Atte Karppinen

ERD Cinema System Tables:

1. Screens
   * Same as cinema auditoriums/movie halls.
   * Capacity, how many customers can fit to movie hall
2. Viewings
   * Movie displays, for example includes data of:
   * Movie name, screen it will be played and starting time

Rest is self-explanatory

1. Movies
2. Sales
3. Customers

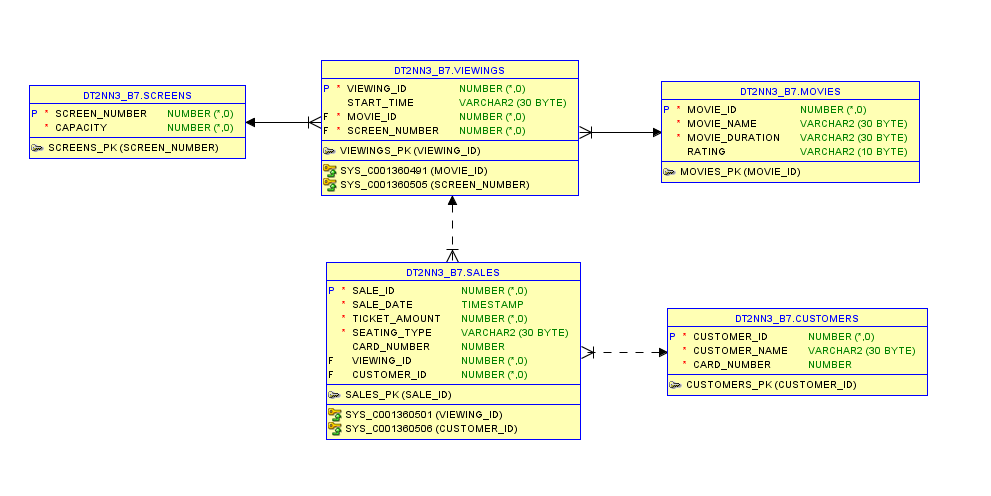
Relationships:

Same movie can be shown in multiple auditoriums.

Same screen can be used in different viewings.

A customer can have multiple sales.

A viewing can have multiple sales.



**CREATE, INSERT, GRANT**

CONNECT dt2nn3\_b7/DT2NN3\_B7;

DROP TABLE movies CASCADE CONSTRAINT;

DROP TABLE viewings CASCADE CONSTRAINT;

DROP TABLE customers CASCADE CONSTRAINT;

DROP TABLE sales CASCADE CONSTRAINT;

DROP TABLE screens CASCADE CONSTRAINT;

CREATE TABLE movies

(

movie\_id INT NOT NULL,

movie\_name VARCHAR(30) NOT NULL,

movie\_duration VARCHAR(30) NOT NULL,

rating VARCHAR(10),

PRIMARY KEY(movie\_id)

);

CREATE TABLE viewings

(

viewing\_id INT NOT NULL,

start\_time VARCHAR(30),

movie\_id INT NOT NULL,

screen\_number INT NOT NULL,

PRIMARY KEY(viewing\_id),

FOREIGN KEY(movie\_id) REFERENCES movies

);

CREATE TABLE customers

(

customer\_id INT NOT NULL,

customer\_name VARCHAR(30) NOT NULL,

card\_number NUMBER NOT NULL,

PRIMARY KEY(customer\_id)

);

CREATE TABLE sales

(

sale\_id INT NOT NULL,

sale\_date TIMESTAMP NOT NULL,

ticket\_amount INT NOT NULL,

seating\_type VARCHAR(30) NOT NULL,

card\_number NUMBER,

viewing\_id INT,

customer\_id INT,

PRIMARY KEY(sale\_id),

FOREIGN KEY(viewing\_id) REFERENCES viewings

);

CREATE TABLE screens

(

screen\_number INT NOT NULL,

capacity INT NOT NULL,

PRIMARY KEY(screen\_number)

);

ALTER TABLE viewings

ADD FOREIGN KEY(screen\_number) REFERENCES screens;

ALTER TABLE sales

ADD FOREIGN KEY(customer\_id) REFERENCES customers;

-- Populate Tables

-- Screens

INSERT INTO screens VALUES(1, 250);

INSERT INTO screens VALUES(2, 150);

INSERT INTO screens VALUES(3, 75);

INSERT INTO screens VALUES(4, 140);

INSERT INTO screens VALUES(5, 100);

INSERT INTO screens VALUES(6, 40);

INSERT INTO screens VALUES(7, 100);

INSERT INTO screens VALUES(8, 150);

INSERT INTO screens VALUES(9, 200);

INSERT INTO screens VALUES(10, 185);

-- Movies

INSERT INTO movies VALUES(1, 'Lord of the Two Towers', '2:00', '12PG');

INSERT INTO movies VALUES(2, 'The Importance of Ear', '1:50', 'U');

INSERT INTO movies VALUES(3, 'Einstein''s Big Adventure', '1:30', 'U');

INSERT INTO movies VALUES(4, 'The House of Horrors', '2:00', '15PG');

INSERT INTO movies VALUES(5, 'Beautiful Horizons', '2:00', '12PG');

INSERT INTO movies VALUES(6, 'Marion and Michelle', '2:30', '18only');

INSERT INTO movies VALUES(7, 'Shawshank damnation', '2:40', '18only');

-- Viewings

INSERT INTO viewings VALUES(1, '11:30', 1, 1);

INSERT INTO viewings VALUES(2, '13:00', 2, 2);

INSERT INTO viewings VALUES(3, '13:00', 1, 1);

INSERT INTO viewings VALUES(4, '14:00', 3, 3);

INSERT INTO viewings VALUES(5, '15:00', 4, 2);

INSERT INTO viewings VALUES(6, '15:15', 5, 1);

INSERT INTO viewings VALUES(7, '15:45', 2, 3);

INSERT INTO viewings VALUES(8, '17:15', 4, 2);

INSERT INTO viewings VALUES(9, '18:00', 6, 1);

INSERT INTO viewings VALUES(10, '18:15', 4, 3);

INSERT INTO viewings VALUES(11, '19:30', 2, 2);

INSERT INTO viewings VALUES(12, '20:15', 6, 1);

INSERT INTO viewings VALUES(13, '21:30', 1, 2);

INSERT INTO viewings VALUES(14, '22:00', 4, 3);

-- Customers

INSERT INTO customers VALUES(1, 'John Doe', 1234);

INSERT INTO customers VALUES(2, 'Jane Doe', 6421);

INSERT INTO customers VALUES(3, 'Peter Mulligan', 7776);

INSERT INTO customers VALUES(4, 'Cathelyn Parker', 9081);

INSERT INTO customers VALUES(5, 'Peter Hannigan', 5525);

-- Sales

INSERT INTO sales VALUES(1, to\_date('02-07-2014', 'DD-MM-YYYY'), 1, 'Premium', 1234, 9, 1);

INSERT INTO sales VALUES(2, to\_date('03-07-2014', 'DD-MM-YYYY'), 3, 'Standard', 6421, 4, 2);

INSERT INTO sales VALUES(3, to\_date('04-07-2014', 'DD-MM-YYYY'), 2, 'Standard', 7776, 7, 3);

INSERT INTO sales VALUES(4, to\_date('05-07-2014', 'DD-MM-YYYY'), 15, 'Premium', 9081, 12, 4);

INSERT INTO sales VALUES(5, to\_date('06-07-2014', 'DD-MM-YYYY'), 14, 'Premium', 9081, 14, 4);

INSERT INTO sales VALUES(6, to\_date('06-07-2014', 'DD-MM-YYYY'), 1, 'Standard', NULL, 7, NULL);

-- Grants

GRANT ALL ON movies TO JHIETA;

GRANT ALL ON viewings TO JHIETA;

GRANT ALL ON customers TO JHIETA;

GRANT ALL ON sales TO JHIETA;

GRANT ALL ON screens TO JHIETA;

GRANT ALL ON movies TO AKARPPINEN;

GRANT ALL ON viewings TO AKARPPINEN;

GRANT ALL ON customers TO AKARPPINEN;

GRANT ALL ON sales TO AKARPPINEN;

GRANT ALL ON screens TO AKARPPINEN;

**TRANSACTION**

CREATE OR REPLACE FUNCTION ADDMOVIE(

M\_name movies.movie\_name%TYPE,

M\_duration movies.movie\_duration%TYPE,

M\_rating movies.rating%TYPE)

RETURN VARCHAR2 IS

PRAGMA AUTONOMOUS\_TRANSACTION; -- To allow commiting

r\_name VARCHAR2(50); -- Name to be returned

largest\_id INTEGER;

next\_id INTEGER;

movie\_exists INTEGER;

BEGIN

-- Fetch most recent id and add one before inserting

SELECT MAX(movie\_id) INTO largest\_id FROM movies;

next\_id := largest\_id + 1;

-- If movie exists, do not add and inform user

SELECT COUNT(\*) INTO movie\_exists FROM movies

WHERE(M\_name = movie\_name);

IF movie\_exists > 0 THEN

RETURN 'Movie already exists in database';

ELSE

INSERT INTO movies VALUES

(next\_id, M\_name, M\_duration, M\_rating);

-- Get and return the movie name

SELECT movie\_name INTO r\_name FROM movies

WHERE (next\_id = movie\_id);

COMMIT;

RETURN r\_name;

END IF;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

RAISE;

END ADDMOVIE;

-- Example SELECT statement:

-- SELECT dt2nn3\_b7.ADDMOVIE('Added Movie Sequel', '3:10', '18only') FROM DUAL;

**QUERIES**

-- Selection

SELECT \* FROM dt2nn3\_b7.sales

WHERE customer\_id = 2;

-- Projection

SELECT ticket\_amount AS Tickets\_sold, seating\_type AS Type, customer\_id AS Customer

FROM dt2nn3\_b7.sales;

-- Aggregation with filters on aggregates. Returns count of movies with same age ratings

SELECT COUNT(movie\_name), rating

FROM dt2nn3\_b7.movies

GROUP BY rating

HAVING COUNT(rating) > 1;

-- Union. Unifies customers and their tickets they have bought

SELECT customer\_id AS ID,

seating\_type AS Name\_And\_Type,

ticket\_amount AS Card\_Tickets

FROM dt2nn3\_b7.sales

UNION

SELECT \* FROM dt2nn3\_b7.Customers

ORDER BY ID;

-- Minus. Returns movies that have no screen time at the moment

SELECT movie\_id FROM dt2nn3\_b7.movies

MINUS

SELECT movie\_id FROM dt2nn3\_b7.viewings;

-- Difference. Left joins returns all movies. Right join returns all movies in both tables. Intersect returns identical rows

SELECT m.movie\_name, v.start\_time

FROM dt2nn3\_b7.movies m

LEFT JOIN dt2nn3\_b7.viewings v ON v.movie\_id = m.movie\_id

INTERSECT

SELECT m.movie\_name, v.start\_time

FROM dt2nn3\_b7.movies m

RIGHT JOIN dt2nn3\_b7.viewings v ON v.movie\_id = m.movie\_id;

-- Inner Join. Returns customers that appear in both tables and name is Peter. See Outer Join:

SELECT dt2nn3\_b7.customers.customer\_name, dt2nn3\_b7.sales.ticket\_amount, dt2nn3\_b7.sales.seating\_type

FROM dt2nn3\_b7.customers

INNER JOIN dt2nn3\_b7.sales ON dt2nn3\_b7.sales.customer\_id = dt2nn3\_b7.customers.customer\_id

WHERE dt2nn3\_b7.customers.customer\_name LIKE '%Peter%';

-- Outer Join. Same query except return all customers named Peter, regardless if they appear on sales or not.

SELECT dt2nn3\_b7.customers.customer\_name, dt2nn3\_b7.sales.ticket\_amount, dt2nn3\_b7.sales.seating\_type

FROM dt2nn3\_b7.customers

LEFT JOIN dt2nn3\_b7.sales ON dt2nn3\_b7.sales.customer\_id = dt2nn3\_b7.customers.customer\_id

WHERE dt2nn3\_b7.customers.customer\_name LIKE '%Peter%';

-- Semi Join. Returns movies (each only once) that are in the viewings table

SELECT m.movie\_id, m.movie\_name

FROM dt2nn3\_b7.movies m

WHERE EXISTS(

SELECT 1

FROM dt2nn3\_b7.viewings v

WHERE v.movie\_id = m.movie\_id

);

-- Anti-join. Returns movies that are not in the viewings table

SELECT m.movie\_id, m.movie\_name

FROM dt2nn3\_b7.movies m

WHERE m.movie\_id NOT IN (

SELECT v.movie\_id

FROM dt2nn3\_b7.viewings v

);

-- Correlated sub-query. Returns all sale records of a customer named Cathelyn

SELECT \* FROM dt2nn3\_b7.sales s

WHERE s.customer\_id = (

SELECT c.customer\_id

FROM dt2nn3\_b7.customers c

WHERE c.customer\_name LIKE '%Cathelyn%'

);

Epilogue

I realised too late that a transaction was required, not a function. But basic concept should be the same.