CS4416 Project 2015

MMmovies Database Design

4/23/2015

Antonela Berches 14125498

Anke Shi 14106051

Emma Murphy 14119587

Contents

1. Name of Students and Parts Worked on

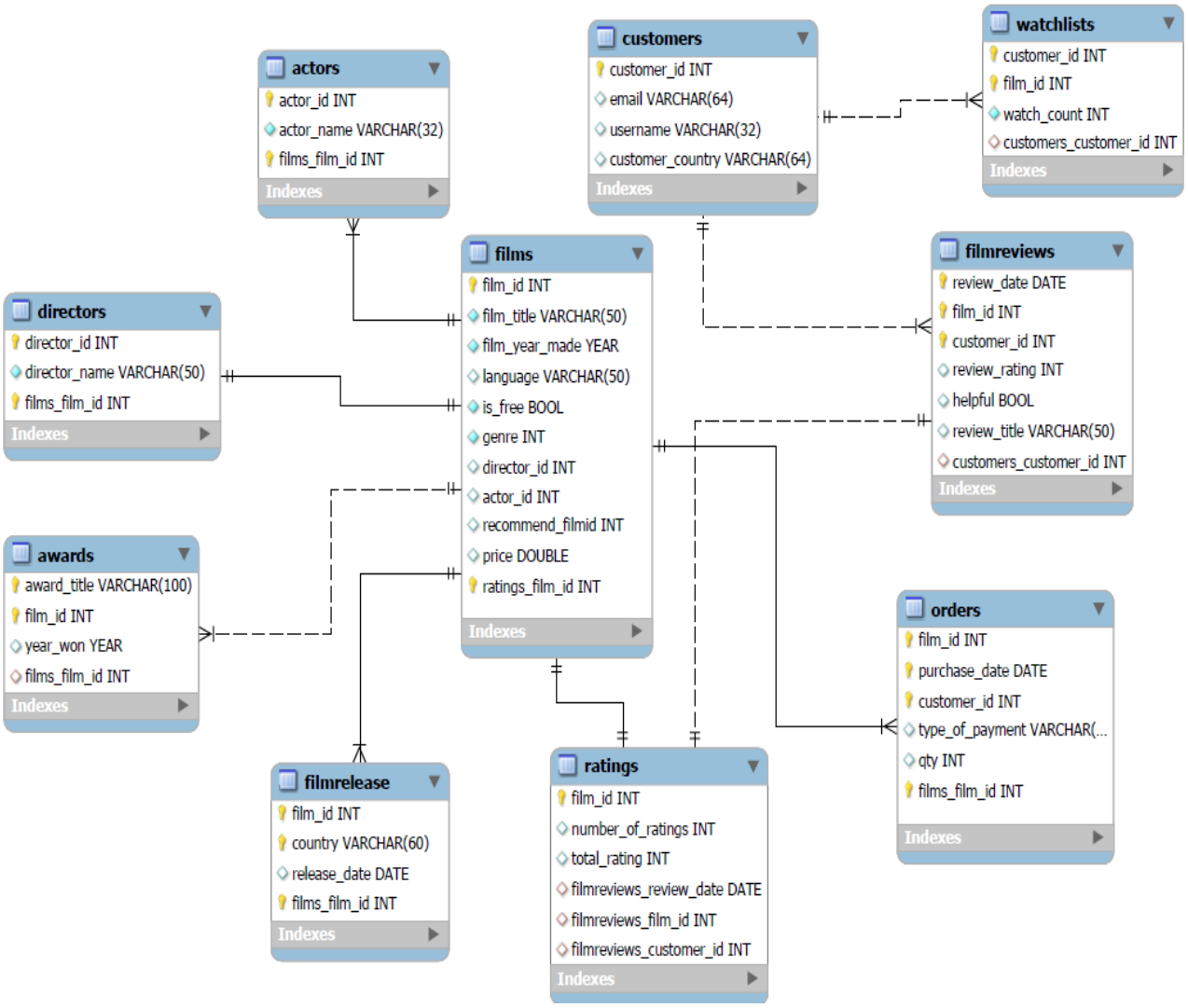
Antonela Berches 14125498

Anke Shi 14106051

Emma Murphy 14119587

All group members collaborated and came up with different ideas for stored procedures and triggers and queries – and equally all members contributed to the normalisation of tables and creation analysis of the queries. The justification of our queries is included.

2. ER Diagram


actors
actor_id INT
actor_name VARCHAR(32)
films_film_id INT
Indexes
customers
customer_id INT
email VARCHAR(64)
username VARCHAR(32)
customer_country VARCHAR(64)
Indexes
watchlists
customer_id INT
film_id INT
watch_count INT
customers_customer_id INT
Indexes
directors
director_id INT
director_name VARCHAR(50)
films_film_id INT
Indexes
films
film_id INT
film_title VARCHAR(50)
film_year_made YEAR
language VARCHAR(50)
is_free BOOL
genre INT
director_id INT
actor_id INT
recommend_filmid INT
price DOUBLE
ratings_film_id INT
Indexes
filmreviews
review_date DATE
film_id INT
customer_id INT
review_rating INT
helpful BOOL
review_title VARCHAR(50)
customers_customer_id INT
Indexes
awards
award_title VARCHAR(100)
film_id INT
year_won YEAR
films_film_id INT
Indexes
orders
film_id INT
purchase_date DATE
customer_id INT
type_of_payment VARCHAR(...
qty INT
films_film_id INT
Indexes
filmrelease
film_id INT
country VARCHAR(60)
release_date DATE
films_film_id INT
Indexes
ratings
film_id INT
number_of_ratings INT
total_rating INT
filmreviews_review_date DATE
filmreviews_film_id INT
filmreviews_customer_id INT
Indexes

3. What our database is about

Our MMmovies database is related to our web development project for which we are designing a film order and customer review website. We have selected 10 tables to use for the database project.

These are: Actors, Awards, Customers, Directors, Films, FilmRelease, FilmReviews, Orders, Ratings, WatchList.

MMmovies allows users to search for films by title, genre, actor and director. It can also let a customer search the database to see what film won which award in the year that they were won. Anyone can use the function to search for films.

Also anyone who uses the site can check to see when the film was released in which country or it will tell you a date in the future that the film will be released.

A customer must register with the site before they can purchase movies to watch. Therefore we have a "customer" table in the database to keep a log of who is registered with the website. The customers can also purchase movies to watch online through the website. There is a cost associated with each film that the customer stored in the orders table.

A customer can also review and rate a film we store this information in the FilmReviews and Rating Tables respectively.

We also store how many times a customer has watched a particular film – we store this in the "Watchlists" table – it uses the film_id and the customer_id and counts the number of times a user has watched the film. This is useful information for our website as we can include more popular film.

4. 3 Queries

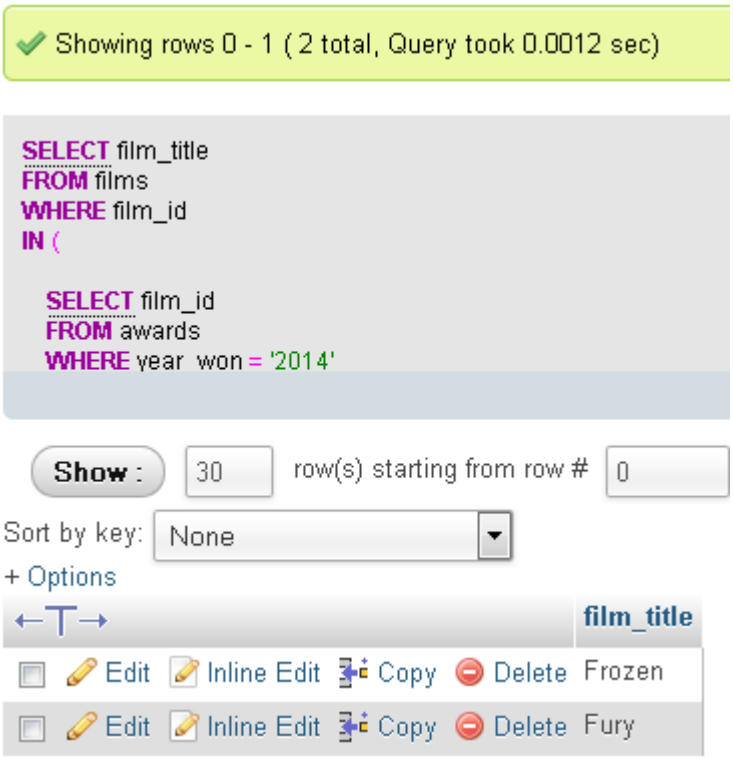
Query 1:

SELECT film_title

From films

WHERE film_id IN

(SELECT film_id from awards WHERE year_won = '2014');



Justification of Query 1

This is a very simple but meaningful sub query as it could be used a lot on the front end for example a customer may wish to search the database to find out which films won awards in a particular year. It could be useful for customer triva for example maybe they are doing some quiz and they just need to find the year a film won an award.

Query 2

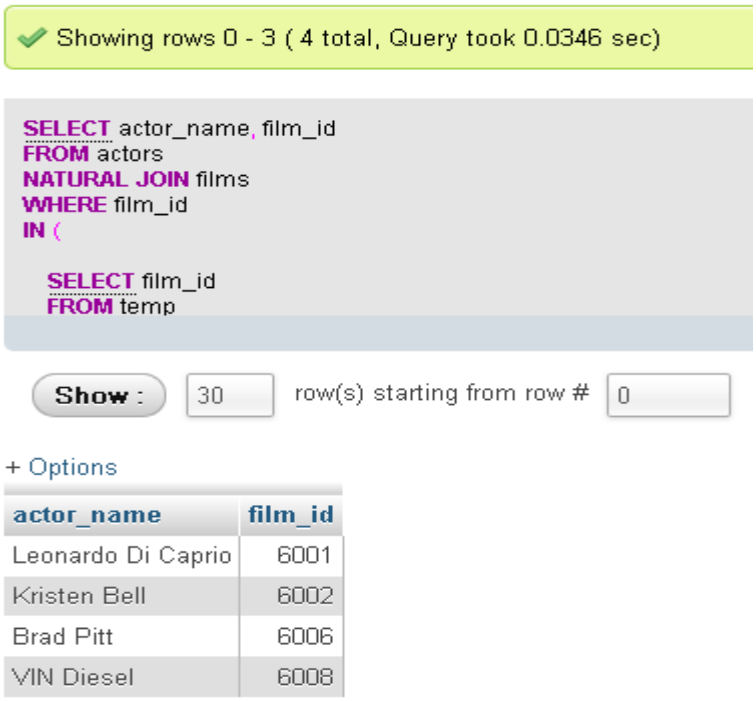
CREATE OR REPLACE VIEW temp AS

SELECT film_id, COUNT(*) AS cs

FROM awards

GROUP BY film_id

```
        ORDER BY film_id;


SELECT actor_name, film_id
FROM actors NATURAL JOIN films
WHERE film_id IN
        (SELECT film_id
         FROM temp
         WHERE cs=
                (SELECT MAX(cs)
                 FROM temp) );
```



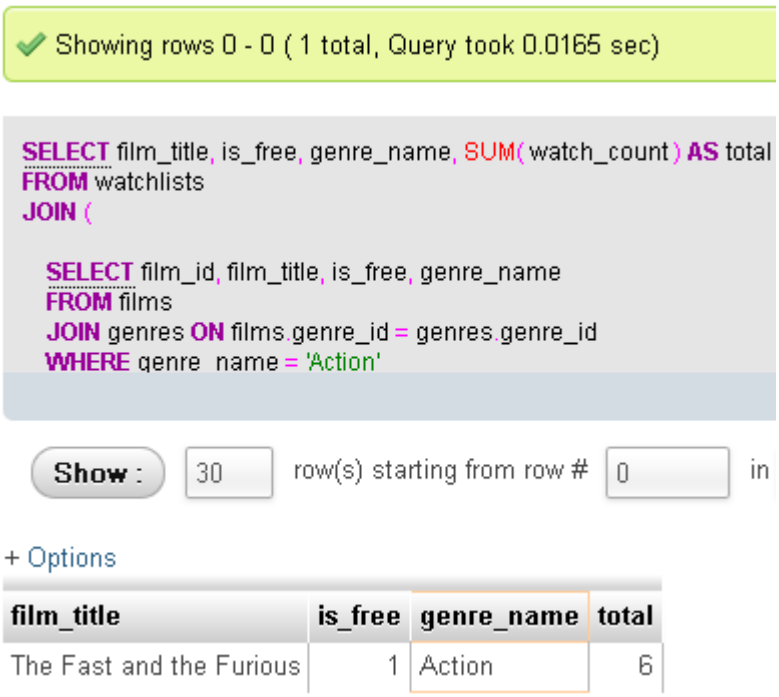
Justification of Query 2

Sub query 2 - This query returns the actors from the films which won the most awards.

This may also be useful for statistical purposes – To know the actor who was in the particular film that won the most awards. It could be useful because if an actor was in a film that won many awards – The website owners could decide to have more films available which feature that actor.

Query 3

```
SELECT film_title, is_free, genre_name,SUM(watch_count) AS total

FROM watchlists JOIN (

        SELECT film_id,film_title, is_free, genre_name

        FROM films JOIN genres ON films.genre_id= genres.genre_id

        WHERE genre_name ='Action') AS T ON watchlists.film_id=T.film_id

GROUP BY T.film_id

HAVING is_free=1;
```



film_title	is_free	genre_name	total
The Fast and the Furious	1	Action	6

Justification of Query 3 – returns the action films which are free and the sum of total rating for that film by each customer

It would be very useful as a database admin /website owner to know the results of this query and this query could be changed by genre – so the database admin could change it and can find out at any time the films that are free and the sum of the total ratings

We could produce multiple queries like this – using a different genre and we could identify the genres that are getting the best ratings by summing the watch_count , this could be useful as we could increase or decrease films by that genre according to popularity.

5. Analysis of the speed of queries

SPEED TEST	
Query 1	
	seconds
1st test:	0.001
2nd test:	0.0004
3rd test:	0.0004
4th test:	0.0004
Average speed	**0.00055**
Query 2	
	seconds
1st test:	0.0076
2nd test:	0.0052
3rd test:	0.0069
4th test:	0.0068
Average speed	**0.006625**
Query 3	
	seconds
1st test:	0.0016
2nd test:	0.0004
3rd test:	0.0005
4th test:	0.0004
Average speed	**0.000725**

6. Triggers

Trigger 1

The insertDirectorandActor Trigger – means that whenever a new film is inserted into the films table that – the trigger will check two tables – actors and directors and if they already exist we will not input the director or actor twice –(we checked to see could we additionally print a message to the screen to tell the administrator that the director name and actor name needs to be added). We are aware that the director_name is inserted as Null for a new director_id inserted into the director table

```
Delimiter //

Create Trigger insertDirectorandActor AFTER INSERT on films

For Each Row Begin


IF (NEW.director_id NOT IN(

SELECT director_id

FROM directors))

THEN

INSERT INTO directors(director_id,director_name)

 VALUES(NEW.director_id,NULL);

END IF;


IF (NEW.actor_id NOT IN(

SELECT actor_id

FROM actors))

THEN

INSERT INTO actors(actor_id,actor_name)

 VALUES(NEW.actor_id,NULL);

END IF;

END;//
```

Trigger 2

```
DELIMITER //

CREATE TRIGGER changePrice

 BEFORE UPDATE ON films

 FOR EACH ROW

BEGIN

 IF NEW.is_free = '0';

THEN

 SET NEW.price = 0;

 ELSE

SET NEW.price= 10;

 END IF;

 END;//
```

This Trigger changePrice – if a record in the film table is updated and the attribute of that film “is_free” is set to 0 in the table – then the new price will be set to 0.00. else the new price of that film will be set to 10.00.

It’s important that if a film is updated and the film is set to free – (is_free value 0) that the price will be updated to 0.00 accordingly.

7. Stored Procedures

Stored Procedure 1

The UpdateRecommend Stored Procedure – compares a film against all of the films of the same genre based on their average rating , if no film in that genre is rated higher than the film then the recommend_filmid in the films table will be updated with that filmid

if the film is not rated higher than the highest rated film in that genre category the recommened_filmid will be updated with the highest rated film_id in for that genre.

```
delimiter //
CREATE PROCEDURE UpdateRecommend(
        IN fid INTEGER(10)
        )
MODIFIES SQL DATA
BEGIN

        DECLARE oid INTEGER;
        DECLARE g INTEGER;
        DECLARE be INTEGER;

        SELECT recommend_filmid INTO oid
        FROM films
        WHERE film_id=fid;

        SELECT  genre_id INTO g
        FROM films
        WHERE film_id=fid;

        CREATE OR REPLACE VIEW rating_genre As
        SELECT A.film_id, A.genre_id,B.total_rating/B.number_of_ratings 'average_rating'
        FROM films A, ratings B
        WHERE A.film_id =B.film_id;

        SELECT film_id INTO be
        FROM rating_genre
        WHERE average_rating=
                        (SELECT MAX(average_rating)
                         FROM rating_genre
                         WHERE genre_id = g);

  START TRANSACTION;
```

```
        IF(be is NULL) THEN
                UPDATE films
                SET recommend_filmid=0
                WHERE film_id=fid;
        ELSE
                UPDATE films
                SET recommend_filmid=be
                WHERE film_id=fid;
        END IF;

    COMMIT;

        DROP VIEW rating_genre;
END;//
```

Stored Procedure 2

The DeleteCustomer Stored Procedure if you delete a customer , it deletes all associated data belong to that customer – in the related tables such as customers, watchlists, orders, filmreviews and ratings.

```
delimiter //
CREATE PROCEDURE DeleteCustomer(
        IN cid INTEGER(10)
        )
MODIFIES SQL DATA
BEGIN

        START TRANSACTION;

        IF(cid IN (SELECT customer_id FROM customers)) THEN
          DELETE FROM Customers WHERE customer_id = cid;
        END IF;

        IF(cid IN (SELECT customer_id FROM watchlists)) THEN
      DELETE FROM watchlists WHERE customer_id = cid;
        END IF;

        IF(cid IN(SELECT customer_id FROM orders)) THEN
      DELETE FROM orders WHERE customer_id = cid;
        END IF;

        IF(cid IN(SELECT customer_id FROM filmReviews)) THEN
      DELETE FROM filmReviews WHERE customer_id = cid;
        END IF;
        UPDATE Ratings
        SET number_of_ratings=
                (SELECT COUNT(*) FROM filmreviews WHERE ratings.film_id=filmreviews.film_id);

        UPDATE Ratings
        SET total_rating=
                (SELECT SUM(review_rating) FROM filmreviews WHERE
ratings.film_id=filmreviews.film_id);

        COMMIT;

END;//
```

Justification of Stored Procedure 1

It is necessary to create a procedure called “UpdateRecommend” so that if a customer does not know what to watch and they pick a film – the recommend_filmid will be updated with a film_id that has a higher rating if the film they choose is lower than the highest rated film in that genre. When the procedure is called the recommend_filmid will be updated if the film has a new higher rating.

Scenario: So on the front end of the website the customer will be informed and know which film is rated higher by the genre of the film they selected.

Justification of Stored Procedure 2

It is necessary to create procedure called “DeleteCustomer” Because – if we delete a customer from our database we want to delete all other instances of the customer on associated tables. Because we do not want any redundant data left in the database or incomplete or inconsistent data.

8. Examples of Tables with Primary keys Defined includes list of Functional Dependencies

We say that X → Y (x functionally determines y in r) if it is true that whenever 2 tuples of R have the same values for all the attributes in X they also have the same values for all attributes in y.

- **Reflexivity**: If *Y* is a subset of *X*, then $X \rightarrow Y$
- **Augmentation**: If $X \rightarrow Y$, then $XZ \rightarrow YZ$
- **Transitivity**: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$
- **Pseudo transitivity:** If $X \rightarrow Y$ and $WY \rightarrow Z$, then $WX \rightarrow Z$

Reflexivity: if B (is a Subset of) A then A → B

Transitivity: if A → B and B → C then A → C

Augmentation: if A → B then AC → B

Pseudo transitivity:
if A → B and BC → D then AC → D

Table 1 Actors

Table Structure:

#	Column	Type	Collation	Attributes	Null	Default
1	**actor_id**	int(10)			No	*None*
2	**actor_name**	varchar(32)	latin1_swedish_ci		Yes	*NULL*

Primary Key : Actor_id

actor_id	actor_name
3001	Leonardo Di Caprio
3002	Kristen Bell
3003	Katya Paskaleva
3004	Brandon Lee
3005	Brad Pitt
3006	VIN Diesel
3007	Dev Patel
3008	Ewan McGregor

Table 1: Actors : Actor_id (A), Actor_name(B)

Functional Dependencies : Actor_Name

Actor_id → Actor_Name

A → B

Reflexivity

- For the purposes of our database we only have two attribute in the actors table
- The actor_name can be functionally determined by the actor_id

Table 2 Awards

Table Structure:

Column	Type	Null	Default	Comments
award_title	varchar(100)	No		
film_id	int(10)	No		
year won	year(4)	Yes	*NULL*	

Composite Primary Key : Award Title and Film_ID

Table Data Sample:

award_title	film_id	year_won
Academy Award for Best Animated Feature Film	6002	2014
BAFTA Award for Best Animated Film	6002	2014
BAFTA Award for Best Screenplay, Adapted	6010	1996
Berlin International Film Festival	6003	2000
Empire Award	6006	2000
Empire Award	6010	1997
Golden Globe Award for Best Motion Picture – Drama	6001	1998
Hollywood Film Awards	6007	2014
Jupiter Award	6006	1999
MTV Movie Award	6001	1998
MTV Movie Awards	6005	1995
MTV Movie Awards	6008	2002
World Stunt Awards	6008	2002

Table 2 : Awards : Award Title(A), Film_ID(B), Year_won(C)

Functional Dependencies :

Award Title & Film_ID → Year_Won

AB→ C

- The assumptions for this table means that the film must win an award before we can determine any other attributes.
- The year_won cannot! Identify the award_title as a film may win several awards in one year.
- Therefore the Award_Title and Film_id serve as a composite primary Key and together they can functionally determine the year that the award was won.

Table 3 Customers

Table Structure:

#	Column	Type	Collation	Attributes	Null	Default
1	**customer_id**	int(10)			No	*None*
2	**email**	varchar(64)	latin1_swedish_ci		Yes	*NULL*
3	**username**	varchar(32)	latin1_swedish_ci		Yes	*NULL*
4	**customer_country**	varchar(64)	latin1_swedish_ci		Yes	*NULL*

Composite Primary Key : customer_id

customer_id	email	username	customer_country
1001	John@Dell.com	Johnny97	Ireland
1002	Sarah@ul.ie	Sarah86	Ireland
1003	Mike@yahoouk	Mikeyboy	UK
1004	SecretJoe@hotmail.com	SecretJoe	France
1005	PAfoskin@ul.ie	PaddyF2K15	USA
1006	jENNAb@DELL.COM	jennaB34	Germany
1007	SHIANKE@ul.ie	SHIANKE1990	China
1008	Anotonella@ul.ie	AntoOle	Italy

Table 3: Customers : Customer_id(A),email(B),username(C),Customer_Country(D)

Functional Dependencies :

A →BCD

- Functional Dependencies are : email,username,customercountry.
- From this table the customer_id, the email and the user name can functionally determine all attributes in the table
- Customer_id is the primary key of the table
- Note : email address /username would not be left Null on the registration page – this is small error on our create table statement it should not be null.

Table 4 Directors

Table Structure:

#	Column	Type	Collation	Attributes	Null	Default
1	director_id	int(10)			No	*None*
2	director_name	varchar(32)	latin1_swedish_ci		Yes	*NULL*

Primary Key : director_id

Functional Dependencies : director_name

director_id	director_name
2001	James Cameron
2002	Chris Buck
2003	Jeffrey Hornaday
2004	Metodi Andonov
2005	Alex Proyas
2006	David Fincher
2007	David Ayer
2008	Rob Cohen
2010	Danny Boyle

Table 3: directors: director_id (A), director_name(B)

Functional Dependencies : director_Name

director_id → director_Name

A → B

Reflexivity

- For the purposes of our database we only have two attribute in the directors table
- The director_name can be functionally determined by the actor_id
- We could potenitally have two directors with the same name , therefore it is the director_id(unique primary key) which will functionally determine the directors name

Release 5 FilmRelases

Table Structure:

#	Column	Type	Collation	Attributes	Null	Default
1	**film_id**	int(10)			No	*None*
2	**country**	varchar(64)	latin1_swedish_ci		No	*None*
3	**release_date**	date			Yes	*NULL*

Composite Primary Key : film_id, country

Functional Dependencies : release_date

film_id	country	release_date
6001	United States	1997-12-19
6002	United States	2013-11-27
6003	United Kingdom	2000-02-11
6004	Bulgaria	1972-10-05
6005	United States	1994-05-13
6006	Germany	1999-11-11
6007	United States	2014-10-17
6008	United States	2001-06-22
6009	United Kingdom	2009-01-09
6010	United Kingdom	1996-07-19

Table 5: FilmRelease: Film_id (A), country(B),release_date(C)

Functional Dependencies : release_date

Film_id,country → release_date

AB → C

Reflexivity

- In our Film_Release date it is the film_id and the country together which will functionally determine the release date , as the same film can be released in several different countries , it will depend on the country what date that the film will be released .
- The film_id and the country separetly cannot tell you anything for example if you have the film_id alone it cannot determine what the release_date will be without the country in which it is released in

Table 6 FilmReviews

Table Structure:

#	Column	Type	Collation	Attributes	Null	Default
1	**review_date**	date			No	0000-00-00
2	**film_id**	int(10)			No	*None*
3	**customer_id**	int(10)			No	0
4	**review_rating**	int(4)			Yes	0
5	**helpful**	tinyint(1)			Yes	0
6	**review_title**	varchar(50)	latin1_swedish_ci		Yes	*NULL*

Composite Primary Key : film_id, customer_id,review_date

review_date	film_id	customer_id	review_rating	helpful	review_title
1995-06-15	6005	1005	2	0	Just so-so.
1997-12-20	6001	1008	5	1	love it
1998-05-15	6001	1008	4	1	enjoy it
1999-03-15	6001	1001	5	1	I love it so so so much
2000-01-02	6001	1002	1	0	It is a disaster
2003-02-02	6008	1008	3	0	epic film
2004-11-11	6008	1001	1	1	terrible one
2010-01-02	6009	1008	5	1	My favorite, a ledgendary
2014-05-15	6002	1008	3	1	It is a nice one
2014-12-16	6007	1006	1	0	waste my time!

Table 6: FilmReviews:
Film_id(A),review_date(B),customer_id(C),review_rating(D),helpful(E),Review_title(F)

Keys : Film_id,Review_date,customer_id → review_rating,helpful,review_title

ABC →DEF

- In our Film_Reviews Table , there will be no Film Review record in the table unless a customer writes a review of a particular film on a particular date – therefore the three attributes review_date,customer_id and film_id become a composite primarykey
- You cannot determine anyother attributes without the superkey
- Also a customer may rate several films on the same date – so it is the film id which will identify the unique record – A customer cannot write two reviews of the same film, they may edit or delete their review but they cannot write two reviews of the same film.
- ABC is the minimal superkey for this table

Table 7 Films

Table Structure:

#	Column	Type	Collation	Attributes	Null	Default	E
1	**film_id**	int(10)			No	*None*	
2	**film_title**	varchar(50)	latin1_swedish_ci		No	*None*	
3	**film_year_made**	year(4)			No	*None*	
4	**language**	varchar(20)	latin1_swedish_ci		Yes	*NULL*	
5	**is_free**	tinyint(1)			No	*None*	
6	**genre_id**	int(10)			No	*None*	
7	**director_id**	int(10)			Yes	*NULL*	
8	**actor_id**	int(10)			Yes	*NULL*	
9	**recommend_filmid**	int(10)			Yes	*NULL*	
10	**price**	double			Yes	*NULL*	

film_id	film_title	film_year_made	language	is_free	genre_id	director_id	actor_id	recommend_filmid	price
6001	Titanic	1997	english	0	4001	2001	3001	*NULL*	0
6002	Frozen	2013	english	0	4002	2002	3002	6002	0
6003	The_Beach	1999	english	1	4003	2003	3001	*NULL*	10
6004	The Goat Horn	1972	bulgarian	1	4004	2004	3003	6004	8
6005	The Crow	1994	english	1	4005	2005	3004	*NULL*	10
6006	Fight Club	1999	english	0	4004	2006	3005	6004	0
6007	Fury	2013	german	1	4006	2007	3005	*NULL*	5
6008	The Fast and the Furious	2001	english	1	4007	2007	3006	*NULL*	6

Table 7: Films: Film_id(A), film_title(B), film_year_made(C),language(D), is _free(E), genre_id(F), director_id(G), actor_id(H), recommend_filmid(I), Price(J)

Film_id →
Film_title,film_year_made,language,is_free,genre_id,director,actor_id,recommend_filmid, price

A →BCDEFGHIJ

- In our Films Table it is the Film_id which determines the rest of the attributes in the films table.
- The film_id is the minimal superkey.
- you could potentially have a film with the same film_title , so the film_title cannot determine anything else about a record in the film table!
- Director_id and actor_id are foreign keys, they also cannot determine anything in this table because a director could direct several films so on its own as an attribute it cannot functionally determine a record/tuple in the film table because it could be determine several records.

Table 8 Orders

Table Structure:

#	Column	Type	Collation	Attributes	Null	Default
1	**film_id**	int(10)			No	0
2	**purchase_date**	date			No	0000-00-00
3	**customer_id**	int(10)			No	0
4	**type_of_payment**	varchar(30)	latin1_swedish_ci		Yes	*NULL*
5	**qty**	int(20)			Yes	*NULL*

Composite Primary Key : film_id,purchase_date,customer_id

film_id	purchase_date	customer_id	type_of_payment	qty
6001	1990-03-10	1001	cash	3
6001	1997-11-15	1008	visa	5
6001	1999-12-02	1002	mastercard	2
6002	2014-03-01	1002	cash	2
6002	2014-04-10	1008	visa	6
6005	1995-06-14	1005	visa	3
6005	2000-03-04	1003	voucher	1
6005	2000-03-04	1006	cash	1
6007	2014-12-16	1006	mastercard	10
6007	2014-12-20	1007	cash	5
6008	2003-01-30	1001	visa	4
6008	2003-01-30	1008	voucher	1
6008	2004-03-30	1004	cash	2
6009	2009-01-30	1008	visa	2
6009	2010-01-01	1001	cash	2
6009	2011-12-07	1006	cash	4

Table 8: Orders: Film_id(A),purchase_date(B),customer_id (C), type of payment(D)qty(E)

Film_id,purcase_date,customer_id→ type of payment,qty

ABC →DE

- In our Orders table we have a composite primary key – we cannot have an order without a customer purchasing a film on a particular date.
- So we need to know the film_id , the customer_id and the purchase_date in order to have an order record in the orders table

Table 9 Ratings

Table Structure:

#	Column	Type	Collation	Attributes	Null	Default
1	**film_id**	int(10)			No	*None*
2	**number_of_ratings**	int(10)			Yes	0
3	**total_rating**	int(10)			Yes	0

Primary Key : film_id

film_id	number_of_ratings	total_rating
6001	4	15
6002	1	3
6005	1	2
6007	1	1
6008	2	4
6009	1	5

Table 9: Ratings: Film_id(A),number_of_ratings(B),total_rating(C)

Film_id → Number of ratings,total_ratings

A→BC

In the ratings table the film_id determines the number of ratings which the film has and the total ratings column

Table 10 WatchLists

Table Structure:

Column	Type	Collation	Attributes	Null	Default	Extra
customer_id	int(10)			No	*None*	
film_id	int(10)			No	*None*	
watch_count	int(100)			No	*None*	

Composite Primary Key : film_id,customer_id

customer_id	film_id	watch_count
1001	6001	1
1001	6008	2
1002	6001	2
1002	6010	4
1005	6005	2
1006	6007	5
1008	6001	3
1008	6002	1
1008	6008	4
1008	6009	1

Table 10: WatchList: Film_id(A),customer_id(B),watch_count(C),

AB→C

- In our WatchList table we have a composite primary key – customer_id and film_id
- The film_id on its own cannot determine how many times a customer watched a particular film and neither can customer_id
- A customer can watch several different films – many different times so the customer id and the film_id will functionally determine how many times that customer watched that film.

9. Proof that the Tables are in 3NF

Question: Is a relation in 3NF or BCNF ?

Answer: A relation is in BCNF if all expressions on the LHS are Superkeys

A relation is in 3NF if it is in BCNF or RHS attributes are prime (a prime attribute means it's a part of anykey)

Table 1: Actors

Actors: Actor_id (A), Actor_name(B)

A → B

{A}+={AB} (superkey and key i.e. A is minimal)

{B}+={B}

The LHS are superkeys. Actor_name is functionally dependent on the key actor_id , this means that this table is in 3NF

Table 2 : Awards

Awards : award Title(A), film_id(B), film_year_won(C)

award Title & film_id→ film_year_won

AB→ C

{A}+={A}

{B}+={B}

{C}+={C}

{AB}+={C} (superkey and key i.e. AB is minimal)

Table 3: Customers

Customers : customer_id(A),email(B),username(C),customer_country(D)

Customer_id→ email , username,=customer_country

Functional Dependencies :

A →BCD

Closure of attributes

{A}+={ABCD} key and minimal superkey

Customer_id is the primary key and it can functionally determine the rest of the attributes in a record of the customers table

Table 4: Directors

directors: director_id (A), director_name(B)

Functional Dependencies : director_Name

director_id → director_Name

A → B

{A}+={AB} (superkey and key i.e. A is minimal)

{B}+={B}

The LHS A is a superkey. director_name is functionally dependent on the key actor_id , this means that this table is in 3NF.

Table 5: FilmRelease:

FilmRelease: Film_id (A), country(B), release_date(C)

Functional Dependencies : release_date

Film_id,country → release_date

AB → C

{A}+={A}

{B}+={B}

{C}+={C}

{AB}+={ABC} Key and minimal superkey

Superkey : A set of attributes k is a superkey for relation R if K functionally determines all attributes of R

3NF: relation R is in 3NF if for each non-trivial FD X→A that holds in R either X is a superkey of R or A is prime

The LHS AB is a minmal superkey. Release date is functionally dependent on the composite key film_id and country , this means that this table is in 3NF.

Table 6: FilmReviews:

Filmreviews:
film_id(A),review_date(B),customer_id(C),review_rating(D),helpful(E),Review_title(F)

Keys : Film_id,Review_date,customer_id → review_rating,helpful,review_title

ABC →DEF

{A}+={A}

{B}+={B}

{C}+={C}

{AB}+={AB}

{CB}+={CB}

{AC}+={AC}

{ABC}+={ABCDEF} key and minimal superkey

Any part (prime attribute) of the composite key cannot functionally determine any other attribute on its own

For example review_date alone cannot tell you the title as there may be several reviews on that date.

You need the film_id the review_date and the customer_id to identify a tuple/row in the filmReviews table –

The assumption is that 1 customer cannot review the same film twice , they can delete their own review or edit it. But they cannot review the same film on the same date twice for example

The filmReviews Table is in 3NF

Table 7: Films:

Films: Film_id(A), film_title(B), film_year_made(C),language(D), is
_free(E), genre_id(F), director_id(G), actor_id(H), recommend_filmid(I), Price(J)

Film_id → Film_title,film_year_made,language,
is_free,genre_id,director,actor_id,recommend_filmid,price

A →BCDEFGHIJ

{A}+={ABCDEFGHIJ}

Key = A /film_id all our other superkeys must include A and so no other superkeys can be minimal.

Table 8: Orders:

Orders: Film_id (A),purchase_date(B),customer_id (C), type of payment(D)qty(E)

Film_id,purchase_date,customer_id→ type of payment,qty

ABC →DE

{ABC}+={ABCDE} superkey and minimal key

A set of attributes k is a superkey for relation r if k functionally determines all attributes of r.

The key ABC functionally determines the rest of the attributes in our relation so this table is in 3NF.

Table 9: Ratings:

Ratings: Film_id(A),number_of_ratings(B),total_rating(C)

Film_id → Number of ratings,total_ratings

A→BC

{A}+={ABC}

A is the minimal superkey and determines all the other attributes in our relation the ratings table is in 3NF

Table 10: WatchLists:

Watchlists: Film_id(A),customer_id(B),watch_count(C),

AB→C

{AB}+={ABC}

AB is the minimal superkey and determines all the other attributes in our relation the watchlists table is in 3NF