|  |
| --- |
| **Base de Dados(MIEI) – 2017/18**  **Relatório De Projeto**  **Plataforma de LiveStreaming**  **Realizado por:**  **Jorge Pereira – 49771**  **Tiago Fornelos – 49780**  **José Leal – 50623**  **[MIEI]**  **Grupo 46 (P7)** |

# Índice

[2 Descrição do Tema 2](#_Toc514540204)

[3 Modelo de Dados 3](#_Toc514540205)

[3.1 Modelo ER 3](#_Toc514540206)

[3.2 Opções de design 4](#_Toc514540207)

[3.3 Esquema Relacional 5](#_Toc514540208)

[3.3.1 Versão 1 5](#_Toc514540209)

[3.3.2 Versão 2 7](#_Toc514540210)

[3.4 Criação da Base de Dados 8](#_Toc514540211)

[3.4.1 Criação das Tabelas 8](#_Toc514540212)

[3.4.2 Criação de Sequências 12](#_Toc514540213)

[3.4.3 Criação de Triggers 13](#_Toc514540214)

[4 Limitações/opções tomadas na implementação da Base de Dados 17](#_Toc514540215)

[5 Descrição da Interface 18](#_Toc514540216)

# Descrição do Tema

Para o desenvolvimento do projeto, tomámos como inspiração as plataformas de Live Streaming que permitem, a um utilizador (Streamer), partilhar conteúdo de diversos jogos e interagir com o seu público (outros utilizadores que assumem um papel passivo de visualizadores).

Assim, decidimos desenvolver a base de dados da nossa aplicação de streaming GameHub.

Esta permite guardar informação dos *Users* registados na aplicação, que são identificados por um identificador único. Sobre estes guarda-se ainda o seu e-mail, password, data de nascimento e os canais que este segue. Como dito anteriormente, um *User* poderá ser identificado como *Streamer*, podendo realizar transmissões em direto.

Um *User* poderá ainda optar por pagar uma mensalidade e obter um estatuto *Premium*, que lhe garante vantagens como o acesso ao *Chat* da stream, participação nos eventos realizados pelo *Streamer*, melhor qualidade de imagem, etc. No pagamento desta mensalidade o *User* poderá optar por qualquer um dos métodos disponíveis no sistema e não é obrigado a pagar todas as mensalidades com o mesmo.

Para aumentar a segurança do *User*, a aplicação mantém um registo de todos os *Loggins* efetuados.

Cada Stream contém informação dos *Users* que o seguem, uma lista de reprodução das varias streams realizadas e ainda listas com os melhores momentos das streams de uma dada semana.

Cada *Stream* terá ainda um chat associado que apenas será utilizado pelos *Users Premium*. Este *Chat* será constituído por *Mensagens* que serão guardadas no sitema.

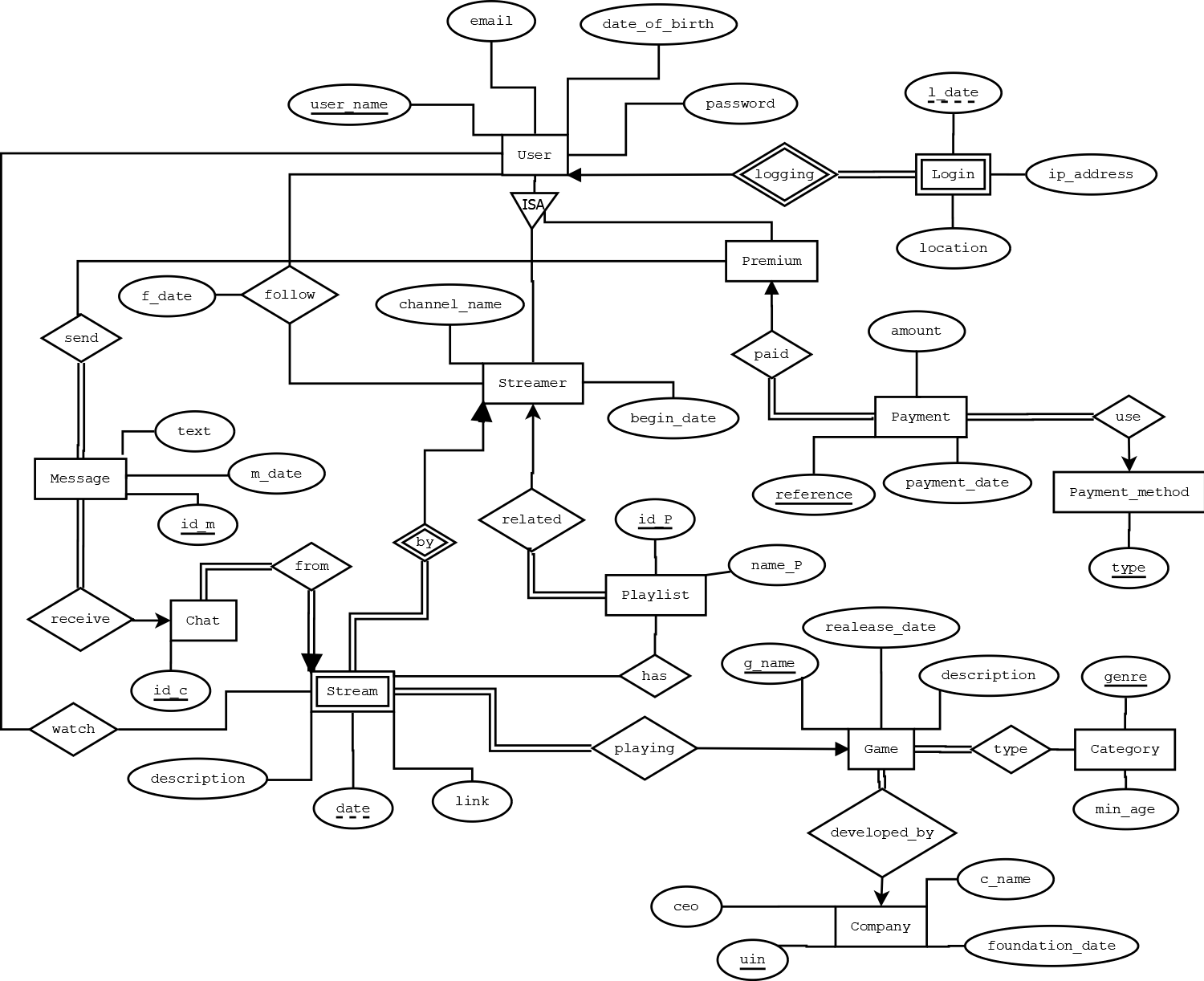
A aplicação guarda informação de cada *Game*. Estes são identificados univocamente pelo nome, a *Company* desenvolvedora, as *Categories*ao qual está associado, uma descrição e a data de lançamento.

Para cada *Company*, é guardado o seu nome, data de fundação e CEO. Adicionalmente temos também associado à *Company* o UIN (Unique Identifier Number) que diferencia estas entidades.

Assume-se que, num determinado momento, a transmissão efetuada por um *Streamer* é composta por apenas um *Game*, isto é, quando um *Streamer* muda de *Game*, é terminada a transmissão anterior e inicia-se uma nova.

# Modelo de Dados

## Modelo ER



## Opções de design

A identidade abstrata foi criada pois as identidades Streamer e Canal são inseparáveis no que diz respeito ao Stream, ou seja, não faz sentido existir um Stream sem Canal e/ou Streamer.

Colocámos o Stream com identidade fraca pois pode existir Streams no mesmo dia/hora, mas tem de ser efectuado por diferentes Streamers e Canais.

Em relação ao Canal cada um possui um chat que pode ser utilizado por Utilizadores Premium em que este recebe mensagens dos Premium.

Em relação ao login tivemos de optar por uma identidade fraca porque pode haver logins à mesma data e hora desde que fossem efetuados por utilizadores diferentes e também porque nos permite manter um registo de todos os Logins feitos no sistema.

O tipo de pagamento foi definido como uma entidade em vez de um atributo, pois queríamos evitar redundância e também permitir que os tipos de pagamento só pudessem ser os disponíveis na aplicação.

## Esquema Relacional

### Versão 1

**Entidades**

User(user\_name ,email, date\_of\_birth, password)

Channel(channel\_id, num\_followers)

Streamer(user\_name, begin\_date ,channel\_id)

Login(l\_date,l\_time,ip\_adress,location,user\_name)

Payment(reference,data,amount,type,user\_name)

Payment\_method(type)

Stream(channel\_id, user\_name, date, time, views, desc, link, g\_name)

Chat(id\_c, channel\_id)

Message(id\_m, text, m\_date, m\_time, id\_c, user\_name)

Playlist(id\_p, channel\_id)

Game(g\_name,release\_date,description,uin)

Category(genre, min\_age)

Company(uin,ceo,c\_name,foundation\_date)

Premium(user\_name)

User\_name é chave estrangeira que refere User

**Relações**

Follow(channel\_id,user\_name,f\_date)

type(genre,g\_name)

watch(user\_name,channel\_id, date,time)

has(id\_p,channel\_id)

### Versão 2

Nesta versão foram eliminadas tabelas de modo a existir mais coerência com o tema do trabalho.

**Entidades**

Categories(genre, min\_age)

Company(uin,ceo,c\_name,foundation\_date)

Game(g\_name,release\_date,description,uin)

Users(user\_name ,email, date\_of\_birth, password)

Login(l\_time,ip\_adress,location,user\_name)

Premium(user\_name)

Streamer(channel\_name, user\_name, begin\_date ,channel\_id)

Payment\_method(type)

Payment(reference,payment\_data,amount,type,user\_name)

Playlist(id\_p, name\_p, streamer\_name)

Stream(streamer\_name,s\_time, views, description, link, g\_name)

Chat(id\_c, streamer\_name)

Message(id\_m, text, m\_time, id\_c, user\_name)

**Relações**

Follow(streamer\_name,user\_name,f\_date)

type(genre,g\_name)

watch(user\_name,streamer\_name,s\_time)

has(id\_p,streamer\_name,s\_time)

## Criação da Base de Dados

### Criação das Tabelas

Em seguida encontra-se o código referente á criação das tabelas.

--Entidades--

CREATE TABLE categories(

genre VARCHAR2(15) PRIMARY KEY,

min\_age NUMBER(2,0)

);

CREATE TABLE company(

uin NUMBER(11) PRIMARY KEY,

ceo VARCHAR2(30),

c\_name VARCHAR2(20),

foundation\_date DATE

);

CREATE TABLE game(

g\_name VARCHAR2(30) PRIMARY KEY,

realease\_date DATE,

description VARCHAR2(150),

uin NUMBER(11),

FOREIGN KEY(uin) REFERENCES company(uin)

);

CREATE TABLE users (

user\_name VARCHAR2(15) PRIMARY KEY,

email VARCHAR2(100) UNIQUE NOT NULL ,

date\_of\_birth DATE NOT NULL,

password VARCHAR2(255) NOT NULL,

CHECK ( REGEXP\_LIKE (email, '^[A-Za-z0-9.\_%+-]+@[A-Za-z0-9.-]+.[A-Za-z]{2,4}$'))

);

CREATE TABLE premium (

user\_name VARCHAR2(15) PRIMARY KEY,

FOREIGN KEY (user\_name) REFERENCES users(user\_name)

);

CREATE TABLE login (

l\_time TIMESTAMP,

ip\_address VARCHAR2(15),

location VARCHAR2(15),

user\_name VARCHAR2(15),

PRIMARY KEY (l\_time,user\_name),

FOREIGN KEY (user\_name) REFERENCES users(user\_name),

CHECK ( REGEXP\_LIKE (ip\_address,'^(([0-9]{1}|[0-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\.){3}([0-9]{1}|[0-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])$' ))

);

CREATE TABLE streamer (

user\_name VARCHAR2(15) PRIMARY KEY,

channel\_name VARCHAR2(15) UNIQUE NOT NULL,

begin\_date DATE NOT NULL,

FOREIGN KEY (user\_name) REFERENCES users(user\_name)

);

CREATE TABLE payment\_Method(

type VARCHAR2(15) PRIMARY KEY

);

CREATE TABLE stream (

streamer\_name VARCHAR2(15) NOT NULL,

s\_time TIMESTAMP NOT NULL,

g\_name VARCHAR2(15) NOT NULL,

description VARCHAR2(300),

link VARCHAR2(50) NOT NULL,

PRIMARY KEY(streamer\_name,s\_time),

FOREIGN KEY (streamer\_name) REFERENCES streamer(user\_name),

FOREIGN KEY (g\_name) REFERENCES game(g\_name)

);

CREATE TABLE playlist(

id\_p NUMBER(12,0) PRIMARY KEY,

name\_p VARCHAR2(20) NOT NULL,

streamer\_name VARCHAR2(15) NOT NULL,

FOREIGN KEY (streamer\_name) REFERENCES streamer(user\_name)

);

CREATE TABLE payment(

reference NUMBER(12,0) PRIMARY KEY,

payment\_date DATE NOT NULL,

amount NUMBER(5,2),check(amount > 0),

user\_name VARCHAR2(15),

type VARCHAR2(15),

FOREIGN KEY (user\_name) REFERENCES premium(user\_name),

FOREIGN KEY (type) REFERENCES payment\_Method(type)

);

CREATE TABLE chat(

id\_c NUMBER(12,0) PRIMARY KEY,

streamer\_name VARCHAR2(15) NOT NULL,

FOREIGN KEY (streamer\_name) REFERENCES streamer(user\_name)

);

CREATE TABLE message(

id\_m NUMBER(12,0) PRIMARY KEY,

m\_time TIMESTAMP NOT NULL,

id\_c NUMBER(12,0) NOT NULL,

user\_name VARCHAR2(15) NOT NULL,

text VARCHAR(250),

FOREIGN KEY (id\_c) REFERENCES chat(id\_c),

FOREIGN KEY (user\_name) REFERENCES premium(user\_name)

);

--Relacões--

CREATE TABLE watch(

user\_name VARCHAR2(15) NOT NULL,

streamer\_name VARCHAR2(15) NOT NULL,

s\_time TIMESTAMP NOT NULL,

PRIMARY KEY (user\_name,streamer\_name,s\_time),

FOREIGN KEY (user\_name) REFERENCES users(user\_name),

FOREIGN KEY (streamer\_name,s\_time) REFERENCES stream(streamer\_name,s\_time),

CHECK (user\_name<>streamer\_name)

);

CREATE TABLE has(

id\_p NUMBER(12,0),

streamer\_name VARCHAR2(15),

s\_time TIMESTAMP,

PRIMARY KEY (id\_p,streamer\_name,s\_time),

FOREIGN KEY (id\_p) REFERENCES playlist(id\_p),

FOREIGN KEY (streamer\_name) REFERENCES streamer(user\_name)

);

CREATE TABLE type(

g\_name VARCHAR2(30),

genre VARCHAR2(15),

PRIMARY KEY(g\_name,genre),

FOREIGN KEY (g\_name) references game(g\_name),

FOREIGN KEY (genre) references categories(genre)

);

CREATE TABLE follow (

user\_name VARCHAR2(15),

streamer\_name VARCHAR2(15),

f\_date TIMESTAMP NOT NULL,

PRIMARY KEY (user\_name,streamer\_name),

FOREIGN KEY (user\_name) REFERENCES users(user\_name),

FOREIGN KEY (streamer\_name) REFERENCES streamer(user\_name),

CHECK (user\_name<>streamer\_name)

);

### Criação de Sequências

As seguintes sequências são utilizadas para obter os vários identificadores únicos utilizados como chaves primárias nas tabelas company, payment, playlist, chat e message. Sempre que ocorre uma inserção numa destas tabelas, é acionado um trigger que atribui o próximo id, através da sequência.

DROP SEQUENCE seq\_company;

CREATE SEQUENCE seq\_company

START WITH 1

INCREMENT BY 1;

DROP SEQUENCE seq\_payment;

CREATE SEQUENCE seq\_payment

START WITH 1

INCREMENT BY 1;

DROP SEQUENCE seq\_playlist;

CREATE SEQUENCE seq\_playlist

START WITH 1

INCREMENT BY 1;

DROP SEQUENCE seq\_chat;

CREATE SEQUENCE seq\_chat

START WITH 1

INCREMENT BY 1;

DROP SEQUENCE seq\_message;

CREATE SEQUENCE seq\_message

START WITH 1

INCREMENT BY 1;

### Criação de Triggers

O trigger *insert\_streamer* permite fazer a inserção de um *streamer* na *view\_streamers*.

CREATE OR REPLACE TRIGGER insert\_streamer INSTEAD OF INSERT ON view\_streamers

FOR EACH ROW

BEGIN

INSERT INTO streamer values(:new.user\_name,:new.channel\_name,:new.begin\_date);

END;

O trigger *delete\_streamer* permite remover um *streamer* na *view\_streamers*.

CREATE OR REPLACE TRIGGER delete\_streamer INSTEAD OF DELETE ON view\_streamers

FOR EACH ROW

BEGIN

DELETE FROM streamer where user\_name=:old.user\_name;

END;

O trigger *insert\_streamer* permite dar update a um *streamer* na *view\_streamers*.

CREATE OR REPLACE TRIGGER update\_streamer INSTEAD OF UPDATE on view\_streamers

FOR EACH ROW

BEGIN

UPDATE streamer

SET

channel\_name=:new.channel\_name

where user\_name=:new.user\_name;

END;

O trigger *insert\_premium* permite fazer a inserção de um *utilizador premium* na *view\_premium*.

CREATE OR REPLACE TRIGGER insert\_premium INSTEAD OF INSERT ON view\_premium

FOR EACH ROW

BEGIN

INSERT INTO premium values(:new.user\_name);

END;

O trigger *delete\_premium* permite remover um *utilizador premium* da *view\_premium*.

CREATE OR REPLACE TRIGGER delete\_premium INSTEAD OF DELETE ON view\_premium

FOR EACH ROW

BEGIN

DELETE FROM premium where user\_name=:old.user\_name;

END;

O trigger *canWatch* garante que, sempre que existe uma inseção na tabela *watch*, o *utilizador* tem secção iniciada para poder assistir a essa transmição.

CREATE OR REPLACE TRIGGER canWatch BEFORE INSERT ON watch

FOR EACH ROW

DECLARE co NUMBER;last\_login TIMESTAMP;

BEGIN

SELECT count(\*) into co

FROM login

WHERE user\_name=:new.user\_name

select max(l\_time) into last\_login

from login

where user\_name=:new.user\_name;

if co = 0 or ((TRUNC(:new.s\_time) - TRUNC(last\_login)) >= 1 or (TRUNC(:new.s\_time) - TRUNC(last\_login)) <0)

then Raise\_Application\_Error (-20100, 'User never logged in or login expired');

end if;

END;

O trigger *canFollow* garante que, sempre que existe uma inseção na tabela *follow*, o *utilizador* tem secção iniciada para poder seguir um *streamer*.

CREATE OR REPLACE TRIGGER canFollow BEFORE INSERT ON follow

FOR EACH ROW

DECLARE co NUMBER;last\_login TIMESTAMP;

BEGIN

SELECT count(\*) into co

FROM login

WHERE user\_name=:new.user\_name

select max(l\_time) into last\_login

from login

where user\_name=:new.user\_name;

if co = 0 or ((TRUNC(:new.f\_date) - TRUNC(last\_login)) >= 1 or (TRUNC(:new.f\_date) - TRUNC(last\_login)) <0)

then Raise\_Application\_Error (-20100, 'User never logged in or login expired');

end if;

END;

O trigger *canPay* garante que, sempre que existe uma inseção na tabela *payment*, o *utilizador* tem secção iniciada para poder seguir um pagar a mensalidade de modo a obter o estatuto de utilizador *premium*.

CREATE OR REPLACE TRIGGER canPay BEFORE INSERT ON payment

FOR EACH ROW

DECLARE co NUMBER;last\_login TIMESTAMP;

BEGIN

SELECT count(\*) into co

FROM login

WHERE user\_name=:new.user\_name;

select max(l\_time) into last\_login

from login

where user\_name=:new.user\_name;

if co = 0 or ((TRUNC(:new.payment\_date) - TRUNC(last\_login)) >= 1 or (TRUNC(:new.payment\_date) - TRUNC(last\_login)) <0)

then Raise\_Application\_Error (-20100, 'User never logged in or login expired');

end if;

END;

O trigger *canSend* garante que, sempre que existe uma inseção na tabela *message*, o utilizador tem secção iniciada para poder enviar uma mensagem no *chat* de uma *stream*.

CREATE OR REPLACE TRIGGER canSend BEFORE INSERT ON message

FOR EACH ROW

DECLARE co NUMBER;last\_login TIMESTAMP;

BEGIN

SELECT count(\*) into co

FROM login

WHERE user\_name=:new.user\_name

select max(l\_time) into last\_login

from login

where user\_name=:new.user\_name;

if co = 0 or ((TRUNC(:new.m\_time) - TRUNC(last\_login)) >= 1 or (TRUNC(:new.m\_time) - TRUNC(last\_login)) <0)

then Raise\_Application\_Error (-20100, 'User never logged in or login expired');

end if;

END;

O trigger *atLeastOneCategory,* após a inserção de um novo jogo,cria uma categoria default, caso não exista, para que possa ser atribuída ao jogo.

CREATE OR REPLACE TRIGGER atLeastOneCategory AFTER INSERT ON game

FOR EACH ROW

DECLARE numb NUMBER;

BEGIN

SELECT count(unique genre) INTO numb

FROM categories

WHERE genre = 'DEFAULT';

IF numb = 0

THEN INSERT INTO CATEGORIES VALUES ('DEFAULT','0');

END IF;

INSERT INTO type VALUES (:new.g\_name,'DEFAULT');

END;

O trigger *delDefault* elimina a categoria default de um jogo após a inserção da primeira *categoria*.

CREATE OR REPLACE TRIGGER delDefault BEFORE INSERT ON type

FOR EACH ROW

DECLARE numb NUMBER;

BEGIN

Select count(\*) into numb

from type

where g\_name= :new.g\_name ;

IF numb = 1

then Delete from type where g\_name = :new.g\_name and genre = 'DEFAULT';

END IF;

END;

O trigger *firstPremium* verifica se foi a primeira vez que um utilizador pagou a mensalidade, passando assim a *premium*. Se sim, é inserido na tabela dos utilizadores *premium*.

CREATE OR REPLACE TRIGGER firstPremium BEFORE INSERT ON payment

FOR EACH ROW

DECLARE id NUMBER;

BEGIN

SELECT count(user\_name) INTO id

FROM premium

WHERE user\_name = :new.user\_name;

IF id = 0

THEN INSERT INTO premium VALUES (:new.user\_name);

END IF;

END;

O trigger *checkAge* verifica se a data introduzida pelo utilizador é valida ( se o utilizador tem no mínimo 13 e no máximo 125, sendo assim uma data plausível)

CREATE OR REPLACE TRIGGER checkAge AFTER INSERT ON users

FOR EACH ROW

BEGIN

IF ((floor(months\_between(SYSDATE, :new.date\_of\_birth) /12)) NOT BETWEEN 13 AND 125)

THEN Raise\_Application\_Error (-20100, 'Error: Please Check birthdate ');

END IF;

END;

O trigger *createChat* atribui um chat a cada novo *streamer* introduzido no sistema.

CREATE OR REPLACE TRIGGER createChat AFTER INSERT ON Streamer

FOR EACH ROW

BEGIN

Insert into chat values(seq\_chat.nextval, :new.user\_name);

END;

7

### Criação de funções

A função *calc\_subs* permite obter o número de *followers* que um *streamer* tem.

create or replace FUNCTION calc\_subs(streamer VARCHAR2)

RETURN NUMBER IS totalsubs NUMBER;

BEGIN

SELECT count(user\_name) INTO totalsubs

FROM follow f

WHERE f.streamer\_name = streamer;

RETURN totalsubs;

END calc\_subs;

/

## Limitações/opções tomadas na implementação da Base de Dados

## Descrição da Interface

# Manual do Utilizador

# Considerações finais