RELATIONAL DESIGN & DATA MIGRATION

FILE STRUCTURES & DATA BASES
COMPUTER SCIENCE AND ENGINEERING
UNIVERSIDAD CARLOS III DE MADRID
FIRST LAB ASSIGNMENT
3.3.2020
USER FSDB322

ANTONIO VIÑUELA PÉREZ - 100383461
MARINA TORELLI POSTIGO - 100383479

REPORT

BS DEGREE IN INFORMATICS ENGINEERING Academic year: 2019/2020 - 2nd year, 2nd term Subject: File Structures and Databases First Assignment's Report: Relational Design and Impl.	uc3m | Universidad Carlos III de Madrid

Lecturers:	Francisco Javier Calle Gómez and Mohammad Hajarian		
Group:	88/89	**Lab User**	FSDB322
Student:	Marina Torelli Postigo	**NIA:**	100383479
Student:	Antonio Viñuela Pérez	**NIA:**	100383461

Introduction

We have been assigned the task of designing and implementing a database relational model for a film-streaming platform. We have been given several statements with an explanation of the client's requirements, and access to their old database, which is heavily flawed and inefficient. Therefore, it is our duty to solve its issues while maintaining as much data as possible.

Our goal is then to redesign the whole in such way that it is prepared to inherit the data stored in the previous tables while achieving a higher efficiency.
The first step will be to redesign from scratch the new tables, key and references considering the restrictions and references established by the client. After this is done, we must proceed with the insertion of old data in the newly created structure.

This document is the lab work report which provides a throughout explanation of the process and the decisions taken in the making of this assignment. It shows explanations of the database created as well as the analysis of the semantics considered. Finally, problems encountered during the implementation on SQL are specified and so are the solutions to fix them.
Along this document, there are two SQL files:

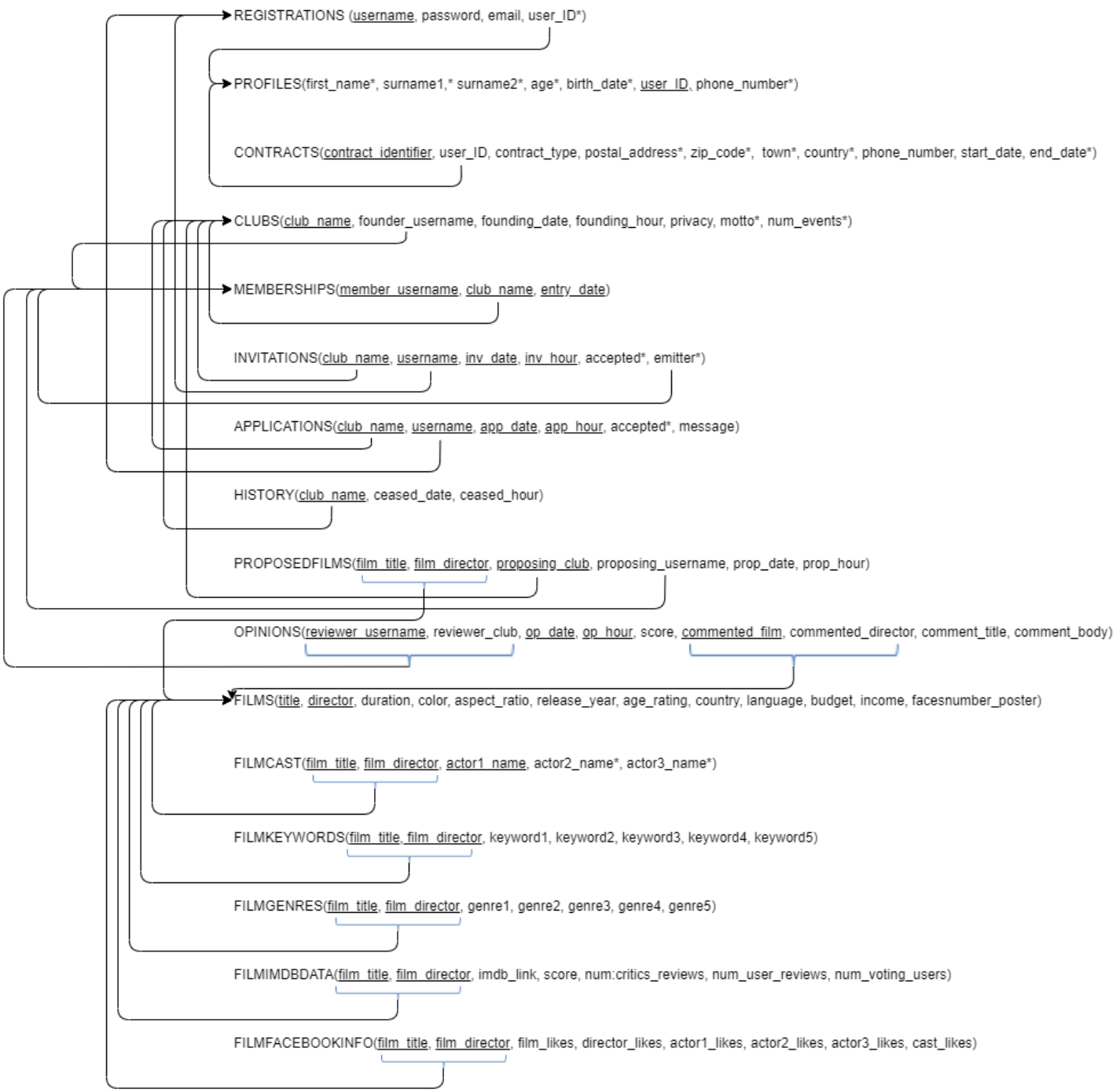
- **creation.sql:** script to create all the tables for the new database.
- **upload.sql:** script to insert all the data needed from the old database into the new one.

The document structure is as follows:

1. Relational Design
    a. Relational Schema
    b. Implicit Semantics
    c. Explicit Semantics
2. Relational Statics Implementation in SQL
    a. Re-incorporated semantics
    b. Incorporated implicit semantics
    c. Excluded semantics
3. Workload

1 Relational Design

A) Relational Schema:



B) Implicit semantics:

Presp_id	Stage	Mechanism	Description
I_1	Design	Primary and Foreign key	Each user will be uniquely identified and referenced by his/her registration's *username*
I_2	Design	Primary and Foreign key	Every user's profile (which can be referenced from the user's registration) will be uniquely identified and referenced by his/her *citizen ID*
I_3	Design	Primary key	As a user can have multiple contracts, each contract will have a unique *contract identifier*
I_4	Design	Primary and Foreign key	As a user can belong to multiple clubs, each club membership will be uniquely identified and referenced by the tuple *[username + club name]*
I_5	Design	Primary and Foreign key	There cannot be two clubs with the same name, so each club will be uniquely identified and referenced by the *club_name*
I_6	Design	Primary key	Each club's invitations and applications will be uniquely identified by the tuple *[club_name + username]* of the invited or applying user. this is assuming that a certain user can only apply or be invited once into a club
I_7	Design	Primary key	As it is impossible to post more than one opinion simultaneously by the same user, each opinion will be identified by the combination *[reviewer_user_name + date + hour]*
I_8	Design	Foreign key	Club founder's username will be stored in the Clubs table, for referencing
I_9	Design	Primary and Foreign key	As there can be two films with the same title, but not same title and director simultaneously, each film will be identified and referenced by the tuple *[title + director_name]*
I_{10}	Design	Primary key	As the same film can be proposed by different clubs, each film proposal will be identified by the combination *[film_title + film_director + proposing_club]*

Table 1: Implicit semantics incorporated into the relational graph

C) Non-observed explicit semantics:

Presp_id	Description
S_1	Passwords must have a minimum of 8 characters.
S_2	Users can modify their data anytime, except from the username as long as they have any activity already recorded
S_3	Depending on the club's privacy (open or closed) their data can be viewed or not by other users.
S_4	Date coherence must be observed for club events.
S_5	If a club is deleted (ceased), the *ceased* field will be changed to TRUE.
S_6	Content viewing can only be done by contract users. It is optional and independent.
S_7	*Accepted* field in Applications and Invitations must change if the correspondent application or invitation is accepted.
S_8	When a club is deleted, its info will be stored in a *history* table.
S_9	Phone numbers have 9 characters.
S_{10}	Citizen IDs must have 9 characters (if DNI) or 10 characters (if passport).
S_{11}	Phone number is not compulsory to create a profile, but it is if the profile wants to have a contract.
S_{12}	It is necessary to provide username, password and email to registrate.

Table 2: Non-observed explicit semantics

2 Relational Statics Implementation in SQL (DDL)

A) Re-incorporated semantics: (identifiers referred to those assigned in table 1)

Presp_id	Solution Description
S_1	Password field length minimum value is 8; a constraint CONSTRAINT CK_REGISTRATIONS CHECK (LENGTHB(password)>7) is added to REGISTRATION table
S_2	We have no mechanism to forbid updating of username if and only if it has related activities. [E1]
S_3	We do not check data integrity as specified (comparing dates) because defining functions (not covered in the lectures yet) is needed to access referenced tables from a CHECK constraint [E3], but we properly adapt the imported old dates to the wanted *yyyy-mm-dd* date or *hh24:mi* hour format.
S_4	Triggers are needed to implement the desired functionality: changing the value of *ceased* field is a club is ceased. Instead, we do not import information about ceased clubs and set default value of *ceased* field to False.
S_5	We do not believe it is possible to represent these functionalities in our model. [E4]
S_6	Triggers needed, not yet covered in lectures [E5]. *Accepted* field left null.
S_7	Triggers needed to change fields of History table when a club is ceased. Instead, we just inherit the already ceased clubs from the old database. [E6]
S_8	Phone length is 9: *phone_number* fields datatypes should be set to VARCHAR2(9). For some reason, when insertion is performed, an error message states that there are phone numbers of size 10 even though they are not found via queries. To avoid this, we simply assign a length of 10: VARCHAR2(10).
S_9	Even though all *citizenID* values in the old table have length 9, we set the length for the correspondent field in the new tables to length 10 because, theoretically, passport numbers can be used (which have a length of 10): VARCHAR2(10).
S_{10}	While *phone_number* field in Profiles table is optional, it is set to NOT NULL in the Contracts table.
S_{11}	*username*, *password* and *email* fields in Registration tables are set to NOT NULL

Table 3: re-incorporated explicit semantics

B) Incorporated implicit semantics: (numbering continues where ended in table 2)

Presp_id	Stage	Mechanism	Description
I_{13}	Implem.	On Delete Cascade	We assume that if a club is deleted, user memberships to that club are deleted too.

Table 1(cont.): implicit semantics incorporated in the definition of each table

C) Excluded semantics:

Presp_id	Description	Cause	Explicit/ Implicit
E_1	Username can only be updated if the user has no related activities	We do not know yet of a mechanism to implement this	Explicit
E_2	Checking date integrity between tables	For our current relational model, we would need to implement functions to access the required fields in different tables. There functions have not yet been covered in class.	Explicit
E_3	Content viewing can only be done by contract users. It is optional and independent.	We did not find a way to represent these functionalities in our model.	Explicit
E_4	*Accepted* field in Applications and Invitations must change if the correspondent application or invitation is accepted.	Triggers needed, not covered in lectures yet.	Explicit
E_5	When a club is deleted, its info will be stored in a *history* table.	Triggers needed, not covered in lectures yet.	Explicit

Table 5: explicit semantics excluded in the creation of each table

3 Workload (DML)

To start the process of populating the new tables, they must first be created. To this end, we divide all the tables into 3 groups:

- **Parent tables:** only referenced
- **Parent/Child tables:** both referenced and referencing
- **Child tables:** only referencing

As so, we started by creating the 3 parent tables (Films, Clubs and Profiles), which do not hold any references and are only referenced by other tables. After, the only parent/child table (Registrations) is created, both referencing other tables and being referenced. To finish, we add the rest of the tables, which all make references but are not referenced.

The next step is to populate the tables using the data contained in the old tables. We decided to make the insertion based on three "groups" of tables:

- **films-related:** Films, FilmKeywords, FilmIMDBdata, FilmGenres, FilmFacebookInfo and FilmCast.
- **users-related:** Registrations, Profiles and Contracts.
- **clubs-related**: Clubs, Invitations, Applications, ProposedFilms, Memberships, Opinions, History.

Underlined tables are the "main" tables of each group, which must be created first to ensure data integrity and the correct functioning of foreign keys.

We then proceed with the insertion of the films-related information. This goes smoothly as expected filling the tables in the order detailed above, adding the correspondent info for the 4814 films existing in the old database, until we get to the FilmCast table. We observed that in the original database there were some films with no cast, but all those films with a cast had at least 1 actor. Therefore, we included actor1_name in the composite primary key and got the insertion to work again. In this case, only 4808 rows are inserted, as those films with no cast are not included.

After this, users-related tables must be created. We immediately come across an inconsistency of the old_users table when trying to insert its data into our Registrations table: there are two users with the same citizenID:

```
SQL> SELECT citizenID FROM old_users GROUP BY citizenID HAVING count(*) > 1;

CITIZENID
---------

82768894J

SQL> SELECT nickname FROM old_users WHERE citizenID = '82768894J';

NICKNAME
--------------------------------------------------------------------------------
carhuamaca3
alfredo40
```

Given this error, we decided to exclude these two users and not add them to the new database, by using the next WHERE statement when performing the insertion:
WHERE (citizenID IS NOT NULL) AND (citizenID<>'82768894J')

After solving this issue, the insertion is performed correctly. As so, we proceed to the next table to be filled: Profiles. The previous data inconsistency causes another error here: in this case, a Foreign Key error. This happens because we again have to exclude the users with repeated citizenIDs from our databases: if we inserted them in this table, an error is raised when trying to reference the Registrations table by their citizenIDs.

```
INSERT INTO registrations SELECT DISTINCT nickname, passw, eMail, citizenID FROM old_users WHERE nickname IS NOT NULL
*
ERROR en linea 1:
ORA-02291: restriccion de integridad (FSDB322.FK_PROFILES) violada - clave
principal no encontrada
```

The last table to be filled (Contracts) does not raise any error. Luckily, neither of the users with repeated citizenIDs had taken a contract. To fill this table, we extract users from the old_users table when they have a value for contractID and phoneN, as they need to have a contracted service and a phone number registered.

We now will dump information into the last group of tables, which happens to be the trickiest of the three groups. This is because the information in the old table is stored in a quite chaotic way, with fields changing their meaning depending on other fields.
We decided to fill the Clubs table taken the information from the the old_events table when Etype is 'foundation', extracting the club founder's username, club name, founding date and motto. This is correctly performed, adding the information for the 399 clubs existing in the old database.

To dump data into Invitations and Applications, we extracted the following information from the old_events table:

- For Invitations: club name, invited username, invitation date, invitation hour and emitter; WHERE Etype = 'invitation'.
- For Applications: club name, applying username, application date, application hour and applier's introduction message; WHERE Etype = 'application'.

The first error for this group of tables comes when filling the ProposedFilms table. In our original relational model, proposed films referenced the Films table. The problem is that there are proposed films in the old database that were not included in the old_movies tables:

```
INSERT INTO proposedfilms SELECT DISTINCT details1, details2, club, client, to_d
*
ERROR en linea 1:
ORA-02291: restriccion de integridad (FSDB322.FK_PROPOSEDFILMS2) violada -
clave principal no encontrada
```

We have two possible solutions here: to only include films with existing entry in the Films table, or to ignore the foreign key constraint and just add all proposed films existing in the old_events table. We went for the second option: even though it is clearly not the optimal solution to remove a foreign key constraint, we were not able to find a way to only insert values if they exist in another table.

The next filled table is Memberships. To do so, we add users either accepted or founders of a club. This happens with no errors.

A similar error to the one observed in the ProposedFilms table occurs when treating the Opinions table: in the old_events table there are opinions from users whose usernames are not registered in the database and therefore do not have any entry in the Registrations table. In our initial model the reviewer's username referenced the Registrations table, so a foreign key violation was obtained:

```
INSERT INTO opinions (reviewer_username, reviewer_club, op_date, hour, commented_film, co
*
ERROR en linea 1:
ORA-02291: restriccion de integridad (FSDB322.FK_OPINIONS1) violada - clave
principal no encontrada
```

We went for the same solution as before: simply not referencing the Registrations table as a foreign key constraint and adding all the information contained in the previous tables.
We also found that in the previous database there were simultaneous opinions (same date and hour) from the same users referring to different films, even though it is technically impossible. To solve this, we added the field *commented_field* to the composite primary key.
In order to extract all the necessary information for the Opinions table, we implemented two INSERT operations for the two Etype values regarding opinion events:

- 'opinion': no score. We only extract reviewer's username, reviewer's club, date, hour, commented film, commented director, comment title, comment body from the old_events table, ignoring the *score* field.
- 'opinion:XX': where XX is the score (0-10). We extract the same information as before, plus the value for the *score* field, obtained from the last two digits of the Etype field value (the two 'X').

For the last remaining table to be filled (History) we simply store the names of the clubs with a ceasing event registered in the old database, as well as the date and hour of the given 'ceasing' event.