLAG := 2;
    END IF;

--Se <flag = 1>, giochera' fuori casa.
    IF FLAG = 1 THEN

--La squadra fuori casa (AWAY_SQ) sara' la squadra dell'utente
        AWAY_SQ := USER_SQ;
        SELECT
            MAX(MATCH_ID)
```

```
        INTO N3
        FROM
            MATCH;

--Seleziono il nome della squadra in <home_name>.
        SELECT
            HOME_SQUAD_NAME
        INTO
            HOME_NAME
        FROM
            MATCH
        WHERE
            MATCH_ID = N3;

--Seleziono i dati della squadra avversaria in <home_sq>.
        SELECT
            *
        INTO HOME_SQ
        FROM
            SQUAD
        WHERE
            NAME = HOME_NAME;

--Calcola il risultato della partita
        FLAG := 3;

--Se flag=2 giochera' in casa
    ELSIF FLAG = 2 THEN

--La squadra che giochera' in casa (HOME_SQ) sara' quella dell'
   utente.
        HOME_SQ := USER_SQ;
        SELECT
            MAX(MATCH_ID)
        INTO N3
        FROM
            MATCH;

--Seleziono il nome della squadra del club avversario in <
   away_name>.
        SELECT
            VISITORS_SQUAD_NAME
        INTO
            AWAY_NAME
        FROM MATCH WHERE MATCH_ID = N3;

--Seleziono i dati della squadra avversaria in <away_sq>.
        SELECT
            *
        INTO AWAY_SQ
        FROM
            SQUAD
        WHERE
                NAME = AWAY_NAME;
--Significa che posso disputare la partita.
        FLAG := 3;
    END IF;
```

```
--Seleziono la forma fisica delle due squadre
--Casa
        SELECT
            SUM(PLAYER_FITNESS)
        INTO FF_HOME
        FROM
              ACTIVE_DATA_PLAYER
        WHERE
                P_CARD_CODE IN (SELECT PLAYER_CARD_CODE
                                FROM
                                    IS_PART_OF
                                WHERE
                                    SQUAD_NAME = HOME_NAME );
--Fuori Casa
        SELECT
            SUM(PLAYER_FITNESS)
        INTO FF_AWAY
        FROM
              ACTIVE_DATA_PLAYER
        WHERE
                P_CARD_CODE IN (SELECT PLAYER_CARD_CODE
                                FROM
                                    IS_PART_OF
                                WHERE
                                    SQUAD_NAME = AWAY_NAME);


--Calcolo del risultato
--  i costrutti if interni controllo il dislivello di
   valutazione della squadra <squad_rating>;
--  quelli interni il dislivello d'intesa <squad_chemistry>;
--  infine il dislivello di forma fisica <player_fitness>.

    IF FLAG = 3 THEN
        IF ABS(HOME_SQ.SQUAD_RATING - AWAY_SQ.SQUAD_RATING) <=
           3 THEN
            IF ABS(HOME_SQ.SQUAD_CHEMISTRY - AWAY_SQ.
               SQUAD_CHEMISTRY) <= 8 THEN
                --Valutazione <rating>, intesa <chemistry> e
                   forma fisisa sono simili.
                IF ABS(FF_HOME - FF_AWAY) <= 10 THEN
                    DRAW_P := 46;
                    HOME_W_P := 27;
                    AWAY_W_P := 27;
                --Valutazione <rating>, intesa <chemistry>
                   simili, forma fisisa maggiore fuori casa.
                ELSIF (FF_HOME - FF_AWAY) < -10 THEN
                    DRAW_P := 46;
                    HOME_W_P := 25;
                    AWAY_W_P := 29;
                --Valutazione <rating>, intesa <chemistry>
                   simili, forma fisisa maggiore in casa.
                ELSIF (FF_HOME - FF_AWAY) > 10  THEN
                    DRAW_P := 46;
                    HOME_W_P := 29;
                    AWAY_W_P := 25;
```

```
            END IF;
        --Valutazione <rating> simile, intesa <chemistry>
           maggiore fuori casa, forma fisisa simile.
        ELSIF ( HOME_SQ.SQUAD_CHEMISTRY - AWAY_SQ.
           SQUAD_CHEMISTRY ) < -8 THEN
            IF ABS(FF_HOME - FF_AWAY) <= 10 THEN
                DRAW_P := 40;
                HOME_W_P := 23;
                AWAY_W_P := 37;
            --Valutazione <rating> simile, intesa <
               chemistry> maggiore fuori casa, forma fisisa
                maggiore fuori casa.
            ELSIF (FF_HOME - FF_AWAY) < -10 THEN
                DRAW_P := 40;
                HOME_W_P := 21;
                AWAY_W_P := 39;
            --Valutazione <rating> simile, intesa <
               chemistry> maggiore fuori casa, forma fisisa
                maggiore in casa.
            ELSIF (FF_HOME - FF_AWAY) > 10  THEN
                DRAW_P := 40;
                HOME_W_P := 25;
                AWAY_W_P := 35;
            END IF;
        --Valutazione <rating> simile, intesa <chemistry>
           maggiore casa, forma fisisa simile.
        ELSIF ( HOME_SQ.SQUAD_CHEMISTRY - AWAY_SQ.
           SQUAD_CHEMISTRY ) > 8 THEN
            IF ABS(FF_HOME - FF_AWAY) <= 10 THEN
                DRAW_P := 40;
                HOME_W_P := 37;
                AWAY_W_P := 23;
        --Valutazione <rating> simile, intesa <chemistry>
           maggiore casa, forma fisisa maggiore fuori casa.
            ELSIF (FF_HOME - FF_AWAY) < -10 THEN
                DRAW_P := 40;
                HOME_W_P := 35;
                AWAY_W_P := 25;
        --Valutazione <rating> simile, intesa <chemistry>
           maggiore casa, forma fisisa maggiore casa.
            ELSIF (FF_HOME - FF_AWAY) > 10  THEN
                DRAW_P := 40;
                HOME_W_P := 39;
                AWAY_W_P := 21;
            END IF;
        END IF;
    ELSIF ( HOME_SQ.SQUAD_RATING - AWAY_SQ.SQUAD_RATING ) <
        -3 THEN
        IF ABS(HOME_SQ.SQUAD_CHEMISTRY - AWAY_SQ.
           SQUAD_CHEMISTRY) <= 8 THEN
    --Valutazione <rating> maggiore fuori casa, intesa <
       chemistry> simile, forma fisisa simile.
            IF ABS(FF_HOME - FF_AWAY) <= 10 THEN
                DRAW_P := 32;
                HOME_W_P := 18;
                AWAY_W_P := 50;
        --Valutazione <rating> maggiore fuori casa, intesa
```

```
            <chemistry> simile, forma fisisa maggiore fuori
            casa.
             ELSIF (FF_HOME - FF_AWAY) < -10 THEN
                 DRAW_P := 32;
                 HOME_W_P := 16;
                 AWAY_W_P := 52;
        --Valutazione <rating> maggiore fuori casa, intesa
            <chemistry> simile, forma fisisa maggiore casa.
             ELSIF (FF_HOME - FF_AWAY) > 10  THEN
                 DRAW_P := 32;
                 HOME_W_P := 18;
                 AWAY_W_P := 50;
             END IF;
        ELSIF ( HOME_SQ.SQUAD_CHEMISTRY - AWAY_SQ.
            SQUAD_CHEMISTRY ) < -8 THEN
        --Valutazione <rating> maggiore fuori casa, intesa
            <chemistry> maggiore fuori casa, forma fisisa
            simile.
             IF ABS(FF_HOME - FF_AWAY) <= 10 THEN
                 DRAW_P := 22;
                 HOME_W_P := 22;
                 AWAY_W_P := 56;
        --Valutazione <rating> maggiore fuori casa, intesa
            <chemistry> maggiore fuori casa, forma fisisa
            maggiore fuori casa.
             ELSIF (FF_HOME - FF_AWAY) < -10 THEN
                 DRAW_P := 22;
                 HOME_W_P := 20;
                 AWAY_W_P := 58;
        --Valutazione <rating> maggiore fuori casa, intesa
            <chemistry> maggiore fuori casa, forma fisisa
            maggiore casa.
             ELSIF (FF_HOME - FF_AWAY) > 10  THEN
                 DRAW_P := 22;
                 HOME_W_P := 24;
                 AWAY_W_P := 54;
             END IF;
        ELSIF ( HOME_SQ.SQUAD_CHEMISTRY - AWAY_SQ.
            SQUAD_CHEMISTRY ) > 8 THEN
        --Valutazione <rating> maggiore fuori casa, intesa
            <chemistry> maggiore casa, forma fisisa simile.
             IF ABS(FF_HOME - FF_AWAY) <= 10 THEN
                 DRAW_P := 18;
                 HOME_W_P := 32;
                 AWAY_W_P := 50;
        --Valutazione <rating> maggiore fuori casa, intesa
            <chemistry> maggiore casa, forma fisisa maggiore
             fuori casa.
             ELSIF (FF_HOME - FF_AWAY) < -10 THEN
                 DRAW_P := 18;
                 HOME_W_P := 30;
                 AWAY_W_P := 52;
        --Valutazione <rating> maggiore fuori casa, intesa
            <chemistry> maggiore casa, forma fisisa maggiore
             casa.
             ELSIF (FF_HOME - FF_AWAY) > 10  THEN
                 DRAW_P := 18;
```

```
                HOME_W_P := 34;
                AWAY_W_P := 48;
            END IF;
        END IF;
    ELSIF ( HOME_SQ.SQUAD_RATING - AWAY_SQ.SQUAD_RATING ) >
        3 THEN
        IF ABS(HOME_SQ.SQUAD_CHEMISTRY - AWAY_SQ.
            SQUAD_CHEMISTRY) <= 8 THEN
        --Valutazione <rating> maggiore casa, intesa <
            chemistry> simile, forma fisisa simile.
            IF ABS(FF_HOME - FF_AWAY) <= 10 THEN
                DRAW_P := 32;
                HOME_W_P := 50;
                AWAY_W_P := 18;
        --Valutazione <rating> maggiore casa, intesa <
            chemistry> simile, forma fisisa maggiore fuori
            casa.
            ELSIF (FF_HOME - FF_AWAY) < -10 THEN
                DRAW_P := 32;
                HOME_W_P := 48;
                AWAY_W_P := 20;
        --Valutazione <rating> maggiore casa, intesa <
            chemistry> simile, forma fisisa maggiore casa.
            ELSIF (FF_HOME - FF_AWAY) > 10  THEN
                DRAW_P := 32;
                HOME_W_P := 52;
                AWAY_W_P := 16;
            END IF;
        ELSIF ( HOME_SQ.SQUAD_CHEMISTRY - AWAY_SQ.
            SQUAD_CHEMISTRY ) < -8 THEN
        --Valutazione <rating> maggiore casa, intesa <
            chemistry> maggiore fuori casa, forma fisisa
            simile.
            IF ABS(FF_HOME - FF_AWAY) <= 10 THEN
                DRAW_P := 18;
                HOME_W_P := 50;
                AWAY_W_P := 32;
        --Valutazione <rating> maggiore casa, intesa <
            chemistry> maggiore fuori casa, forma fisisa
            maggiore fuori casa.
            ELSIF (FF_HOME - FF_AWAY) < -10 THEN
                DRAW_P := 18;
                HOME_W_P := 48;
                AWAY_W_P := 34;
        --Valutazione <rating> maggiore casa, intesa <
            chemistry> maggiore fuori casa, forma fisisa
            maggiore casa.
            ELSIF (FF_HOME - FF_AWAY) > 10  THEN
                DRAW_P := 18;
                HOME_W_P := 52;
                AWAY_W_P := 30;
            END IF;
        ELSIF ( HOME_SQ.SQUAD_CHEMISTRY - AWAY_SQ.
            SQUAD_CHEMISTRY ) > 8 THEN
        --Valutazione <rating> maggiore casa, intesa <
            chemistry> maggiore casa, forma fisisa simile.
            IF ABS(FF_HOME - FF_AWAY) <= 10 THEN
```

```
                    DRAW_P := 22;
                    HOME_W_P := 56;
                    AWAY_W_P := 22;
            --Valutazione <rating> maggiore casa, intesa <
               chemistry> maggiore casa, forma fisisa maggiore
               fuori casa.
                ELSIF (FF_HOME - FF_AWAY) < -10 THEN
                    DRAW_P := 22;
                    HOME_W_P := 54;
                    AWAY_W_P := 24;
            --Valutazione <rating> maggiore casa, intesa <
               chemistry> maggiore casa, forma fisisa maggiore
               casa.
                ELSIF (FF_HOME - FF_AWAY) > 10  THEN
                    DRAW_P := 22;
                    HOME_W_P := 58;
                    AWAY_W_P := 20;
                END IF;
            END IF;
        END IF;

        RAND_VALUE := DBMS_RANDOM.VALUE(LOW => 1, HIGH => 100);
        IF RAND_VALUE <= DRAW_P THEN
--Pareggio.
            RAND_VALUE := DBMS_RANDOM.VALUE(LOW => 0, HIGH =>
               7);
            NEW_MATCH.RESULTS := RAND_VALUE || '-' ||
               RAND_VALUE;
            NEW_MATCH.HOME_POINTS := DBMS_RANDOM.VALUE(LOW =>
               15, HIGH => 30);
            NEW_MATCH.VISITORS_POINTS := NEW_MATCH.HOME_POINTS;
            NEW_MATCH.HOME_CREDITS := DBMS_RANDOM.VALUE(LOW =>
               100, HIGH => 130);
            NEW_MATCH.VISITORS_CREDITS := NEW_MATCH.
               HOME_CREDITS;

        ELSIF RAND_VALUE <= DRAW_P + HOME_W_P THEN
--Vince la squadra in casa.
            RAND_VALUE := DBMS_RANDOM.VALUE(LOW => 1, HIGH =>
               7);
            RAND_VALUE_2 := DBMS_RANDOM.VALUE(LOW => 0, HIGH =>
                RAND_VALUE - 1);
            NEW_MATCH.RESULTS := RAND_VALUE || '-' ||
               RAND_VALUE_2;
            NEW_MATCH.HOME_POINTS := DBMS_RANDOM.VALUE(LOW =>
               45, HIGH => 55);
            NEW_MATCH.VISITORS_POINTS := DBMS_RANDOM.VALUE(LOW
               => - 30, HIGH => - 15);
            NEW_MATCH.HOME_CREDITS := DBMS_RANDOM.VALUE(LOW =>
               200, HIGH => 230);
            NEW_MATCH.VISITORS_CREDITS := DBMS_RANDOM.VALUE(LOW
                => 45, HIGH => 55);

        ELSE
--Vince la squadra fuori casa.
            RAND_VALUE := DBMS_RANDOM.VALUE(LOW => 1, HIGH =>
               7);
```

```
            RAND_VALUE_2 := DBMS_RANDOM.VALUE(LOW => 0, HIGH =>
                RAND_VALUE - 1);
            NEW_MATCH.RESULTS := RAND_VALUE_2 || '-' ||
               RAND_VALUE;
            NEW_MATCH.HOME_POINTS := DBMS_RANDOM.VALUE(LOW => -
                30, HIGH => - 15);
            NEW_MATCH.VISITORS_POINTS := DBMS_RANDOM.VALUE(LOW
               => 45, HIGH => 55);
            NEW_MATCH.HOME_CREDITS := DBMS_RANDOM.VALUE(LOW =>
               45, HIGH => 55);
            NEW_MATCH.VISITORS_CREDITS := DBMS_RANDOM.VALUE(LOW
                => 200, HIGH => 230);
        END IF;

--Aggiornamento punti e crediti della squadre.
        UPDATE CLUB
        SET
            DR_POINTS = DR_POINTS + NEW_MATCH.HOME_POINTS,
            CREDITS = CREDITS + NEW_MATCH.HOME_CREDITS
        WHERE
            CLUB.CLUB_NAME = HOME_SQ.SQUAD_CLUB_NAME;

        UPDATE CLUB
        SET
            DR_POINTS = DR_POINTS + NEW_MATCH.VISITORS_POINTS,
            CREDITS = CREDITS + NEW_MATCH.VISITORS_CREDITS
        WHERE
            CLUB.CLUB_NAME = AWAY_SQ.SQUAD_CLUB_NAME;

--Controllo che i punti non siano negativi.
        UPDATE CLUB
        SET
            DR_POINTS = 0
        WHERE
            DR_POINTS < 0;

        SELECT
            MAX(MATCH_ID)
        INTO N3
        FROM
            MATCH;

--Inserisco il risultato del match.
        UPDATE MATCH
        SET
            M_DATE = SYSDATE,
            RESULTS = NEW_MATCH.RESULTS,
            HOME_POINTS = NEW_MATCH.HOME_POINTS,
            VISITORS_POINTS = NEW_MATCH.VISITORS_POINTS,
            HOME_CREDITS = NEW_MATCH.HOME_CREDITS,
            VISITORS_CREDITS = NEW_MATCH.VISITORS_CREDITS
        WHERE
                HOME_SQUAD_NAME = HOME_SQ.NAME
            AND VISITORS_SQUAD_NAME = AWAY_SQ.NAME
            AND MATCH_ID = N3;

        UPDATE TMP_MATCH
```

```
        SET MATCH_DATE = SYSDATE
        WHERE
                HOME_SQUAD_NAME = HOME_SQ.NAME
            AND VISITORS_SQUAD_NAME = AWAY_SQ.NAME
            AND MATCH_ID = N3;

--Se ci sono giocatori non titolari infortunati, sottraggo uno
   dalle partite di stop.
        UPDATE STOPS
        SET
            DAYS = DAYS - 1
        WHERE
            P_CARD_CODE IN (
                SELECT
                    PLAYER_CARD_CODE
                FROM
                    IS_PART_OF
                WHERE
                    SQUAD_NAME = HOME_SQ.NAME
                    OR
                    SQUAD_NAME = AWAY_SQ.NAME
            );

--Se un giocatore non ha piu' giornate di stop, elimino la
   tupla.
        DELETE
        FROM    STOPS
        WHERE   DAYS = 0;

--Confermo
        COMMIT;
    END IF;

EXCEPTION
    WHEN NO_SQUAD THEN
        RAISE_APPLICATION_ERROR(-20001, 'Squadra non trovata!')
           ;
    WHEN PLAYER_MIN THEN
        RAISE_APPLICATION_ERROR(-20006, 'Non ci sono abbastanza
            giocatori!');
    WHEN MATCH_EXISTING THEN
        RAISE_APPLICATION_ERROR(-20005, 'La squadra '
                                          || S_NAME
                                          || ' del club '
                                          || C_NAME
                                          || ' e'' in attesa di
                                              disputare una
                                             partita!');
    WHEN CLUB_EXISTING THEN
        RAISE_APPLICATION_ERROR(-20005, 'Il club '
                                          || C_NAME
                                          || ' e'' in attesa di
                                              disputare una
                                             partita!');
END INSERT_MATCH;
/
```

11.1.10 Add Training

```
-- Tabelle interessate: 4
--  -> CLUB_CARD, IS_PART_OF, ACTIVE_DATA_PLAYER, SQUAD;

-- Funzioni interessate: 2
--  -> IS_ADMIN, CARD_IN_CLUB;

-- INPUT:
--      -> c_code:  codice carta di tipo <consumable> di
   categoria <training> o <GKTraining>;
--      -> p_code:  codice carta di tipo <player> alla quale
   applicare il <consumable>;
--  -> s_name:  nome della squadra dove gioca il player;
--      -> cc_name: se il chiamante e' amministratore: contiene
    il nome del club su cui effettuare le operazioni;
--              se il chiamante non e' amministratore: contiene
    o NULL, oppure il nome del club dell'utente chiamante.
-- OUTPUT:
--  -> Rende attivo il consumabile <c_code> sul player <p_code
   >.

CREATE OR REPLACE PROCEDURE ADD_TRAINING (
C_CODE   CLUB_CARD.CARD_CODE%TYPE,
P_CODE   CLUB_CARD.CARD_CODE%TYPE,
S_NAME   SQUAD.NAME%TYPE,
CC_NAME  SQUAD.SQUAD_CLUB_NAME%TYPE
) IS

N1            NUMBER(2, 0); --Controlla che il club possieda il
    consumabile <c_code> e il player <p_code>.
C_NAME        CLUB.CLUB_NAME%TYPE; --Nome effettivo del club (
   is_admin).
C_CONSUMABLE  CONSUMABLE.CATEGORY%TYPE; --Categoria del
   consumabile 'Training' o 'GKTraining'.
P_POSITION    PLAYER.POSITION%TYPE; --Posizione del player.
FLAG          NUMBER(1, 0) := 0; --Se e' uguale ad 1, tutti i
   controlli sono superati.

--Due casi:
-- 1. Si cerca di utilizzare un consumabile di tipo 'Training'
   su un player in posizione 'GK' cioe' portiere.
-- 2. Si cerca di utilizzare un consumabile di tipo 'GKTraining
   ' su un player non 'GK' portiere.
TRAINING_ERROR EXCEPTION;
NO_PLAYER_FOUND EXCEPTION;-- il player <p_code> non e' stato
   trovato.

BEGIN
--La funzione is_admin, ritorna il nome del club in base all'
   utente che chiama la funzione ADD_TRAINING;
--Se e' l'amministratore a chiamare la funzione il nome del
   club <c_name> sara' uguale a <cc_name>
--Altrimenti il nome del club <c_name> sara' quello dell'utente
   .
--In breve l'amministratore puo' eseguire la funzione su
   qualsiasi club, l'utente solo sul suo club.
```

```
SELECT
    IS_ADMIN(CC_NAME, USER)
INTO C_NAME
FROM
    DUAL;

--La funzione card_in_club, controlla che il club identificato
   dal nome <c_name> possieda la carta identificata dal <c_code
   >
-- se n1 != 0 il club possiede la carta, altrimenti no.
SELECT
    CARD_IN_CLUB(C_CODE, C_NAME)
INTO N1
FROM
    DUAL;

--Se <n1 = 0>, il club non possiede il consumabile <c_code>,
   oppure e' stato inserito un <c_code> errato.
IF N1 = 0 THEN
    RAISE NO_DATA_FOUND;
END IF;

--Controllo se il club possiede il player <p_code>.
SELECT
    CARD_IN_CLUB(P_CODE, C_NAME)
INTO N1
FROM
    DUAL;

--Se <n1 = 0>, il club non possiede il player <p_code>, oppure
   e' stato inserito un <p_code> errato.
IF N1 = 0 THEN
    RAISE NO_PLAYER_FOUND;
END IF;

--Seleziono il tipo del consumabile <c_code> in <c_consumable>.
SELECT
    CATEGORY
INTO C_CONSUMABLE
FROM
            CONSUMABLE C
    JOIN CLUB_CARD CC ON C.CARD_ID = CC.CONSUMABLE_ID
                            AND CC.CARD_CODE = C_CODE;

--Seleziono la posizione del player <p_code>.
SELECT
    PLAYER_POSITION
INTO P_POSITION
FROM
    IS_PART_OF
WHERE
    PLAYER_CARD_CODE = P_CODE;

--Se il consumabile <c_code> e di tipo <training> e la
   posizione del player <p_code> e diversa da 'GK' portiere.
IF
    C_CONSUMABLE = 'Training'
```

```
    AND P_POSITION <> 'GK'
THEN
    FLAG := 1; --Posso procedere ad applicare il consumabile <
       c_code>.
--Se il consumabile <c_code> e di tipo <GKtraining> e la
   posizione del player <p_code> e uguale a 'GK' portiere.
ELSIF
    C_CONSUMABLE = 'GKTraining'
    AND P_POSITION = 'GK'
THEN
    FLAG := 1; --Posso procedere ad applicare il consumabile <
       c_code>.
--Altrimenti non posso procedere poiche' il tipo di consumabile
    non corrisponde alla posizione del player.
ELSE
    FLAG := 0;
    RAISE TRAINING_ERROR;
END IF;

--Se <flag = 1>, posso procedere ad applicare il consumabile <
   c_code>.
IF FLAG = 1 THEN

--Aggiorno il campo <active_training> del player <p_code>.
    UPDATE ACTIVE_DATA_PLAYER
    SET
        ACTIVE_TRAINING = C_CODE
    WHERE
        P_CARD_CODE = P_CODE;

--Aggiungo un bonus di 2 dato dal training, alla valutazione <
   player_rating> del player.
    UPDATE IS_PART_OF
    SET
        PLAYER_RATING = PLAYER_RATING + 2
    WHERE
        PLAYER_CARD_CODE = P_CODE;

--Aggiorno la valutazione della squadra, poiche' e appena
   variata la valutazione di un player.
    UPDATE SQUAD
    SET
        SQUAD_RATING = (
            SELECT
                GET_SQUAD_RATING(S_NAME, C_NAME)
            FROM
                DUAL
        )
    WHERE
            NAME = S_NAME
        AND SQUAD_CLUB_NAME = C_NAME;

--Elimino il consumabile <c_code> utilizzato.
    DELETE FROM CLUB_CARD
    WHERE
        CARD_CODE = C_CODE;
```

```
--Confermo
    COMMIT;
END IF;

EXCEPTION
WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR('-20001', 'Il training con codice
       carta ('
                                        || C_CODE
                                        || ') non e'' stato
                                           trovato!');
WHEN NO_PLAYER_FOUND THEN
    RAISE_APPLICATION_ERROR('-20001', 'Il giocatore con codice
       carta ('
                                        || P_CODE
                                        || ') non e'' stato
                                           trovato!');
WHEN TRAINING_ERROR THEN
    RAISE_APPLICATION_ERROR('-20003', 'Il consumabile e'' di
       tipo ('
                                        || C_CONSUMABLE
                                        || ') la posizione del
                                           player e'' ('
                                        || P_POSITION
                                        || ')!');
END ADD_TRAINING;
/
```

11.1.11 Add Positioning

```
-- Tabelle interessate: 3
--  -> CLUB_CARD, IS_PART_OF, CONSUMABLE;

-- Funzioni interessate: 2
--  -> IS_ADMIN, CARD_IN_CLUB;

-- INPUT:
--      -> c_code:  codice carta di tipo <consumable> di
   categoria <positioning>;
--      -> p_code:  codice carta di tipo <player>;
--      -> cc_name: se il chiamante e' amministratore: contiene
    il nome del club su cui effettuare le operazioni;
--              se il chiamante non e' amministratore: contiene
    o NULL, oppure il nome del club dell'utente chiamante.
-- OUTPUT:
--  -> Cambia la posizione <player_position> del <player> con
   quella indicata dal consumabile.
CREATE OR REPLACE PROCEDURE ADD_POSITIONING (
C_CODE   CLUB_CARD.CARD_CODE%TYPE,
P_CODE   CLUB_CARD.CARD_CODE%TYPE,
CC_NAME  CLUB.CLUB_NAME%TYPE
) IS

N1          NUMBER(2, 0);
C_T         CONSUMABLE.TYPE%TYPE; --Posizione indicata dal
   consumabile <c_code>.
P_POSITION  IS_PART_OF.PLAYER_POSITION%TYPE; --Posizione
   attuale del <player> <p_code>.
RESULT      NUMBER(1, 0); --Variabile di controllo, se e'
   uguale ad 0 la posizione del player e' stata cambiata.
C_NAME      CLUB.CLUB_NAME%TYPE; --Nome effettivo del club (
   is_admin).
P_NAME      PLAYER.PLAYER_NAME%TYPE; --Nome del player.

NO_PLAYER_FOUND EXCEPTION; --Il <player> con codice carta <
   p_code> non e' stato trovato.
POSITION_ERROR EXCEPTION; --Se la posizione del player non
   corrisponde a quella indicata dal consumabile.

BEGIN
RESULT := 1; -- se e' uguale a 0 la posizione del player e'
   stata cambiata

--La funzione is_admin, ritorna il nome del club in base all'
   utente che chiama la funzione ADD_POSITIONING;
--Se e' l'amministratore a chiamare la funzione il nome del
   club <c_name> sara' uguale a <cc_name>
--Altrimenti il nome del club <c_name> sara' quello dell'utente
   .
--In breve l'amministratore puo' eseguire la funzione su
   qualsiasi club, l'utente solo sul suo club.
SELECT
    IS_ADMIN(CC_NAME, USER)
INTO C_NAME
FROM
```

```
    DUAL;
--La funzione card_in_club, controlla che il club identificato
   dal nome <c_name> possieda la carta identificata dal <c_code
   >
-- se n1 != 0 il club possiede la carta, altrimenti no.
SELECT
    CARD_IN_CLUB(C_CODE, C_NAME)
INTO N1
FROM
    DUAL;

-- Se <n1 = 0>, il club non possiede il consumabile <c_code>,
   oppure e' stato inserito un <c_code> errato.
IF N1 = 0 THEN
    RAISE NO_DATA_FOUND;

-- Altrimenti vediamo di che tipo e' il consumabile <c_code>.
ELSE
    SELECT
        TYPE
    INTO C_T
    FROM CONSUMABLE
    WHERE CARD_ID IN (
            SELECT
                CONSUMABLE_ID
            FROM
                CLUB_CARD
            WHERE
                CARD_CODE = C_CODE
        );

--Verifico che il club possiede il <player> con <card_code> <
   p_code>.
    SELECT
        CARD_IN_CLUB(P_CODE, C_NAME)
    INTO N1
    FROM
        DUAL;

-- Se <n1 = 0>, il club non possiede il player <p_code>, oppure
    e' stato inserito un <p_code> errato.
    IF N1 = 0 THEN
        RAISE NO_PLAYER_FOUND;
    END IF;

--Seleziono la posizione attuale del <player>.
    SELECT
        PLAYER_POSITION
    INTO P_POSITION
    FROM
        IS_PART_OF
    WHERE
        PLAYER_CARD_CODE = P_CODE;

-- In base al tipo di consumabile, se la posizione del player
   corrisponde, aggiorno la posizione a quella indicata dal
   consumabile.
```

```
-- Se l'operazione ha successo imposto <result> uguale a 0.
    IF C_T = 'LWBLB' THEN
        IF P_POSITION != 'LWB' THEN
            RAISE POSITION_ERROR;
        ELSE
            UPDATE IS_PART_OF
            SET
                PLAYER_POSITION = 'LB'
            WHERE
                PLAYER_CARD_CODE = P_CODE;

            RESULT := 0;
        END IF;
    ELSIF C_T = 'LBLWB' THEN
        IF P_POSITION != 'LB' THEN
            RAISE POSITION_ERROR;
        ELSE
            UPDATE IS_PART_OF
            SET
                PLAYER_POSITION = 'LWB'
            WHERE
                PLAYER_CARD_CODE = P_CODE;

            RESULT := 0;
        END IF;
    ELSIF C_T = 'RWBRB' THEN
        IF P_POSITION != 'RWB' THEN
            RAISE POSITION_ERROR;
        ELSE
            UPDATE IS_PART_OF
            SET
                PLAYER_POSITION = 'RB'
            WHERE
                PLAYER_CARD_CODE = P_CODE;

            RESULT := 0;
        END IF;
    ELSIF C_T = 'RBRWB' THEN
        IF P_POSITION != 'RB' THEN
            RAISE POSITION_ERROR;
        ELSE
            UPDATE IS_PART_OF
            SET
                PLAYER_POSITION = 'RWB'
            WHERE
                PLAYER_CARD_CODE = P_CODE;

            RESULT := 0;
        END IF;
    ELSIF C_T = 'LMLW' THEN
        IF P_POSITION != 'LM' THEN
            RAISE POSITION_ERROR;
        ELSE
            UPDATE IS_PART_OF
            SET
                PLAYER_POSITION = 'LW'
            WHERE
```

```
                PLAYER_CARD_CODE = P_CODE;

            RESULT := 0;
        END IF;
    ELSIF C_T = 'LWLM' THEN
        IF P_POSITION != 'LW' THEN
            RAISE POSITION_ERROR;
        ELSE
            UPDATE IS_PART_OF
            SET
                PLAYER_POSITION = 'LM'
            WHERE
                PLAYER_CARD_CODE = P_CODE;

            RESULT := 0;
        END IF;
    ELSIF C_T = 'RMRW' THEN
        IF P_POSITION != 'RM' THEN
            RAISE POSITION_ERROR;
        ELSE
            UPDATE IS_PART_OF
            SET
                PLAYER_POSITION = 'RW'
            WHERE
                PLAYER_CARD_CODE = P_CODE;

            RESULT := 0;
        END IF;
    ELSIF C_T = 'RWRM' THEN
        IF P_POSITION != 'RW' THEN
            RAISE POSITION_ERROR;
        ELSE
            UPDATE IS_PART_OF
            SET
                PLAYER_POSITION = 'RM'
            WHERE
                PLAYER_CARD_CODE = P_CODE;

            RESULT := 0;
        END IF;
    ELSIF C_T = 'LWLF' THEN
        IF P_POSITION != 'LW' THEN
            RAISE POSITION_ERROR;
        ELSE
            UPDATE IS_PART_OF
            SET
                PLAYER_POSITION = 'LF'
            WHERE
                PLAYER_CARD_CODE = P_CODE;

            RESULT := 0;
        END IF;
    ELSIF C_T = 'LFLW' THEN
        IF P_POSITION != 'LF' THEN
            RAISE POSITION_ERROR;
        ELSE
            UPDATE IS_PART_OF
```

```
            SET
                PLAYER_POSITION = 'LW'
            WHERE
                PLAYER_CARD_CODE = P_CODE;

            RESULT := 0;
        END IF;
    ELSIF C_T = 'RWRF' THEN
        IF P_POSITION != 'RW' THEN
            RAISE POSITION_ERROR;
        ELSE
            UPDATE IS_PART_OF
            SET
                PLAYER_POSITION = 'RF'
            WHERE
                PLAYER_CARD_CODE = P_CODE;

            RESULT := 0;
        END IF;
    ELSIF C_T = 'RFRW' THEN
        IF P_POSITION != 'RF' THEN
            RAISE POSITION_ERROR;
        ELSE
            UPDATE IS_PART_OF
            SET
                PLAYER_POSITION = 'RW'
            WHERE
                PLAYER_CARD_CODE = P_CODE;

            RESULT := 0;
        END IF;
    ELSIF C_T = 'CMCAM' THEN
        IF P_POSITION != 'CM' THEN
            RAISE POSITION_ERROR;
        ELSE
            UPDATE IS_PART_OF
            SET
                PLAYER_POSITION = 'CAM'
            WHERE
                PLAYER_CARD_CODE = P_CODE;

            RESULT := 0;
        END IF;
    ELSIF C_T = 'CAMCM' THEN
        IF P_POSITION != 'CAM' THEN
            RAISE POSITION_ERROR;
        ELSE
            UPDATE IS_PART_OF
            SET
                PLAYER_POSITION = 'CM'
            WHERE
                PLAYER_CARD_CODE = P_CODE;

            RESULT := 0;
        END IF;
    ELSIF C_T = 'CDMCM' THEN
        IF P_POSITION != 'CDM' THEN
```

```
            RAISE POSITION_ERROR;
        ELSE
            UPDATE IS_PART_OF
            SET
                PLAYER_POSITION = 'CM'
            WHERE
                PLAYER_CARD_CODE = P_CODE;

            RESULT := 0;
        END IF;
    ELSIF C_T = 'CMCDM' THEN
        IF P_POSITION != 'CM' THEN
            RAISE POSITION_ERROR;
        ELSE
            UPDATE IS_PART_OF
            SET
                PLAYER_POSITION = 'CDM'
            WHERE
                PLAYER_CARD_CODE = P_CODE;

            RESULT := 0;
        END IF;
    ELSIF C_T = 'CAMCF' THEN
        IF P_POSITION != 'CAM' THEN
            RAISE POSITION_ERROR;
        ELSE
            UPDATE IS_PART_OF
            SET
                PLAYER_POSITION = 'CF'
            WHERE
                PLAYER_CARD_CODE = P_CODE;

            RESULT := 0;
        END IF;
    ELSIF C_T = 'CFCAM' THEN
        IF P_POSITION != 'CF' THEN
            RAISE POSITION_ERROR;
        ELSE
            UPDATE IS_PART_OF
            SET
                PLAYER_POSITION = 'CAM'
            WHERE
                PLAYER_CARD_CODE = P_CODE;

            RESULT := 0;
        END IF;
    ELSIF C_T = 'CFST' THEN
        IF P_POSITION != 'CF' THEN
            RAISE POSITION_ERROR;
        ELSE
            UPDATE IS_PART_OF
            SET
                PLAYER_POSITION = 'ST'
            WHERE
                PLAYER_CARD_CODE = P_CODE;

            RESULT := 0;
```

```
        END IF;
    ELSIF C_T = 'STCF' THEN
        IF P_POSITION != 'ST' THEN
            RAISE POSITION_ERROR;
        ELSE
            UPDATE IS_PART_OF
            SET
                PLAYER_POSITION = 'CF'
            WHERE
                PLAYER_CARD_CODE = P_CODE;

            RESULT := 0;
        END IF;
    END IF;

--Se <result = 0>, il consumabile <c_code> e' stato utilizzato
   e va eliminato.
    IF RESULT = 0 THEN
        DELETE FROM CLUB_CARD
        WHERE CARD_CODE = C_CODE;
        COMMIT;
    END IF;
END IF;

EXCEPTION
WHEN POSITION_ERROR THEN
-- Seleziona il nome del player in <p_name>.
    SELECT
        PLAYER_NAME
    INTO P_NAME
    FROM PLAYER
    WHERE CARD_ID IN (
            SELECT
                PLAYER_ID
            FROM
                CLUB_CARD
            WHERE
                CARD_CODE = P_CODE
        );
    RAISE_APPLICATION_ERROR('-20002', 'Posizione non valida per
        '
                                        || P_NAME
                                        || ' ('
                                        || P_CODE
                                        || ')!');
WHEN NO_PLAYER_FOUND THEN
    RAISE_APPLICATION_ERROR('-20001', 'Il giocatore con codice
       carta ('
                                        || P_CODE
                                        || ') non e'' stato
                                           trovato!');
END ADD_POSITIONING;
/
```

11.1.12 Add Healing

```
-- Tabelle interessate: 3
--  -> CLUB_CARD, CONSUMABLE, STOPS;

-- Funzioni interessate: 3
--  -> IS_ADMIN, CARD_IN_CLUB;

-- INPUT:
--      -> c_code: codice carta di tipo <consumable> di
   categoria <healing>;
--      -> p_code: codice carta di tipo <player>;
--      -> cc_name: se il chiamante e' amministratore: contiene
    il nome del club su cui effettuare le operazioni;
--              se il chiamante non e' amministratore: contiene
    o NULL, oppure il nome del club dell'utente chiamante.
-- OUTPUT:
--  -> Applica il consumabile al player.
CREATE OR REPLACE PROCEDURE ADD_HEALING (
C_CODE   CLUB_CARD.CARD_CODE%TYPE,
P_CODE   CLUB_CARD.CARD_CODE%TYPE,
CC_NAME  CLUB.CLUB_NAME%TYPE
) IS

N1      NUMBER(2, 0); --Controlla che il club possieda le carte
   .
B_PART  STOPS.BODY_PART%TYPE; --Parte del corpo dove il player
   e' infortunato.
C_TYPE  CONSUMABLE.TYPE%TYPE; --Parte del corpo indicata dal
   consumabile.
C_NAME  CLUB.CLUB_NAME%TYPE; --Nome del club effettivo (
   is_admin).
RES     NUMBER(1,0); --Giorni indicati dal consumabile da
   sottrarre allo stop.


NO_PLAYER_FOUND EXCEPTION; --Il player <p_code> non e' stato
   trovato.
NO_STOPS_FOUND EXCEPTION; --Il player <p_code> non e'
   infortunato.
CONSUMABLE_WRONG_TYPE EXCEPTION; --Nel caso in cui la parte del
    corpo che si vuole curare non corrisponde a quella indicata
    dal consumabile.

BEGIN
--La funzione is_admin, ritorna il nome del club in base all'
   utente che chiama la funzione ADD_HEALING;
--Se e' l'amministratore a chiamare la funzione il nome del
   club <c_name> sara' uguale a <cc_name>
--Altrimenti il nome del club <c_name> sara' quello dell'utente
   .
--In breve l'amministratore puo' eseguire la funzione su
   qualsiasi club, l'utente solo sul suo club.
SELECT
    IS_ADMIN(CC_NAME, USER)
INTO C_NAME
FROM
```

```
    DUAL;

--La funzione card_in_club, controlla che il club identificato
   dal nome <c_name> possieda la carta identificata dal <c_code
   >
-- se n1 != 0 il club possiede la carta, altrimenti no.
SELECT
    CARD_IN_CLUB(C_CODE, C_NAME)
INTO N1
FROM
    DUAL;

--Se <n1 = 0> il club non possiede il consumabile, oppure il <
   c_code> e' errato.
IF N1 = 0 THEN
    RAISE NO_DATA_FOUND;
ELSE
--Controllo se il club possiede il <player>.
    SELECT
        CARD_IN_CLUB(P_CODE, C_NAME)
    INTO N1
    FROM
        DUAL;

--Se <n1 = 0> il club non possiede il player, oppure il <p_code
   > e' errato.
    IF N1 = 0 THEN
        RAISE NO_PLAYER_FOUND;
    END IF;

--Controllo se il player e' infortunato.
    SELECT
        COUNT(P_CARD_CODE)
    INTO N1
    FROM
        STOPS
    WHERE
        P_CARD_CODE = P_CODE;

--Se <n1 = 0> il player non risulta infortunato.
    IF N1 = 0 THEN
        RAISE NO_STOPS_FOUND;
    END IF;

--Seleziono la parte del corpo dov'e' infortunato il <player>
   in <b_part>;
    SELECT
        BODY_PART
    INTO B_PART
    FROM STOPS
    WHERE P_CARD_CODE = P_CODE;

--Seleziono la parte del corpo indicata dal consumabile <c_code
   > in <c_type>.
    SELECT
        TYPE
    INTO C_TYPE
```

```
    FROM CONSUMABLE C
        JOIN CLUB_CARD CC
            ON C.CARD_ID = CC.CONSUMABLE_ID
                AND CC.CARD_CODE = C_CODE;

--Se il consumabile non cura tutte le parti del corpo <All> e
   il <player> e' infortunato in una zona <b_part> diversa da
-- quella indicata dal consumabile <c_type>, si genera un
   eccezione.
--Esempio:  b_part = 'Head'; c_type = 'Foot';
--Sto cercando di curare un infortunio alla testa <b_part> con
   un consumabile che cura gli infortuni al piede <c_type>.
    IF C_TYPE <> 'All'
        AND C_TYPE <> B_PART
    THEN
        RAISE CONSUMABLE_WRONG_TYPE;
--Se il consumabile cura tutte le parti del corpo, oppure la
   parte del corpo dove il <player> e' infortunato <b_part>
-- corrisponde a quella indicata dal consumabile.
    ELSIF C_TYPE = 'All' OR C_TYPE = B_PART THEN

--Seleziono il quantitativo di giorni da sottrarre allo stop.
        SELECT AMOUNT
        INTO RES
        FROM CONSUMABLE
        WHERE CARD_ID IN (SELECT CONSUMABLE_ID
                          FROM CLUB_CARD
                          WHERE CARD_CODE = C_CODE);

        UPDATE STOPS
        SET DAYS = DAYS - RES
        WHERE P_CARD_CODE = P_CODE AND DAYS > 1;

--Elimino i player guariti.
        DELETE FROM STOPS
        WHERE DAYS = 0;

--Cancello il consumabile <c_code> appena utilizzato.
        DELETE FROM CLUB_CARD
        WHERE CARD_CODE = C_CODE;
        COMMIT;
    END IF;
END IF;

EXCEPTION
WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR('-20001', 'Il consumabile con
       codice carta ('
                                        || C_CODE
                                        || ') non e'' stato
                                           trovato!');
WHEN NO_PLAYER_FOUND THEN
    RAISE_APPLICATION_ERROR('-20001', 'Il player con codice
       carta ('
                                        || P_CODE
                                        || ') non e'' stato
                                           trovato!');
```

```
WHEN NO_STOPS_FOUND THEN
    RAISE_APPLICATION_ERROR('-20001', 'Il player con codice
       carta ('
                                        || C_CODE
                                        || ') non e''
                                           infortunato!');
WHEN CONSUMABLE_WRONG_TYPE THEN
    RAISE_APPLICATION_ERROR('-20002', 'Il consumabile di tipo (
       '
                                        || C_TYPE
                                        || ') non puo'' essere
                                           usato per un
                                           infortunio di tipo (
                                           '
                                        || B_PART || ')');
END ADD_HEALING;
/
```

11.1.13 Add Contract

```
-- Tabelle interessate: 2
--  -> CLUB_CARD, ACTIVE_DATA_PLAYER;

-- Funzioni interessate: 3
--  -> IS_ADMIN, CARD_IN_CLUB, GET_CONTRACT;

-- INPUT:
--      -> c_code: codice carta di tipo <consumable> di
   categoria <contract>;
--      -> p_card_code: codice carta di tipo <player>;
--      -> cc_name: se il chiamante e' amministratore: contiene
    il nome del club su cui effettuare le operazioni;
--              se il chiamante non e' amministratore: contiene
    o NULL, oppure il nome del club dell'utente chiamante.
-- OUTPUT:
--  -> Applica il consumabile al player.

CREATE OR REPLACE PROCEDURE ADD_CONTRACT (
C_CODE       CLUB_CARD.CARD_CODE%TYPE,
P_CARD_CODE  CLUB_CARD.CARD_CODE%TYPE,
CC_NAME      CLUB.CLUB_NAME%TYPE
) IS

S_NAME  SQUAD.NAME%TYPE; --Nome delle squadra dov'e' schierato
   il player <p_card_code>.
C_NAME  SQUAD.SQUAD_CLUB_NAME%TYPE; --Nome del club che
   possiede la squadra <s_name>.
RESULT  NUMBER(2, 0); --Conterra' il nuovo valore di <contracts
   > dopo aver applicato il consumabile.
N1      NUMBER(2, 0);

BEGIN
--La funzione is_admin, ritorna il nome del club in base all'
   utente che chiama la funzione ADD_CONTRACT;
--Se e' l'amministratore a chiamare la funzione il nome del
   club <c_name> sara' uguale a <cc_name>;
--Altrimenti il nome del club <c_name> sara' quello dell'utente
   .
--In breve, l'amministratore puo' eseguire la funzione su
   qualsiasi club, l'utente solo sul proprio club.
SELECT
    IS_ADMIN(CC_NAME, USER)
INTO C_NAME
FROM
    DUAL;

--La funzione card_in_club, controlla che il club identificato
   dal nome <c_name> possieda la carta identificata dal <c_code
   >
-- se n1 != 0 il club possiede la carta, altrimenti no.
SELECT
    CARD_IN_CLUB(C_CODE, C_NAME)
INTO N1
FROM
    DUAL;
```

```
-- Se n1 e' ancora uguale a 0 il club non possiede la carta <
   c_code>, o e' stato inserito un <c_code> errato.
IF N1 = 0 THEN
    RAISE NO_DATA_FOUND;
ELSE
-- Altrimenti il club possiede la carta; Calcolo il nuovo
   valore di <contracts> con la funzione GET_CONTRACT
-- la quale prende in input il codice carta <c_code> del
   consumabile, il codice carta <p_card_code> del player
-- e il tipo del consumabile, quest'ultimo puo' essere 'Player'
    o 'Manager'.
-- La funzione restituisce in <result> il nuovo valore di <
   contracts> calcolato dopo aver applicato il consumabile
-- e considerando che il player/manager non puo' superare i 99
   contratti.
    SELECT
        GET_CONTRACT(C_CODE, P_CARD_CODE, 'Player')
    INTO RESULT
    FROM
        DUAL;

-- Aggiorno i contratti <contracts> del player.
    UPDATE ACTIVE_DATA_PLAYER
    SET
        CONTRACTS = RESULT
    WHERE
        P_CARD_CODE = P_CARD_CODE;

-- Cancello il consumabile <c_code> appena utilizzato.
    DELETE FROM CLUB_CARD
    WHERE
        CARD_CODE = C_CODE;

--Confermo
    COMMIT;
END IF;

EXCEPTION
WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR('-20001', 'La carta con card_code (
       '
                                        || C_CODE
                                        || ') non e''  stata
                                           trovata!');
END ADD_CONTRACT;
/
```

11.1.14 Add Fitness

```
-- Tabelle interessate: 3
--  -> CLUB_CARD, CONSUMABLE, ACTIVE_DATA_PLAYER;

-- Funzioni interessate: 2
--  -> IS_ADMIN, CARD_IN_CLUB;

-- INPUT:
--  -> s_name: nome della squadra;
--      -> cc_name: se il chiamante e' amministratore: contiene
    il nome del club su cui effettuare le operazioni;
--              se il chiamante non e' amministratore: contiene
    o NULL, oppure il nome del club dell'utente chiamante.
--      -> c_code: codice carta di tipo <club_item>;
--      -> p_code:  codice carta fi tipo player, oppure <NULL>
   se si vuole applicare il consumabile alla squadra;
-- OUTPUT:
--  -> Rende attivo il <consumable> di tipo <fitness> sul
   player <p_car_code>, o sulla squadra <s_name>.
CREATE OR REPLACE PROCEDURE ADD_FITNESS (
S_NAME   SQUAD.NAME%TYPE,
CC_NAME  CLUB.CLUB_NAME%TYPE,
C_CODE   CLUB_CARD.CARD_CODE%TYPE,
P_CODE   CLUB_CARD.CARD_CODE%TYPE
) IS

N1             NUMBER(2, 0); --Controlla che il club possieda
   la carta <c_code> e <P_CODE>.
RESULT         NUMBER(2, 0);  --Di quanto dev'essere
   incrementata la forma fisica <player_fitness>.
TOTAL_FITNESS  NUMBER(3, 0); --Valore totale della forma fisica
    <player_fitness>.
C_NAME         CLUB.CLUB_NAME%TYPE; --Nome effettivo del club (
   is_admin).

TYPE ARRAY_CC_T IS VARRAY(18) OF VARCHAR(8);
SQUAD_C_CODE   ARRAY_CC_T; --Array per i codici carta dei
   player della squdra <s_name>.
BEGIN
--La funzione is_admin, ritorna il nome del club in base all'
   utente che chiama la funzione ADD_FITNESS;
--Se e' l'amministratore a chiamare la funzione il nome del
   club <c_name> sara' uguale a <cc_name>
--Altrimenti il nome del club <c_name> sara' quello dell'utente
   .
--In breve l'amministratore puo' eseguire la funzione su
   qualsiasi club, l'utente solo sul suo club.
SELECT
    IS_ADMIN(CC_NAME, USER)
INTO C_NAME
FROM
    DUAL;

--La funzione card_in_club, controlla che il club identificato
   dal nome <c_name> possieda la carta identificata dal <c_code
   >
```

```
-- se n1 != 0 il club possiede la carta, altrimenti no.
SELECT
    CARD_IN_CLUB(C_CODE, C_NAME)
INTO N1
FROM
    DUAL;

-- Se n1 e' ancora uguale a 0, il club non possiede la carta
   oppure e' stato inserito un <c_code> errato!
IF N1 = 0 THEN
    RAISE NO_DATA_FOUND;
ELSE
--Seleziono di quanto deve essere aumentata la forma fisica <
   player_fitness> del player in result.
    SELECT
        AMOUNT
    INTO RESULT
    FROM
        CONSUMABLE
    WHERE
        CARD_ID IN (
            SELECT
                CONSUMABLE_ID
            FROM
                CLUB_CARD
            WHERE
                CARD_CODE = C_CODE
        );
--Se il codice carta <P_CODE> non e' NULL il consumabile va
   applicato al player.
    IF P_CODE IS NOT NULL THEN

--Seleziono la forma fisica <player_fitness> attuale in <
   total_fitness>.
        SELECT
            PLAYER_FITNESS
        INTO TOTAL_FITNESS
        FROM
            ACTIVE_DATA_PLAYER
        WHERE
            P_CARD_CODE = P_CODE;

        TOTAL_FITNESS := TOTAL_FITNESS + RESULT;
        IF TOTAL_FITNESS > 99 THEN
            TOTAL_FITNESS := 99;
        END IF;

-- Aggiorno il valore di <player_fitness> sommato a <result>.
        UPDATE ACTIVE_DATA_PLAYER
        SET
            PLAYER_FITNESS = TOTAL_FITNESS
        WHERE
            P_CARD_CODE = P_CODE;

--Cancello il consumabile <c_code> appena utilizzato.
        DELETE FROM CLUB_CARD
        WHERE CARD_CODE = C_CODE;
```

```
        COMMIT;
-- <P_CODE> e' NULL, quindi il consumabile va applicato a tutta
    la squadra.
    ELSE
--Aggiorno la forma fisica <player_fitness> di tutti i player
   della squadra <s_name>.
--Seleziono i codici carta dei player della squadra per
   controllare che la loro forma fisica non superi il massimo
   (99).
        SELECT
            PLAYER_CARD_CODE
        BULK COLLECT
        INTO SQUAD_C_CODE
        FROM
            IS_PART_OF
        WHERE
                SQUAD_NAME = S_NAME;

        FOR I IN 1..SQUAD_C_CODE.COUNT LOOP
            SELECT
                PLAYER_FITNESS
            INTO TOTAL_FITNESS
            FROM
                ACTIVE_DATA_PLAYER
            WHERE
                P_CARD_CODE = SQUAD_C_CODE(I);

            TOTAL_FITNESS := TOTAL_FITNESS + RESULT;
            IF TOTAL_FITNESS > 99 THEN
                TOTAL_FITNESS := 99;
            END IF;
            UPDATE ACTIVE_DATA_PLAYER
            SET
                PLAYER_FITNESS = TOTAL_FITNESS
            WHERE
                P_CARD_CODE = SQUAD_C_CODE(I);

        END LOOP;

-- Cacello il consumabile <c_code> appena utilizzato.
        DELETE FROM CLUB_CARD
        WHERE CARD_CODE = C_CODE;

        COMMIT;
    END IF;
END IF;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RAISE_APPLICATION_ERROR('-20007', 'Il consumabile ('
                                          || C_CODE
                                          || ') non e'' stato
                                              trovato!');
END ADD_FITNESS;
```

11.1.15 Add Manager League

```
-- Tabelle interessate: 3
--  -> CLUB_CARD, ACTIVE_DATA_MANAGER, CONSUMABLE;

-- Funzioni interessate: 2
--  -> IS_ADMIN, CARD_ID_CLUB;

-- INPUT:
--  -> c_code: codice carta di tipo <consumable> di categoria <
   manager league>.
--      -> m_code: codice carta di tipo manager.
--      -> cc_name: se il chiamante e' amministratore: contiene
    il nome del club su cui effettuare le operazioni;
--              se il chiamante non e' amministratore: contiene
    o NULL, oppure il nome del club dell'utente chiamante.
-- OUTPUT:
--  -> Cambia la lega del manager con quella specificata dal
   consumabile.
CREATE OR REPLACE PROCEDURE ADD_MANAGER_LEAGUE (
C_CODE   CLUB_CARD.CARD_CODE%TYPE,
M_CODE   CLUB_CARD.CARD_CODE%TYPE,
CC_NAME  CLUB.CLUB_NAME%TYPE
) IS

N1        NUMBER(2, 0);
M_LEAGUE  MANAGER.LEAGUE_NAME%TYPE; --La lega del manager
   indicata dal consumabile.
C_NAME    CLUB.CLUB_NAME%TYPE; --Il nome del club effettivo (
   is_admin).

NO_CONSUMABLE_FOUND EXCEPTION; --Se il consumabile <c_code> non
    e' stato trovato.
NO_MANAGER_FOUND EXCEPTION; --Se la squadra non ha manager <
   m_code> attivo.

BEGIN
--La funzione is_admin, ritorna il nome del club in base all'
   utente che chiama la funzione ADD_MANAGER_LEAGUE;
--Se e' l'amministratore a chiamare la funzione il nome del
   club <c_name> sara' uguale a <cc_name>
--Altrimenti il nome del club <c_name> sara' quello dell'utente
   .
--In breve l'amministratore puo' eseguire la funzione su
   qualsiasi club, l'utente solo sul suo club.
SELECT IS_ADMIN(CC_NAME, USER)
INTO C_NAME
FROM DUAL;

--La funzione card_in_club, controlla che il club identificato
   dal nome <c_name> possieda la carta identificata dal <c_code
   >
-- se n1 != 0 il club possiede la carta, altrimenti no.
SELECT CARD_IN_CLUB(C_CODE, C_NAME)
INTO N1
FROM DUAL;
```

```
-- Se <n1 = 0>, il club non possiede il consumabile, oppure e'
   stato inserito un codice carta <c_code> errato.
IF N1 = 0 THEN
    RAISE NO_CONSUMABLE_FOUND;
END IF;
SELECT CARD_IN_CLUB(M_CODE, C_NAME)
INTO N1
FROM DUAL;

-- Se <n1 = 0>, il club non ha un manager attivo, oppure e'
   stato inserito un codice carta <m_code> errato.
IF N1 = 0 THEN
    RAISE NO_MANAGER_FOUND;
END IF;

--Seleziono la lega del consumabile <c_code> in <m_league>.
SELECT
    TYPE
INTO M_LEAGUE
FROM CONSUMABLE
WHERE CARD_ID IN (
        SELECT CONSUMABLE_ID
        FROM CLUB_CARD
        WHERE CARD_CODE = C_CODE
    );

-- Aggiorno il valore <manager_league> uguale alla nuova lega <
   m_league>.
UPDATE ACTIVE_DATA_MANAGER
SET MANAGER_LEAGUE = M_LEAGUE
WHERE M_CARD_CODE = M_CODE;

-- Cancello il consumabile <c_code> appena utilizzato.
DELETE FROM CLUB_CARD
WHERE CARD_CODE = C_CODE;
COMMIT;

EXCEPTION
WHEN NO_CONSUMABLE_FOUND THEN
    RAISE_APPLICATION_ERROR('-20001', 'Il consumabile ('
                                        || C_CODE
                                        || ') non e'' stato
                                           trovato!');
WHEN NO_MANAGER_FOUND THEN
    RAISE_APPLICATION_ERROR('-20001', 'Il manager con codice
       carta ('
                                        || M_CODE
                                        || ') non e'' stato
                                           trovato');
END ADD_MANAGER_LEAGUE;
```

11.1.16 Add Club Item

```
-- Tabelle interessate: 2
--  -> CLUB_CARD, CLUB;

-- Funzioni interessate: 2
--  -> IS_ADMIN, CARD_IN_CLUB;

-- INPUT:
--      -> c_code:  codice carta di tipo <club_item>;
--      -> cc_name: se il chiamante e' amministratore: contiene
    il nome del club su cui effettuare le operazioni;
--              se il chiamante non e' amministratore: contiene
    o NULL, oppure il nome del club dell'utente chiamante.
-- OUTPUT:
--  -> Rende attivo il <club_item>.
CREATE OR REPLACE PROCEDURE ADD_CLUB_ITEM (
C_CODE   CLUB_CARD.CARD_CODE%TYPE,
CC_NAME  CLUB.CLUB_NAME%TYPE
) IS

N1      NUMBER(1, 0);
N2      NUMBER(1, 0); --Controllo su <club_item> di tipo 'Kit'
   (Firts).
N3      NUMBER(1, 0); --Controllo su <club_item> di tipo 'Kit'
   (Second).
N4      NUMBER(1, 0); --Controllo su <club_item> di tipo 'Badge
   '.
N5      NUMBER(1, 0); --Controllo su <club_item> di tipo 'Ball
   '.
N6      NUMBER(1, 0); --Controllo su <club_item> di tipo '
   Stadium'.
A_CODE  CLUB_CARD.CARD_CODE%TYPE; --Codice carta del <club_item
   > che diventera' attivo.
C_NAME  CLUB.CLUB_NAME%TYPE; --Nome del club effettivo (
   is_admin).
C_ITEM  CLUB_ITEM.CATEGORY_ITEM%TYPE; --Categoria del club item
   .
SAME_ITEM EXCEPTION; --Nel caso in cui si prova ad attivare un
   <club_item> gia' attivo.
INVALID_CATEGORY EXCEPTION; --Nel caso in cui la categoria del
   <club_item> sia errata.

BEGIN
--La funzione is_admin, ritorna il nome del club in base all'
   utente che chiama la funzione ADD_CLUB_ITEM;
--Se e' l'amministratore a chiamare la funzione il nome del
   club <c_name> sara' uguale a <cc_name>;
--Altrimenti il nome del club <c_name> sara' quello dell'utente
   .
--In breve, l'amministratore puo' eseguire la funzione su
   qualsiasi club, l'utente solo sul proprio club.
SELECT IS_ADMIN(CC_NAME, USER)
INTO C_NAME
FROM DUAL;

--La funzione card_in_club, controlla che il club identificato
```

```
    dal nome <c_name> possiede la carta identificata dal <c_code
    >
-- se n1 != 0 il club possiede la carta, altrimenti no.
SELECT CARD_IN_CLUB(C_CODE, C_NAME)
INTO N1
FROM DUAL;

--Controlla che il <club_item> non sia gia' attivo.
SELECT
    COUNT(ACTIVE_FIRST_KIT_CODE),
    COUNT(ACTIVE_SECOND_KIT_CODE),
    COUNT(ACTIVE_BADGE_CODE),
    COUNT(ACTIVE_BALL_CODE),
    COUNT(ACTIVE_STADIUM_CODE)
INTO N2, N3, N4, N5, N6
FROM
    CLUB
WHERE
    ACTIVE_FIRST_KIT_CODE = C_CODE
    OR ACTIVE_SECOND_KIT_CODE = C_CODE
    OR ACTIVE_BADGE_CODE = C_CODE
    OR ACTIVE_BALL_CODE = C_CODE
    OR ACTIVE_STADIUM_CODE = C_CODE;

SELECT CATEGORY_ITEM
INTO C_ITEM
FROM CLUB_ITEM C
    JOIN CLUB_CARD CC
    ON C.CARD_ID = CC.CLUB_ITEM_ID
    AND CC.CARD_CODE = C_CODE;

--Se <n1 = 0> il club non possiede la carta, oppure e' stato
    inserito un <c_code> errato.
IF N1 = 0 THEN
    RAISE NO_DATA_FOUND;
ELSIF N2 != 0 OR N3 != 0 OR N4 != 0 OR N5 != 0 OR N6 != 0 THEN
    RAISE SAME_ITEM;
ELSE--Altrimenti controllo la categoria <c_item> del <club_item
    >.
    IF C_ITEM = 'Kit' THEN
--Controllo che non ci sia un <club_item> gia' attivo.
        SELECT
            COUNT(ACTIVE_FIRST_KIT_CODE),
            COUNT(ACTIVE_SECOND_KIT_CODE)
        INTO N1, N2
        FROM CLUB
        WHERE CLUB_NAME = C_NAME;

--Se il primo kit e' libero, inserisco <c_code> in <
    active_first_kit_code>.
        IF N1 = 0 THEN
            UPDATE CLUB
            SET
                ACTIVE_FIRST_KIT_CODE = C_CODE
            WHERE
                CLUB_NAME = C_NAME;
```

```
            COMMIT;
--Se il secondo kit e' libero, inserisco <c_code> in <
   active_second_kit_code>.
        ELSIF N2 = 0 THEN
            UPDATE CLUB
            SET
                ACTIVE_SECOND_KIT_CODE = C_CODE
            WHERE
                CLUB_NAME = C_NAME;

            COMMIT;
        END IF;

    ELSIF C_ITEM = 'Ball' THEN
        UPDATE CLUB
        SET
            ACTIVE_BALL_CODE = C_CODE
        WHERE
            CLUB_NAME = C_NAME;

        COMMIT;
    ELSIF C_ITEM = 'Badge' THEN
        UPDATE CLUB
        SET
            ACTIVE_BADGE_CODE = C_CODE
        WHERE
            CLUB_NAME = C_NAME;

        COMMIT;
    ELSIF C_ITEM = 'Stadium' THEN
        UPDATE CLUB
        SET
            ACTIVE_STADIUM_CODE = C_CODE
        WHERE
            CLUB_NAME = C_NAME;

        COMMIT;
    ELSE
        RAISE INVALID_CATEGORY;
    END IF;
END IF;

EXCEPTION
WHEN SAME_ITEM THEN
    RAISE_APPLICATION_ERROR('-20001', 'Il ['
                                        || C_ITEM
                                        || '] e'' gia'' attivo!
                                            ');
WHEN INVALID_CATEGORY THEN
    RAISE_APPLICATION_ERROR('-20002', 'La ['
                                        || C_ITEM
                                        || '] non e'' valida!')
                                            ;
WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR('-20003', 'La carta non e'' stata
       trovata!');
END ADD_CLUB_ITEM;
```

11.1.17 Add Manager

```
-- Tabelle interessate: 6
--  -> CLUB_CARD, MANAGER, MANGES, ACTIVE_DATA_MANAGER, PLAYER,
    IS_PART_OF;

-- Funzioni interessate: 2
--  -> IS_ADMIN, CARD_IN_CLUB;

-- INPUT:
--      -> m_card_code: codice carta di tipo manager;
--      -> s_name:  nome della squadra;
--      -> cc_name: se il chiamante e' amministratore: contiene
    il nome del club su cui effettuare le operazioni;
--              se il chiamante non e' amministratore: contiene
    o NULL, oppure il nome del club dell'utente chiamante.
-- OUTOUT:
--  -> Schiera il manager <m_card_code> nella squadra <s_name>.
CREATE OR REPLACE PROCEDURE ADD_MANAGER (
M_CARD_CODE  IN  CLUB_CARD.CARD_CODE%TYPE,
S_NAME       IN  SQUAD.NAME%TYPE,
CC_NAME      IN  SQUAD.NAME%TYPE
) IS
--Controlli
N1              NUMBER(2, 0); --Controlla che il club possiede
   le carte.
N2              NUMBER(2, 0); --Controlla che il manager non
   allena gia' una squadra.
FLAG            NUMBER(1, 0); --Variabile di controllo.
C_NAME          CLUB.CLUB_NAME%TYPE; --Nome del club effettivo (
   is_admin).

--Manager
A_M_CARD_CODE  CLUB_CARD.CARD_CODE%TYPE; --<card_code> del
   manager qualora c'e' gia un manager attivo nella squadra.
M_LEAGUE       MANAGER.LEAGUE_NAME%TYPE; --Lega del manager.
M_NATION       MANAGER.NATIONALITY%TYPE; --Nazionalita' del
   manager.
M_ID           MANAGER.CARD_ID%TYPE; --<card_id> del manager.

--Player
P_NATION       PLAYER.NATIONALITY%TYPE; --Nazionalita' dei
   player della squadra.
P_LEAGUE       PLAYER.LEAGUE_NAME%TYPE; --Lega preferita dei
   player della squadra.

TYPE ARRAY_CC_T IS VARRAY(18) OF VARCHAR2(8);
ARR_CARD_CODE  ARRAY_CC_T; --Array che conterra' i <card_code>
   dei player della squadra.

NO_MANAGER_FOUND EXCEPTION; --Il manager <m_card_code> non e'
   stato trovato.
EXISTING_MANAGER EXCEPTION; --La squadra <s_name> e' gia'
   allenata dal manager <m_card_code>.

BEGIN
FLAG := 0; --Se e' uguale ad 1, significa che c'e' un manager
```

```
   nella squadra <s_name> e va sostituito.

--La funzione is_admin, ritorna il nome del club in base all'
   utente che chiama la funzione ADD_MANAGER;
--Se e' l'amministratore a chiamare la funzione il nome del
   club <c_name> sara' uguale a <cc_name>
--Altrimenti il nome del club <c_name> sara' quello dell'utente
   .
--In breve l'amministratore puo' eseguire la funzione su
   qualsiasi club, l'utente solo sul suo club.
SELECT
    IS_ADMIN(CC_NAME, USER)
INTO C_NAME
FROM
    DUAL;

--La funzione card_in_club, controlla che il club identificato
   dal nome <c_name> possieda la carta identificata dal <c_code
   >
-- se n1 != 0 il club possiede la carta, altrimenti no.
SELECT
    CARD_IN_CLUB(M_CARD_CODE, C_NAME)
INTO N1
FROM
    DUAL;

IF N1 = 0 THEN
    RAISE NO_MANAGER_FOUND;
END IF;

--Controllo che la squadra <s_name> non abbia gia' un'
   allenatore.
SELECT
    COUNT(MANAGER_CARD_CODE)
INTO N2
FROM
    MANAGES
WHERE
        SQUAD_NAME = S_NAME;

--Se <n2 > 0> la squadra <s_name> ha gia' un allenatore
-- controllo se e' uguale al manager che voglio schierare.
IF N2 > 0 THEN

--Seleziono il codice carta <card_code> del manager della
   squadra <s_name>.
    SELECT
        MANAGER_CARD_CODE
    INTO A_M_CARD_CODE
    FROM
        MANAGES
    WHERE
            SQUAD_NAME = S_NAME;

--Se e' uguale al codice carta <m_card_code> del manager che
   voglio aggiungere, genero un eccezione.
    IF A_M_CARD_CODE = M_CARD_CODE THEN
```

```
        RAISE EXISTING_MANAGER;
--Altrimenti pongo <flag = 1> ovvero c'e' un manager che va
   sostituito.
    ELSIF A_M_CARD_CODE IS NOT NULL THEN
        FLAG := 1;
    END IF;

END IF;

--Seleziono il <card_id> del manager, per poi selezionare
   nazionalita e lega.
SELECT
    MANAGER_ID
INTO M_ID
FROM
    CLUB_CARD
WHERE
    CARD_CODE = M_CARD_CODE;

--Seleziono nazionalita e lega del manager.
SELECT
    NATIONALITY,
    LEAGUE_NAME
INTO
    M_NATION,
    M_LEAGUE
FROM
            MANAGER M
    JOIN CLUB_CARD C ON M.CARD_ID = C.MANAGER_ID
                        AND C.CARD_CODE = M_CARD_CODE;

--Se <flag = 1>, c'e' un manager da sostituire.
IF FLAG = 1 THEN
--Aggiorno il <manager_card_code> con quello del nuovo manager.
    UPDATE MANAGES
    SET
        MANAGER_CARD_CODE = M_CARD_CODE
    WHERE
            SQUAD_NAME = S_NAME;

ELSE--Se la squadra <s_name> non ha un'allenatore.

--Inserisco la tupla in manages
    INSERT INTO MANAGES VALUES (
        M_CARD_CODE,
        C_NAME,
        S_NAME
    );

END IF;

--Se il manager aveva gia' dati attivi, <n1 > 0> non devo
   inserirli nuovamente.
SELECT
    COUNT(*)
INTO N1
FROM
```

```
    ACTIVE_DATA_MANAGER
WHERE
    M_CARD_CODE = M_CARD_CODE;

--Se il manager non aveva dati attivi, inserisco la nuova tupla
   .
IF N1 = 0 THEN
    INSERT INTO ACTIVE_DATA_MANAGER VALUES (
        M_CARD_CODE,
        M_LEAGUE,
        7
    );

END IF;

--Confermo
COMMIT;

--Seleziono i codici carta <card_code> dei <player> della
   squadra <s_name>.
SELECT
    PLAYER_CARD_CODE
BULK COLLECT
INTO ARR_CARD_CODE
FROM
    IS_PART_OF
WHERE
        SQUAD_NAME = S_NAME;

FOR I IN 1..ARR_CARD_CODE.COUNT LOOP
--Seleziono nazionalita' e lega del <player> i-esimo.
    SELECT
        NATIONALITY,
        LEAGUE_NAME
    INTO
        P_NATION,
        P_LEAGUE
    FROM
                PLAYER P
        JOIN CLUB_CARD C ON P.CARD_ID = C.PLAYER_ID
                            AND C.CARD_CODE = ARR_CARD_CODE(I);

--Se la nazionalita' o la lega del player corrispondono a
   quella del manager
-- il player guadagna un bonus di 1 sull'intesa <
   player_chemistry>.
    IF P_NATION = M_NATION OR P_LEAGUE = M_LEAGUE THEN

--Aggiorno l'intesa <player_chemistry> del player i-esimo.
        UPDATE
        IS_PART_OF
        SET
            PLAYER_CHEMISTRY = PLAYER_CHEMISTRY + 1
        WHERE
                PLAYER_CARD_CODE = ARR_CARD_CODE(I)
            AND PLAYER_CHEMISTRY < 10;
```

```
--Confermo
        COMMIT;
    END IF;

END LOOP;

EXCEPTION
WHEN NO_MANAGER_FOUND THEN
    RAISE_APPLICATION_ERROR('-20007', 'Il manager con codice
       carta ('
                                        || M_CARD_CODE
                                        || ') non e'' stato
                                           trovato!');
WHEN EXISTING_MANAGER THEN
    RAISE_APPLICATION_ERROR('-20007', 'Il manager con card_code
        ('
                                        || M_CARD_CODE
                                        || ') e'' gia presente!
                                           ');
END ADD_MANAGER;
/
```

11.1.18 Remove Manager

```
-- Tabelle interessate: 2
--  -> CLUB_CARD, MANAGES, MANAGER, IS_PART_OF, PLAYER;

-- Funzioni interessate: 1
--  -> IS_ADMIN;

-- INPUT:
--      -> m_code:  codice carta del manager da rimuovere.
--      -> s_name:  nome della squadra allenata dal manager.
--      -> cc_name: se il chiamante e' amministratore: contiene
    il nome del club su cui effettuare le operazioni;
--              se il chiamante non e' amministratore: contiene
    o NULL, oppure il nome del club dell'utente chiamante.
-- OUTPUT:
--  -> Rimuove il manager <m_code> dalla squadra <s_name>.

CREATE OR REPLACE PROCEDURE REMOVE_MANAGER (
M_CODE   IN  CLUB_CARD.CARD_CODE%TYPE,
S_NAME   IN  SQUAD.NAME%TYPE,
CC_NAME  IN  SQUAD.SQUAD_CLUB_NAME%TYPE
) IS
--Contatori di controllo.
N1              NUMBER(2, 0);
N2              NUMBER(2, 0);

--Nome effettivo del club (is_admin).
C_NAME          CLUB.CLUB_NAME%TYPE;

--Bonus intesa del manager
P_NATION        PLAYER.NATIONALITY%TYPE; --Nazionalita' del <
   player>.
P_LEAGUE        PLAYER.LEAGUE_NAME%TYPE; --Lega del <player>
M_NATION        MANAGER.NATIONALITY%TYPE; --Nazionalita' del
   manager <m_code>.
M_LEAGUE        MANAGER.LEAGUE_NAME%TYPE; --Lega del manager <
   m_code>.

--Array per i codici carta dei player della squadra <s_name>.
TYPE ARRAY_CC_T IS VARRAY(18) OF VARCHAR2(8);
ARR_CARD_CODE  ARRAY_CC_T;
BEGIN
--La funzione is_admin, ritorna il nome del club in base all'
   utente che chiama la funzione REMOVE_MANAGER;
--Se e' l'amministratore a chiamare la funzione il nome del
   club <c_name> sara' uguale a <cc_name>
--Altrimenti il nome del club <c_name> sara' quello dell'utente
   .
--In breve l'amministratore puo' eseguire la funzione su
   qualsiasi club, l'utente solo sul suo club.
SELECT
    IS_ADMIN(CC_NAME, USER)
INTO C_NAME
FROM
    DUAL;
```

```
--Controllo che il manager <m_code> allena la squadra <s_name>.
SELECT
    COUNT(*)
INTO N1
FROM
    MANAGES
WHERE
        SQUAD_NAME = S_NAME
    AND MANAGER_CARD_CODE = M_CODE;

--Se <n1 = 0>, il manager con codice carta <m_code> non e'
   stato trovato.
IF N1 = 0 THEN
    RAISE NO_DATA_FOUND;
ELSE
--Elimino il manager dalla squadra <s_name>.
    DELETE FROM MANAGES
    WHERE
            MANAGER_CARD_CODE = M_CODE
        AND SQUAD_NAME = S_NAME;

--Seleziono i codici carta dei player della squadra <s_name> in
    <arr_card_code>.
    SELECT
        PLAYER_CARD_CODE
    BULK COLLECT
    INTO ARR_CARD_CODE
    FROM
        IS_PART_OF
    WHERE
            SQUAD_NAME = S_NAME;

--Seleziono nazionalita' e lega del manager <m_code>.
    SELECT
        NATIONALITY,
        LEAGUE_NAME
    INTO
        M_NATION,
        M_LEAGUE
    FROM
                MANAGER M
        JOIN CLUB_CARD C ON M.CARD_ID = C.MANAGER_ID
                            AND C.CARD_CODE = M_CODE;

--Per ogni player della squadra <s_name> che ha nazionalita' o
   lega uguale al manager.
-- Cioe' per ogni player che ha ricevuto il bonus sull'intesa <
   player_chemistry> dal manager,
--  rimuovo il bonus.
    FOR I IN 1..ARR_CARD_CODE.COUNT LOOP
        SELECT
            NATIONALITY,
            LEAGUE_NAME
        INTO
            P_NATION,
            P_LEAGUE
        FROM
```

```
                    PLAYER P
            JOIN CLUB_CARD C ON P.CARD_ID = C.PLAYER_ID
                                AND C.CARD_CODE = ARR_CARD_CODE
                                    (I);

--Aggiorno l'intesa <player_chemistry> dei player.
        IF P_NATION = M_NATION OR P_LEAGUE = M_LEAGUE THEN
            UPDATE IS_PART_OF
            SET
                PLAYER_CHEMISTRY = PLAYER_CHEMISTRY - 1
            WHERE
                    PLAYER_CARD_CODE = ARR_CARD_CODE(I)
                AND PLAYER_CHEMISTRY > 0;

--Confermo
            COMMIT;
        END IF;
    END LOOP;
END IF;

EXCEPTION
WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR('-20008', 'La squadra '
                                        || S_NAME
                                        || ' del club '
                                        || C_NAME
                                        || ' non ha un manager
                                            attivo!');
END REMOVE_MANAGER;
/
```

11.1.19 Check Duration

```
-- Procedura per Job Transaction:
--  -> Elimina le transazioni scadute (duration = 0),
--       e decrementa di uno l'attributo duration di transaction
   , viene chiamata solo dal job
--       transaction che si attiva ogni ora.
CREATE OR REPLACE PROCEDURE CHECK_DURATION IS
BEGIN
   --Elimino le transazioni scadute.
   DELETE FROM TRANSACTION
   WHERE
         DURATION = 0
      AND TRANSITION_B_CLUB_NAME IS NULL;

   --Aggiorno la durata rimanante poiche' e' passata un'ora.
   UPDATE TRANSACTION
   SET
      DURATION = DURATION - 1
   WHERE
      DURATION > 0;

   --Confermo
   COMMIT;
END CHECK_DURATION;
/
```

11.1.20 Gift

```
-- Procedura per Job Gift:
--     -> Gestisce la distribuzione dei premi settimanali per i
   club, in base alla divisione in cui militano.
CREATE OR REPLACE PROCEDURE GIFT IS
   N1        NUMBER(4, 0);
   TYPE U_ARR_T IS
      VARRAY(100) OF VARCHAR2(16);
   USR_ARR  U_ARR_T;
BEGIN
--Seleziono i nomi utenti di tutti i club.
   SELECT
      USER_NICK
   BULK COLLECT
   INTO USR_ARR
   FROM
      CLUB;

--Per ogni nome utente.
   FOR I IN 1..USR_ARR.COUNT LOOP
--Seleziono il numero di divione in cui si trova l'utente i-
   esimo.
      SELECT
         DIVISION_NUMBER
      INTO N1
      FROM
         CLUB
      WHERE
         USER_NICK = USR_ARR(I);

--Divione 9;
      IF N1 = 9 THEN
         PACK_OPENING('Bronze Pack', USR_ARR(I));
         PACK_OPENING('Bronze Player Pack', USR_ARR(I));
         UPDATE CLUB
         SET
            CREDITS = CREDITS + 1000
         WHERE
            USER_NICK = USR_ARR(I);
--Divione 8;
      ELSIF N1 = 8 THEN
         PACK_OPENING('Silver Pack', USR_ARR(I));
         PACK_OPENING('Silver Player Pack', USR_ARR(I));
         UPDATE CLUB
         SET
            CREDITS = CREDITS + 2000
         WHERE
            USER_NICK = USR_ARR(I);
--Divione 7;
      ELSIF N1 = 7 THEN
         PACK_OPENING('Silver Pack', USR_ARR(I));
         PACK_OPENING('Silver Player Pack', USR_ARR(I));
         PACK_OPENING('Silver Player Pack', USR_ARR(I));
         UPDATE CLUB
         SET
            CREDITS = CREDITS + 3000
```

```
        WHERE
            USER_NICK = USR_ARR(I);
--Divione 6;
    ELSIF N1 = 6 THEN
        PACK_OPENING('Silver Pack', USR_ARR(I));
        PACK_OPENING('Silver Player Pack', USR_ARR(I));
        PACK_OPENING('Silver Player Pack', USR_ARR(I));
        PACK_OPENING('Consumable Pack', USR_ARR(I));
        UPDATE CLUB
        SET
            CREDITS = CREDITS + 4000
        WHERE
            USER_NICK = USR_ARR(I);
--Divione 5;
    ELSIF N1 = 5 THEN
        PACK_OPENING('Gold Pack', USR_ARR(I));
        PACK_OPENING('Silver Player Pack', USR_ARR(I));
        PACK_OPENING('Gold Player Pack', USR_ARR(I));
        PACK_OPENING('Consumable Pack', USR_ARR(I));
        UPDATE CLUB
        SET
            CREDITS = CREDITS + 5000
        WHERE
            USER_NICK = USR_ARR(I);
--Divione 4;
    ELSIF N1 = 4 THEN
        PACK_OPENING('Gold Pack', USR_ARR(I));
        PACK_OPENING('Gold Player Pack', USR_ARR(I));
        PACK_OPENING('Premium Gold Pack', USR_ARR(I));
        PACK_OPENING('Consumable Pack', USR_ARR(I));
        UPDATE CLUB
        SET
            CREDITS = CREDITS + 7000
        WHERE
            USER_NICK = USR_ARR(I);
--Divione 3;
    ELSIF N1 = 3 THEN
        PACK_OPENING('Gold Pack', USR_ARR(I));
        PACK_OPENING('Gold Player Pack', USR_ARR(I));
        PACK_OPENING('Gold Player Pack', USR_ARR(I));
        PACK_OPENING('Consumable Pack', USR_ARR(I));
        PACK_OPENING('Jumbo Gold Pack', USR_ARR(I));
        UPDATE CLUB
        SET
            CREDITS = CREDITS + 9000
        WHERE
            USER_NICK = USR_ARR(I);
--Divione 2;
    ELSIF N1 = 2 THEN
        PACK_OPENING('Gold Pack', USR_ARR(I));
        PACK_OPENING('Gold Player Pack', USR_ARR(I));
        PACK_OPENING('Gold Player Pack', USR_ARR(I));
        PACK_OPENING('Consumable Pack', USR_ARR(I));
        PACK_OPENING('Jumbo Gold Pack', USR_ARR(I));
        PACK_OPENING('Rare Mega Pack', USR_ARR(I));
        UPDATE CLUB
        SET
```

```
            CREDITS = CREDITS + 12000
         WHERE
            USER_NICK = USR_ARR(I);
--Divione 1;
      ELSIF N1 = 1 THEN
         PACK_OPENING('Gold Pack', USR_ARR(I));
         PACK_OPENING('Gold Player Pack', USR_ARR(I));
         PACK_OPENING('Gold Player Pack', USR_ARR(I));
         PACK_OPENING('Consumable Pack', USR_ARR(I));
         PACK_OPENING('Jumbo Gold Pack', USR_ARR(I));
         PACK_OPENING('Rare Mega Pack', USR_ARR(I));
         PACK_OPENING('Ultimate Pack', USR_ARR(I));
         UPDATE CLUB
         SET
            CREDITS = CREDITS + 15000
         WHERE
            USER_NICK = USR_ARR(I);

      END IF;
   END LOOP;

   COMMIT;
END GIFT;
/
```

11.1.21 Number Of Cards

```
-- Tabelle interessate: 5
--  -> CLUB_CARD, IS_FOUND, PACK_PURCHASE, CLUB, TRANSACTION;

-- INPUT:
--      -> usr: nome utente;
-- OUTPUT:
--  -> Verifica del volume delle carte che non puo' superare le
    1500;
CREATE OR REPLACE PROCEDURE NUMBER_OF_CARDS (
USR CLUB.USER_NICK%TYPE
) IS
N1  NUMBER(4, 0);
N2  NUMBER(4, 0);
MAX_CARDS EXCEPTION;

BEGIN

--Quante carte provengono da pacchetti = N1.
SELECT
   COUNT(*)
INTO N1
FROM
   CLUB_CARD
WHERE
   CARD_CODE IN (
      SELECT
         CARD_CODE
      FROM
         IS_FOUND
      WHERE
         P_ID IN (
            SELECT
               PURCHASE_ID
            FROM
               PACK_PURCHASE
            WHERE
               BUYING_CLUB_NAME IN (
                  SELECT
                     CLUB_NAME
                  FROM
                     CLUB
                  WHERE
                     LOWER(USER_NICK) = LOWER(USR)
               )
         )
   );

--Quante carte ha comprato = N2.
SELECT
   COUNT(*)
INTO N2
FROM
   CLUB_CARD
WHERE
   CARD_CODE IN (
```

```
        SELECT
            T_CARD_CODE
        FROM
            TRANSACTION
        WHERE
            TRANSITION_B_CLUB_NAME IN (
                SELECT
                    CLUB_NAME
                FROM
                    CLUB
                WHERE
                    LOWER(USER_NICK) = LOWER(USR)
            )
    );

--Carte totali possedute dal cclub = N1
N1 := N1 + N2;
IF N1 > 1500 THEN
    RAISE MAX_CARDS;
ELSE
    DBMS_OUTPUT.PUT_LINE('Hai collezionato ' || N1 || ' carte,
        restano altri '
                         ||(1500 - N1)
                         || ' posti disponibili!');
END IF;

EXCEPTION
    WHEN MAX_CARDS THEN
        RAISE_APPLICATION_ERROR('-20005', 'Attenzione! Possiedi '
                                          || N1
                                          || ' cards, hai
                                             raggiunto il limite
                                             massimo!');
END NUMBER_OF_CARDS;
/
```

11.1.22 Club Delete

```
-- Tabelle interessate: 2
--  -> CLUB_CARD, CLUB;

-- Funzioni interessate: 1
--  -> IS_ADMIN;

-- INPUT:
--      -> cc_name: se il chiamante e' amministratore: contiene
    il nome del club su cui effettuare le operazioni;
--              se il chiamante non e' amministratore: contiene
    o NULL, oppure il nome del club dell'utente chiamante.
-- OUTPUT:
--  -> Cancella il club.

CREATE OR REPLACE PROCEDURE CLUB_DELETE (
CC_NAME USER_FUT.NICKNAME%TYPE
) IS
N1      NUMBER(2, 0); --Controlla che il club esista.
C_NAME  CLUB.CLUB_NAME%TYPE; --Nome del club effettivo (
   is_admin).

NO_CLUB_FOUND EXCEPTION; --Nel caso in cui non viene trovato il
    club da eliminare.

BEGIN
--La funzione is_admin, ritorna il nome del club in base all'
   utente che chiama la funzione CLUB_DELETE;
--Se e' l'amministratore a chiamare la funzione il nome del
   club <c_name> sara' uguale a <cc_name>
--Altrimenti il nome del club <c_name> sara' quello dell'utente
   .
--In breve l'amministratore puo' eseguire la funzione su
   qualsiasi club, l'utente solo sul suo club.
SELECT
    IS_ADMIN(CC_NAME, USER)
INTO C_NAME
FROM
    DUAL;

--Controllo che il club se si vuole eliminare, esista.
SELECT
    COUNT(*)
INTO N1
FROM
    CLUB
WHERE
    CLUB_NAME = C_NAME;

--Se <n1 = 0>, il club <c_name> non e' stato trovato.
IF N1 = 0 THEN
    RAISE NO_CLUB_FOUND;
END IF;

--Elimino tutte le carte del club.
DELETE FROM CLUB_CARD
```

```
WHERE
    CARD_CODE IN (
        SELECT
            CARD_CODE
        FROM
            IS_FOUND
        WHERE
            P_ID IN (
                SELECT
                    PURCHASE_ID
                FROM
                    PACK_PURCHASE
                WHERE
                    BUYING_CLUB_NAME = C_NAME
            )
    );

--Elimino il club
DELETE FROM CLUB
WHERE
    CLUB_NAME = C_NAME;

--Confermo
COMMIT;
EXCEPTION
WHEN NO_CLUB_FOUND THEN
    RAISE_APPLICATION_ERROR('-20001', 'Il club '
                                        || C_NAME
                                        || ' non e'' stato
                                           trovato');
END CLUB_DELETE;
/
```

11.2 User Functions

11.2.1 Get Squad

```
-- Tabelle interessate: 4
--  -> PLAYER, CLUB_CARD, ACTIVE_DATA_PLAYER, IS_PART_OF;

-- Function interessate: 1
--      -> IS_ADMIN;

-- INPUT:
--      -> s_club_name: nome del club;
--      -> s_name: nome della squadra;
-- OUTPUT:
--      -> Dettagli di tutti i player che fanno parte della
   squadra.

CREATE OR REPLACE FUNCTION GET_SQUAD (
S_CLUB_NAME  SQUAD.SQUAD_CLUB_NAME%TYPE,
S_NAME       SQUAD.NAME%TYPE
) RETURN SYS_REFCURSOR IS
GETSQUAD  SYS_REFCURSOR;          --Valore di ritorno.
C_NAME    CLUB.CLUB_NAME%TYPE;      --Nome del club.
BEGIN
--La funzione is_admin, ritorna il nome del club in base all'
   utente che chiama la funzione GET_SQUAD;
--Se e' l'amministratore a chiamare la funzione il nome del
   club <c_name> sara' uguale a <s_club_name>
--Altrimenti il nome del club <c_name> sara' quello dell'utente
   .
--In breve l'amministratore puo' eseguire la funzione su
   qualsiasi club, l'utente solo sul suo club.
SELECT
    IS_ADMIN(S_CLUB_NAME, USER)
INTO C_NAME
FROM
    DUAL;

OPEN GETSQUAD FOR
    SELECT
        CARD_ID,
        CARD_CODE,
        PLAYER_NAME,
        POSITION         AS P_POS,
        PLAYER_POSITION  AS S_POS,
        HOLDER,
        CONTRACTS,
        PLAYER_FITNESS,
        ACTIVE_TRAINING  AS TRAIN,
        PLAYER_CHEMISTRY,
        PLAYER_RATING,
        NATIONALITY,
        LEAGUE_NAME
    FROM
            PLAYER P
        JOIN CLUB_CARD           C                --JOIN: per
           selezionare il CARD_CODE della carta.
```

```
        ON P.CARD_ID = C.PLAYER_ID
        JOIN ACTIVE_DATA_PLAYER  A       --JOIN: per
           selezionare CONTRACTS,PLAYER_FITNESS della carta.
        ON A.P_CARD_CODE = C.CARD_CODE
        JOIN IS_PART_OF          I              --JOIN: per
           selezionare PLAYER_POSITION,HOLDER,PLAYER_CHEMISTRY,
           PLAYER_RATING.
        ON I.PLAYER_CARD_CODE = A.P_CARD_CODE
                          AND I.SQUAD_NAME = S_NAME;
RETURN GETSQUAD;
END GET_SQUAD;
/
```

11.2.2 Search Card For Sale

```
-- INPUT:
--      -> cc_name: se il chiamante e' amministratore: contiene
    il nome del club su cui effettuare le operazioni;
--              se il chiamante non e' amministratore: contiene
    o NULL, oppure il nome del club dell'utente chiamante.
--  -> c_type: ('Player','Consumable','Club Item','Manager')
   tipologia di carta;
--  -> c_name: ('Player_name','Category','Category','F_name')
   nome della carta in base alla tipologia;
-- OUTPUT:
--  -> Se la carta cercata e' in vendita ritorna il codice
   carta e il prezzo.

CREATE OR REPLACE FUNCTION SEARCH_CARD_FOR_SALE (
CLUB_NN  CLUB.CLUB_NAME%TYPE,
C_TYPE   VARCHAR2,
C_NAME   VARCHAR2
) RETURN SYS_REFCURSOR IS
S_CARD  SYS_REFCURSOR;          --Valore di ritorno.
N1      NUMBER(2, 0);            --Contatore.
CLUB_N  CLUB.CLUB_NAME%TYPE;    --Nome del club effettivo (
   is_admin).
BEGIN
--La funzione is_admin, ritorna il nome del club in base all'
   utente che chiama la funzione SEARCH_CARD_FOR_SALE;
--Se e' l'amministratore a chiamare la funzione il nome del
   club <c_name> sara' uguale a <cc_name>
--Altrimenti il nome del club <c_name> sara' quello dell'utente
   .
--In breve l'amministratore puo' eseguire la funzione su
   qualsiasi club, l'utente solo sul proprio.
SELECT
    IS_ADMIN(CLUB_NN, USER)
INTO CLUB_N
FROM
    DUAL;

--Se la tipologia di carta che stiamo cercando e' di tipo <
   player>.
IF C_TYPE = 'Player' THEN

--Controllo se c'e' in vendita il <player> cercato.
    SELECT
        COUNT(PLAYER_NAME)
    INTO N1
    FROM
        PLAYER
    WHERE
            PLAYER_NAME = C_NAME
        AND CARD_ID IN (
            SELECT
                PLAYER_ID
            FROM
                CLUB_CARD
            WHERE
```

```
                CARD_CODE IN (
                    SELECT
                        T_CARD_CODE
                    FROM
                        TRANSACTION
--Il club compratore deve essere null, altrimenti e' stato
   acquistato;
-- il club venditore deve essere diverso dal club che sta
   cercando la carta in vendita.
                    WHERE
                        TRANSITION_B_CLUB_NAME IS NULL
                        AND TRANSITION_S_CLUB_NAME != CLUB_N
                )
        );
--Se n1 != 0 ho trovato il player, seleziono il codice carta e
   il prezzo.
    IF N1 <> 0 THEN
        OPEN S_CARD FOR
            SELECT
                T_CARD_CODE,
                PRICE
            FROM
                TRANSACTION
            WHERE
                T_CARD_CODE IN (
                    SELECT
                        CARD_CODE
                    FROM
                        CLUB_CARD
                    WHERE
                        PLAYER_ID IN (
                            SELECT
                                CARD_ID
                            FROM
                                PLAYER
                            WHERE
                                PLAYER_NAME = C_NAME
                        )
                );

        RETURN S_CARD;
    ELSE--Se non trovo il <player> che sto cercando.
        RAISE NO_DATA_FOUND;
    END IF;
--Se la tipologia di carta scelta e' di tipo <Consumable>.
ELSIF C_TYPE = 'Consumable' THEN

--La categoria <c_name> del consumabile puo' essere:
-- ('Contract','Fitness','Healing','Training','GKTraining','
   Positioning','Manager League').
    SELECT
        COUNT(CATEGORY)
    INTO N1
    FROM
        CONSUMABLE
    WHERE
            CATEGORY = C_NAME
```

```
        AND CARD_ID IN (
            SELECT
                CONSUMABLE_ID
            FROM
                CLUB_CARD
            WHERE
                CARD_CODE IN (
                    SELECT
                        T_CARD_CODE
                    FROM
                        TRANSACTION
                    WHERE
                        TRANSITION_B_CLUB_NAME IS NULL
                        AND TRANSITION_S_CLUB_NAME != CLUB_N
                )
        );

    IF N1 <> 0 THEN
        OPEN S_CARD FOR
            SELECT
                T_CARD_CODE,
                PRICE
            FROM
                TRANSACTION
            WHERE
                T_CARD_CODE IN (
                    SELECT
                        CARD_CODE
                    FROM
                        CLUB_CARD
                    WHERE
                        CONSUMABLE_ID IN (
                            SELECT
                                CARD_ID
                            FROM
                                CONSUMABLE
                            WHERE
                                CATEGORY = C_NAME
                        )
                );

        RETURN S_CARD;
    ELSE
        RAISE NO_DATA_FOUND;
    END IF;
--Se la tipologia di carta e' di tipo <Club item>.
ELSIF C_TYPE = 'Club item' THEN

--La categoria <c_name> del club_item puo' essere:
--  ('Kit','Ball','Badge','Stadium').
    SELECT
        COUNT(CATEGORY_ITEM)
    INTO N1
    FROM
        CLUB_ITEM
    WHERE
            CATEGORY_ITEM = C_NAME
```

```
        AND CARD_ID IN (
            SELECT
                CLUB_ITEM_ID
            FROM
                CLUB_CARD
            WHERE
                CARD_CODE IN (
                    SELECT
                        T_CARD_CODE
                    FROM
                        TRANSACTION
                    WHERE
                        TRANSITION_B_CLUB_NAME IS NULL
                        AND TRANSITION_S_CLUB_NAME != CLUB_N
                )
        );

    IF N1 <> 0 THEN
        OPEN S_CARD FOR
            SELECT
                T_CARD_CODE,
                PRICE
            FROM
                TRANSACTION
            WHERE
                T_CARD_CODE IN (
                    SELECT
                        CARD_CODE
                    FROM
                        CLUB_CARD
                    WHERE
                        CLUB_ITEM_ID IN (
                            SELECT
                                CARD_ID
                            FROM
                                CLUB_ITEM
                            WHERE
                                CATEGORY_ITEM = C_NAME
                        )
                );

        RETURN S_CARD;
    ELSE
        RAISE NO_DATA_FOUND;
    END IF;

--Se la tipologia di carta e' di tipo <Manager>.
ELSIF C_TYPE = 'Manager' THEN
    SELECT
        COUNT(F_NAME)
    INTO N1
    FROM
        MANAGER
    WHERE
            F_NAME = C_NAME
        AND CARD_ID IN (
            SELECT
```

```
                MANAGER_ID
            FROM
                CLUB_CARD
            WHERE
                CARD_CODE IN (
                    SELECT
                        T_CARD_CODE
                    FROM
                        TRANSACTION
                    WHERE
                        TRANSITION_B_CLUB_NAME IS NULL
                        AND TRANSITION_S_CLUB_NAME != CLUB_N
                )
        );

    IF N1 <> 0 THEN
        OPEN S_CARD FOR
            SELECT
                T_CARD_CODE,
                PRICE
            FROM
                TRANSACTION
            WHERE
                T_CARD_CODE IN (
                    SELECT
                        CARD_CODE
                    FROM
                        CLUB_CARD
                    WHERE
                        MANAGER_ID IN (
                            SELECT
                                CARD_ID
                            FROM
                                MANAGER
                            WHERE
                                F_NAME = C_NAME
                        )
                );

        RETURN S_CARD;
    ELSE
        RAISE NO_DATA_FOUND;
    END IF;

END IF;

EXCEPTION
WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('La carta '
                         || C_TYPE
                         || ' '
                         || C_NAME
                         || ' non e'' in vendita''');
END SEARCH_CARD_FOR_SALE;
/
```

11.3 User View

11.3.1 Get Club Info

```
-- Tabelle interessate: 1
--   -> CLUB;
-- OUTPUT:
--   -> Dettagli deL club dell'utente.
CREATE OR REPLACE VIEW GET_CLUB_INFO AS
( SELECT
    *
FROM
    CLUB
WHERE
    LOWER(USER_NICK) = LOWER(USER)
);
```

11.3.2 Get Aavilable Packs

```
-- Tabelle interessate: 1
    -> PACK;
-- OUTPUT:
--   -> Dettagli dei pacchetti con relativo prezzo, disponibili
   per l'acquisto.
CREATE OR REPLACE VIEW GET_AVAILABLE_PACKS AS
( SELECT
    PACK_NAME,
    PRICE
FROM
    PACK
WHERE
    AVAILABLE = '1'
);
```

11.3.3 Get Pack Purchases

```
-- Tabelle interessate: 2
--   -> PACK_PURCHASE, PACK, CLUB;
-- OUTPUT:
--   -> Cronologia dei pacchetti comprati dall'utente.
CREATE OR REPLACE VIEW GET_PACK_PURCHASES AS
( SELECT
    PURCHASE_ID,
    BUYING_PACK_NAME,
    P_DATE,
    PRICE
FROM
        PACK_PURCHASE P
    JOIN PACK PP
        ON P.BUYING_PACK_NAME = PP.PACK_NAME
            AND P.BUYING_CLUB_NAME IN (
        SELECT CLUB_NAME
        FROM CLUB
        WHERE LOWER(USER_NICK) = LOWER(USER)
    )
);
```

11.3.4 Get Players For Sale

```
-- Tabelle interessate: 6
--  -> CLUB_CARD, PLAYER, CLUB, TRANSACTION;
-- OUTPUT:
--  -> Dettagli dei player in vendita sul mercato.
CREATE OR REPLACE VIEW GET_PLAYERS_FOR_SALE AS
( SELECT
    PRICE,
    CARD_CODE,
    PLAYER_NAME,
    POSITION,
    NATIONALITY,
    LEAGUE_NAME
FROM
            PLAYER P
    JOIN CLUB_CARD   C ON P.CARD_ID = C.PLAYER_ID
    JOIN TRANSACTION  T ON T.T_CARD_CODE = C.CARD_CODE
                           AND C.CARD_CODE IN (
        SELECT T_CARD_CODE
        FROM TRANSACTION
        WHERE TRANSITION_B_CLUB_NAME IS NULL
            AND TRANSITION_S_CLUB_NAME IN (
                SELECT CLUB_NAME
                FROM CLUB
                WHERE LOWER(USER_NICK) <> LOWER(USER)))
);
```

11.3.5 Get Players

```
-- Tabelle interessate: 6
--  -> CLUB_CARD, PLAYER, IS_FOUND, PACK_PURCHSE, CLUB,
   TRANSACTION;
-- OUTPUT:
--  -> Dettagli dei player posseduti dal club dell'utente.
CREATE OR REPLACE VIEW GET_PLAYERS AS
( SELECT
    CARD_CODE,
    PLAYER_NAME,
    POSITION,
    OVERALL,
    NATIONALITY,
    LEAGUE_NAME,
    CONTRACTS,
    PLAYER_FITNESS,
    ACTIVE_TRAINING AS TRAIN
FROM
            PLAYER P
    JOIN CLUB_CARD C ON P.CARD_ID = C.PLAYER_ID
    JOIN ACTIVE_DATA_PLAYER A ON A.P_CARD_CODE = C.CARD_CODE
                        AND C.CARD_CODE IN (
        SELECT
            CARD_CODE
        FROM
            IS_FOUND
        WHERE
            P_ID IN (
                SELECT
                    PURCHASE_ID
                FROM
                    PACK_PURCHASE
                WHERE
                    BUYING_CLUB_NAME IN (
                        SELECT
                            CLUB_NAME
                        FROM
                            CLUB
                        WHERE
                            LOWER(USER_NICK) = LOWER(USER)
                    )
            )
    )
UNION
SELECT
    CARD_CODE,
    PLAYER_NAME,
    POSITION,
    OVERALL,
    NATIONALITY,
    LEAGUE_NAME,
    CONTRACTS,
    PLAYER_FITNESS,
    ACTIVE_TRAINING AS TRAIN
FROM
            PLAYER P
```

```
    JOIN CLUB_CARD C ON P.CARD_ID = C.PLAYER_ID
    JOIN ACTIVE_DATA_PLAYER A ON A.P_CARD_CODE = C.CARD_CODE
                        AND C.CARD_CODE IN (
        SELECT
            T_CARD_CODE
        FROM
            TRANSACTION
        WHERE
            TRANSITION_B_CLUB_NAME IN (
                SELECT
                    CLUB_NAME
                FROM
                    CLUB
                WHERE
                    LOWER(USER_NICK) = LOWER(USER)
            )
    )
);
```

11.3.6 Get Players Stops

```
-- Tabelle interessate: 6
--  -> CLUB_CARD, PLAYER, IS_PART_OF, IS_FOUND, PACK_PURCHSE,
   CLUB, TRANSACTION;
-- OUTPUT:
--  -> Dettagli dei player infortunati del club dell'utente.
CREATE OR REPLACE VIEW GET_PLAYERS_STOPS AS
( SELECT
    P_CARD_CODE,
    PLAYER_NAME,
    PLAYER_POSITION,
    HOLDER,
    S_DATE,
    BODY_PART,
    DAYS
FROM
            STOPS S
    JOIN CLUB_CARD   C ON S.P_CARD_CODE = C.CARD_CODE
    JOIN PLAYER      P ON C.PLAYER_ID = P.CARD_ID
    JOIN IS_PART_OF  I ON I.PLAYER_CARD_CODE = C.CARD_CODE
                          AND C.CARD_CODE IN (
        SELECT CARD_CODE
        FROM IS_FOUND
        WHERE P_ID IN (
                SELECT PURCHASE_ID
                FROM PACK_PURCHASE
                WHERE BUYING_CLUB_NAME IN (
                        SELECT CLUB_NAME
                        FROM CLUB
                        WHERE LOWER(USER_NICK) = LOWER(USER))))
UNION
SELECT
    P_CARD_CODE,
    PLAYER_NAME,
    PLAYER_POSITION,
    HOLDER,
    S_DATE,
    BODY_PART,
    DAYS
FROM
            STOPS S
    JOIN CLUB_CARD   C ON S.P_CARD_CODE = C.CARD_CODE
    JOIN PLAYER      P ON C.PLAYER_ID = P.CARD_ID
    JOIN IS_PART_OF  I ON I.PLAYER_CARD_CODE = C.CARD_CODE
                          AND C.CARD_CODE IN (
        SELECT T_CARD_CODE
        FROM TRANSACTION
        WHERE TRANSITION_B_CLUB_NAME IN (
                SELECT CLUB_NAME
                FROM CLUB
                WHERE LOWER(USER_NICK) = LOWER(USER)))
);
```

11.3.7 Get Club Items

```
-- Tabelle interessate: 5
--  -> CLUB_ITEM, CLUB_CARD, IS_FOUND, PACK_PURCHASE,
   TRANSACTION;
-- OUTPUT:
--  -> Dettagli dei club_item posseduti dal club dell'utente.
CREATE OR REPLACE VIEW GET_CLUB_ITEMS AS
( SELECT
    CARD_CODE,
    CARD_ID,
    CATEGORY_ITEM,
    TEAM_NAME,
    RARITY_NAME,
    MIN_PRICE,
    MAX_PRICE
FROM
            CLUB_ITEM C
    JOIN CLUB_CARD CC ON C.CARD_ID = CC.CLUB_ITEM_ID
                         AND CC.CARD_CODE IN (
        SELECT
            CARD_CODE
        FROM IS_FOUND
        WHERE P_ID IN (
                SELECT PURCHASE_ID
                FROM PACK_PURCHASE
                WHERE BUYING_CLUB_NAME IN (
                        SELECT CLUB_NAME
                        FROM CLUB
                        WHERE LOWER(USER_NICK) = LOWER(USER))))
UNION
SELECT
    CARD_CODE,
    CARD_ID,
    CATEGORY_ITEM,
    TEAM_NAME,
    RARITY_NAME,
    MIN_PRICE,
    MAX_PRICE
FROM
            CLUB_ITEM C
    JOIN CLUB_CARD CC ON C.CARD_ID = CC.CLUB_ITEM_ID
                         AND CC.CARD_CODE IN (
        SELECT
            T_CARD_CODE
        FROM TRANSACTION
        WHERE TRANSITION_B_CLUB_NAME IN (
                SELECT CLUB_NAME
                FROM CLUB
                WHERE LOWER(USER_NICK) = LOWER(USER)))
);
```

11.3.8 Get Active Club Items

```
-- Tabelle interessate: 3
--  -> CLUB_ITEM, CLUB_CARD, CLUB;
-- OUTPUT:
--  -> Dettagli dei club_item attivi del club dell'utente.
CREATE OR REPLACE VIEW GET_ACTIVE_CLUB_ITEMS
 AS
(
--Seleziona i dettagli che riguardano lo stemma <badge> del
   club.
    SELECT
    CARD_ID,
    CATEGORY_ITEM,
    TEAM_NAME
FROM
            CLUB_ITEM C
    JOIN CLUB_CARD CC ON C.CARD_ID = CC.CLUB_ITEM_ID
                         AND CC.CARD_CODE IN (
        SELECT
            ACTIVE_BADGE_CODE
        FROM CLUB
        WHERE CLUB_NAME IN (
                SELECT
                    CLUB_NAME
                FROM
                    CLUB
                WHERE
                    LOWER(USER_NICK) = LOWER(USER)))
UNION
--Seleziona i dettagli che riguardano lo stadio del club.
SELECT
    CARD_ID,
    CATEGORY_ITEM,
    TEAM_NAME
FROM
            CLUB_ITEM C
    JOIN CLUB_CARD CC ON C.CARD_ID = CC.CLUB_ITEM_ID
                         AND CC.CARD_CODE IN (
        SELECT
            ACTIVE_STADIUM_CODE
        FROM CLUB
        WHERE CLUB_NAME IN (
                SELECT
                    CLUB_NAME
                FROM
                    CLUB
                WHERE
                    LOWER(USER_NICK) = LOWER(USER)))
UNION
--Seleziona i dettagli che riguardano il pallone del club.
SELECT
    CARD_ID,
    CATEGORY_ITEM,
    TEAM_NAME
FROM
            CLUB_ITEM C
```

```
    JOIN CLUB_CARD CC ON C.CARD_ID = CC.CLUB_ITEM_ID
                            AND CC.CARD_CODE IN (
        SELECT
            ACTIVE_BALL_CODE
        FROM CLUB
        WHERE CLUB_NAME IN (
                SELECT
                    CLUB_NAME
                FROM
                    CLUB
                WHERE
                    LOWER(USER_NICK) = LOWER(USER)))
UNION
--Seleziona i dettagli che riguardano la divisa in casa del
   club.
SELECT
    CARD_ID,
    CATEGORY_ITEM,
    TEAM_NAME
FROM
            CLUB_ITEM C
    JOIN CLUB_CARD CC ON C.CARD_ID = CC.CLUB_ITEM_ID
                            AND CC.CARD_CODE IN (
        SELECT
            ACTIVE_FIRST_KIT_CODE
        FROM CLUB
        WHERE CLUB_NAME IN (
                SELECT
                    CLUB_NAME
                FROM
                    CLUB
                WHERE
                    LOWER(USER_NICK) = LOWER(USER)))
UNION
--Seleziona i dettagli che riguardano la divisa in trasferta
   del club.
SELECT
    CARD_ID,
    CATEGORY_ITEM,
    TEAM_NAME
FROM
            CLUB_ITEM C
    JOIN CLUB_CARD CC ON C.CARD_ID = CC.CLUB_ITEM_ID
                            AND CC.CARD_CODE IN (
        SELECT
            ACTIVE_SECOND_KIT_CODE
        FROM CLUB
        WHERE CLUB_NAME IN (
                SELECT
                    CLUB_NAME
                FROM
                    CLUB
                WHERE
                    LOWER(USER_NICK) = LOWER(USER)))
);
```

11.3.9 Get Consumables

```
-- Tabelle interessate: 6
--  -> CLUB_CARD, CONSUMABLE, IS_FOUND, PACK_PURCHSE, CLUB,
   TRANSACTION;
-- OUTPUT:
--  -> Dettagli dei consumabili posseduti dal club dell'utente.
CREATE OR REPLACE VIEW GET_CONSUMABLES AS
( SELECT
    CARD_CODE,
    CARD_ID,
    RARITY_NAME,
    TYPE,
    CATEGORY,
    AMOUNT,
    BRONZE_CONTRACT,
    SILVER_CONTRACT,
    GOLD_CONTRACT,
    MIN_PRICE,
    MAX_PRICE
FROM CONSUMABLE C
    JOIN CLUB_CARD CC ON C.CARD_ID = CC.CONSUMABLE_ID
                         AND CC.CARD_CODE IN (
        SELECT
            CARD_CODE
        FROM IS_FOUND
        WHERE P_ID IN (
                SELECT PURCHASE_ID
                FROM PACK_PURCHASE
                WHERE BUYING_CLUB_NAME IN (
                        SELECT CLUB_NAME
                        FROM CLUB
                        WHERE LOWER(USER_NICK) = LOWER(USER))))
UNION
SELECT
    CARD_CODE,
    CARD_ID,
    RARITY_NAME,
    TYPE,
    CATEGORY,
    AMOUNT,
    BRONZE_CONTRACT,
    SILVER_CONTRACT,
    GOLD_CONTRACT,
    MIN_PRICE,
    MAX_PRICE
FROM CONSUMABLE C
    JOIN CLUB_CARD CC ON C.CARD_ID = CC.CONSUMABLE_ID
                         AND CC.CARD_CODE IN (
        SELECT
            T_CARD_CODE
        FROM TRANSACTION
        WHERE TRANSITION_B_CLUB_NAME IN (
                SELECT CLUB_NAME
                FROM CLUB
                WHERE LOWER(USER_NICK) = LOWER(USER)))
);
```

11.3.10 Get Managers

```
-- Tabelle interessate: 8
--  -> CLUB_CARD, MANAGER, ACTIVE_DATA_MANAGER, MANAGES,
   IS_FOUND, PACK_PURCHSE, CLUB, TRANSACTION;
-- OUTPUT:
--  -> Dettagli dei dati attivi dei manager posseduti dal club
   dell'utente.
CREATE OR REPLACE VIEW GET_MANAGERS AS
( SELECT
    CARD_CODE,
    F_NAME,
    S_NAME,
    CONTRACTS,
    LEAGUE_NAME     AS PREF_LEAGUE,
    MANAGER_LEAGUE  AS ACTIVE_LEAGUE,
    NATIONALITY
FROM
            MANAGER M
    JOIN CLUB_CARD            C ON M.CARD_ID = C.MANAGER_ID
    JOIN ACTIVE_DATA_MANAGER  A ON A.M_CARD_CODE = C.CARD_CODE
                        AND C.CARD_CODE IN (
        SELECT
            CARD_CODE
        FROM
            IS_FOUND
        WHERE
            P_ID IN (
                SELECT
                    PURCHASE_ID
                FROM
                    PACK_PURCHASE
                WHERE
                    BUYING_CLUB_NAME IN (
                        SELECT
                            CLUB_NAME
                        FROM
                            CLUB
                        WHERE
                            LOWER(USER_NICK) = LOWER(USER)
                    )
            )
    )
UNION
SELECT
    CARD_CODE,
    F_NAME,
    S_NAME,
    CONTRACTS,
    LEAGUE_NAME     AS PREF_LEAGUE,
    MANAGER_LEAGUE  AS ACTIVE_LEAGUE,
    NATIONALITY
FROM
            MANAGER M
    JOIN CLUB_CARD            C ON M.CARD_ID = C.MANAGER_ID
    JOIN ACTIVE_DATA_MANAGER  A ON A.M_CARD_CODE = C.CARD_CODE
                        AND C.CARD_CODE IN (
```

```
        SELECT
            T_CARD_CODE
        FROM
            TRANSACTION
        WHERE
            TRANSITION_B_CLUB_NAME IN (
                SELECT
                    CLUB_NAME
                FROM
                    CLUB
                WHERE
                    LOWER(USER_NICK) = LOWER(USER)
            )
    )
);
```

11.3.11 Get Manages

```
-- Tabelle interessate: 8
--  -> CLUB_CARD, MANAGER, ACTIVE_DATA_MANAGER, MANAGES,
   IS_FOUND, PACK_PURCHSE, CLUB, TRANSACTION;
-- OUTPUT:
--  -> Dettagli dei dati attivi dei manager con relativa
   squadra, del club dell'utente.
CREATE OR REPLACE VIEW GET_MANAGES AS
( SELECT
    CARD_CODE,
    F_NAME,
    S_NAME,
    CONTRACTS,
    LEAGUE_NAME      AS PREF_LEAGUE,
    MANAGER_LEAGUE   AS ACTIVE_LEAGUE,
    NATIONALITY,
    SQUAD_NAME       AS SQUAD_MANAGES,
    CLUB_NAME
FROM MANAGER M
    JOIN CLUB_CARD             C ON M.CARD_ID = C.MANAGER_ID
    JOIN ACTIVE_DATA_MANAGER   A ON A.M_CARD_CODE = C.CARD_CODE
    JOIN MANAGES               MM ON MM.MANAGER_CARD_CODE = A.
       M_CARD_CODE
                      AND C.CARD_CODE IN (
        SELECT CARD_CODE
        FROM IS_FOUND
        WHERE P_ID IN (
                SELECT PURCHASE_ID
                FROM PACK_PURCHASE
                WHERE BUYING_CLUB_NAME IN (
                        SELECT CLUB_NAME
                        FROM CLUB
                        WHERE LOWER(USER_NICK) = LOWER(USER))))
UNION
SELECT
    CARD_CODE,
    F_NAME,
    S_NAME,
    CONTRACTS,
    LEAGUE_NAME      AS PREF_LEAGUE,
    MANAGER_LEAGUE   AS ACTIVE_LEAGUE,
    NATIONALITY,
    SQUAD_NAME       AS SQUAD_MANAGES,
    CLUB_NAME
FROM MANAGER M
    JOIN CLUB_CARD             C ON M.CARD_ID = C.MANAGER_ID
    JOIN ACTIVE_DATA_MANAGER   A ON A.M_CARD_CODE = C.CARD_CODE
    JOIN MANAGES               MM ON MM.MANAGER_CARD_CODE = A.
       M_CARD_CODE
                      AND C.CARD_CODE IN (
        SELECT T_CARD_CODE
        FROM TRANSACTION
        WHERE TRANSITION_B_CLUB_NAME IN (
                SELECT CLUB_NAME
                FROM CLUB
                WHERE LOWER(USER_NICK) = LOWER(USER))));
```

11.3.12 Get Match

```
-- Tabelle interessate: 3
--  -> MATCH, SQUAD, CLUB;
-- OUTPUT:
--  -> Cronologia delle partite giocate dal club dell'utente.
CREATE OR REPLACE VIEW GET_MATCH AS
( SELECT
    M_DATE,
    HOME_SQUAD_NAME,
    VISITORS_SQUAD_NAME,
    RESULTS,
    HOME_POINTS   AS POINTS,
    HOME_CREDITS  AS CREDITS
FROM
            MATCH M
    JOIN SQUAD S ON M.HOME_SQUAD_NAME = S.NAME
                    AND S.SQUAD_CLUB_NAME IN (
        SELECT
            CLUB_NAME
        FROM
            CLUB
        WHERE
            LOWER(USER_NICK) = LOWER(USER)
    )
UNION
SELECT
    M_DATE,
    HOME_SQUAD_NAME,
    VISITORS_SQUAD_NAME,
    RESULTS,
    VISITORS_POINTS,
    VISITORS_CREDITS
FROM
            MATCH M
    JOIN SQUAD S ON M.VISITORS_SQUAD_NAME = S.NAME
                    AND S.SQUAD_CLUB_NAME IN (
        SELECT
            CLUB_NAME
        FROM
            CLUB
        WHERE
            LOWER(USER_NICK) = LOWER(USER)
    )
);
```

11.4 Admin Functions

11.4.1 Is Admin

```
-- Tabelle interessate: 1
--  -> CLUB

-- INPUT:
--      -> input_club_name: nome del club dato in ingresso;
--      -> user_logon: nome utente dell'utente che chiama la
   funzione;
-- OUTPUT:
--      -> Se non e' l'admin a chiamare la funzione, ritorna il
    nome del club del utente chiamante.
--  -> Altrimenti ritorna il nome del club, che corrisponde a <
   input_club_name>, poiche' l'amministratore
--  ->  puo' chiamare le funzioni su qualsiasi club, mentre l'
   utente solo sul proprio.

CREATE OR REPLACE FUNCTION IS_ADMIN (
INPUT_CLUB_NAME  CLUB.CLUB_NAME%TYPE,
USER_LOGON       CLUB.USER_NICK%TYPE
) RETURN VARCHAR2 IS

N1                NUMBER(2, 0);             --Contatore.
USER_CLUB_NAME    CLUB.CLUB_NAME%TYPE;     --Nome del club dell'
   utente.
ACTUAL_CLUB_NAME  CLUB.CLUB_NAME%TYPE;     --Variabile d'
   appoggio.

NO_CLUB_FOUND EXCEPTION; --Se il club <input_club_name> non
   viene trovato.
--Nel caso in cui l'utente A chiama la funzione con il nome del
    club dell'utente B
-- o con un nome club non valido.
USER_CLUB_ERROR EXCEPTION;

BEGIN
--Copio il nome del club dato in input.
ACTUAL_CLUB_NAME := INPUT_CLUB_NAME;

--Se l'utente chiamante non e' amministratore.
IF USER_LOGON <> 'ADMINFUT' THEN

--Controllo che esiste un club di propieta' dell'utente loggato
    <user_logon>.
    SELECT
        COUNT(*)
    INTO N1
    FROM
        CLUB
    WHERE
        LOWER(USER_NICK) = LOWER(USER_LOGON);

--Se non esiste il club.
    IF N1 = 0 THEN
        RAISE NO_CLUB_FOUND;
```

```
    END IF;

--Se esiste il lcub, seleziono il nome del club in <
   user_club_name>.
    SELECT
        CLUB_NAME
    INTO USER_CLUB_NAME
    FROM
        CLUB
    WHERE
        LOWER(USER_NICK) = LOWER(USER_LOGON);

ELSE
--E' l'amministratore il chiamante, il nome del club dato in
   input non varia.
    USER_CLUB_NAME := ACTUAL_CLUB_NAME;
END IF;

--Se il nome del club <input_club_name> che e' uguale a <
   actual_club_name> non e' NULL,
--  e non corrisponde al nome del club dell'utente chiamante <
   user_club_name>,
--  e l'utente chiamante non e' amministratore.
--Esempio: user = PLAYER1; il nome del club del PLAYER1 e' '
   club1';
--          <actual_club_name> = 'club2';
--          <user_club_name> = 'club1';
--Il PLAYER1 sta chiamando una funzione con un nome del club
   che non corrisponde al suo, e non puo';
--  solo l'amministratore puo', per questo viene generata un
   eccezione <user_club_error>.
IF
    ACTUAL_CLUB_NAME IS NOT NULL
    AND USER_CLUB_NAME <> ACTUAL_CLUB_NAME
    AND USER_LOGON <> 'ADMINFUT'
THEN
    RAISE USER_CLUB_ERROR;
END IF;

--Ritorno il nome del club.
RETURN USER_CLUB_NAME;
EXCEPTION
WHEN NO_CLUB_FOUND THEN
    RAISE_APPLICATION_ERROR('-20001', 'L''utente '
                                        || USER_LOGON
                                        || ' non ha un club!');
WHEN USER_CLUB_ERROR THEN
    RAISE_APPLICATION_ERROR('-20002', 'Il club '
                                        || INPUT_CLUB_NAME
                                        || ' non appartiene all
                                           ''utente '
                                        || USER_LOGON);
END IS_ADMIN;
/
```

11.4.2 Card in Club

```
-- Tabelle interessate: 4
--      -> CLUB_CARD, IS_FOUND, PACK_PURCHASE, TRANSACTION

-- INPUT:
--      -> c_code: il codice carta;
--      -> c_name: il nome del club che possiede la carta
   identificata dal <c_code>;
-- OUTPUT:
--      -> number: 1 -> se il club possiede la carta,
   altrimenti 0.

CREATE OR REPLACE FUNCTION CARD_IN_CLUB (
C_CODE  CLUB_CARD.CARD_CODE%TYPE,
C_NAME  CLUB.CLUB_NAME%TYPE
) RETURN NUMBER IS
N1 NUMBER(2, 0);    --Valore di ritorno
BEGIN

--Controlla se la carta <c_code> e' stata trovata in un
   pacchetto <pack>.
SELECT
    COUNT(*)
INTO N1
FROM
    CLUB_CARD
WHERE CARD_CODE = C_CODE
    AND CARD_CODE IN (
        SELECT CARD_CODE
        FROM IS_FOUND
        WHERE P_ID IN (
                SELECT PURCHASE_ID
                FROM PACK_PURCHASE
                WHERE BUYING_CLUB_NAME = C_NAME));

--Se n1 = 0, la carta non e' stata trovata in un <pack>;
--Controllo se e' stata comprata.
IF N1 = 0 THEN
    SELECT
        COUNT(CARD_CODE)
    INTO N1
    FROM CLUB_CARD
    WHERE CARD_CODE = C_CODE
        AND CARD_CODE IN (
            SELECT T_CARD_CODE
            FROM TRANSACTION
            WHERE TRANSITION_B_CLUB_NAME = C_NAME);
END IF;
--Ritorno il risultato.
    RETURN N1;
END CARD_IN_CLUB;
/
```

11.4.3 Get Contract

```
-- Tabelle interessate: 4
--      -> CONSUMABLE, CLUB_CARD, IS_PART_OF, PLAYER


-- INPUT:
--      -> c_code: il codice carta della carta di tipo
   consumabile;
--      -> p_code: il codice carta del player o manager alla
   quale applicare il consumabile <c_code>;
--      -> tp:     il tipo <type> del consumabile che puo'
   essere (Player,Manager);
-- OUTPUT:
--      -> number: nuovo valore di contratti <contracts> del
   player <p_code> calcolato dopo
--         aver applicato il consumabile <c_code>;

CREATE OR REPLACE FUNCTION GET_CONTRACT (
C_CODE  IN  CLUB_CARD.CARD_CODE%TYPE,
P_CODE  IN  CLUB_CARD.CARD_CODE%TYPE,
TP      IN  CONSUMABLE.TYPE%TYPE
) RETURN NUMBER IS

N1          NUMBER(2, 0); --Contatore di controllo.
C_TYPE      CONSUMABLE.TYPE%TYPE; --tipo del consumabile.
BRONZE_CTC  CONSUMABLE.BRONZE_CONTRACT%TYPE; --Numero di
   contratti se il <player> e di rarita' <bronze>.
SILVER_CTC  CONSUMABLE.SILVER_CONTRACT%TYPE; --Numero di
   contratti se il <player> e di rarita' <silver>.
GOLD_CTC    CONSUMABLE.GOLD_CONTRACT%TYPE;   --Numero di
   contratti se il <player> e di rarita' <gold>.

P_RARITY    PLAYER.RARITY_NAME%TYPE; --Rarita' del <player>.

INVALID_CLASS EXCEPTION; --Se il tipo in ingresso non
   corrisponde al tipo del consumabile <c_code>.
MAX_CONTRACTS EXCEPTION; --Se il <player> ha 99 contratti che e
   ' il valore massimo.

BEGIN

--Seleziono i dati del consumabile <c_code>.
SELECT
    TYPE,
    BRONZE_CONTRACT,
    SILVER_CONTRACT,
    GOLD_CONTRACT
INTO
    C_TYPE,
    BRONZE_CTC,
    SILVER_CTC,
    GOLD_CTC
FROM
    CONSUMABLE
WHERE
    CARD_ID IN (
```

```
        SELECT
            CONSUMABLE_ID
        FROM
            CLUB_CARD
        WHERE
            CARD_CODE = C_CODE
    );

--Se il tipo <C_TYPE> del consumabile <c_code> e' diverso dal
   tipo <TP> fornito in input.
IF C_TYPE != TP THEN
    RAISE INVALID_CLASS; --Comunica all'utente che il tipo e'
       errato.
END IF;

--Se il tipo <TP> e' uguale a 'Player' e corrisponde a quello
   del cosnumabile.
IF
    TP = 'Player'
    AND C_TYPE = TP
THEN

--Seleziono i contratti del <player>.
    SELECT
        CONTRACTS
    INTO N1
    FROM
        ACTIVE_DATA_PLAYER
    WHERE
        P_CARD_CODE = P_CODE;

--Se il <player> ha 99 contratti, ha gia' il numero massimo e
   lo comunico all'utente.
    IF N1 = 99 THEN
        RAISE MAX_CONTRACTS;
    ELSE --Se il <player> ha meno di 99 contratti.
--Seleziono la rarita' del <player>.
        SELECT
            RARITY_NAME
        INTO P_RARITY
        FROM
            PLAYER
        WHERE
            CARD_ID IN (
                SELECT
                    PLAYER_ID
                FROM
                    CLUB_CARD
                WHERE
                    CARD_CODE = P_CODE
            );

--Se il <player> e' di rarita' <Bronze>.
        IF P_RARITY = 'Bronze - Rare' OR P_RARITY = 'Bronze -
           Non-Rare' THEN
--Controllo che il valore dei contratti del <player> (<n1> + i)
    contratti <bronze_ctc> dati dal consumabile
```

```
-- non superano il massimo numero di contratti (99), se lo
   superano pongo (n1 = 99).
            IF N1 + BRONZE_CTC >
            99 THEN
                N1 := 99;
            ELSE--Altrimenti n1 sara' uguale ad (n1 + i)
               contratti dati dalla carta consumabile.
                N1 := BRONZE_CTC + N1;
            END IF;
--Se il <player> e' di rarita' <Silver>.
        ELSIF P_RARITY = 'Silver - Rare' OR P_RARITY = 'Silver
           - Non-Rare' THEN
            IF N1 + SILVER_CTC > 99 THEN
                N1 := 99;
            ELSE
                N1 := SILVER_CTC + N1;
            END IF;
--Se il <player> e' di rarita' <Gold> o altre rarita'.
        ELSE
            IF N1 + GOLD_CTC > 99 THEN
                N1 := 99;
            ELSE
                N1 := GOLD_CTC + N1;
            END IF;
        END IF;

    END IF;

ELSIF
    TP = 'Manager'
    AND C_TYPE = TP
THEN
--Seleziono i contratti del <manager>.
    SELECT
        CONTRACTS
    INTO N1
    FROM
        ACTIVE_DATA_MANAGER
    WHERE
        M_CARD_CODE = P_CODE;

--Se il <manager> ha 99 contratti.
    IF N1 = 99 THEN
        RAISE MAX_CONTRACTS;
    ELSE --Se il <manager> ha meno di 99 contratti.
--Seleziono la rarita' del <manager>.
        SELECT
            RARITY_NAME
        INTO P_RARITY
        FROM
            MANAGER
        WHERE
            CARD_ID IN (
                SELECT
                    MANAGER_ID
                FROM
                    CLUB_CARD
```

```
                WHERE
                    CARD_CODE = P_CODE
            );

--Se il <manager> e' di rarita' <Bronze>.
        IF P_RARITY = 'Bronze - Rare' OR P_RARITY = 'Bronze -
           Non-Rare' THEN
--Controllo che il valore dei contratti del <manager> (<n1> + i
   ) contratti <bronze_ctc> dati dal consumabile,
-- non superano il massimo numero di contratti (99), se lo
   superano pongo (n1 = 99).
            IF N1 + BRONZE_CTC >
            99 THEN
                N1 := 99;
            ELSE
                N1 := BRONZE_CTC + N1;
            END IF;
--Se il <manager> e' di rarita' <Silver>.
        ELSIF P_RARITY = 'Silver - Rare' OR P_RARITY = 'Silver
           - Non-Rare' THEN
            IF N1 + SILVER_CTC > 99 THEN
                N1 := 99;
            ELSE
                N1 := SILVER_CTC + N1;
            END IF;
--Se il <manager> e' di rarita' <Gold> o altre rarita'.
        ELSE
            IF N1 + GOLD_CTC > 99 THEN
                N1 := 99;
            ELSE
                N1 := GOLD_CTC + N1;
            END IF;
        END IF;

    END IF;

END IF;
--Ritorno il valore dei contratti del player o del manager.
RETURN N1;
EXCEPTION
WHEN INVALID_CLASS THEN
    RAISE_APPLICATION_ERROR('-20003', 'La tipo '
                                        || TP
                                        || ' scelta non e''
                                           valida per la carta
                                           scelta!');
WHEN MAX_CONTRACTS THEN
    RAISE_APPLICATION_ERROR('20004', 'La carta ha il massimo
       dei contratti!');
END GET_CONTRACT;
/
```

11.4.4 Get Player Chemistry

```
-- Function interessate: 1
--      -> IDX_POSITION;

-- INPUT:
--      -> p_position: la posizione <player_position> del <
   player> nella squadra;
--      -> pos: la posizione <position> naturale dal <player>;
-- OUTPUT:
--      -> number: la nuova intesa <chemistry> del <player>.

CREATE OR REPLACE FUNCTION GET_PLAYER_CHEMISTRY (
P_POSITION  PLAYER.POSITION%TYPE,
POS         PLAYER.POSITION%TYPE
) RETURN NUMBER IS
--Data una posizione esempio: 'RB' <terzino destro> il ruolo
   sara' uguale a 2;
--Nello specifico sara':
--      1 = portiere, 2 = difensore, 3 = centrocampista, 4 =
   attaccante.
RUOLO_SCELTO  INTEGER;
RUOLO_PREF    INTEGER;
CHEMISTRY     NUMBER(2, 0);  --Valore di ritorno, intesa del
   giocatore.
M_CHEMISTRY   NUMBER(2, 0);  --Bonus intesa data dal manager.
BEGIN
--IDX_POSITION(pos,'r'):
--input: pos la posizione di cui si vuole ottenere l'indice o
   il ruolo;
--       'r' la funzione ritora il ruolo (1=Porta,2=Difesa,3=
   Centrocampo,4=Attacco).
--       'i' la funzione ritorna l'indice intero della
   posizione.
SELECT
    IDX_POSITION(P_POSITION, 'r'),
    IDX_POSITION(POS, 'r')
INTO
    RUOLO_SCELTO,
    RUOLO_PREF
FROM
    DUAL;

--Calcolo intesa <player_chemistry> del <player>.
IF RUOLO_SCELTO = 1 THEN
--Portiere
--Se il ruolo del <player> corrisponde al ruolo naturale.
    IF P_POSITION = POS
        AND ABS(RUOLO_SCELTO - RUOLO_PREF) = 0
    THEN
        CHEMISTRY := 9;
--Se un <player> non portiere, viene schierato in porta
--l'intesa <chemistry> del <player> sara' = 0.
    ELSIF P_POSITION != POS
        AND ABS(RUOLO_SCELTO - RUOLO_PREF) > 0
    THEN
        CHEMISTRY := 0;
```

```
    END IF;
-- Se non si tratta del portiere.
ELSE
--Se la posizione scelta <p_position> corrisponde alla
   posizione naturale <pos>
--      e il ruolo <ruolo_scelto> corrisponde al ruolo
   preferito <ruolo_pref>
--Esempio:      p_position = 'CB' (Central Back); RUOLO_SCELTO
   = 2 (Difesa);
--              pos = 'CB'; RUOLO_PREF = 2;
    IF P_POSITION = POS
        AND ABS(RUOLO_SCELTO - RUOLO_PREF) = 0
    THEN
        CHEMISTRY := 9;

--Se la posizione scelta <p_position> NON corrisponde alla
   posizione naturale <pos>
--      e il ruolo <ruolo_scelto> corrisponde al ruolo
   preferito <ruolo_pref>
--Esempio:      p_position = 'CB' (Central Back); RUOLO_SCELTO
   = 2 (Difesa);
--              pos = 'RB' (Right Back); RUOLO_PREF = 2;
    ELSIF P_POSITION != POS
        AND ABS(RUOLO_SCELTO - RUOLO_PREF) = 0
    THEN
        CHEMISTRY := 7;

--Se la posizione scelta <p_position> NON corrisponde alla
   posizione naturale <pos>
--      e il ruolo <ruolo_scelto> NON corrisponde al ruolo
   preferito <ruolo_pref>
--Esempio:      p_position = 'CB' (Central Back); RUOLO_SCELTO
   = 2 (Difesa);
--              pos = 'CM' (Central Midfield); RUOLO_PREF = 3 (
   Centrocampo);
    ELSIF P_POSITION != POS
        AND ABS(RUOLO_SCELTO - RUOLO_PREF) = 1
    THEN
        CHEMISTRY := 5;

--Se la posizione scelta <p_position> NON corrisponde alla
   posizione naturale <pos>
--      e il ruolo <ruolo_scelto> NON corrisponde al ruolo
   preferito <ruolo_pref>
--Esempio:      p_position = 'CB' (Central Back); RUOLO_SCELTO
   = 2 (Difesa);
--              pos = 'ST' (Striker); RUOLO_PREF = 4 (Attacco);
    ELSIF P_POSITION != POS
        AND ABS(RUOLO_SCELTO - RUOLO_PREF) > 1
    THEN
        CHEMISTRY := 3;
    END IF;
END IF;

RETURN CHEMISTRY;
END GET_PLAYER_CHEMISTRY;
/
```

11.4.5 Get Squad Rating

```
-- Tabelle interessate: 1
--       -> IS_PART_OF

-- INPUT:
--       -> s_name: il nome della squadra;
--       -> c_name: il nome del club;
-- OUTPUT:
--       -> number: valutazione della squadra.
CREATE OR REPLACE FUNCTION GET_SQUAD_RATING (
S_NAME  IN  SQUAD.NAME%TYPE,
C_NAME  IN  CLUB.CLUB_NAME%TYPE
) RETURN NUMBER IS

OVERALL_SUM    NUMBER(3, 0); --Somma totale della valutaione <
   overall> dei players nella squadra.
AR             NUMBER(5, 2); --Media somma_valutazioni/
   players_totali <overallsum/total_players>.
TOTAL_PLAYERS  NUMBER(2, 0); --Players totali.
CF             NUMBER(5, 2); --Fattore di correzione.
S_RATING       NUMBER(3, 0); --Valutazione della squadra <
   squad_rating>.
BEGIN
CF := 0;
-- <ar>: conterra' la media della valutazione <rating> dei <
   player>.
-- <total_players>: quanti players ci sono in squadra.
-- <overall_sum>: la somma delle valutazioni <rating> dei
   players in squadra.
SELECT
    AVG(PLAYER_RATING),
    COUNT(PLAYER_CARD_CODE),
    SUM(PLAYER_RATING)
INTO AR, TOTAL_PLAYERS, OVERALL_SUM
FROM IS_PART_OF
WHERE SQUAD_NAME = S_NAME;

--Fattore di correzione:
-- <cf> = [(valutazione_1 - ar) + ... + (valutazione_n - ar)/2]
    ,
--  per ogni (valutazione_i > ar).
SELECT ( SUM(PLAYER_RATING) - AR * COUNT(PLAYER_CARD_CODE) )
INTO CF
FROM IS_PART_OF
WHERE SQUAD_NAME = S_NAME
    AND PLAYER_RATING > AR;

CF := CF / 2;
--Calcolo della valutazione <rating> della squadra.
S_RATING := ( OVERALL_SUM + CF ) / TOTAL_PLAYERS;
--Ritorna la valutazione <rating> della squadra.
RETURN S_RATING;
END GET_SQUAD_RATING;
/
```

11.4.6 Idx Positions

```
-- INPUT-OUTPUT:
--      -> pos:     posizione del <player>;
--  -> choice:  se e' uguale ad 'r' la funzione ritorna il
   ruolo del player secondo la posizione del <player>;
--              se e' uguale ad 'i' la funzione ritorna l'
   indice intero della posizione del <player>.
CREATE OR REPLACE FUNCTION IDX_POSITION (
POS     IN  PLAYER.POSITION%TYPE,
CHOICE  IN  CHAR
) RETURN INTEGER IS

RUOLO    INTEGER; --Valore di ritorno nel caso in cui la <
   choice = 'r'>.
IDX_POS  INTEGER; --Valore di ritorno nel caso in cui la <
   choice = 'i'>.
TYPE ARRAY_POS IS VARRAY(17) OF VARCHAR2(3);
POS_ARR  ARRAY_POS; --Array che contiene tutte le posizioni.
INVALID_CHOICE EXCEPTION; --Se la <choice> e' diversa da 'r' o
   'i'.

BEGIN
POS_ARR := ARRAY_POS('GK','RB','CB','LB','RWB','LWB','CDM','CM'
   ,'CAM','RM','LM','RW','LW','CF','RF','LF','ST');
FOR I IN 1..17 LOOP
    IF POS_ARR(I) = POS THEN
        IDX_POS := I;
--ruolo 1 = portiere.
        IF I = 1 THEN
            RUOLO := 1;
--ruolo 2 = difensore.
        ELSIF I > 1 AND I < 7 THEN
            RUOLO := 2;
--ruolo 3 = centrocampista.
        ELSIF I >= 7 AND I < 12 THEN
            RUOLO := 3;
--ruolo 4 = attaccante.
        ELSIF I >= 12 AND I < 18 THEN
            RUOLO := 4;
        END IF;
    END IF;
END LOOP;

IF CHOICE = 'r' THEN
    RETURN RUOLO;
ELSIF CHOICE = 'i' THEN
    RETURN IDX_POS;
ELSE
    RAISE INVALID_CHOICE;
END IF;

EXCEPTION
WHEN INVALID_CHOICE THEN
    RAISE_APPLICATION_ERROR('-20003', 'La scelta (' || CHOICE
       || ') non e'' valida!');
END IDX_POSITION;
```

11.4.7 Manager Chemistry

```
-- Tabelle interessate: 4
--  -> CLUB_CARD, SQUAD, MANAGER, PLAYER;

-- INPUT:
--  -> player_id: il card id del player;
--  -> s_name: il nome della squadra che contiene il player;
--  -> c_name: il nome del club che possiede la squadra <s_name
   >;
-- OUTPUT:
--  -> numero: 1 se il player ha la stessa nazionalita' del
   manager, o la stessa lega <league>,
--             0 altrimenti.
CREATE OR REPLACE FUNCTION MANAGER_CHEMISTRY (
PLAYER_ID  IN  PLAYER.CARD_ID%TYPE,
P_CODE     IN  CLUB_CARD.CARD_CODE%TYPE,
S_NAME     IN  SQUAD.NAME%TYPE,
C_NAME     IN  CLUB.CLUB_NAME%TYPE
) RETURN NUMBER IS

M_CNT        NUMBER(1, 0);      --Se e' > 0 c'e' l'allenatore
   altrimenti no.
M_NATION     VARCHAR2(64);      --Nazionalita' del manager.
M_LEAGUE     VARCHAR2(64);      --Liga del manager.
M_ID         NUMBER(2, 0);      --Id del manager.
P_NATION     VARCHAR2(64);      --Nazionalita' del player.
P_LEAGUE     VARCHAR2(64);      --Lega del player.
P_TEAM       VARCHAR2(64);      --Squadra del player.
M_CHEMISTRY  NUMBER(1, 0);      --Bonus intesa <
   player_chemistry> dato dal manager.

BEGIN
--Controllo che la squadra abbia un manager schierato.
SELECT
    COUNT(MANAGER_ID)
INTO M_CNT
FROM
    CLUB_CARD
WHERE
    CARD_CODE IN (
        SELECT
            MANAGER_CARD_CODE
        FROM
            MANAGES
        WHERE SQUAD_NAME = S_NAME
    );

--Se c'e' il manager.
IF M_CNT > 0 THEN
--Seleziono l'ID del manager.
    SELECT
        MANAGER_ID
    INTO M_ID
    FROM
        CLUB_CARD
    WHERE
```

```
        CARD_CODE IN (
            SELECT
                MANAGER_CARD_CODE
            FROM
                MANAGES
            WHERE SQUAD_NAME = S_NAME
        );

--Seleziono nazionalita e lega del manager.
    SELECT
        NATIONALITY,
        LEAGUE_NAME
    INTO
        M_NATION,
        M_LEAGUE
    FROM
        MANAGER
    WHERE
        CARD_ID = M_ID;

--Seleziono nazionalita' e lega del player.
    SELECT
        NATIONALITY,
        LEAGUE_NAME,
        TEAM_NAME
    INTO
        P_NATION,
        P_LEAGUE,
        P_TEAM
    FROM
        PLAYER
    WHERE
            CARD_ID = PLAYER_ID
        AND CARD_ID IN (
            SELECT
                PLAYER_ID
            FROM
                CLUB_CARD
            WHERE
                CARD_CODE = P_CODE
        );

--Se la nazionalita' del player o la lega corrispondono a
   quelle del manager.
    IF P_NATION = M_NATION OR P_LEAGUE = M_LEAGUE THEN
        M_CHEMISTRY := 1;
    ELSE
        M_CHEMISTRY := 0;
    END IF;
END IF;
    RETURN M_CHEMISTRY;
END MANAGER_CHEMISTRY;
/
```

11.4.8 Module Positions

```
-- Type:
--  -> array_pos_t:  is VARRAY(11) of VARCHAR2(3);

-- INPUT:
--      -> module: modulo della squadra, esempio: '4-4-2';
-- OUTPUT:
--  -> array: array che contiene la posizioni del modulo dato
   in ingresso.

-- ESEMPIO:
--  -> chiamata: select MODULE_POSITIONS('4-3-3') FROM DUAL;
--              output -> ('GK','LB','CB','CB','RB','CM','CM','
   CM','LW','ST','RW');
CREATE OR REPLACE FUNCTION MODULE_POSITIONS (
MODULE IN SQUAD.MODULES%TYPE
) RETURN ARRAY_POS_T IS
ARR ARRAY_POS_T;
MODULE_IN_ERROR EXCEPTION;
BEGIN
IF MODULE = '4-4-2' THEN
    ARR := ARRAY_POS_T('GK', 'LB', 'CB', 'CB', 'RB',
                       'LM', 'CM', 'CM', 'RM', 'ST', 'ST');
ELSIF MODULE = '3-1-4-2' THEN
    ARR := ARRAY_POS_T('GK', 'CB', 'CB', 'CB', 'LM',
                       'CM', 'CDM', 'CM', 'RM', 'ST', 'ST');
ELSIF MODULE = '3-4-1-2' THEN
    ARR := ARRAY_POS_T('GK', 'CB', 'CB', 'CB', 'LM',
                       'CM', 'CM', 'RM', 'CAM', 'ST', 'ST');
ELSIF MODULE = '3-4-3' THEN
    ARR := ARRAY_POS_T('GK', 'CB', 'CB', 'CB', 'LM',
                       'CM', 'CM', 'RM', 'LW', 'ST', 'RW');
ELSIF MODULE = '3-5-2' THEN
    ARR := ARRAY_POS_T('GK', 'CB', 'CB', 'CB', 'LM',
                       'CDM', 'CDM', 'RM', 'CAM', 'ST', 'ST');
ELSIF MODULE = '4-1-2-1-2' THEN
    ARR := ARRAY_POS_T('GK', 'LB', 'CB', 'CB', 'RB',
                       'LM', 'CDM', 'RM', 'CAM', 'ST', 'ST');
ELSIF MODULE = '4-1-3-2' THEN
    ARR := ARRAY_POS_T('GK', 'LB', 'CB', 'CB', 'RB',
                       'CDM', 'LM', 'CM', 'RM', 'ST', 'ST');
ELSIF MODULE = '4-1-4-1' THEN
    ARR := ARRAY_POS_T('GK', 'LB', 'CB', 'CB', 'RB',
                       'CDM', 'LM', 'CM', 'CM', 'RM', 'ST');
ELSIF MODULE = '4-2-2-2' THEN
    ARR := ARRAY_POS_T('GK', 'LB', 'CB', 'CB', 'RB',
                       'CAM', 'CDM', 'CDM', 'CAM', 'ST', 'ST')
                          ;
ELSIF MODULE = '4-2-3-1' THEN
    ARR := ARRAY_POS_T('GK', 'LB', 'CB', 'CB', 'RB',
                       'CDM', 'CAM', 'CDM', 'CAM', 'CAM', 'ST'
                          );
ELSIF MODULE = '4-2-4' THEN
    ARR := ARRAY_POS_T('GK', 'LB', 'CB', 'CB', 'RB',
                       'CM', 'CM', 'LW', 'RW', 'ST', 'ST');
ELSIF MODULE = '4-3-1-2' THEN
```

```
    ARR := ARRAY_POS_T('GK', 'LB', 'CB', 'CB', 'RB',
                       'CM', 'CM', 'CM', 'CAM', 'ST', 'ST');
ELSIF MODULE = '4-3-2-1' THEN
    ARR := ARRAY_POS_T('GK', 'LB', 'CB', 'CB', 'RB',
                       'CM', 'CM', 'CM', 'LF', 'ST', 'RF');
ELSIF MODULE = '4-3-3' THEN
    ARR := ARRAY_POS_T('GK', 'LB', 'CB', 'CB', 'RB',
                       'CM', 'CM', 'CM', 'LW', 'ST', 'RW');
ELSIF MODULE = '4-4-1-1' THEN
    ARR := ARRAY_POS_T('GK', 'LB', 'CB', 'CB', 'RB',
                       'LM', 'CM', 'CM', 'RM', 'CF', 'ST');
ELSIF MODULE = '4-5-1' THEN
    ARR := ARRAY_POS_T('GK', 'LB', 'CB', 'CB', 'RB',
                       'LM', 'CM', 'RM', 'CAM', 'CAM', 'ST');
ELSIF MODULE = '5-2-1-2' THEN
    ARR := ARRAY_POS_T('GK', 'LWB', 'CB', 'CB', 'CB',
                       'RWB', 'CM', 'CM', 'CAM', 'ST', 'ST');
ELSIF MODULE = '5-2-2-1' THEN
    ARR := ARRAY_POS_T('GK', 'LWB', 'CB', 'CB', 'CB',
                       'RWB', 'CM', 'CM', 'LW', 'ST', 'RW');
ELSIF MODULE = '5-3-2' THEN
    ARR := ARRAY_POS_T('GK', 'LWB', 'CB', 'CB', 'CB',
                       'RWB', 'CM', 'CM', 'CM', 'ST', 'ST');
ELSIF MODULE = '5-4-1' THEN
    ARR := ARRAY_POS_T('GK', 'LWB', 'CB', 'CB', 'CB',
                       'RWB', 'LM', 'CM', 'CM', 'RM', 'ST');
ELSIF MODULE = '5-2-1-2' THEN
    ARR := ARRAY_POS_T('GK', 'LWB', 'CB', 'CB', 'CB',
                       'RWB', 'CM', 'CM', 'CAM', 'ST', 'ST');
ELSE
    RAISE MODULE_IN_ERROR;
END IF;

RETURN ARR;
EXCEPTION
WHEN MODULE_IN_ERROR THEN
    RAISE_APPLICATION_ERROR(-20005, 'Modules ('
                                    || MODULE
                                    || ') not found!');
END MODULE_POSITIONS;
/
```

11.4.9 Get Active Club Item F

```
-- Tabelle interessate: 3
--  -> CLUB_ITEM, CLUB_CARD, CLUB;

-- INPUT:
--      -> c_name:  nome del club;
-- OUTPUT:
--  -> Dettagli dei club item attivi del club <c_name>.
CREATE OR REPLACE FUNCTION GET_ACTIVE_CLUB_ITEM_F (
    C_NAME CLUB.CLUB_NAME%TYPE
) RETURN SYS_REFCURSOR IS
    G_C_INFO SYS_REFCURSOR;
BEGIN
    OPEN G_C_INFO FOR
--Seleziona i dettagli che riguardano lo stemma del club.
        SELECT
            CARD_ID,
            CATEGORY_ITEM,
            TEAM_NAME
        FROM
                CLUB_ITEM C
            JOIN CLUB_CARD CC ON C.CARD_ID = CC.CLUB_ITEM_ID
                                 AND CC.CARD_CODE IN (
                SELECT
                    ACTIVE_BADGE_CODE
                FROM CLUB
                WHERE CLUB_NAME = C_NAME)
        UNION
    --Seleziona i dettagli che riguardano lo stadio del club.
        SELECT
            CARD_ID,
            CATEGORY_ITEM,
            TEAM_NAME
        FROM
                CLUB_ITEM C
            JOIN CLUB_CARD CC ON C.CARD_ID = CC.CLUB_ITEM_ID
                                 AND CC.CARD_CODE IN (
                SELECT
                    ACTIVE_STADIUM_CODE
                FROM CLUB
                WHERE CLUB_NAME = C_NAME)
        UNION
    --Seleziona i dettagli che riguardano il pallone del club.
        SELECT
            CARD_ID,
            CATEGORY_ITEM,
            TEAM_NAME
        FROM
                CLUB_ITEM C
            JOIN CLUB_CARD CC ON C.CARD_ID = CC.CLUB_ITEM_ID
                                 AND CC.CARD_CODE IN (
                SELECT
                    ACTIVE_BALL_CODE
                FROM CLUB
                WHERE CLUB_NAME = C_NAME)
        UNION
```

```
    --Seleziona i dettagli che riguardano la divisa in casa del
        club.
        SELECT
            CARD_ID,
            CATEGORY_ITEM,
            TEAM_NAME
        FROM
                CLUB_ITEM C
            JOIN CLUB_CARD CC ON C.CARD_ID = CC.CLUB_ITEM_ID
                                 AND CC.CARD_CODE IN (
                SELECT
                    ACTIVE_FIRST_KIT_CODE
                FROM CLUB
                WHERE CLUB_NAME = C_NAME)
        UNION
    --Seleziona i dettagli che riguardano la divisa fuori casa
       del club.
        SELECT
            CARD_ID,
            CATEGORY_ITEM,
            TEAM_NAME
        FROM
                CLUB_ITEM C
            JOIN CLUB_CARD CC ON C.CARD_ID = CC.CLUB_ITEM_ID
                                 AND CC.CARD_CODE IN (
                SELECT
                    ACTIVE_SECOND_KIT_CODE
                FROM CLUB
                WHERE CLUB_NAME = C_NAME);

    RETURN G_C_INFO;
END GET_ACTIVE_CLUB_ITEM_F;
/
```

11.4.10 Get Club Item F

```
-- Tabelle interessate: 3
--  -> CLUB_ITEM, CLUB_CARD, IF_FOUND, TRANSACTION;

-- INPUT:
--      -> c_name:  nome del club;
-- OUTPUT:
--  -> Dettagli dei club_item del club fornito in ingresso.
CREATE OR REPLACE FUNCTION GET_CLUB_ITEM_F (
    USR_CLUB_NAME CLUB.CLUB_NAME%TYPE
) RETURN SYS_REFCURSOR IS
    G_C_ITEM SYS_REFCURSOR;
BEGIN
    OPEN G_C_ITEM FOR
        SELECT
            CARD_ID,
            CARD_CODE,
            CATEGORY_ITEM,
            TEAM_NAME,
            RARITY_NAME,
            MIN_PRICE,
            MAX_PRICE
        FROM
                CLUB_ITEM C
            JOIN CLUB_CARD CC ON C.CARD_ID = CC.CLUB_ITEM_ID
                                 AND CC.CARD_CODE IN (
                SELECT
                    CARD_CODE
                FROM IS_FOUND
                WHERE P_ID IN (
                        SELECT PURCHASE_ID
                        FROM PACK_PURCHASE
                        WHERE BUYING_CLUB_NAME = USR_CLUB_NAME)
                    )
        UNION
        SELECT
            CARD_ID,
            CARD_CODE,
            CARD_ID,
            CATEGORY_ITEM,
            TEAM_NAME,
            RARITY_NAME,
            MIN_PRICE,
            MAX_PRICE
        FROM
                CLUB_ITEM C
            JOIN CLUB_CARD CC ON C.CARD_ID = CC.CLUB_ITEM_ID
                                 AND CC.CARD_CODE IN (
                SELECT
                    T_CARD_CODE
                FROM TRANSACTION
                WHERE TRANSITION_B_CLUB_NAME = USR_CLUB_NAME);

    RETURN G_C_ITEM;
END GET_CLUB_ITEM_F;
/
```

11.4.11 Get Consumables F

```
-- Tabelle interessate: 5
--  -> CLUB_CARD, CONSUMABLE, IS_FOUND, PACK_PURCHASE,
   TRANSACTION;

-- INPUT:
--      -> c_name:  nome del club;
--  -> c_category: categoria della carta di tipo consumabile;
--                 ('Contract','Fitness','Training','Healing
   ','GKTraining','Manager League','Positioning').
-- OUTPUT:
--  -> Dettagli dei consumabili in base alla categoria <
   c_category>, posseduti dal club fornito in ingresso.
CREATE OR REPLACE FUNCTION GET_CONSUMABLES_F (
    C_NAME      SQUAD.SQUAD_CLUB_NAME%TYPE,
    C_CATEGORY  CONSUMABLE.CATEGORY%TYPE
) RETURN SYS_REFCURSOR IS
    G_CONSUMABLES SYS_REFCURSOR;
BEGIN
    OPEN G_CONSUMABLES FOR
        SELECT
            CARD_CODE,
            CARD_ID,
            RARITY_NAME,
            TYPE,
            CATEGORY,
            AMOUNT,
            BRONZE_CONTRACT,
            SILVER_CONTRACT,
            GOLD_CONTRACT,
            MIN_PRICE,
            MAX_PRICE
        FROM
                CONSUMABLE C
            JOIN CLUB_CARD CC ON C.CARD_ID = CC.CONSUMABLE_ID
                                 AND C.CATEGORY = C_CATEGORY
                                 AND CC.CARD_CODE IN (
                SELECT
                    CARD_CODE
                FROM IS_FOUND
                WHERE P_ID IN (
                        SELECT PURCHASE_ID
                        FROM PACK_PURCHASE
                        WHERE BUYING_CLUB_NAME = C_NAME))
        UNION
        SELECT
            CARD_CODE,
            CARD_ID,
            RARITY_NAME,
            TYPE,
            CATEGORY,
            AMOUNT,
            BRONZE_CONTRACT,
            SILVER_CONTRACT,
            GOLD_CONTRACT,
            MIN_PRICE,
```

```
            MAX_PRICE
        FROM
                CONSUMABLE C
            JOIN CLUB_CARD CC ON C.CARD_ID = CC.CONSUMABLE_ID
                                AND C.CATEGORY = C_CATEGORY
                                AND CC.CARD_CODE IN (
                SELECT
                    T_CARD_CODE
                FROM TRANSACTION
                WHERE TRANSITION_B_CLUB_NAME = C_NAME);

    RETURN G_CONSUMABLES;
END GET_CONSUMABLE_F;
/
```

11.4.12 Get Manager F

```
-- Tabelle interessate: 5
--  -> CLUB_CARD, MANAGER, IS_FOUND, PACK_PURCHASE, TRANSACTION
   ;

-- INPUT:
--      -> c_name:  nome del club;
-- OUTPUT:
--  -> Dettagli dei manager posseduti dal club fornito in
   ingresso.
CREATE OR REPLACE FUNCTION GET_MANAGER_F (
    C_NAME CLUB.CLUB_NAME%TYPE
) RETURN SYS_REFCURSOR IS
    G_MANAGER SYS_REFCURSOR;
BEGIN
    OPEN G_MANAGER FOR
        SELECT
            CARD_CODE,
            CARD_ID,
            F_NAME,
            S_NAME,
            NATIONALITY,
            LEAGUE_NAME AS REAL_LEAGUE,
            MANAGER_LEAGUE AS ACTUAL_LEAGUE,
            CONTRACTS,
            MIN_PRICE,
            MAX_PRICE
        FROM
                MANAGER M
            JOIN CLUB_CARD C ON M.CARD_ID = C.MANAGER_ID
            JOIN ACTIVE_DATA_MANAGER A ON A.M_CARD_CODE = C.
               CARD_CODE
                                AND C.CARD_CODE IN (
                SELECT
                    CARD_CODE
                FROM IS_FOUND
                WHERE P_ID IN (
                        SELECT PURCHASE_ID
                        FROM PACK_PURCHASE
                        WHERE BUYING_CLUB_NAME = C_NAME))
        UNION
        SELECT
            CARD_CODE,
            CARD_ID,
            F_NAME,
            S_NAME,
            NATIONALITY,
            LEAGUE_NAME AS REAL_LEAGUE,
            MANAGER_LEAGUE AS ACTUAL_LEAGUE,
            MIN_PRICE,
            MAX_PRICE
        FROM
                MANAGER M
            JOIN CLUB_CARD C ON M.CARD_ID = C.MANAGER_ID
            JOIN ACTIVE_DATA_MANAGER A ON A.M_CARD_CODE = C.
               CARD_CODE
```

```
                                AND C.CARD_CODE IN (
                SELECT
                    T_CARD_CODE
                FROM TRANSACTION
                WHERE TRANSITION_B_CLUB_NAME = C_NAME);
    RETURN G_MANAGER;
END GET_MANAGER_F;
/
```

11.4.13 Get Manages F

```
-- Tabelle interessate: 8
--  -> CLUB_CARD, MANAGER, ACTIVE_DATA_MANAGER, MANAGES,
   IS_FOUND, PACK_PURCHSE, CLUB, TRANSACTION;

-- OUTPUT:
--  -> Dettagli dei dati attivi delle carte manager con
   relativa squadra.
CREATE OR REPLACE FUNCTION GET_MANAGES_F (
    C_NAME CLUB.CLUB_NAME%TYPE
) RETURN SYS_REFCURSOR IS
    G_MANAGES SYS_REFCURSOR;
BEGIN
    OPEN G_MANAGES FOR
        SELECT
            CARD_CODE,
            F_NAME,
            S_NAME,
            CONTRACTS,
            LEAGUE_NAME      AS PREF_LEAGUE,
            MANAGER_LEAGUE   AS ACTIVE_LEAGUE,
            NATIONALITY,
            SQUAD_NAME       AS SQUAD_MANAGES,
        FROM
                MANAGER M
            JOIN CLUB_CARD             C ON M.CARD_ID = C.
               MANAGER_ID
            JOIN ACTIVE_DATA_MANAGER   A ON A.M_CARD_CODE = C.
               CARD_CODE
            JOIN MANAGES               MM ON MM.
               MANAGER_CARD_CODE = A.M_CARD_CODE
                           AND C.CARD_CODE IN (
                SELECT
                    CARD_CODE
                FROM IS_FOUND
                WHERE P_ID IN (
                        SELECT PURCHASE_ID
                        FROM PACK_PURCHASE
                        WHERE BUYING_CLUB_NAME = C_NAME))
        UNION
        SELECT
            CARD_CODE,
            F_NAME,
            S_NAME,
            CONTRACTS,
            LEAGUE_NAME      AS PREF_LEAGUE,
            MANAGER_LEAGUE   AS ACTIVE_LEAGUE,
            NATIONALITY,
            SQUAD_NAME       AS SQUAD_MANAGES,
        FROM
                MANAGER M
            JOIN CLUB_CARD             C ON M.CARD_ID = C.
               MANAGER_ID
            JOIN ACTIVE_DATA_MANAGER   A ON A.M_CARD_CODE = C.
               CARD_CODE
            JOIN MANAGES               MM ON MM.
```

```
                MANAGER_CARD_CODE = A.M_CARD_CODE
                            AND C.CARD_CODE IN (
                 SELECT
                     T_CARD_CODE
                 FROM TRANSACTION
                 WHERE TRANSITION_B_CLUB_NAME = C_NAME);
    RETURN G_MANAGES;
END GET_MANAGES_F;
/
```

11.4.14 Get Pack Purchase F

```
-- Tabelle interessate: 2
--  -> PACK_PURCHASE, PACK;

-- INPUT:
--      -> c_name:  nome del club;
-- OUTPUT:
--  -> Cronologia dei pacchetti acquistati dal club fornito in
   ingresso.
CREATE OR REPLACE FUNCTION GET_PACK_PURCHASE_F (
    C_NAME CLUB.CLUB_NAME%TYPE
) RETURN SYS_REFCURSOR IS
    G_PACK_PURCHASE SYS_REFCURSOR;
BEGIN
    OPEN G_PACK_PURCHASE FOR
        SELECT
            PURCHASE_ID,
            BUYING_PACK_NAME,
            P_DATE,
            PRICE
        FROM
                PACK_PURCHASE P
            JOIN PACK PP ON P.BUYING_PACK_NAME = PP.PACK_NAME
                            AND P.BUYING_CLUB_NAME = C_NAME;

    RETURN G_PACK_PURCHASE;
END GET_PACK_PURCHASE_F;
/
```

11.4.15 Get Match F

```
-- Tabelle interessate: 5
--  -> MATCH, SQUAD;

-- INPUT:
--      -> c_name:  nome del club;
-- OUTPUT:
--  -> Dettagli dei match disputati dalle squadre del club
   fornito in ingresso.
CREATE OR REPLACE FUNCTION GET_MATCH_F (
    C_NAME CLUB.CLUB_NAME%TYPE
) RETURN SYS_REFCURSOR IS
    G_MATCH SYS_REFCURSOR;
BEGIN
    OPEN G_MATCH FOR
        SELECT
            M_DATE,
            HOME_SQUAD_NAME,
            VISITORS_SQUAD_NAME,
            RESULTS,
            HOME_POINTS   AS POINTS,
            HOME_CREDITS  AS CREDITS
        FROM
                MATCH M
            JOIN SQUAD S ON M.HOME_SQUAD_NAME = S.NAME
                            AND S.SQUAD_CLUB_NAME = C_NAME
        UNION
        SELECT
            M_DATE,
            HOME_SQUAD_NAME,
            VISITORS_SQUAD_NAME,
            RESULTS,
            VISITORS_POINTS,
            VISITORS_CREDITS
        FROM
                MATCH M
            JOIN SQUAD S ON M.VISITORS_SQUAD_NAME = S.NAME
                            AND S.SQUAD_CLUB_NAME = C_NAME;

    RETURN G_MATCH;
END GET_MATCH_F;
/
```

11.4.16 Get Players F

```
-- Tabelle interessate: 5
--  -> CLUB_CARD, PLAYER, IS_FOUND, PACK_PURCHASE, TRANSACTION;

-- INPUT:
--      -> c_name:  nome del club;
-- OUTPUT:
--  -> Dettagli dei player posseduti dal club fornito in
   ingresso.
CREATE OR REPLACE FUNCTION GET_PLAYERS_F (
    C_NAME SQUAD.SQUAD_CLUB_NAME%TYPE
) RETURN SYS_REFCURSOR IS
    VIEWPLAYER SYS_REFCURSOR;
BEGIN
    OPEN VIEWPLAYER FOR
        SELECT
            CARD_CODE,
            CARD_ID,
            PLAYER_NAME,
            POSITION,
            NATIONALITY,
            LEAGUE_NAME
        FROM
                    PLAYER P
            JOIN CLUB_CARD C ON P.CARD_ID = C.PLAYER_ID
                                AND C.CARD_CODE IN (
                SELECT
                    CARD_CODE
                FROM IS_FOUND
                WHERE P_ID IN (
                        SELECT PURCHASE_ID
                        FROM PACK_PURCHASE
                        WHERE BUYING_CLUB_NAME = C_NAME))
        UNION
        SELECT
            CARD_CODE,
            CARD_ID,
            PLAYER_NAME,
            POSITION,
            NATIONALITY,
            LEAGUE_NAME
        FROM
                    PLAYER P
            JOIN CLUB_CARD C ON P.CARD_ID = C.PLAYER_ID
                                AND C.CARD_CODE IN (
                SELECT
                    T_CARD_CODE
                FROM TRANSACTION
                WHERE TRANSITION_B_CLUB_NAME = C_NAME);

    RETURN VIEWPLAYER;
END GET_PLAYERS_F;
/
```

11.4.17 Get Players Stops F

```
-- Tabelle interessate: 7
--  -> CLUB_CARD, STOPS, PLAYER, IS_PART_OF, IS_FOUND,
   PACK_PURCHASE, TRANSACTION;

-- INPUT:
--      -> c_name:  nome del club;
-- OUTPUT:
--  -> Dettagli dei player infortunati del club fornito in
   ingresso.
CREATE OR REPLACE FUNCTION GET_PLAYERS_STOPS_F (
    C_NAME CLUB.CLUB_NAME%TYPE
) RETURN SYS_REFCURSOR IS
    G_P_STOPS SYS_REFCURSOR;
BEGIN
    OPEN G_P_STOPS FOR
        SELECT
            P_CARD_CODE      AS CARD_CODE,
            PLAYER_NAME      AS NAME,
            PLAYER_POSITION  AS POSITION,
            HOLDER,
            S_DATE,
            BODY_PART        AS PART,
            DAYS
        FROM STOPS S
            JOIN CLUB_CARD   C ON S.P_CARD_CODE = C.CARD_CODE
            JOIN PLAYER      P ON C.PLAYER_ID = P.CARD_ID
            JOIN IS_PART_OF  I ON I.PLAYER_CARD_CODE = C.
               CARD_CODE
                                  AND C.CARD_CODE IN (
                SELECT CARD_CODE
                FROM IS_FOUND
                WHERE P_ID IN (
                        SELECT PURCHASE_ID
                        FROM PACK_PURCHASE
                        WHERE BUYING_CLUB_NAME = C_NAME))
        UNION
        SELECT
            P_CARD_CODE      AS CARD_CODE,
            PLAYER_NAME      AS NAME,
            PLAYER_POSITION  AS POSITION,
            HOLDER,
            S_DATE,
            BODY_PART        AS PART,
            DAYS
        FROM STOPS S
            JOIN CLUB_CARD   C ON S.P_CARD_CODE = C.CARD_CODE
            JOIN PLAYER      P ON C.PLAYER_ID = P.CARD_ID
            JOIN IS_PART_OF  I ON I.PLAYER_CARD_CODE = C.
               CARD_CODE
                                  AND C.CARD_CODE IN (
                SELECT T_CARD_CODE
                FROM TRANSACTION
                WHERE TRANSITION_B_CLUB_NAME = C_NAME);
    RETURN G_P_STOPS;
END GET_PLAYERS_STOPS_F;
```

11.5 Triggers

11.5.1 Email to MD5

```
-- Tabelle interessate: 1
--  -> USER_FUT;

-- Scopo:
--  -> Cripta l'email di un nuovo utente con l'algoritmo MD5.

CREATE OR REPLACE TRIGGER EMAIL2MD5 BEFORE
INSERT ON USER_FUT
FOR EACH ROW
DECLARE
MD5RAW     RAW(2000);
OUT_EMAIL  RAW(16); --Email criptata.

BEGIN
MD5RAW := UTL_RAW.CAST_TO_RAW(:NEW.EMAIL);
DBMS_OBFUSCATION_TOOLKIT.MD5(INPUT => MD5RAW, CHECKSUM =>
   OUT_EMAIL);

--Inserisco la nuova email criptata.
SELECT
    OUT_EMAIL
INTO :NEW.EMAIL
FROM
    DUAL;

END EMAIL2MD5;
/
```

11.5.2 Insert user

```
-- Scopo: Crea nel db l''utente che e'' stato appena inserito
   nel gioco, e gli concede i privilegi.
CREATE OR REPLACE TRIGGER INSERT_USER AFTER
    INSERT ON USER_FUT
    FOR EACH ROW
DECLARE
    PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN

EXECUTE IMMEDIATE 'CREATE USER ' || :NEW.NICKNAME || '
   IDENTIFIED BY ' || :NEW.NICKNAME || '';
EXECUTE IMMEDIATE 'GRANT CONNECT, CREATE SESSION TO ' || :NEW.
   NICKNAME || '';

--View
EXECUTE IMMEDIATE 'GRANT SELECT ON GET_ACTIVE_CLUB_ITEMS   TO '
    || :NEW.NICKNAME || '';
EXECUTE IMMEDIATE 'GRANT SELECT ON GET_AVAILABLE_PACKS     TO '
    || :NEW.NICKNAME || '';
EXECUTE IMMEDIATE 'GRANT SELECT ON GET_CLUB_ITEMS          TO '
    || :NEW.NICKNAME || '';
EXECUTE IMMEDIATE 'GRANT SELECT ON GET_CONSUMABLES         TO '
    || :NEW.NICKNAME || '';
EXECUTE IMMEDIATE 'GRANT SELECT ON GET_MANAGERS            TO '
    || :NEW.NICKNAME || '';
EXECUTE IMMEDIATE 'GRANT SELECT ON GET_MANAGES             TO '
    || :NEW.NICKNAME || '';
EXECUTE IMMEDIATE 'GRANT SELECT ON GET_MATCH               TO '
    || :NEW.NICKNAME || '';
EXECUTE IMMEDIATE 'GRANT SELECT ON GET_PACK_PURCHASES      TO '
    || :NEW.NICKNAME || '';
EXECUTE IMMEDIATE 'GRANT SELECT ON GET_PLAYERS_FOR_SALE    TO '
    || :NEW.NICKNAME || '';
EXECUTE IMMEDIATE 'GRANT SELECT ON GET_PLAYERS             TO '
    || :NEW.NICKNAME || '';
EXECUTE IMMEDIATE 'GRANT SELECT ON GET_CLUB_INFO           TO '
    || :NEW.NICKNAME || '';
EXECUTE IMMEDIATE 'GRANT SELECT ON GET_PLAYERS_STOPS       TO '
    || :NEW.NICKNAME || '';

--Function
EXECUTE IMMEDIATE 'GRANT EXECUTE ON GET_SQUAD              TO '
    || :NEW.NICKNAME || '';
EXECUTE IMMEDIATE 'GRANT EXECUTE ON SEARCH_CARD_FOR_SALE   TO '
    || :NEW.NICKNAME || '';

--Procedure
EXECUTE IMMEDIATE 'GRANT EXECUTE ON ADD_CLUB_ITEM          TO '
    || :NEW.NICKNAME || '';
EXECUTE IMMEDIATE 'GRANT EXECUTE ON ADD_CONTRACT           TO '
    || :NEW.NICKNAME || '';
EXECUTE IMMEDIATE 'GRANT EXECUTE ON ADD_FITNESS            TO '
    || :NEW.NICKNAME || '';
EXECUTE IMMEDIATE 'GRANT EXECUTE ON ADD_HEALING            TO '
    || :NEW.NICKNAME || '';
```

```
EXECUTE IMMEDIATE 'GRANT EXECUTE ON ADD_MANAGER            TO '
    || :NEW.NICKNAME || '';
EXECUTE IMMEDIATE 'GRANT EXECUTE ON ADD_MANAGER_LEAGUE     TO '
    || :NEW.NICKNAME || '';
EXECUTE IMMEDIATE 'GRANT EXECUTE ON ADD_PLAYER             TO '
    || :NEW.NICKNAME || '';
EXECUTE IMMEDIATE 'GRANT EXECUTE ON ADD_POSITIONING        TO '
    || :NEW.NICKNAME || '';
EXECUTE IMMEDIATE 'GRANT EXECUTE ON ADD_TRAINING           TO '
    || :NEW.NICKNAME || '';
EXECUTE IMMEDIATE 'GRANT EXECUTE ON BUY_CARD               TO '
    || :NEW.NICKNAME || '';
EXECUTE IMMEDIATE 'GRANT EXECUTE ON CLUB_DELETE            TO '
    || :NEW.NICKNAME || '';
EXECUTE IMMEDIATE 'GRANT EXECUTE ON INSERT_MATCH           TO '
    || :NEW.NICKNAME || '';
EXECUTE IMMEDIATE 'GRANT EXECUTE ON PACK_OPENING           TO '
    || :NEW.NICKNAME || '';
EXECUTE IMMEDIATE 'GRANT EXECUTE ON REMOVE_MANAGER         TO '
    || :NEW.NICKNAME || '';
EXECUTE IMMEDIATE 'GRANT EXECUTE ON SELL_CARD              TO '
    || :NEW.NICKNAME || '';
EXECUTE IMMEDIATE 'GRANT EXECUTE ON SQUAD_CREATION         TO '
    || :NEW.NICKNAME || '';
EXECUTE IMMEDIATE 'GRANT EXECUTE ON CHANGE_PLAYER          TO '
    || :NEW.NICKNAME || '';

END INSERT_USER;
/
```

11.5.3 Check Player Fitness

```
-- Trigger CHECK_PLAYER_FITNESS

-- Tabelle interessate: 4
--  -> CLUB_CARD, PLAYER, IS_PART_OF, ACTIVE_DATA_PLAYER;
-- Scopo:
--  -> Controlla la forma fisica dei player di una squadra che
   sta per disputare un match,
--      se e' minore di 10, notifica all'utente i player
   stanchi.
CREATE OR REPLACE TRIGGER CHECK_PLAYER_FITNESS BEFORE
INSERT OR UPDATE ON MATCH
FOR EACH ROW
DECLARE
--Contatori di controllo.
N1              NUMBER(2, 0);
N2              NUMBER(2, 0);
FLAG            NUMBER(1, 0) := 0;
P_CARD_CODE     CLUB_CARD.CARD_CODE%TYPE; --Codice carta.
P_NAME          PLAYER.PLAYER_NAME%TYPE; --Nome del player.
TYPE ARRAY_CC_T IS VARRAY(18) OF VARCHAR2(8);
ARR_CARD_CODE   ARRAY_CC_T; --Array codici carta.

BEGIN
--Quando si sta inserendo un nuovo match, cioe' un squdra si
   mette in attesa di un avversario per disputare il match.
IF INSERTING THEN
--Quanti player sono stanchi della squadra, che e' in attesa,
   in casa.
    SELECT
        COUNT(*)
    INTO N1
    FROM IS_PART_OF I
        JOIN ACTIVE_DATA_PLAYER A
            ON I.PLAYER_CARD_CODE = A.P_CARD_CODE
                AND I.SQUAD_NAME = :NEW.HOME_SQUAD_NAME
                AND :NEW.VISITORS_SQUAD_NAME IS NULL
                AND A.PLAYER_FITNESS < 10;
--Se <n1 = 0>, o non ci sono player stanchi, oppure non c'e'
   una squadra in attesa in casa, controllo la squadra fuori
   casa.
    IF N1 = 0 THEN
--Quanti player sono stanchi della squadra, che e' in attesa,
   fuori casa.
        SELECT
            COUNT(*)
        INTO N2
        FROM IS_PART_OF I
            JOIN ACTIVE_DATA_PLAYER A
                ON I.PLAYER_CARD_CODE = A.P_CARD_CODE
                    AND I.SQUAD_NAME = :NEW.VISITORS_SQUAD_NAME
                    AND :NEW.HOME_SQUAD_NAME IS NULL
                    AND A.PLAYER_FITNESS < 10;
    END IF;
--Se <n2 = 0>, non ci sono player stanchi e il trigger termina.
    IF N2 = 0 THEN
```

```
        RETURN;
    ELSE
--Se <n1 != 0>, ci sono player stanchi della squadra in casa.
        IF N1 <> 0 THEN
--Seleziono i codici carta dei player.
            SELECT
                PLAYER_CARD_CODE
            BULK COLLECT
            INTO ARR_CARD_CODE
            FROM IS_PART_OF I
                JOIN ACTIVE_DATA_PLAYER A
                    ON I.PLAYER_CARD_CODE = A.P_CARD_CODE
                        AND I.SQUAD_NAME = :NEW.HOME_SQUAD_NAME
                        AND A.PLAYER_FITNESS < 10;
--Ci sono player stanchi nella squadra fuori casa <n2 != 0>.
--Ricordo che siamo in 'INSERTING' significa che c'e' una sola
   squadra (in casa o fuori casa) in attesa del avversario.
        ELSIF
            N1 = 0
            AND N2 <> 0
        THEN
            SELECT
                PLAYER_CARD_CODE
            BULK COLLECT
            INTO ARR_CARD_CODE
            FROM IS_PART_OF I
                JOIN ACTIVE_DATA_PLAYER A
                    ON I.PLAYER_CARD_CODE = A.P_CARD_CODE
                        AND I.SQUAD_NAME = :NEW.
                            VISITORS_SQUAD_NAME
                        AND A.PLAYER_FITNESS < 10;
        END IF;

--Notifico al utente quali player sono stanchi con relativo
   codice carta, in modo da facilitare l'eventuale attivazione
   di una carta di tipo consumabile.
        DBMS_OUTPUT.PUT_LINE('ATTENZIONE! I seguenti player
           hanno forma fisica al minimo!' || CHR(10));
        FOR I IN 1..ARR_CARD_CODE.COUNT LOOP
            SELECT PLAYER_NAME
            INTO P_NAME
            FROM PLAYER
            WHERE CARD_ID IN (
                    SELECT PLAYER_ID
                    FROM CLUB_CARD
                    WHERE CARD_CODE = ARR_CARD_CODE(I)
                );

            DBMS_OUTPUT.PUT_LINE('['
                                    || I
                                    || '] '
                                    || P_NAME
                                    || ' ('
                                    || ARR_CARD_CODE(I)
                                    || ')'
                                    || CHR(10));
```

```
        END LOOP;
    END IF;
--Se stiamo aggiornando un match 'UPDATING', ci troviamo dalla
   parte dell'avversario.
ELSIF UPDATING THEN
--Se prima la squadra in casa era NULL, significa che l'ultima
   squadra inserita giochera' in casa.
    IF :OLD.HOME_SQUAD_NAME IS NULL THEN
--Verifico quanti player sono stanchi della squadra in casa.
        SELECT
            COUNT(*)
        INTO N1
        FROM IS_PART_OF I
            JOIN ACTIVE_DATA_PLAYER A
                ON I.PLAYER_CARD_CODE = A.P_CARD_CODE
                    AND I.SQUAD_NAME = :NEW.HOME_SQUAD_NAME
                    AND A.PLAYER_FITNESS < 10;

--Se <n1 = 0>, non ci sono player stanchi.
        IF N1 = 0 THEN
            RETURN;
        ELSE
--Seleziono i codici carta dei player della squadra.
            SELECT
                PLAYER_CARD_CODE
            BULK COLLECT
            INTO ARR_CARD_CODE
            FROM IS_PART_OF I
                JOIN ACTIVE_DATA_PLAYER A
                    ON I.PLAYER_CARD_CODE = A.P_CARD_CODE
                        AND I.SQUAD_NAME = :NEW.HOME_SQUAD_NAME
                        AND A.PLAYER_FITNESS < 10;
            FLAG := 1; --Notifica al utente i player stanchi.
        END IF;

    ELSIF :OLD.VISITORS_SQUAD_NAME IS NULL THEN
--Quanti player sono stanchi nella squadra fuori casa.
        SELECT
            COUNT(*)
        INTO N1
        FROM IS_PART_OF I
            JOIN ACTIVE_DATA_PLAYER A
                ON I.PLAYER_CARD_CODE = A.P_CARD_CODE
                    AND I.SQUAD_NAME = :NEW.VISITORS_SQUAD_NAME
                    AND A.PLAYER_FITNESS < 10;

--Se <n1 = 0>, non ci sono player stanchi.
        IF N1 = 0 THEN
            RETURN;
        ELSE
--Seleziono i codici carta dei player stanchi.
            SELECT
                PLAYER_CARD_CODE
            BULK COLLECT
            INTO ARR_CARD_CODE
            FROM IS_PART_OF I
                JOIN ACTIVE_DATA_PLAYER A
```

```
                    ON I.PLAYER_CARD_CODE = A.P_CARD_CODE
                        AND I.SQUAD_NAME = :NEW.
                            VISITORS_SQUAD_NAME
                        AND A.PLAYER_FITNESS < 10;

            FLAG := 1; --Notifica al utente i player stanchi.

        END IF;

    END IF;

    IF FLAG = 1 THEN
--Notifico all'utente i player che sono stanchi con relativo
   codice carta.
        DBMS_OUTPUT.PUT_LINE('ATTENZIONE! I seguenti player
           hanno forma fisica al minimo!' || CHR(10));
        FOR I IN 1..ARR_CARD_CODE.COUNT LOOP
            SELECT
                PLAYER_NAME
            INTO P_NAME
            FROM PLAYER
            WHERE CARD_ID IN (
                    SELECT PLAYER_ID
                    FROM CLUB_CARD
                    WHERE CARD_CODE = ARR_CARD_CODE(I)
                );
            DBMS_OUTPUT.PUT_LINE('['
                                    || I
                                    || '] '
                                    || P_NAME
                                    || ' ('
                                    || ARR_CARD_CODE(I)
                                    || ')'
                                    || CHR(10));
        END LOOP;
    END IF;
END IF;
END CHECK_PLAYER_FITNESS;
/
```

11.5.4 Verify Squad

```
-- Tabelle interessate: 6
--  -> CLUB_CARD, SQUAD, IS_PART_OF, STOPS, PLAYER,
   ACTIVE_DATA_PLAYER;

-- Scopo:
--  -> Verifica che i player della squadra che vuole disputare
   un match, abbiano almeno un contratto,
--      e che non ci siano player titolari infortunati, in caso
    affermativo la squadra non puo' disputare il match.
--     Inoltre notifica all'utente se ha player non titolari
   infortunati, in questo caso puo' comunque disputare il match
   .
CREATE OR REPLACE TRIGGER VERIFY_SQUAD BEFORE
INSERT OR UPDATE ON MATCH
FOR EACH ROW

DECLARE
C_NAME          SQUAD.SQUAD_CLUB_NAME%TYPE; --Nome del club.
S_NAME          SQUAD.NAME%TYPE; --Nome della squadra.
N1              NUMBER(5, 0);
N2              NUMBER(2, 0);
N3              NUMBER(2, 0);
N4              NUMBER(2, 0);
P_NAME          PLAYER.PLAYER_NAME%TYPE; --Nome del player.
B_PART          STOPS.BODY_PART%TYPE; --Parte del corpo
   infortunata.
TYPE ARRAY_CC_T IS VARRAY(18) OF VARCHAR2(8);
ARR_CARD_CODE  ARRAY_CC_T; --Array dei codici carta dei player
   della squadra.
VERIFY_SQUAD EXCEPTION; --Se la squadra non rispetta i
   requisiti per disputare il match.
OLD_MATCH EXCEPTION; --Ritorna senza fare nulla.

BEGIN
--Se si sta inserendo un nuovo match, una delle due squadre e'
   NULL, poiche' si
-- sta inserendo in attesa dell'avversario.
IF INSERTING THEN
--Controllo se la squadra da verificare e' quella in casa <
   home_squad_name>.
SELECT
    COUNT(SQUAD_CLUB_NAME)
INTO N1
FROM
    SQUAD
WHERE
        NAME = :NEW.HOME_SQUAD_NAME
    AND :NEW.VISITORS_SQUAD_NAME IS NULL;



--Se <n1 = 0>, non e' la squadra in casa, controllo quella
   fuori casa.
IF N1 = 0 THEN
```

```
    SELECT
        COUNT(SQUAD_CLUB_NAME)
    INTO N1
    FROM
        SQUAD
    WHERE
            NAME = :NEW.VISITORS_SQUAD_NAME
        AND :NEW.HOME_SQUAD_NAME IS NULL;

--Se <n1 = 0>, genero un'eccezione poiche' non ho trovato la
   squadra.
    IF N1 = 0 THEN
        RAISE NO_DATA_FOUND;
    END IF;

    --Seleziono nome <s_name>, e nome del club <c_name> della
       squadra fuori casa.
    S_NAME:=:NEW.VISITORS_SQUAD_NAME;
    SELECT SQUAD_CLUB_NAME INTO C_NAME
    FROM SQUAD WHERE NAME = S_NAME;
--Se <n1 != 0>, la squadra da verificare e' quella in casa.
ELSIF N1 <> 0 THEN
    S_NAME:=:NEW.HOME_SQUAD_NAME;
    SELECT SQUAD_CLUB_NAME INTO C_NAME
    FROM SQUAD WHERE NAME = S_NAME;
END IF;

--Controllo quanti player titolari sono senza contratti = N1.
SELECT
    COUNT(*)
INTO N1
FROM IS_PART_OF I
    JOIN ACTIVE_DATA_PLAYER A
        ON I.PLAYER_CARD_CODE = A.P_CARD_CODE
            AND I.SQUAD_NAME = S_NAME
            AND I.HOLDER = '1'
            AND A.CONTRACTS < 1;

--Contorllo quanti player titolari sono infortunati = N2.
SELECT
    COUNT(P_CARD_CODE)
INTO N2
FROM STOPS S
    JOIN IS_PART_OF I
        ON S.P_CARD_CODE = I.PLAYER_CARD_CODE
            AND I.SQUAD_NAME = S_NAME
            AND I.HOLDER = '1';



--Controllo quanti player NON titolari sono infortunati = N3.
SELECT
    COUNT(P_CARD_CODE)
INTO N3
FROM STOPS S
```

```
    JOIN IS_PART_OF I
        ON S.P_CARD_CODE = I.PLAYER_CARD_CODE
            AND I.SQUAD_NAME = S_NAME
            AND I.HOLDER = '0';

    --Controllo i contratti del manager.
SELECT
        COUNT(*)
INTO N4
FROM
    MANAGES M
    JOIN ACTIVE_DATA_MANAGER MM
        ON M.MANAGER_CARD_CODE = MM.M_CARD_CODE
            AND M.SQUAD_NAME = S_NAME
            AND MM.CONTRACTS < 1;

--Se uno dei contatori di controllo e' maggiore di 0 genero la
   verifica della squadra.
IF N1 > 0 OR N2 > 0 OR N3 > 0 OR N4 > 0 THEN
    RAISE VERIFY_SQUAD;
ELSE
    RETURN;
END IF;
--Se stiamo aggiornando una tupla in match, va analizzata solo
   l'ultima squadra inserita;
-- poiche' la squadra che aspettava l'avversario e' stata gia'
   analizzata quando e' stata inserita.
ELSIF UPDATING THEN
--Mi assicuro di analizzare solo l'ultimo match.
SELECT
    MAX(MATCH_ID)
INTO N1
FROM
    TMP_MATCH;

--Se il match corrente e' l'ultimo.
IF N1 = :OLD.MATCH_ID THEN

    --Controllo se l'ultima squadra inserita e' quella in casa.
    SELECT
        COUNT(SQUAD_CLUB_NAME)
    INTO N1
    FROM
        SQUAD
    WHERE
            NAME = :NEW.HOME_SQUAD_NAME
        AND :OLD.HOME_SQUAD_NAME IS NULL;

    --Se non e' quella in casa, controllo quella fuori casa.
    IF N1 = 0 THEN
        SELECT
            COUNT(SQUAD_CLUB_NAME)
        INTO N1
        FROM
            SQUAD
        WHERE
                NAME = :NEW.VISITORS_SQUAD_NAME
```

```
            AND :OLD.VISITORS_SQUAD_NAME IS NULL;

        --E' una partita vecchia, genero un eccezione.
        IF N1 = 0 THEN
            RAISE OLD_MATCH;
        END IF;

    --Seleziono nome <s_name>, e nome del club <c_name> della
       squadra fuori casa.
        SELECT
            SQUAD_CLUB_NAME,
            NAME
        INTO
            C_NAME,
            S_NAME
        FROM
            SQUAD
        WHERE
            NAME = :NEW.VISITORS_SQUAD_NAME;

    ELSIF N1 <> 0 THEN
    --Seleziono nome <s_name>, e nome del club <c_name> della
       squadra in casa.
        SELECT
            SQUAD_CLUB_NAME,
            NAME
        INTO
            C_NAME,
            S_NAME
        FROM SQUAD
        WHERE
            NAME = :NEW.HOME_SQUAD_NAME;
    END IF;

    --Quanti player titolari sono senza contratti = N1.
    SELECT
        COUNT(*)
    INTO N1
    FROM IS_PART_OF I
        JOIN ACTIVE_DATA_PLAYER A
            ON I.PLAYER_CARD_CODE = A.P_CARD_CODE
                AND I.SQUAD_NAME = S_NAME
                AND I.HOLDER = '1'
                AND A.CONTRACTS < 1;

    --Quanti player titolari sono infortunati = N2.
    SELECT
        COUNT(*)
    INTO N2
    FROM
        IS_PART_OF
    WHERE
            SQUAD_NAME = S_NAME
        AND HOLDER = '1'
        AND PLAYER_CARD_CODE IN (
            SELECT P_CARD_CODE
            FROM STOPS );
```

```
    --Quanti player NON titolari sono infortunati = N3;
    SELECT
        COUNT(*)
    INTO N3
    FROM
        IS_PART_OF
    WHERE
            SQUAD_NAME = S_NAME
        AND HOLDER = '0'
        AND PLAYER_CARD_CODE IN (
            SELECT
                P_CARD_CODE
            FROM
                STOPS
        );

    --Controllo i contratti del manager.
    SELECT
        COUNT(*)
    INTO N4
    FROM
        MANAGES M
        JOIN ACTIVE_DATA_MANAGER MM
            ON M.MANAGER_CARD_CODE = MM.M_CARD_CODE
                AND M.SQUAD_NAME = S_NAME
                AND MM.CONTRACTS < 1;

--Se uno dei contatori di controllo e' maggiore di 0 genero la
   verifica della squadra.
    IF N1 > 0 OR N2 > 0 OR N3 > 0 OR N4 > 0 THEN
        RAISE VERIFY_SQUAD;
    ELSE
        RETURN;
    END IF;

ELSE
    RETURN;
END IF;

END IF;
EXCEPTION
    WHEN OLD_MATCH THEN
        RETURN;
    WHEN NO_DATA_FOUND THEN
        RAISE_APPLICATION_ERROR('-20001', 'No squad found!');
    WHEN VERIFY_SQUAD THEN
    --Se <n1 > 0> la squadra ha <n1> player titolari senza
       contratti.
    --Se <n2 > 0> la squadra ha <n2> player titolari
       infortunati.
    IF N1 > 0 OR N2 > 0 THEN

        IF N1 > 0 THEN

            --Se <n4 > 0>, c'e' il manager della squadra senza
                contratti.
```

```
            IF N4 > 0 THEN
                --Seleziono il codice carta e il nome del
                   manager.
                SELECT MANAGER_CARD_CODE,F_NAME
                INTO M_CODE, M_NAME
                FROM MANAGER M
                JOIN CLUB_CARD C
                ON M.CARD_ID = C.MANAGER_ID
                JOIN MANAGES MM
                ON C.CARD_CODE = MM.MANAGER_CARD_CODE
                AND MM.SQUAD_NAME = S_NAME;

                DBMS_OUTPUT.PUT_LINE('Manager ' || m_name || '
                   (' || m_code || ')');
            END IF;

            DBMS_OUTPUT.PUT_LINE('ATTENZIONE! Contratti
               esauriti!');

            --Seleziono i codici carta dei player titolari
               senza contratti.
            SELECT
                PLAYER_CARD_CODE
            BULK COLLECT
            INTO ARR_CARD_CODE
            FROM IS_PART_OF I
                JOIN ACTIVE_DATA_PLAYER A
                    ON I.PLAYER_CARD_CODE = A.P_CARD_CODE
                        AND I.SQUAD_NAME = S_NAME
                        AND I.HOLDER = '1'
                        AND A.CONTRACTS < 1;

            FOR I IN 1..ARR_CARD_CODE.COUNT LOOP
                SELECT
                    PLAYER_NAME
                INTO P_NAME
                FROM
                    PLAYER
                WHERE
                    CARD_ID IN (
                        SELECT
                            PLAYER_ID
                        FROM
                            CLUB_CARD
                        WHERE
                            CARD_CODE = ARR_CARD_CODE(I)
                    );

                DBMS_OUTPUT.PUT_LINE('[' || I || '] ' || P_NAME
                    || ' (' || ARR_CARD_CODE(I) || ')' || CHR
                   (10));

            END LOOP;
        END IF;
    --Se <n2 > 0> la squadra ha <n2> player titolari
       infortunati.
        IF N2 > 0 THEN
```

```
            DBMS_OUTPUT.PUT_LINE('ATTENZIONE! Giocatori
               titolari infortunati!');
            --Seleziono i codici carta dei player titolari
               infortunati.
            SELECT
                P_CARD_CODE
            BULK COLLECT
            INTO ARR_CARD_CODE
            FROM
                        STOPS S
                JOIN IS_PART_OF I ON S.P_CARD_CODE = I.
                   PLAYER_CARD_CODE
                                        AND I.SQUAD_NAME =
                                           S_NAME
                                        AND I.HOLDER = '1';

            FOR I IN 1..N2 LOOP
                SELECT
                    BODY_PART
                INTO B_PART
                FROM
                    STOPS
                WHERE
                    P_CARD_CODE = ARR_CARD_CODE(I);

                SELECT
                    PLAYER_NAME
                INTO P_NAME
                FROM
                    PLAYER
                WHERE
                    CARD_ID IN (
                        SELECT
                            PLAYER_ID
                        FROM
                            CLUB_CARD
                        WHERE
                            CARD_CODE = ARR_CARD_CODE(I)
                    );

                DBMS_OUTPUT.PUT_LINE('[' || I || '] ' || P_NAME
                    || ' (' || ARR_CARD_CODE(I) || ') - (' ||
                       B_PART || ')' || CHR(10));

            END LOOP;
        END IF;
        RAISE_APPLICATION_ERROR('-20008', '>>>>>>>>La partita
           non puo'' essere disputata!<<<<<<<<');

    --Se <n3 > 0>, la squadra ha player NON titolari
       infortunati, vengono comunicati i dettagli dai player
       all'utente che puo'
    -- comunque disputare il match.
    ELSIF N3 > 0 THEN
        DBMS_OUTPUT.PUT_LINE('ATTENZIONE! Giocatori infortunati
           !');
```

```
        --Seleziono i codici carta dei player non titolari
           infortunati.
        SELECT
            P_CARD_CODE
        BULK COLLECT
        INTO ARR_CARD_CODE
        FROM STOPS S
            JOIN IS_PART_OF I
                ON S.P_CARD_CODE = I.PLAYER_CARD_CODE
                    AND I.SQUAD_NAME = S_NAME
                    AND I.HOLDER = '0';

        FOR I IN 1..N3 LOOP
            SELECT
                BODY_PART
            INTO B_PART
            FROM
                STOPS
            WHERE
                P_CARD_CODE = ARR_CARD_CODE(I);

            SELECT
                PLAYER_NAME
            INTO P_NAME
            FROM
                PLAYER
            WHERE
                CARD_ID IN (
                    SELECT
                        PLAYER_ID
                    FROM
                        CLUB_CARD
                    WHERE
                        CARD_CODE = ARR_CARD_CODE(I)
                );

            DBMS_OUTPUT.PUT_LINE('[' || I || '] ' || P_NAME ||
               ' (' || ARR_CARD_CODE(I) || ') - (' || B_PART ||
                ')' || CHR(10));

        END LOOP;
    END IF;
END VERIFY_SQUAD;
/
```

11.5.5 Division Update

```
-- Tabelle interessate: 3
--  -> SQUAD, CLUB, DIVISION_RIVALS;

-- Scopo:
--  -> Gestisce la promozione/retrocessione di un club dopo
   aver giocato una partita.

CREATE OR REPLACE TRIGGER DIVISION_UPDATE AFTER
UPDATE OF RESULTS ON MATCH
FOR EACH ROW
DECLARE
N1              NUMBER(5, 0); --Id dell'ultimo match disputato.
HOME_POINTS     NUMBER(5, 0); --Punti della squadra in casa.
AWAY_POINTS     NUMBER(5,0); --Punti della squadra fuori casa.
TH              NUMBER(4, 0); --Soglia delle divisioni.
D1              NUMBER(2, 0); --Numero di divisione squadra in
   casa.
D2              NUMBER(2, 0); --Numero di divisione squadra
   fuori casa.
BEGIN
--Seleziono l'id dell'ultimo match.
SELECT
    MAX(MATCH_ID)
INTO N1
FROM
    TMP_MATCH;

--Controllo che sto analizzando l'ultimo match.
IF N1 = :OLD.MATCH_ID THEN
--Seleziono punti e numero divisione della squadra in casa.
    SELECT
        DR_POINTS,
        DIVISION_NUMBER
    INTO
        HOME_POINTS,
        D1
    FROM
        CLUB
    WHERE
        CLUB_NAME IN
            (SELECT SQUAD_CLUB_NAME
            FROM SQUAD
            WHERE NAME = :NEW.HOME_SQUAD_NAME);

--Seleziono punti e numero divisione della squadra fuori casa.
    SELECT
        DR_POINTS,
        DIVISION_NUMBER
    INTO
        AWAY_POINTS,
        D2
    FROM
        CLUB
    WHERE
        CLUB_NAME IN
```

```
            (SELECT SQUAD_CLUB_NAME
            FROM SQUAD
            WHERE NAME = :NEW.VISITORS_SQUAD_NAME);

--Seleziono la soglia della divisione successiva a quella in
   cui si trova il club <home_club_name> che gioca in casa.
    SELECT
        THRESHOLD
    INTO TH
    FROM
        DIVISION_RIVALS
    WHERE
        DIVISION_NUMBER = D1 - 1;

--Risultato match
    DBMS_OUTPUT.PUT_LINE('ATTENZIONE! Risultati match:');
    DBMS_OUTPUT.PUT_LINE('Risultato   ('
                         || :NEW.HOME_SQUAD_NAME
                         || ') -> '
                         || :NEW.RESULTS
                         || ' <- ('
                         || :NEW.VISITORS_SQUAD_NAME
                         || ')'
                         || CHR(10));

    DBMS_OUTPUT.PUT_LINE('Punti       ('
                         || :NEW.HOME_POINTS
                         || ')           ('
                         || :NEW.VISITORS_POINTS
                         || ')'
                         || CHR(10));

    DBMS_OUTPUT.PUT_LINE('Crediti     ('
                         || :NEW.HOME_CREDITS
                         || ')           ('
                         || :NEW.VISITORS_CREDITS
                         || ')');

--Se i punti del club della squadra in casa superano la soglia
   <th>, si ha una promozione.
    IF HOME_POINTS > TH THEN
        UPDATE CLUB
        SET
            DIVISION_NUMBER = DIVISION_NUMBER - 1
        WHERE
            CLUB_NAME IN
                (SELECT SQUAD_CLUB_NAME
                FROM SQUAD
                WHERE NAME = :NEW.HOME_SQUAD_NAME);

        DBMS_OUTPUT.PUT_LINE('| La squadra '
                             || :NEW.HOME_SQUAD_NAME
                             || ' viene promossa in
                                divisione '
                             ||(D1 - 1)
                             || ' !');
```

```
    END IF;

--Seleziono la soglia della divisione successiva a quella in
   cui si trova il club <away_club_name> che gioca fuori casa.
    SELECT
        THRESHOLD
    INTO TH
    FROM
        DIVISION_RIVALS
    WHERE
        DIVISION_NUMBER = D2 - 1;

--Se i punti del club della squadra fuori casa superano la
   soglia <th>, si ha una promozione.
    IF AWAY_POINTS > TH THEN
        UPDATE CLUB
        SET
            DIVISION_NUMBER = DIVISION_NUMBER - 1
        WHERE
            CLUB_NAME IN
                (SELECT SQUAD_CLUB_NAME
                FROM SQUAD
                WHERE NAME = :NEW.VISITORS_SQUAD_NAME);

        DBMS_OUTPUT.PUT_LINE('| Il club '
                             || :NEW.VISITORS_SQUAD_NAME
                             || ' viene promosso in
                                divisione '
                             ||(D2 - 1)
                             || ' !');

    END IF;

--Retrocessione, a meno che non ci si trovi in divisione 10 (
   Soglia della divisione 9 = 100), si puo' retrocedere.
    IF HOME_POINTS >= 100 THEN

--Seleziono la soglia della divisione del club in casa.
        SELECT
            THRESHOLD
        INTO TH
        FROM
            DIVISION_RIVALS
        WHERE
            DIVISION_NUMBER = D1;

--Se i punti del club in casa sono minori della soglia, viene
   retrocessa.
        IF HOME_POINTS < TH THEN
            UPDATE CLUB
            SET
                DIVISION_NUMBER = DIVISION_NUMBER + 1
            WHERE
                CLUB_NAME IN
                    (SELECT SQUAD_CLUB_NAME
                    FROM SQUAD
                    WHERE NAME = :NEW.HOME_SQUAD_NAME);
```

```
            DBMS_OUTPUT.PUT_LINE('| Il club '
                                 || :NEW.HOME_SQUAD_NAME
                                 || ' viene retrocesso in
                                    divisione '
                                 ||(D1 + 1)
                                 || ' !');

        END IF;

    END IF;

--Retrocessione, a meno che non ci si trovi in divisione 10 (
   Soglia della divisione 9 = 100), si puo' retrocedere.
    IF AWAY_POINTS >= 100 THEN

--Seleziono la soglia della divisione del club fuori casa.
        SELECT
            THRESHOLD
        INTO TH
        FROM
            DIVISION_RIVALS
        WHERE
            DIVISION_NUMBER = D2;

--Se i punti del club fuori casa sono minori della soglia,
   viene retrocessa.
        IF AWAY_POINTS < TH THEN
            UPDATE CLUB
            SET
                DIVISION_NUMBER = DIVISION_NUMBER + 1
            WHERE
                CLUB_NAME IN
                    (SELECT SQUAD_CLUB_NAME
                    FROM SQUAD
                    WHERE NAME = :NEW.VISITORS_SQUAD_NAME);

            DBMS_OUTPUT.PUT_LINE('| Il club '
                                 || :NEW.VISITORS_SQUAD_NAME
                                 || ' viene retrocesso in
                                    divisione '
                                 ||(D2 + 1)
                                 || ' !');

        END IF;

    END IF;

ELSE
    RETURN;
END IF;

END DIVISION_UPDATE;
/
```

11.5.6 Player Update After Match

```
-- Tabelle interessate: 4
--  -> SQUAD, IS_PART_OF, ACTIVE_DATA_PLAYER, TMP_MATCH;

-- Scopo:
--  -> Aggiorna i dati attivi dei player dopo che il club ha
   disputato un match.
--     Aggiorna la forma fisica <player_fitness>;
--     Rimuove un contratto, un giorno da eventuali infortuni e
    training attivi.
CREATE OR REPLACE TRIGGER PLAYER_UPDATE_AFTER_MATCH AFTER
UPDATE OF HOME_POINTS, VISITORS_POINTS, HOME_CREDITS,
   VISITORS_CREDITS ON MATCH
REFERENCING
        OLD AS OLD
        NEW AS NEW
FOR EACH ROW

DECLARE
N1          NUMBER(5, 0);--Id dell'ultimo match.
HOME_SQUAD  VARCHAR2(16);--Nome della squadra in casa.
HOME_CLUB   CLUB.CLUB_NAME%TYPE;--Nome del club della squadra
   in casa.
AWAY_SQUAD  VARCHAR2(16);--Nome della squadra fuori casa.
AWAY_CLUB   CLUB.CLUB_NAME%TYPE;--Nome del club della squadra
   fuori casa.
RAND_VALUE  NUMBER(2, 0);--Numero random.

BEGIN
--Seleziono l'id dell'ultimo match in <n1>.
SELECT MAX(MATCH_ID)
INTO N1
FROM TMP_MATCH;

--Se il match corrente e' l'ultimo disputato.
IF N1 = :NEW.MATCH_ID THEN
--Seleziono i nomi delle squadre che hanno appena giocato.
    SELECT
        NAME,
        SQUAD_CLUB_NAME
    INTO
        HOME_SQUAD,
        HOME_CLUB
    FROM
        SQUAD
    WHERE
        NAME = :NEW.HOME_SQUAD_NAME;

    SELECT
        NAME,
        SQUAD_CLUB_NAME
    INTO
        AWAY_SQUAD,
        AWAY_CLUB
    FROM
        SQUAD
```

```
    WHERE
        NAME = :NEW.VISITORS_SQUAD_NAME;

--Rimuovo i benefici del training (player_rating + 2).
    UPDATE IS_PART_OF
    SET
        PLAYER_RATING = PLAYER_RATING - 2
    WHERE
        ( SQUAD_NAME = HOME_SQUAD
            OR SQUAD_NAME = AWAY_SQUAD )
        AND HOLDER = '1'
        AND PLAYER_CARD_CODE IN (
            SELECT
                P_CARD_CODE
            FROM
                ACTIVE_DATA_PLAYER
            WHERE
                ACTIVE_TRAINING IS NOT NULL
        );

--Aggiorno la nuova valutazione <squad_rating> della squadra in
    casa.
    UPDATE SQUAD
    SET
        SQUAD_RATING = (
            SELECT
                GET_SQUAD_RATING(HOME_SQUAD, HOME_CLUB)
            FROM
                DUAL
        )
    WHERE
            NAME = HOME_SQUAD
        AND SQUAD_CLUB_NAME = HOME_CLUB;

--Aggiorno la nuova valutazione <squad_rating> della squadra
   fuori casa.
    UPDATE SQUAD
    SET
        SQUAD_RATING = (
            SELECT
                GET_SQUAD_RATING(AWAY_SQUAD, AWAY_CLUB)
            FROM
                DUAL
        )
    WHERE
            NAME = AWAY_SQUAD
        AND SQUAD_CLUB_NAME = AWAY_CLUB;

--Aggiorno i contratti e rimuovo i training.
    UPDATE ACTIVE_DATA_PLAYER
    SET
        CONTRACTS = CONTRACTS - 1,
        ACTIVE_TRAINING = NULL
    WHERE
        P_CARD_CODE IN (
            SELECT
                PLAYER_CARD_CODE
```

```
            FROM
                IS_PART_OF
            WHERE
                ( SQUAD_NAME = HOME_SQUAD
                    OR SQUAD_NAME = AWAY_SQUAD )
                AND HOLDER = '1'
        );

--Aggiorno i contratti dei manager.
    UPDATE ACTIVE_DATA_MANAGER
    SET
        CONTRACTS = CONTRACTS - 1
    WHERE
        M_CARD_CODE IN (
            SELECT
                M_CARD_CODE
            FROM
                MANAGES
            WHERE
                ( SQUAD_NAME = HOME_SQUAD
                    OR SQUAD_NAME = AWAY_SQUAD )
        );

--Aggiorno la forma fisica <player_fitness>.
    RAND_VALUE := DBMS_RANDOM.VALUE(LOW => 2, HIGH => 3);
--Posizione player in (GK).
    UPDATE ACTIVE_DATA_PLAYER
    SET
        PLAYER_FITNESS = PLAYER_FITNESS - RAND_VALUE
    WHERE
        P_CARD_CODE IN (
            SELECT
                PLAYER_CARD_CODE
            FROM
                IS_PART_OF
            WHERE
                ( SQUAD_NAME = HOME_SQUAD
                    OR SQUAD_NAME = AWAY_SQUAD )
                AND PLAYER_POSITION = 'GK'
                AND HOLDER = '1'
        );

-- Posizione player in (CB)
    RAND_VALUE := DBMS_RANDOM.VALUE(LOW => 3, HIGH => 5);
    UPDATE ACTIVE_DATA_PLAYER
    SET
        PLAYER_FITNESS = PLAYER_FITNESS - RAND_VALUE
    WHERE
        P_CARD_CODE IN (
            SELECT
                PLAYER_CARD_CODE
            FROM
                IS_PART_OF
            WHERE
                ( SQUAD_NAME = HOME_SQUAD
                    OR SQUAD_NAME = AWAY_SQUAD )
                AND PLAYER_POSITION = 'CB'
```

```
                AND HOLDER = '1'
        );

-- Posizione player in (RB LB LW RW RWB LWB RM LM)
    RAND_VALUE := DBMS_RANDOM.VALUE(LOW => 8, HIGH => 10);
    UPDATE ACTIVE_DATA_PLAYER
    SET
        PLAYER_FITNESS = PLAYER_FITNESS - RAND_VALUE
    WHERE
        P_CARD_CODE IN (
            SELECT
                PLAYER_CARD_CODE
            FROM
                IS_PART_OF
            WHERE
                ( ( SQUAD_NAME = HOME_SQUAD
                    OR SQUAD_NAME = AWAY_SQUAD )
                    AND ( PLAYER_POSITION = 'LB'
                        OR PLAYER_POSITION = 'RB'
                        OR PLAYER_POSITION = 'LW'
                        OR PLAYER_POSITION = 'RW'
                        OR PLAYER_POSITION = 'RWB'
                        OR PLAYER_POSITION = 'LWB'
                        OR PLAYER_POSITION = 'LM'
                        OR PLAYER_POSITION = 'RM' )
                    AND HOLDER = '1' )
        );

-- Posizione player in (CM ST CAM CDM CF RF LF)
    RAND_VALUE := DBMS_RANDOM.VALUE(LOW => 5, HIGH => 8);
    UPDATE ACTIVE_DATA_PLAYER
    SET
        PLAYER_FITNESS = PLAYER_FITNESS - RAND_VALUE
    WHERE
        P_CARD_CODE IN (
            SELECT
                PLAYER_CARD_CODE
            FROM
                IS_PART_OF
            WHERE
                ( SQUAD_NAME = HOME_SQUAD
                    OR SQUAD_NAME = AWAY_SQUAD )
                AND ( PLAYER_POSITION = 'CM'
                        OR PLAYER_POSITION = 'ST'
                        OR PLAYER_POSITION = 'CAM'
                        OR PLAYER_POSITION = 'CDM'
                        OR PLAYER_POSITION = 'CF'
                        OR PLAYER_POSITION = 'RF'
                        OR PLAYER_POSITION = 'LF' )
                AND HOLDER = '1'
        );

--Controllo che non sia negativa la forma fisica <
   player_fitness>.
    UPDATE ACTIVE_DATA_PLAYER
    SET PLAYER_FITNESS = 1
    WHERE PLAYER_FITNESS < 1;
```

```
ELSE
    RETURN;
END IF;
END PLAYER_UPDATE_AFTER_MATCH;
```

11.5.7 Stops Randomizer

```
-- Tabelle interessate: 5
--  -> STOPS, IS_PART_OF, TMP_MATCH, PLAYER, CLUB_CARD;

-- Scopo:
--  -> Gestisce gli infortuni dei player dopo che hanno
   disputato un match.
CREATE OR REPLACE TRIGGER STOPS_RANDOMIZER AFTER
UPDATE OF RESULTS ON MATCH
FOR EACH ROW

DECLARE
N1            NUMBER(5, 0);--Id dell'ultimo match.
N2            NUMBER(2, 0) := 1;--Contatore generico,
   corrisponde al numero di giocatori infortunati.
P_NAME        VARCHAR2(32);--Nome del player infortunato.
HOME_SQUAD    VARCHAR2(16);--Nome della squadra in casa.
HOME_CLUB     CLUB.CLUB_NAME%TYPE;--Nome del club della squadra
   in casa.
AWAY_SQUAD    VARCHAR2(16);--Nome della squadra fuori casa.
AWAY_CLUB     CLUB.CLUB_NAME%TYPE;--Nome del club della squadra
   fuori casa.
NEW_STOP      STOPS%ROWTYPE;--Nuovo infortunio.
STOP_P        NUMBER(2, 0);--Percentuale d'infortunio.
RAND_VALUE    NUMBER(3, 0);--Numero random.
INJURY_TYPE   NUMBER(1, 0);--Tipo d'infortunio.
DAYS          NUMBER(1, 0);--Giorni di fermo per l'infortunio.

--Record che contiene i codici carta e la forma fisica <
   player_fitness> dei giocatori della squadra.
TYPE P_FF IS RECORD (
    C_CODE  ACTIVE_DATA_PLAYER.P_CARD_CODE%TYPE,
    FF      ACTIVE_DATA_PLAYER.PLAYER_FITNESS%TYPE
);

--Lista del record <p_ff>.
TYPE P_FF_INFO IS TABLE OF P_FF;
P_FF_LIST     P_FF_INFO;

TYPE ARRAY_BP IS VARRAY(7) OF VARCHAR2(16);
BP_ARR        ARRAY_BP; --Array del tipo di infortunio.

BEGIN
--Tipo d'infortunio.
BP_ARR := ARRAY_BP('Foot', 'Leg', 'Knee', 'Arm', 'UpperBody','
   Head');
--Seleziono l'id dell'ultimo match.
SELECT
    MAX(MATCH_ID)
INTO N1
FROM
    TMP_MATCH;



--Se il match corrente e' l'ultimo disputato.
```

```
IF N1 = :OLD.MATCH_ID THEN
--Seleziono nome della squadra, e nome del club delle squadre
   che hanno appena giocato.
    HOME_SQUAD:=:NEW.HOME_SQUAD_NAME;
    SELECT SQUAD_CLUB_NAME INTO HOME_CLUB
    FROM SQUAD WHERE NAME = HOME_SQUAD;

    AWAY_SQUAD:=:NEW.VISITORS_SQUAD_NAME;
    SELECT SQUAD_CLUB_NAME INTO AWAY_CLUB
    FROM SQUAD WHERE NAME = AWAY_SQUAD;
--Seleziono codice carta, e forma fisica dei player infortunati
   .
    SELECT
        P_CARD_CODE,
        PLAYER_FITNESS
    BULK COLLECT
    INTO P_FF_LIST
    FROM
        ACTIVE_DATA_PLAYER
    WHERE
        P_CARD_CODE IN (
            SELECT
                PLAYER_CARD_CODE
            FROM
                IS_PART_OF
            WHERE
                ( SQUAD_NAME = HOME_SQUAD
                    OR SQUAD_NAME = AWAY_SQUAD )
                AND HOLDER = '1'
        );

    FOR INDX IN 1..P_FF_LIST.COUNT LOOP
--Se un giocatore ha forma fisica >= 80 non si puo' infortunare
   .
        STOP_P := 80 - P_FF_LIST(INDX).FF;
        RAND_VALUE := DBMS_RANDOM.VALUE(LOW => 1, HIGH => 100);
--Il player potrebbe infortunarsi.
        IF RAND_VALUE < STOP_P THEN

            RAND_VALUE := DBMS_RANDOM.VALUE(LOW => 1, HIGH =>
               100);
--Il giocatore si infortunia.
            IF RAND_VALUE < 5 THEN
--Tipologia di infortunio.
            INJURY_TYPE := DBMS_RANDOM.VALUE(LOW => 1, HIGH =>
               7);
--Giorni di infortunio.
            DAYS := DBMS_RANDOM.VALUE(LOW => 1, HIGH => 5);

--Preparo la nuova tupla di infortunio <stop>.
            SELECT
                SYSDATE,
                BP_ARR(INJURY_TYPE),
                DAYS,
                P_FF_LIST(INDX).C_CODE
            INTO NEW_STOP
            FROM
```

```
                DUAL;

--Inserisco la nuova tupla.
            INSERT INTO STOPS VALUES NEW_STOP;

--Seleziono il nome del player, per notificare all'utente quale
    giocatore ha subito un infortunio.
            SELECT
                PLAYER_NAME
            INTO P_NAME
            FROM
                        CLUB_CARD CC
                JOIN PLAYER P ON CC.PLAYER_ID = P.CARD_ID
            WHERE
                CC.CARD_CODE = P_FF_LIST(INDX).C_CODE;

            DBMS_OUTPUT.PUT_LINE('Giocatore infortunato -> ['
                                 || N2
                                 || '] '
                                 || P_NAME
                                 || ' ('
                                 || P_FF_LIST(INDX).C_CODE
                                 || ') - ('
                                 || NEW_STOP.BODY_PART
                                 || ')'
                                 || CHR(10));
--Incremento il numero di giocatori infortunati.
            N2 := N2 + 1;
        END IF;
    END IF;
    END LOOP;
--Azzero il numero di giocatori infortunati.
    N2 := 1;
END IF;

END STOPS_RANDOMIZER;
/
```

11.5.8 Player In Squad

```
-- Tabelle interessate: 1
--  -> IS_PART_OF;

-- Scopo:
--  -> gestisce il caso in cui venga messo in vendita un player
    che e' schierato in una squadra
--      in questo caso il player verra' rimosso dalla squadra
   prima di essere messo in vendita.
CREATE OR REPLACE TRIGGER PLAYER_IN_SQUAD BEFORE
INSERT OR UPDATE ON TRANSACTION
FOR EACH ROW

DECLARE
N1 NUMBER(1, 0);
PLAYER_HOLDER EXCEPTION;

BEGIN

SELECT
    COUNT(PLAYER_CARD_CODE)
INTO N1
FROM
    IS_PART_OF
WHERE
    PLAYER_CARD_CODE = :NEW.T_CARD_CODE;

--Se il player che si sta provando a vendere e schierato in una
    squadra.
IF N1 != 0 THEN
    RAISE PLAYER_HOLDER;
END IF;
EXCEPTION
WHEN PLAYER_HOLDER THEN
--Cancello il player dalla squadra.
    DELETE FROM IS_PART_OF
    WHERE
        PLAYER_CARD_CODE = :NEW.T_CARD_CODE;

END PLAYER_IN_SQUAD;
/
```

11.6 Jobs

11.6.1 Job Transaction

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB (
        job_name => 'job_transaction',
        job_type => 'STORED_PROCEDURE',
        job_action => 'CHECK_DURATION', --Esegue la procedura
           check_transition
        start_date => trunc(sysdate), --Inizia alle 11:00 AM
           del giorno in cui viene eseguito.
        repeat_interval => 'FREQ=DAILY;BYDAY=MON,TUE,WED,THU,
           FRI,SAT,SUN; BYHOUR
           =0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23;
           ',
        enabled => TRUE,
        comments => 'Cancellazione transazioni non vendute');
END;
/
```

11.6.2 Job Gift

```
--Scopo: Ogni giovedi alle 9:00 distribuisce i premi in base
   alla divisione in cui si trova un club
--              con l'ausilio della procedura GIFT.
BEGIN
DBMS_SCHEDULER.CREATE_JOB (
        job_name => 'job_gift',
        job_type => 'STORED_PROCEDURE',
        job_action => 'GIFT', --Esegue la procedura GIFT.
        start_date => trunc(sysdate),
        repeat_interval => 'FREQ=WEEKLY;BYDAY=THU;BYHOUR=9;',
           --Ogni giovedi alle 9:00 AM.
        enabled => TRUE ,
        comments => 'Regali in base alla divisione!');
END;
/
```

11.6.3 Job Old Match

```
BEGIN
    DBMS_SCHEDULER.CREATE_JOB (
    job_name                        =>      'job_rollout_match',
    job_type                        =>      'PLSQL_BLOCK',
    job_action          =>      'BEGIN
                            DELETE FROM TMP_MATCH
                            WHERE MATCH_DATE<SYSDATE-1;
                        END;',
    start_date          => TRUNC(SYSDATE),
    repeat_interval => 'FREQ=DAILY;BYDAY=MON,TUE,WED,THU,FRI,
       SAT,SUN; BYHOUR=0;', --Ogni giorno a mezzanotte.
    enabled                         =>      TRUE,
    comments            =>      'Cancellazione dei match del
       giorno precedente, dalla tabella TMP_MATCH');
END;
/
```

11.7 Sequence

11.7.1 Match ID Sequence

```
-- Usata per il aggiungere il <MATCH_ID> alla tabella MATCH.
CREATE SEQUENCE MATCH_ID_SEQ
  INCREMENT BY 1
  START WITH 1
  MAXVALUE 999
  NOCACHE
  NOCYCLE;
/
```

11.8 Type

11.8.1 Type

```
--Utilizzato come parametro di output nella funzione
   MODULE_POSITIONS.
--E' un array che conterra' le posizioni dei player in base al
   modulo scelto.
create OR REPLACE TYPE array_pos_t is VARRAY(11) of VARCHAR2(3)
   ;
```