# ERD, Redundancy, Normalization, Keys
*Advanced Database Management*

**Suryateja Chalapati**
MS in Business Analytics and Information Systems
University of South Florida
October 2, 2020

# Contents

# 1 Extended ERD

Comparing the Entity-Relationship Diagrams.

## 1.1 ERD - Before

The current movies database design is functional but its limited to what it can do. The Entity-Relation diagram of the movies database is as shown in Fig. 1.

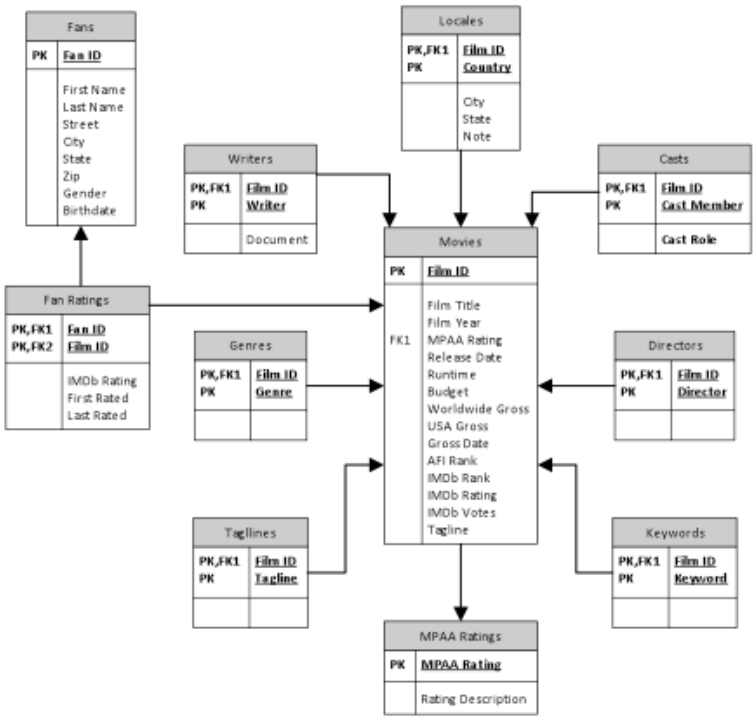


Figure 1: Movies ERD

## 1.2 ERD - After

The extended movies Entity-Relation diagram adds additional functionality to the database. Extended diagram is as shown in Fig. 2. ERD is color coded so as to clearly identify ”NEW ENTITIES”. Green color tables are the new entities that are added to the existing ERD, which is pink in color.

# 2 Data Redundancy

Data Redundancy occurs when a piece of data appears multiple times in a database. In the extended ERD design, attempt has been made to eliminate the redundant data by achieving normalization for any existing errors or repetitions in the database.

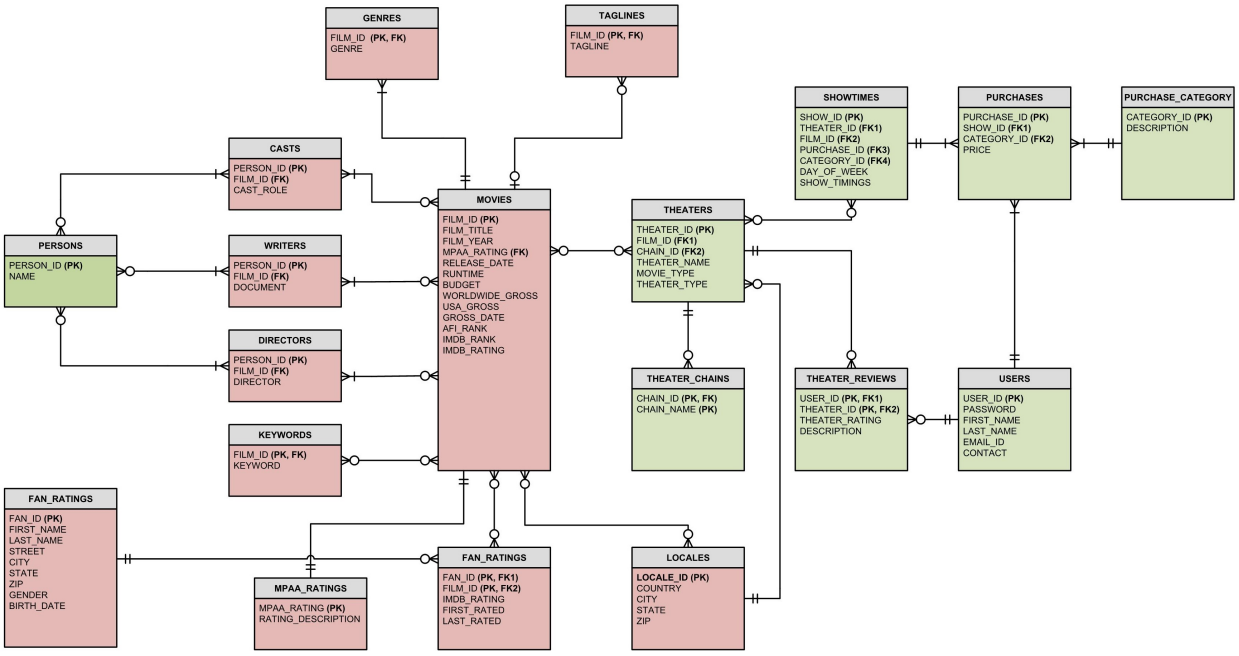


Figure 2: Movies Extended ERD

## 2.1 Movies Table

Movies table consists of ”Taglines” attribute which is removed in the extended ERD to avoid repetition since there exists a table called Taglines and is connected with Movies table.

## 2.2 Fans Table

Original Database didn’t have a relation defined between the tables FANS, FAN_RATINGS and MOVIES. It has been corrected in the extended ERD where we connected these tables establishing a relation.

Database optimization has been achieved by mapping format of ”State” in Users and Locales table to same, for example, ”FL”, before eliminating repetition of the column in Users. Street Address is added to Users table since it could be different for each user. This will reduce the system resources required to fulfill a query while returning the same results more efficiently.

## 2.3 Locales Table

”Note” column is removed from Locales table in the extended database during the process of getting rid of redundant data since the column does not contain any information and consists mainly of ”null” observations.

### 2.4 Persons Table

Persons table is created which consists of writers, directors and casts. Doing this removes repetitions since same person can pursue all three activities.

## 3 Normalization

We have normalized the database to the Third Normal Form (3NF). For a database to be in the Third Normal Form following conditions have to be met:

1. No repeating attributes or lists (1NF).
2. No attributes with partial dependencies on a composite key (2NF).
3. No attributes with dependencies on non-key attributes (3NF).

### 3.1 Tables

We have linked theater table to the movies table. We have created a bunch of new tables as well. Those are as follows.

- **List of New Tables:**
    1. USERS: This table stores registered user details.
    2. THEATER_REVIEWS: This table stores the user reviews of theaters.
    3. THEATERS: This table stores theater details and movies playing.
    4. THEATER_CHAINS: Consists of which theaters belong to which chain ID and chain name.
    5. SHOWTIMES: This table stores the details of showtimes from all theaters.
    6. PURCHASES: This table stores the ticket price.
    7. PURCHASE_CATEGORY: This table stores the ticket category (e.g., kids, adults, seniors, students, military etc.).
    8. PERSONS: This table stores activities by people as director, writer, cast.

# 4 Synthetic Keys

## 4.1 Statements on Keys

Here we list all the candidate keys for each synthetic key created. If no candidate key exists then we will state so explicitly.

- **List of Candidate Keys:**

  1. FAN_ID:
     Candidate Keys: Email_ID, PH_NUMBER.

  2. SHOW_ID:
     Candidate Keys: DAY_OF_WEEK, SHOW_TIMINGS.

  3. PURCHASE_ID
     Candidate Keys: PRICE.

  4. CATEGORY_ID
     Candidate Keys: DESCRIPTION.

  5. PERSON_ID
     Candidate Keys: NO CANDIDATE KEY

  6. LOCALE_ID
     Candidate Keys: COUNTRY, CITY, STATE, ZIP.

# 5 Assumptions

1. A movie can be screened multiple times a day at a particular theater.

2. Each user must register with a unique email ID and phone number combination.

3. A theater can screen movies simultaneously.

4. Multiple users can have the same first and last names.

5. Each user can leave a review on multiple theatres, provided each review can be traced back to the user based on the FAN_ID.

6. Each user can update the description and ratings as many times as they want.

7. Multiple theaters can belong to a single chain.

8. A theater chain can operate theaters in any locations.

9. A user can purchase multiple tickets for a movie, which can be done by having an account. These purchases can be tracked based on the PURCHASE_ID.

10. A person can do multiple activities in a movie which can be traced back based on PERSON_ID.

# 6 Interesting Queries

## 6.1 Query 1

Display the movie name, showtime in specific area, and the price of the ticket. Assume that the basic search for the movie application is the movie name and area which the user would like to go. Using INDEX to improve the performance of searching. Show the fastest showtime first

```
create index relmdb_movie_fcsearch_idx
on relmdb.movies (film_title, city);

SELECT
    m.film_title,
    t.theater_name,
    l.city,
    p.price,
    tc.chain_name,
    s.show_timings
FROM
    relmdb.movies m
    INNER JOIN relmdb.locales l
    ON m.film_id = l.film_id
    INNER JOIN relmdb.theaters t
    ON m.film_id = t.film_id
    INNER JOIN relmdb.theater_chain tc
    ON t.chain_id = tc.chain_id
```

```
    INNER JOIN relmdb.showtimes s
    ON t.theater_id = s.theaters_id
    INNER JOIN relmdb.purchases p
    ON s.show_id = p.show_id
WHERE
    (film_title LIKE 'Pulp Fiction') AND
    (city LIKE 'Tampa')
ORDER BY s.show_timings;
```

## 6.2 Query 2

Display the movies list available in one day in a certain theater. It is the basic search for the people who doesn't have a plan what movie he is going to watch. It will show movie name, date of the show, type of cinema, and the price. The movies will arrange by the name, and the fastest date.

```
create index relmdb_movie_tdsearch_idx
on relmdb.movies (theater_name, day_of_week);

SELECT
    m.film_title,
    s.show_timings
    p.price,
    tc.chain_name
FROM
    relmdb.movies m
    INNER JOIN relmdb.locales l
    ON m.film_id = l.film_id
    INNER JOIN relmdb.theaters t
    ON m.film_id = t.film_id
    INNER JOIN relmdb.theater_chains tc
    ON t.chain_id = tc.chain_id
    INNER JOIN relmdb.showtimes s
    ON t.theater_id = s.theater_id
    INNER JOIN relmdb.purchases p
    ON s.show_id = p.show_id
WHERE
    (t.thtr_name LIKE 'Tampa Theatre') AND
```

```
23      (day_of_week Like 'Wednesday')
24  ORDER BY m.flim_title, s.show_timings;
```