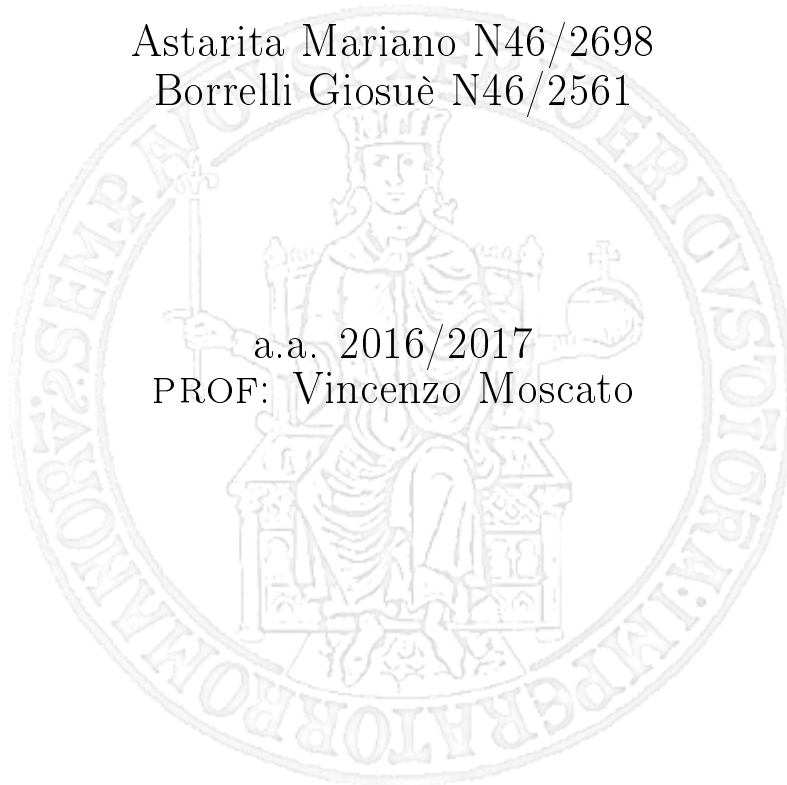


TESINA DI BASI DI DATI

L^AT_EX

Astarita Mariano N46/2698
Borrelli Giosuè N46/2561

a.a. 2016/2017
PROF: Vincenzo Moscato



Contents

1	Introduzione	1
1.1	Specifiche sui dati	1
2	Progettazione Concettuale della Base Di Dati	2
2.1	Schema E/R Portante	2
2.2	Analisi delle associazioni dello schema portante	4
3	Progettazione Logica	6
3.1	Fase di trasformazione	6
3.1.1	Risoluzione degli attributi composti e multivalore	6
3.1.2	Risoluzione delle gerarchie	7
3.2	Fase di traduzione	9
4	Progettazione Fisica	10
4.1	Dimensionamento della Base Di Dati	10
4.2	Creazione Database	18
4.3	Creazione oggetti e definizione dei vincoli	19
4.4	Creazione dei Ruoli	26
5	Query	27
6	Viste	29
7	Stored Procedure	31
8	Trigger	34
9	Controllo Di Concorrenza	36
10	Controllo Di Affidabilità	37
10.1	File di log	37
10.2	Checkpoint	37
10.3	Backup	37

List of Figures

1	Schema E/R Portante	3
2	Schema E/R Finale	8
3	Dimensionamento Tabella Utenti	11
4	Dimensionamento Tabella Prenotazioni	11
5	Dimensionamento Tabella Serie	12
6	Dimensionamento Tabella Posti	12
7	Dimensionamento Tabella Palinsesti	13
8	Dimensionamento Tabella Proiezioni	13
9	Dimensionamento Tabella Sale	14
10	Dimensionamento Tabella Cinema	14
11	Dimensionamento Tabella Film	15
12	Dimensionamento Tabella Film In Uscita	15
13	Dimensionamento Tabella Artisti	16
14	Dimensionamento Tabella Telefoni Artisti	16
15	Dimensionamento Tabella Cast	17
16	Dimensionamento Totale	17
17	creazione della tabella Utenti	19
18	creazione della tabella Artisti	19
19	creazione della tabella Prenotazioni	20
20	creazione della tabella Serie	20
21	creazione della tabella Posti	20
22	creazione della tabella Palinsesti	21
23	creazione della tabella Proiezioni	21
24	creazione della tabella Cinema	22
25	creazione della tabella Sale	23
26	creazione della tabella Film	23
27	creazione della tabella Film In Uscita	24
28	creazione della tabella Telefoni Artisti	24
29	creazione della tabella Cast	24
30	Query Attori	27
31	Query Cinema	28
32	Vista_Palinsesto	30
33	Stored Procedure Prenotazioni Storiche	32
34	Stored Procedure Contatore Palinsesto	33
35	Trigger sulla tabella Utenti	35
36	Trigger sulla tabella Palinsesti	35

1 Introduzione

Questo lavoro ha l'obiettivo di progettare una base di dati per la Gestione dei Cinema collocati sul territorio Svizzero. In particolare ogni società gestisce la prenotazione per i cinema di un singolo cantone. La suddetta società intende fornire ai propri utenti un servizio centralizzato per la consultazione della programmazione delle proiezioni di film presso i vari cinema ed l'acquisto on-line dei biglietti. Ovviamente le informazioni riguardanti le disponibilità dei film offerti da ogni cinema sono di competenza delle singole strutture, ciascuna delle quali ha nel tempo sviluppato un proprio sistema informativo.

1.1 Specifiche sui dati

Al fine di una visione più chiara della progettazione della base di dati, si sono analizzate le specifiche dei dati e ristrutturati i requisiti nel seguente modo:

- i **Cinema** (circa 50 per ogni Cantone Svizzero) sono rappresentati da un codice, dal nome, indirizzo, città, nazione, cantone in cui risiede, numero telefonico e numero di sale.
- le **Sale** di ciascun cinema sono identificate univocamente da un ID e presentano il nome, la superficie espressa in metri quadrati e la relativa capienza.
- i **Film in programmazione** (circa 1000) presentano il titolo, la recensione, la regia, il genere, l'anno di uscita, il paese, la durata, gli attori che vi hanno recitato, la data di uscita e il distributore.
- gli **Utenti** (circa 1000) sono identificati con username, password, cognome, nome ed indirizzo e-mail.
- la **Programmazione** delle proiezioni dei film nelle varie sale dei cinema presenta, per ogni film in programmazione, il giorno, l'orario e il prezzo del biglietto.
- le **Prenotazioni** e l'acquisto di biglietti per i posti prescelti dagli utenti, per le varie proiezioni, sono identificate con un codice di prenotazione, con la data in cui è stata effettuata e l'avvenuto pagamento.

Dove i dati numerici fanno riferimento alla mole di dati che la base di dati dovrà gestire, prevista per il prossimo anno.

2 Progettazione Concettuale della Base Di Dati

2.1 Schema E/R Portante

Il primo passo per la progettazione dello schema E/R è stato quello di individuare le entità fondamentali, al fine di costruire uno schema E/R portante. Esse sono risultate essere: *Palinsesto* (rappresentante la programmazione), *Prenotazione*, *Utente*, *Cinema*, *Sala* e *Film in Uscita*. In figura 1 alla pagina seguente, si può notare come l'entità *Utente* sia collegata alla *Prenotazione*, essendo, appunto, nelle possibilità dell'utente quella di poter effettuare una o più prenotazioni; questa, a sua volta, è collegata con l'entità *Palinsesto*. Le entità *Cinema*, *Film In Uscita* e *Sala* sono collegate a *Palinsesto* che raccoglie, appunto, tutte le informazioni riguardo al *Film In Uscita*, il *Cinema* nel quale è proiettato, la tipologia di proiezione, il prezzo, la data e l'ora (questi ultimi due valori non devono essere necessariamente conosciuti, fin dall'inizio). Una singola occorrenza di *Palinsesto* presenta, quindi, tutte le informazioni utili per un *Utente* che vuole effettuare una *Prenotazione*. *Posto* è stato costruito come un attributo composto di *Prenotazione*, rappresenta, infatti, il numero di posto e la fila in cui l'utente vuole effettuare la prenotazione. Inoltre, lo stesso attributo composto (*Posto*) lo ritroviamo collegato anche alla *Sala*, dato che la *Sala* di un cinema è costituita da un certo numero di posti. *Film In Uscita* può essere vista come un'entità figlia di *Film*, per scindere il film in uscita da tutti i film, in uscita o già proiettati in passato; per tale motivo, la gerarchia (Film-Film In Uscita) è stata realizzata secondo la configurazione *parziale/disgiunta*.

Inoltre, è stata aggiunta l'entità *Artista* che si è scelto di non considerare come semplice attributo dell'entità *Film In Uscita* ma come entità autonoma, avente come attributi Matricola, Telefoni, Nome e Cognome. Altre entità sono state aggiunte allo schema, che sono *Regista* e *Attore*. Quest'ultime sono specializzazioni dell'entità *Artista*, per cui è stata realizzata una gerarchia del tipo *totale/sovrapposta*; sovrapposta perché un dato artista può assumere sia il ruolo di regista che quello di attore, e totale perché si suppone che gli Artisti di interesse in questa applicazione siano i soli registi e attori.

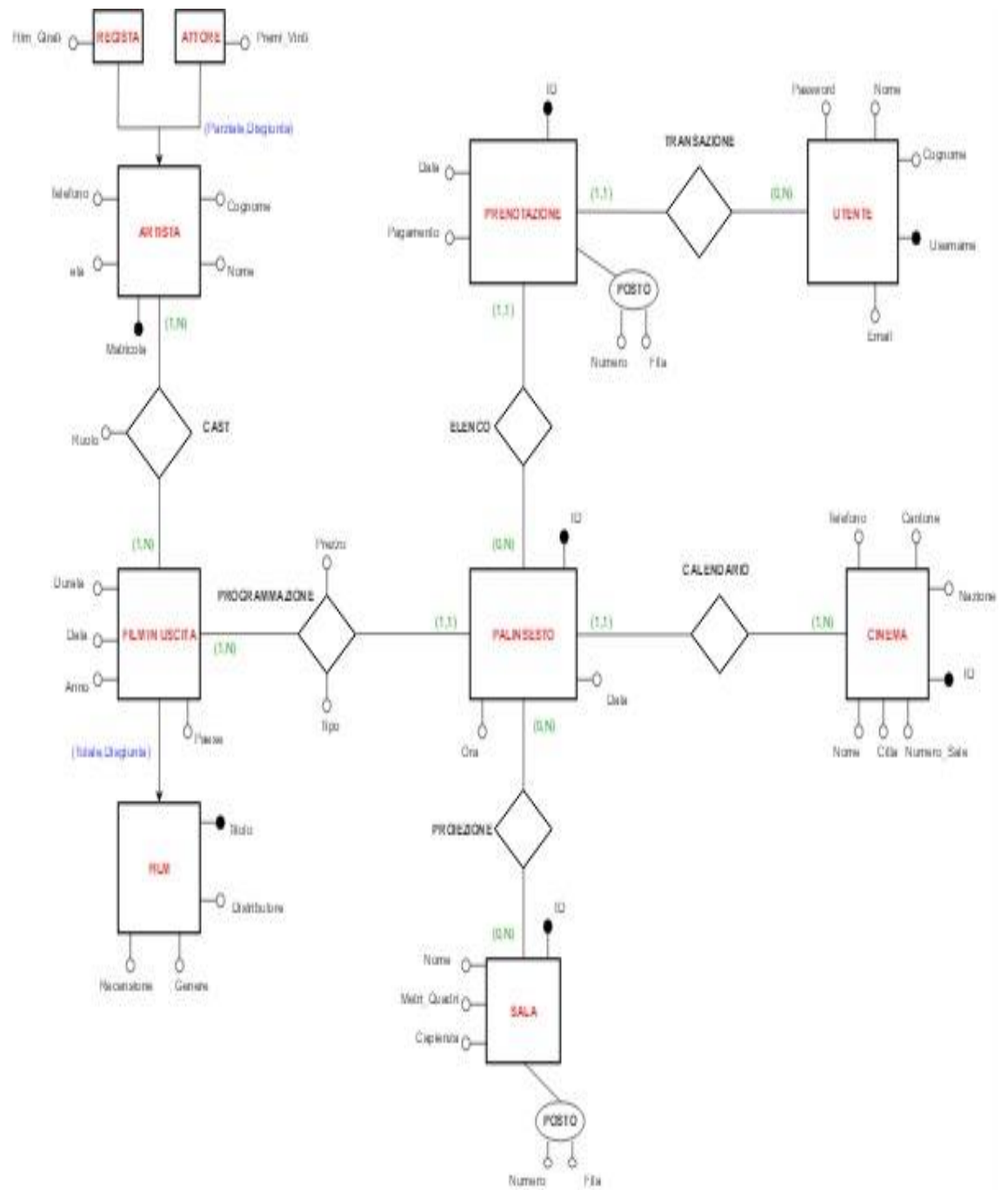


Figure 1: Schema E/R Portante

2.2 Analisi delle associazioni dello schema portante

Con il termine *associazione* si fa riferimento al legame concettuale fra due o più entità, rilevate in una applicazione di interesse. In figura 1 sono mostrate le diverse associazioni, di seguito descritte:

- *Transazione*: in questa associazione, l'entità Utente partecipa con cardinalità $(0,N)$, infatti egli può decidere di effettuare una o più prenotazioni, ma anche di non effettuarne affatto; invece, Prenotazione partecipa all'associazione con cardinalità $(1,1)$ dato che una determinata Prenotazione potrà far riferimento al solo utente che l'ha effettuata ed è creata solo nel momento in cui l'Utente è in procinto di confermare le operazioni effettuate.
- *Elenco*: in questa associazione, l'entità Palinsesto partecipa con cardinalità $(0,N)$; una singola occorrenza di Palinsesto può avere, da parte dell'utente, da 0 fino ad N prenotazioni. Prenotazione, invece, partecipa all'associazione con cardinalità $(1,1)$ perchè una determinata Prenotazione fa riferimento ad una sola occorrenza di palinsesto e viceversa.
- *Calendario*: questa associazione collega le entità Palinsesto e Cinema. Ovviamente, una determinata occorrenza di Palinsesto può riferirsi ad un solo Cinema, mentre si può ritrovare uno stesso Cinema in una o N occorrenze di Palinsesto (almeno in una, altrimenti quel cinema non avrebbe ragione di esistere, per i nostri scopi).
- *Proiezioni*: in questa associazione Palinsesto partecipa con cardinalità $(0,N)$, dove la cardinalità minima può essere giustificata considerando che in un'occorrenza di Palinsesto si potrebbe non avere, subito, la conoscenza della Sala che proietterà il determinato film, mentre la cardinalità massima mette in evidenza la possibilità che lo stesso film possa uscire in più sale contemporaneamente alla stessa ora. Sala partecipa all'associazione con cardinalità $(0,N)$, può infatti succedere che in una determinata sala non vi sia alcuna proiezione o che vi siano proiettati uno o più film e che quindi appaia N occorrenze di Palinsesto.
- *Programmazione*: questa associazione ha come attributi propri *tipo* e *prezzo*, dove il primo indica se il film in uscita è in 2D o in 3D ed il secondo il relativo prezzo. Palinsesto partecipa all'associazione con cardinalità $(1,1)$ perchè in una singola occorrenza di palinsesto vi sarà un solo film in uscita. Film In Uscita partecipa all'associazione con cardinalità $(1,N)$ un film deve comparire in almeno un'occorrenza di Palinsesto (altrimenti non avrebbe ragione di esistere).

- *Cast*: Quest'associazione ha come attributo *ruolo* che indica il ruolo effettivo dell' Artista nel Film (ad esempio: protagonista, antagonista, figurante ecc). Film In Uscita partecipa all'associazione con cardinalità $(1,N)$, dato che in un film vi deve essere almeno un artista (altrimenti non avrebbe ragione di esistere). Mentre Artista partecipa all'associazione con cardinalità $(1,N)$, perché un artista partecipa ad almeno un film (altrimenti non avrebbe ragione di esistere) ma può aver recitato anche in altri film, quindi N .



3 Progettazione Logica

In questa fase è stato definito lo schema relazionale della base di dati in termini di relazioni e vincoli. Si divide in 2 parti:

1. Trasformazione dello schema concettuale in uno schema analogo ma finale, risolvendo le generalizzazioni, gli attributi multivalore e composti.
2. Traduzione dello schema finale nello schema relazionale.

3.1 Fase di trasformazione

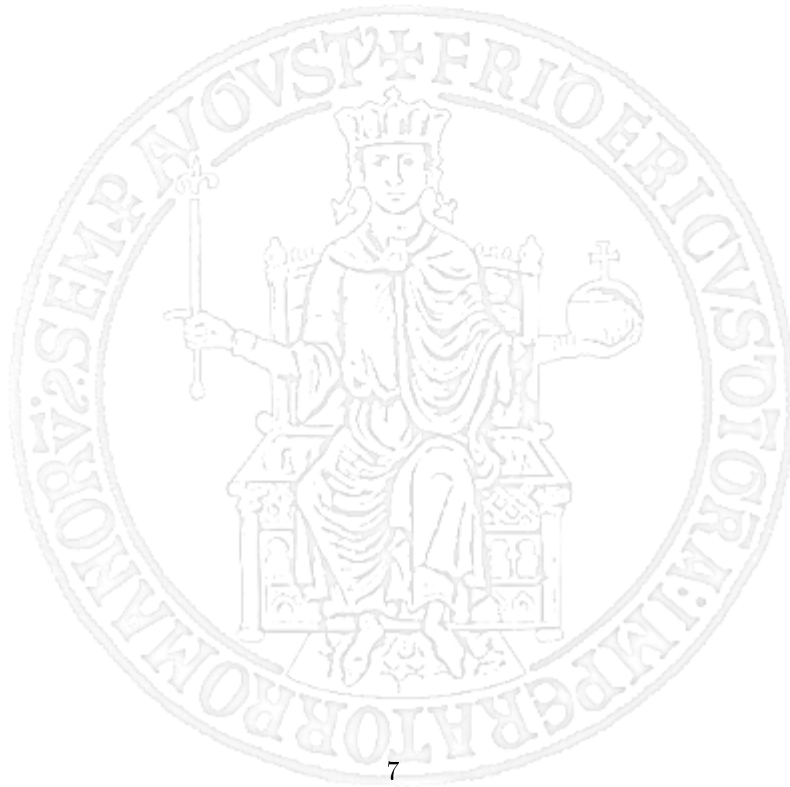
3.1.1 Risoluzione degli attributi composti e multivalore

Applicando la regola di trasformazione, un attributo composto può essere semplificato sostituendolo con tanti attributi semplici quanti sono gli attributi componenti. Per quanto riguarda l'attributo multivalore, deve essere eliminato e deve essere introdotta una nuova entità, corrispondente al concetto rappresentato dall'attributo che si sta eliminando, legata all'entità originale da una associazione del tipo uno a molti, che conserverà lo stesso nome e gli stessi attributi ad eccezione dell'attributo multivalore. Quindi, nel nostro caso, avremmo dovuto sostituire l'attributo composto Posto, con i relativi attributi semplici; tuttavia, siccome si tratta di un oggetto realmente esistente, che può avere un'esistenza autonoma, è stato sostituito da una nuova entità collegata a Prenotazione e a Sala, in modo che l'utente, all'atto della prenotazione, possa scegliere il posto che desidera. L'entità Posto avrà come attributi il codice identificativo (ID), il Numero e la fila. L'associazione che collega il posto alla prenotazione prende il nome di Serie. Posto partecipa all'associazione con cardinalità (0,1) perchè il posto può non essere prenotato, oppure può essere prenotato da un unico Utente, mentre Prenotazione partecipa all'associazione con cardinalità (1,N) perchè la prenotazione effettuata da un Utente può contenere al minimo un posto o al massimo N. L'associazione che collega Posto alla Sala viene chiamata Appartenenza. Posto partecipa all'associazione con cardinalità (1,1) perchè il posto sarà relativo ad una determinata sala, mentre Sala partecipa all'associazione con cardinalità (1,N) perchè una sala potrà avere al minimo un posto e al massimo N. L'attributo multivalore Telefoni dell'entità Artista è stato trasformato in una nuova entità Telefono, con il relativo attributo. Chiamiamo Recapiti l'associazione che collega Telefono ad Artisti; Telefono partecipa all'associazione con cardinalità (1,1) perchè quel numero di telefono può essere posseduto da un solo Artista, mentre Artista partecipa all'associazione con cardinalità (1,N) perchè un Artista può avere almeno un numero di telefono e al massimo N.

3.1.2 Risoluzione delle gerarchie

In questo sottoparagrafo, è descritta la risoluzione delle gerarchie dello schema E/R Portante. Facendo riferimento alla figura 2, la gerarchia composta da Artista, che si specializza in Regista-Attore è stata risolta applicando l' accorpamento delle sottoclassi nella superclasse: questa soluzione prevede l'eliminazione delle sottoclassi, con la conseguente aggiunta alla superclasse di tutti gli attributi e tutte le associazioni cui le sottoclassi partecipano. Alla superclasse viene, inoltre, aggiunto un attributo per distinguere il tipo di un'occorrenza, ovvero per distinguere a quale delle sottoclassi tale occorrenza appartiene. Nel nostro caso abbiamo aggiunto l'attributo Tipo sull'entità Artista, che specifica se si tratta di un attore o di un regista, considerando il valore opzionale ad entrambi gli attributi ,cioè (0,1), perchè un Artista può essere solo un Regista, solo un Attore ma può assumere anche entrambi i ruoli; infine sono stati aggiunti gli attributi Film_Girati e Premi_Vinti.

La gerarchia Film, Film In Uscita è stata risolta applicando una soluzione mista con la creazione di una nuova associazione Tipologia. Film partecipa a questa associazione con cardinalità (0,1) perchè un film potrebbe essere già uscito e potrà uscire un unica volta, mentre Film In Uscita partecipa all'associazione con cardinalità (1,1).



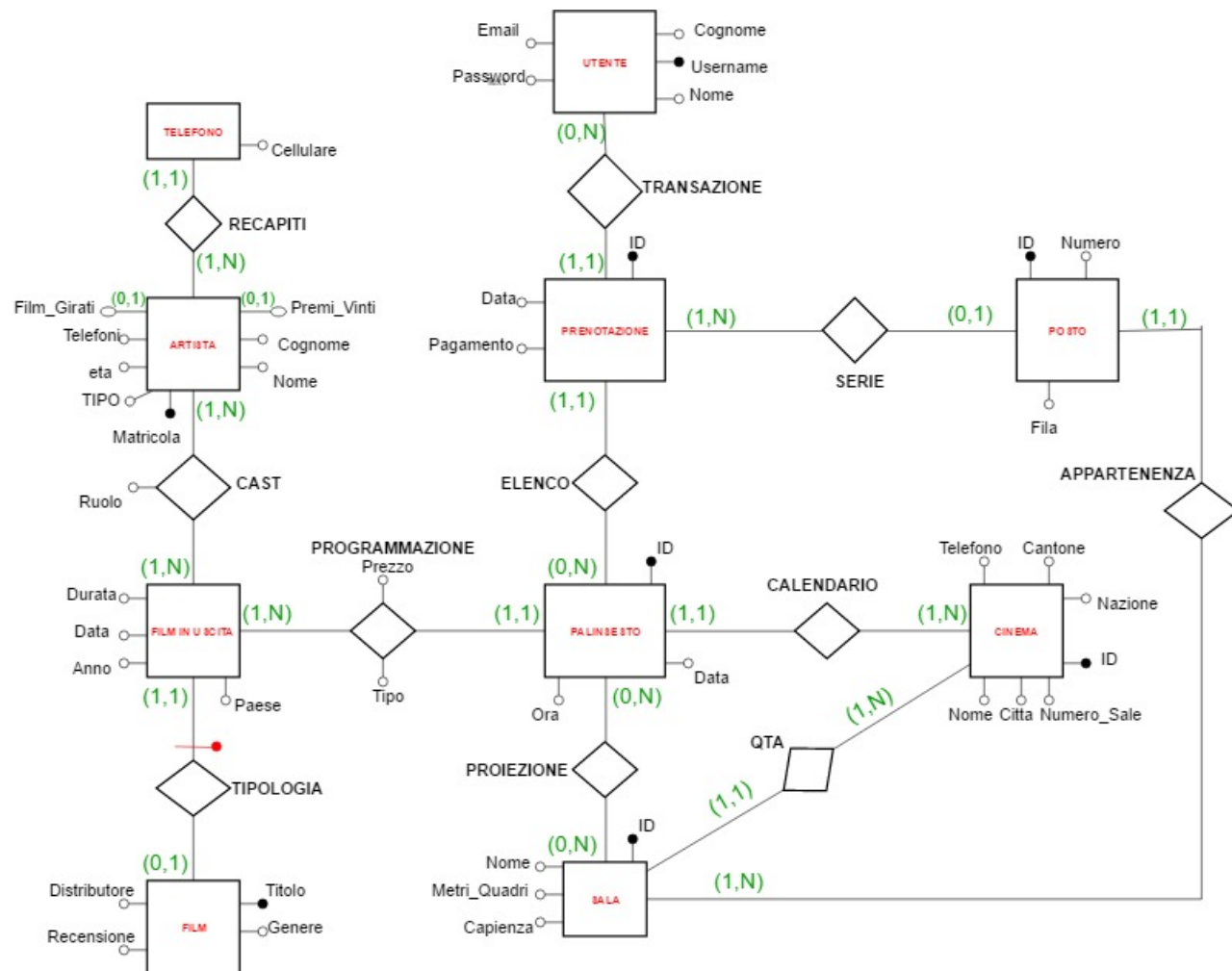


Figure 2: Schema E/R Finale

3.2 Fase di traduzione

Al fine di completare la progettazione logica della nostra base di dati, come mostrato in figura 2, è stato creato uno schema relazionale attenendoci alle usuali regole di traduzione. Le entità sono state, quindi, trasformate in relazioni aventi come attributi gli attributi delle entità e come chiavi primarie gli identificatori delle entità.

UTENTI (*USERNAME*, Password, Nome, Cognome, e-mail);

PRENOTAZIONI (*ID*, Pagamento, Palinsesto:PALINSESTI, Data, Utente:UTENTI);

PALINSESTI (*ID*, Ora , Data, film_in_uscita:FILM_IN_USCITA, Tipo, Prezzo, Cinema:CINEMA);

SALE (*ID*, Nome, Capienza, Metri_quadri, Cinema:CINEMA);

PROIEZIONI (*Palinsesto*:PALINSESTI , *Sala*:SALE);

CINEMA (*ID*, Nome, Città , Nazione, Telefono, Cantone, Via, Civico, Cap, Nazione);

FILM (*TITOLO*, Recensione, Genere, Distributore);

FILM_IN_USCITA (*Film*:FILM, Durata, anno, paese, Data);

TELEFONI_ARTISTA (Cellulare, Artista:ARTISTI);

CAST (*Artista*:ARTISTI, *Film_In_Uscita*:FILM_IN_USCITA, Ruolo);

ARTISTI (Matricola, Nome, Cognome, Età, Tipo, Film_Girati, Premi_Vinti)
;

SERIE(*Prenotazione*:PRENOTAZIONI, Posto:POSTI);

POSTI(*ID*, Numero, Fila, Sala:SALE);

4 Progettazione Fisica

La progettazione fisica di una base di dati prevede l'attuazione delle seguenti fasi:

1. dimensionamento fisico della base di dati, sia globale che a livello dei singoli oggetti, in termini di calcolo di storage richiesto su disco;
2. creazione del database;
3. creazione degli oggetti della base di dati e definizione dei vincoli;
4. creazione degli utenti/ruoli.

4.1 Dimensionamento della Base Di Dati

A partire dalle informazioni sul volume dei dati e, una volta scelti i tipi degli attributi da utilizzare nell'implementazione di ogni tabella, è possibile effettuare una stima dei costi in termini di occupazione di memoria della base di dati per la successiva fase di implementazione. Si ricorda che l'occupazione in termini di byte per il DBMS Oracle dei tipi di dati più diffusi è:

- NUMBER (x): $(\lfloor x/2 \rfloor + 2)$ byte;
- DATE 7 byte;
- CHAR(x): x byte;
- VARCHAR2(x): x byte.

Di seguito viene riportata per ogni tabella una stima dell'occupazione di memoria a regime, all'atto della messa in esercizio della base di dati (initial). Inoltre, in figura 11, è stato inserito il tipo CLOB per la memorizzazione di testi di grandi dimensioni (fino a 4 GB). I dati veri e propri relativi ai CLOB andranno memorizzati su uno spazio di memoria dedicato, mentre sullo spazio corrente verranno memorizzati solo i puntatori (4 byte) a tali dati.

UTENTI			
			1000 UTENTI
ATTRIBUTO	TIPO	BYTE	INITIAL
USERNAME	VARCHAR2(16)	16	16K
PASSWORD	VARCHAR2(16)	16	16K
NOME	VARCHAR2(50)	50	50K
COGNOME	VARCHAR2(50)	50	50K
EMAIL	VARCHAR2(20)	20	20K
TOTALE			152K

Figure 3: Dimensionamento Tabella Utenti

PRENOTAZIONI			
			8000 OCC.
ATTRIBUTO	TIPO	BYTE	INITIAL
ID	NUMBER(7)	6	48K
PAGAMENTO	CHAR(1)	1	8K
PALINSESTO	NUMBER(7)	6	48K
DATA	DATE	7	56K
UTENTE	VARCHAR2(16)	16	128K
TOTALE			288K

Figure 4: Dimensionamento Tabella Prenotazioni

SERIE			
			960'000 OCC.
ATTRIBUTO	TIPO	BYTE	INITIAL
PRENOTAZIONE	NUMBER(7)		6 5760K
POSTO	NUMBER(7)		6 5760K
			TOTALE
			11520K

Figure 5: Dimensionamento Tabella Serie

POSTI			
			6000 OCC.
ATTRIBUTO	TIPO	BYTE	INITIAL
ID	NUMBER(7)		6 36K
NUMERO	NUMBER(2)		3 18K
FILA	CHAR(1)		1 6K
SALA	NUMBER(7)		6 36K
			TOTALE
			96K

Figure 6: Dimensionamento Tabella Posti

PALINSESTI			
			200000 OCC.
ATTRIBUTO	TIPO	BYTE	INITIAL
ID	NUMBER(7)		6 1200K
DATA	DATE		7 1400K
FILM IN USCITA	VARCHAR2(50)		50 10000K
TIPO	VARCHAR2(2)		3 600K
PREZZO	NUMBER(2,2)		3 600K
CINEMA	NUMBER(7)		6 1200K
			TOTALE
			16200K

Figure 7: Dimensionamento Tabella Palinsesti

PROIEZIONI			
			200000 OCC.
ATTRIBUTO	TIPO	BYTE	INITIAL
PALINSESTO	NUMBER(7)		6 1200K
SALA	NUMBER(7)		6 1200K
			TOTALE
			2400K

Figure 8: Dimensionamento Tabella Proiezioni

SALE			
			800 OCC.
ATTRIBUTO	TIPO	BYTE	INITIAL
ID	NUMBER(7)		6 4,8K
NOME	VARCHAR2(2)		2 1,6K
CAPIENZA	NUMBER(3)		3 2,4K
METR_QUADRI	NUMBER(4)		4 3,2K
CINEMA	NUMBER(7)		6 4,8K
NUMERO	NUMBER(7)		6 4,8K
			TOTALE
			21,2K

Figure 9: Dimensionamento Tabella Sale

CINEMA			
			50 CINEMA
ATTRIBUTO	TIPO	BYTE	INITIAL
ID	NUMBER(7)		6 0,3K
NOME	VARCHAR2(50)		50 2,5K
CITTA	VARCHAR2(50)		20 1K
VIA	VARCHAR2(20)		20 1K
CIVICO	VARCHAR2(5)		5 0,25K
CAP	NUMBER(4)		4 0,2K
TELEFONO	NUMBER(10)		7 0,35K
NAZIONE	VARCHAR2(20)		20 1K
CANTONE	VARCHAR2(20)		20 1K
			TOTALE
			7,6K

Figure 10: Dimensionamento Tabella Cinema

FILM			
			1000 occ.
TITOLO	VARCHAR2(50)	50	50K
RECENSIONE	CLOB	4	4K
GENERE	VARCHAR2(50)	50	50K
DISTRIBUTORE	VARCHAR2(50)	50	50K
TOTALE			154K

Figure 11: Dimensionamento Tabella Film

FILM IN USCITA			
			1000 occ.
FILM	VARCHAR2(50)	50	50K
DURATA	DATE	7	7K
ANNO	NUMBER(4)	4	4K
PAESE	VARCHAR2(20)	20	20K
DATA	DATE	7	7K
TOTALE			88K

Figure 12: Dimensionamento Tabella Film In Uscita

ARTISTI			
			100'000 occ.
MATRICOLA	VARCHAR2(10)	10	1000K
NOME	VARCHAR2(50)	50	5000k
COGNOME	VARCHAR2(50)	50	5000K
ETA'	NUMBER(2)	3	300K
TIPO	VARCHAR2(20)	20	2000K
PREMI VINTI	VARCHAR2(50)	50	5000K
FILM GIRATI	VARCHAR2(50)	50	5000K
			TOTALE
			23300K

Figure 13: Dimensionamento Tabella Artisti

TELEFONI ARTISTI			
			100'000 OCC.
CELLULARE	NUMBER(10)	7	700K
ARTISTA	VARCHAR(10)	10	1000K
			TOTALE
			1700K

Figure 14: Dimensionamento Tabella Telefoni Artisti

CAST			
			100'000 occ.
ARTISTA	VARCHAR2(10)	10	1000K
FILM	VARCHAR2(50)	50	5000K
RUOLO	VARCHAR2(20)	20	2000K
			TOTALE
			8000K

Figure 15: Dimensionamento Tabella Cast

TABELLA	INITIAL
UTENTI	152K
PRENOTAZIONI	288K
SERIE	11520K
POSTI	96K
PALINSESTI	16200K
PROIEZIONI	2400K
SALE	21,2K
CINEMA	7,6K
FILM	154K
FILM IN USCITA	88K
ARTISTI	23300K
TELEFONI ARTISTI	1700K
CAST	8000K
	TOTALE
	64 MB

Figure 16: Dimensionamento Totale

4.2 Creazione Database

In questa fase saranno creati due tablespace uno atto a contenere i dati delle varie tabelle ed uno dedicato alla memorizzazione dei tipi CLOB. Il peso totale della base di dati risulta essere di 64MB, si alloca quindi un tablespace di 128MB per tenere conto di eventuali errori e di altri oggetti come viste, trigger e indici. Essendo loggati quindi come SYSTEM, si alloca nel database un tablespace di 128MB:

```
CREATE TABLESPACE ts_cinema DATAFILE  
'C:/tablespace_CINEMA/ts_cinema.dbf' SIZE 128 M;
```

```
CREATE TABLESPACE ts_recensione DATAFILE  
'C:/tablespace_CINEMA/ts_Recensione.dbf' SIZE 100 M;
```

In seguito, si procede alla creazione dell'utente che servirà da DBA per la nostra base, attraverso il seguente statement:

```
CREATE USER Amministratore DEFAULT TABLESPACE ts_cinema IDENTIFIED BY 12345;  
GRANT DBA, UNLIMITED TABLESPACE TO Amministratore;
```



4.3 Creazione oggetti e definizione dei vincoli

In questa fase creiamo gli oggetti del database, ossia tabelle e sequenze, e definiamo i vincoli di integrità referenziale attraverso il comando ALTER TABLE.

```
create table utenti(  
    username varchar2(16) primary key,  
    password varchar2(16) not null,  
    Nome varchar2(50) not null,  
    cognome varchar2(50) not null,  
    email varchar2(20),  
    constraint PK_US check(length(username)>6),  
    constraint PK_PW check(length(password)>8)  
)storage (initial 152 K);
```

Figure 17: creazione della tabella Utenti

```
create table artisti(  
    matricola varchar2(10) primary key,  
    nome varchar2(50) not null,  
    cognome varchar2(50) not null,  
    eta number(2) not null,  
    tipo varchar2(20) not null,  
    premi_vinti varchar2(50),  
    film_girati varchar2(50)  
)storage (initial 23300 K);
```

Figure 18: creazione della tabella Artisti

```

create table prenotazioni(
    id number(7) primary key,
    pagamenti char(1) not null,
    palinsesto number(7) not null,
    data date,
    utente varchar2(16) not null,
    constraint CK_PAG check( pagamenti=0
    or pagamenti=1)
)storage(initial 288 K);

```

Figure 19: creazione della tabella Prenotazioni

```

create table serie(
    prenotazione number(7) primary key,
    posto number(7)not null
)storage(initial 11520 K);

```

Figure 20: creazione della tabella Serie

```

create table posti(
    id number(7)primary key,
    numero number(2)not null,
    fila char(1) not null,
    sala number(7)not null
)storage(initial 96 K);

```

Figure 21: creazione della tabella Posti

```

create table palinsesti(
    id number(7) primary key,
    data date,
    film_in_uscita varchar2(50) not null,
    tipo varchar2(2) not null,
    prezzo number (2) not null,
    cinema number (7) not null
    constraint CH_TIPO check((tipo='2D')
    or (tipo='3D'))
)storage(initial 16200 K);

```

Figure 22: creazione della tabella Palinsesti

```

create table proiezioni(
    palinsesto number(7),
    sala number(7),
    primary key (palinsesto,sala)
)storage(initial 2400 K);

```

Figure 23: creazione della tabella Proiezioni


```

create table cinema(
    id number(7) primary key,
    nome varchar2(50) not null,
    citta varchar2(50) not null,
    via varchar2(20) not null,
    civico varchar2(5) not null,
    cap number(4) not null,
    telefono number(10) not null,
    nazione varchar2(20) not null,
    cantone varchar2(20) not null,
    constraint CH_TEL check(length(telefono)=10),
    constraint CH_CANT check (cantone='Argovie' OR
cantone='Appenzello Interno'
OR cantone='Appenzello Esterno'
OR cantone='Basilea-Città'
OR cantone='Basilea-Campagnarna'
OR cantone='Friburgo' OR cantone='Ginevra'
OR cantone='Glarona'
OR cantone='Grigioni' OR cantone='Giura'
OR cantone='Lucerna'
OR cantone='Neuchâtel' OR cantone='Nidvaldo'
OR cantone='Obvaldo'
OR cantone='Sciaffusa' OR cantone='Svitto'
OR cantone='Soletta'
OR cantone='San Gallo' OR cantone='Turgovia'
OR cantone='Ticino'
OR cantone='Uri' OR cantone='Vallese'
OR cantone='Vaud'
OR cantone='Zugo' OR cantone='Zurigo')
) storage(initial 7,6 K);

```

Figure 24: creazione della tabella Cinema

```

create table sale(
    id number(7) primary key,
    nome varchar2(2) not null,
    capienza number(3) not null,
    metri_quadrati number(4),
    cinema number(7) not null,
    numero number(7)
) storage (initial 21,2 K);

```

Figure 25: creazione della tabella Sale

```

create table film(
    titolo varchar2(50) primary key,
    recensione clob,
    genere varchar2(50) not null,
    distributore varchar2(50) not null

) lob (recensione) store as
Lob_film (tablespace ts_recensione)
storage (initial 154 K );

```

Figure 26: creazione della tabella Film

```

create table film_in_uscita(
    film varchar2(50) primary key,
    durata date not null,
    anno number(4),
    data date not null,
    paese varchar2(20) not null
)storage (initial 88 K);

```

Figure 27: creazione della tabella Film In Uscita

```

create table telefoni_artisti (
    cellulare number(10) not null,
    artista varchar2(10) not null,
    constraint CH_CIN check (length(cellulare)=10)
)storage (initial 1700 K);

```

Figure 28: creazione della tabella Telefoni Artisti

```

create table cast(
    artista varchar2(10),
    film_in_uscita varchar2(50),
    ruolo varchar2(20) not null,
    primary key(artist, film_in_uscita)
)storage (initial 8000 K);

```

Figure 29: creazione della tabella Cast

Il linguaggio SQL fornisce apposite primitive per la manipolazione degli schemi delle basi di dati. In particolare, sugli schemi di basi di dati precedentemente creati risulta possibile modificare le definizioni di relazioni mediante l'utilizzo del comando ALTER TABLE:

```
ALTER TABLE SERIE ADD CONSTRAINT FK_SER_PRE  
FOREIGN KEY (PRENOTAZIONE) REFERENCES PRENOTAZIONI(ID);
```

```
ALTER TABLE SERIE ADD CONSTRAINT FK_SER_POS  
FOREIGN KEY (POSTO) REFERENCES POSTI(ID);
```

```
ALTER TABLE POSTI ADD CONSTRAINT FK_POS_SAL  
FOREIGN KEY (SALA) REFERENCES SALE(ID);
```

```
ALTER TABLE PRENOTAZIONI ADD CONSTRAINT FK_PRE_UTE  
FOREIGN KEY (UTENTE) REFERENCES UTENTI(USERNAME);
```

```
ALTER TABLE PALINSESTI ADD CONSTRAINT FK_PAL_FILM  
FOREIGN KEY (FILM_IN_USCITA) REFERENCES FILM_IN_USCITA(FILM);
```

```
ALTER TABLE PALINSESTI ADD CONSTRAINT FK_PAL_CINEMA  
FOREIGN KEY (CINEMA) REFERENCES CINEMA(ID);
```

```
ALTER TABLE SALE ADD CONSTRAINT FK_SAL_CIN  
FOREIGN KEY (CINEMA) REFERENCES CINEMA(ID);
```

```
ALTER TABLE PROIEZIONI ADD CONSTRAINT FK_LOC_PAL  
FOREIGN KEY (PALINSESTO) REFERENCES PALINSESTI(ID);
```

```
ALTER TABLE PROIEZIONI ADD CONSTRAINT FK_LOC_SAL  
FOREIGN KEY (SALA) REFERENCES SALE(ID);
```

```
ALTER TABLE FILM_IN_USCITA ADD CONSTRAINT FK_FI_FILM  
FOREIGN KEY (FILM) REFERENCES FILM(TITOLO);
```

```
ALTER TABLE TELEFONI_ARTISTI ADD CONSTRAINT FK_TEL_ART  
FOREIGN KEY (ARTISTA) REFERENCES ARTISTI(MATRICOLA);
```

```
ALTER TABLE CAST ADD CONSTRAINT FK_CAST_ART  
FOREIGN KEY (ARTISTA) REFERENCES ARTISTI(MATRICOLA);
```

```
ALTER TABLE CAST ADD CONSTRAINT FK_CAST_FILM  
FOREIGN KEY (FILM_IN_USCITA) REFERENCES FILM_IN_USCITA(FILM);
```

4.4 Creazione dei Ruoli

In alcuni DBMS, se si crea una risorsa con le credenziali di un utente che ne ha il privilegio, non solo l'utente diviene proprietario della risorsa stessa, ma viene generato in automatico una schema con lo stesso nome dell'utente che conterrà tutte le risorse che questo in futuro andrà a definire. Si può quindi affermare che, in generale, ad ogni risorsa è associato in maniera implicita o esplicita una schema di appartenenza. In un DBMS, oltre all'utente DBA già definito, si considerano altri 2 utenti della base di dati. Di seguito il codice SQL della creazione dei ruoli:

```
Create role utente;
```

```
Grant connect to utente;
```

```
grant select, update on utenti to utente;
```

```
grant select, update on prenotazioni to utente;
```

```
grant select on palinsesti to utente;
```

```
grant select on film_in_uscita to utente;
```

```
grant select on sale to utente;
```

```
grant select on cinema to utente;
```

```
grant select on cast to utente;
```

```
grant select on posti to utente;
```

```
Create role operatore;
```

```
Grant connect to operatore;
```

```
grant all privileges on prenotazioni to operatore;
```

```
grant all privileges on palinsesti to operatore;
```

```
grant select on film_in_uscita to operatore;
```

```
grant select,update on cinema to operatore;
```

```
grant select,update on sale to operatore;
```

```
grant select,update on posti to operatore;
```

```
grant select,update on serie to operatore;
```

5 Query

Le query su una Base di Dati utilizzano gli operatori dell' Algebra Relazionale: Proiezioni, Selezione e Join, quindi le operazioni classiche del modello relazionale per il recupero delle informazioni che operano sia su una singola tabella che su più, eventualmente aggregando informazioni. In primo esempio, come in figura 30, è richiesto di selezionare solo gli attori che recitano nel film I Guardiani Della Galassia. In secondo esempio, come è riportato in figura 31, è richiesta la visualizzazione dei cinema che proiettano King Arthur.

```
SQL> select  a.nome as nome_attore, c.ruolo as ruolo_attore
  2  from cast c join artisti a on c.artista=a.matricola
  3  where a.tipo = 'Attore';
```

NOME_ATTORE	RUOLO_ATTORE
Megan	antagonista
Sabrina	figurante

```
SQL> _
```

Figure 30: Query Attori

SQL Run SQL Command Line

```
SQL*Plus: Release 11.2.0.2.0 Production on Lun Mag 15 14:43:36 2017
Copyright (c) 1982, 2014, Oracle. All rights reserved.

SQL> connect
Enter user-name: Amministratore
Enter password:
Connected.
SQL> select c.nome as Nome_Cinema
  2  from cinema c join palinsesti p on c.id=p.cinema
  3  where p.film_in_uscita = 'King Arthur';

NOME_CINEMA
-----
The Space
Armida
Montil
```

Figure 31: Query Cinema

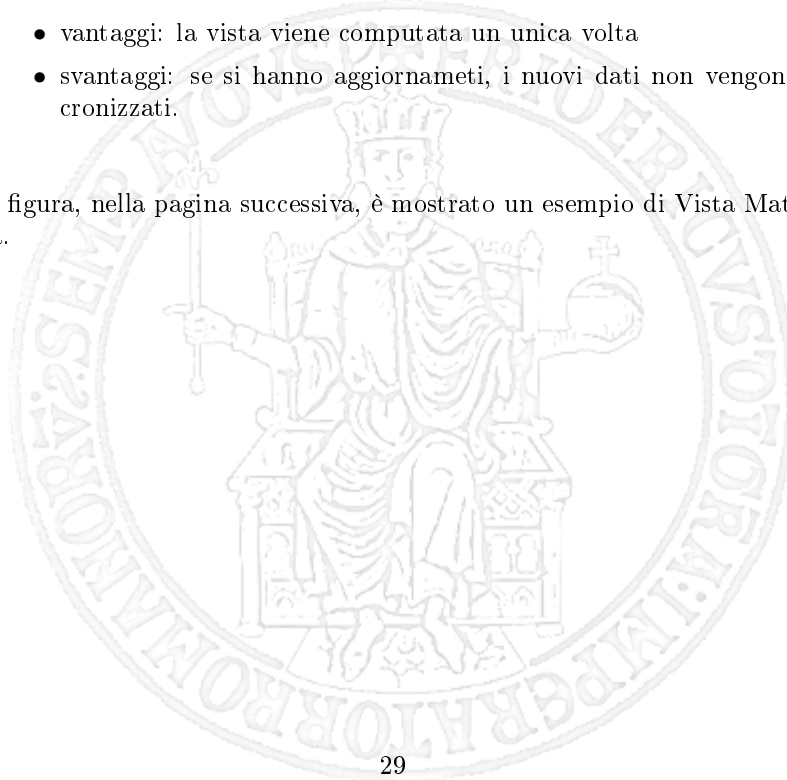


6 Viste

Una vista è una tabella che viene descritta in termini di altre tabelle, dette di base. In generale, una vista è una relazione virtuale, nel senso che le sue tuple non sono effettivamente memorizzate nella base di dati, quanto piuttosto ricavabili, attraverso interrogazioni, da altre tuple presenti nella base dati. Una vista costituisce, dunque, una interfaccia da mettere a disposizione di utenti o di applicazione per le interrogazioni: si tratta di dati usati di frequente che possono anche non esistere fisicamente. Ciò limita le operazioni sulle viste, in particolare quelle di aggiornamento. Quindi le viste sono tabelle che non fanno parte di una Base Dati ma sono generate attraverso le interrogazioni SQL. Le viste possono essere di due tipi:

1. Viste create al momento dell'interrogazione, quindi ricreate quando servono.
 - svantaggi: la vista viene computata ogni qualvolta serve.
 - vantaggi: la vista permette di aggiornare i dati dalle tabelle da cui deriva.
2. Viste Materializzate che vengono create come tabelle nel Data Base e permangono nello schema della Base Dati.
 - vantaggi: la vista viene computata un'unica volta
 - svantaggi: se si hanno aggiornamenti, i nuovi dati non vengono sincronizzati.

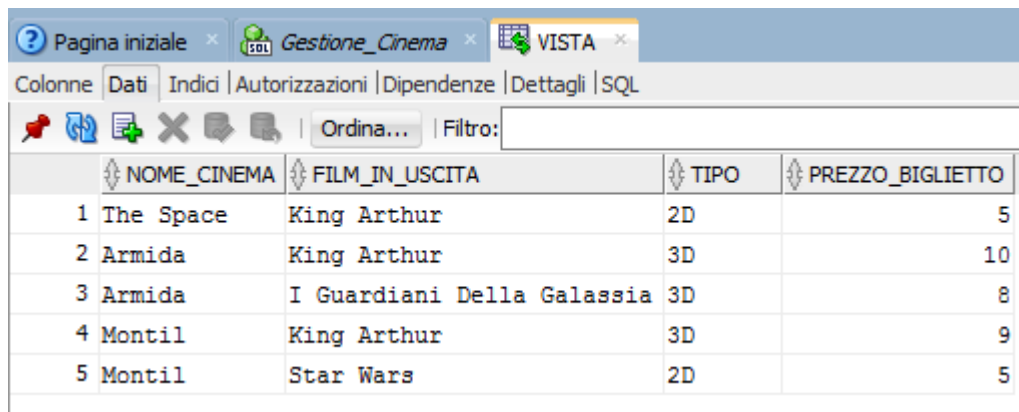
In figura, nella pagina successiva, è mostrato un esempio di Vista Materializzata.




```

create materialized view vista as
select c.nome as Nome_Cinema,
p.film_in_uscita
as Film_In_Uscita,
p.tipo as Tipo, p.prezzo as Prezzo_Biglietto
from palinsesti p join cinema c
on c.id=p.cinema

```



	NOME_CINEMA	FILM_IN_USCITA	TIPO	PREZZO_BIGLIETTO
1	The Space	King Arthur	2D	5
2	Armida	King Arthur	3D	10
3	Armida	I Guardiani Della Galassia	3D	8
4	Montil	King Arthur	3D	9
5	Montil	Star Wars	2D	5

Figure 32: Vista_Palinsesto

7 Stored Procedure

Il PL/SQL (Procedural Language/ Structured Query Language) è un sofisticato linguaggio di programmazione utilizzato per accedere ed elaborare le informazioni gestite da un database ORACLE. Esso permette di realizzare le cosiddette stored procedure, ovvero programmi che risiedono sul DBMS e che quindi possono essere elaborati più velocemente ed efficientemente, utili nella pratica, per implementare operazioni che richiedono una forte interazione con i dati presenti nella base di dati.

PL/SQL colma il gap che sussiste tra la tecnologia dei database relazionale ed linguaggio procedurale estendendo le istruzioni SQL con le caratteristiche tipiche dei linguaggi procedurali. Il primo esempio di stored procedure, mostrato in figura 33, rappresenta una query che quando ritorna una tabella contenente un certo numero di righe, permette di definire un cursore per elaborare tutte le righe tornate dalla query e tenere traccia di quella che è attualmente in elaborazione, risolvendo così il problema del conflitto di impedenza tra le variabili del linguaggio e l'insieme dei record restituito da un operazione di SELECT.

Un secondo esempio, mostrato in figura 34, permette di inserire una nuova occorrenza di Palinsesto, creando l' ID (contatore) in maniera automatica.

La seguente procedura, figura 33, ha come obbiettivo di spostare le prenotazioni vecchie degli utenti in una tabella storica prenotazioni:

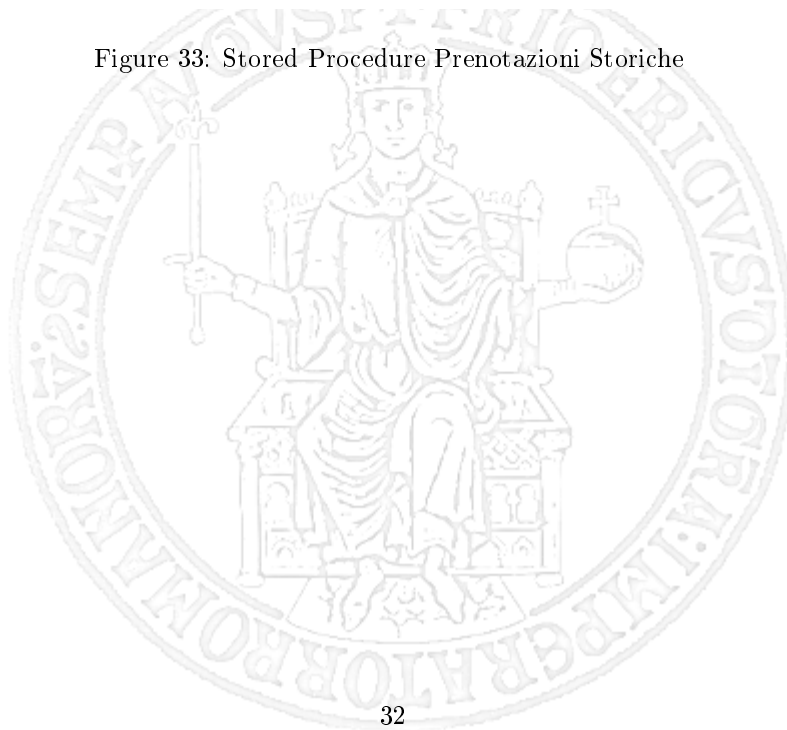


```

spostamento_prenotazioni;
declare
    id_pre prenotazioni.id%type;
    utente_pre prenotazioni.utente%type;
    cursor c is
        select id,utente
        from prenotazioni
        where data < '10-MAG-17';
begin
    open c;
    loop
        fetch c into id_pre,utente_pre;
        delete from prenotazioni
        where id=id_pre;
        insert into storiche_prenotazioni
        values (id_pre, utente_pre);
    exit when c%notfound ;
    end loop;
    close c;
    commit;
end;

```

Figure 33: Stored Procedure Prenotazioni Storiche



```

create or replace procedure insert_palinsesti(
    data_in palinsesti.data%type,
    film_in_uscita_in palinsesti.film_in_uscita%type,
    tipo_in palinsesti.tipo%type,
    prezzo_in palinsesti.prezzo%type,
    cinema_in palinsesti.cinema%type)
as
begin
    declare
        count palinsesti.id%type;
    begin
        select mexi(id) into count
        from palinsesti
        insert into palinsesti(id,data,film_in_uscita
        tipo,prezzo,cinema)
        values(count + 1, data_in,filme_in_uscita_in
        tipo_in,prezzo_in,cinema_in);
        commit;
    end
end insert_palinsesti;

```

Figure 34: Stored Procedure Contatore Palinsesto

Inoltre, Oracle mette a disposizione l'oggetto sequenza per gestire l'ID. Quindi invece di scrivere la procedura possiamo anche scrivere:

```

create sequence s;
insert into palinsesti values(s.nextval,...):

```

8 Trigger

I Trigger sono specifiche di azioni su una Base Dati che devono essere eseguite in automatico, li possiamo vedere come una particolare stored procedure, attivata (Fire) automaticamente dal DBMS a valle di eventi che possono presentarsi su di una Base Dati. I trigger rendono una Base Dati attiva, implicano regole di Business che sono particolari vincoli che non riusciamo ad esprimere con l' SQL. Con i Trigger possiamo applicare la politica di reazione di Cascade seguita da un Update che in generale non è supportata da oracle. I Trigger soddisfano il paradigma E.C.A:

- E: operazione su una base dati che fa partire in automatico il trigger.
- C: condizione che viene valutata prima che il trigger parte.
- A: L'azione che il trigger deve compiere.

I Trigger sulla base dell'evento che li scatena si dividono in:

- Trigger DML: scatenati da eventi del tipo insert, update e delete su tabelle dette TARGET.
- Trigger DDL: scatenati da eventi di Data Definition Language, ad esempio la creazione di una tabella o la cancellazione.

Inoltre abbiamo due modalità di avvio del Trigger:

1. Immediata: il Trigger parte subito dopo l'evento, che a sua volta si divide in:
 - Before: il Trigger parte prima che avvenga l'inserimento.
 - After: il Trigger parte dopo che l'occorrere dell'evento.
2. Differita: Il Trigger aspetta un evento esterno prima di partire.

Sulla base della Granularità, cioè il numero di volte che viene eseguito il Trigger, esso si divide in altre due categorie:

1. Row - Level: i Trigger si attivano per ogni riga modificata dall'evento nella tabella Target.
2. Statement - Level: i Trigger si attivano una volta sola.

Ogni Trigger opera su tabelle dette di TRANSIZIONI, quindi non operano direttamente sulle tabelle ma su copie. Le tabelle di Transizione sono costituite dall'insieme delle righe della tabella TARGET coinvolta nell'EVENTO. In figura 35, è riportato un esempio di Trigger che dopo 1000 Utenti non permette più di effettuare nuove registrazioni. Invece, in un secondo esempio, figura 36, è riportato un trigger che propaga l'aggiornamento dell'identificativo di un Cinema sull'occorrenza di Palinsesto.

```

create or replace trigger controllo_utenti
before insert on utenti
for each statement
begin
    declare errore exception;
    count integer;
    count2 integer;
begin
    select count (*) into count
    from utenti;
    if(count > 1000) then
        raise errore;
    end if;
    when errore raise
    _application_error(-100,limite raggiunto);
    end;
    select count2 (*) into count2
    from:new
    if (count + count2 > 1000)
end;

```

Figure 35: Trigger sulla tabella Utenti

```

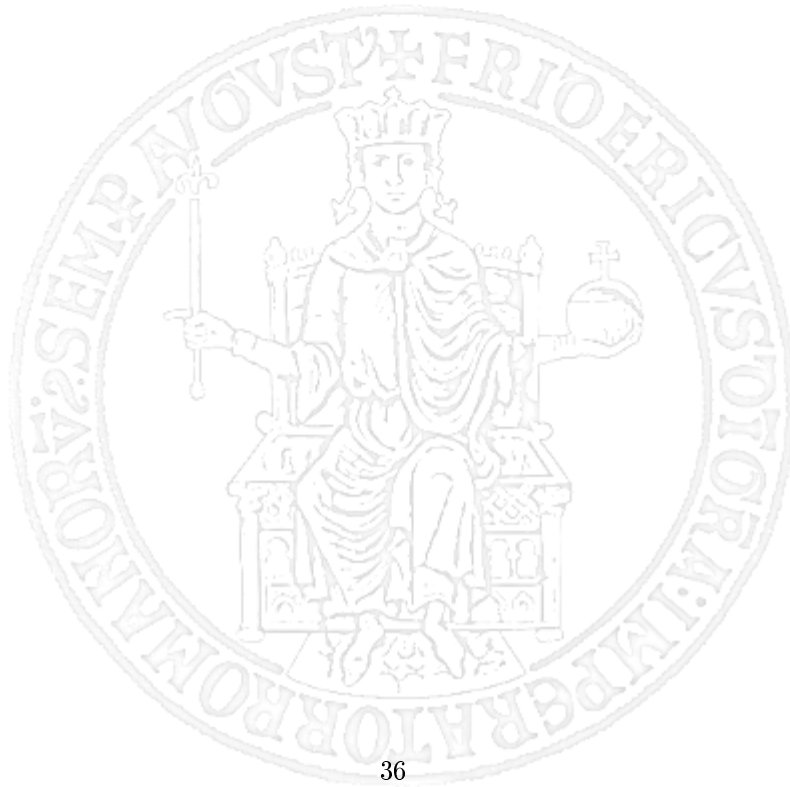
create or replace trigger update_cascade
after update on cinema
for each row
begin update palinsesti set cinema =: new.id
where cinema =: old.id;
end;
end;

```

Figure 36: Trigger sulla tabella Palinsesti

9 Controllo Di Concorrenza

Per il sistema in esame è stata prevista una tecnica di gestione della concorrenza pessimistica basata su lock, ovvero dei speciali blocchi applicati ad ogni oggetto della base di dati. Le transazioni che vogliono accedere agli oggetti devono, innanzitutto, acquisire i lock in scrittura o lettura e poi effettuare le operazioni, per poi "rilasciare" i lock. Una gestione di questo tipo è necessaria per il nostro centro di prenotazione, in quanto esso può ricevere anche molte richieste contemporaneamente, dovendo gestire più strutture distribuite sul territorio. Per evitare anomalie di qualsivoglia tipo, che nel nostro caso potrebbero risultare fatali, la scelta del metodo basato su lock ci è sembrata la più appropriata. In particolare, verrà messa in atto una delle tecniche più note: il Locking a due fasi. Con l'utilizzo di questa tecnica viene imposto che ogni transazione sulla nostra base di dati deve, in una prima fase detta crescente, acquisire tutti i lock necessari per portare a termine le operazioni, e nella seconda fase, detta decrescente, rilasciare tutti i lock precedentemente acquisiti. In questo modo si avrà che ogni transazione, una volta terminate le operazioni e rilasciati i lock, non potrà più acquisire altri lock. Questa tecnica ci permette di prevenire possibili anomalie, quali letture inconsistenti e aggiornamenti fantasma. Per prevenire anche eventuali anomalie di lettura sporca, aggiungiamo la condizione che i lock possano essere rilasciati soltanto dopo i commit o gli abort, facendo così del nostro metodo un Locking a due fasi stretto (2PL stretto).



10 Controllo Di Affidabilità

10.1 File di log

Per garantire la stabilità del sistema e la conservazione dei dati, è stato previsto per il sistema un Gestore di Affidabilità. Per esso è previsto, innanzitutto, un file di log che registra tutte le operazioni svolte dalle transazioni nel loro ordine di esecuzione; in particolare ci si è avvalsi della regola del commit precedenza, ovvero ogni transazione scriverà sul log i relativi record prima di effettuare il commit, garantendo in questo modo che le azioni possano essere rifatte. Le scritture sulla base di dati verranno eseguite in modalità immediata, ovvero ogni scrittura sul file di log sarà immediatamente seguita da una scrittura sulla base di dati, in modo da evitare inconsistenze e gestire in modo semplice il file di log.

10.2 Checkpoint

Sono previsti inoltre dei checkpoint ogni 3 giorni, durante la notte, quando verranno sospese le accettazioni di richieste di ogni tipo dalle transazioni e verranno trasferiti in memoria di massa tutti i dati allo stato attuale, per poi riprendere l'accettazione delle operazioni entro il mattino.

10.3 Backup

Tutto il contenuto della base di dati verrà copiato su due supporti di memorizzazione (hard disk) esterni, i quali verranno settimanalmente aggiornati e sincronizzati con il contenuto della base di dati e verranno conservati in località diverse. Queste ultime copie di backup potranno essere utilizzate in caso di perdita dei dati sul supporto principale.

