GIORDANO VINCENZO
N86003039
vincenzo.giordano14@studenti.unina.it

Progetto per gli Insegnamenti di Basi di Dati e di Object Orientation

Realizzazione di un database relazione e di un applicativo Java che permetta la gestione di un cinema multisala (traccia gruppo singolo)

SINTESI DEI CONTENUTI

Il tutto sintetizza in modo semplice e dettagliato l'attività svolta per entrambi i corsi con linee guida per la maggior comprensione

DOCENTI:
Prof.re Adriano Peron
Prof.re Silvio Barra
Prof.re Vincenzo Norman Vitale

INDICE

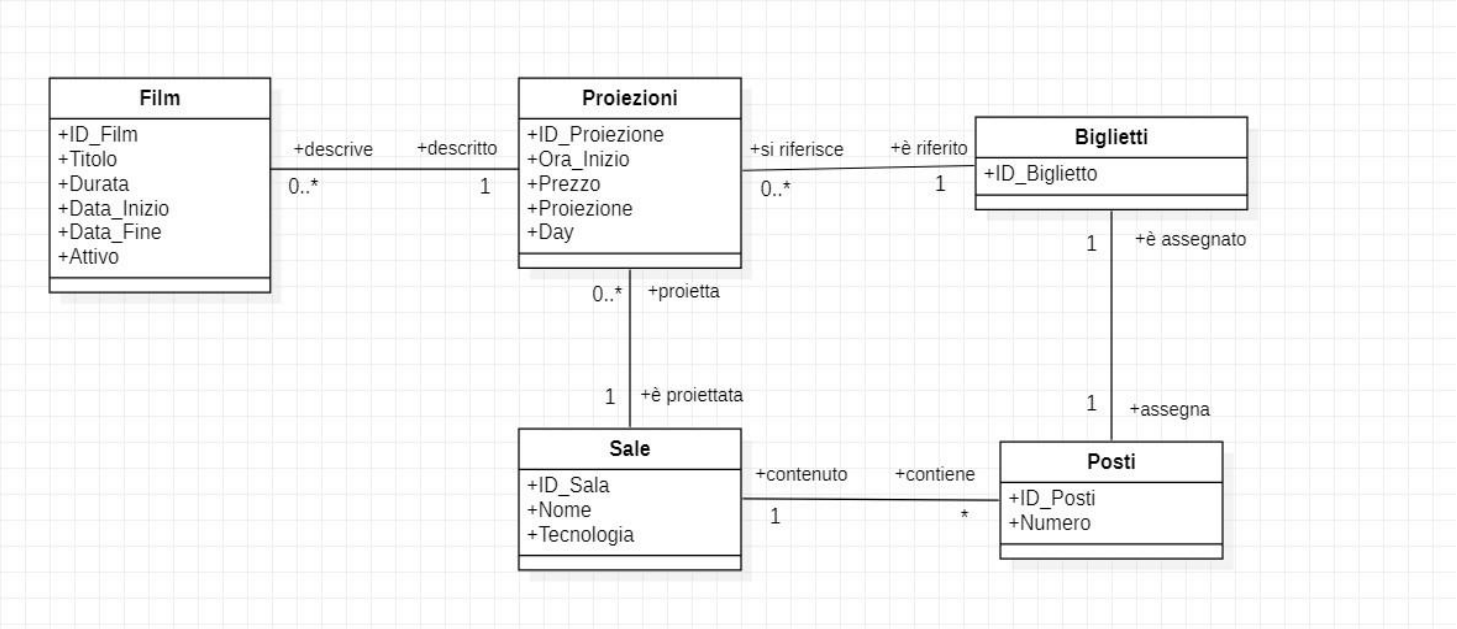
Descrizione

Si progetterà ed implementerà un sistema informativo che permette la gestione di un cinema multisala.
È stata progettata una Base di Dati idonea alla memorizzazione e gestione di classi e oggetti che vi si trovano all'interno del cinema.
Il db sarà in grado di gestire i film proiettati in ciascuna sala.
L'utente che ne farà uso potrà utilizzare il db a suo piacimento, inserendo, cancellando e/o visualizzando il contenuto di qualsiasi classe.
Il db sarà in grado di stimare quali sono le fasce orarie di maggior affluenza, gli spettacoli più remunerativi e le sale maggiormente occupate durante gli orari di maggior affluenza.
Sono state implementate cinque entità che contengono tutte le informazioni necessarie e utili allo svolgimento del sistema informativo (Film, Proiezioni, Sale, Biglietti, Posti).

Progettazione concettuale

Partiamo dal livello di astrazione più alto, la progettazione concettuale. Possiamo definire uno schema concettuale che permette di rappresentare meglio il problema descritto sopra. In tale schema, che verrà rappresentato usando un Class Diagram UML, si evidenzieranno le entità rilevanti ai fini della rappresentazione dei dati e le relazioni che intercorrono tra esse. Si evidenzieranno anche eventuali vincoli da far rispettare.

a. Class Diagram



b. Dizionario dei dati

In questo paragrafo verranno mostrati con maggiore dettaglio le classi, le associazioni ed eventuali vincoli

i. Dizionario delle classi

CLASSE	DESCRIZIONE	ATTRIBUTI	IDENTIFICATORE
FILM	Mostra i film presenti nel cinema	IdFilm, Titolo, Durata, DataInizio, DataFine, Attivo	IdFilm
PROIEZIONI	Rappresenta la specifica proiezione del cinema, ovviamente associata ad una sala e ad un film presente	IdProiezione, OraInizio, CodFilm, CodSala, Prezzo, Proiezioni, Day	IdProiezione
BIGLIETTI	Consente la gestione dei singoli biglietti per le varie proiezioni	IdBiglietto, CodPosto, Cod_Proiezione	IdBiglietto
SALE	Rappresenta la specifica sala del cinema con l'eventuale tecnologia (IMAX, Audio Dolby)	IdSala, Nome, Tecnologia	IdSala
POSTI	Vi sono presenti i posti in ciascuna sala con la relativa numerazione	IdPosti, Numero, Lettera, Cod_Sala	IdPosti

ii. Dizionario delle associazioni

ASSOCIAZIONE	DESCRIZIONE	CLASSI E MOLTEPLICITÀ
DESCRITTO - DESCRIVE	Un film è descritto da una proiezione, la proiezione descrive un film	Film - Proiezione 1 : N
PROIETTA – È PROIETTATA	La proiezione è proiettata in una sala, una sala proietta la proiezione	Proiezione - Sala 1 : N
SI RIFERISCE – È RIFERITO	Il biglietto si riferisce ad una proiezione, la proiezione è riferita al biglietto	Proiezione - Biglietto 1 : N
È ASSEGNATO - ASSEGNA	Il biglietto è assegnato ad un posto, il posto assegna un biglietto	Biglietto - Posto 1 : 1
CONTIENE- CONTENUTO	La sala contiene posti , i posti sono contenuti in una sala	Sale – Posti 1 : N

iii. Dizionario dei vincoli

NOME VINCOLO	DESCRIZIONE
CHECK_FILM_PROIEZIONI	Trigger che entra in azione per controllare se una sala è già occupata durante la visione di un film, se lo è genera un messaggio di errore

Progettazione Logica

In questo capitolo verrà tradotto lo schema concettuale in uno **schema logico**.

Si precisa che negli schemi relazionali che seguiranno le **chiavi primarie** sono indicate con una singola sottolineatura mentre le **chiavi esterne** con una doppia sottolineatura.

a. Schema relazionale

FILM (IdFilm, Titolo, Durata, DataInizio, DataFine, Attivo)

PROIEZIONI (IdProiezione, Ora_Inizio, CodFilm, CodSala, Prezzo, Proiezione, Day)

BIGLIETTI (IdBiglietto, CodPosto, CodProiezione)

SALE (IdSala, Nome, Tecnologia)

POSTI (IdPosti, Numero, Lettera, Lettera, CodSala)

Progettazione Fisica

L'ultimo capitolo e quindi l'ultima fase della progettazione di una base di dati è la **progettazione fisica**.

Prima di iniziare la progettazione fisica occorre scegliere un **DBMS** (*DataBase Management System*) che implementi il modello dei dati dello schema logico.

Il **DBMS** utilizzato in questo caso è **PostgreSQL**.

SQL (*Structured Query Language*) è il linguaggio di interrogazione più diffuso tra quelli usati per l'iterazione con i principali **DBMS**.

a. Definizione delle tabelle

```
CREATE TABLE Biglietti
(
  IdBiglietto  INTEGER  NOT NULL,
  CodPosto  INTEGER  NOT NULL,
  CodProiezione  INTEGER  NOT NULL,

  CONSTRAINT pkbiglietti PRIMARY KEY (IdBiglietto),

  CONSTRAINT fkposto FOREIGN KEY (CodPosto) REFERENCES Posti (IdPosti)
  ON UPDATE NO ACTION
  ON DELETE NO ACTION,

  CONSTRAINT fkproiezione FOREIGN KEY (CodProiezione) REFERENCES Proiezioni
  (IdProiezione)
  ON UPDATE NO ACTION
  ON DELETE CASCADE
);
```

```
CREATE TABLE Film
(
  IdFilm  INTEGER  NOT NULL,
  Titolo  VARCHAR(50)  NOT NULL,
  Durata  INTEGER  NOT NULL,
  DataInizio  DATE  NOT NULL,
  DataFine  DATE  NOT NULL,

 CONSTRAINT pkfilm PRIMARY KEY (IdFilm)
);


CREATE TABLE Posti
(
 IdPosti  INTEGER  NOT NULL,
 Numero  INTEGER  NOT NULL,
 Lettera  CHAR  NOT NULL,
 CodSala  INTEGER ,

 CONSTRAINT pkposti PRIMARY KEY (IdPosti),

 CONSTRAINT fksala FOREIGN KEY (CodSala) REFERENCES Sale (IdSala)
 ON UPDATE NO ACTION
 ON DELETE CASCADE
);


CREATE TABLE Proiezioni
(
 IdProiezione  INTEGER  NOT NULL,
 OraInizio  TIME  NOT NULL,
 CodFilm  INTEGER  NOT NULL,
 CodSala  INTEGER  NOT NULL,
 Prezzo  NUMERIC(5,2)  NOT NULL,

 CONSTRAINT pkproiezioni PRIMARY KEY (IdProiezione),

 CONSTRAINT fkfilm FOREIGN KEY (CodFilm) REFERENCES Film (IdFilm)
```

```
  ON UPDATE NO ACTION
  ON DELETE CASCADE,

  CONSTRAINT fksala FOREING KEY (CodSala) REFERENCES Sala (IdSala)
  ON UPDATE NO ACTION
  ON DELETE CASCADE
);
```

```
CREATE TABLE Sale
(
  IdSala  INTEGER  NOT NULL,
  Nome  VARCHAR (50)  NOT NULL,
  Tecnologia VARCHAR (),

  CONSTRAINT pksale PRIMARY KEY (IdSala)
);
```

b. Definizione di eventuali trigger e funzioni

```
CREATE FUNCTION multisala.check_film_proiezioni()

    RETURNS trigger

    LANGUAGE 'plpgsql'

    COST 100

    VOLATILE NOT LEAKPROOF

AS $BODY$

DECLARE

v_idfilm INTEGER;

v_idsala INTEGER;

v_orainiziofilm TIME WITHOUT TIME ZONE;
```

```
v_orafinefilm TIME WITHOUT TIME ZONE;
v_day DATE;
v_duratafilm INTEGER;
BEGIN
   -- dichiarazione nuove variabili
   SELECT NEW."Cod_Film" INTO v_idfilm;
   SELECT NEW."Cod_Sala" INTO v_idsala;
   SELECT NEW."Ora_Inizio" INTO v_orainiziofilm;
   SELECT NEW."Day" INTO v_day;
    SELECT "Durata" INTO v_duratafilm FROM multisala."Film" WHERE "ID_Film" =
v_idfilm;
   SELECT v_orainiziofilm + (v_duratafilm * interval '1 minute') INTO v_orafinefilm;
   -- Controllo nelle proiezioni se la sala è già occupata
   IF(v_orafinefilm < v_orainiziofilm)
   THEN
      SELECT '23:59' INTO v_orafinefilm;
   END IF;
   IF EXISTS (
      SELECT *
      FROM multisala."Proiezioni"
      WHERE "Proiezione" = true AND "Day" = v_day
      AND "Ora_Inizio" BETWEEN v_orainiziofilm AND v_orafinefilm
      AND "Cod_Sala" = v_idsala
   ) THEN
```

```
        RAISE EXCEPTION 'Sala occupata in questo orario';
    ELSEIF EXISTS (
        SELECT *
        FROM multisala."Proiezioni"
        WHERE "Proiezione" = true AND "Day" = v_day
        AND "Ora_Inizio" BETWEEN v_orainiziofilm AND v_orafinefilm
        AND "Cod_Film" = v_idfilm
    ) THEN
        RAISE EXCEPTION 'Film già in programmazione in altra sala';
    ELSE
        RETURN NEW;
    END IF;
END
$BODY$;

ALTER FUNCTION multisala.check_film_proiezioni()
    OWNER TO postgres;

CREATE OR REPLACE FUNCTION multisala.spettacoli_remunerativi()
    RETURNS TABLE(codice_film integer, prezzo numeric)
    LANGUAGE 'sql'
    COST 100
    VOLATILE PARALLEL UNSAFE
    ROWS 1000
```

```
AS $BODY$
SELECT "Cod_Film", SUM("Prezzo")
FROM multisala."Biglietti" LEFT JOIN multisala."Proiezioni" ON "Cod_Proiezione" =
"ID_Proiezione"
GROUP BY "Cod_Film";
$BODY$;

ALTER FUNCTION multisala.spettacoli_remunerativi()
    OWNER TO postgres;

CREATE OR REPLACE FUNCTION multisala.orario_di_maggiore_affluenza()
    RETURNS TABLE(ora time without time zone, numero integer)
    LANGUAGE 'sql'
    COST 100
    VOLATILE PARALLEL UNSAFE
    ROWS 1000

AS $BODY$
SELECT "Ora_Inizio", COUNT(*)
FROM multisala."Biglietti" LEFT JOIN multisala."Proiezioni" ON "Cod_Proiezione" =
"ID_Proiezione"
GROUP BY "Ora_Inizio"
ORDER BY COUNT(*);
$BODY$;
```

```
ALTER FUNCTION multisala.orario_di_maggiore_affluenza()
    OWNER TO postgres;

CREATE OR REPLACE FUNCTION multisala.get_proiezioni(giorno date)
    RETURNS TABLE(id_proiezione integer, ora_inizio time without time zone, film character varying, sala character varying, prezzo numeric, best_orario integer)
    LANGUAGE 'sql'
    COST 100
    VOLATILE PARALLEL UNSAFE
    ROWS 1000

AS $BODY$
SELECT MIN("ID_Proiezione"),"Ora_Inizio", "Titolo", "Nome", "Prezzo",
CASE
    WHEN orario."ora" is null
        THEN 0
    ELSE 1
END AS Best_Orario
FROM multisala."Proiezioni"
LEFT JOIN multisala."Film" ON "Cod_Film" = "ID_Film"
LEFT JOIN multisala."Sale" ON "Cod_Sala" = "ID_Sala"
LEFT JOIN (SELECT "numero", "ora" FROM multisala.orario_di_maggiore_affluenza()) AS orario ON orario."ora" = "Ora_Inizio"
WHERE "Proiezione" = true AND "Day" = giorno AND "Attivo" = true
GROUP BY "Ora_Inizio", "Titolo", "Nome", "Prezzo", orario."ora"
ORDER BY "Ora_Inizio", "Titolo", "Nome", "Prezzo"
```

```
$BODY$;

ALTER FUNCTION multisala.get_proiezioni(date)
    OWNER TO postgres;

CREATE OR REPLACE FUNCTION multisala.get_new_biglietto(
id_proiezione integer,
numero_posto integer,
lettera_posto character)
    RETURNS integer
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE PARALLEL UNSAFE
AS $BODY$
DECLARE
v_sala_proiezione int;
v_id_posto int;
BEGIN
SELECT "Cod_Sala" INTO v_sala_proiezione
FROM multisala."Proiezioni"
WHERE "ID_Proiezione" = id_proiezione;

SELECT "ID_Posti" INTO v_id_posto
FROM multisala."Posti"
WHERE "Numero" = numero_posto AND "Lettera" = lettera_posto;
```

```
INSERT INTO multisala."Biglietti"("Cod_Posto", "Cod_Proiezione") VALUES
(v_id_posto, v_sala_proiezione);


RETURN LASTVAL();

END

$BODY$;


ALTER FUNCTION multisala.get_new_biglietto(integer, integer, character)

   OWNER TO postgres;


CREATE OR REPLACE FUNCTION multisala.get_film()

   RETURNS TABLE(id_film integer, titolo text, durata integer, data_inizio date,
data_fine date, attivo boolean, best_price integer)

   LANGUAGE 'sql'

   COST 100

   VOLATILE PARALLEL UNSAFE

   ROWS 1000


AS $BODY$

SELECT "ID_Film", "Titolo", "Durata", "Data_Inizio", "Data_Fine", "Attivo",

CASE

   WHEN costi."prezzo" is null

      THEN 0

   ELSE 1

END AS best_price
```

```
FROM multisala."Film"

LEFT JOIN (SELECT "codice_film", "prezzo" FROM
multisala.spettacoli_remunerativi()) AS costi ON costi."codice_film" = "ID_Film"

WHERE "Attivo" = true

$BODY$;


ALTER FUNCTION multisala.get_film()

    OWNER TO postgres;


CREATE OR REPLACE FUNCTION multisala.get_biglietti_disponibili (id_proiezione
integer, giorno date)

    RETURNS TABLE(id_posti integer, numero integer, lettera character varying,
cod_sala integer, occupato integer)

    LANGUAGE 'sql'

    COST 100

    VOLATILE PARALLEL UNSAFE

    ROWS 1000


AS $BODY$

SELECT posti."ID_Posti", posti."Numero", posti."Lettera", posti."Cod_Sala",

CASE

    WHEN biglietti."ID_Biglietto" is null

        THEN 0

    ELSE 1

END AS Occupato
```

```
FROM multisala."Posti" AS posti

LEFT JOIN multisala."Proiezioni" AS proiezioni ON proiezioni."Cod_Sala" =
posti."Cod_Sala"

LEFT JOIN multisala."Biglietti" AS biglietti ON biglietti."Cod_Posto" = posti."ID_Posti"
AND biglietti."Cod_Proiezione" = id_proiezione

WHERE proiezioni."ID_Proiezione" = id_proiezione AND proiezioni."Day" = giorno

ORDER BY posti."Lettera";

$BODY$;


ALTER FUNCTION multisala.get_biglietti_disponibili(integer, date)

    OWNER TO postgres;
```