**ASSIGNMENT SQL SOURCE CODE**

This document contains the SQL source code required to create and populate a relational database with the data included in the assignment given to us.

SQL Server might display the ‘ and ’ characters as invalid when pasting this text, although they are correct. To solve the issue, just type them again. Additionally, when pasting the code for the “check\_if\_is\_actor” and “check\_if\_is\_director\_or\_other” functions, the variable starting with the @ symbol may not be accepted inside the “if” statement. If so, delete the @ symbol and start writing the variable’s name again, so that the auto-fill menu pops up. When it does, select the variable from that menu.

Everything below the dotted line is my source code.

---------------------------------------------------------------------------------------------------------------------

--General comments:

--A Primary Key is unique by default, without the need to define it when assigning one.

--Composite PKs are unique as a whole (their parts do not have to be unique as well).

--To create the database, press the button “New Query” from the command menu and make sure that “master” is chosen in the “Available databases” combo box, right to the left and below the “New Query” button.

create database AimoDB;

create table FUNDING (

GrantName varchar (300) not null primary key,

FundingBody varchar (200) not null,

MaxAmountPerCompany decimal(10,2) not null check (MaxAmountPerCompany > 0),

DeadlineOfRequests date not null

);

--This table lists every grant by its name.

--Grant names and funding bodies’ names can be very long and that is why those fields

--can take up to 300 characters.

--The GrantName field has been chosen as the PK because even if two different

--authorities offered a grant for the same reasons, the grants would still not have the

--same name.

--The date entered in the DeadlineOfRequests field is not checked, to allow the input of

--old entries.

create table NAME\_REGISTRATION\_AUTHORITY (

NameRegistrationActNo int not null primary key,

AuthorityName varchar (100) not null,

RegistrationDate date not null,

RegistrationFee int not null check (RegistrationFee > 0),

FeePaymentCurrency varchar (25) not null

);

--This table lists the acts of every name registration authority by act number.

--The table’s primary key is the act’s number and not the authority’s name, because an

--authority could have more than one movie company’s name registered, if there existed

--more than one movie companies in the territory over which the authority’s jurisdiction

--lies.

--The AuthorityName field allows 100 characters, to make sure every name is

--accommodated.

--The date entered in the RegistrationDate field is not checked, to allow the input of old

--entries.

create table MOVIE\_PRODUCTION\_COMPANY (

CompanyName varchar (100) not null primary key,

NameRegistrationActNo int not null foreign key references NAME\_REGISTRATION\_AUTHORITY (NameRegistrationActNo),

CompanyType varchar (50) not null,

TotalNumberOfEmployees int not null check (TotalNumberOfEmployees > 0),

Assets decimal (11,2) not null check (Assets > 0),

Liabilities decimal (9,2) not null check (Liabilities > 0),

IsDefunct char (3) not null

);

alter table MOVIE\_PRODUCTION\_COMPANY

add NetWorth as (Assets - Liabilities) persisted;

--The CompanyName field accepts up to 100 characters, to make sure every company’s

--name can be inserted.

--The NameRegistrationActNo field is not unique because two companies may have

--had their names registered in the same act. The table’s PK is the company’s name,

--which prevents the same company from being registered twice.

--The NetWorth column is automatically updated each time the Assets or Liabilities

--fields are updated.

create table FUNDING\_REQUEST (

RequestApplicationID varchar (20) not null primary key,

RequestedGrantName varchar (300) not null foreign key references FUNDING(GrantName),

ApplicantCompanyName varchar (100) not null foreign key references MOVIE\_PRODUCTION\_COMPANY(CompanyName),

ApplicationDate date not null,

AmountRequested decimal (10,2) not null check (AmountRequested > 0),

RequestOutcome varchar (300)

);

--This table lists the applications for a grant that the companies have made.

--The RequestedGrantName field is not unique because the same grant can be shared/

--requested by more than one companies and because a company can request a grant

--a second time, if its application was rejected in the first place.

--The ApplicantCompanyName field is not unique because each production company

--may have received/requested more than one grants.

--The date entered in the ApplicationDate field is not checked, to allow the input of old

--entries.

--The RequestOutcome field accepts 300 characters and is not checked, to allow the

--input of the reasons a request was rejected, if needed. Moreover, it is nullable

--because the request’s outcome may not be known when the entry is inserted into the

--database.

create table SHAREHOLDER (

ShareholderPassportNo char (9) not null primary key,

ShareholderNationalInsuranceNo varchar (20) not null unique,

ShareholderTelephoneNo varchar (20) not null unique,

ShareholderFirstName varchar (70),

ShareholderLarstName varchar (70),

ShareholderFatherFirstName varchar (70) not null,

ShareholderMotherMaidenName varchar (70) not null,

ShareHolderTownOfBirth varchar (50) not null

);

--This table stores the shareholders’ personal information.

--The ShareholderPassportNo field accepts 9 characters and uses the char instead of

--the varchar data type because all passports, globally, have a 9-character serial

--number.

--The ShareholderTelephoneNo field is a varchar data type because the shareholders

--might not possess a phone number from the country the company is registered in,

--meaning that the “+” symbol might have to be inserted in the field.

--The ShareholderFirstName and ShareholderLastName fields are nullable because

--they are not mandatory.

create table HAS\_INVESTED (

ShareholderPassportNo char (9) not null foreign key references SHAREHOLDER(ShareholderPassportNo),

CompanyName varchar (100) not null foreign key references MOVIE\_PRODUCTION\_COMPANY(CompanyName),

primary key (ShareholderPassportNo, CompanyName),

IsStillAShareholder char (3) not null

);

--This table matches the shareholders’ info with the info of the company they have

--invested in.

--The ShareholderPassportNo field is not unique because a shareholder may have

--invested in more than one movie companies.

--The CompanyName field is not unique because a company may have many

--shareholders.

create table ADDRESS (

BuildingName varchar (20) not null primary key,

CompanyName varchar (100) not null foreign key references MOVIE\_PRODUCTION\_COMPANY(CompanyName),

Street varchar (20),

Number varchar (15),

ZIPCode varchar (15) not null,

City varchar (50) not null,

Country varchar (30) not null,

IsCurrentBuildingAddress char (3) not null

);

--This table lists the address of every building a company might consist of.

--The Street and Number fields are nullable because there are regions (e.g. small

--villages) that have no street names.

--The Number field is of varchar data type because addresses can contain the “-“

--symbol or letters (e.g. 221B Baker st.).

create table MOVIE (

MovieInternationalSerialNo varchar (50) not null primary key,

CompanyName varchar (100) not null foreign key references MOVIE\_PRODUCTION\_COMPANY(CompanyName),

MovieTitle varchar (100) not null,

PremiereFullDate date not null

);

--This tables lists each movie made by a company.

--The CompanyName field is not unique because movie production companies (almost)

--always make more than one movie.

create table EMPLOYEE (

EmployeeID varchar (15) not null primary key,

CompanyName varchar (100) not null foreign key references MOVIE\_PRODUCTION\_COMPANY(CompanyName),

EmployeeFirstName varchar (70) not null,

EmployeeLastName varchar (70) not null,

EmployeeMiddleName char (2),

EmployeeEmailAddress varchar (50) not null unique,

EmployeeDateOfBirth date not null,

EmployeeContractStartDate date not null,

PaymentCurrency varchar (25) not null,

IsStillEmployed char (3) not null

);

--This table lists all employees’ basic personal information.

create table EMPLOYEE\_PHONE (

EmployeePhoneNumber varchar (20) not null primary key,

PhoneUsage varchar (15) not null

);

--This table stores info about the location/usage of the employees’ phone numbers.

create table OWNS (

EmployeePhoneNumber varchar (20) not null foreign key references EMPLOYEE\_PHONE(EmployeePhoneNumber),

EmployeeID varchar (15) not null foreign key references EMPLOYEE(EmployeeID),

Primary key (EmployeePhoneNumber, EmployeeID),

IsInUse char (3) not null

);

--This table stores information about where each employee’s phone is located and in

--what way it is being used (e.g. landline, work etc.)

--Both fields are not unique because there could exist employees that live together and

--share the same landline or offices with many employees but one phone device. Also,

--because one employee can have more than 1 phone numbers.

create function check\_if\_ID\_is\_staff (@EmployeeID varchar(15))

returns int as

begin

if exists (select StaffEmployeeID from STAFF where StaffEmployeeID = @EmployeeID)

return 1

return 0

end;

create function check\_if\_ID\_is\_crew (@EmployeeID varchar(15))

returns int as

begin

if exists (select CrewEmployeeID from CREW where CrewEmployeeID = @EmployeeID)

return 1

return 0

end;

--The above two functions check if the inputted employee ID number has already been

--inserted in the database as a staff or crew member respectively and they will be used

--as a check constraint, to ensure that an employee can only be inserted as a crew or

--as a staff member and not as both.

create table CREW (

CrewEmployeeID varchar (15) not null unique foreign key references EMPLOYEE(EmployeeID),

CrewEmployeeOccupation varchar (50) not null check (CrewEmployeeOccupation = 'Actor' or CrewEmployeeOccupation = 'Director' or CrewEmployeeOccupation like 'Other%'),

HourlyPay decimal (8,2) not null check (HourlyPay > 0),

primary key (CrewEmployeeID)

);

--This table lists some of the personal info of the employees that are crew and not staff

--members.

--To accept an entry in the CrewEmployeeOccupation field, the program checks that it

--starts with the word “Actor” or “Director” or a string starting with “Other” (e.g. “Other –

--Producer”).

create function check\_if\_is\_actor (@ActorEmployeeID varchar(15))

returns int as

begin

if ((select CrewEmployeeOccupation from CREW where CREW.CrewEmployeeID=@ActorEmployeeID) = ‘Actor’)

return 1

return 0

end;

create function check\_if\_is\_director\_or\_other (@ActorEmployeeID varchar(15))

returns int as

begin

if ((select CrewEmployeeOccupation from CREW where CREW.CrewEmployeeID=@ActorEmployeeID) = ‘Director’ or (select CrewEmployeeOccupation from CREW where CREW.CrewEmployeeID=@ActorEmployeeID) like 'Other%')

return 1

return 0

end;

--The above two functions check if the inputted employee ID has been inserted in the

--database as an actor or as a director/other personnel respectively, to prevent entering

--an actor in the DIRECTOR\_OR\_OTHER table and vice versa.

create table ACTOR (

ActorEmployeeID varchar (15) not null unique foreign key references CREW(CrewEmployeeID),

BonusPerWorkDay decimal (10,2) not null check (BonusPerWorkDay > 0),

BonusPerSceneCompleted decimal (10,2) not null check (BonusPerSceneCompleted > 0),

check (dbo.check\_if\_is\_actor (ActorEmployeeID) = 1),

primary key (ActorEmployeeID)

);

--Before adding the entry, the employee’s ID is checked, to confirm it belongs to a

--crew member that is an actor.

create table DIRECTOR\_OR\_OTHER (

DirectorOrOtherEmployeeID varchar (15) not null unique foreign key references CREW(CrewEmployeeID),

EndOfShootingBonus decimal (10,2) not null check (EndOfShootingBonus > 0),

check (dbo.check\_if\_is\_director\_or\_other (DirectorOrOtherEmployeeID) = 1),

primary key (DirectorOrOtherEmployeeID)

);

--Before adding the entry, the employee’s ID is checked, to confirm it belongs to a

--crew member that is a director or has another job (entered in the database as “Other”).

create table INVOLVED (

CrewEmployeeID varchar (15) not null foreign key references CREW(CrewEmployeeID),

MovieInternationalSerialNo varchar (50) not null foreign key references MOVIE (MovieInternationalSerialNo),

MovieRoleDescription varchar (60) not null,

primary key (CrewEmployeeID, MovieInternationalSerialNo, MovieRoleDescription)

);

--This table contains info about the members of the crew that are in some way involved

--in the making of a movie, be they actors, directors or something else.

--All fields are not unique, to allow inputting crew members that have taken part in more

--than one movies or.

create table DEPARTMENT (

DepartmentID int not null primary key,

CompanyName varchar (100) not null foreign key references MOVIE\_PRODUCTION\_COMPANY(CompanyName),

DepartmentName varchar (40) not null,

IsCurrentlyManned char (3) not null

);

create table STAFF (

StaffEmployeeID varchar (15) not null unique foreign key references EMPLOYEE(EmployeeID),

DepartmentID int not null foreign key references DEPARTMENT(DepartmentID),

StaffContractSerialNo varchar (15),

StaffMonthlySalary decimal (9,2) not null check (StaffMonthlySalary > 0),

StaffWorkingHours varchar (30) not null,

primary key (StaffEmployeeID),

check (dbo.check\_if\_ID\_is\_crew (StaffEmployeeID) = 0)

);

--The StaffWorkingHours field in the STAFF table accepts varchar-type data, to allow

--the input of both numbers (e.g. 0700-1500) and letters (e.g. full-time).

--The inputted employee’s ID is accepted only if it is not included in the CREW table

--(check constraint).

--The table’s PK is the company’s name, which prevents the same staff member from

--being entered in two different departments.

alter table CREW

add check (dbo.check\_if\_ID\_is\_staff (CrewEmployeeID) = 0);

--Alters the CREW table, to make it accept an ID only after checking that it has not

--already been entered as a staff member.

create table HOUSES (

BuildingName varchar (20) not null foreign key references ADDRESS(BuildingName),

DepartmentID int not null foreign key references DEPARTMENT (DepartmentID),

primary key (BuildingName, DepartmentID),

IsCurrentDepartmentAddress char (3) not null

);

--This table contains info about the address each company’s department is located in.

--The fields are not unique because a department might span more than one buildings

--and a building may house more than one departments.

insert into FUNDING

values (‘Movie props grant’, ‘European Movie Helpers’, 5000, ‘20140425’);

insert into FUNDING

values (‘Economic boost for animated movies visual FX’, ‘Peter Ustinov Org.’, 10145.25, ‘20220810’);

insert into NAME\_REGISTRATION\_AUTHORITY

values (1544, ‘Derby Regional Trademark Authority’, ‘20110515’, 14322, ‘Euro’);

insert into NAME\_REGISTRATION\_AUTHORITY

values (0935, ‘Companies House’, ‘20150131’, 10000, ‘Pound sterling’);

insert into MOVIE\_PRODUCTION\_COMPANY

values (‘Atlas Studios’, 1544, ‘LLC’, 674, 10000, 6500.66, ‘No’);

insert into MOVIE\_PRODUCTION\_COMPANY

values (‘Bravo Productions’, 0935, ‘Private Ltd’, 301, 4732.1, 5574, ‘No’);

insert into FUNDING\_REQUEST

values (‘AtlasProps1’, ‘Movie props grant’, ‘Atlas Studios’, ‘20140209’, 3000, ‘Approved’);

insert into FUNDING\_REQUEST

values (‘AtlasFX1’, ‘Economic boost for animated movies visual FX’, ‘Atlas Studios’, ‘20210401’, 8000.3, null);

insert into FUNDING\_REQUEST

values (‘BravoFX1’, ‘Economic boost for animated movies visual FX’, ‘Bravo Productions’, ‘20210827’, 10000, ‘Approved’);

insert into SHAREHOLDER

values (‘AF1485550’, ‘1256FFG12’, ‘+49 783 00444’, ‘Hans’, ‘Schnitzel’, ‘Dieter’, ‘Wurst’, ‘Hamburger’);

insert into SHAREHOLDER

values (‘ED1460058’, ‘GLH44BB91’, ‘001 954 23 712’, ‘Kevin’, ‘Bacon’, ‘Edmund’, ‘Holmes’, ‘Philadelphia’);

insert into HAS\_INVESTED

values (‘AF1485550’, ‘Atlas Studios’, ‘Yes’);

insert into HAS\_INVESTED

values (‘ED1460058’, ‘Atlas Studios’, ‘Yes’);

insert into ADDRESS

values (‘AtlasMain’, ‘Atlas Studios’, ‘Mahogany’, ‘12C’, ‘DE1 2AW’, ‘Derby’, ‘England’, ‘Yes’);

insert into ADDRESS

values (‘Atlas2’, ‘Atlas Studios’, ‘Ebony’, ‘64’, ‘44885’, ‘Lille’, ‘France’, ‘Yes’);

insert into MOVIE

values (‘AA8804’, ‘Bravo Productions’, ‘The Trip 3’, ‘20180620’);

insert into MOVIE

values (‘UW794’, ‘Bravo Productions’, ‘Last Chance’, ‘20210403’);

insert into EMPLOYEE

values (‘333’, ‘Atlas Studios’, ‘Samuel’, ‘Jackson’, ‘L.’, ‘sammyjack@hotmail.com’, ‘19481221’, ‘20191120’, ‘US Dollar’, ‘Yes’);

insert into EMPLOYEE

values (‘444’, ‘Atlas Studios’, ‘Leslie’, ‘Nielsen’, null, ‘shirley@gmail.co.uk’, ‘19260211’, ‘20110907’, ‘US Dollar’, ‘Yes’);

insert into EMPLOYEE

values (‘555’, ‘Bravo Productions’, ‘Donald’, ‘Donaldson’, null, ‘ddonaldson@yahoo.com’, ‘19550427’, ‘20201010’, ‘Euro’, ‘Yes’);

insert into EMPLOYEE

values (‘666’, ‘Bravo Productions’, ‘Gabriel’, ‘Byrne’, null, ‘gabriel@gmail.com’, ‘19500512’, ‘20190319’, ‘Euro’, ‘Yes’);

insert into EMPLOYEE\_PHONE

values (‘748362’, ‘Landline’);

insert into EMPLOYEE\_PHONE

values (’55 11 1047’, ‘Mobile phone’);

insert into OWNS

values (‘748362’, ‘444’, ‘Yes’);

insert into OWNS

values (‘55 11 1047’, ‘444’, ‘Yes’);

insert into CREW

values (‘333’, ‘Actor’, 500);

insert into CREW

values (‘444’, ‘Director’, 880.46);

insert into CREW

values (‘666’, ‘Other - Producer’, 1000);

insert into ACTOR

values (‘333’, 300, 1000);

insert into DIRECTOR\_OR\_OTHER

values (‘444’, 5000);

insert into DIRECTOR\_OR\_OTHER

values (‘666’, 10000);

insert into INVOLVED

values (‘333’, ‘AA8804’, ‘Male lead’);

insert into INVOLVED

values (‘333’, ‘UW794’, ‘As himself’);

insert into INVOLVED

values (‘444’, ‘UW794’, ‘Supporting - Alex Cummings’);

insert into INVOLVED

values (‘333’, ‘UW794’, ‘Cameo - Street peddler’);

insert into DEPARTMENT

values (1, ‘Atlas Studios’, ‘Mocap’, ‘Yes’);

insert into DEPARTMENT

values (2, ‘Bravo Productions’, ‘HR’, ‘Yes’);

insert into STAFF

values (‘555’, 2, ‘45364’, 954.3, ‘Part-time (0800-1200)’);

insert into HOUSES

values (‘AtlasMain’, 1, ‘Yes’);

insert into HOUSES

values (‘Atlas2’, 1, ‘Yes’);

--To connect this (and any other) database to a Microsoft Access file, you have to follow

--the following instructions:

--1) Open Microsoft Access and create a new (blank) database, by clicking on the

--pre-selected button in the upper left corner and typing a name for the new file. A table

--will be automatically created which will; not be needed and can be deleted.

--2) Next, click on the “External Data” tab and select “New Data Source”, then “From

--Database” and then “From SQL Server”. In the next window, select “Link to the data source by creating a linked table”.

--3) The next step is to create a new data source, by clicking on the “New” button and

--next “ODBC Driver 17 for SQL Server”. You, then, select a name for your data source,

--Press “Next” and then “Finish” and a new Data Source file is created.

--4) The following windows require information about the SQL Server database that

--we want to establish a connection with. Define a name for it, in the text box next to the

--“Decription” text label. In the “Server” text box, write the name of the computer in

--which SQL Server is installed. You can find it by searching for “This PC” from the Start

--menu, right clicking “This PC” and selecting “Properties”.

--5) The window now displayed will assist in logging in the SQL Server. Choose “With

--Integrated Windows Authentication” and press “Next”. After that, check the box that

--says “Change the default database to:” and then select the database you want to

--connect from the combo box (in my case, it is “AimoDB”). Click, “Next”, “Finish”

--and finally “Test Data Source”, to perform some tests. If everything worked correctly,

--press “OK”, to finish this part of the procedure.

--6) For the remaining steps, select your newly-created Data Source file and press

--“OK”. Select the tables you want to link and press “OK”, to finish. Your tables should

--now be visible in Microsoft Access and MUST have a globe to the left. You can keep

--working on the database as if you were using SQL Server.