UNIVERSITÀ DEGLI STUDI DI NAPOLI PARTHENOPE
DIPARTIMENTO DI SCIENZE E TECNOLOGIE
CORSO DI BASI DI DATI E LABORATORIO

Data di consegna: 22-07-2022
Anno accademico: 2021/22

Progetto CINETICKETDB

"Realizzazione di una base di dati per la gestione di una catena di cinema"

Alunni

Simone Palladino	0124002316
Luca Tartaglia	0124002294
Mattia Di Palma	0124002448

Sommario

Informazioni generali

Questo progetto è finalizzato alla realizzazione di una base di dati che consente la gestione e la pianificazione degli spettacoli all'interno di una catena di cinema. Le risorse umane del cinema sono costituite dagli utenti e dai gestori, i quali dirigono le attività e si occupano della parte "economica". Una base di dati che gestisce una catena di cinema consente di poter visualizzare gli spettacoli inerenti a quel film, poter acquistare delle carte fedeltà che successivamente permettono agli utenti di possedere degli sconti, acquistare biglietti e scegliere il tipo delle sale. Successivamente spiegheremo come verrà realizzata questa specifica base di dati passo per passo, introducendo tutti i requisiti minimi specificati nel progetto prototipo.

Specifiche e analisi dei requisiti

Il database CINETICKET gestisce le prenotazioni per gli spettacoli di una catena di cinema a cui gli utenti possono assistere, l'obiettivo finale del progetto è una corretta e coerente conservazione di tutti i dati trattati che verranno successivamente approfonditi.
Nel database, ogni utente ha la possibilità di inserire uno o più biglietti all'interno del suo carrello, ogni biglietto è associato ad uno spettacolo che a sua volta è associato ad un cinema identificato da una chiave naturale composta basata sulla sua posizione geografica. Il biglietto è inoltre collegato ad una sala e ad un posto. L'utente dopo aver acquistato il biglietto può inserire una recensione attribuendo un voto per ogni categoria del film.
Di quest'ultimo conosciamo la compagnia di produzione che lo ha realizzato e chi ha partecipato, sia come attore sia come regista.
La catena di cinema consente all'utente di possedere una carta fedeltà che permette a sua volta di ricevere delle promozioni applicate sui carrelli. Ogni cinema possiede varie sale di tipologie diverse collegate ai vari spettacoli in programmazione. L'utente può in modo autonomo "effettuare degli acquisti" modificando la data del pagamento del suo carrello, i biglietti collegati a spettacoli terminati e mai acquistati vengono in automatico eliminati.
Allo stesso modo l'utente può cancellare un ordine ed avere un "rimborso" se la cancellazione è richiesta almeno un giorno prima dalla data dello spettacolo più vicina.
Nel nostro progetto riconosciamo anche un'altra figura che può attivamente interagire al database, ovvero il gestore.
Quest'ultimo può gestire la struttura dei vari cinema inserendo i dati riguardanti gli spettacoli, le sale e i posti.
Il gestore ha anche il compito di verificare che i vari utenti svolgano in modo corretto le operazioni di inserimento delle recensioni e modificare l'abilitazione a suoi acquisti futuri.
Le diverse operazioni che possono essere eseguite sia dal gestore che dall'utente verranno dettagliatamente analizzate nei capitoli successivi.

L'analisi dei requisiti produce quindi una base di dati composta dalle seguenti entità:

- **Utente** (Attributi: Username, Nome, Cognome, Sesso, Data_di_nascita, Luogo_di_nascita, Abilitato)
- **Carta Fedeltà** (Attributi: Codice carta fedeltà, Data scadenza)
- **Promozione** (Attributi: Codice promozione, Nome promozione, Data inizio promozione, Data fine promozione)

- **Carrello** (Attributi: Codice carrello, Data pagamento)
- **Carta di credito** (Attributi: Numero carta, Codice CVV, Data Scadenza)
- **Biglietto** (Attributi: Codice biglietto, prezzo)
- **Recensione** (Attributi: Id recensione, Voto: Voto Protagonista, Voto Antagonista, Voto Trama, Voto Sceneggiatura, Voto Colonna Sonora)
- **Posto** (Attributi: Nome Posto: Fila Posto, Numero Posto)
- **Sala** (Attributi: Codice Sala, Tipo sala, Numero sala, Massima capienza)
- **Cinema** (Attributi: Indirizzo cinema: Via ,Città, CAP; Nome cinema)
- **Spettacolo** (Attributi: Data spettacolo: Ora inizio, Ora fine; Lingua)
- **Film** (Attributi: Codice film, Trama, Genere, Classificazione, Anno produzione, Durata, Titolo)
- **Compagnia di produzione** (Attributi: Nome compagnia, Città)
- **Professionista** (Codice professionista, Nome, Data di nascita)

Il collegamento tra le varie entità viene effettuato attraverso le seguenti associazioni/relazioni:

- L'utente **Rinnova** la carta fedeltà (Attributi associazione: Data sottoscrizione)
- La carta fedeltà **Offre** la promozione
- La promozione **Viene applicato** sul carrello
- Carrello **paga con** Carta di credito
- Carrello **possiede** Biglietto
- L'utente **inserisce in** Carrello (Attributi associazione: Data inserimento)
- Biglietto **associa** Posto
- Sala **contiene** Posto
- Cinema **è composto** da Sala
- Biglietto **Scrive** Recensione (Attributi associazione: Data pubblicazione)
- Biglietto **Da accesso a** Spettacolo
- Spettacolo **Proiettato in** Sala
- Recensione **riguarda** Film
- Spettacolo **include** Film
- Film **prodotto da** Compagnia di produzione
- Film **Diretto da** Regista (**Regista** Entità figlia di **Professionista**)
- Film **Recitato da** Attore (**Attore** Entità figlia di **Professionista**, Attributi associazione: Nome Ruolo)

Progettazione

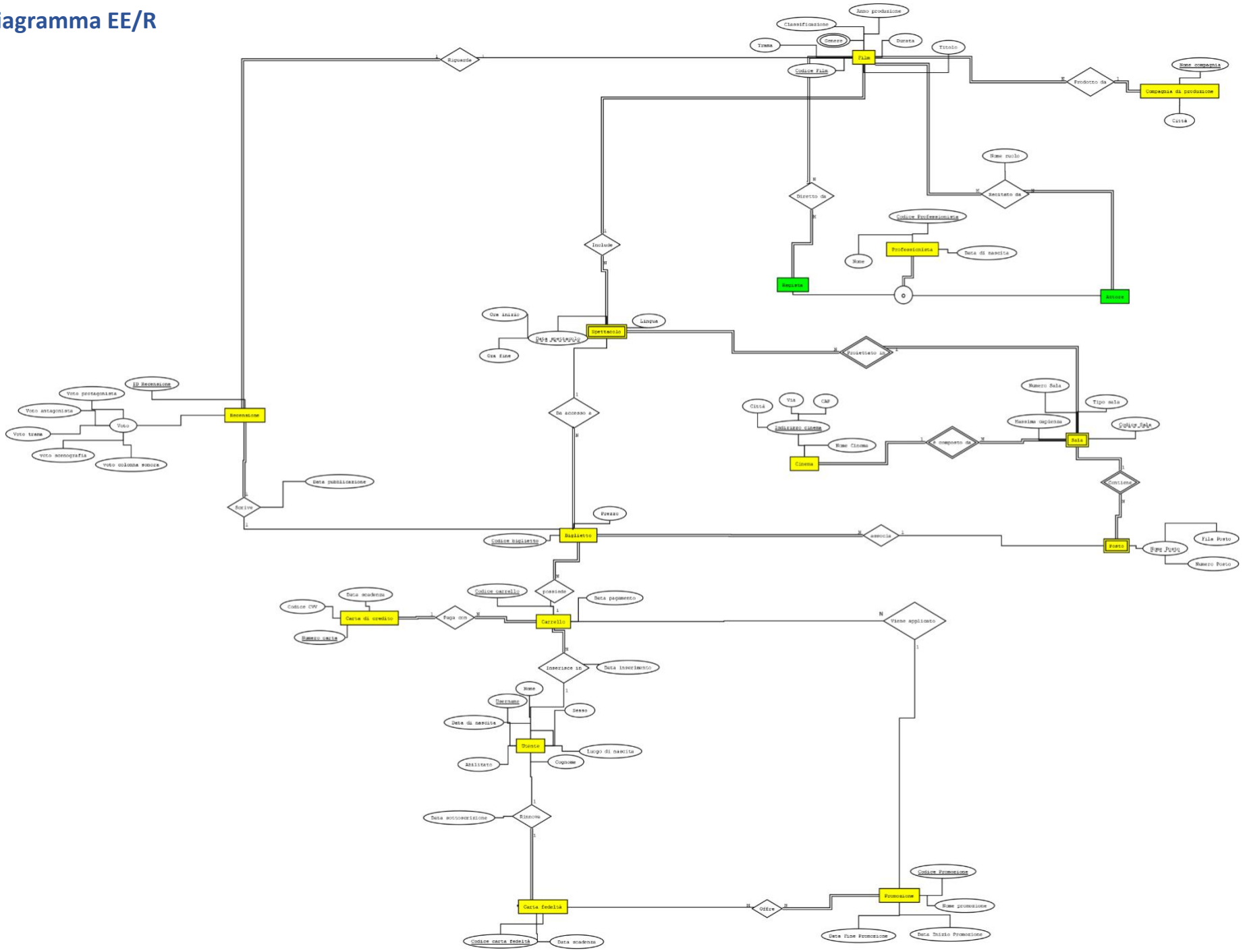
Glossario dei termini

In questa fase viene specificato il glossario dei termini che sintetizza le informazioni principali relative alle entità fondamentali del nostro schema concettuale, questo permette di individuare successivamente anomalie o ambiguità all'interno della nostra base di dati.

Entità	Descrizione	Sinonimi	Entità collegate
Utente	Cliente del cinema che oltre a vedere i film può scegliere se recensire o no i film visti.	Cliente, Fruitore	Carrello, Carta fedeltà
Carta fedeltà	Oggetto che possiede il cliente del cinema e consente di poter ricevere delle promozioni	Carta cinema, pass cinema	Utente, Promozione
Promozione	Offerta che viene eventualmente rilasciata dal cinema nel momento in cui l'utente possiede la carta fedeltà	Promozione carta fedeltà	Carta fedeltà, Carrello
Carrello	Oggetto che permette l'inserimento dei biglietti e l'acquisto di essi da parte dell'utente	Carrello cinema	Biglietto, carta di credito, promozione, Utente
Biglietto	Componente acquistata dall'utente che permette la visualizzazione e la recensione del film	Biglietto spettacolo	Recensione, Carrello, Spettacolo, Posto
Spettacolo	Componente che contiene le informazioni del film	Show, Esibizione	Biglietto, Film, Sala
Carta di credito	Componente che consente l'acquisto dei biglietti	Carta acquisto	Carrello
Recensione	L'utente può recensire un qualsiasi film visto associato ad un biglietto pagato attribuendo voti a varie categorie.	Revisione, Critica, Giudizio	Biglietto, Film
Compagnia di produzione	Agenzia che produce il film e lo distribuisce.	Produttore, Distributore	Film

Una volta stabilito il glossario si può passare alla definizione dello schema concettuale che sintetizza, attraverso una rappresentazione grafica, la nostra base di dati con le sue specifiche definite nei primi punti.

Diagramma EE/R



Utenti e le loro categorie

L'acquisto di biglietti multipli avviene attraverso l'**user** registrato, mentre la gestione degli spettacoli e del comportamento onesto da parte degli utenti è affidata alla figura del **gestore**. Dunque il DB avrà solo tre utenti: **l'amministratore, il gestore e l'user**.
Segue l'elenco di tutti i permessi di ciascun utente.

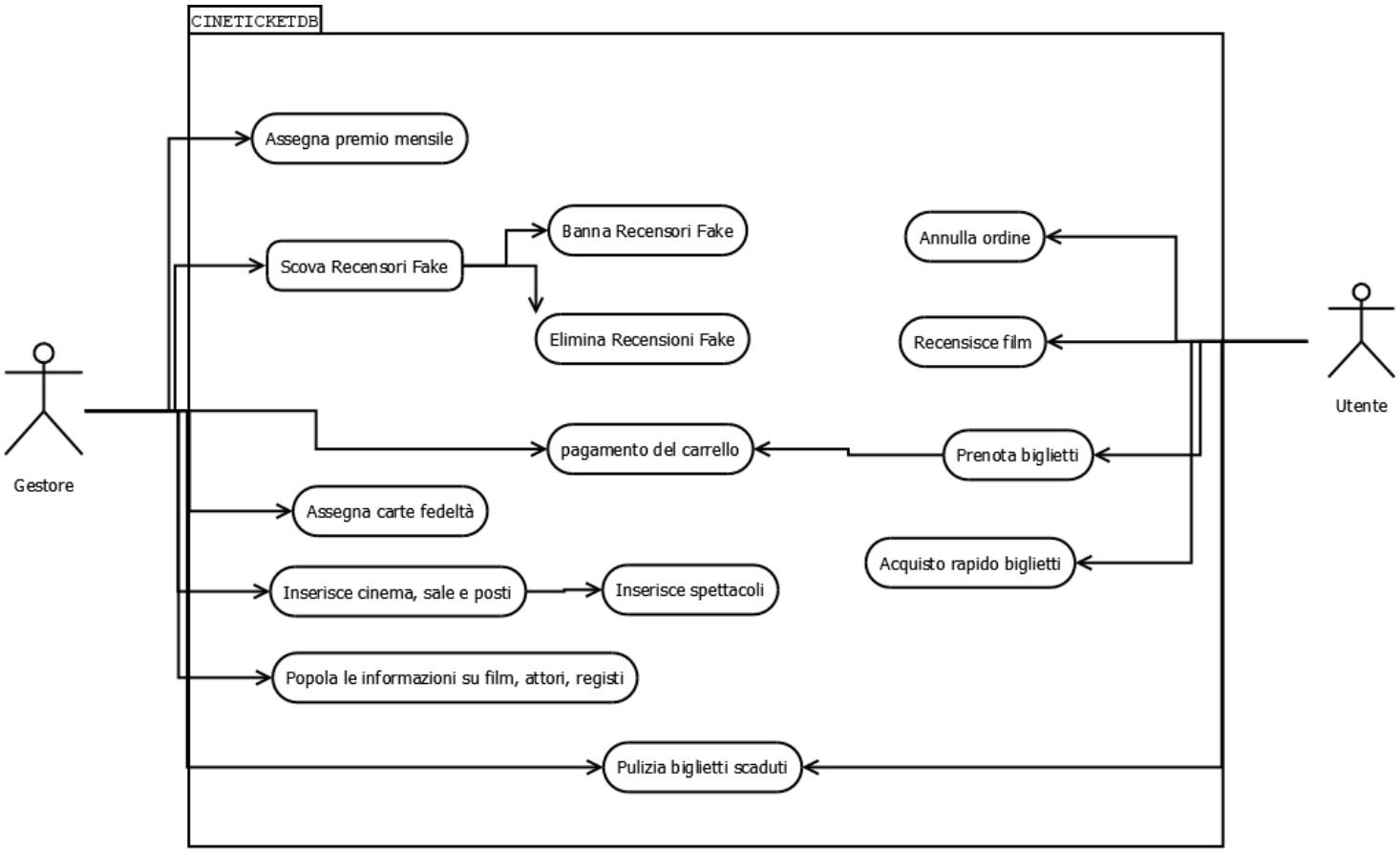
Utente	**Tipo**	**Permessi**
CineTicketDB	Amministratore	ALL
Gestore	Comune	ALL ON Utente ALL ON Promozione ALL ON Promozione_fedelta ALL ON Carrello ALL ON Biglietto SELECT ON Recensione DELETE ON Recensione ALL ON Spettacolo ALL ON Sala ALL ON Cinema ALL ON Posto ALL ON Film ALL ON Genere_film ALL ON Genere ALL ON Diretto_da ALL ON Recitato_da ALL ON Professionista ALL ON Compagnia_di_produzione EXECUTE update_data_pagamento EXECUTE cliente_del_mese EXECUTE inserisci_posti EXECUTE annulla_ordine EXECUTE FakeUser EXECUTE Assegna_carta_fedelta EXECUTE Elimina_biglietti_scaduti
User	Comune	INSERT ON Utente INSERT ON Carrello INSERT ON Biglietto INSERT ON Carta_di_credito DELETE ON Carta_di_credito ALL ON Recensione SELECT ON Spettacolo SELECT ON SalaIMAX3D SELECT ON MediaRecensione SELECT ON RegistaAttore SELECT ON VistaFilm SELECT ON Orari_spettacoli EXECUTE update_data_pagamento EXECUTE annulla_ordine EXECUTE auto_acquista EXECUTE Elimina_biglietti_scaduti

Operazioni di base e degli utenti.

Una volta stabilito il nostro schema concettuale è opportuno elencare tutte le operazioni di base degli utenti che verranno effettuate sulla nostra base di dati. Le operazioni di base in questo caso sono gli inserimenti. Le operazioni degli utenti come da requisito coinvolgono almeno tre tabelle. Le operazioni degli utenti o procedure sono:

1) Il gestore inserisce un cinema e lo popola in base ai dati della capienza: sarà necessario specificare il numero massimo dei posti e quanti posti sono popolabili per fila.
2) Il gestore annulla o rimborsa l'ordine effettuato se si è entro i limiti consentiti: il richiedente dell'annullamento/rimborso deve effettuare la richiesta entro il giorno precedente allo spettacolo.
3) L'utente può effettuare in modo rapido l'acquisto di biglietti multipli.
4) Il gestore determina se l'utente specificato fa parte della categoria di "recensori fake". Un recensore è "fake" se pubblica spesso recensioni troppo positive rispetto alla media, se acquista molte volte biglietti per lo stesso film e se recensisce solo i film di una compagnia determinata. Sarà cura del gestore decidere se bloccare l'utente dal sistema e se cancellare tutte le sue recensioni.
5) Il gestore tramite lo scheduler può assegnare carte fedeltà agli utenti che rispettano determinate condizioni: l'utente deve essere maggiorenne, deve essere sprovvisto di carta fedeltà, oppure possederne una scaduta, deve aver acquistato almeno 20 biglietti nel corso dell'ultimo mese, deve aver visto almeno 5 Film diversi l'uno dall'altro nel corso delle ultime tre settimane, deve aver pubblicato almeno la metà delle recensioni possibili.
6) Il gestore può assegnare due biglietti omaggio all'utente che ha fatto più acquisti nell'ultimo mese.
7) L'utente e/o il gestore aggiorna la data del pagamento del carrello preso in input se i biglietti associati allo spettacolo sono ancora validi e quindi lo spettacolo ancora non è iniziato. Anche il gestore può aggiornare il pagamento nell'eventualità che quest'ultimo sia effettutato alla "cassa".
8) L'utente e/o il gestore manualmente possono eliminare tutti i biglietti associati a spettacoli scaduti e non acquistati all'interno del carrello.

Schema Use Case



Descrittore di Operazioni

OPERAZIONE	**elimina_biglietti_scaduti**
SCOPO:	**eliminare biglietti scaduti dal carrello**
ARGOMENTI:	**codice carrello**
RISULTATO:	**eliminazione dei biglietti scaduti**
ERRORI:	**se il carrello è vuoto e non ci sono biglietti**
USA:	**biglietto**
MODIFICA:	**biglietto**
PRIMA:	**ci sono biglietti scaduti**
POI:	**non ci sono più biglietti scaduti**

OPERAZIONE	update_data_pagamento
SCOPO:	aggiorna data pagamento nel carrello
ARGOMENTI:	codice carrello, data pagamento
RISULTATO:	aggiornamento della data di pagamento
ERRORI:	data di sistema non è coerente
USA:	biglietto, carrello
MODIFICA:	carrello
PRIMA:	la data di pagamento deve essere aggiornata
POI:	la data di pagamento è stata aggiornata

OPERAZIONE	inserisci_posti
SCOPO:	inserimento del cinema e popolamento di esso
ARGOMENTI:	Nome, Citta, Via, CAP, totale_sale, totale_posti, massimo_fila
RISULTATO:	inserimento di un intero cinema popolato
ERRORI:	totale sale massimo di 99, totale posti minino di 1, massimo_fila minore di 1
USA:	cinema, sala, posti
MODIFICA:	cinema, sala, posti
PRIMA:	…
POI:	è stato inserito e popolato un cinema in modo automatico

OPERAZIONE	annulla_ordine
SCOPO:	annullare o rimborsare ordine effettuato
ARGOMENTI:	codice carrello, data eliminazione
RISULTATO:	annullamento/rimborso del biglietto acquisto dall'utente
ERRORI:	i biglietti non annullabili (se sono presenti delle recensioni) sono maggiori di 0 allora l'ordine non è annullabile. è scaduto il tempo l'ordine non è annullabile.
USA:	biglietto, recensione, carrello
MODIFICA:	biglietto, carrello
PRIMA:	L'ordine non è stato pagato, per cui è annullabile
POI:	L'ordine è stato annullato

OPERAZIONE	auto_acquista
SCOPO:	Consentire l'acquisto di biglietti multipli
ARGOMENTI:	Username, numero biglietti, data acquisto, numero carta
RISULTATO:	Acquisto di biglietti multipli da parte dell'utente
ERRORI:	il numero di biglietti è minore di 1 esso non è valido. i posti sono terminati non è possibile acquistare biglietti.
USA:	biglietto, carrello, posto, spettacolo
MODIFICA:	biglietto, carrello
PRIMA:	Nulla
POI:	L'utente ha acquistato uno o più biglietti in modo rapido

OPERAZIONE	FakeUser
SCOPO:	Controllare gli utente se sono recensori "fake"
ARGOMENTI:	Username, elimina recensioni, blocca utente
RISULTATO:	Scova i recensori "fake"
ERRORI:	l'utente non ha recensito il film più volte della media. l'utente non ha dato solo massimi voti per un film. l'utente non ha recensito il film ogni volta che è andato a vederlo l'utente non ha recensito più volte il film di un'unica compagnia
USA:	biglietto, recensione, film, carrello, spettacolo
MODIFICA:	recensione, utente
PRIMA:	Un utente poteva essere "fake"
POI:	Se il gestore decide di bloccare l'utente "fake", l'utente fake viene bloccato; se il gestore decide di eliminare le recensioni dell'utente "fake", le sue recensioni vengono eliminate; se l'utente non è "fake" o se il gestore non specifica nulla, allora non accade nulla.

OPERAZIONE	Assegna_carta_fedelta
SCOPO:	Assegnare carta fedeltà agli utenti
ARGOMENTI:	-
RISULTATO:	Assegna la carta fedeltà agli utenti che rispettano determinate condizioni
ERRORI:	L'utente possiede già una carta fedeltà. L'utente ha acquistato almeno 20 biglietti nell'ultimo mese L'utente ha visto almeno 5 film nel corso delle ultime tre settimane L'utente ha pubblicato almeno la metà delle recensioni pubblicabili
USA:	biglietto, carrello, spettacolo, recensione, carta_fedelta, utente
MODIFICA:	carta_fedelta
PRIMA:	Le carte fedeltà non venivano assegnate automaticamente
POI:	Le carte fedeltà vengono assegnate automaticamente

OPERAZIONE	Cliente_del_mese
SCOPO:	Assegnare due biglietti omaggio all'utente che ha fatto più acquisti nell'ultimo mese.
ARGOMENTI:	-
RISULTATO:	Inserimenti biglietti omaggio nel carrello vincente
ERRORI:	Non esiste utente vincente.
USA:	carrello, biglietto, spettacolo, posto, utente
MODIFICA:	carrello, biglietto
PRIMA:	Non esiste l'utente vincente
POI:	Esiste l'utente vincente

TUTTE LE OPERAZIONE CITATE VERRANNO SUCCESSIVAMENTE APPROFONDITE NEL CAPITOLO RIGUARDANTE LE IMPLEMENTAZIONI.

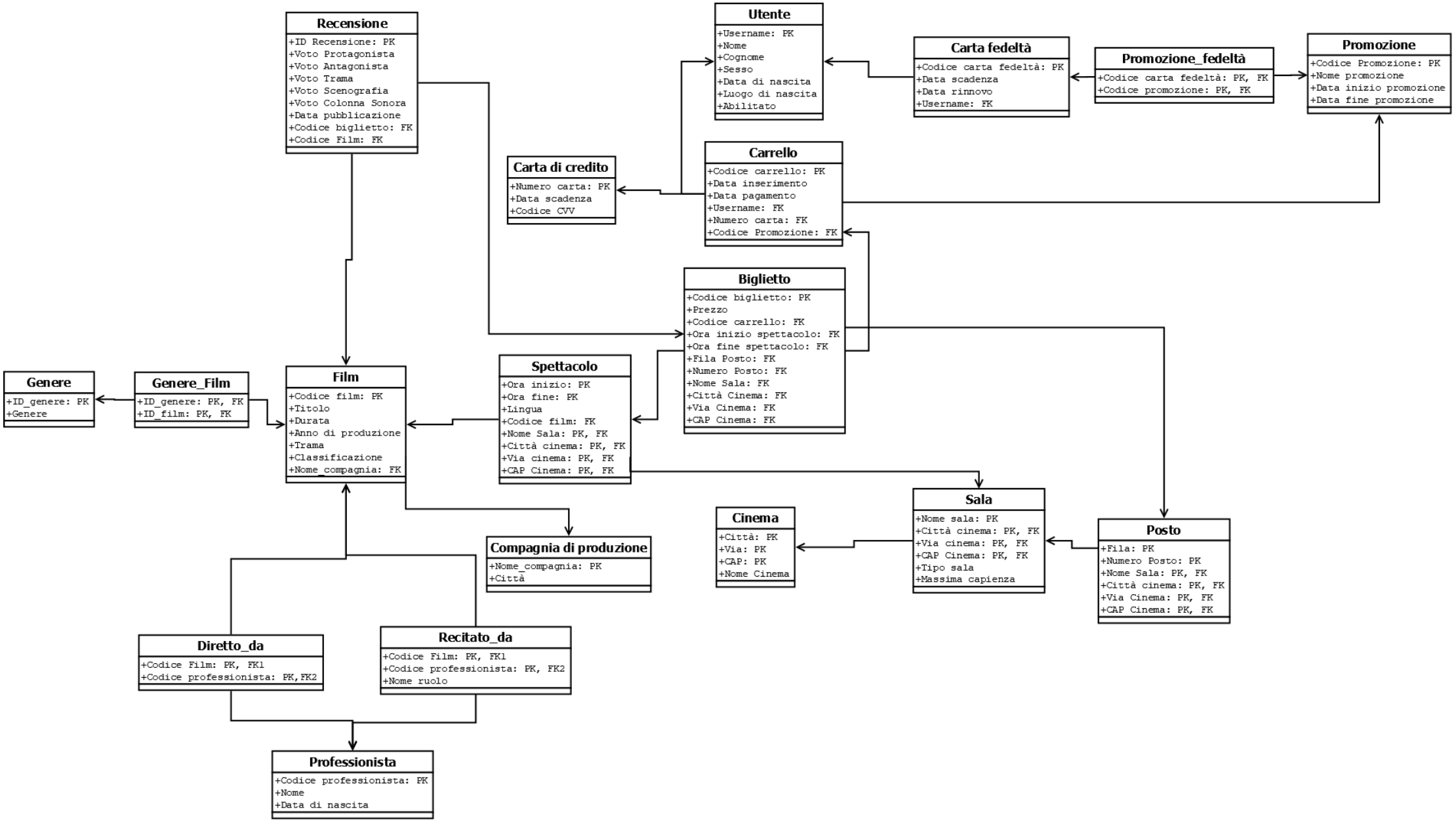
Traduzione in schema relazionale

Per procede con la realizzazione della nostra base di dati è necessario tradurre il nostro schema concettuale in schema relazionale. La traduzione comporta il passaggio delle entità in tabelle o relazioni, bisogna poi definire: vincoli, campi chiave, attributi composti, i vari tipi di unione e gli attributi multivalore. È necessario seguire le regole di traduzione del modello relazionale. Lo schema concettuale della nostra base di dati produce il seguente schema relazionale:

- Tabella **UTENTE** (Username: char<<PK>>, Nome: varchar, Cognome: varchar, Sesso: char, Data di nascita: date, Luogo di nascita: varchar, Abilitato: char) – Vincoli: Nome(NOT NULL), Cognome(NOT NULL), Sesso(unico carattere: M o F), Abilitato(unico carattere: V o F);
- Tabella **CARTA FEDELTÀ** (Codice carta fedeltà: char<<PK>>, Data scadenza: date, Data rinnovo: date, Username: char<<FK(UTENTE)>>) – Vincoli: Username(NOT NULL);
- Tabella **PROMOZIONE** (Codice promozione: char<<PK>>, Nome promozione: varchar, Data inizio promozione: date, Data fine promozione: date) – Vincoli: Data inizio promozione(NOT NULL), Data fine promozione(NOT NULL);
- Tabella **PROMOZIONE_FEDELTÀ** (Codice carta fedeltà: char<<PK>><<FK(Carta Fedeltà)>>, Codice promozione: char<<PK>><<FK(Promozione)>>);
- Tabella **CARTA DI CREDITO** (Numero carta: varchar<<PK>>, Data scadenza: date, CVV: char) – Vincoli: Data scadenza(NOT NULL), CVV(NOT NULL);
- Tabella **CARRELLO** (Codice carrello: char<<PK>>, Data inserimento: date, Data pagamento: date, Username: char<<FK(Utente)>>, Numero carta: varchar<<FK(Carta di credito)>>, Codice Promozione: char<<FK(Promozione)>>) – Vincoli: Data inserimento(NOT NULL), Username(NOT NULL);
- Tabella **BIGLIETTO** (Codice biglietto: char<<PK>>, Prezzo: number, Codice carrello: char<<FK(Carrello)>>, Ora inizio spettacolo: date <<FK(Spettacolo)>>, Ora fine spettacolo: date <<FK(Spettacolo)>>, Fila posto: char <<FK(Posto)>>, Numero posto: number <<FK(Posto)>>, Nome sala: char<<FK(Sala)>>, Citta cinema: varchar<<FK(Cinema)>>, Via cinema: varchar<<FK(Cinema)>>, CAP cinema: char<<FK(Cinema)>>) – Vincoli: Codice carrello(NOT NULL), Ora inizio spettacolo(NOT NULL), Ora fine spettacolo(NOT NULL), Fila posto(NOT NULL), Numero posto(NOT NULL), Nome sala(NOT NULL), Citta cinema(NOT NULL), Via cinema(NOT NULL), CAP cinema(NOT NULL);
- Tabella **SPETTACOLO** (Ora inizio: date<<PK>>, Ora fine: date<<PK>>, Lingua: varchar, Codice film: char<<FK(Film)>>, Nome sala: char<<PK>><<FK(Sala)>>, Citta cinema: varchar<<PK>><FK(Cinema)>>, Via cinema: varchar<<PK>><<FK(Cinema)>>, CAP cinema: char<<PK>><<FK(Cinema)>>);
- Tabella **CINEMA** (Citta: varchar<<PK>>, Via: varchar<<PK>>, CAP: char<<PK>>, Nome cinema: varchar) – Vincoli: Nome cinema(NOT NULL);
- Tabella **SALA** (Nome sala: char<<PK>>, Citta cinema: varchar<<PK>><<FK(Cinema)>>, Via cinema: varchar<<PK>><<FK(Cinema)>>, CAP cinema: char<<PK>><<FK(Cinema)>>, Tipo sala: varchar, Massima capienza: number) – Vincoli: Tipo sala(NOT NULL), Massima capienza(NOT NULL);

- Tabella **POSTO** (Fila: char<<PK>>, Numero posto: number<<PK>>, Nome sala: char<<PK>><<FK(Sala)>>, Citta cinema: varchar<<PK>><<FK(Cinema)>>, Via cinema: varchar<<PK>><<FK(Cinema)>>, CAP cinema: char<<PK>><<FK(Cinema)>>);
- Tabella **COMPAGNIA DI PRODUZIONE** (Nome_compagnia: varchar<<PK>>, Citta: varchar) – Vincoli: Citta(NOT NULL);
- Tabella **GENERE** (ID_genere: char<<PK>>, Genere: varchar) – Vincoli: Genere(NOT NULL);
- Tabella **FILM** (Codice film: char<<PK>>, Titolo: varchar, Durata: number, Anno di produzione: number, Nome_compagnia: varchar<<FK(Compagnia di produzione)>>, Classificazione: number) – Vincoli: Titolo(NOT NULL), Durata(NOT NULL), Anno di produzione(NOT NULL), Nome_compagnia(NOT NULL);
- Tabella **GENERE_FILM** (Id_genere: char<<PK>><<FK(Genere)>>, Id_film: char<<PK>><<FK(Film)>>);
- Tabella **RECENSIONE** (Id_recensione: char<<PK>>, Voto Protagonista: number, Voto Antagonista: number, Voto Trama: number, Voto Colonna Sonora: number, Voto Scenografia: number, Data pubblicazione: date, Codice biglietto: char<<FK(Biglietto)>>, Codice film: char<<FK(Film)>>) – Vincoli: Voto Protagonista(NOT NULL, minore o uguale di 10), Voto Antagonista(NOT NULL, minore o uguale di 10), Voto Trama(NOT NULL, minore o uguale di 10), Voto Colonna Sonora(NOT NULL, minore o uguale di 10), Voto Scenografia(NOT NULL, minore o uguale di 10), Data pubblicazione(NOT NULL), Codice biglietto(NOT NULL);
- Tabella **PROFESSIONISTA** (Codice professionista: char<<PK>>, Nome: varchar, Data di nascita: date) – Vincoli: Nome(NOT NULL);
- Tabella **DIRETTO_DA** (Codice film: char<<PK>><<FK(Film)>>, Codice professionista: char<<PK>><<FK(Professionista)>>);
- Tabella **RECITATO_DA** (Codice film: char<<PK>><<FK(Film)>>, Codice professionista: char<<PK>><<FK(Professionista)>>, Nome ruolo: varchar) – Vincoli: Nome ruolo(NOT NULL);

Schema Relazionale



Verifica di normalità dello schema relazionale

Nella verifica di normalità si combatte la ridondanza attraverso la caccia alle dipendenze funzionali anomale. E' necessario rispettare le tre forme normali per assicurarsi che non ci siano anomalie, anche se in alcuni casi è non è obbligatorio: per ragioni di prestazioni a volte è preferibile tenere una forma normale più bassa.

Prima forma normale

Una tabella si dice in prima forma normale quando:

- Tutte le sue righe hanno lo stesso numero di attributi
- Una colonna contiene valori tutti dello stesso tipo
- Non esistono due righe uguali
- L'ordine di inserimento non conta
- E soprattutto: quando non esistono attributi ripetuti o composti (cioè, i valori in un dominio devono essere atomici)

Nel nostro caso il DataBase rispetta la prima forma normale, poiché non presenta in alcuna entità gruppi di attributi che si ripetono tra loro e poiché ciascuna tabella presenta una chiave primaria che identifica in modo univoco ogni tupla della relazione. I vari campi data e ora sono tipi primitivi nella maggior parte dei linguaggi di programmazione, e pur essendo dati strutturati sono convenzionalmente considerati atomici; vale lo stesso per il campo data e ora di Oracle DBMS.

Seconda forma normale

Una tabella si dice in seconda forma normale quando:

- Rispetta già la prima forma normale
- Tutti gli attributi non appartenenti alla chiave dipendono dall'intera chiave (e non da una parte di essa)

Nel nostro caso le varie relazioni (come nel caso delle entità deboli Spettacolo, Sala e Posto) sono state suddivise ed è stata definita una nuova attinenza per ogni chiave parziale e i suoi corrispondenti attributi dipendenti. Dunque, ogni relazione contiene sia la chiave primaria iniziale, sia tutti gli attributi funzionalmente dipendenti da essa in modo completo.

Terza forma normale

Una tabella si dice in terza forma normale quando:

- Rispetta già la seconda forma normale (e quindi anche la prima)
- Tutti gli attributi non appartenenti alla chiave (o non-chiave) dipendono solo dalla chiave, ossia non esistono attributi che dipendono da altri attributi non-chiave

Nel nostro Database è rispettata anche questa condizione, poiché in nessun caso un attributo (o gli alttributi) non-chiave dipende da un altro attributo (o altri attributi) non-chiave.

Forma normale di Boys e Codd

Una relazione R si dice che rispetta la BCNF se e solo se:

- E' in terza forma normale
- Per ogni dipendenza funzionale non banale, X è una superchiave della relazione R

Ogni relazione in BCNF è anche in 3FN, mentre una relazione in 3FN non è necessariamente in BCNF. Per cui non è possibile garantire sempre il raggiungimento di questa forma normale e il mancato raggiungimento di questo obiettivo è indice che la base di dati è affetta da un'anomalia di cancellazione.

Nel nostro database, ciascuna dipendenza anomala (ad esempio la presenza della chiave esterna "codice_film" nell'entità "Recensione") è controllata tramite vincoli di tupla statici e dinamici (come nel Trigger "controllo_recensione", viene controllato se "codice_film" corrisponde allo stesso film a cui l'utente ha accesso con l'acquisto del biglietto).
Sia la terza forma normale che la BCNF sono rispettate.

Implementazione

Completata la progettazione della nostra base di dati, convertiremo le specifiche in un codice eseguibile. Il codice si riferisce al DBMS Oracle e il linguaggio adottato è il PL/SQL.

Creazione utenti

È necessario accedere al DBMS come amministratore del sistema e creare l'utente proprietario della base di dati. È possibile creare altri utenti ma il nostro schema comprende l'amministratore e l'utente che dispone di alcuni privilegi

```
CREATE USER amministratore IDENTIFIED BY admin;
CREATE USER lavoratore IDENTIFIED BY gestore;
CREATE USER spettatore IDENTIFIED BY user;
GRANT ALL PRIVILEGES TO amministratore;
```

L'amministratore della basi di dati avrà tutti i permessi per creare tutti gli oggetti che occorrono nella basi di dati. È necessario disconnettersi e poi riconnettersi con le credenziali dell'utente amministratore.

Creazione delle tabelle (Data Definition Language)

Il data definition Language (DDL) è il riflesso dello schema relazionale, tutte le tabelle che si trovano nello schema relazionale vengono scritte e create in SQL tramite il comando CREATE TABLE che includono tutti i vincoli di integrità.

```
create table Utente (
    Username char(10) constraint utente_pk primary key,
    Nome varchar(20) not null,
    Cognome varchar(20) not null,
    Sesso char(1) check (Sesso IN ('M','F')),
    Data_di_nascita date,
    Luogo_di_nascita varchar2(20),
    abilitato char(1) default 'V' check (abilitato IN ('V','F'))
);

create table Carta_fedelta (
    Codice_carta_fedelta char(10) constraint carta_fedelta_pk primary key,
    Data_scadenza date,
    Data_rinnovo date,
    Username char(10) not null,
    CONSTRAINT fk_Carta_fedelta_username FOREIGN KEY (USERNAME) REFERENCES UTENTE(USERNAME),
    CONSTRAINT uq_carta_fedelta_username UNIQUE (Username)
);

create table Promozione (
```

```
    Codice_promozione char(10) constraint promozione_pk primary key,
    Nome_promozione varchar(20),
    Data_inizio_promozione date not null,
    Data_fine_promozione date not null

);

create table Promozione_fedelta (
    Codice_carta_fedelta char(10),
    Codice_promozione char(10),
    constraint promozione_fedelta_pk primary key (Codice_carta_fedelta, Codice_promozione),
    constraint fk_promozionefedelta_codicecartafedelta foreign key (codice_carta_fedelta) references
Carta_fedelta(codice_carta_fedelta) on delete cascade,
        constraint fk_promozionefedelta_codicepromozione foreign key (codice_promozione) references
        Promozione(codice_promozione) on delete cascade
);


create table Carta_di_credito (
    Numero_carta varchar(16) constraint cartadicredito_pk primary key,
    Data_scadenza date not null,
    CVV char(3) not null
);

create table Carrello (
    Codice_carrello char(10) constraint carrello_pk primary key,
    Data_inserimento date not null,
    Data_pagamento date,
    Username char(10) not null,
    Numero_carta varchar(16) not null,
    Codice_promozione char(10),
    constraint fk_carrello_username foreign key (Username) references utente(username),
    constraint fk_carrello_numerocarta foreign key (numero_carta) references carta_di_credito(numero_carta),
    constraint fk_carrello_codicepromozione foreign key (codice_promozione) references
promozione(codice_promozione)
);

create table Biglietto (
    Codice_biglietto char(10) constraint biglietto_pk primary key,
    Prezzo number,
    Codice_carrello char(10) not null,
    Ora_inizio_spettacolo date not null,
    Ora_fine_spettacolo date not null,
    Fila_posto char(1) not null,
    Numero_posto number not null,
    Nome_sala char(2) not null,
    Citta_cinema varchar(20) not null,
    Via_cinema varchar(50) not null,
    CAP_cinema char(5) not null
);
```

```
create table Spettacolo (
    Ora_inizio date,
    Ora_fine date,
    Lingua varchar(20),
    Codice_film char(10),
    Nome_sala char(2),
    Citta_cinema varchar(20),
    Via_cinema varchar(50),
    CAP_cinema char(5),
    constraint spettacolo_pk primary key (Ora_inizio, Ora_fine, Nome_sala, Citta_cinema, Via_cinema,
CAP_cinema)
);


alter table Biglietto add constraint fk_biglietto_spettacolo foreign key (Ora_inizio_spettacolo,
Ora_fine_spettacolo,Nome_sala,Citta_cinema,via_cinema,CAP_cinema) references
Spettacolo(Ora_inizio,Ora_fine,Nome_sala,Citta_cinema,via_cinema,CAP_cinema);


create table Cinema (
    Citta varchar(20),
    Via varchar(50),
    CAP char(5),
    Nome_cinema varchar(50) not null,
    constraint cinema_pk primary key (Citta, Via, CAP)
);

create table Sala (
    Nome_sala char(2),
    Citta_cinema varchar(20),
    Via_cinema varchar(50),
    CAP_cinema char(5),
    Tipo_sala varchar(10) not null,
    Massima_capienza number not null,
    constraint sala_pk primary key (Nome_sala, Citta_cinema, Via_cinema, CAP_cinema),
    constraint fk_sala_indirizzocinema foreign key (Citta_cinema, Via_cinema, CAP_cinema) references
Cinema(Citta, Via, CAP)
);

alter table Spettacolo add constraint fk_spettacolo_sala foreign key (Citta_cinema, Via_cinema, CAP_cinema,
Nome_sala) references Sala(Citta_cinema, Via_cinema, CAP_cinema, Nome_sala);


create table Posto (
    Fila char(1),
    Numero_posto number,
    Nome_sala char(2),
    Citta_cinema varchar(20),
    Via_cinema varchar(50),
    CAP_cinema char(5),
    constraint posto_pk primary key (Fila, Numero_posto, Nome_sala, Citta_cinema, Via_cinema, CAP_cinema),
```

```
    constraint fk_posto foreign key (Nome_sala, Citta_cinema, Via_cinema, CAP_cinema) references
Sala(Nome_sala, Citta_cinema, Via_cinema, CAP_cinema)
);

alter table Biglietto add constraint fk_biglietto_posto foreign key (Fila_posto, Numero_posto, Nome_sala,
Citta_cinema, Via_cinema, CAP_cinema) references Posto(Fila, Numero_posto, Nome_sala, Citta_cinema,
Via_cinema, CAP_cinema);


create table Compagnia_di_produzione (
    Nome_compagnia varchar(20) constraint compagnia_di_produzione_pk primary key,
    Citta varchar(20) not null
);


create table Genere (
    ID_genere char(3) constraint genere_pk primary key,
    Genere varchar(40) not null
);

create table Film (
    Codice_film char(10) constraint film_pk primary key,
    Titolo varchar(50) not null,
    Durata number not null,
    Anno_di_produzione number not null,
    Nome_compagnia varchar(20) not null,
    Classificazione number,
    constraint fk_film_nomecompagnia foreign key (nome_compagnia) references
Compagnia_di_produzione(nome_compagnia)
);

alter table Spettacolo add constraint fk_spettacolo_codicefilm foreign key (Codice_film) references
Film(Codice_film);

create table Genere_film (
    ID_genere char(3),
    ID_film char(10),
    constraint generefilm_pk primary key (id_genere, id_film),
    constraint fk_generefilm_idfilm foreign key (id_film) references Film(codice_film),
    constraint fk_generefilm_idgenere foreign key (id_genere) references Genere(id_genere)
);


create table Recensione (
    ID_recensione char(5) constraint recensione_pk primary key,
    VotoProtagonista number not null, CHECK (VotoProtagonista<=10),
    VotoAntagonista number not null, CHECK (VotoAntagonista<=10),
    VotoTrama number not null, CHECK (VotoTrama<=10),
    VotoColonnaSonora number not null, CHECK (VotoColonnaSonora<=10),
    VotoScenografia number not null, CHECK (VotoScenografia<=10),
    Data_pubblicazione date not null,
```

```
    Codice_biglietto char(10) not null,
    Codice_film char(10),
    constraint fk_recensione_codicebiglietto foreign key (codice_biglietto) references
Biglietto(codice_biglietto),
    constraint fk_recensione_codicefilm foreign key (codice_film) references Film(codice_film),
      constraint uq_recensione UNIQUE (codice_biglietto, codice_film)
);


create table Professionista (
    Codice_professionista char(5) constraint professionista_pk primary key,
    Nome varchar(40) not null,
    Data_di_nascita date
);


create table Diretto_da (
    Codice_film char(10),
    Codice_professionista char(5),
    constraint diretto_da_pk primary key (Codice_film, Codice_professionista),
    constraint fk_direttoda_codicefilm foreign key (codice_film) references Film(codice_film),
        constraint fk_direttoda_codiceprofessionista foreign key (codice_professionista) references
        Professionista(codice_professionista)
);


create table Recitato_da (
    Codice_film char(10),
    Codice_professionista char(5),
    Nome_ruolo varchar(40) not null,
    constraint recitato_da_pk primary key (Codice_film, Codice_professionista),
    constraint fk_recitatoda_codicefilm foreign key (codice_film) references Film(codice_film),
    constraint fk_recitatoda_codiceprofessionista foreign key (codice_professionista) references
Professionista(codice_professionista)
);
```

Vincoli di integrità statici

Definiamo i vincoli di integrità della nostra specifica base di dati, ci sono vincoli di integrità statici che sono definiti all'inizio della nostra base di dati e consentono un corretto inserimento dei dati all'interno delle tabelle e poi ci sono i vincoli di integrità dinamici che sono implementati con i trigger.
I vincoli di integrità statici sono:

- Gli attributi nome, cognome nell'entità Utente non devono avere valori nulli
- L'attributo Sesso nell'entità Utente può avere un solo carattere (M o F)
- L'attributo abilitato nell'entità Utente può avere un solo carattere (V o F)
- L'attributo Username nell'entità Carta Fedeltà non deve avere valore nullo
- Gli attributi Data_inizio_promozione e Data_fine_promozione nell'entità Promozione non devono avere valori nulli
- Gli attributi Data_scadenza e CVV nell'entità Carta di credito non devono avere valori nulli
- Gli attributi Data_inserimento, Username, Numero_carta nell'entità Carrello non devono avere valori nulli
- Gli attributi Codice_carrello, Ora_inizio_spettacolo, Ora_fine_spettacolo, Fila_posto, Numero_posto, Nome_sala, Citta_cinema, Via_cinema, CAP_cinema nell'entità Biglietto non devono avere valori nulli
- L'attributo Nome_cinema nell'entità Cinema non deve avere valore nullo
- Gli attributi Tipo_sala, Massima_capienza nell'entità Sala non devono avere valori nulli
- L'attributo Citta nell'entità Compagnia di produzione non deve avere valore nullo
- L'attributo Genere nell'entità Genere non deve avere valore nullo
- Gli attributi Titolo, Durata, Anno_di_produzione, Nome_compagnia nell'entità Film non devono avere valori nulli
- Gli attributi VotoProtagonista, VotoAntagonista, VotoTrama, VotoColonnaSonora, VotoScenografia dell'entità Recensione non devono avere valori nulli e non devono avere valori superiori a 10
- Gli attributi Data_pubblicazione, Codice_biglietto dell'entità Recensione non devono avere valori nulli
- L'attributo Nome dell'entità Professionista non deve avere valore nullo
- L'attributo Nome_ruolo dell'associazione Recitato_da con molteplicità M a N non deve avere valore nullo

I vincoli di integrità dinamici sono implementati con i trigger, definiamo quindi cos'è un trigger.

Definizione dei Trigger (vincoli di integrità dinamici)

I trigger sono blocchi di istruzioni procedurali che vengono richiamati dal sistema quando un certo evento si verifica all'interno della tabella. Infatti, i trigger si basano sul modello ECA (Evento – Condizione – Azione).
Nel nostro caso è risultato opportuno istituire condizioni basate sulle regole della nostra ipotetica catena di cinema, come ad esempio la verifica dei codici promozionali in un carrello, il controllo delle sale e della capienza di ciascuna di essa, la verifica del biglietto e del suo costo insieme a tutte le possibili sfaccettature.

TRIGGER promozione_scaduta

Il nostro primo Trigger è stato idealizzato allo scopo di controllare se i valori della carta fedeltà e della promozione inserita risultano essere scaduti.
Se il controllo non procede a buon fine allora il valore del codice promozionale viene impostato a NULL, facendo in modo che l'utente poi successivamente non potrà usufruire di alcuno sconto nel prezzo del primo biglietto inserito nel carrello.

```
CREATE OR REPLACE TRIGGER Promozione_scaduta
BEFORE INSERT OR UPDATE OF Codice_promozione ON Carrello
FOR EACH ROW
DECLARE
   conteggio1 NUMBER;
   conteggio2 NUMBER;
   conteggio3 NUMBER;
BEGIN
   SELECT COUNT(*) INTO conteggio1 FROM promozione PRO WHERE PRO.Data_fine_promozione <
:NEW.Data_inserimento AND PRO.Codice_promozione = :NEW.Codice_promozione;
   SELECT COUNT(*) INTO conteggio2 FROM Carta_fedelta CAR WHERE CAR.Data_scadenza <
:NEW.Data_inserimento AND CAR.Username = :NEW.Username;
   SELECT COUNT(*) INTO conteggio3 FROM Carta_fedelta car WHERE car.Username=:NEW.Username;
   IF conteggio1+conteggio2 > 0 OR conteggio3 = 0 THEN
      IF conteggio3 = 0 THEN
         dbms_output.put_line('L''utente ' || TRIM(:NEW.Username) || ' non possiede una carta fedelta''!');
      ELSIF conteggio2 > 0 THEN
         dbms_output.put_line('La carta fedelta'' dell''utente ' || TRIM(:NEW.Username) || ' e'' scaduta!');
      ELSIF conteggio1 > 0 THEN
         dbms_output.put_line('La promozione ''' || TRIM(:NEW.Codice_promozione) || ''' e'' scaduta!');
      END IF;
      :NEW.Codice_promozione := NULL;
   END IF;
END;
/
```

Supponiamo di essere l'amministratore del sistema allo scopo di effettuare un esempio pratico. Inseriamo i valori di una promozione all'interno della tabella "Promozione" e modifichiamo la carta fedeltà associata all'utente "Luke3012", rendendola inutilizzabile per la promo che si vuole aggiungere.

```
UPDATE Carta_fedelta
```

```
SET Data_scadenza = TO_DATE('10/06/2022','dd/mm/yyyy'), Data_rinnovo = TO_DATE('10/06/2021','dd/mm/yyyy')
WHERE Username = 'Luke3012';
insert into promozione values ('promo02', 'Clienti best vecchio', TO_DATE('01/06/2022', 'dd/mm/yyyy'), TO_DATE('15/06/2022', 'dd/mm/yyyy'));

insert into carrello values ('cart003', TO_DATE('16/06/2022 20:14', 'dd/mm/yyyy hh24:mi'), 'Luke3012', '1000735724964861', 'promo02');
insert into carrello values ('cart004', TO_DATE('16/06/2022 20:14', 'dd/mm/yyyy hh24:mi'), 'Luke3012', '1000735724964861', 'promo01');
```

Effettuando una SELECT su carrello, come ad esempio:

```
select * from carrello order by codice_carrello;
```

… osserviamo che per la tupla con codice_carrello “cart003” il valore “promo02” nella colonna codice_promozione non c’è poiché non è valida!

TRIGGER carta_credito_scaduta

Il Trigger “carta_credito_scaduta” effettua un’operazione semplice ma fondamentale: per accettarsi che il sistema di gestione dei pagamenti possa procedere con la verifica dei dati della carta, il trigger controlla se la carta associata al carrello è scaduta. Di certo quest’ultima dovrebbe essere un’operazione gestita da un applicativo esterno, che ci limiteremo ad immaginare; ma ai fini di “controllo” è utile gestire la situazione con un’ulteriore verifica per una miglior correttezza dei dati.

```
CREATE OR REPLACE TRIGGER carta_credito_scaduta
BEFORE INSERT OR UPDATE ON carrello
FOR EACH ROW
DECLARE
        carta_scaduta number;
        scaduta exception;
BEGIN
        SELECT count (*) into carta_scaduta
        from carta_di_credito car
        WHERE :new.Numero_carta = car.Numero_carta AND car.Data_scadenza < SYSDATE;
        IF inserting THEN
                IF carta_scaduta >= 1 THEN
                        RAISE scaduta;
                END IF;
        END IF;

        IF updating THEN
                if carta_scaduta >= 1 THEN
                        RAISE scaduta;
                end if;
        END IF;
exception
WHEN scaduta THEN
        raise_application_error('-20001','La carta associata al carrello e'' scaduta e non puo'' fare acquisti');
END;
/
```

TRIGGER autoPromo

Il Trigger "AutoPromo" si occupa di rendere automatica la procedura di assegnazione di una promozione ad un utente. Se durante l'inserimento della tupla nella tabella "carrello" il valore di "codice_promozione" è impostato a NULL, allora il Trigger in questione entra in azione scegliendo casualmente una delle promozioni associate alla carta fedeltà dell'utente, disponibili nell'entità "promozione_fedelta". Al contrario, se il valore non è NULL, allora il Trigger non verrà eseguito.

```
CREATE OR REPLACE TRIGGER AutoPromo
BEFORE INSERT ON Carrello
FOR EACH ROW
DECLARE
   promo CHAR(10);
   fedelta CHAR(10);
BEGIN
   IF :NEW.Codice_promozione IS NOT NULL THEN
      RETURN;
   END IF;

   SELECT ca.codice_carta_fedelta INTO fedelta FROM Carta_fedelta ca WHERE ca.Username=:NEW.username;

   SELECT pro.Codice_promozione INTO promo FROM Promozione_fedelta pf JOIN Promozione pro ON
pf.codice_promozione=pro.codice_promozione
   JOIN Carta_fedelta ca ON ca.codice_carta_fedelta=pf.codice_carta_fedelta
   WHERE ca.Codice_carta_fedelta=fedelta AND ca.Data_scadenza >= :NEW.Data_Inserimento AND
pro.Data_fine_promozione >= :NEW.Data_Inserimento
   ORDER BY DBMS_RANDOM.VALUE
   FETCH FIRST 1 ROWS ONLY;

   :NEW.Codice_promozione := promo;
   dbms_output.put_line('All''utente ' || TRIM(:NEW.Username) || ' e'' stata attribuita la promo ' ||
TRIM(promo));
EXCEPTION WHEN NO_DATA_FOUND THEN
   RETURN;
END;
/
```

TRIGGER controllo_data_pagamento

Il Trigger "controllo_data_pagamento" assicura che la data del pagamento sia settata a NULL durante l'inserimento della tupla.

```
CREATE OR REPLACE TRIGGER controllo_data_pagamento
BEFORE INSERT ON carrello
FOR EACH ROW
DECLARE
        temp number ;
        controllo exception;
```

```
BEGIN
        SELECT count(*) INTO temp
        FROM biglietto bg JOIN carrello cr ON bg.codice_carrello = :new.codice_carrello;

        if temp = 0 AND :new.data_pagamento IS NOT NULL then
           RAISE controllo;
        end if;

        EXCEPTION
        WHEN controllo THEN
           :NEW.data_pagamento := NULL;
           DBMS_OUTPUT.PUT_LINE('IL CARRELLO CON CODICE : ' || TRIM(:NEW.codice_carrello) || ' E''VUOTO
        LA DATA DEL PAGAMENTO VIENE SETTATA A NULL');
END;
/
```

TRIGGER verifica_cancellazione_carrello

Il Trigger "verifica_cancellazione_carrello" verifica che non ci siano dei biglietti associati al carrello prima della sua eliminazione. Questo TRIGGER è concettualmente banale ed è facilmente rimpiazzabile applicando una COSTRAINT alla chiave esterna "codice_carrello" che si trova in biglietto: ON DELETE CASCADE. Preferiamo gestirla in questo modo per un fattore estetico, ad esempio nel caso la procedura "annulla_ordine" dovesse fallire o nel caso un amministratore volesse annullare un intero ordine, capirebbe sin da subito il motivo della sua problematica.

```
CREATE OR REPLACE TRIGGER Verifica_cancellazione_carrello
BEFORE DELETE ON Carrello
FOR EACH ROW
DECLARE
   conta NUMBER;
BEGIN
   SELECT count(*) INTO conta FROM Biglietto bi
   WHERE bi.codice_carrello=:OLD.codice_carrello;

   IF conta > 0 THEN
      raise_application_error('-20030','Impossibile eliminare il carrello '''||TRIM(:OLD.codice_carrello)||''', ci sono
ancora '||conta||' biglietti associati ad esso.');
   END IF;
END;
/
```

TRIGGER utente_bannato_carrello

Il Trigger "utente_bannato_carrello" è l'ultimo trigger che è stato creato per l'entità carrello. Lo scopo è già intuibile dal titolo: se l'utente viene bannato (o bloccato) dal sistema, non potrà acquistare biglietti. Per cui, non potrà inserire elementi nel carrello e conseguentemente, non potrà neppure fare recensioni.

```
CREATE OR REPLACE TRIGGER Utente_bannato_carrello
BEFORE INSERT OR UPDATE ON Carrello
FOR EACH ROW
DECLARE
   banned CHAR(1);
BEGIN
   SELECT ut.abilitato INTO banned FROM Utente ut WHERE ut.Username=:NEW.Username;
   IF banned = 'F' THEN
      raise_application_error('-20050','Impossibile procedere, l''utente ' || TRIM(:NEW.Username) || ' è stato
bannato.');
   END IF;
END;
/
```

TRIGGER utente_bannato_fedelta

Terminati i vincoli dinamici sull'entità "Carrello", passiamo adesso ad un Trigger dedicato all'entità "Carta_fedelta". In modo analogo al Trigger precedente, se l'utente è bannato, allora non può rinnovare la propria carta fedeltà.

```
CREATE OR REPLACE TRIGGER Utente_bannato_fedelta
BEFORE INSERT OR UPDATE ON Carta_fedelta
FOR EACH ROW
DECLARE
   banned CHAR(1);
BEGIN
   SELECT ut.abilitato INTO banned FROM Utente ut WHERE ut.Username=:NEW.Username;
   IF banned = 'F' THEN
      raise_application_error('-20051','Impossibile procedere, l''utente ' || TRIM(:NEW.Username) || ' è stato
bannato.');
   END IF;
END;
/
```

TRIGGER sala_piena

Il Trigger "sala_piena" è un trigger dedicato all'impiegato lavoratore (gestore). Quest'ultimo potrebbe erroneamente inserire all'interno dell'entità "posto" una tupla in più rispetto a quelle necessarie. Per cui è opportuno effettuare un controllo sulla capienza, verificando se il totale delle tuple inserite corrisponde al valore di "massima_capienza" della tabella "sala".

```
CREATE OR REPLACE TRIGGER Sala_piena
BEFORE INSERT ON Posto
FOR EACH ROW
DECLARE
   contsala NUMBER;
   capienza NUMBER;
BEGIN
```

```
   SELECT Massima_capienza INTO capienza from sala sl where :new.nome_sala = sl.nome_sala and
:new.Citta_cinema = sl.Citta_cinema AND :new.Via_cinema = sl.Via_cinema AND :new.CAP_cinema =
sl.CAP_cinema;
   select count(*) INTO contsala from posto po where :new.nome_sala = po.nome_sala AND
:new.Citta_cinema = po.Citta_cinema AND :new.Via_cinema = po.Via_cinema AND :new.CAP_cinema =
po.CAP_cinema;
   IF contsala = capienza THEN
      raise_application_error(-20000, 'massima capienza raggiunta');
   END IF;
END;
/
```

TRIGGER biglietti_terminati (e FUNCTION verifica_posti)

Il Trigger "biglietti_terminati" non funzione senza la procedura "verifica_posti". Quest'ultima verifica se il numero di posti richiesto è disponibile per un potenziale acquisto nella sala specificata in "nome_sala": se il risultato dell'operazione sarà vero, allora la funzione ritorna il valore TRUE, altrimenti FALSE. Il vincolo d'integrità dinamico preso in questione non fa altro che richiamare la function: se i posti risultano essere occupati, allora il biglietto non potrà essere registrato nel database.

```
CREATE OR REPLACE FUNCTION verifica_posti (numero_richiesto NUMBER, Ora_inizio DATE, Ora_fine DATE,
Sala CHAR, Citta VARCHAR, Via VARCHAR, CAP CHAR)
RETURN BOOLEAN
IS
   numero_posti NUMBER;
   biglietti_venduti NUMBER;
BEGIN
   select count(*) into biglietti_venduti
   from biglietto bg
   where (bg.ora_inizio_spettacolo = ora_inizio) AND (bg.ora_fine_spettacolo = ora_fine)
   AND (bg.nome_sala = sala) AND (bg.citta_cinema = citta) AND (bg.cap_cinema = cap) AND (bg.via_cinema
= via);


   --E' possibile misurare la capienza in base al valore di "Massima_capienza" presente nell'entità Sala
   --select distinct sl.Massima_capienza into numero_posti
   --from spettacolo sp JOIN sala sl ON (sp.nome_sala = sl.nome_sala AND sp.Citta_cinema = sl.citta_cinema
AND sp.via_cinema = sl.via_cinema AND sp.CAP_cinema = sl.CAP_cinema)
   --where ora_inizio = sp.ora_inizio AND ora_fine = sp.ora_fine AND sala = sp.nome_sala
   --AND Citta=sp.citta_cinema AND CAP=sp.cap_cinema AND Via=sp.via_cinema;

   --Ma è preferibile misurare la capienza in base al numero dei posti effettivamente inseriti per evitare errori
   select count(*) into numero_posti from posto po
   where po.citta_cinema=citta and po.via_cinema=via and po.cap_cinema=cap and po.nome_sala=sala;

   if (biglietti_venduti+numero_richiesto) > numero_posti then
      RETURN FALSE;
   end if;
   RETURN TRUE;
```

```
END verifica_posti;
/


CREATE OR REPLACE TRIGGER biglietti_terminati
BEFORE INSERT ON biglietto
FOR EACH ROW
DECLARE
   errore exception;
BEGIN
   if verifica_posti(1, :New.Ora_inizio_spettacolo, :New.Ora_fine_spettacolo, :new.nome_sala, :new.citta_cinema,
:new.via_cinema, :new.cap_cinema) = FALSE THEN
      RAISE errore;
   end if;

EXCEPTION
WHEN errore then
   raise_application_error('-20010','Posti terminati! Biglietto non acquistabile.');
END;
/
```

TRIGGER limite_eta

Il Trigger "limite_eta" verifica se l'utente ha l'età giusta per poter acquistare il biglietto. Ciascun tupla in FILM contiene il valore "rating", ovvero un tipo NUMBER che indica l'età minima consentita per la visione del film; ciascun utente invece possiede il valore "data_di_nascita". Paragonando il valore di entrami i valori, è possibile determinare la soluzione del problema.

```
CREATE OR REPLACE TRIGGER Limite_eta
BEFORE INSERT ON Biglietto
FOR EACH ROW
DECLARE
   rating NUMBER;
   eta NUMBER;
BEGIN
   SELECT TRUNC((SYSDATE - Data_di_nascita)/ 365.25) INTO eta FROM (utente us JOIN carrello cr ON
us.username=cr.username) WHERE cr.codice_carrello=:NEW.codice_carrello;
   SELECT Classificazione INTO rating FROM (Spettacolo sp JOIN Film fi ON sp.Codice_film=fi.Codice_film)
   WHERE sp.ora_inizio=:NEW.ora_inizio_spettacolo AND sp.ora_fine=:NEW.ora_fine_spettacolo
   AND sp.nome_sala=:NEW.nome_sala AND sp.citta_cinema=:NEW.citta_cinema AND
sp.cap_cinema=:NEW.cap_cinema AND :NEW.via_cinema=sp.via_cinema;
   IF eta < rating THEN
      raise_application_error(-20000, 'L''utente non ha l''eta'' adatta per poter acquistare biglietti per questo
film!');
   END IF;
END;
/
```

TRIGGER verificaposto_spettacolo

Il Trigger "verificaposto_spettacolo" controlla l'eventuale esistenza dello spettacolo che si vuole integrare nel biglietto. Per fare ciò, viene sfruttata la FUNCTION "verifica_posto", la quale non è altro che una replica si "verifica_posti" pensata al singolare, cioè alla verifica dello specifico singolo posto occupato piuttosto che alla disponibilità dei multipli numeri di posti.

```
CREATE OR REPLACE FUNCTION verifica_posto (Ora_inizio DATE, Ora_fine DATE, Sala CHAR, Citta VARCHAR,
Via VARCHAR, CAP CHAR, Fila CHAR, Numero NUMBER)
RETURN BOOLEAN
IS
   conta_posto NUMBER;
BEGIN
   SELECT COUNT (*) INTO conta_posto FROM Biglietto bi WHERE bi.ora_inizio_spettacolo=ora_inizio AND
bi.ora_fine_spettacolo=ora_fine
   AND bi.nome_sala=sala AND bi.citta_cinema=citta AND bi.via_cinema=via AND bi.Cap_cinema=cap AND
bi.fila_posto=fila AND bi.numero_posto=numero;
   IF conta_posto = 1 THEN
      RETURN FALSE;
   END IF;
   RETURN TRUE;
END verifica_posto;
/

CREATE OR REPLACE TRIGGER VerificaPosto_Spettacolo
BEFORE INSERT ON Biglietto
FOR EACH ROW
DECLARE
   conta_spettacolo NUMBER;
   conta_posto NUMBER;
BEGIN
   SELECT COUNT(*) INTO conta_spettacolo FROM Spettacolo sp
   WHERE sp.ora_inizio=:NEW.ora_inizio_spettacolo AND sp.ora_fine=:NEW.ora_fine_spettacolo
   AND sp.nome_sala=:NEW.nome_sala AND sp.citta_cinema=:NEW.citta_cinema AND
sp.via_cinema=:NEW.via_cinema AND sp.cap_cinema=:NEW.cap_cinema;
   IF conta_spettacolo <> 1 THEN
      raise_application_error(-20099, 'Non esiste uno spettacolo con queste caratteristiche!');
   END IF;

   IF verifica_posto(:New.Ora_inizio_spettacolo, :New.Ora_fine_spettacolo, :new.nome_sala, :new.citta_cinema,
:new.via_cinema, :new.cap_cinema, :new.fila_posto, :new.numero_posto) = FALSE THEN
      raise_application_error(-20100, 'Questo posto ('||:NEW.fila_posto||:NEW.numero_posto||') risulta gia''
occupato!');
   END IF;
END;
/
```

TRIGGER prezzo_biglietto (e FUNCTION DeterminaPrezzo)

Il Trigger "prezzo_biglietto" determina automaticamente il prezzo del singolo biglietto che l'utente sta per acquistare. Tutto il lavoro sporco è fatto dalla Function "DeterminaPrezzo", la quale verifica

se l'utente possiede una carta fedeltà: in caso positivo, si verifica che il carrello attuale dell'utente non possiede già altri biglietti scontati e si procede allo sconto del prezzo (dimezzandolo); in caso negativo, cioè se l'utente non possiede una carta fedeltà o se ha già usufruito dello sconto, allora si determina il prezzo in base al giorno della settimana attuale (10€ se nei weekend, 8€ se in settimana).

```
CREATE OR REPLACE FUNCTION DeterminaPrezzo(codice_biglietto char, codice_carrell char, ora_inizio DATE)
RETURN NUMBER
IS
   costo NUMBER;
   conteggio NUMBER;
BEGIN
   IF to_char(ora_inizio, 'D') = '1' OR to_char(ora_inizio, 'D') = '7' THEN
      --Il costo il sabato e la domenica
      costo := 10;
   ELSE
      costo := 8;
   END IF;

   SELECT COUNT(*) INTO conteggio FROM Biglietto bi JOIN Carrello car ON
bi.codice_carrello=car.codice_carrello
   WHERE bi.codice_carrello=codice_carrell
   AND EXISTS (SELECT NULL FROM Carta_fedelta ca WHERE ca.Username=car.username AND
ca.Data_scadenza>=ora_inizio);

   IF conteggio = 0 THEN
      --Se l'utente ha una carta fedeltà e non ha già acquistato un biglietto scontato, applica lo sconto
      dbms_output.put_line('Al biglietto ''' || TRIM(codice_biglietto) || ''' e'' stato applicato un buono sconto.');
      costo := costo/2;
   END IF;

   RETURN costo;
END DeterminaPrezzo;
/

CREATE OR REPLACE TRIGGER Prezzo_biglietto
BEFORE INSERT ON Biglietto
FOR EACH ROW
BEGIN
   IF :NEW.Prezzo IS NOT NULL THEN
      RETURN;
   END IF;

   :NEW.Prezzo := DeterminaPrezzo(:NEW.codice_biglietto, :NEW.codice_carrello, :NEW.ora_inizio_spettacolo);
END;
/
```

TRIGGER controllo_recensione

Il Trigger "controllo_recensione" verifica se l'utente ha visto il film, se ha pagato il biglietto e se ha già recensito il film. Se l'utente non ha ancora visto il film o non ha ancora pagato il suo ordine o ha già recensito il film nel carrello corrente, allora non potrà pubblicare la recensione. Il Trigger in questione effettua anche un controllo sul "codice_film" inserito: se quest'ultimo combacia con quello del biglietto, allora sarà possibile completare l'inserimento, altrimenti verrà mostrato un errore. E' possibile migliorare la situazione evitando la memorizzazione di "codice_film" nella recensione, recuperando quest'ultimo direttamente dal Biglietto tramite la chiave "codice_biglietto", ma per facilitare alcune query abbiamo preferito strutturare l'entità in questo modo.

```
CREATE OR REPLACE TRIGGER controllo_recensione
BEFORE INSERT OR UPDATE ON recensione
FOR EACH ROW
DECLARE
   cont number;
   codice char(10);
   biglietto_corrente Biglietto%ROWTYPE;
   film_nonvisto exception;
   non_pagato exception;
   recensione_gia_fatta exception;
   no_codice_film exception;
BEGIN
   select count(*) INTO cont
   from biglietto bg WHERE bg.codice_biglietto=:NEW.codice_biglietto AND
:NEW.data_pubblicazione>bg.ora_fine_spettacolo;
   if cont < 1 THEN
      RAISE film_nonvisto;
   end if;

   select count(*) INTO cont
   from biglietto bg JOIN carrello ca ON bg.codice_carrello=ca.codice_carrello
   WHERE bg.codice_biglietto=:NEW.codice_biglietto AND ca.data_pagamento IS NOT NULL;
   if cont < 1 THEN
      RAISE non_pagato;
   END IF;

   --Si assicura che il film venga recensito solo una volta per spettacolo da parte dell'utente
   SELECT * INTO biglietto_corrente FROM Biglietto WHERE codice_biglietto=:NEW.codice_biglietto;
   IF INSERTING THEN
      select count(*) INTO cont
      from recensione re join biglietto bi on re.codice_biglietto=bi.codice_biglietto
      where bi.codice_carrello = biglietto_corrente.codice_carrello
      and bi.ora_inizio_spettacolo = biglietto_corrente.ora_inizio_spettacolo
      and bi.ora_fine_spettacolo = biglietto_corrente.ora_fine_spettacolo
      and bi.nome_sala = biglietto_corrente.nome_sala
      and bi.citta_cinema = biglietto_corrente.citta_cinema
      and bi.cap_cinema = biglietto_corrente.cap_cinema
      and bi.via_cinema = biglietto_corrente.via_cinema;
```

```
      IF cont > 0 THEN
         RAISE recensione_gia_fatta;
      END IF;
   END IF;

   SELECT spe.codice_film INTO codice FROM Spettacolo spe JOIN Biglietto bi ON
spe.citta_cinema=bi.citta_cinema
   AND spe.cap_cinema=bi.cap_cinema AND spe.via_cinema=bi.via_cinema AND
spe.nome_sala=bi.nome_sala
   AND spe.Ora_inizio=bi.Ora_inizio_spettacolo AND spe.Ora_fine=bi.Ora_fine_spettacolo
   WHERE bi.codice_biglietto=:NEW.codice_biglietto;

   IF :NEW.codice_film IS NULL THEN
      :NEW.codice_film := codice;
   ELSIF codice <> :NEW.codice_film THEN
      RAISE no_codice_film;
   END IF;

EXCEPTION
WHEN film_nonvisto THEN
   raise_application_error('-20001','Non puoi recensire un film non visto!');
WHEN non_pagato THEN
   raise_application_error('-20002','Non puoi recensire un film se non hai pagato il biglietto!');
WHEN recensione_gia_fatta THEN
   raise_application_error('-20003','Non puoi recensire un film due volte nello stesso carrello!');
WHEN no_codice_film THEN
   raise_application_error('-20004','Il codice del film della recensione non combacia con quello del biglietto!');
END;
/
```

TRIGGER AssociaPromoCarta

Il Trigger "AssociaPromoCarta" consente l'associazione automatica di una promozione a una carta fedeltà, se e solo se quest'ultima non è scaduta e se il sistema decide di assegnarla tramite la generazione di un numero randomico. Se quest'ultimo è > 50 allora la promo potrà essere assegnata, altrimenti no!

```
CREATE OR REPLACE TRIGGER AssociaPromoCarta
AFTER INSERT OR UPDATE ON Promozione
FOR EACH ROW
DECLARE
   CURSOR fedelta IS
      SELECT *
      FROM Carta_fedelta
      WHERE Data_scadenza>=:NEW.Data_inizio_promozione;
   fedelta_corrente carta_fedelta%ROWTYPE;
BEGIN
   OPEN fedelta;
   LOOP
      FETCH fedelta INTO fedelta_corrente;
```

```
        EXIT WHEN fedelta%NOTFOUND;

        IF round(DBMS_RANDOM.VALUE (0, 100)) > 49 THEN
           INSERT INTO promozione_fedelta values (fedelta_corrente.codice_carta_fedelta,
:NEW.codice_promozione);
           dbms_output.put_line('Promozione ''' || TRIM(:NEW.codice_promozione) || ''' assegnata alla carta
fedelta'' ' || fedelta_corrente.codice_carta_fedelta);
        END IF;
    END LOOP;
    CLOSE fedelta;
END;
/
```

TRIGGER controllo_giorno_spettacolo

Il Trigger "controllo_giorno_spettacolo" verifica la coerenza dei valori in "data_inizio" e "data_fine" durante l'inserimento o l'aggiornamento di una tulpa nell'entità "Spettacolo". Siccome uno spettacolo deve terminare il giorno stesso in cui viene proposto, viene verificato se l'anno, il mese e il giorno di "data_inizio" corrisponde con quello di "data_fine" e inoltre viene controllato che l'orario del primo sia antecedente a quello del secondo.

```
CREATE OR REPLACE TRIGGER controllo_giorno_spettacolo
BEFORE INSERT OR UPDATE ON spettacolo
FOR EACH ROW
DECLARE
temp number ;
controllo exception;

BEGIN
temp := 0 ;
if EXTRACT( YEAR from :new.ora_inizio  ) = EXTRACT( YEAR from :new.ora_fine ) AND
       EXTRACT( MONTH FROM :new.ora_inizio ) = EXTRACT( MONTH from :new.ora_fine ) AND
       EXTRACT( DAY FROM :new.ora_inizio ) = EXTRACT(DAY from :new.ora_fine ) AND
        EXTRACT( HOUR FROM CAST( :new.ora_inizio AS TIMESTAMP )) < EXTRACT( HOUR FROM CAST(
:new.ora_fine AS TIMESTAMP))
then
    temp := 1;
  end if ;

IF temp = 0  THEN
    RAISE controllo ;
   end if;

exception
when controllo then
    raise_application_error ('-20001','ERRORE SONO STATE INSERITE DATE NON VALIDE PER L''INIZIO E LA
FINE DELLO SPETTACOLO');
 END;
/
```

Definizione delle Procedure

PROCEDURE update_data_pagamento e elimina_biglietti_scaduti

La procedura **"elimina_biglietti_scaduti"** consente di eliminare i biglietti associati a spettacoli ormai terminati che non sono stati pagati.

```
CREATE OR REPLACE PROCEDURE elimina_biglietti_scaduti (CAR char)
IS
        temp number;

BEGIN
        select count(*) into temp
        from biglietto
        where codice_carrello = CAR AND ora_inizio_spettacolo < (select sysdate from dual);

if temp <= 0  then
        DBMS_OUTPUT.PUT_LINE('Nel carrello ' || CAR || ' non ci sono biglietti scaduti, se il carrello non è
vuoto puoi procedere all''acquisto');
ELSE
        DELETE
        FROM biglietto
        where codice_carrello = CAR AND ora_inizio_spettacolo < (select sysdate from dual);
end if;
end;
/
```

La procedura **"update_data_pagamento"** aggiorna la data del pagamento del carrello preso in input se i biglietti associati allo spettacolo sono ancora validi e quindi lo spettacolo ancora non è iniziato.

```
CREATE OR REPLACE PROCEDURE update_data_pagamento (CAR char,data_p date, data_sistema date default
NULL)
IS
temp number;
temp1 date;
pagamento number;
conteggio number ;
CURSOR c1 IS
  select *
    from biglietto
    where codice_carrello = CAR and ora_inizio_spettacolo <data_p;
    biglietto_scaduto biglietto%ROWTYPE;

BEGIN
    conteggio := 0 ;
    if data_sistema IS NULL THEN
        SELECT SYSDATE into temp1
        from dual;
    ELSE
        temp1 := data_sistema;
    END IF;
```

```
    SELECT count(*) into pagamento
    from carrello cr join biglietto bg ON bg.codice_carrello = cr.codice_carrello
    where cr.codice_carrello = CAR and cr.data_pagamento is null;

    if data_p < temp1 OR pagamento = 0 then
            IF data_p < temp1 then
           raise_application_error('-20001','La data di pagamento non è valida, non puoi tornare indietro nel
tempo per pagare');
    else
       raise_application_error('-20001','Non c''è alcun biglietto da pagare');
    END IF;

    END IF;

    select COUNT ( DISTINCT bg.codice_carrello) INTO temp
    from carrello cr JOIN biglietto bg ON bg.codice_carrello = cr.codice_carrello
    where cr.codice_carrello = CAR AND data_p < ALL (
    select bg.ora_inizio_spettacolo
    from biglietto bg
    where bg.codice_carrello = CAR ) ;

    if temp = 1 then
            UPDATE carrello
            SET data_pagamento = data_p
            where codice_carrello = CAR;
    else
     DBMS_OUTPUT.PUT_LINE('Attenzione! Nel carrello ci sono dei biglietto associati a spettacoli terminati :');
           OPEN c1;
           LOOP
           FETCH c1 INTO biglietto_scaduto ;

              if c1%FOUND then
                 DBMS_OUTPUT.PUT_LINE(biglietto_scaduto.codice_biglietto);
                 conteggio := conteggio +1;
              end if;

             EXIT WHEN c1%NOTFOUND;
           END LOOP;
      DBMS_OUTPUT.PUT_LINE('I biglietti scaduti verranno in automatico eliminati dal carrello per eseguire poi
la modifica della data di pagamento.');
      SELECT count(*) into pagamento
    from carrello cr join biglietto bg ON bg.codice_carrello = cr.codice_carrello
    where cr.codice_carrello = CAR;
elimina_biglietti_scaduti(CAR);
     if pagamento - conteggio > 0 then
      UPDATE carrello
            SET data_pagamento = data_p
            where codice_carrello = CAR;
            end if;
```

```
    end if;
end;
/
```

DI SEGUITO I CASI D'USO:

ACQUISTO ED ELIMINAZIONE DI BIGLIETTI SCADUTI DAL CARRELLO CON IL CODICE 'CART010'

CODICE_CARRELLO	DATA_INSERIMENTO	DATA_PAGAMENTO	USERNAME	NUMERO_CARTA	CODICE_PROMOZIONE
cart010	26-MAR-22	-	Light97	1002341234567890	promo05

Download CSV

```
Statement processed.
ATTENZIONE NEL CARRELLO CI SONO DEI BIGLIETTI ASSOCIATI A SPETTACOLOLI TERMINATI :
ticket021
ticket022
I BIGLIETTI SCADUTI VERRANNO IN AUTOMATICO ELIMINATI DAL CARRELLO PER ESEGUIRE POI LA MODIFICA DELLA DATA DI PAGAMENTO.
```

CODICE_CARRELLO	DATA_INSERIMENTO	DATA_PAGAMENTO	USERNAME	NUMERO_CARTA	CODICE_PROMOZIONE
cart010	26-MAR-22	12-JUL-22	Light97	1002341234567890	promo05

PROCEDURA cliente_del_mese

La procedura **"cliente_del_mese"** assegna un premio di due biglietti omaggio all'utente che ha effettuato più acquisti nell'ultimo mese.
I biglietti vengono inseriti nel carrello con un prezzo settato a zero.
Questa procedura può essere eseguita dall'amministratore e dal gestore.

```
CREATE OR REPLACE PROCEDURE cliente_del_mese
IS

carrello_premio char(10);
citta_spettacolo varchar(20);
spettacolo_premio spettacolo%ROWTYPE;
posto_corrente posto%ROWTYPE;
utente_vincente char(10);
BEGIN

select cr.codice_carrello into carrello_premio
from (select count(bg.codice_biglietto) AS numero_biglietti, cr.codice_carrello
      from  biglietto bg JOIN carrello cr ON bg.codice_carrello = cr.codice_carrello
      where cr.data_pagamento IS NOT NULL AND cr.data_pagamento <= sysdate and cr.data_pagamento
>= ADD_MONTHS(SYSDATE, -1)
      group by cr.codice_carrello
      order by dbms_random.value ) cr
where cr.numero_biglietti = (select max(numero_biglietti)
                  from  (select count(bg.codice_biglietto) AS numero_biglietti, cr.codice_carrello
                        from  biglietto bg JOIN carrello cr ON bg.codice_carrello = cr.codice_carrello
```

```
                        where cr.data_pagamento IS NOT NULL AND cr.data_pagamento <= sysdate and
cr.data_pagamento >= ADD_MONTHS(SYSDATE, -1)
                         group by cr.codice_carrello) cr )
and rownum = 1 ;

            --DBMS_OUTPUT.PUT_LINE(carrello_premio);

            select al.citta_cinema into citta_spettacolo
            from (select bg.citta_cinema, bg.codice_carrello
                   from carrello cr JOIN biglietto bg ON cr.codice_carrello = bg.codice_carrello
                  order By dbms_random.value ) al
            WHERE rownum = 1 and al.codice_carrello = carrello_premio;

            -- DBMS_OUTPUT.PUT_LINE(citta_spettacolo);

            select * into spettacolo_premio
            from (select *
                 from spettacolo
                 where citta_cinema = citta_spettacolo and ora_inizio > sysdate
                 order by dbms_random.value) al
            where rownum = 1 ;

             DBMS_OUTPUT.PUT_LINE(spettacolo_premio.ora_inizio);
            SELECT * INTO posto_corrente FROM Posto po
          WHERE po.nome_sala=spettacolo_premio.nome_sala AND
po.citta_cinema=spettacolo_premio.citta_cinema AND po.cap_cinema=spettacolo_premio.CAP_cinema AND
po.via_cinema=spettacolo_premio.via_cinema
                  AND NOT EXISTS (
                       SELECT NULL FROM Biglietto bi
                       WHERE bi.ora_inizio_spettacolo=spettacolo_premio.ora_inizio AND
bi.ora_fine_spettacolo=spettacolo_premio.ora_fine AND
                        bi.citta_cinema=spettacolo_premio.citta_cinema AND
bi.via_cinema=spettacolo_premio.via_cinema AND bi.cap_cinema=spettacolo_premio.CAP_cinema AND
bi.nome_sala=spettacolo_premio.nome_sala
                       AND bi.fila_posto=po.fila AND bi.numero_posto=po.numero_posto
                       )
            ORDER BY po.Fila
            FETCH FIRST 1 ROWS ONLY;

             --DBMS_OUTPUT.PUT_LINE(posto_corrente.numero_posto);
             --DBMS_OUTPUT.PUT_LINE(posto_corrente.fila);

             select distinct (username ) into utente_vincente
             from carrello
             where codice_carrello = carrello_premio;

              DBMS_OUTPUT.PUT_LINE ('COMPLIMENTI UTENTE : '||utente_vincente||' SEI IL CLIENTE DEL
MESE.');
              DBMS_OUTPUT.PUT_LINE ('IN REGALO DUE BIGLIETTI PER LO SPETTACOLO DEL :
'||spettacolo_premio.ora_inizio||' a '||spettacolo_premio.citta_cinema);
```

```
insert into biglietto values ('ticket'||(TO_CHAR (TRUNC(DBMS_RANDOM.value(100,900)))), 0, carrello_premio,
spettacolo_premio.ora_inizio, spettacolo_premio.ora_fine , posto_corrente.fila, posto_corrente.numero_posto,
spettacolo_premio.nome_sala, spettacolo_premio.citta_cinema, spettacolo_premio.via_cinema,
spettacolo_premio.CAP_cinema);
 SELECT * INTO posto_corrente FROM Posto po
            WHERE po.nome_sala=spettacolo_premio.nome_sala AND
po.citta_cinema=spettacolo_premio.citta_cinema AND po.cap_cinema=spettacolo_premio.CAP_cinema AND
po.via_cinema=spettacolo_premio.via_cinema
                        AND NOT EXISTS (
                            SELECT NULL FROM Biglietto bi
                            WHERE bi.ora_inizio_spettacolo=spettacolo_premio.ora_inizio AND
bi.ora_fine_spettacolo=spettacolo_premio.ora_fine AND
                             bi.citta_cinema=spettacolo_premio.citta_cinema AND
bi.via_cinema=spettacolo_premio.via_cinema AND bi.cap_cinema=spettacolo_premio.CAP_cinema AND
bi.nome_sala=spettacolo_premio.nome_sala
                            AND bi.fila_posto=po.fila AND bi.numero_posto=po.numero_posto
                            )
                ORDER BY po.Fila
                FETCH FIRST 1 ROWS ONLY;
insert into biglietto values ('ticket'||(TO_CHAR (TRUNC(DBMS_RANDOM.value(100,900)))), 0, carrello_premio,
spettacolo_premio.ora_inizio, spettacolo_premio.ora_fine , posto_corrente.fila, posto_corrente.numero_posto,
spettacolo_premio.nome_sala, spettacolo_premio.citta_cinema, spettacolo_premio.via_cinema,
spettacolo_premio.CAP_cinema);


end;
```

DI SEGUITO I CASI D'USO:

PROCEDURA CLIENTE DEL MESE

```
Statement processed.
29-MAY-23
COMPLIMENTI UTENTE : AntodeRoma SEI IL CLIENTE DEL MESE.
IN REGALO DUE BIGLIETTI PER LO SPETTACOLO DEL : 29-MAY-23 a Casoria
```

CODICE_BIGLIETTO	PREZZO	CODICE_CARRELLO	ORA_INIZIO_SPETTACOLO	ORA_FINE_SPETTACOLO	FILA_POSTO	NUMERO_POSTO	NOME_SALA	CITTA_CINEMA	VIA_CINEMA
ticket031	8	cart011	29-MAY-23	29-MAY-23	B	3	01	Casoria	Via mediterraneo
ticket014	8	cart011	04-SEP-22	04-SEP-22	A	2	01	Salerno	Viale antonio
ticket018	8	cart011	04-SEP-22	04-SEP-22	A	1	01	Salerno	Viale antonio
ticket309	0	cart011	29-MAY-23	29-MAY-23	A	3	01	Casoria	Via mediterraneo
ticket739	0	cart011	29-MAY-23	29-MAY-23	A	2	01	Casoria	Via mediterraneo

PROCEDURA inserisci_posti

La procedura "inserisci_posti" automatizza l'inserimento dei cinema con le corrispettive sale e i posti da parte del gestore grazie ai parametri passati in input, quali:

- "Nome", dove viene specificato il nome del cinema che si vuole aggiungere.
- "Via", dove viene specificata la via del cinema che si vuole aggiungere.

- “CAP”, dove viene specificato il CAP del cinema che si vuole aggiungere.
- “Citta”, dove viene specificata la città del cinema che si vuole aggiungere.
- “totale_sale”, importante per specificare quante sale si vogliono aggiungere;
- “totale_posti”, (facoltativo) utilizzato per specificare il totale dei posti per ogni sala;
- “massimo_fila”, (facoltativo) utilizzato per specificare il massimo di posti per ogni fila;
- “tipo_sale”, (facoltativo) utilizzato per specificare la tipologia di sale che si sta aggiungendo.

```
CREATE OR REPLACE PROCEDURE inserisci_posti (Nome VARCHAR, Citta VARCHAR, CAP CHAR, VIA
VARCHAR, totale_sale NUMBER, totale_posti NUMBER DEFAULT NULL, massimo_fila NUMBER DEFAULT 1,
tipo_sale VARCHAR DEFAULT NULL)
IS
   indice NUMBER;
   indice2 NUMBER;
   indice3 NUMBER;
   posti_da_inserire NUMBER;
   posti_inseriti NUMBER;
   fila_corrente CHAR;
   numero NUMBER;
   tipologia VARCHAR(10);
   posti_c NUMBER;
BEGIN
   IF totale_sale > 99 THEN
      raise_application_error('-20040', 'Impossibile aggiungere un cinema con piu'' di 99 sale!');
   END IF;
   IF totale_posti < 1 THEN
      raise_application_error('-20041', 'Inserire un numero di posti corretto!');
   END IF;
   IF massimo_fila < 1 THEN
      raise_application_error('-20042', 'Inserire un numero di file corretto!');
   END IF;
   INSERT INTO Cinema VALUES (Citta, Via, CAP, Nome);

   FOR indice IN 1..totale_sale
   LOOP
      --Genera un tipo di sala randomico se non è specificato in input
      IF tipo_sale IS NULL THEN
         numero := round(DBMS_RANDOM.VALUE (0, 10));
         IF numero < 6 THEN
            tipologia := 'Standard';
         ELSIF numero > 5 AND numero < 8 THEN
            tipologia := '3D';
         ELSIF numero > 8 AND numero < 11 THEN
            tipologia := 'IMAX';
         END IF;
      ELSE
         tipologia := tipo_sale;
      END IF;
      dbms_output.put_line('La sala '||TO_CHAR(indice, 'fm00')||' e'' di tipologia '||tipologia||'.');

      IF totale_posti IS NULL THEN
```

```
            posti_c := round(DBMS_RANDOM.VALUE (1, 20));
        ELSE
            posti_c := totale_posti;
        END IF;
        dbms_output.put_line('La sala '||TO_CHAR(indice, 'fm00')||' ha massima capienza: '||posti_c||'.');

        INSERT INTO Sala VALUES (TO_CHAR(indice, 'fm00'), Citta, VIA, CAP, tipologia, posti_c);

        posti_inseriti := 0;
        fila_corrente := 'A';
        FOR indice2 IN 1..massimo_fila+1
        LOOP
            posti_da_inserire := posti_c/massimo_fila;

            IF posti_da_inserire+posti_inseriti > posti_c THEN
                posti_da_inserire := posti_c - posti_inseriti;
            END IF;

            FOR indice3 IN 1..posti_da_inserire
            LOOP
                insert into posto values (fila_corrente, indice3, TO_CHAR(indice, 'fm00'), citta, via, cap);
                posti_inseriti := posti_inseriti + 1;
            END LOOP;

            fila_corrente := CHR(ASCII(fila_corrente) + 1);
            EXIT WHEN posti_inseriti = posti_c;
        END LOOP;
    END LOOP;
END inserisci_posti;
/
```

Se volessimo richiamare la procedura, un caso d'uso sarebbe il seguente:

```
CALL inserisci_posti('Cinema dei fiori', 'Pomigliano d''Arco', '80038', 'Via Mauro Leone', 3, NULL, 2, NULL);
select * from posto where via_cinema='Via Mauro Leone' order by nome_sala;
```

PROCEDURA annulla_ordine

La procedura "annulla_ordine" annulla o rimborsa l'ordine effettuato se si è entro i limiti consentiti: il richiedente del rimborso deve effettuare l'operazione entro 24 ore precedenti all'ora dell'inizio dello spettacolo. Per fare ciò sono passati come parametri i valori di:

- "codice_carrello", il quale rappresenta il codice dell'ordine che si vuole annullare;
- "data_eliminazione" (facoltativo), in linea generale deve essere SYSDATE, ovvero l'ora esatta del momento in cui si sta eseguendo l'istruzione.

```
CREATE OR REPLACE PROCEDURE annulla_ordine(codice_carrell CHAR, data_eliminazione DATE DEFAULT
SYSDATE)
IS
    conta_biglietti number;
    conta_biglietti_annullabili number;
    prezzo_r number := 0;
```

```
    conta_carrello number;
    indice number;
    biglietto_corrente char;
    data_pagamento_r date;
BEGIN
    select count(*) INTO conta_biglietti from biglietto bigl
    where bigl.codice_carrello = codice_carrell;
    select count(*) INTO conta_carrello from carrello car
    where car.codice_carrello = codice_carrell;

    --Verifica se tutti i biglietti dell'ordine sono annullabili!
    --[Verifica se sono presenti delle recensioni]
    select count(*) INTO conta_biglietti_annullabili
    from biglietto bg join recensione rec on bg.codice_biglietto=rec.codice_biglietto
    where codice_carrell = bg.codice_carrello;
    if conta_biglietti_annullabili > 0 then
        raise_application_error('-20022','Ordine non annullabile! Ci sono '||conta_biglietti_annullabili||' recensioni
fatte!');
    end if;

    --[Verifica se si è ancora in tempo per annullare l'ordine (il giorno prima dello spettacolo)]
    select count(*), sum(bg.prezzo) INTO conta_biglietti_annullabili, prezzo_r from biglietto bg join carrello car
ON bg.codice_carrello=car.codice_carrello
    where codice_carrell = bg.codice_carrello and data_eliminazione<TO_DATE(bg.ora_inizio_spettacolo-1);
    if conta_biglietti_annullabili < conta_biglietti then
        raise_application_error('-20023','Ordine non annullabile! Ci sono '||conta_biglietti-
conta_biglietti_annullabili||' biglietti non annullabili!');
    end if;

    IF conta_biglietti > 0 THEN
        FOR indice IN 1..conta_biglietti
        LOOP
            --Seleziona un biglietto desiderato alla volta
            select bg.codice_biglietto INTO biglietto_corrente from biglietto bg
            where codice_carrell = bg.codice_carrello
            FETCH FIRST 1 ROWS ONLY;

            delete from biglietto where codice_biglietto=biglietto_corrente;
        END LOOP;

        SELECT ca.data_pagamento INTO data_pagamento_r FROM carrello ca WHERE
ca.codice_carrello=codice_carrell;
        DELETE FROM carrello ca WHERE ca.codice_carrello=codice_carrell;

        IF data_pagamento_r IS NOT NULL AND prezzo_r > 0 THEN
            DBMS_OUTPUT.PUT_LINE('L''importo di '||prezzo_r||' € sara'' rimborsato sulla propria carta.');
        ELSE
            DBMS_OUTPUT.PUT_LINE('Ordine annullato, sono stati eliminati '||conta_biglietti||' biglietti.');
        END IF;
    ELSIF conta_biglietti = 0 AND conta_carrello = 1 THEN
            DBMS_OUTPUT.PUT_LINE('Il carrello non contiene biglietti, annullo l''ordine.');
```

```
        DELETE FROM carrello ca WHERE ca.codice_carrello=codice_carrell;
    END IF;
END;
/
```

PROCEDURA auto_acquista

La procedura "auto_acquista" consente l'acquisto automatico da parte dell'utente di un determinato numero di biglietti per uno spettacolo. Si pensa ad esempio all'acquisto rapido di Amazon: per fare in fretta, è possibile cliccare su "acquista ora" al posto di aggiungere al carrello e procedere al pagamento solo in un secondo momento. Tale procedura consentirebbe all'utente di acquistare i biglietti più in fretta. Il funzionamento è semplice: viene prima di tutto verificato se ci sono i posti disponibili (con la FUNCTION "verifica_posti"), successivamente vengono determinati questi ultimi e una volta inserite le tuple nell'entità Biglietto si procede al pagamento dell'ordine tramite la procedura "update_data_pagamento".
Gli argomenti che devono essere passati alla procedura sono i seguenti:

- "username", dove si specifica l'utente che deve effettuare l'ordine;
- "numero_biglietti", dove si specificano il numero di biglietti che si vogliono acquistare;
- "data_acquisto", ovvero la data del momento in cui si sta effettuando l'acquisto (deve essere SYSDATE);
- "numero_carta", il numero della carta con cui si sta effettuando l'acquisto (deve corrispondere ad una delle carte che sono già state salvate nel sistema);
- "ora_fine" e "ora_inizio", i valori di quando termina e comincia lo spettacolo;
- "citta", "via" e "cap", il luogo (o cinema) in cui si terrà lo spettacolo;
- "sala", la sala in cui si terrà lo spettacolo;
- "fila" (facoltativo), ovvero il nome della fila (esempio: 'A') di dove si vuole prendere posto;
- "numero" (facoltativo), cioè il numero della fila corrente scelta nel quale si vuole prendere posto.

Evitando l'assegnazione di questi ultimi due valori, si prevede alla scelta di un posto casuale del singolo spettatore.

Le function "ultimo_biglietto" e "ultimo_carrello" non fanno altro che determinare rispettivamente l'ultimo biglietto e l'ultimo carrello inseriti, in modo tale da inserire la numerazione corretta ed evitare la duplicazione di dati.

```
--Restituisce come tipo number l'ultimo biglietto acquistato
CREATE OR REPLACE FUNCTION ultimo_biglietto
RETURN NUMBER
IS
    codice CHAR(10);
BEGIN
    SELECT Codice_biglietto INTO codice FROM Biglietto ORDER BY Codice_biglietto DESC FETCH FIRST 1
ROWS ONLY;
    RETURN TO_NUMBER(REPLACE(codice, 'ticket'));
EXCEPTION WHEN OTHERS THEN
    --Se da errore perché non ci sono biglietti (o per altre questioni), allora ritorna 0
    RETURN 0;
END ultimo_biglietto;
```

```
/

--Restituisce come tipo number l'ultimo carrello aggiunto
CREATE OR REPLACE FUNCTION ultimo_carrello
RETURN NUMBER
IS
   codice CHAR(10);
BEGIN
   SELECT Codice_carrello INTO codice FROM Carrello ORDER BY Codice_carrello DESC FETCH FIRST 1 ROWS
ONLY;
   RETURN TO_NUMBER(REPLACE(codice, 'cart'));
EXCEPTION WHEN OTHERS THEN
   --Se da errore perché non ci sono carrelli (o per altre questioni), allora ritorna 0
   RETURN 0;
END ultimo_carrello;
/

--Procedura che consente l'acquisto di biglietti multipli da parte dell'utente
CREATE OR REPLACE PROCEDURE auto_acquista(username CHAR, numero_biglietti NUMBER, data_acquisto
DATE, numero_carta CHAR,
ora_inizio DATE, Ora_fine DATE, Citta VARCHAR, Via VARCHAR, CAP CHAR, Sala CHAR, Fila CHAR DEFAULT
NULL, Numero NUMBER DEFAULT NULL)
IS
   indice NUMBER;
   posto_corrente Posto%ROWTYPE;
   numero_corrente NUMBER;
BEGIN
   IF numero_biglietti < 1 THEN
      raise_application_error('-20021','Inserisci un numero di biglietti valido!');
   END IF;
   numero_corrente := numero_biglietti;

   if verifica_posti(numero_corrente, ora_inizio, ora_fine, sala, citta, via, cap) = FALSE THEN
      raise_application_error('-20020','Posti terminati! Biglietto/i non acquistabile/i.');
   end if;

   insert into carrello values ('cart' || TO_CHAR(ultimo_carrello+1, 'fm000'), data_acquisto, NULL, username,
numero_carta, NULL);

   IF Fila IS NOT NULL AND Numero IS NOT NULL THEN
      IF verifica_posto(ora_inizio, ora_fine, sala, citta, via, cap, Fila, Numero) = TRUE THEN
         Insert into biglietto values ('ticket'||TO_CHAR(ultimo_biglietto+1, 'fm000'), 10,
'cart'||TO_CHAR(ultimo_carrello, 'fm000'), ora_inizio, ora_fine, Fila, Numero, Sala, Citta, Via, CAP);
         numero_corrente := numero_corrente-1;
      END IF;
   END IF;


   FOR indice IN ultimo_biglietto+1..ultimo_biglietto+numero_corrente
   LOOP
      SELECT * INTO posto_corrente FROM Posto po
```

```
        WHERE po.nome_sala=sala AND po.citta_cinema=citta AND po.cap_cinema=cap AND
po.via_cinema=via
        AND NOT EXISTS (
            SELECT NULL FROM Biglietto bi
            WHERE bi.ora_inizio_spettacolo=ora_inizio AND bi.ora_fine_spettacolo=ora_fine AND
            bi.citta_cinema=citta AND bi.via_cinema=via AND bi.cap_cinema=cap AND bi.nome_sala=sala
            AND bi.fila_posto=po.fila AND bi.numero_posto=po.numero_posto
        )
        ORDER BY po.Fila
        FETCH FIRST 1 ROWS ONLY;

        insert into biglietto values ('ticket'||TO_CHAR(indice, 'fm000'), NULL, 'cart'||TO_CHAR(ultimo_carrello,
'fm000'), ora_inizio, ora_fine, posto_corrente.Fila, posto_corrente.Numero_posto, Sala, Citta, Via, CAP);
    END LOOP;

    update_data_pagamento('cart'||TO_CHAR(ultimo_carrello, 'fm000'), data_acquisto, data_acquisto);
END auto_acquista;
/

--Esempio di utilizzo
EXECUTE auto_acquista('Luke3012', 2, SYSDATE, '1000735724964861', TO_DATE('2022-04-30 20:00', 'YYYY-
MM-DD hh24:mi'), TO_DATE('2022-04-30 21:30', 'YYYY-MM-DD hh24:mi'), 'Mondragone', 'Corso Umberto I',
'81034', '01', 'A', 1);
select * from carrello ca join biglietto bi on ca.codice_carrello=bi.codice_carrello order by ca.codice_carrello
desc, bi.codice_biglietto desc;
```

```
Statement processed.
All'utente Luke3012 e' stata attribuita la promo promo02
```

CODICE_CARRELLO	DATA_INSERIMENTO	DATA_PAGAMENTO	USERNAME	NUMERO_CARTA	CODICE_PROMOZIONE	CODICE_BIGLIETTO	PREZZO	CODICE_CARRELLO
cart002	30-APR-22	30-APR-22	Luke3012	1000735724964861	promo02	ticket002	10	cart002
cart002	30-APR-22	30-APR-22	Luke3012	1000735724964861	promo02	ticket001	10	cart002

```
Download CSV
2 rows selected.
```

PROCEDURA FakeUser

La procedura “FakeUser” determina se l’utente specificato fa parte della categoria di **recensori fake**. Un recensore è fake se pubblica spesso recensioni troppo positive rispetto alla media, se acquista molte volte biglietti per lo stesso film e se recensisce solo i film di una compagnia determinata. Sarà cura dell’amministratore decidere se bloccare l’utente dal sistema e se cancellare tutte le sue recensioni. Gli argomento da passare a questa procedura sono:

- “utente”, dove viene specificato l’utente che si desidera controllare;
- “se_elimina_recensioni” (facoltativo), dove si specifica se si vogliono eliminare le recensioni dell’utente in caso esso faccia parte della categoria recensore fake;

- “se_blocca_utente” (facoltativo), dove si specifica se bloccare l’utente dal sistema, impedendogli così di poter eseguire qualsiasi operazione di acquisto, di recensione e di sconti fedeltà nel sistema.

La Function “AggiustaVoto” è una semplice function che controlla se il voto calcolato supera il numero 10: un voto può avere massimo 10 come punteggio, per cui se l’operazione da esito positivo ritorna 10, altrimenti restituisce il numero che è stato passato come parametro. E’ un semplice controllo che risparmia alcune righe di codice.

```
CREATE OR REPLACE FUNCTION AggiustaVoto (Voto NUMBER)
RETURN NUMBER
IS
BEGIN
   IF Voto > 10 THEN
      RETURN 10;
   END IF;

   RETURN Voto;
END AggiustaVoto;
/

CREATE OR REPLACE PROCEDURE FakeUser (utente CHAR, se_elimina_recensioni BOOLEAN DEFAULT FALSE,
se_blocca_utente BOOLEAN DEFAULT FALSE)
IS
   CURSOR Recensioni IS
      SELECT fi.codice_film, fi.titolo, avg(re.votoprotagonista) as protagonista,
      avg(re.votoantagonista) as antagonista, avg(re.vototrama) as trama,
      avg(re.votocolonnasonora) as colonna, avg(re.votoscenografia) as scenografia
      FROM Recensione re JOIN Film fi ON re.codice_film=fi.codice_film
      JOIN Biglietto bi ON re.codice_biglietto=bi.codice_biglietto
      JOIN Carrello ca ON bi.codice_carrello=ca.codice_carrello
      WHERE ca.username=utente
      GROUP BY fi.codice_film, fi.titolo;
   recensioni_corrente Recensioni%ROWTYPE;

   protagonista NUMBER;
   antagonista NUMBER;
   trama NUMBER;
   colonna NUMBER;
   scenografia NUMBER;

   conteggio NUMBER := 0;
   conteggio1 NUMBER;
   conteggio2 NUMBER;
   temp VARCHAR(20);
BEGIN
   OPEN Recensioni;

   LOOP
```

```
        FETCH recensioni INTO recensioni_corrente;
        EXIT WHEN recensioni%NOTFOUND;

        --Salva la media delle recensioni del film legata agli altri utenti (tutti eccetto l'utente corrente)
        SELECT avg(mr.votoprotagonista), avg(mr.votoantagonista), avg(mr.vototrama),
        avg(mr.votocolonnasonora), avg(mr.votoscenografia) into protagonista, antagonista, trama, colonna,
scenografia
        from Recensione mr JOIN Film fi ON mr.codice_film=fi.codice_film
        JOIN Biglietto bi ON mr.codice_biglietto=bi.codice_biglietto JOIN Carrello ca ON
bi.codice_carrello=ca.codice_carrello
        WHERE ca.username<>utente AND mr.codice_film=recensioni_corrente.codice_film;

        --Se le recensioni sono più alte della media di almeno 2 punti, allora fai ulteriori verifiche
        IF recensioni_corrente.protagonista >= AggiustaVoto(protagonista+2) AND
recensioni_corrente.antagonista >= AggiustaVoto(antagonista+2)
        AND recensioni_corrente.trama >= AggiustaVoto(trama+2) AND recensioni_corrente.colonna >=
AggiustaVoto(colonna+2)
        AND recensioni_corrente.scenografia >= AggiustaVoto(scenografia+2) THEN
            conteggio := conteggio + 1;

            --Verifica se l'utente ha recensito il film più volte della media
            SELECT COUNT(*) INTO conteggio1 FROM Recensione re WHERE re.codice_film =
recensioni_corrente.codice_film;
            SELECT COUNT(*) INTO conteggio2 FROM Recensione re JOIN Biglietto bi ON
re.codice_biglietto=bi.codice_biglietto
            JOIN Carrello ca ON bi.codice_carrello=ca.codice_carrello
            WHERE re.codice_film = recensioni_corrente.codice_film AND ca.username=utente;
            conteggio1 := conteggio1 - conteggio2;
            IF conteggio1 < conteggio2 THEN
               conteggio := conteggio + 1;
            END IF;

            --Verifica se l'utente ha dato solo i massimi voti per questo film
            IF recensioni_corrente.protagonista = 10 AND recensioni_corrente.antagonista = 10
            AND recensioni_corrente.trama = 10 AND recensioni_corrente.colonna = 10
            AND recensioni_corrente.scenografia = 10 THEN
               conteggio := conteggio + 2;
            END IF;

            --Verifica se l'utente ha recensito il film ogni volta che è andato a vederlo
            --(le recensioni fatte dall'utente sono state già calcolate prima in conteggio2)
            SELECT COUNT(*) INTO conteggio1 FROM Biglietto bi JOIN Spettacolo spe ON
bi.cap_cinema=spe.cap_cinema AND
            bi.via_cinema=spe.via_cinema AND bi.citta_cinema=spe.citta_cinema AND
bi.nome_sala=spe.nome_sala AND
            bi.ora_inizio_spettacolo=spe.ora_inizio AND bi.ora_fine_spettacolo=spe.ora_fine
            JOIN Carrello car ON bi.codice_carrello=car.codice_carrello
            WHERE spe.codice_film = recensioni_corrente.codice_film AND car.username=utente;
            IF conteggio1 = conteggio2 THEN
               conteggio := conteggio + 1;
            END IF;
```

```
        ELSE
            --Il film non è sospetto, decrementa il contatore di 1
            conteggio := conteggio - 1;
        END IF;
    END LOOP;

    dbms_output.put_line('L''utente '''||TRIM(utente)||''' ha recensito '||recensioni%ROWCOUNT||' film.');

    --Verifica se l'utente ha recensito più volte soltanto i film di un'unica compagnia
    SELECT COUNT(*) INTO conteggio1 FROM
    (SELECT fi.nome_compagnia, count(*) FROM compagnia_di_produzione com JOIN Film fi ON
com.nome_compagnia=fi.nome_compagnia
    JOIN (SELECT fi.codice_film, avg(re.votoprotagonista) as protagonista,
        avg(re.votoantagonista) as antagonista, avg(re.vototrama) as trama, avg(re.votocolonnasonora) as
colonna, avg(re.votoscenografia) as scenografia
        FROM Recensione re JOIN Film fi ON re.codice_film=fi.codice_film
        JOIN Biglietto bi ON re.codice_biglietto=bi.codice_biglietto
        JOIN Carrello ca ON bi.codice_carrello=ca.codice_carrello
        WHERE ca.username=utente
        GROUP BY fi.codice_film)
        mr ON mr.codice_film=fi.codice_film
    GROUP BY fi.nome_compagnia);
    IF conteggio1 = 1 AND recensioni%ROWCOUNT > 1 THEN
        SELECT fi.Nome_compagnia INTO temp FROM Film fi WHERE
fi.codice_film=recensioni_corrente.codice_film;
        dbms_output.put_line('L''utente '''||TRIM(utente)||''' ha recensito solo i film della compagnia '||temp||'.');
        conteggio := conteggio + (3*recensioni%ROWCOUNT);
    END IF;

    --Verifica se il "punteggio" raggiunge o supera 5 punti moltiplicati per il numero di film visti
    dbms_output.put_line('Punteggio totalizzato/Massimo possibile:
'||conteggio||'/'||5*recensioni%ROWCOUNT);
    IF conteggio >= 5*recensioni%ROWCOUNT THEN
        dbms_output.put_line('L''utente '''||TRIM(utente)||''' e'' un possibile fake user.');

        IF se_elimina_recensioni = TRUE THEN
            DELETE FROM Recensione rec WHERE rec.id_recensione IN (
            SELECT rec2.id_recensione FROM Recensione rec2 JOIN Biglietto bi ON
rec2.codice_biglietto=bi.codice_biglietto
            JOIN Carrello car ON car.codice_carrello=bi.codice_carrello
            WHERE car.username=utente);

            dbms_output.put_line('Le recensioni dell''utente '''||TRIM(utente)||''' sono state eliminate.');
        END IF;
        IF se_blocca_utente = TRUE THEN
            UPDATE Utente SET abilitato='F' WHERE Username=Utente;
            dbms_output.put_line('L''utente '''||TRIM(utente)||''' e'' stato bloccato.');
        END IF;
    ELSE
        dbms_output.put_line('L''utente '''||TRIM(utente)||''' e'' un bravo utente.');
    END IF;
```

```
    CLOSE Recensioni;
END FakeUser;
/
```

Per dimostrare il **funzionamento** della procedura, proseguiamo con l'inserimento di alcuni dati appositi. Definiamo prima degli spettacoli che riguardano un'unica compagnia di produzione:

```
insert into spettacolo values (TO_DATE('2022-07-15 20:00', 'yyyy-mm-dd hh24:mi'), TO_DATE('2022-07-15 21:30', 'yyyy-mm-dd hh24:mi'), 'Italiano', 'marvel001', '01', 'Mondragone', 'Corso Umberto I', '81034');
insert into spettacolo values (TO_DATE('2022-07-10 20:00', 'yyyy-mm-dd hh24:mi'), TO_DATE('2022-07-10 21:30', 'yyyy-mm-dd hh24:mi'), 'Italiano', 'marvel001', '01', 'Mondragone', 'Corso Umberto I', '81034');
insert into spettacolo values (TO_DATE('2022-06-10 20:00', 'yyyy-mm-dd hh24:mi'), TO_DATE('2022-06-10 22:30', 'yyyy-mm-dd hh24:mi'), 'Italiano', 'marvel002', '01', 'Mondragone', 'Corso Umberto I', '81034');
insert into spettacolo values (TO_DATE('2022-06-11 20:00', 'yyyy-mm-dd hh24:mi'), TO_DATE('2022-06-11 22:30', 'yyyy-mm-dd hh24:mi'), 'Italiano', 'marvel002', '01', 'Mondragone', 'Corso Umberto I', '81034');
insert into spettacolo values (TO_DATE('2022-05-15 19:00', 'yyyy-mm-dd hh24:mi'), TO_DATE('2022-05-15 21:30', 'yyyy-mm-dd hh24:mi'), 'Italiano', 'marvel003', '01', 'Mondragone', 'Corso Umberto I', '81034');
insert into spettacolo values (TO_DATE('2022-04-11 17:30', 'yyyy-mm-dd hh24:mi'), TO_DATE('2022-04-11 19:30', 'yyyy-mm-dd hh24:mi'), 'Italiano', 'marvel003', '01', 'Mondragone', 'Corso Umberto I', '81034');
insert into spettacolo values (TO_DATE('2022-03-15 17:30', 'yyyy-mm-dd hh24:mi'), TO_DATE('2022-03-15 19:30', 'yyyy-mm-dd hh24:mi'), 'Italiano', 'marvel004', '01', 'Mondragone', 'Corso Umberto I', '81034');
insert into spettacolo values (TO_DATE('2022-04-10 17:30', 'yyyy-mm-dd hh24:mi'), TO_DATE('2022-04-10 19:30', 'yyyy-mm-dd hh24:mi'), 'Italiano', 'marvel004', '01', 'Mondragone', 'Corso Umberto I', '81034');
insert into spettacolo values (TO_DATE('2022-02-10 17:30', 'yyyy-mm-dd hh24:mi'), TO_DATE('2022-02-10 19:30', 'yyyy-mm-dd hh24:mi'), 'Italiano', 'marvel004', '01', 'Mondragone', 'Corso Umberto I', '81034');
```

Dopodiché eseguiamo la procedura "auto_acquista" per automatizzare (e facilitare) il processo di acquisto! Acquistiamo e recensiamo prima dei biglietti all'utente buono, dopo inseriamo molteplici valori all'utente fake.

```
EXECUTE auto_acquista('Luke3012', 2, TO_DATE('2022-07-15 19:00', 'yyyy-mm-dd hh24:mi'), '1000735724964861', TO_DATE('2022-07-15 20:00', 'yyyy-mm-dd hh24:mi'), TO_DATE('2022-07-15 21:30', 'yyyy-mm-dd hh24:mi'), 'Mondragone', 'Corso Umberto I', '81034', '01');
--Recensice il film appena acquistato
insert into recensione values ('rec50', 6, 5, 4, 7, 6, TO_DATE('16/07/2022', 'dd/mm/yyyy'), 'ticket'||TO_CHAR(ultimo_biglietto, 'fm000'), 'marvel001');
EXECUTE auto_acquista('Luke3012', 3, TO_DATE('2022-03-14 17:30', 'yyyy-mm-dd hh24:mi'), '1000735724964861', TO_DATE('2022-03-15 17:30', 'yyyy-mm-dd hh24:mi'), TO_DATE('2022-03-15 19:30', 'yyyy-mm-dd hh24:mi'), 'Mondragone', 'Corso Umberto I', '81034', '01');
--Recensice il film appena acquistato
insert into recensione values ('rec55', 1, 1, 1, 1, 1, TO_DATE('16/03/2022', 'dd/mm/yyyy'), 'ticket'||TO_CHAR(ultimo_biglietto, 'fm000'), 'marvel004');

--Ecco la persona fake che recensisce tutti i film marvel come perfetti (rispetto alla media) ogni volta che va al cinema
EXECUTE auto_acquista('Simo', 1, TO_DATE('2022-07-15 19:00', 'yyyy-mm-dd hh24:mi'), '1000735724964812', TO_DATE('2022-07-15 20:00', 'yyyy-mm-dd hh24:mi'), TO_DATE('2022-07-15 21:30', 'yyyy-mm-dd hh24:mi'), 'Mondragone', 'Corso Umberto I', '81034', '01');
insert into recensione values ('rec51', 10, 10, 10, 10, 10, TO_DATE('16/07/2022', 'dd/mm/yyyy'), 'ticket'||TO_CHAR(ultimo_biglietto, 'fm000'), 'marvel001');
```

```
EXECUTE auto_acquista('Simo', 1, TO_DATE('2022-06-09 20:00', 'yyyy-mm-dd hh24:mi'), '1000735724964812',
TO_DATE('2022-06-10 20:00', 'yyyy-mm-dd hh24:mi'), TO_DATE('2022-06-10 22:30', 'yyyy-mm-dd hh24:mi'),
'Mondragone', 'Corso Umberto I', '81034', '01');
insert into recensione values ('rec52', 10, 10, 10, 10, 10, TO_DATE('11/06/2022', 'dd/mm/yyyy'),
'ticket'||TO_CHAR(ultimo_biglietto, 'fm000'), 'marvel002');
EXECUTE auto_acquista('Simo', 1, TO_DATE('2022-05-15 17:00', 'yyyy-mm-dd hh24:mi'), '1000735724964812',
TO_DATE('2022-05-15 19:00', 'yyyy-mm-dd hh24:mi'), TO_DATE('2022-05-15 21:30', 'yyyy-mm-dd hh24:mi'),
'Mondragone', 'Corso Umberto I', '81034', '01');
insert into recensione values ('rec53', 10, 10, 10, 10, 10, TO_DATE('16/05/2022', 'dd/mm/yyyy'),
'ticket'||TO_CHAR(ultimo_biglietto, 'fm000'), 'marvel003');
EXECUTE auto_acquista('Simo', 1, TO_DATE('2022-03-14 17:30', 'yyyy-mm-dd hh24:mi'), '1000735724964812',
TO_DATE('2022-03-15 17:30', 'yyyy-mm-dd hh24:mi'), TO_DATE('2022-03-15 19:30', 'yyyy-mm-dd hh24:mi'),
'Mondragone', 'Corso Umberto I', '81034', '01');
insert into recensione values ('rec54', 10, 10, 10, 10, 10, TO_DATE('16/03/2022', 'dd/mm/yyyy'),
'ticket'||TO_CHAR(ultimo_biglietto, 'fm000'), 'marvel004');
EXECUTE auto_acquista('Simo', 1, TO_DATE('2022-04-10 15:30', 'yyyy-mm-dd hh24:mi'), '1000735724964812',
TO_DATE('2022-04-10 17:30', 'yyyy-mm-dd hh24:mi'), TO_DATE('2022-04-10 19:30', 'yyyy-mm-dd hh24:mi'),
'Mondragone', 'Corso Umberto I', '81034', '01');
insert into recensione values ('rec56', 10, 10, 10, 10, 10, TO_DATE('11/04/2022', 'dd/mm/yyyy'),
'ticket'||TO_CHAR(ultimo_biglietto, 'fm000'), 'marvel004');
EXECUTE auto_acquista('Simo', 1, TO_DATE('2022-04-11 15:30', 'yyyy-mm-dd hh24:mi'), '1000735724964812',
TO_DATE('2022-04-11 17:30', 'yyyy-mm-dd hh24:mi'), TO_DATE('2022-04-11 19:30', 'yyyy-mm-dd hh24:mi'),
'Mondragone', 'Corso Umberto I', '81034', '01');
insert into recensione values ('rec57', 10, 10, 10, 10, 10, TO_DATE('12/04/2022', 'dd/mm/yyyy'),
'ticket'||TO_CHAR(ultimo_biglietto, 'fm000'), 'marvel003');
EXECUTE auto_acquista('Simo', 1, TO_DATE('2022-07-10 15:00', 'yyyy-mm-dd hh24:mi'), '1000735724964812',
TO_DATE('2022-07-10 20:00', 'yyyy-mm-dd hh24:mi'), TO_DATE('2022-07-10 21:30', 'yyyy-mm-dd hh24:mi'),
'Mondragone', 'Corso Umberto I', '81034', '01');
insert into recensione values ('rec58', 10, 10, 10, 10, 10, TO_DATE('11/07/2022', 'dd/mm/yyyy'),
'ticket'||TO_CHAR(ultimo_biglietto, 'fm000'), 'marvel001');
EXECUTE auto_acquista('Simo', 1, TO_DATE('2022-06-11 15:00', 'yyyy-mm-dd hh24:mi'), '1000735724964812',
TO_DATE('2022-06-11 20:00', 'yyyy-mm-dd hh24:mi'), TO_DATE('2022-06-11 22:30', 'yyyy-mm-dd hh24:mi'),
'Mondragone', 'Corso Umberto I', '81034', '01');
insert into recensione values ('rec59', 10, 10, 10, 10, 10, TO_DATE('12/06/2022', 'dd/mm/yyyy'),
'ticket'||TO_CHAR(ultimo_biglietto, 'fm000'), 'marvel002');
EXECUTE auto_acquista('Simo', 1, TO_DATE('2022-02-10 15:00', 'yyyy-mm-dd hh24:mi'), '1000735724964812',
TO_DATE('2022-02-10 17:30', 'yyyy-mm-dd hh24:mi'), TO_DATE('2022-02-10 19:30', 'yyyy-mm-dd hh24:mi'),
'Mondragone', 'Corso Umberto I', '81034', '01');
insert into recensione values ('rec60', 10, 10, 10, 10, 10, TO_DATE('11/02/2022', 'dd/mm/yyyy'),
'ticket'||TO_CHAR(ultimo_biglietto, 'fm000'), 'marvel004');
```

Dopodiché eseguiamo la procedura “FakeUser” per determinare quale dei due utenti è effettivamente fake.

```
execute FakeUser('Luke3012');
execute FakeUser('Simo');
```

```
Statement processed.
L'utente 'Luke3012' ha recensito 2 film.
L'utente 'Luke3012' ha recensito solo i film della compagnia Marvel Studios.
Punteggio totalizzato/Massimo possibile: 4/10
L'utente 'Luke3012' e' un bravo utente.

Statement processed.
L'utente 'Simo' ha recensito 4 film.
L'utente 'Simo' ha recensito solo i film della compagnia Marvel Studios.
Punteggio totalizzato/Massimo possibile: 20/20
L'utente 'Simo' e' un possibile fake user.
```

PROCEDURA Assegna_carta_fedelta

La procedura "Assegna_carta_fedelta" consente al gestore e allo scheduler di assegnare carte fedeltà agli utenti che rispettano determinate condizioni:

- l'utente deve essere maggiorenne
- l'utente deve essere sprovvisto di carta fedeltà, oppure possederne una scaduta
- l'utente deve aver acquistato almeno 20 biglietti nel corso dell'ultimo mese
- l'utente deve aver visto almeno 5 Film diversi l'uno dall'altro nel corso delle ultime tre settimane
- l'utente deve aver pubblicato almeno la metà delle recensioni possibili

Per creare un JOB che deve essere eseguito ogni giorno all'1 di notte, assegniamo questo codice. La parte riguardante lo scheduler verrà poi approfondita nell'apposita sezione.

```
BEGIN
  DBMS_SCHEDULER.create_job (
    job_name        => 'assegna_carte_fedelta_run',
    job_type        => 'STORED_PROCEDURE',
    job_action      => 'daily_run',
    start_date      => SYSTIMESTAMP,
    repeat_interval => 'freq=hourly; byhour=01',
    end_date        => NULL,
    enabled         => TRUE);
END;
/
dbms_scheduler.create_schedule('daily_run', repeat_interval =>
  'FREQ=DAILY;BYHOUR=01');

--Successivamente il JOB esegue la procedura 'assegna_carte_fedelta'
DBMS_SCHEDULER.CREATE_PROGRAM (
   program_name => 'assegna_carte_fedelta_job',
   program_type   => 'STORED_PROCEDURE',
   program_action    => 'assegna_carte_fedelta',
   enabled   => TRUE);

CREATE OR REPLACE PROCEDURE Assegna_carta_fedelta
IS
```

```
    CURSOR Utenti IS
        SELECT * FROM Utente
        WHERE TO_NUMBER((SYSDATE - Data_di_nascita)/365.25)>17
        AND abilitato='V';

    utente_corrente Utenti%ROWTYPE;
    codice_fedelta_c CHAR(10);
    codice_fedelta_n NUMBER;
    conteggio NUMBER;
    conteggio_massimo NUMBER;
    utenti_assegnatari NUMBER := 0;
BEGIN
    OPEN Utenti;

    LOOP
        FETCH utenti INTO utente_corrente;
        EXIT WHEN utenti%NOTFOUND;

        --Se l'utente ha già una carta fedeltà abilitata, salta avanti
        SELECT COUNT(*) INTO conteggio FROM Carta_fedelta cf WHERE
cf.Username=utente_corrente.Username AND cf.Data_scadenza>SYSDATE;
        IF conteggio = 1 THEN
            CONTINUE;
        END IF;

        --Verifica se l'utente ha acquistato almeno 20 biglietti nell'ultimo mese
        SELECT COUNT(*) INTO conteggio FROM Biglietto bi JOIN Carrello car ON
bi.codice_carrello=car.codice_carrello
        WHERE car.username=utente_corrente.username AND car.data_pagamento >= add_months(SYSDATE, -
1);
        IF conteggio < 20 THEN
            CONTINUE;
        END IF;

        --Verifica se l'utente ha visto almeno 5 film nel corso delle ultime tre settimane
        SELECT COUNT(DISTINCT spe.codice_film) INTO conteggio FROM Spettacolo spe JOIN Biglietto bi ON
bi.ora_inizio_spettacolo=spe.ora_inizio AND bi.ora_fine_spettacolo=spe.ora_fine
        AND bi.nome_sala=spe.nome_sala AND bi.citta_cinema=spe.citta_cinema AND
bi.cap_cinema=spe.cap_cinema AND bi.via_cinema=spe.via_cinema
        JOIN Carrello car ON car.codice_carrello=bi.codice_carrello
        WHERE car.username=utente_corrente.username AND bi.ora_inizio_spettacolo>TO_DATE(SYSDATE -
21);
        IF conteggio < 5 THEN
            CONTINUE;
        END IF;

        --Verifica se l'utente ha pubblicato almeno la metà delle recensioni pubblicabili (una recensione per film
diverso in un carrello)
        SELECT COUNT(*) INTO conteggio_massimo FROM (SELECT DISTINCT bi.ora_inizio_spettacolo,
bi.ora_fine_spettacolo, bi.nome_sala,
```

```
        bi.citta_cinema, bi.cap_cinema, bi.via_cinema, bi.codice_carrello FROM Biglietto bi JOIN Carrello car ON
bi.codice_carrello=car.codice_carrello
        WHERE car.Username=utente_corrente.Username);
        SELECT COUNT(*) INTO conteggio FROM Recensione rec JOIN Biglietto bi ON
rec.codice_biglietto=bi.codice_biglietto
        JOIN Carrello car ON car.codice_carrello=bi.codice_carrello WHERE
car.username=utente_corrente.username;
        IF conteggio < conteggio_massimo/2 THEN
            CONTINUE;
        END IF;

        SELECT COUNT(*) INTO conteggio FROM Carta_fedelta cf WHERE
cf.Username=utente_corrente.username;
        codice_fedelta_n := 0;
        IF conteggio > 0 THEN
            SELECT TO_NUMBER(cf.codice_carta_fedelta) INTO codice_fedelta_n FROM Carta_fedelta cf WHERE
cf.Username=utente_corrente.Username;
        END IF;
        IF codice_fedelta_n = 0 THEN
            SELECT COUNT(*) INTO conteggio FROM Carta_fedelta;
            IF conteggio > 0 THEN
                SELECT codice_carta_fedelta INTO codice_fedelta_c FROM Carta_fedelta ORDER BY
Codice_carta_fedelta DESC FETCH FIRST 1 ROWS ONLY;
                codice_fedelta_n := TO_NUMBER(codice_fedelta_c) + 1;
            ELSE
                codice_fedelta_n := 0;
            END IF;
            dbms_output.put_line('All''utente '''||TRIM(utente_corrente.username)||''' e'' stata creata una nuova
carta fedelta'' con codice '''||TO_CHAR(codice_fedelta_n, 'fm0000000000')||'''.');
        ELSE
            dbms_output.put_line('All''utente '''||TRIM(utente_corrente.username)||''' e'' stata rinnovata la sua carta
fedelta'' con codice '''||TO_CHAR(codice_fedelta_n, 'fm0000000000')||'''.');
        END IF;

        INSERT INTO carta_fedelta VALUES (TO_CHAR(codice_fedelta_n, 'fm0000000000'),
add_months(SYSDATE, 12), SYSDATE, utente_corrente.username);

        utenti_assegnatari := utenti_assegnatari + 1;
    END LOOP;

    dbms_output.put_line('Da un totale di '||utenti%ROWCOUNT||' possibili utenti, '||utenti_assegnatari||' sono
assegnatari di una carta fedeltà.');

    CLOSE Utenti;
END Assegna_carta_fedelta;
/
```

Viste

La vista è l'alias di una query, ovvero serve per dare un nome ad una query e trattare quest'ultima come se fosse una tabella, la vista quindi è una singola tabella che deriva da altre tabelle. Possono essere inoltre tabelle base o anche altre viste definite precedentemente. Il vantaggio delle viste è che sono dinamiche, ovvero vengono eseguite ogni volta che vengono richiamate e quindi di conseguenza non vengono memorizzate mai fisicamente all'interno della base di dati, le righe della vista possono essere modificate o recuperate come una tabella. Gli svantaggi delle viste invece sono l'aggiornamento dei dati e inoltre la vista ha un costo computazionale abbastanza alto.

In questa specifica basi di dati abbiamo creato cinque viste che ci possono aiutare alla visualizzazione di alcune query molto richieste. Per una questione di comodità noi utilizziamo il comando CREATE OR REPLACE VIEW che serve per rinominare una vista, in questo modo abbiamo potuto testare le varie viste all'interno di ORACLE LIVE SQL.

VISTA SalaIMAX3D

La prima vista visualizza tutte le sale 'IMAX' e '3D' dei cinema. Effettua una join tra la tabella cinema e la tabella sala e ci seleziona le sale in cui il Tipo_sala = 'IMAX' o in cui il Tipo_sala = '3D'

```
CREATE OR REPLACE VIEW SalaIMAX3D AS
select * from cinema ci join sala sa on (ci.Citta = sa.Citta_cinema and ci.Via = sa.Via_cinema and ci.CAP =
sa.CAP_cinema) WHERE Tipo_sala = 'IMAX' or Tipo_sala = '3D';
```

VISTA MediaRecensione

La seconda vista visualizza la media dei voti suddivisi in cinque categorie delle recensioni. Effettua una join tra la tabella recensione e la tabella film, infine vengono ordinati tramite titolo.

```
CREATE OR REPLACE VIEW MediaRecensione AS
select fi.codice_film, fi.titolo, avg(re.votoprotagonista) as protagonista, avg(re.votoantagonista) as
antagonista,
avg(re.vototrama) as trama, avg(re.votocolonnasonora) as colonna, avg(re.votoscenografia) as scenografia
from recensione re join film fi on re.codice_film=fi.codice_film
group by fi.codice_film, fi.titolo
order by fi.titolo;
```

VISTA RegistaAttore

La terza vista visualizza tutti i film in cui il professionista svolge sia la professione di Regista che la professione di Attore all'interno dello stesso film. In questo caso vengono effettuate delle join che servono a collegare tre tabelle: Professionista, Diretto_da e Recitato_da. Infine vengono ordinati tramite nome_ruolo.

```
CREATE OR REPLACE VIEW RegistaAttore AS
select pro.nome AS Nome_professionista, re.nome_ruolo AS Personaggio, pro.codice_professionista  from
professionista pro
join diretto_da di on pro.codice_professionista = di.codice_professionista
join recitato_da re on pro.codice_professionista = re.codice_professionista
order by re.nome_ruolo;
```

VISTA VistaFilm

La quarta vista visualizza tutti I film disponibili all'interno della nostra base di dati con i loro multipli generi, abbiamo utilizzato il comando LISTAGG per concatenare i vari generi e sono separati con il carattere seguente: ",".

```
CREATE OR REPLACE VIEW VistaFilm AS
SELECT Codice_film, Titolo, LISTAGG(ge.genere, ',') WITHIN GROUP (ORDER BY genere) AS Genere
FROM Film fi JOIN genere_film gf ON fi.codice_film=gf.id_film
JOIN genere ge ON ge.id_genere=gf.id_genere
GROUP BY Codice_film, Titolo;
```

VISTA Orari_spettacoli

La quinta vista visualizza tutti gli spettacoli con i vari orari di inizio e fine spettacolo richiamando la vista precedente "VistaFilm" e facendo la join con la tabella spettacolo, infine ordiniamo la tabella con Titolo.

```
CREATE OR REPLACE VIEW Orari_spettacoli AS
SELECT fi.titolo, fi.genere,to_char(spe.ora_inizio, 'hh24:mi') as Ora_inizio, to_char(spe.ora_fine, 'hh24:mi') as
Ora_fine
FROM VistaFilm fi JOIN Spettacolo spe ON fi.codice_film=spe.codice_film
ORDER BY fi.titolo;
```

Scheduler

Lo scheduler viene utilizzato per programmare azioni svolte in automatico dal sistema in un determinato istante di tempo, facilitando nel nostro caso il lavoro del gestore.
Lo scheduler è associato ad un evento legato alla data di sistema permettendo l'esecuzione di una procedura ad una condizione stabilita.
Nel nostro caso utilizziamo un **JOB** che esegue la procedura "assegna_carta_fedelta" ogni giorno alle "01:00".

Di seguito il codice implementato:

```
--Crea un JOB che viene eseguito ogni giorno all'1 di notte.
BEGIN
  DBMS_SCHEDULER.create_job (
    job_name        => 'assegna_carte_fedelta_run',
    job_type        => 'STORED_PROCEDURE',
    job_action      => 'daily_run',
    start_date      => SYSTIMESTAMP,
    repeat_interval => 'freq=hourly; byhour=01',
    end_date        => NULL,
    enabled         => TRUE);
END;
```

```
/
dbms_scheduler.create_schedule('daily_run', repeat_interval =>
  'FREQ=DAILY;BYHOUR=01');

--Successivamente il JOB esegue la procedura 'assegna_carte_fedelta'
DBMS_SCHEDULER.CREATE_PROGRAM (
   program_name => 'assegna_carte_fedelta_job',
   program_type   => 'STORED_PROCEDURE',
   program_action    => 'assegna_carte_fedelta',
   enabled   => TRUE);
```

Data Manipulation Language

Il popolamento della maggior parte delle tabelle avviene attraverso le operazioni dirette di INSERT; solo “promozione_fedelta” è popolata automaticamente grazie al Trigger “AssociaPromoCarta”, per cui ad ogni inserimento di una Promozione il sistema automaticamente verifica se assegnare la promozione alla carta fedeltà.
Di seguito mostriamo solo alcuni inserimenti, per evitare lo spreco di carta digitale allo scopo di rendere più scorrevole la visualizzazione del documento. E’ possibile visualizzare tutto il codice allegato nei file “.sql” all’interno della cartella del progetto.

```
--Utenti (username, nome, cognome, sesso, data_di_nascita, luogo_di_nascita)
INSERT INTO utente VALUES ('Luke3012', 'Luca', 'Tartaglia', 'M', TO_DATE('30/12/2001', 'dd/mm/yyyy'),
'Pollena Trocchia', 'V');
INSERT INTO utente VALUES ('Simo', 'Simone', 'Palladino', 'M', TO_DATE('06/07/2000', 'dd/mm/yyyy'),
'Acerra', 'V');
INSERT INTO utente VALUES ('MarioTart', 'Mario', 'Tartulla', 'M', TO_DATE('15/10/2001', 'dd/mm/yyyy'),
'Venezia', 'V');
INSERT INTO utente VALUES ('LucaAa', 'Luca', 'Di Palma', 'M', TO_DATE('30/12/1999', 'dd/mm/yyyy'), 'Milano',
'V');
INSERT INTO utente VALUES ('GnackGnack', 'Mattia', 'Di Palma', 'M', TO_DATE('30/12/1997', 'dd/mm/yyyy'),
'Napoli', 'V');
INSERT INTO utente VALUES ('MarkPigna', 'Marco', 'Pignatelli', 'M', TO_DATE('17/09/2003', 'dd/mm/yyyy'),
'Marano di Napoli', 'V');
INSERT INTO utente VALUES ('Arena', 'Simone', 'Arenare', 'M', TO_DATE('30/12/2004', 'dd/mm/yyyy'),
'Genova', 'V');


--Cinema (citta, via, cap, nome_cinema)
Insert into cinema values ('Mondragone', 'Corso Umberto I', '81034', 'The Space Mondragone');
Insert into cinema values ('Fuorigrotta', 'Viale giochi del mediterraneo', '80125', 'The Space Fuorigrotta');
Insert into cinema values ('Marano di Napoli', 'Via XV Maggio', '80016', 'The Space Marano');
Insert into cinema values ('Pomigliano d''arco', 'Via Carlo', '80038', 'The Space Pomigliano');

--Sale (nome_sala, citta_cinema, via_cinema, cap_cinema, tipo_sala, massima_capienza)
insert into sala values ('01', 'Mondragone', 'Corso Umberto I', '81034', 'IMAX', 15);
insert into sala values ('02', 'Mondragone', 'Corso Umberto I', '81034', 'Standard', 10);
insert into sala values ('03', 'Mondragone', 'Corso Umberto I', '81034', 'Standard', 10);
insert into sala values ('01', 'Fuorigrotta', 'Viale giochi del mediterraneo', '80125', 'IMAX', 10);
insert into sala values ('02', 'Fuorigrotta', 'Viale giochi del mediterraneo', '80125', '3D', 8);
insert into sala values ('03', 'Fuorigrotta', 'Viale giochi del mediterraneo', '80125', 'Standard', 6);
insert into sala values ('04', 'Fuorigrotta', 'Viale giochi del mediterraneo', '80125', 'Standard', 6);
insert into sala values ('01', 'Marano di Napoli', 'Via XV Maggio', '80016', 'IMAX', 10);
insert into sala values ('02', 'Marano di Napoli', 'Via XV Maggio', '80016', 'Standard', 8);
insert into sala values ('03', 'Marano di Napoli', 'Via XV Maggio', '80016', 'Standard', 6);
insert into sala values ('01', 'Pomigliano d''arco', 'Via Carlo', '80038', 'IMAX', 10);
insert into sala values ('02', 'Pomigliano d''arco', 'Via Carlo', '80038', 'Standard', 8);
insert into sala values ('03', 'Pomigliano d''arco', 'Via Carlo', '80038', '3D', 6);


--Posti (fila, numero_posto, nome_sala, citta_cinema, via_cinema, cap_cinema)
insert into posto values ('A', 1, '01', 'Mondragone', 'Corso Umberto I', '81034');
```

```
insert into posto values ('A', 2, '01', 'Mondragone', 'Corso Umberto I', '81034');
insert into posto values ('B', 1, '01', 'Mondragone', 'Corso Umberto I', '81034');
insert into posto values ('C', 3, '01', 'Mondragone', 'Corso Umberto I', '81034');
insert into posto values ('A', 1, '01', 'Pomigliano d''arco', 'Via Carlo', '80038');
--... e così via


--Carte fedeltà (codice_carta_fedelta, data_scadenza, data_rinnovo, username)
insert into carta_fedelta values ('1111111111', TO_DATE('10/06/2023','dd/mm/yyyy'), TO_DATE('10/06/2022',
'dd/mm/yyyy'), 'Luke3012');
insert into carta_fedelta values ('1111111112', TO_DATE('06/07/2025','dd/mm/yyyy'), TO_DATE('06/07/2022',
'dd/mm/yyyy'), 'Simo');


--Promozioni (codice_promozione, nome_promozione, data_inizio_promozione, data_fine_promozione)
insert into promozione values ('promo01', 'Clienti best', TO_DATE('01/06/2022', 'dd/mm/yyyy'),
TO_DATE('30/06/2022', 'dd/mm/yyyy'));
insert into promozione values ('promo02', 'Clienti lusso', TO_DATE('01/06/2021', 'dd/mm/yyyy'),
TO_DATE('31/07/2022', 'dd/mm/yyyy'));
insert into promozione values ('promo03', 'Clienti extra', TO_DATE('09/06/2022', 'dd/mm/yyyy'),
TO_DATE('09/07/2022', 'dd/mm/yyyy'));
insert into promozione values ('promo04', 'Popcorn Gratis', TO_DATE('10/06/2022', 'dd/mm/yyyy'),
TO_DATE('10/07/2022', 'dd/mm/yyyy'));


--Carte di credito (numero_carta(16), data_scadenza, cvv(3))
insert into carta_di_credito values ('1000735724964861', TO_DATE('10/10/2024', 'dd/mm/yyyy'), 101);
insert into carta_di_credito values ('1000735724964812', TO_DATE('10/10/2028', 'dd/mm/yyyy'), 134);
insert into carta_di_credito values ('1000735724964836', TO_DATE('10/10/2023', 'dd/mm/yyyy'), 134);

--Carrelli (codice_carrello, data_inserimento, data_pagamento, username, numero_carta, codice_promozione)
--senza promozione (assegnata poi in automatico dal Trigger)
insert into carrello values ('cart001', TO_DATE('01/05/2022 16:30', 'dd/mm/yyyy
hh24:mi'),TO_DATE('01/05/2022 16:31', 'dd/mm/yyyy hh24:mi'), 'Luke3012', '1000735724964861', NULL);
insert into carrello values ('cart019', TO_DATE('01/05/2022 17:00', 'dd/mm/yyyy
hh24:mi'),TO_DATE('01/05/2022 17:01', 'dd/mm/yyyy hh24:mi'), 'TullyS09', '3021735714234852', NULL);

--con promozione (inserimento manuale)
insert into carrello values ('cart002', TO_DATE('05/03/2022 20:14', 'dd/mm/yyyy
hh24:mi'),TO_DATE('05/06/2022 20:15', 'dd/mm/yyyy hh24:mi'), 'Luke3012', '1000735724964861', 'promo01');
insert into carrello values ('cart014', TO_DATE('21/06/2020 17:00', 'dd/mm/yyyy
hh24:mi'),TO_DATE('21/06/2029 17:01', 'dd/mm/yyyy hh24:mi'), 'Simo', '1000735724964812', 'promo02');

--Compagnie di produzione (nome_compagnia, citta)
insert into compagnia_di_produzione values ('Marvel Studios', 'Burbank');
insert into compagnia_di_produzione values ('20th Century Fox', 'Century City');
insert into compagnia_di_produzione values ('Pixar', 'Emeryville');
insert into compagnia_di_produzione values ('LucasFilm', 'San Francisco');

--Genere di film (id_genere, genere)
insert into genere values ('001', 'Azione');
```

```
insert into genere values ('002', 'Film di supereroi');
insert into genere values ('003', 'Fantascienza');
insert into genere values ('004', 'Avventura');
insert into genere values ('005', 'Cinema fantastico');
insert into genere values ('006', 'Animazione');
insert into genere values ('007', 'Biografico');
insert into genere values ('008', 'Cappa e Spada');
insert into genere values ('009', 'Catastrofico');
insert into genere values ('010', 'Comico');
insert into genere values ('011', 'Commedia');
insert into genere values ('012', 'Drammatico');
insert into genere values ('013', 'Fantasy');
insert into genere values ('014', 'Erotico');
insert into genere values ('015', 'Giallo');
insert into genere values ('016', 'Horror');
insert into genere values ('017', 'Musicale');
insert into genere values ('018', 'Western');
insert into genere values ('019', 'Spionaggio');
insert into genere values ('020', 'Politico-sociale');

--Film (codice_film, titolo, durata, anno_di_produzione, nome_compagnia, classificazione)
insert into film values ('marvel001', 'Doctor Strange nel Multiverso della Follia', 126, 2022, 'Marvel Studios',
14);
insert into film values ('marvel002', 'Thor Love and Thunder', 119, 2022, 'Marvel Studios', 13);
insert into film values ('marvel003', 'Ant-Man and the Wasp: Quantumania', 136, 2023, 'Marvel Studios', 13);
insert into film values ('marvel004', 'Guardiani della Galassia Vol.3', 144, 2023, 'Marvel Studios', 13);
insert into film values ('disney001', 'Il Re Leone', 118, 2019, 'Disney', 12);
insert into film values ('warner001', 'The Flash', 130, 2023, 'DC Studios', 13);
insert into film values ('warner002', 'Animali Fantastici I segreti di Silente', 142, 2022, 'Warner Bros', 13);

--Qui viene collegato il singolo film ai vari generi (id_genere, id_film)
insert into genere_film values ('001', 'marvel001');
insert into genere_film values ('002', 'marvel001');
insert into genere_film values ('003', 'marvel001');
insert into genere_film values ('001', 'marvel002');
insert into genere_film values ('002', 'marvel002');
insert into genere_film values ('003', 'marvel002');
insert into genere_film values ('001', 'marvel003');
insert into genere_film values ('002', 'marvel003');
insert into genere_film values ('003', 'marvel003');
insert into genere_film values ('001', 'marvel004');
insert into genere_film values ('002', 'marvel004');
insert into genere_film values ('003', 'marvel004');
insert into genere_film values ('004', 'disney001');
insert into genere_film values ('001', 'warner001');
insert into genere_film values ('002', 'warner001');
insert into genere_film values ('003', 'warner001');
insert into genere_film values ('001', 'warner002');
insert into genere_film values ('013', 'warner002');

--Spettacoli! (ora_inizio, ora_fine, lingua, codice_film, nome_sala, citta_cinema, via_cinema, cap_cinema)
```

```
insert into spettacolo values (TO_DATE('2022-04-30 20:00', 'YYYY-MM-DD hh24:mi'), TO_DATE('2022-04-30 21:30', 'YYYY-MM-DD hh24:mi'), 'Italiano', 'marvel001', '01', 'Mondragone', 'Corso Umberto I', '81034');
insert into spettacolo values (TO_DATE('2022-05-07 19:00', 'YYYY-MM-DD hh24:mi'), TO_DATE('2022-05-07 20:30', 'yyyy-mm-dd hh24:mi'), 'Italiano', 'marvel001', '01', 'Mondragone', 'Corso Umberto I', '81034');
insert into spettacolo values (TO_DATE('2022-05-31 20:00', 'yyyy-mm-dd hh24:mi'), TO_DATE('2022-05-31 21:30', 'yyyy-mm-dd hh24:mi'), 'Italiano', 'marvel001', '01', 'Milano', 'Via santa', '20121');
insert into spettacolo values (TO_DATE('2019-08-01 20:00', 'yyyy-mm-dd hh24:mi'), TO_DATE('2019-08-01 22:30', 'yyyy-mm-dd hh24:mi'), 'Italiano', 'disney001', '01', 'Genova', 'Via magazzini', '16128');
insert into spettacolo values (TO_DATE('2019-08-01 20:00', 'yyyy-mm-dd hh24:mi'), TO_DATE('2019-08-01 22:30', 'yyyy-mm-dd hh24:mi'), 'Italiano', 'disney001', '02', 'Genova', 'Via magazzini', '16128');
insert into spettacolo values (TO_DATE('2023-09-09 17:50', 'yyyy-mm-dd hh24:mi'), TO_DATE('2023-09-09 19:30', 'yyyy-mm-dd hh24:mi'), 'Italiano', 'warner001', '01', 'Milano', 'Via santa', '20121');

--Biglietti (codice_biglietto, prezzo, codice_carrello, ora_inizio_spettacolo, ora_fine_spettacolo, fila_posto, numero_posto, nome_sala, citta_cinema, via_cinema, cap_cinema)
insert into biglietto values ('ticket001', 10, 'cart001', TO_DATE('2022-04-30 20:00', 'YYYY-MM-DD hh24:mi'), TO_DATE('2022-04-30 21:30', 'YYYY-MM-DD hh24:mi'), 'A', 1, '01', 'Mondragone', 'Corso Umberto I', '81034');
insert into biglietto values ('ticket002', 5, 'cart002', TO_DATE('2022-05-07 19:00', 'YYYY-MM-DD hh24:mi'), TO_DATE('2022-05-07 20:30', 'yyyy-mm-dd hh24:mi'), 'A', 2, '01', 'Mondragone', 'Corso Umberto I', '81034');
insert into biglietto values ('ticket020', 10, 'cart001', TO_DATE('2023-05-29 20:00', 'yyyy-mm-dd hh24:mi'), TO_DATE('2023-05-29 21:30', 'yyyy-mm-dd hh24:mi'), 'B', 1, '01', 'Casoria', 'Via mediterraneo', '80026');

--Per effettuare il pagamento dei biglietti è necessario eseguire l'apposita procedura
execute update_data_pagamento('cart001', TO_DATE('01/05/2022 16:31', 'dd/mm/yyyy hh24:mi'));
execute update_data_pagamento('cart002', TO_DATE('01/05/2022 17:01', 'dd/mm/yyyy hh24:mi'));
execute update_data_pagamento('cart014', TO_DATE('21/06/2029 17:01', 'dd/mm/yyyy hh24:mi'));

--Professionisti (codice_professionista, nome, data_di_nascita)
insert into professionista values ('prf01', 'Benedict Cumberbatch', TO_DATE('19/07/1976', 'dd/mm/yyyy'));
insert into professionista values ('prf02', 'Sam Raimi', TO_DATE('19/07/1976', 'dd/mm/yyyy'));
insert into professionista values ('prf03', 'Elizabeth Olsen', TO_DATE('16/02/1989', 'dd/mm/yyyy'));
insert into professionista values ('prf04', 'Chris Hemsworth', TO_DATE('11/08/1983', 'dd/mm/yyyy'));
insert into professionista values ('prf05', 'Taika Waititi', TO_DATE('16/08/1975', 'dd/mm/yyyy'));

--Chi recita chi? (codice_film, codice_professionista, nome_ruolo)
insert into recitato_da values ('marvel001', 'prf01', 'Dottor Strange');
insert into recitato_da values ('marvel001', 'prf03', 'Wanda Maximoff');
insert into recitato_da values ('marvel002', 'prf04', 'Thor');

--Chi dirige chi? (codice_film, codice_professionista)
insert into diretto_da values ('marvel001', 'prf02');
insert into diretto_da values ('marvel002', 'prf05');

--Recensioni, con un voto da 1 a 10 ☹ (id_recensione, votoprotagonista, votoantagonista, vototrama, votocolonnasonora, votoscenografia, data_pubblicazione, codice_biglietto, codice_film)
insert into recensione values ('rec01', 6, 5, 9, 7, 10, TO_DATE('1/06/2022', 'dd/mm/yyyy'), 'ticket001', 'marvel001');
```

Per un elenco completo degli inserimenti, guardare il file relativo al DML.

Data Control Language

L'utente "gestore" può gestire in pieno controllo gli inserimenti dei valori nelle entità Cinema, Sala, Posto, Spettacolo, Biglietto, Utente, Promozione, Promozione_fedelta, Carrello, Film, Genere_film, Genere, Diretto_da, Recitato_da, Professionista e Compagnia_di_produzione. E' suo compito gestire tutti i valori importanti all'interno del DataBase, come lo è anche quello di assicurare un comportamento adeguato da parte degli utenti.
La procedura Assegna_carta_fedelta viene inoltre eseguita giornalmente dallo scheduler, come mostrato in precedenza.
Un gestore non può visualizzare le carte di credito degli utenti, ma può eseguire la procedura "update_data_pagamento", per aggiornare la data di pagamento dell'ordine in caso un utente decida di pagare in contanti una prenotazione.

```
GRANT CONNECT, CREATE SESSION TO gestore;
GRANT ALL ON Utente TO gestore;
GRANT ALL ON Promozione TO gestore;
GRANT ALL ON Promozione_fedelta TO gestore;
GRANT ALL ON Carrello TO gestore;
GRANT ALL ON Biglietto TO gestore;
GRANT SELECT ON Recensione TO gestore;
GRANT DELETE ON Recensione TO gestore;
GRANT ALL ON Spettacolo TO gestore;
GRANT ALL ON Sala TO gestore;
GRANT ALL ON Cinema TO gestore;
GRANT ALL ON Posto TO gestore;
GRANT ALL ON Film TO gestore;
GRANT ALL ON Genere_film TO gestore;
GRANT ALL ON Genere TO gestore;
GRANT ALL ON Diretto_da TO gestore;
GRANT ALL ON Recitato_da TO gestore;
GRANT ALL ON Professionista TO gestore;
GRANT ALL ON Compagnia_di_produzione TO gestore;
GRANT EXECUTE update_data_pagamento TO gestore;
GRANT EXECUTE cliente_del_mese TO gestore;
GRANT EXECUTE inserisci_posti TO gestore;
GRANT EXECUTE annulla_ordine TO gestore;
GRANT EXECUTE FakeUser TO gestore;
GRANT EXECUTE Assegna_carta_fedelta TO gestore;
GRANT EXECUTE elimina_biglietti_scaduti TO gestore;
```

L'utente "user" ha un accesso limitato alle varie entità del DB, siccome il suo scopo è essenzialmente quello di essere un semplice utente che acquista biglietti, scrive recensioni e annulla ordini in caso di cambio di idea. L'unico controllo completo che ciascun utente registrato possiede è quello sull'entità "Recensione", in quanto oltre che aggiungere sarà possibile modificare, eliminare le proprie recensioni.
Le viste sono utili per avere una lista dei film da vedere al cinema, con le query adeguate, oltre che avere un resoconto generale della valutazione di un film in particolare. E' possibile pagare dall'app tramite carta di credito, "update_data_pagamento" ha infatti lo scopo di aggiornare la data di pagamento e consentirgli la visione.

```
GRANT CONNECT, CREATE SESSION TO user;
GRANT INSERT ON Utente TO user;
GRANT INSERT ON Carrello TO user;
GRANT INSERT ON Biglietto TO user;
GRANT INSERT ON Carta_di_credito TO user;
GRANT DELETE ON Carta_di_credito TO user;
GRANT ALL ON Recensione TO user;
GRANT SELECT ON Spettacolo TO user;
GRANT SELECT ON SalaIMAX3D TO user;
GRANT SELECT ON MediaRecensione TO user;
GRANT SELECT ON RegistaAttore TO user;
GRANT SELECT ON VistaFilm TO user;
GRANT SELECT ON Orari_spettacoli TO user;
GRANT EXECUTE update_data_pagamento TO user;
GRANT EXECUTE annulla_ordine TO user;
GRANT EXECUTE auto_acquista TO user;
GRANT EXECUTE elimina_biglietti_scaduti TO user;
```