COMP40725 – Intro to RD & SQL Programming

Nicolas Di Costanzo – 21209060
nicolas.dicostanzo@ucdconnect.ie

NUTFLUX

Contents

1. Introduction

This project will try to describe, at the data level, what a visual content database could be for different kind of target audience. With this database design, I will try to capture all relevant information for a visual content application. A website and/or a mobile application that has two main objectives:
- to provide content to the user, movies, series, shows etc...
- to provide information about this content.

The first point is not described in the database. Indeed, it is not a data modelling problem. However, each entity that represents a content would be linked to an audio-visual content present on a server. But the existence of this audio-visual content is not captured by the database design itself although each entity in the *content* table represents, in addition to the information recorded by the database, the corresponding audio-visual file.

The second point concerns the information contained by the application. This is where the work on the database modelling can make the application relevant (or not). Relevant modelling can not only lead users to find the content they are looking for if they have a specific idea in mind, but also to make relevant suggestions based on the content they have previously consumed. Also, the addition of relatively unusual information, such as quotes and information about the private lives of people related to a piece of content, can help to engage the user and make them more active in the content they are consuming.

There are two main audiences targeted by this application. "Movie nuts" (pro-users) and casual viewers (standard-users). The base must contain information that are relevant for both. All data relevant to standard-users will be relevant to pro-users but all data relevant to pro-users will not necessarily be relevant to standard-users. However, the database must contain both set of data. The difference will be managed in the way the information appears to users. And this information and the way it is displayed will depend on the status of the user (standard or pro user). The database design does not make any difference between pieces of information that are specifics to a certain type of user: this will be managed on the application itself. However, in this report (in the *Database Views* section) I will show which kind of information will be available to which kind of user.

This database will be used for a public application (pro-users are also part of the public) so it does not need to store very specific information that would only be useful for professionals, for example highly technical data such as standards related to cinematography.

I would like to propose a database which, in addition to giving the classical and necessary information on a video content (date, name, actors etc...) would give information on the ethics of the people involved in its production. For example, if one of the actors has been accused of sexual or physical assault or embezzlement. How far these accusations have gone: whether there were only rumors, or whether there was a trial or conviction. These additional pieces of information about a content or a person would not only be available to pro-user. Indeed, we can easily imagine that standard users and people who are not "movie-nuts" per say could also be interested in this type of since it is not a characteristic directly linked to the world of cinema, but of ethics, detached from any cinematographic interest. Any type of consumer can be interested in this information and choose to watch or not a content on these criteria.

More and more people are interested in this aspect of industrial and cultural production. We can see this with the development of organic and fair-trade labels on food products for example. In the world of culture, the numerous controversies that take place each time a work by Roman Polanski is released are good examples of this tendency. The director was convicted in 1977 by the American justice system for the rape of a 13-year-old girl and is still considered a fugitive by the USA. As a result, many people boycott his productions and protest when he receives awards.

More recently, the Weinstein affair has had an international impact. It has brought to light practices of intimidation, abuse of power and sexual assault in the film industry. As a result, some moviegoers have decided to boycott Weinstein productions and even more broadly Hollywood productions, as Hollywood is seen as a central player in the trivialization of these practices.

Of course, ethics encompasses many different parameters such as the inclusion of people from minorities, who are gender fluid, disabled etc... But in this work, I will not take these aspects into account, I will only focus on rumors, accusations, and convictions of crimes as it is sufficient to showcase the idea's potential.

The difference between standard and pro-users is that standard users will only have access to data explicitly given by the application: contents' name, year, persons working on it etc… While pro-users will be able to fetch pieces of information, they will want to access to, through SQL queries. On their account page, pro-users will have a "Do my own searches" section that will give them access to a text field in which to write SQL queries. If they make a valid query, the information given by that query will then be displayed at the bottom of the text field. If they make an error in their request, they will see the error messages that any SQL-user sees.

The information contained in the database about the content, such as the name of the film, the name of the characters, the studio that produced it, is taken from IMDb. Information related to social relationships and involvement in a crime is taken from newspaper articles or Wikipedia.

2. Database Plan: A Schematic View

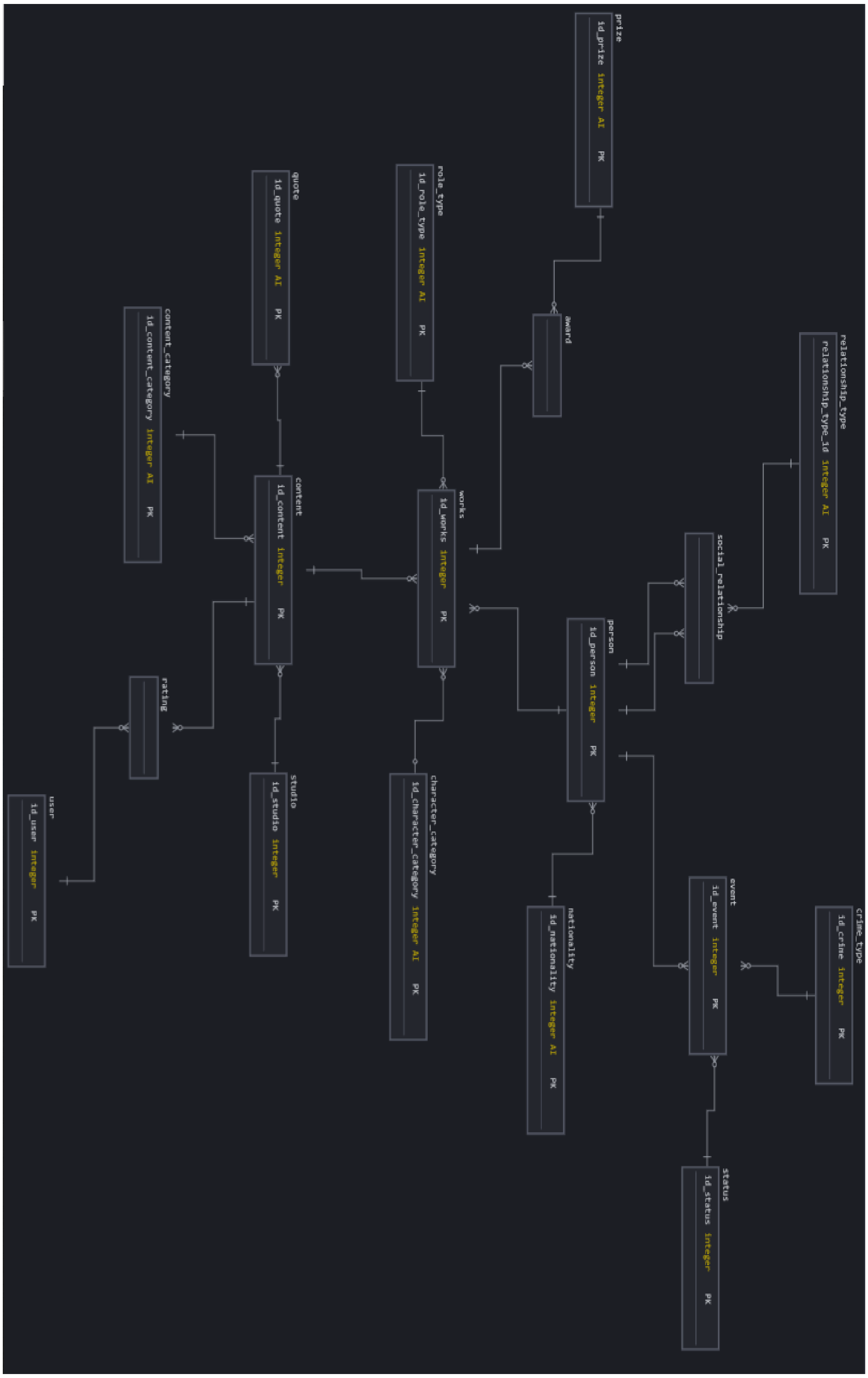
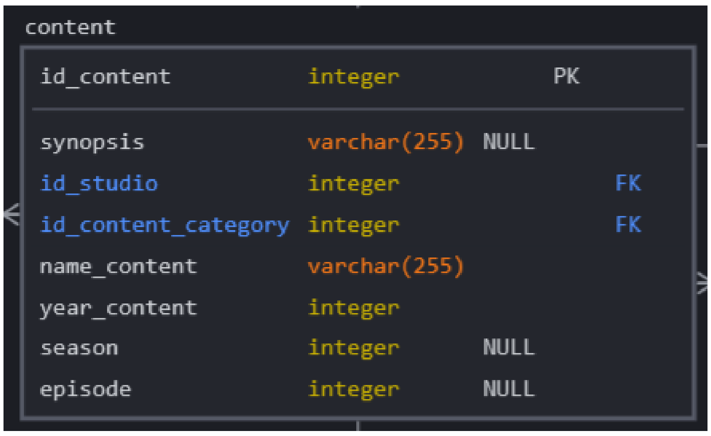


Figure 1: Database design (only primary keys are shown)

Here are the main tables and their attributes' description:

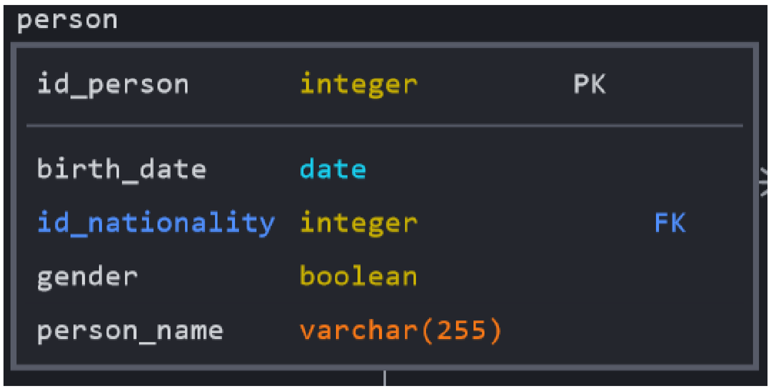


content			
id_content	integer		PK
synopsis	varchar(255)	NULL	
id_studio	integer		FK
id_content_category	integer		FK
name_content	varchar(255)		
year_content	integer		
season	integer	NULL	
episode	integer	NULL	

Figure 2: Content table

Table which stores a visual content (movie or series instance for example).

- synopsis: content synopsis
- id_studio: studio's ID which produced the content
- id_content_category: content category's ID (if it is comedy, action, horror…)
- name_content: content's name
- year_content: released year
- season: if it is part of a TV series, then the season of that episode. Is NULL otherwise
- episode: if it is part of a TV series, then the episode number. Is NULL otherwise

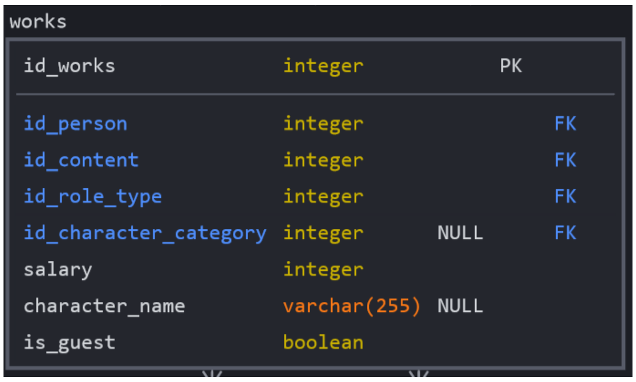


person		
id_person	integer	PK
birth_date	date	
id_nationality	integer	FK
gender	boolean	
person_name	varchar(255)	

Figure 3: Peron table

Table which stores each person present in the database. A person can be an actor, a director, a writer or have several roles at the same time. This is the reason why it is not an *actor* table for example.

- birth_date: date of birth
- id_nationality: nationality’s ID
- gender: gender (is 0 for a male, 1 for a woman)
- person_name: person’s full name



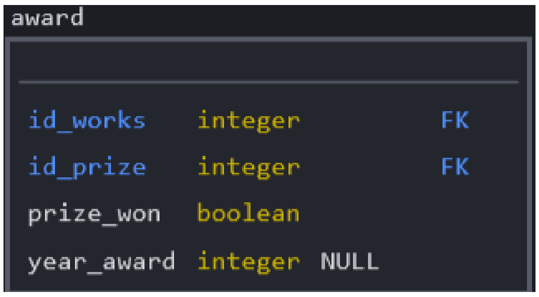
works

id_works	integer		PK	
id_person	integer			FK
id_content	integer			FK
id_role_type	integer			FK
id_character_category	integer	NULL		FK
salary	integer			
character_name	varchar(255)	NULL		
is_guest	boolean			

Figure 4: works table

Table which stores the relationship between the *content* and *person* tables. *works* table will captures all relevant information about someone's work on a piece of content. Since a person can have several roles on a same content, it has to be a separate table from the *person*’s table.

- id_person: ID of the person’s instance about whom this work is about
- id_content: content’s ID on which the person has worked
- id_role_type: role type’s ID (actor, director, voice actor etc…)
- id_character_category: if it is an acting job, which type of character have been player? Spy, detective, villain etc…
- salary: salary perceived for this work
- character_name: character’s name if there is one.
- is_guest: if this person worked as a guest on this content.



award

id_works	integer		FK
id_prize	integer		FK
prize_won	boolean		
year_award	integer	NULL	

Figure 5: award table

Table which stores relevant information about an award. This table is a relationship table between *works* and *prize* table.

- id_works: work related to that award
- id_prize: prize's id
- prize_won: is 1 if the prize has been won, is 0 if it has just been a nomination.
- year_award: nomination year

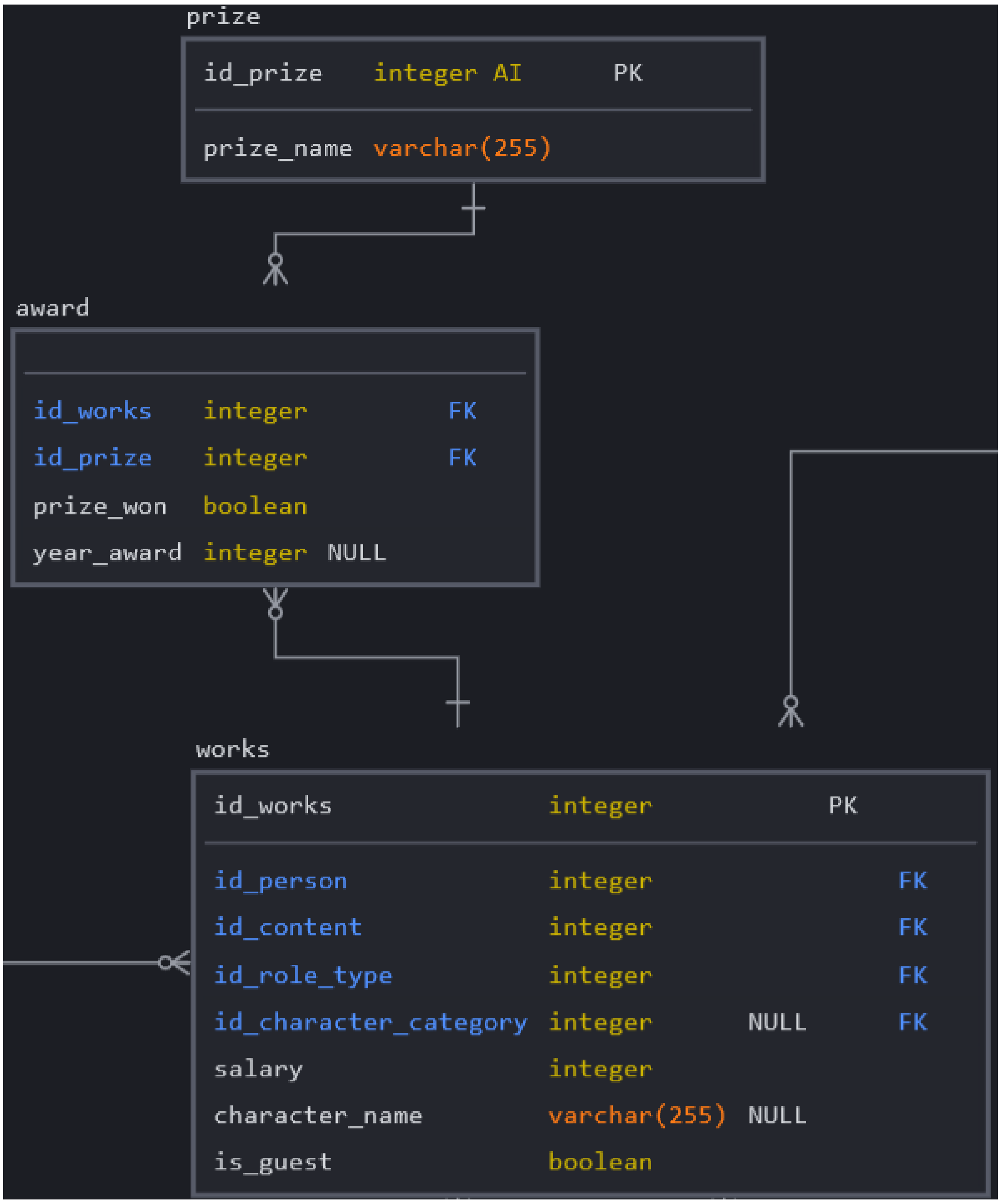
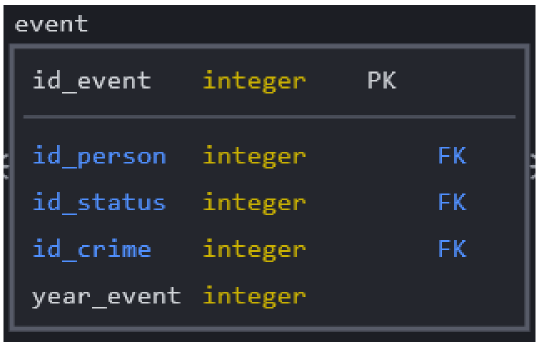


Figure 6: relationship between a work and a prize

An award is linked to a person's work and not directly to the person. Also, a person can receive several prizes and/or nominations for a same work as well as a prize can concerned different nominations. That is why a relationship table is required to link *prize* and *works* table and why this relationship table has references to the person and prize's ID.



```
event
id_event    integer    PK
id_person   integer         FK
id_status   integer         FK
id_crime    integer         FK
year_event  integer
```

Figure 7: event table

The *event* table is a link between a person instance and an event. An even is a condemnation, a rumor, an acquittal… for a crime.

- id_person: person's ID linked to that event
- id_status: event status. If it is a rumor, if the person has been condemned, acquitted, if charges has been dropped etc…
- id_crime: type of crime's ID
- year_event: when the event occurred

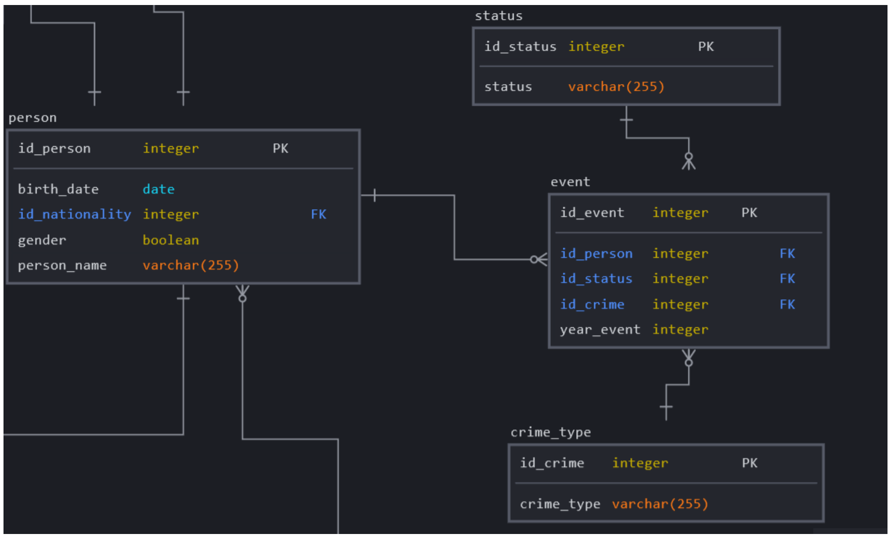
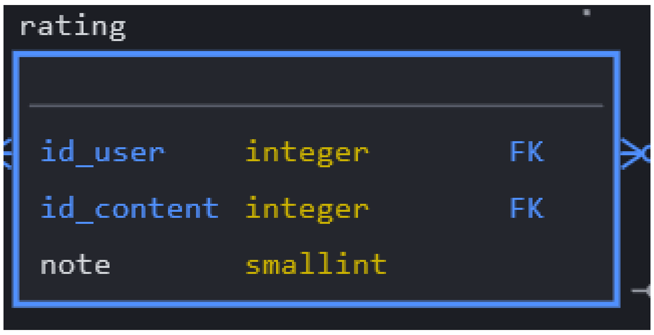


Figure 8: Modelling of the relationship between a crime and a person

A person instance can have several events associated with, so we need a relationship table between a person and a crime type. This relationship table has to store the type of crime, the "status" of the event and the person. At first, the primary key of this table was a concatenation of these three foreign keys. However, I realized that a same person can be linked the same way to a same type of crime. Therefore, this relationship table needs a dedicated primary key.



```
rating
id_user      integer     FK
id_content   integer     FK
note         smallint
```

Figure 9: rating table

A content can be rated several times and a user can rates several contents. However, a user is only able to rate each content once. So, we need a relationship table between a content and a user.

The *rating* table is the table which describes the relationship between a content and a user. In this database design, this is the only link which exists between these two tables. A user will be able to rate a content and each note has to capture three pieces of information:

- *id_user*: user's id who rated
- *id_content*: content rated
- note: note (on 10) given by the user to the content

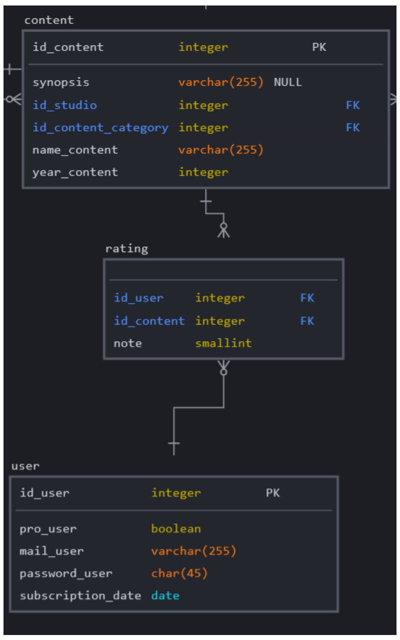
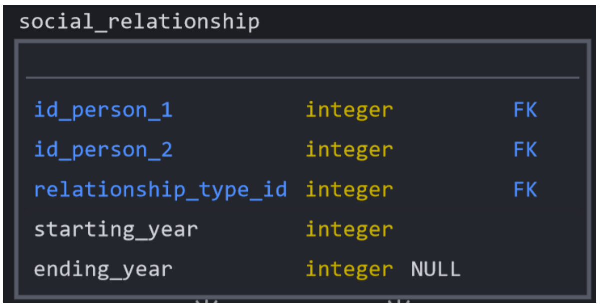


Figure 10: Modelling of the relationship between a content and a user

Another important piece of information to capture in the database is the relationship between persons. A relationship can be a love, family, friendship, or professional relationship between two persons. So, we need a relationship table between two persons instances. That is why there are two foreign keys from the *person* table. Also, information is needed on the nature of the relationship, that is what relationship_type_id does by referencing the ID of the relationship table in the relationship_type table. Lastly, we would like to know when the relationship has started and, if it is not an actual relationship, when it ended.

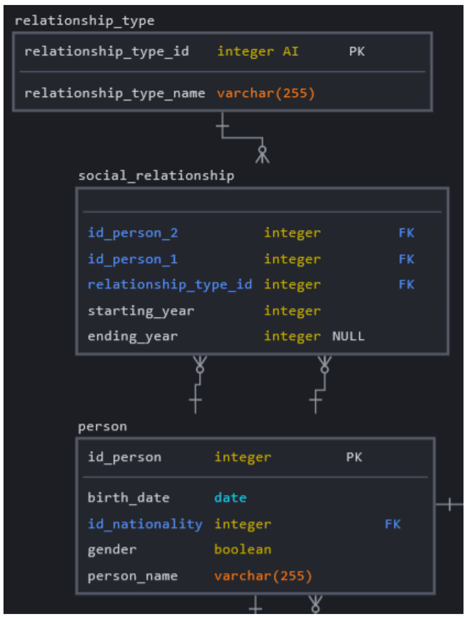


social_relationship

id_person_1	integer	FK
id_person_2	integer	FK
relationship_type_id	integer	FK
starting_year	integer	
ending_year	integer NULL	

Figure 11: social_relationship table

- id_person1: First reference to the *person* table
- id_person2: Second reference to the *person* table
- relationship_type_id: Reference to the *relationship_type* table
- starting_year: relationship starting year
- ending_year: relationship ending year (can be NULL if the relationship has not yet ended or cannot end)



relationship_type

relationship_type_id	integer AI	PK
relationship_type_name	varchar(255)	

social_relationship

id_person_2	integer	FK
id_person_1	integer	FK
relationship_type_id	integer	FK
starting_year	integer	
ending_year	integer NULL	

person

id_person	integer	PK
birth_date	date	
id_nationality	integer	FK
gender	boolean	
person_name	varchar(255)	

Figure 12: Relationship modelling between two persons and the status of their relationship

3. Database Structure: A Normalized View

1NF

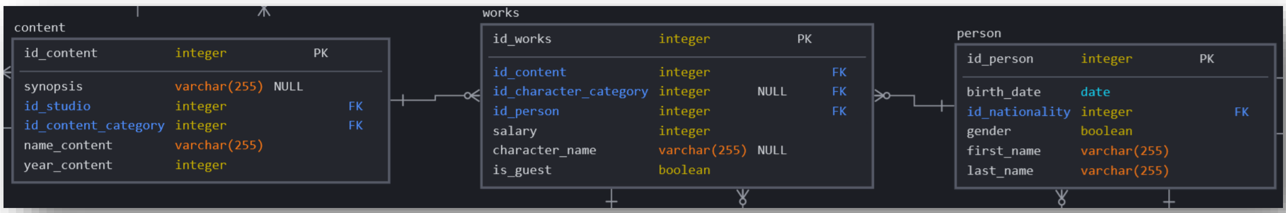
For a database to be in first normal form, it has to follow these four rules:

- Each column has to store only one value.
- All the values in a column have to be of the same type.
- Each column's name has to be unique.
- Every column has to relate to the key.

Let's look at our main tables to check how they respect these rules.

"Each column has to store only one value" rule

The database has obviously a "content" table which registers the main information about a content. Its name and its released year for example. Another interesting piece of information about content is the identity and role of the people working on it. However, this is not a relevant information to store in a column. If I had to store this information in a column named "actors" for example, I would have to put every actor's name in it and separate them with commas. But it would violate one of the 1NF rule since only one value has to be stored in a column. To solve this problem, I have a "person" and a "works" tables that link the content with every single person that worked on it. Each instance of the person table stores information about only one person, and each works' table instance stores information about the nature of the functions that the person had on this content.



content

id_content	integer		PK
synopsis	varchar(255)	NULL	
id_studio	integer		FK
id_content_category	integer		FK
name_content	varchar(255)		
year_content	integer		

works

id_works	integer		PK
id_content	integer		FK
id_character_category	integer	NULL	FK
id_person	integer		FK
salary	integer		
character_name	varchar(255)	NULL	
is_guest	boolean		

person

id_person	integer		PK
birth_date	date		
id_nationality	integer		FK
gender	boolean		
first_name	varchar(255)		
last_name	varchar(255)		

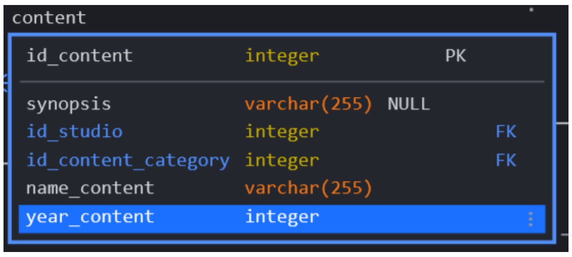
Figure 13: content, works and person table relationship

“All the values in a column have to be of the same type” rule

Let’s take the content table as an example again. The two most important information about a content is its name and its released year since it is thanks to these two properties that we manage to uniquely identify a content. But these two pieces of information have a different type since one if a number and the other one is one or several words. So, we cannot store them in the same column. That is why there is one column of type *varchar* to store the name and one of the type *integer* to store its released year.

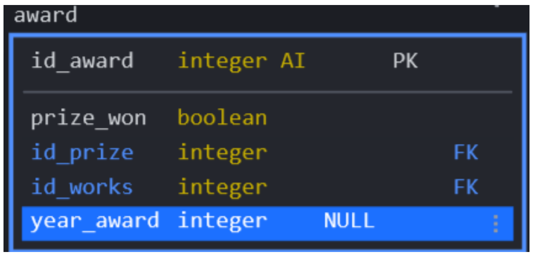
“Each column’s name has to be unique” rule

Some columns have the same meaning within a table. It is the case for the *year* property which appears in several different entities. To differentiate them, I simply add the table column after the property name such as follow:



```
content
id_content           integer            PK
synopsis             varchar(255)  NULL
id_studio            integer                 FK
id_content_category  integer                 FK
name_content         varchar(255)
year_content         integer
```

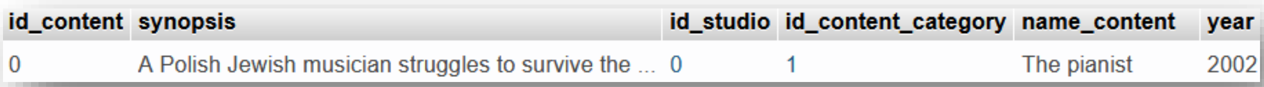
Figure 14: content table (1)



```
award
id_award    integer AI       PK
prize_won   boolean
id_prize    integer              FK
id_works    integer              FK
year_award  integer    NULL
```

Figure 15: award table

“Every column has to relate to the key” rule



id_content	synopsis	id_studio	id_content_category	name_content	year
0	A Polish Jewish musician struggles to survive the ...	0	1	The pianist	2002

Figure 16: instance of the "content" table

As we can see in this instance of the *content* table there is a reference to the IDs of the studio and the content's category. I could have directly stored these two pieces of information as *varchar* but then these columns would not have been connected to a key since several instances of the "content" table can share a same value for these properties. Instances sharing the same value for this property would not have been connected to each other in the database. This is the case if we reference the IDs of the instances of these values.

2NF

For a database to be in second normal form, it should not contain any partial dependency. Partial dependency happens when a non-prime attribute only depends on part of a prime key and not on the whole prime key. To remove Partial dependency, we can divide a table, remove the attribute, which is causing partial dependency, and move it to some other table where it fits in well.

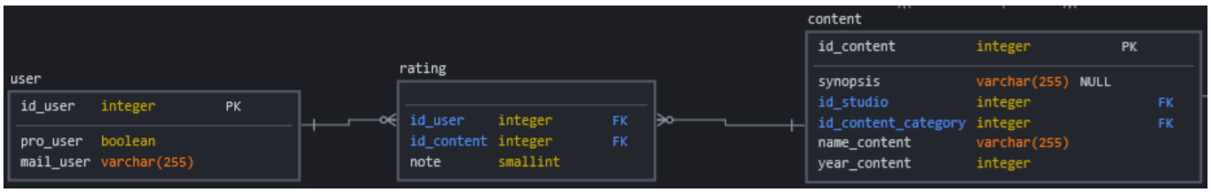


Figure 17: relationship between a user and a content

The relationship between these tables gives an example on how I manage the second normal form. If I had recorded the note as a property (and not as a foreign key) of the *user* table, it would not have relevant since it is not a sufficient information to know what the rating is. Indeed, a rating is dependent on the user ID AND on the content ID. So, we need to store these two pieces of information in a dedicated table.

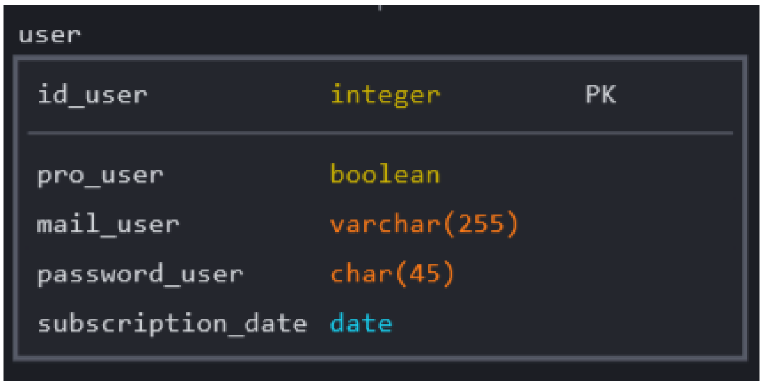
3NF

For a table to be in third normal form it needs to be in second normal form, and it should not have transitive dependency. Transitive dependency happens when an attribute depends on some non-prime attribute and not on the prime attribute.

Let's take again the example above. When I added the *note* property, I could have added it to the *content* table. But *note* does not dependent on the content ID solely. It also depends on the user ID. If I had done that, we would have an instance of each content for each note given. Since *note* property is an attribute in the *rating* table, each content has only one instance in the *content* table and several ratings an be associated to each of these content instances.

BCNF

The other table which is not in BCNF is the user's table:



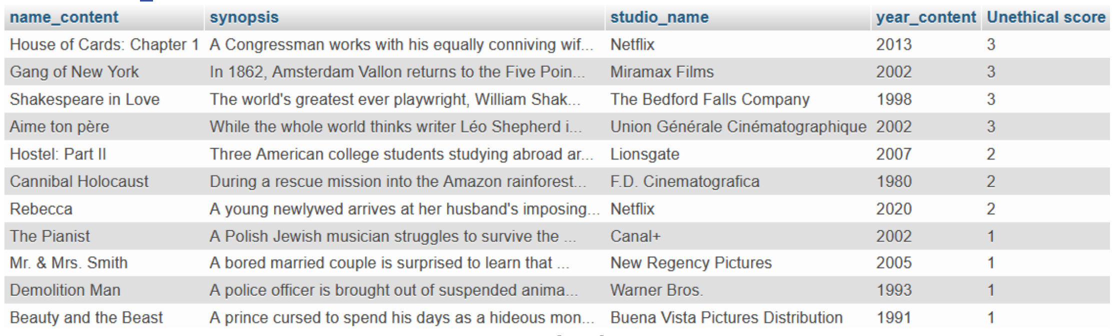
```
user
id_user            integer        PK
pro_user           boolean
mail_user          varchar(255)
password_user      char(45)
subscription_date  date
```

Figure 13: User table

User's table is not in BCNF because the attributes depend on two properties: *id_user* and *mail_user* (since an email address in unique). However, since the email address can be modified by the user, it is relevant to have a unique ID for each user. This ID will not change unlike the email address which can change.

4. Database Views

unethical_score view

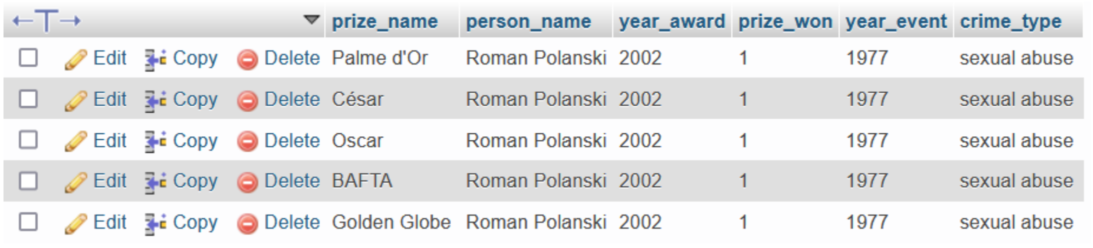


name_content	synopsis	studio_name	year_content	Unethical score
House of Cards: Chapter 1	A Congressman works with his equally conniving wif...	Netflix	2013	3
Gang of New York	In 1862, Amsterdam Vallon returns to the Five Poin...	Miramax Films	2002	3
Shakespeare in Love	The world's greatest ever playwright, William Shak...	The Bedford Falls Company	1998	3
Aime ton père	While the whole world thinks writer Léo Shepherd i...	Union Générale Cinématographique	2002	3
Hostel: Part II	Three American college students studying abroad ar...	Lionsgate	2007	2
Cannibal Holocaust	During a rescue mission into the Amazon rainforest...	F.D. Cinematografica	1980	2
Rebecca	A young newlywed arrives at her husband's imposing...	Netflix	2020	2
The Pianist	A Polish Jewish musician struggles to survive the ...	Canal+	2002	1
Mr. & Mrs. Smith	A bored married couple is surprised to learn that ...	New Regency Pictures	2005	1
Demolition Man	A police officer is brought out of suspended anima...	Warner Bros.	1993	1
Beauty and the Beast	A prince cursed to spend his days as a hideous mon...	Buena Vista Pictures Distribution	1991	1

Figure 14: Unethical score view

This view gives main information about each of the content in the database. However, it adds something the *content* table does not have: the "unethical score" for each content. This score is "calculated" by counting the number of times a content is linked to a person which is linked to the *event_table* table. For example, if an actor of the movie A is or has been involved in 1 trial, and another actor of the same movie is or has been involved in 3 rumors, the "unethical score" of the movie A will be 4. So, the bigger the score, the less ethical the content. A user will then be able to watch a content or not accordingly.

unethical_prize view



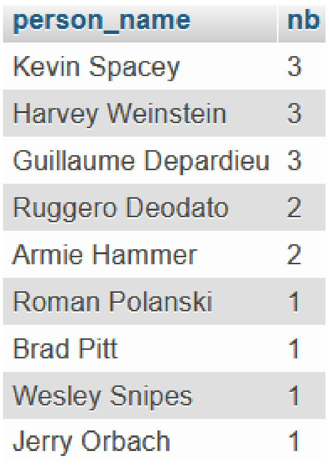
←T→				prize_name	person_name	year_award	prize_won	year_event	crime_type
☐	Edit	Copy	Delete	Palme d'Or	Roman Polanski	2002	1	1977	sexual abuse
☐	Edit	Copy	Delete	César	Roman Polanski	2002	1	1977	sexual abuse
☐	Edit	Copy	Delete	Oscar	Roman Polanski	2002	1	1977	sexual abuse
☐	Edit	Copy	Delete	BAFTA	Roman Polanski	2002	1	1977	sexual abuse
☐	Edit	Copy	Delete	Golden Globe	Roman Polanski	2002	1	1977	sexual abuse

Figure 15: Unethical prizes

This view informs the user on the all the awards won by persons who have been awarded AFTER being convicted of a crime.

This data would not be available for standard-users as it would not be store in the database nor be shown anywhere on the application but would need to be fetched via a SQL query, functionality which is only available to pro-users.

unethical_persons_unrelevant view

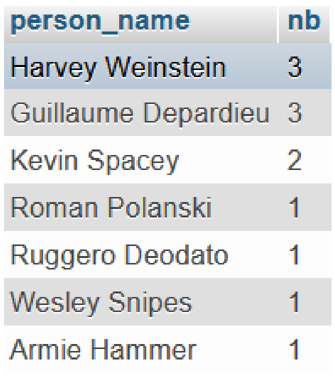


person_name	nb
Kevin Spacey	3
Harvey Weinstein	3
Guillaume Depardieu	3
Ruggero Deodato	2
Armie Hammer	2
Roman Polanski	1
Brad Pitt	1
Wesley Snipes	1
Jerry Orbach	1

Figure 16: unethical_persons (unrelevant version)

This view shows, for each person in the database, how many times he or she has been involved in a crime, in any way. That is to say that there are no differences made between the status of the prosecution: it does not consider if a person has been convicted for a crime or acquitted or if it was just rumors. This kind of information would only be available to pro-users since it would need to be fetched for the database via queries. Indeed, this is an irrelevant information to show since it mixes everything up and does not take into account the status of the accusation. In the next view, we will look at a more meaningful version of this data.

unethical_persons_relevant view

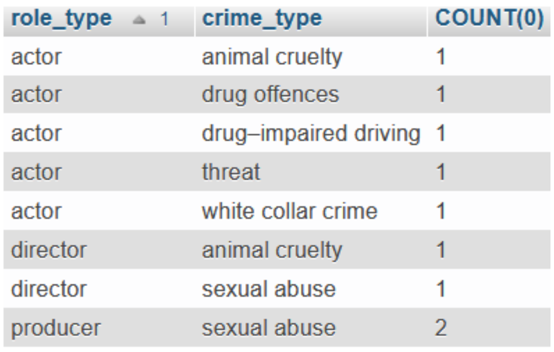


person_name	nb
Harvey Weinstein	3
Guillaume Depardieu	3
Kevin Spacey	2
Roman Polanski	1
Ruggero Deodato	1
Wesley Snipes	1
Armie Hammer	1

Figure 17: unethical_persons (relevant version)

As mentioned, this view shows a more relevant version of the last view. Indeed, it does not take into account events where the person has been acquitted or where there were only rumors or where charges has been dropped. However, for the last case, we know that charges are sometimes dropped not because the accused was not guilty but because there has been a settlement between the two parties. So this view may not be the best one. However, this piece of information would not be displayed in the application but only be available to pro-users per say. This one is just another example of what kind of information can be fetched from the database. It is up to the pro-users to manipulate the information in the way they see fit.

crime_by_role view



role_type ▲ 1	crime_type	COUNT(0)
actor	animal cruelty	1
actor	drug offences	1
actor	drug–impaired driving	1
actor	threat	1
actor	white collar crime	1
director	animal cruelty	1
director	sexual abuse	1
producer	sexual abuse	2

Figure 19: crime_by_role

This view shows how many times a role type has been convicted for each type of crime. That is to say, how many times an actor has been convicted for white collar crime, or how many times a director has been convicted for sexual abuse etc… This view does not make any difference between the persons, only between the roles. However, it can be misleading since a person can be represented several times if it has different types of roles. In this example, Ruggero Deodato is represented as a director AND as an actor convicted for animal cruelty. Indeed, Deodato has directed Cannibal Holocaust and acted in Hostel: Part II. So he appears twice in this view but has been convicted only once. I think this view is still relevant since we cannot make any difference between between Deodata the director and Deodato the actor: they are the same person and the person is responsible for a crime, not the director of Cannibal Holocaust or the actor of Hostel: Part 2. It is meaningful because this view tries to show what kind of work people involved in what kind of crime are doing.

5. Procedural Elements

Updating subscription price

The following procedure updates the subscription price for a user. The user's index is given as a parameter to the procedure. This procedure is useful because we only want to update this price when the payment deadline is reached or when the user wants to check what will be the price to pay for next month. This price depends on two properties of the *user* table:

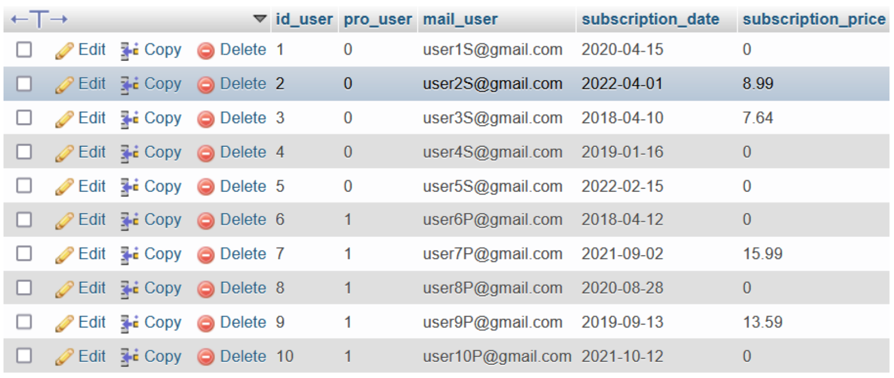
- *pro_user*: which indicates if the user is a pro user (this value is 0 if the user is a standard user and 1 if it is a pro user).
- *subscription_date*: gives the date on which the user registered.

For a standard user, the subscription is 8.99€, while it is 15.99€ for a pro-user. Moreover, if the account is registered for more than one year, the user will have a 15% discount.

It is relevant to update this information with a procedure since the price depends on the time elapsed between the payment and the subscription date. So, it needs to be updated at the user's subscription, if it changes its subscription plan, or after a payment. Indeed, after each payment, this procedure would be run again to update the price for next payment.

Here is the procedure which manages the subscription price:

```
BEGIN
DECLARE  standardPrice FLOAT DEFAULT 8.99;
DECLARE  proPrice FLOAT DEFAULT 15.99;
DECLARE  discount FLOAT DEFAULT 15;

UPDATE user SET user.subscription_price = standardPrice
WHERE user.pro_user = 0
AND user.id_user = a_userID;

UPDATE user SET user.subscription_price = proPrice
WHERE user.pro_user = 1
AND user.id_user = a_userID;

UPDATE user
SET user.subscription_price
= ROUND(user.subscription_price * ((100 - discount) / 100), 2)
WHERE user.subscription_date < DATE_SUB(NOW(),INTERVAL 1 YEAR)
AND user.id_user = a_userID;
END
```



				id_user	pro_user	mail_user	subscription_date	subscription_price
☐	Edit	Copy	Delete	1	0	user1S@gmail.com	2020-04-15	0
☐	Edit	Copy	Delete	2	0	user2S@gmail.com	2022-04-01	8.99
☐	Edit	Copy	Delete	3	0	user3S@gmail.com	2018-04-10	7.64
☐	Edit	Copy	Delete	4	0	user4S@gmail.com	2019-01-16	0
☐	Edit	Copy	Delete	5	0	user5S@gmail.com	2022-02-15	0
☐	Edit	Copy	Delete	6	1	user6P@gmail.com	2018-04-12	0
☐	Edit	Copy	Delete	7	1	user7P@gmail.com	2021-09-02	15.99
☐	Edit	Copy	Delete	8	1	user8P@gmail.com	2020-08-28	0
☐	Edit	Copy	Delete	9	1	user9P@gmail.com	2019-09-13	13.59
☐	Edit	Copy	Delete	10	1	user10P@gmail.com	2021-10-12	0

Figure 17: user's table after updating subscription_price for 4 users

Updating content name if it starts with "The "

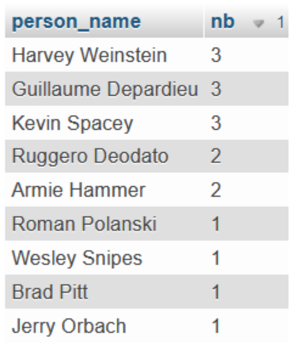
This trigger is called each time a new insertion is made to the *content* table. If a new content is inserted and its name is of the form "The content name", thanks to this trigger it will be modified into "Content name, The". It is useful for better indexing. For example, if we insert the movie "The Return of the King", it will be stored as "Return of the King, The".

This feature needs to be implemented via a trigger since this is a check that must be done before inserting new content into the database. Here is the trigger:

```
DROP TRIGGER IF EXISTS `UpdateContentName`
DELIMITER $$
CREATE DEFINER = `root`@`localhost` TRIGGER `UpdateContentName` BEFORE
INSERT ON `content`

FOR EACH ROW BEGIN
	IF NEW.name_content LIKE "The %" THEN
		SET NEW.name_content = REPLACE(NEW.name_content, "The ", "");
		SET NEW.name_content = CONCAT(NEW.name_content, ", The");
	END IF;
END$$
DELIMITER ;
```

6. Example Queries: Your Database in Action

Query 1: Number of times a person has been involved in a crime



person_name	nb
Harvey Weinstein	3
Guillaume Depardieu	3
Kevin Spacey	3
Ruggero Deodato	2
Armie Hammer	2
Roman Polanski	1
Wesley Snipes	1
Brad Pitt	1
Jerry Orbach	1

Figure 18: Number of times a person has been involved in a crime

```
SELECT person_name, COUNT(*) AS nb
FROM event_table E, status_table S, person P
WHERE E.id_person = P.id_person
AND E.id_status = S.id_status
AND (S.status != "rumour" OR S.status != "acquitted")
GROUP BY person_name
ORDER BY nb DESC;
```

This query counts the number of times a person has been involved in a crime. He or she has been to court or is currently the target of a police investigation. This piece of information is the basis of the application. Indeed, the strong idea of this design is to add information about the ethics of people involved in the film industry and this query, although basic, is the heart of what the database wants to provide to users.

Here, we only need to use three different tables:

- *person* table which stores all information about a person. We need it to display the name of the person concerned as well as make a link between the person's ID as well as the event.

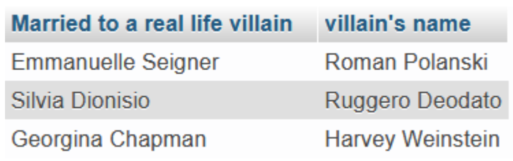
- *event_table* is needed because it is the relationship table between a person and a crime accusation's status. So, we need to join person table and *event_table* to make a connection between a person and each of his “crime”

- *status_table* is only needed to give explicit information about the status. I could have not used that table and doing the query below and get the same result:

```
SELECT person_name, COUNT(*) AS nb
FROM event_table E, person P
WHERE E.id_person = P.id_person
AND (E.id_status != 2 OR E.id_status != 3)
GROUP BY person_name
ORDER BY nb DESC;
```

However, for clarity, it is a better practice to use explicit names when it is possible, instead of IDs which can be confusing.

Query 2: People married to a real-life villain



Married to a real life villain	villain's name
Emmanuelle Seigner	Roman Polanski
Silvia Dionisio	Ruggero Deodato
Georgina Chapman	Harvey Weinstein

Figure 19: People who are or have been married to someone who has been convicted for a crime

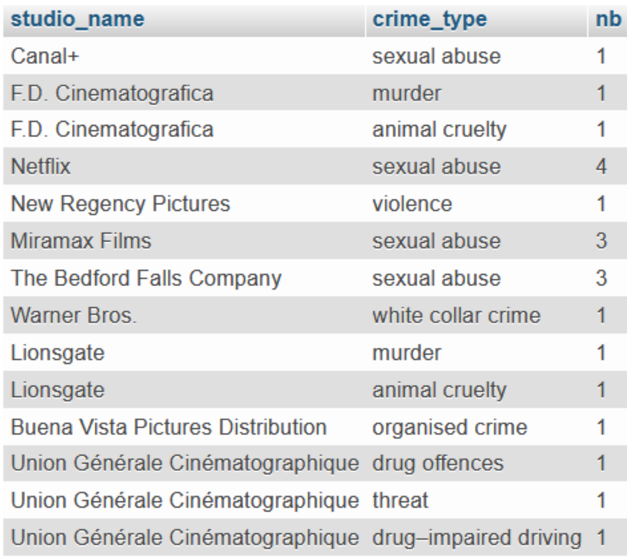
```
SELECT P1.person_name AS "Married to a real life villain", P2.person_name AS
"villain's name"
FROM person P1, person P2, social_relationship SR, relationship_type R,
event_table E, status_table S WHERE P1.id_person = SR.id_person_1 AND
P2.id_person = SR.id_person_2
AND SR.relationship_type_id = R.relationship_type_id
AND P2.id_person = E.id_person
AND E.id_status = S.id_status
AND S.status = "convicted"
AND R.relationship_type_name = "marriage"
UNION
SELECT P2.person_name AS "Married to a real life villain", P1.person_name AS
"villain's name"
FROM person P1, person P2, social_relationship SR, relationship_type R,
event_table E, status_table S
WHERE P2.id_person = SR.id_person_2
AND P1.id_person = SR.id_person_1
AND SR.relationship_type_id = R.relationship_type_id
AND P1.id_person = E.id_person
AND E.id_status = S.id_status
AND S.status = "convicted"
AND R.relationship_type_name = "marriage";
```

The query above shows all people who are or have been married to someone who has been convicted for a crime (people “married to a real-life villain”). To get this information from the database we need to use the relationship-related tables (*social_relationship* and *social_relationship_type* tables), the event-related tables (*event_table* and *status_table*) as well as the the *person* table twice.

The relationship-related tables are needed to get all marriages between two persons. That is why we need two *person* tables, to get a reference to the two people involved in the relationship. The event-related tables are used to link the bride and the groom to a crime (if this link exists) thanks to the property *id_person* which is common to the *person* table and the *event_table* table. The *status_table* is required to filter the results and only get people who has been convicted of a crime, which is the only information we are interested in.

Finally, since we do not know if the person, we are looking for is in the *social_relationship* under the property *id_person_1* or *id_person_2*, a UNION is used to get both set of results.

Query 3: Most popular crime by studio



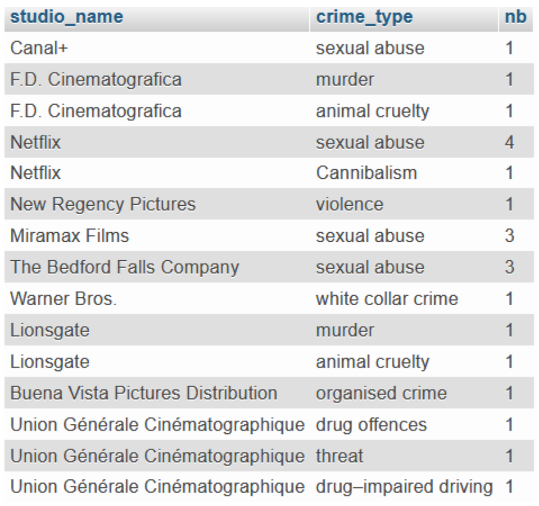
studio_name	crime_type	nb
Canal+	sexual abuse	1
F.D. Cinematografica	murder	1
F.D. Cinematografica	animal cruelty	1
Netflix	sexual abuse	4
New Regency Pictures	violence	1
Miramax Films	sexual abuse	3
The Bedford Falls Company	sexual abuse	3
Warner Bros.	white collar crime	1
Lionsgate	murder	1
Lionsgate	animal cruelty	1
Buena Vista Pictures Distribution	organised crime	1
Union Générale Cinématographique	drug offences	1
Union Générale Cinématographique	threat	1
Union Générale Cinématographique	drug–impaired driving	1

Figure 20: Most popular crime by studio

```
DROP VIEW IF EXISTS tempview;

CREATE VIEW tempView AS
(SELECT S.studio_name, CT.crime_type, COUNT(*) AS nb
FROM studio S, content C, works W, person P, event_table E,
crime_type CT
WHERE S.id_studio = C.id_studio
AND C.id_content = W.id_content
AND W.id_person = P.id_person
AND P.id_person = E.id_person
AND E.id_crime = CT.id_crime
GROUP BY S.id_studio, CT.id_crime
ORDER BY S.id_studio);

SELECT t1.*
FROM tempview t1
INNER JOIN
(SELECT studio_name, MAX(nb) AS max_nb
FROM tempview
GROUP BY studio_name) AS t2
ON t1.studio_name = t2.studio_name
AND t1.nb = t2.max_nb;[1]

DROP VIEW tempview;
```

This query gives information about the "most popular" crime by studio. It associates each studio with a crime. This crime is the one which has been the most committed by people who worked for the studio. If one or several crimes has been committed the same number of times, all of them are shown.

A studio is linked to a content since each content is released by a studio, a content is linked by a person through the *works* table. And the *person* table is related to a crime thanks to the *event_table* table. To make this query I use a temporary view (*tempview*) to make things easier:



studio_name	crime_type	nb
Canal+	sexual abuse	1
F.D. Cinematografica	murder	1
F.D. Cinematografica	animal cruelty	1
Netflix	sexual abuse	4
Netflix	Cannibalism	1
New Regency Pictures	violence	1
Miramax Films	sexual abuse	3
The Bedford Falls Company	sexual abuse	3
Warner Bros.	white collar crime	1
Lionsgate	murder	1
Lionsgate	animal cruelty	1
Buena Vista Pictures Distribution	organised crime	1
Union Générale Cinématographique	drug offences	1
Union Générale Cinématographique	threat	1
Union Générale Cinématographique	drug–impaired driving	1

Figure 21: tempview

[1] https://stackoverflow.com/a/12102216

Once this view is created, we need to compare two instances: *t1* which is the view as it is, and *t2* which gives the maximum value for the *nb* property for each studio. If the value given by the *t1.nb* is the same as the maximum value for the *nb* property given by *t2*, then it is the *crime_type* value we want to show to the user.
This piece of information is specific so it would not be available to standard users. However, it could be an information directly linked to each studio but only available to pro-users. But pro-users would not need to request it since it would be shown on each studio's page.

7. Conclusions

Through this report we have seen the design of a video content (movies, series, documentaries etc...) database. This design tries to capture the most important elements that users might need to decide what content to consume.

One of the important things that NUTFLUX Inc. asked for was to stand out from other competitive offers to attract new subscribers. Indeed, there are many different platforms of visual content and an application that enters the market needs to stand out in one way or another. Of course, it is important that the scope of the application's offer is as broad as possible. However, this is not something that we, as database administrators, have any control over. That is why we thought of a way to stand out from the competition through the design of the database, i.e., through the information it stores.

With the growing public interest in the ethical issues of those involved in a production, we decided to incorporate this dimension into our database. We did this by linking to each person the possible crimes to which that person is linked. This allows consumers to be aware, without having to do any further research on their own, of the behavior of those involved in the creation of the content available. Thus, for each piece of content a user could have access to the history of all the registered persons linked to that content to see if they are of virtue.

However, in order not to hinder the completeness of the offer, the content should not be filtered upstream by NUTFLUX Inc.. All possible content should be available, regardless of its "ethical level". If a user wants to watch a content where all actors and producers are murderous pyromaniacs and mysogins sentenced to life, they should be able to do so. The purpose of the application is not to not provide certain content based on ethic, but to allow users to judge whether they want to watch content on that basis, in addition to the more usual criteria (actors, directors, awards etc...) which are also present in the design.

As I said in the introduction, ethics encompasses many more criteria than just criminal convictions, but this design only takes into account that type of information. However, a few changes would be necessary to consider other types of ethical issues. Let's say we want to inform users about the cultural diversity of people who have worked on a project. We already have a *nationality* table that allows us to know the number of different nationalities that participated in the creation of a content. If we want to take into account sexual orientation, we already have the *social_relationship* table which records social relationships between people, and a *gender* field in the *person* table. By combining these pieces of information, we can find out who is or has been involved in homosexual relationships for example.

We can see that a lot of different information is already contained in the current design and that further development of the project's main idea, i.e. informing users about the ethical aspect, only needs to be deepened. However, the current database would already allow the expansion of this project without having to change the design.

Acknowledgements

I acknowledge that the work is entirely my own and that every sentence in this report has been written by me and myself only, except where explicitly stated.

References

Michael A. Peters & Tina Besley (2019) Weinstein, sexual predation, and ‘Rape Culture’: Public pedagogies and Hashtag Internet activism, Educational Philosophy and Theory, 51:5, 458-464, DOI: 10.1080/00131857.2018.1427850

Shelley Cobb & Tanya Horeck (2018) Post Weinstein: gendered power and harassment in the media industries, Feminist Media Studies, 18:3, 489-491, DOI: 10.1080/14680777.2018.1456155

Sophie Hennekan & Dawn Benett (2017) Sexual Harassment in the Creative Industries: Tolerance, Culture and the Need for Change, Gender, Work & Organization
Volume 24, Issue 4, 417-434, DOI: 10.111/12176

Rosales Law Firm, RLF (2019) Famous White-Collar Crime Cases | From Gangsters to Actors and Television Stars, Available at: https://www.rosaleslawfirm.com/blog/2019/08/famous-white-collar-crime-cases-from-gangsters-to-actors-and-television-stars/
[Accessed: 24/04/2022]

Isobel Lewis, IL. (2020). Mark Wahlberg racist hate crimes: The full list of actor's racially motivated attacks. [Independant]
Mark Wahlberg racist hate crimes: The full list of actor's racially motivated attacks.
Available at: https://www.independent.co.uk/arts-entertainment/films/news/mark-wahlberg-racist-hate-crimes-wikipedia-history-george-floyd-blm-protests-a9554191.html
[Accessed: 24/04/2022]

Glamour, G (2019). Post-Weinstein, These Are the Powerful Men Facing Sexual Harassment Allegations. [Glamour]
Post-Weinstein, These Are the Powerful Men Facing Sexual Harassment Allegations
Available at: https://www.glamour.com/gallery/post-weinstein-these-are-the-powerful-men-facing-sexual-harassment-allegations
[Accessed: 24/04/2022]