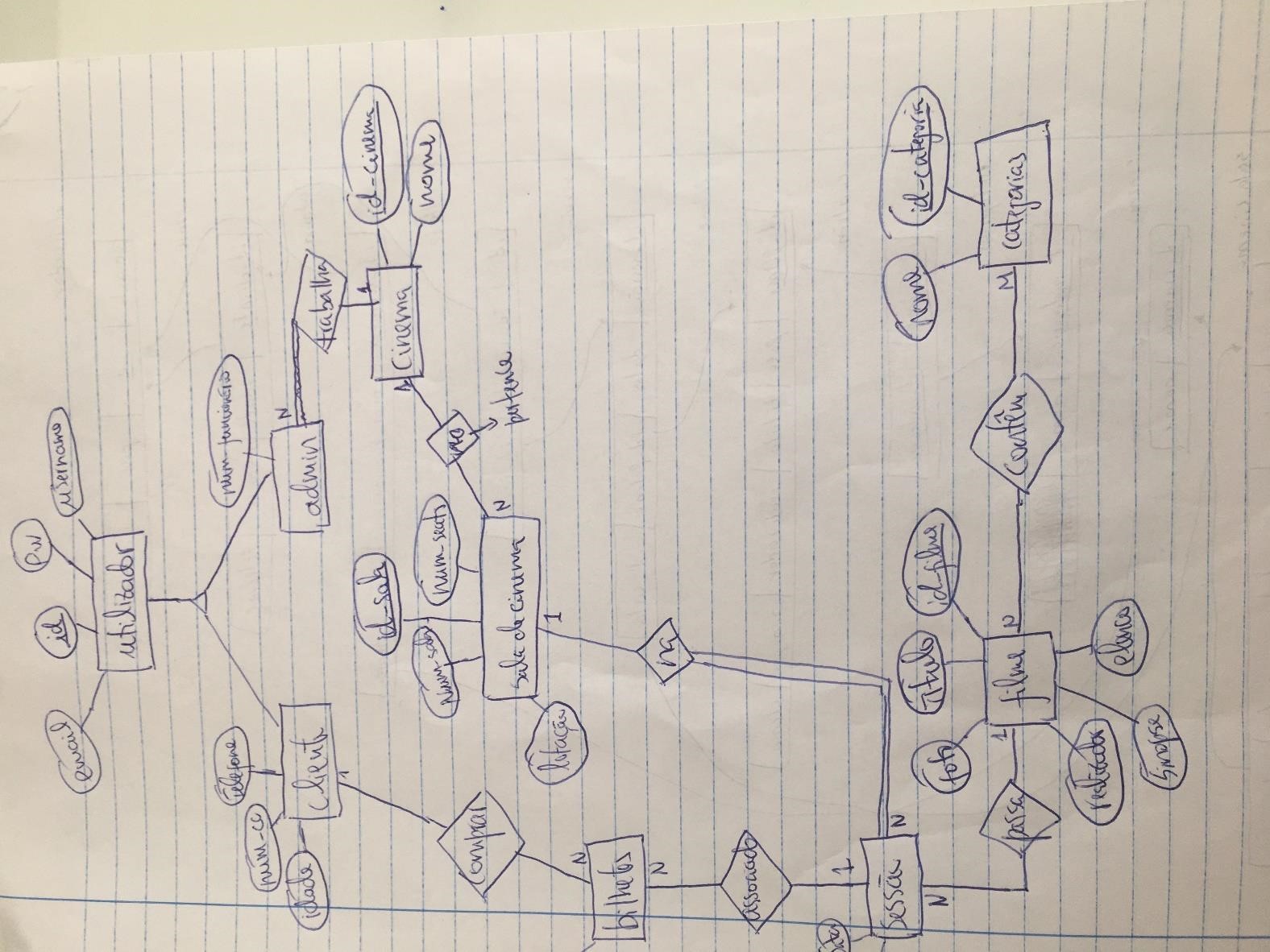
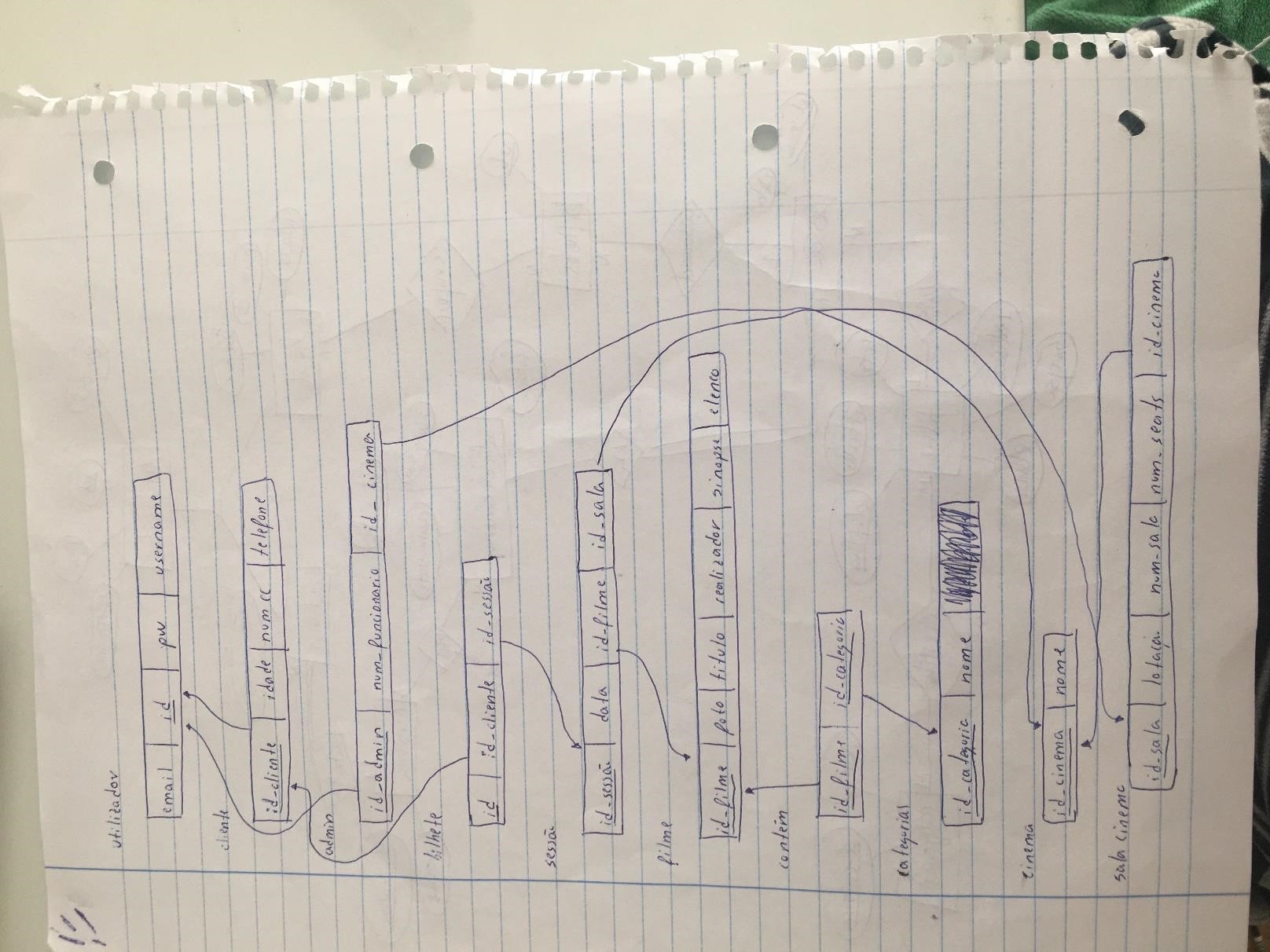
Base de Dados Grupo P1G9

Diogo Torrinhas nª98440, João Torrinhas nº98435

**Diagrama entidade relacionamento**



**Esquema Relacional**



**REQUESITOS FUNCIONAIS**:

-O sistema permite aos utilizadores criar uma conta (não implementado)

-O sistema permite o login dos utilizadores

-O admin deve conseguir fazer a gestão da sala de cinema

-O sistema deve permitir ao admin registar filmes

-O sistema deve deixar os utilizadores(clientes) pesquisar filmes

-O admin deve conseguir criar sessões

-Os clientes podem comprar bilhetes (não implementado)

-Os clientes podem ver sessões

-O cliente consegue ver estatísticas acerca dos bilhetes mais vendidos para cada sessão

**REQUESITOS NÃO FUNCIONAIS**:

-O Sistema deverá permitir até um máximo de 100 palavras na sinopse de um filme

-Filme devera ter no mínimo 1 categoria

**SQL SCRIPTS**

Todo o código desenvolvido para o projeto encontra-se na pasta SQL Scripts

**Setup**

O ficheiro contém o código SQL que permite a criação das tabelas do projeto, descritas anteriormente.

**Nota:** Nós inicialmente definimos que o tipo do atributo pw era NVARCHAR mas o correto deveria ser VARBINARY(128) e quando fosse-mos inserir valores seria, por exemplo, INSERT INTO PROJ\_CINEMAS.UTILZADOR VALUES (4, ‘rogerio@gmail.com’, EncryptByPassPhrase(‘ThePassphrase’,’r12345’),’rogerio10’) e quando fosse para fazer o login tínhamos de mudar a udf **getUserByUsernamePw** para retornar (SELECT id, email, DecryptByPassPhrase(’ThePassphrase’, pw) , username FROM PROJ\_CINEMAS.UTILIZADOR WHERE username=@username AND pw =@pw), por exemplo. Mas devido à falta de tempo não conseguimos implementar.

**Drop**

O ficheiro contém o código SQL que permite a inserção de dados nas tabelas do projeto, descritas anteriormente

**Inserts**

O ficheiro contém o código SQL que permite a inserção de dados nas tabelas do projeto, descritas anteriormente

**Views**

O ficheiro views.sql contém o código que permite a criação de views do projeto.

Foram definidas 3 views, PROJ\_CINEMAS.V\_FILMES, PROJ\_CINEMAS.V\_SESSOESFILMES, PROJ\_CINEMAS.V\_BILHETESESSAO, para juntar atributos de várias tabelas, por exemplo a PROJ\_CINEMAS.V\_SESSOESFILMES foi criada para sabermos o número de sessões disponíveis para cada filme

**Triggers**

O código SQL que contem os triggers do projeto encontra-se no ficheiro triggers.PROG\_CINEMAS.

Os triggers desenvolvidos foram:

**delete\_filme:**

Instead of delete

Se um filme for apagado, também são apagadas as sessões com esse filme

**delete\_sessao:**

Instead of delete

Se uma sessão for apagada, são apagados os bilhetes para essa sessão

**delete\_salas:**

Instead of delete

Se uma sala for apagada, então apaga as sessões com essa sala

**Stored Procedure’s**

O código SQL que contem os stored procedures do projeto encontra-se no ficheiro storedProcedures\_Cinemas.

Os stored procedures do projeto foram criados com a intenção de que as operações de alteração da base de dados (adicionar/alterar/remover tabelas) nunca deveriam ser executadas diretamente sobre a mesma. Em vez disso chama-se o procedure o que faz com que as ações executadas na interface não ponham em risco a integridade do conteúdo da base de dados.

Em alguns procedures, usámos “transaction” de maneira a controlar as transações nas tabelas para garantir a integridade dos dados e lidar com erros na base de dados.

- **create\_sessao** -> Cria uma sessão nova. Recebe como parâmetros o id da sessão, a data da mesma, o id do filme e o id da sala onde vai ser passada a sessão. Implementámos sem “transaction” porque dentro do procedure é efetuado apenas um insert na tabela.

- **create\_filme** -> Cria um filme novo. Recebe como parâmetros o id do filme, a foto, o título, o realizador, a sinopse e o elenco. Implementámos sem “transaction” porque dentro do procedure é efetuado apenas um insert na tabela.

- **create\_sala** -> Cria uma sala nova. Recebe como parâmetros o id da sala, a lotação da mesma, o número da sala, o número de lugares da sala e o id do cinema onde a sala pertence. Implementámos sem “transaction” porque dentro do procedure é efetuado apenas um insert na tabela.

- **delete\_sessao\_byId** -> Apaga uma sessão. Recebe como parâmetro o id da sessão a eliminar. Implementámos com transaction porque ao apagar a sessão vai apagar os bilhetes associados a essa sessão.

- **delete\_filme\_byId** -> Apaga um filme. Recebe como parâmetro o id do filme a eliminar. Implementámos com transaction porque ao apagar o filme vai apagar as sessões associadas a esse filme, os bilhetes associados às sessões que vão ser eliminadas e os tuplos da tabela “contem” com o id do filme que foi apagado.

- **delete\_sala\_byId** -> Apaga uma sala. Recebe como parâmetro o id da sala a eliminar. Implementámos com transaction porque ao apagar a sala vai apagar as sessões associadas a essa sala e os bilhetes associados à sessão que foi eliminada.

- **alter\_sessao** -> Alterar um sessão. Recebe como parâmetros todos os atributos associados a uma sessão. Implementámos com transaction.

- **alter\_sala** -> Alterar uma sala. Recebe como parâmetros todos os atributos associados a uma sala. Implementámos com transaction.

- **alter\_filme** -> Alterar um filme. Recebe como parâmetros todos os atributos associados a um filme. Implementámos com transaction.

**UDF’s**

O código SQL que contem os udf’s do projeto encontra-se no ficheiro udfs\_Cinemas.

Tal como no caso anterior, foram criadas funções de consulta para que a interação com a base de dados não seja feita através de querys diretos, mas sim através de chamadas a funções.

- **getAdminByNumFuncionario** -> Dado um número de funcionário devolve um admin caso exista.

- **getUserByUsernamePw** -> Dado um username e uma pw devolve um user caso exista.

- **getSalaByIdCinema** -> Dado um id do cinema devolve uma sala caso exista.

- **getSessaoByIdSala** -> Dado um id da sala devolve uma sessão caso exista

- **getSessaoByIdFilme** -> Dado um id do filme retorna a sessão para esse filme caso exista

- **id\_sessaoExists** -> Dado um id ver se o mesmo já existe na tabela sessões

- **id\_filmeExists** -> Dado um id ver se o mesmo já existe na tabela filmes

- **id\_salaExists** -> Dado um id ver se o mesmo já existe na tabela das salas de cinema

**Indexes**

O código SQL que contem os index’s do projeto encontra-se no ficheiro indexs\_Cinemas.

- Index na tabela SALACINEMA, coluna id\_cinemaSALACINEMA

- Index na tabela SESSAO, coluna id\_salaSESSAO

- Index na tabela SESSAO, coluna id\_filmeSESSAO

Para a escolha dos indexes, tivemos em conta as pesquisas que são feitas mais frequentemente, que consideramos mais importantes para a interface e se os atributos permitem valores repetidos. Por outro lado, consideramos que os index’s não seriam necessários nem relevantes (neste caso) uma vez que as tabelas têm poucos registos/tuplos.

**Interface**

Para a interface do projeto decidimos usar um conjunto de tabela que consideramos importantes. As tabelas escolhidas foram a ADMINISTRADOR, UTILIZADOR, SESSAO, FILME, SALACINEMA. De salientar que as estatísticas, na interface do cliente, (quanto ao número de bilhetes vendidos em cada sessão) não se vai alterar (a não ser que uma sessão ou um filme seja removido) porque na interface não mexemos na tabela BILHETE. Para o número de bilhetes, para cada sessão, ser alterado tinha de se mexer manualmente, no sql, na tabela BILHETE.