Homework 2, William Berg and Gustav Henningsson, group 104

1. New requirements

The Product Owners of the MSS initially wanted the system to only have movies. However, they now require the system to also include functionality for TV-series. In particular, they require your database to be redesigned in such a way that the following use-cases are satisfied:

- Users should be able to watch a TV-series.
- TV-series have seasons and episodes.
- TV-series can have different directors for different episodes.
- Users should be able to rate the the TV-series as a whole, not necessarily per episode.

1.1. Make changes such that these above requirements are included.

The first logical step seems to be to create a new relation for TV-series, let's call it TVseries. Since the TV-series should have seasons and episodes, it seems reasonable to create an Episodes relation with a season attribute (seeing as each episode should have its own information, such as a different directors for different episodes). Below are the schemas for these new relations, with relevant attributes:

1. TVseries(TVseriesID:integer, name:string, year:integer, country:string, genre:string, numberOfSeasons:integer, rating:numeric)
2. Episodes(episodeID:integer, TVseries:integer, name:string, director:string, length:numeric, season:integer, rating:numeric)

1.2. Modify your database design so that it may include any other information you deem appropriate. (Note that this question is more free-form than the questions above, and you need to make some decisions about multiplicities of relationships, appropriate types, and even what information needs to be represented.) Note: a minimum of one necessary modification.

The Customers relation currently contains an integer attribute upID stating how many user profiles this Customer has. This is a bit useless, especially since we have created a UserProfiles relation already. We will therefore modify the database to include a string of comma separated upIDs so that we can know what user profiles correspond to what customer (the design flaw in this implementation will be addressed in question 2.1).

We will also add the episodeID attribute as another foreign key to the Watchlist relation so that it can also track TV-series. Other than that we will also add a Directors relation:
Directors(directorID:integer, DoB:date, gender:char, PoB:string)
The design flaw with the DoB implementation will be addressed in question 2.1.
NOTE: The director attribute in the Episodes relation will have an integer domain (and not string) after this addition.

We also added functionality for movie franchises, by adding a MediaFranchise relation (containing the original film) and three associative relations for remakes, prequels and sequels respectively.

A final change that we deemed appropriate is the addition of the customer (custID) foreign key as an attribute in the SubscriptionPlan relation. This way we can know what customer has what subscription plan.

1.3. Identify new keys and referential integrity constraints for these new changes.

The new TVseries relation has TVseriesID as its primary key, the Episode relation's primary key is episodeID (and contains the TVseriesID foreign key) and the Directors relation has directorID as its primary key. The new associative relations form a composite primary key from the two foreign keys that they contain (mediaID + actorID and TVseriesID + actorID). The creation of the Directors relation has also made it so that the Episodes relation and the Media relation contain the directorID foreign key. custID is now also a foreign key in the SubscriptionPlan relation. The additional support for movie franchises means that the three associative relations for sequels, prequels and remakes all contain a composite primary key consisting of mediaID and franchiseID (the primary key for the MediaFranchise relation).

2. Normalization

In order to get a good overview of your database design, decompose your relations as necessary into Boyce-Codd normal form (BCNF) and apply the following:

2.1. Describe every step you did towards this goal. Justifying your design choices, you should give some (informal) arguments for why some attributes of your solution are not functionally determined by other attributes. For example, what assumptions have you made of these non-functionally determined attributes?

For homework 1 we had a comma separated string listing all of the actors for each movie. This is not normalized (it does not even pass the requirements for first normal form) and therefore not desirable. To solve this problem we will start off by creating an actors relation; the schema can be seen below:
Actors(actorID:integer, name:string, DoB:date, gender:char, PoB:string)
Now we can create two associative relations (one for media and one for TV-series), let's call them MediaActors and TVseriesActors. These will contain two attributes each. The primary key from the Media/TVseries relation (mediaID/TVseriesID) and the primary key from the Actors relation. Together these two attributes will form the primary key for the two associative relations. The many-to-many (one actor can star in many movies/tv-series and one movie/tv-series can star many actors) relationship schemas can be seen below:
MediaActors(movie:integer, actor:integer)
TVseriesActors(TVseries:integer, actor:integer)
Because of this we can now remove the actors string attribute from the Media relation.

We will make a similar change for the Customers relation that currently contains a comma separated string listing the different user profiles for a specific customer. We will create a new relation called CustomerProfiles that contains two attributes; userProfile (upID) and customer (custID). It is now represented with this one-to-many (one customer can have many user profiles but a user profile only belongs to one customer) relationship schema:
CustomerProfiles(userProfile:integer, customer:integer)

Our relations are already almost in BCNF after these changes. Having an ID attribute for each relation, that exists solely to serve as a primary key is a good way to ensure that a relation is in BCNF (we have a unique ID for each relation for this very reason). This is because it is an easy way to make sure that the left side of every non-trivial functional dependency contains a key (that the left side is a superkey). One relation that needs to be decomposed however is the SubscriptionPlan relation. This is because it contains transitive dependencies; paymentAmount depends on subType and expirationDate depends on startDate. These dependencies need to be resolved in order for the relation to be in third normal form, which in turn is a requirement for it to be in BCNF. The other attributes in SubscriptionPlan are fine however, since the creditCardInfo attribute for instance is not functionally determined by any attribute that is not the primary key. We decomposed the SubscriptionPlan relation into the following relations

to solve the problem:

Before:

SubscriptionPlan(subID:integer, subType:string, startDate:date, expirationDate:date, paymentAmount:integer, creditCardInfo:string, customer:integer)

After:

SubscriptionDuration(startDate:date, expirationDate:date)
SubscriptionPayment(subType:string, paymentAmount:integer)
SubscriptionPlan(subId:integer, startDate:date, subType:string, creditCardInfo:string, customer:integer)

We are assuming here that the subType does not affect the duration of the subscription, which we think is a fair assumption. There is a final adjustment to be made to reach BCNF. Since the PoB attribute in the Actors and Directors relations currently contains two values (that of the city and the country), it is not atomic. We have therefore created a Birthplace relation for this information, and birthplaceID will be a foreign key in the Actors and Directors relations for conveying the PoB information of an actor or director.
Birthplace(birthplaceID:integer, city:string, country:string)

2.2. Create a relation schema for each relation.

1. Media(mediaID:integer, name:string, year:integer, country:string, director:integer, genre:string, length:numeric, rating:numeric)
2. Customers(custID:integer, name:string, DoB:date, email:string, phoneNumber:string, address:string, discountPercentage:integer)
3. Administrators(adminID:integer, position:string, name:string, DoB:date, email:string, phoneNumber:string, address:string)
4. Watchlist(listEntryID:integer, userProfile:integer, movie:integer, episode:integer, rating:numeric, progress:integer)
5. SubscriptionDuration(startDate:date, expirationDate:date)
6. SubscriptionPayment(subType:string, paymentAmount:integer)
7. SubscriptionPlan(subId:integer, startDate:date, subType:string, creditCardInfo,:string customer:integer)
8. UserProfiles(upID:integer, name:string, email:string)
9. Directors(directorID:integer, name:string, DoB:date, gender:char, PoB:integer)
10. TVseries(TVseriesID:integer, name:string, year:integer, country:string, genre:string, numberOfSeasons:integer, rating:numeric)
11. Episodes(episodeID:integer, TVseries:integer, name:string, director:integer, length:numeric, season:integer, rating:numeric)
12. Actors(actorID:integer, name:string, DoB:date, gender:char, PoB:integer)
13. CustomerProfiles(upID:integer, customer:integer)
14. MediaFranchise(franchiseID:integer, original:string)
15. franchisePrequels(franchise:integer, prequel:string)
16. franchiseSequels(franchise:integer, sequel:string)
17. franchiseRemakes(franchise:integer, remake:string)
18. MediaActors(movie:integer, actor:integer)
19. TVseriesActors(TVseries:integer, actor:integer)
20. Birthplace(birthplaceID:integer, city:string, country:string)

2.3. Generate sample data based on your database schema (Note: three to four tuples in each).

Media:

mediaID	name	year	country	director	genre	length	rating
1	The Movie	1999	USA	1	comedy	1.50	10

mediaID	name	year	country	director	genre	length	rating
2	The Wow	2000	India	4	comedy	1.40	9
3	The What	2001	China	43	romance	2.20	4

Customers:

custID	name	DoB	email	phoneNumber	address	discountPercentage	userProfile
1	John Doe	1789-07-14	john@email.com	0704356187	Country Roads 1	0	1
2	Jane Doe	1780-01-15	jane@email.com	0704632785	Takeme Home 2	0	2
3	Johnson Doe	1432-03-12	johnson@email.com	0704567832	Toda Place 3	0	3

Administrators:

adminID	position	name	DoB	email	phoneNumber	address
1	Junior	Janeson Doe	1232-09-13	janeson@email.com	0708324567	Road Gate 1
2	Senior	Ronda Crane	2000-02-13	ronda@email.com	0706584743	Gate Blvd 2
3	Senior	Howard Hare	1876-05-19	hare@email.com	0705432456	Blvd Road 3

Watchlist:

listEntryID	userProfile	movie	episode	rating	progress
1	2	3	NULL	8	40
2	2	NULL	87	10	90
3	5	44	NULL	3	20

SubscriptionDuration:

startDate	expirationDate
2020-01-23	2021-01-23
2020-01-24	2021-01-24
2020-01-27	2021-01-27

SubscriptionPayment:

subType	paymentAmount
Individual	49
Couples	69

subType	paymentAmount
Family	99

SubscriptionPlan:

subID	subType	startDate	creditCardInfo	customer
1	Couples	1999-01-18	Vysa CCN 12/25 John Doe	13
2	Family	2000-01-31	Vysa CCN 10/26 Jane Doe	222
3	Family	2015-05-15	Vysa CCN 11/24 Crane Snow	41

UserProfiles:

upID	name	email
1	John Doe	john@email.com
2	Jane Doe	jane@email.com
3	Johnson Doe	johnson@email.com

Directors:

directorID	name	DoB	gender	PoB
1	Name Namerson	1992-12-31	M	1
2	Cyka Blyat	1967-01-23	M	1
3	Murica Bill	1999-12-30	M	3

TVseries:

TVseriesID	name	year	country	genre	numberOfSeasons	rating
1	Who of What	1999	USA	comedy	4	10
2	When of Where	2001	USA	drama	3	9
3	To and From	1669	Sweden	drama	8	8

Episodes:

episodeID	TVseries	name	director	length	season	rating
1	1	Pilot	2	50	1	10
67	3	Deep Fried	27	44	3	8
560	18	Just a Memer	500	22	7	9

Actors:

actorID	DoB	gender	PoB	name

actorID	DoB	gender	PoB	name
1	1934-02-12	M	3	Brotherman Bill
2	1946-02-15	F	2	Martha West
3	1984-05-16	M	4	Jan Slaussen

CustomerProfiles:

userProfile	customer
1	1
2	1
3	2

MediaFranchises:

franchiseID	original
1	34
2	350
3	67

FranchisePrequels:

franchise	prequel
1	54
1	55
2	56

FranchiseSequels:

franchise	sequel
5	549
5	550
12	198

FranchiseRemakes:

franchise	remake
12	89
15	151
15	152

MediaActors:

movie	actor
1	443
1	621
1	23

TVseriesActors:

TVseries	actor
1	421
1	732
1	10

Birthplace:

birthplaceID	city	country
1	Moscow	Russia
2	London	England
3	New York	USA
4	Oslo	Norway

3. Data Model

In order for the Product Owners to understand your reworked design, you are expected to present a full data model of the designed database.

3.1. Create CHEN E/R-diagram(s) of the whole database.

- You MUST use the correct arrows and notations to indicate the multiplicity of a relationship.
- Students MUST hand in clear easy to read diagrams.