CSE 581 Final Project Spring 2020

Database for Recruitment Branch of Human Resources

Lichen Liang

lliang11@syr.edu

Abstract

Human Resources is a major branch of any company. It has many sub-branches such as Recruitment, Training and Learning, Labor and Employee Relations, and Organization Development. It is very important to have a correct and efficient database for each sub-branch in order to have a flow of procedures in the company. In this report, I proposed a design for Recruitment branch and tested the database.

Introduction

In this design, I used 21 tables to demonstrate a real-world example of how the recruitment branch should run. I also included 10 candidates who applies for a job position in the company. After inserting all necessary data into the database, I also created views, stored procedures, functions, transactions, triggers, and scripts, and all of them are tested with real-world scenarios.

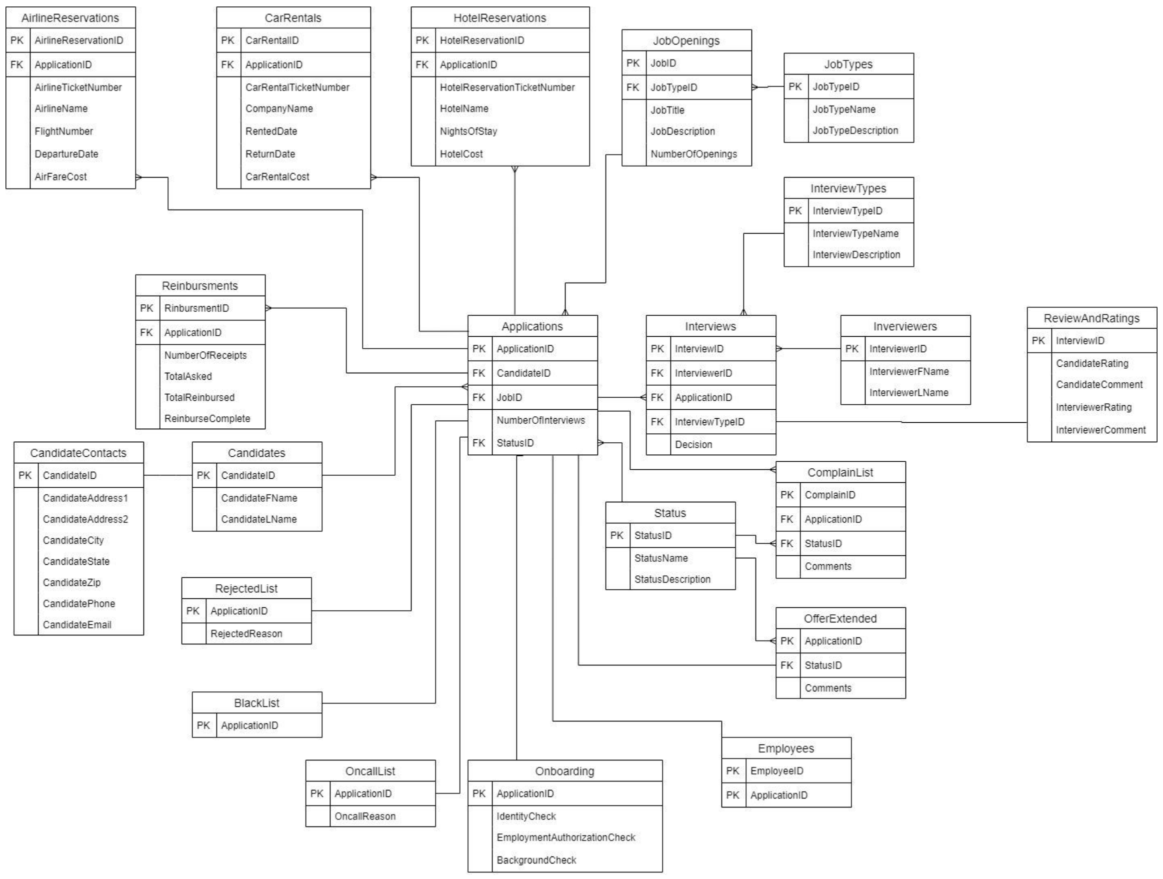


Figure 1. Database Design ER Diagram.

Figure 1 is an ER diagram for the database. The underlined words show how two tables are related to another. The logic as follows:

- Each candidate has a unique ID and contact information (one to one)
- Each candidate can apply to different positions, and apply multiple times (one to many)
- There are three types of jobs, such as summer intern, full time, contract based, and each type can by in multiple job openings (one to many)
- For each job opening, it can be in different applications (one to many)
- There can be two type of interviews: onsite and online; any type can be in any interview session (one to many)
- An interviewer can be in multiple different interviews (one to many)

- The interview table acts as a linking table between the applications, interviewer, interview type. Each interview has a unique record, which also keeps track of the decision made in that interview.
- The application table act as a linking table to candidates, interviews, and job position. This way a candidate can apply to multiple job positions and a job position can be offered to different people (many to many). This table also keeps track of status and number of interviews on this application has occurred.
- The status table stores different status. Therefore, it can be used in any table who need to use it, and any status can be referred to here. (one to many with any other table). A detailed status records is provided in Table 1.
- Interviews can be on-site for any application. The application table is also used to link to other reimbursements to the candidate such as airline reservation, hotel reservation, car rental, and overall reimbursement table (one to many).
- For each application that has been rejected, it is added to a rejected list. (one to one)
- If a candidate wants to complain after an interview, a unique complaint is filed along with the application and the status of the complaint is updated accordingly. Since a candidate can have multiple interviews, then an application can be used for multiple complaints. (one to many)
- For each application that has been accepted (a position is offered), this application is added to the offer extended table. This table keeps track of candidates' decision/status. (one to one)
- If a candidate accepts the offer, but this position is no longer available or for any other valid reason, then this application is moved to on-call list, with a priority of being checked before another interview of the same job position takes place. (one to one)
- If a candidate accepts the offer and the position is still available, this application is moved to on-boarding. This table has three conditions that must be met before a candidate become an official employee. (one to one)
- If all condition in the on-boarding table has been met, the candidate's application is moved to the employee table and the task of this database is done. Note that employees table is not fully constructed but only used as a demonstration of the ending because there is another complex database connected to it as well. (one to one)
- If a candidate completes on-boarding but does not show up, his/her application is moved to blacklist which can be traced back to this candidate. (one to one)
- Last but not the least, for each interview session, there is a rating & review to keep track any feedback between the candidate and the interviewer. (one to one).

Note that the applications have a status that is constantly updated. Therefore, it is important to keep the correct status for each application. The airline, hotel, and car rental tables are just an approximate because they are a tiny part of a whole larger database that's not part of this project

Table 1 shows the Status table, with each code and its corresponding meaning.

Table 1. Status Table

Status ID	Status Name	Description
1	Complain Received	Candidate filed a complain
2	Complain Waiting	Complain under review
3	Complain Accepted	Re-interview
4	Complain Rejected	Candidate Rejected
5	Application Received	Candidate submitted application
6	Application Accepted	Schedule Interview
7	Application Rejected	Candidate Rejected
8	Offer Accepted	Candidate accepts offer
9	Offer Rejected	Candidate rejects offer
10	Offer Negotiating	Negotiation in progress

I used 10 candidates in this design for testing. Many of them are in different scenarios and steps of the process. It is clearer to view each candidate's story in a table format than a paragraph.

Note that only candidate No.3 is on-site interview, with all kinds of reimbursements. All others are conducted online.

Table 2 shows each candidate's situation listed by their ID.

Table 2. Candidate situation table

Candidate ID	Number of Interviews Conducted	Situation/Status
1	0	Applies and immediately rejected
2	0	Applies and immediately rejected
3	2	Accepted in 1st interview and rejected in 2nd interview. Complains, and status complain received.
4	2	Accepted in both interviews, accepts offer, currently onboarding
5	2	Rejected in 1st interview, complains, complain accepted and accepted in 2nd interview. Job offered but currently negotiating
6	2	Accepted in both interviews, offer accepted and completed onboarding, but Blacklisted
7	0	Applies and immediately rejected
8	2	Accepted in both interviews, smooth flow and already official employee
9	1	Accepted in only 1 interview, completed onboarding, but on-call due to position already filled.
10	1	Accepted in 1 interview, but offer rejected by candidate

Tables 3, 4, 5 shows Jobs Available, Job Types, and Interview Types respectively.

Table 3. Jobs Available

Job ID	Job Type ID	Title	Description	Number of Openings
1	1	Entry Level Software Developer	Entry level work, with training	2
2	1	Electrical Engineer	Circuit Design	1
3	2	Database Admin	Manage and maintain database	1

Table 4. Job Types

Job Type ID	Title	Description
1	Summer Internship	Candidate works for two months as an intern
2	Full Time	Candidate works for at least 12 months as full time employee with benefits
3	Contract Based	Candidate signs contract with company, and get paid based on amount of work done

Table 5. Interview Types

Interview Type ID	Title	Description
1	On Site	Candidate must personally meet with interviewer
2	Online	Candidate meet interviewer online through face time

Database, Table, and Data

In this section I have attached the code used for creating the database, tables, and inserting the data. Followed by comments for the code and finally the screenshot verification. Some indentations may be off due to copy and pasting issues.

```
USE master;
GO

/*if database already exist, drop it*/
IF  DB_ID('Recruitments') IS NOT NULL
    DROP DATABASE Recruitments;
GO

CREATE DATABASE Recruitments;
GO

USE Recruitments;


CREATE TABLE Candidates (
  CandidateID       INT          NOT NULL        PRIMARY KEY IDENTITY,
  CandidateFname    VARCHAR(100) NOT NULL,
  CandidateLname    VARCHAR(100) NOT NULL
);
GO

CREATE TABLE CandidateContact (
  CandidateID           INT             NOT NULL,
  CandidateAddress1     VARCHAR(255)    NULL,
  CandidateAddress2     VARCHAR(255)    NULL,
  CandidateCity         VARCHAR(100)    NULL,
  CandidateState        CHAR(2)         NULL,
  CandidateZip          INT             NULL,
  CandidatePhone        VARCHAR(50)     NULL,
  CandidateEmail        VARCHAR(100)    NULL,
  PRIMARY KEY CLUSTERED(CandidateID)
);
GO

CREATE TABLE Interviewers (
  InterviewerID       INT               NOT NULL        PRIMARY KEY IDENTITY,
  InterviewerFname    VARCHAR(100)      NOT NULL,
  InterviewerLname    VARCHAR(100)      NOT NULL
);
GO

CREATE TABLE JobTypes (
  JobTypeID             INT             NOT NULL        PRIMARY KEY IDENTITY,
  JobTypeName           VARCHAR(100)    NOT NULL,
  JobTypeDescription    VARCHAR(300)    NULL,
);
GO

CREATE TABLE Status (
  StatusID              INT             NOT NULL        PRIMARY KEY IDENTITY,
  StatusName            VARCHAR(100)    NOT NULL,
  StatusDescription     VARCHAR(300)    NULL,
);
GO

CREATE TABLE InterviewTypes (
  InterviewTypeID               INT             NOT NULL        PRIMARY KEY IDENTITY,
  InterviewTypeName             VARCHAR(100)    NOT NULL,
  InterviewTypeDescription      VARCHAR(300)    NULL,
);
GO
```

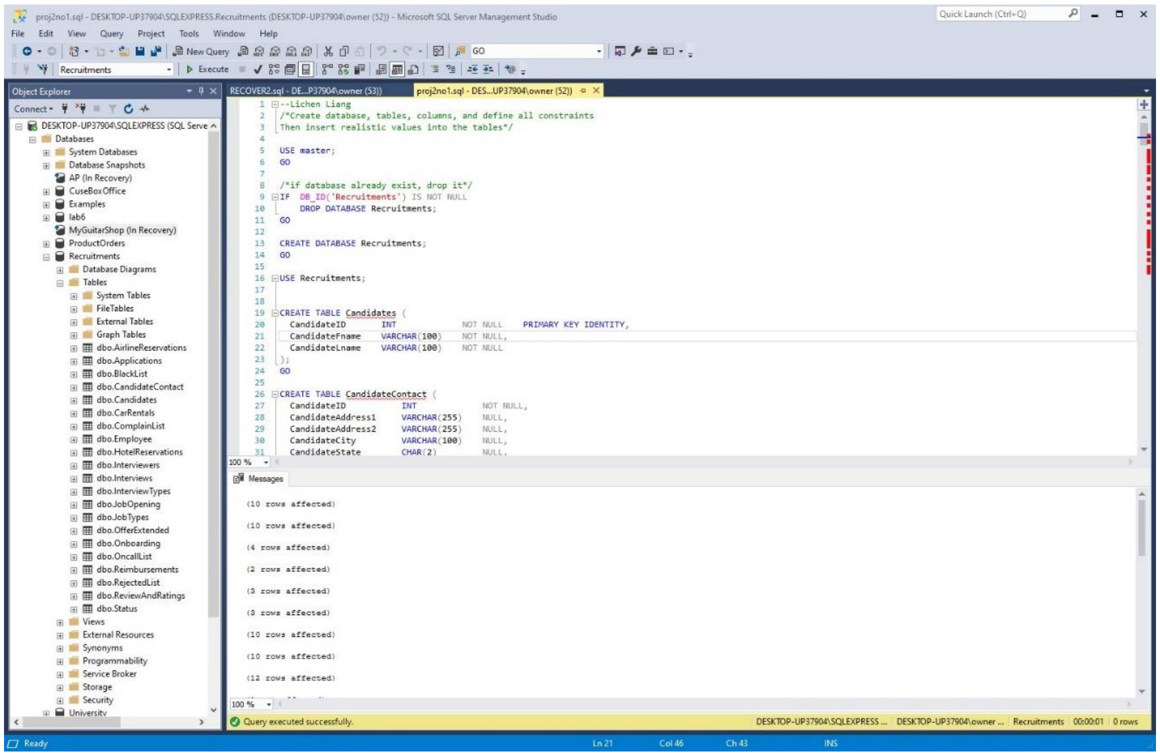
```
CREATE TABLE JobOpening (
  JobID                 INT             NOT NULL        PRIMARY KEY IDENTITY,
  JobTypeID             INT             NOT NULL,
  JobTitle              VARCHAR(100)    NOT NULL,
  JobDescription        VARCHAR(300)    NULL,
  NumberOfOpenings      INT             NOT NULL        DEFAULT 1,
  CONSTRAINT FK_JobOpening_JobTypes FOREIGN KEY(JobTypeID) REFERENCES JobTypes(JobTypeID)
);
GO

CREATE TABLE Applications (
  ApplicationID         INT             NOT NULL        PRIMARY KEY IDENTITY,
  CandidateID           INT             NOT NULL,
  JobID                 INT             NOT NULL,
  NumberOfInterviews    INT             NOT NULL        DEFAULT 0,
  StatusID              INT             NOT NULL,
  CONSTRAINT FK_Applications_Candidates FOREIGN KEY(CandidateID) REFERENCES
Candidates(CandidateID),
  CONSTRAINT FK_Applications_JobOpening FOREIGN KEY(JobID) REFERENCES JobOpening(JobID),
  CONSTRAINT FK_Applications_Status FOREIGN KEY(StatusID) REFERENCES Status(StatusID)
);
GO

CREATE TABLE AirlineReservations (
  AirlineReservationID          INT             NOT NULL        PRIMARY KEY IDENTITY,
  ApplicationID                 INT             NOT NULL,
  AirlineTicketNumber           VARCHAR(50)     NOT NULL,
  AirlineName                   VARCHAR(100)    NOT NULL,
  FlightNumber                  VARCHAR(50)     NOT NULL,
  DepartureDate                 DATETIME        NOT NULL,
  AirFareCost                   MONEY           NOT NULL,
  CONSTRAINT FK_Airline_Application FOREIGN KEY(ApplicationID) REFERENCES
Applications(ApplicationID),
);
GO

CREATE TABLE HotelReservations (
  HotelReservationID            INT             NOT NULL        PRIMARY KEY IDENTITY,
  ApplicationID                 INT             NOT NULL,
  HotelReservationTicketNumber  VARCHAR(50)     NOT NULL,
  HotelName                     VARCHAR(200)    NOT NULL,
  NightsOfStay                  INT             NOT NULL,
  HotelCost                     MONEY           NOT NULL,
  CONSTRAINT FK_Hotel_Application FOREIGN KEY(ApplicationID) REFERENCES
Applications(ApplicationID),
);
GO

CREATE TABLE CarRentals (
  CarRentalID                   INT             NOT NULL        PRIMARY KEY IDENTITY,
  ApplicationID                 INT             NOT NULL,
  CarRentalTicketNumber         VARCHAR(50)     NOT NULL,
  CarRentalCompanyName          VARCHAR(100)    NOT NULL,
  RentedDate                    DATETIME        NOT NULL,
  ReturnedDate                  DATETIME        NULL,
  CarRentalCost                 MONEY           NOT NULL,
  CONSTRAINT FK_CarRental_Application FOREIGN KEY(ApplicationID) REFERENCES
Applications(ApplicationID),
);
GO

CREATE TABLE Reimbursements (
  ReimbursementID               INT             NOT NULL        PRIMARY KEY IDENTITY,
  ApplicationID                 INT             NOT NULL,
  NumberOfReceipts              INT             NOT NULL        DEFAULT 1,
  TotalAsked                    MONEY           NOT NULL,
  TotalReimbursed               MONEY           NOT NULL,
  ReimburseComplete             BIT             NOT NULL,
  CONSTRAINT FK_Reimbursement_Application FOREIGN KEY(ApplicationID) REFERENCES
Applications(ApplicationID),
);
GO

CREATE TABLE Interviews (
  InterviewID                 INT             NOT NULL        PRIMARY KEY IDENTITY,
  InterviewerID               INT             NOT NULL,
  ApplicationID               INT             NOT NULL,
  InterviewTypeID             INT             NOT NULL,
  Decision                    BIT             NOT NULL,
  CONSTRAINT FK_Interview_Interviewer FOREIGN KEY(InterviewerID) REFERENCES
Interviewers(InterviewerID),
  CONSTRAINT FK_Interview_Application FOREIGN KEY(ApplicationID) REFERENCES
Applications(ApplicationID),
  CONSTRAINT FK_Interview_InterviewType FOREIGN KEY(InterviewTypeID) REFERENCES
InterviewTypes(InterviewTypeID)
);
GO

CREATE TABLE RejectedList (
  ApplicationID         INT             NOT NULL,
  RejectedReason        VARCHAR(500)    NULL,
  PRIMARY KEY CLUSTERED(ApplicationID)
);
GO

CREATE TABLE BlackList (
  ApplicationID         INT             NOT NULL,
  PRIMARY KEY CLUSTERED(ApplicationID)
);
GO

CREATE TABLE OncallList (
  ApplicationID         INT             NOT NULL,
  OncallReason          VARCHAR(500)    NULL,
  PRIMARY KEY CLUSTERED(ApplicationID)
);
GO

CREATE TABLE OfferExtended (
  ApplicationID               INT             NOT NULL,
  StatusID                    INT             NOT NULL,
  Comments                    VARCHAR(500)    NULL,
  PRIMARY KEY CLUSTERED(ApplicationID),
  CONSTRAINT FK_OffereExtended_Status FOREIGN KEY(StatusID) REFERENCES Status(StatusID)
);
GO

CREATE TABLE Onboarding (
  ApplicationID               INT             NOT NULL,
  IdentityCheck               BIT             NOT NULL        default 0,
  EmploymentAuthorizationCheck   BIT          NOT NULL        default 0,
  BackgroundCheck             BIT             NOT NULL        default 0,
  PRIMARY KEY CLUSTERED(ApplicationID)
);
GO

CREATE TABLE ComplainList (
  ComplainID                  INT             NOT NULL        PRIMARY KEY IDENTITY,
  ApplicationID               INT             NOT NULL,
  StatusID                    INT             NOT NULL,
  Comments                    VARCHAR(500)    NULL,
  CONSTRAINT FK_ComplainList_Application FOREIGN KEY(ApplicationID) REFERENCES
ApplicationS(ApplicationID),
  CONSTRAINT FK_ComplainList_Status FOREIGN KEY(StatusID) REFERENCES Status(StatusID),
);
GO
```

```
CREATE TABLE Employee (
  EmployeeID             INT      NOT NULL         PRIMARY KEY IDENTITY,
  ApplicationID          INT      NOT NULL,
  CONSTRAINT FK_Employee_Application FOREIGN KEY(ApplicationID) REFERENCES
Applications(ApplicationID)
);
GO

CREATE TABLE ReviewAndRatings (
  InterviewID                   INT              NOT NULL,
  CandidateRating               INT              NULL,
  CandidateComment              VARCHAR(500)     NULL,
  InterviewerRating             INT              NULL,
  InterviewerComment            VARCHAR(500)     NULL
  CONSTRAINT FK_ReviewAndRating_Interview FOREIGN KEY(InterviewID) REFERENCES
Interviews(InterviewID),
);
GO



SET IDENTITY_INSERT Candidates ON;
INSERT INTO Candidates(CandidateID,CandidateFname,CandidateLname) VALUES
(1,'Kaan','Rose'),
(2,'Joan','Byrd'),
(3,'Khalid','Patel'),
(4,'Joel','Liu'),
(5,'Talha','Hall'),
(6,'Ethel','Kinney'),
(7,'Timothy','Pollard'),
(8,'Franky','Wagner'),
(9,'Chloe','Robins'),
(10,'Brenden','Wilder');
SET IDENTITY_INSERT Candidates OFF;

INSERT INTO CandidateContact(CandidateID,CandidateAddress1,CandidateAddress2,CandidateCity,
                         CandidateState,CandidateZip,CandidatePhone,CandidateEmail) VALUES
(1,'2091  University Drive','Apt 33','Washinton','DC',20029,'312-729-7514','rose@hotmail.com'),
(2,'71  Lynch Street','Apt 2','New Berlin','WI',53151,'920-988-8307','bjoan@gmail.com'),
(3,'1366  Sardis Station','Apt B','Minneapolis','MN',55415,'612-386-
6800','patelkhalid222@hotmail.com'),
(4,'1285  Hudson Street','C66','Haledon','NJ',05508,'973-904-3200','jliuuuuu@yahoo.com'),
(5,'4530  Cedar Lane','Apt 95','DALLAS','TX',75373,'617-572-8443','halloffamertalha@outlook.com'),
(6,'311  Park Boulevard','7A','Mason City','IA',50401,'641-494-8058','bestkinney@icloud.com'),
(7,'4704  Plainfield Avenue','Apt 1048','Utica','NY',13502,'315-600-6658','timmyking@yahoo.com'),
(8,'447  Graystone Lakes','apt 9804','HOLTSVILLE','NY',14851,'478-453-
7354','wagwagwagner@icloud.com'),
(9,'255  Geneva Street','322','Mineola','NY',11501,'917-509-6700','cr7537@gmail.com'),
(10,'3644  Twin Oaks Drive','apt 11','Pellston','MI',49769,'231-539-9755','brendenw@outlook.com');


SET IDENTITY_INSERT Interviewers ON;
INSERT INTO Interviewers(InterviewerID,InterviewerFname,InterviewerLname) VALUES
(1,'Ramone','Gamble'),
(2,'Ted','Richard'),
(3,'Fred','Hurley'),
(4,'Vinnie','North');
SET IDENTITY_INSERT Interviewers OFF;


SET IDENTITY_INSERT InterviewTypes ON;
INSERT INTO InterviewTypes(InterviewTypeID,InterviewTypeName,InterviewTypeDescription) VALUES
(1,'On Site','Candidate must personally meet with interviewer'),
(2,'Online','Candidate meet interviewer online through face time');
SET IDENTITY_INSERT InterviewTypes OFF;


SET IDENTITY_INSERT JobTypes ON;
INSERT INTO JobTypes(JobTypeID,JobTypeName,JobTypeDescription) VALUES
```

```
(1,'Summer Internship','Candidate works for two months as an intern'),
(2,'Full Time','Candidate works for at least 12 months as full time employee with benefits'),
(3,'Contract Based','Candidate signs contract with company, and get paid based on amount of work
done');
SET IDENTITY_INSERT JobTypes OFF;

SET IDENTITY_INSERT JobOpening ON;
INSERT INTO JobOpening(JobID,JobTypeID,JobTitle,JobDescription,NumberOfOpenings) VALUES
(1,1,'Entry Level Software Developper','Entry level work, with training', 2),
(2,1,'Elecrical Engineer','Circuit Design', 1),
(3,2,'Database Admin','Manage and maintain database', 1);
SET IDENTITY_INSERT JobOpening OFF;


SET IDENTITY_INSERT Status ON;
INSERT INTO Status(StatusID,StatusName,StatusDescription) VALUES
(1,'Complain Received','Candidate filed a complain'),
(2,'Complain Waiting','Complain under review'),
(3,'Complain Accepted','Re-interview'),
(4,'Complain Rejected','Candidate Rejected'),
(5,'Application Received','Candidate submitted application'),
(6,'Application Accepted','Schedule Interview'),
(7,'Application Rejected','Candidate Rejected'),
(8,'Offer Accepted','Candidate accepts offer'),
(9,'Offer Rejected','Candidate rejects offer'),
(10,'Offer Negotiating','Negotiation in progress');
SET IDENTITY_INSERT Status OFF;


SET IDENTITY_INSERT Applications ON;
INSERT INTO Applications(ApplicationID,CandidateID,JobID,NumberOfInterviews,StatusID) VALUES
(1,1,1,0,7),
(2,2,1,0,7),
(3,3,2,2,1),
(4,4,1,2,8),
(5,5,1,2,10),
(6,6,1,2,8),
(7,7,2,0,7),
(8,8,3,2,8),
(9,9,3,1,8),
(10,10,1,1,9);
SET IDENTITY_INSERT Applications OFF;

SET IDENTITY_INSERT Interviews ON;
INSERT INTO Interviews(InterviewID,InterviewerID,ApplicationID,InterviewTypeID,Decision) VALUES
(1,1,3,1,1),
(2,2,4,2,1),
(3,1,5,2,0),
(4,3,6,2,1),
(5,4,3,1,0),
(6,2,4,2,1),
(7,3,8,2,1),
(8,1,9,2,1),
(9,4,10,2,1),
(10,4,5,2,1),
(11,1,6,2,1),
(12,3,8,2,1);
SET IDENTITY_INSERT Interviews OFF;


SET IDENTITY_INSERT AirlineReservations ON;
INSERT INTO AirlineReservations(AirlineReservationID,ApplicationID,AirlineTicketNumber,
                                 AirlineName,FlightNumber,DepartureDate,AirFareCost) VALUES
(1,3,'TN359462YH','Delta','DL2394','04/10/2020',420);
SET IDENTITY_INSERT AirlineReservations OFF;
```

```sql
SET IDENTITY_INSERT CarRentals ON;
INSERT INTO CarRentals(CarRentalID,ApplicationID,CarRentalTicketNumber,
                        CarRentalCompanyName,RentedDate,ReturnedDate,CarRentalCost) VALUES
(1,3,'657439KK','Enterprise','04/10/2020','4/13/2020',300);
SET IDENTITY_INSERT CarRentals OFF;


SET IDENTITY_INSERT HotelReservations ON;
INSERT INTO HotelReservations(HotelReservationID,ApplicationID,HotelReservationTicketNumber,
                        HotelName,NightsOfStay,HotelCost) VALUES
(1,3,'NKY15GS9716','Hilton',3,600);
SET IDENTITY_INSERT HotelReservations OFF;


SET IDENTITY_INSERT Reimbursements ON;
INSERT INTO Reimbursements(ReimbursementID,ApplicationID,NumberOfReceipts,
                        TotalAsked,TotalReimbursed,ReimburseComplete) VALUES
(1,3,6,1500,1500,1);
SET IDENTITY_INSERT Reimbursements OFF;


INSERT INTO RejectedList(ApplicationID,RejectedReason) VALUES
(3,'Skill Lacking'),
(1,'Not enough experience'),
(2,'Skill lacking'),
(7,'Not enough experience');


INSERT INTO BlackList(ApplicationID) VALUES
(6);


INSERT INTO OncallList(ApplicationID,OncallReason) VALUES
(9,'Position already filled');


INSERT INTO Onboarding(ApplicationID,IdentityCheck,EmploymentAuthorizationCheck,BackgroundCheck)
VALUES
(4,1,1,0);


SET IDENTITY_INSERT Employee ON;
INSERT INTO Employee(EmployeeID,ApplicationID) VALUES
(1,8);
SET IDENTITY_INSERT Employee OFF;


INSERT INTO OfferExtended(ApplicationID,StatusID,Comments) VALUES
(4,8,'Onboarding process'),
(5,10,'Offer in negotiate'),
(6,8,''),
(8,8,''),
(9,8,''),
(10,9,'');


SET IDENTITY_INSERT ComplainList ON;
INSERT INTO ComplainList(ComplainID,ApplicationID,StatusID, Comments) VALUES
(1,3,1,''),
(2,5,3,'Complain valid');
SET IDENTITY_INSERT ComplainList OFF;


INSERT INTO
ReviewAndRatings(InterviewID,CandidateRating,CandidateComment,InterviewerRating,InterviewerComment
) VALUES
(1,10,'',9,''),
(2,8,'',10,''),
(3,7,'',7,''),
(4,10,'',10,''),
(5,5,'Candidate no dress code',3,'Interviewer was rude'),
(6,9,'',10,''),
(7,10,'',7,''),
(8,8,'',10,''),
(9,10,'',9,'Interview too long'),
(10,9,'',10,''),
(11,10,'',8,''),
(12,10,'',10,'');
```

First, check if database exists, if so, drop it and create a new using CREATE DATABASE. Then for each table, use CREATE TABLE to create table and set column names along with any constrains. The primary and foreign keys are linked in the creation of table. Lastly, use INSERT INTO…VALUES to add entries to the table in the same order of the columns. The SET IDENTITY_INSERT ON/OFF allows insertion of primary keys.

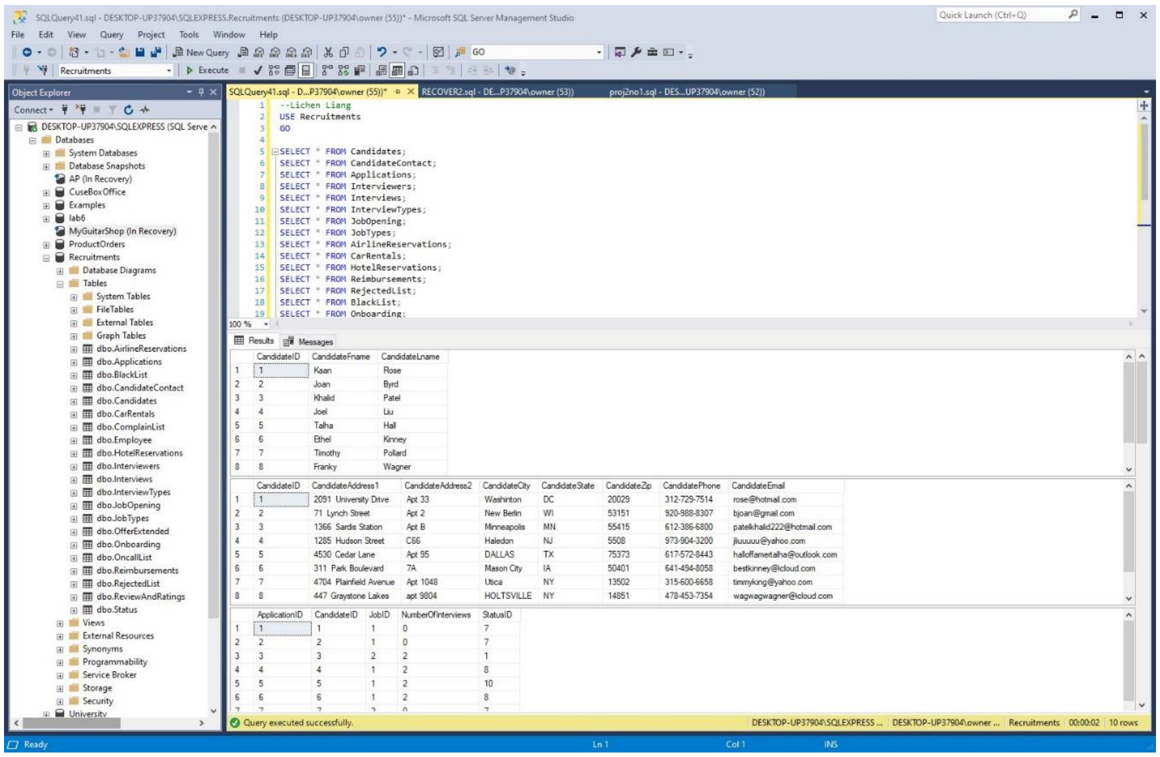

proj2no1.sql - DESKTOP-UP37904\SQLEXPRESS.Recruitments (DESKTOP-UP37904\owner (52)) - Microsoft SQL Server Management Studio
File Edit View Query Project Tools Window Help
Object Explorer
RECOVER2.sql - DE...P37904\owner (53))
proj2no1.sql - DES...UP37904\owner (52))

```
--Lichen Liang
/*Create database, tables, columns, and define all constraints
Then insert realistic values into the tables*/

USE master;
GO

/*if database already exist, drop it*/
IF  DB_ID('Recruitments') IS NOT NULL
    DROP DATABASE Recruitments;
GO

CREATE DATABASE Recruitments;
GO

USE Recruitments;


CREATE TABLE Candidates (
  CandidateID       INT             NOT NULL    PRIMARY KEY IDENTITY,
  CandidateFname    VARCHAR(100)    NOT NULL,
  CandidateLname    VARCHAR(100)    NOT NULL
);
GO

CREATE TABLE CandidateContact (
  CandidateID           INT             NOT NULL,
  CandidateAddress1     VARCHAR(255)    NULL,
  CandidateAddress2     VARCHAR(255)    NULL,
  CandidateCity         VARCHAR(100)    NULL,
  CandidateState        CHAR(2)         NULL,
```

```
(10 rows affected)

(10 rows affected)

(4 rows affected)

(2 rows affected)

(3 rows affected)

(3 rows affected)

(10 rows affected)

(10 rows affected)

(12 rows affected)
```

Query executed successfully. DESKTOP-UP37904\SQLEXPRESS ... DESKTOP-UP37904\owner ... Recruitments 00:00:01 0 rows

Figure 2. Database, Tables, and Columns are created successfully.



```sql
--Lichen Liang
USE Recruitments
GO

SELECT * FROM Candidates;
SELECT * FROM CandidateContact;
SELECT * FROM Applications;
SELECT * FROM Interviewers;
SELECT * FROM Interviews;
SELECT * FROM InterviewTypes;
SELECT * FROM JobOpening;
SELECT * FROM JobTypes;
SELECT * FROM AirlineReservations;
SELECT * FROM CarRentals;
SELECT * FROM HotelReservations;
SELECT * FROM Reimbursements;
SELECT * FROM RejectedList;
SELECT * FROM BlackList;
SELECT * FROM Onboarding;
```

	CandidateID	CandidateFname	CandidateLname
1	1	Kaan	Rose
2	2	Joan	Byrd
3	3	Khalid	Patel
4	4	Joel	Liu
5	5	Talha	Hall
6	6	Ethel	Kinney
7	7	Timothy	Pollard
8	8	Franky	Wagner

	CandidateID	CandidateAddress1	CandidateAddress2	CandidateCity	CandidateState	CandidateZip	CandidatePhone	CandidateEmail
1	1	2091 University Drive	Apt 33	Washinton	DC	20029	312-729-7514	rose@hotmail.com
2	2	71 Lynch Street	Apt 2	New Berlin	WI	53151	920-988-8307	bjoan@gmail.com
3	3	1366 Sardis Station	Apt B	Minneapolis	MN	55415	612-386-6800	patelkhalid222@hotmail.com
4	4	1285 Hudson Street	C66	Haledon	NJ	5508	973-904-3200	jliuuuuu@yahoo.com
5	5	4530 Cedar Lane	Apt 95	DALLAS	TX	75373	617-572-8443	halloffamertalha@outlook.com
6	6	311 Park Boulevard	7A	Mason City	IA	50401	641-494-8058	bestkinney@icloud.com
7	7	4704 Plainfield Avenue	Apt 1048	Utica	NY	13502	315-600-6658	timmyking@yahoo.com
8	8	447 Graystone Lakes	apt 9804	HOLTSVILLE	NY	14851	478-453-7354	wagwagwagner@icloud.com

	ApplicationID	CandidateID	JobID	NumberOfInterviews	StatusID
1	1	1	1	0	7
2	2	2	1	0	7
3	3	3	2	2	1
4	4	4	1	2	8
5	5	5	1	2	10
6	6	6	1	2	8
7	7	7	2	0	7

Figure 3. Verify that all data has been inserted successfully

Now the database has been created successfully with data inserted. It is time to test them.

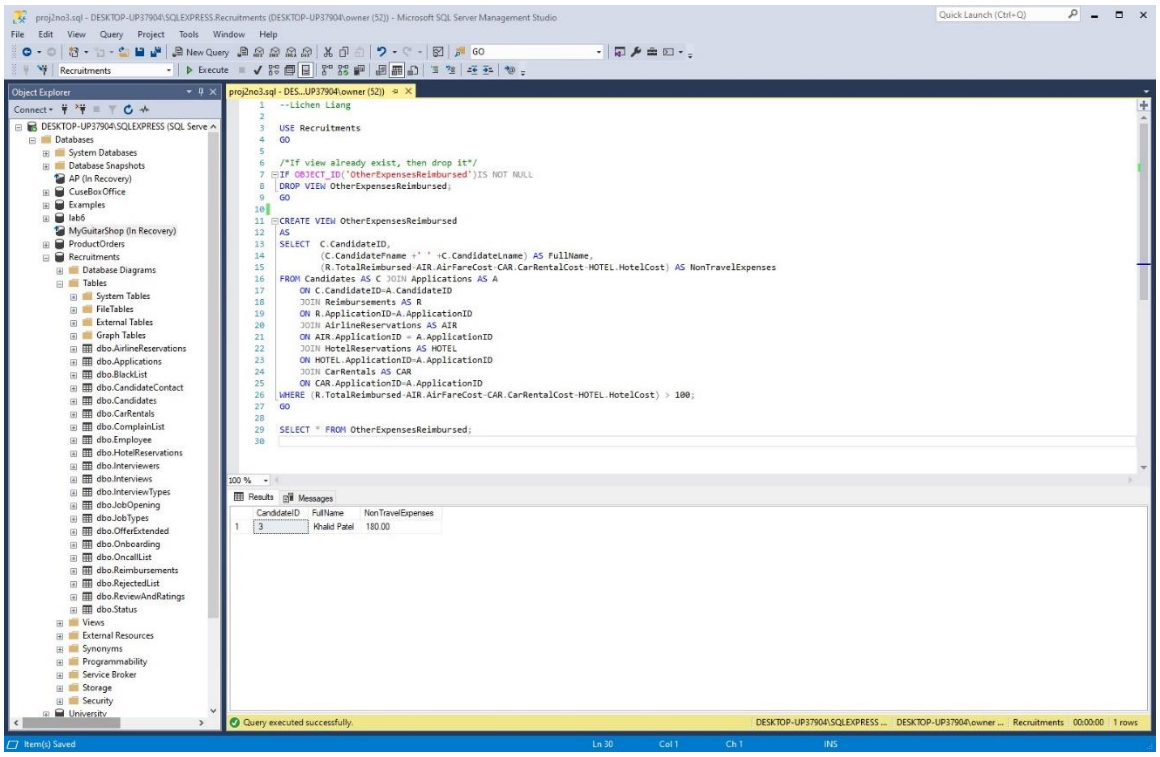
Views

A view is a temporary table or a named query. I have simulated four scenarios (four views) to test the data. In this section you will see the scenario description, source code, comment for the source code, and screenshot verification.

1. The first scenario lists the candidate and find the 'non-travel' expenses that have been reimbursed by the candidate where this kind of expenses surpasses 100. For example, such expenses include food, local transportation, and any other valid reasons that this amount must be reimbursed, but excludes hotel, car rental, and air travel.

```
USE Recruitments
GO

/*If view already exist, then drop it*/
IF OBJECT_ID('OtherExpensesReimbursed')IS NOT NULL
DROP VIEW OtherExpensesReimbursed;
GO

CREATE VIEW OtherExpensesReimbursed
AS
SELECT  C.CandidateID,
        (C.CandidateFname +' ' +C.CandidateLname) AS FullName,
        (R.TotalReimbursed-AIR.AirFareCost-CAR.CarRentalCost-HOTEL.HotelCost) AS
        NonTravelExpenses
FROM Candidates AS C JOIN Applications AS A
        ON C.CandidateID=A.CandidateID
        JOIN Reimbursements AS R
        ON R.ApplicationID=A.ApplicationID
        JOIN AirlineReservations AS AIR
        ON AIR.ApplicationID = A.ApplicationID
        JOIN HotelReservations AS HOTEL
        ON HOTEL.ApplicationID=A.ApplicationID
        JOIN CarRentals AS CAR
        ON CAR.ApplicationID=A.ApplicationID
WHERE (R.TotalReimbursed-AIR.AirFareCost-CAR.CarRentalCost-HOTEL.HotelCost) > 100;
GO

SELECT * FROM OtherExpensesReimbursed;
```

First, check if the view is already in the database, if so, drop it and re-create. Create a view using CREATE VIEW…AS. The SELECT statement returns three columns and the tables are connected using JOIN..ON. Tables are connected with application ID except candidates and applications through candidate ID. The condition is set in the WHERE clause where the reimbursed amount should be greater than 100.



```sql
--Lichen Liang

USE Recruitments
GO

/*If view already exist, then drop it*/
IF OBJECT_ID('OtherExpensesReimbursed')IS NOT NULL
DROP VIEW OtherExpensesReimbursed;
GO

CREATE VIEW OtherExpensesReimbursed
AS
SELECT  C.CandidateID,
        (C.CandidateFname +' ' +C.CandidateLname) AS FullName,
        (R.TotalReimbursed-AIR.AirFareCost-CAR.CarRentalCost-HOTEL.HotelCost) AS NonTravelExpenses
FROM Candidates AS C JOIN Applications AS A
    ON C.CandidateID=A.CandidateID
    JOIN Reimbursements AS R
    ON R.ApplicationID=A.ApplicationID
    JOIN AirlineReservations AS AIR
    ON AIR.ApplicationID = A.ApplicationID
    JOIN HotelReservations AS HOTEL
    ON HOTEL.ApplicationID=A.ApplicationID
    JOIN CarRentals AS CAR
    ON CAR.ApplicationID=A.ApplicationID
WHERE (R.TotalReimbursed-AIR.AirFareCost-CAR.CarRentalCost-HOTEL.HotelCost) > 100;
GO

SELECT * FROM OtherExpensesReimbursed;
```

	CandidateID	FullName	NonTravelExpenses
1	3	Khalid Patel	180.00

```
Query executed successfully.
```

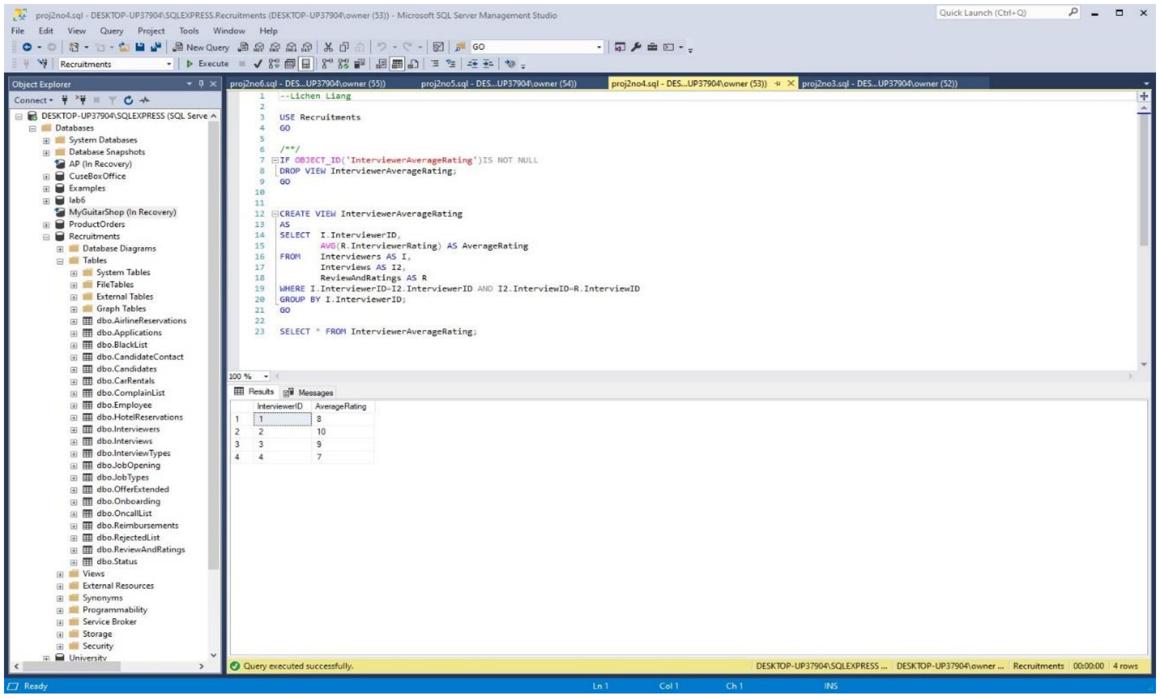
Figure 4. Candidates with non-travel reimbursement over 100.

Candidate 3 is the only one with on-site interview. Therefore, this is expected result.

2. For every interview, an interviewer will get a rating. In this view, I want to return the average rating for every interviewer.

```sql
USE Recruitments
GO

IF OBJECT_ID('InterviewerAverageRating')IS NOT NULL
DROP VIEW InterviewerAverageRating;
GO


CREATE VIEW InterviewerAverageRating
AS
SELECT I.InterviewerID,
       AVG(R.InterviewerRating) AS AverageRating
FROM   Interviewers AS I,
       Interviews AS I2,
       ReviewAndRatings AS R
WHERE I.InterviewerID=I2.InterviewerID AND I2.InterviewID=R.InterviewID
GROUP BY I.InterviewerID;
GO

SELECT * FROM InterviewerAverageRating;
```

Again, check for existence of the view. Use the same method to CREATE VIEW. The SELECT statement returns the interviewer ID and its corresponding average rating using AVG(). The tables are matched in the WHERE clause, with table alias. Interviewer, interview, review and ratings, are linked with interviewer ID and interview ID respectively. Finally, GROUP BY the interviewer ID to calculate average.



```sql
--Lichen Liang

USE Recruitments
GO

/**/
IF OBJECT_ID('InterviewerAverageRating')IS NOT NULL
DROP VIEW InterviewerAverageRating;
GO


CREATE VIEW InterviewerAverageRating
AS
SELECT  I.InterviewerID,
        AVG(R.InterviewerRating) AS AverageRating
FROM    Interviewers AS I,
        Interviews AS I2,
        ReviewAndRatings AS R
WHERE I.InterviewerID=I2.InterviewerID AND I2.InterviewID=R.InterviewID
GROUP BY I.InterviewerID;
GO

SELECT * FROM InterviewerAverageRating;
```

	InterviewerID	AverageRating
1	1	8
2	2	10
3	3	9
4	4	7

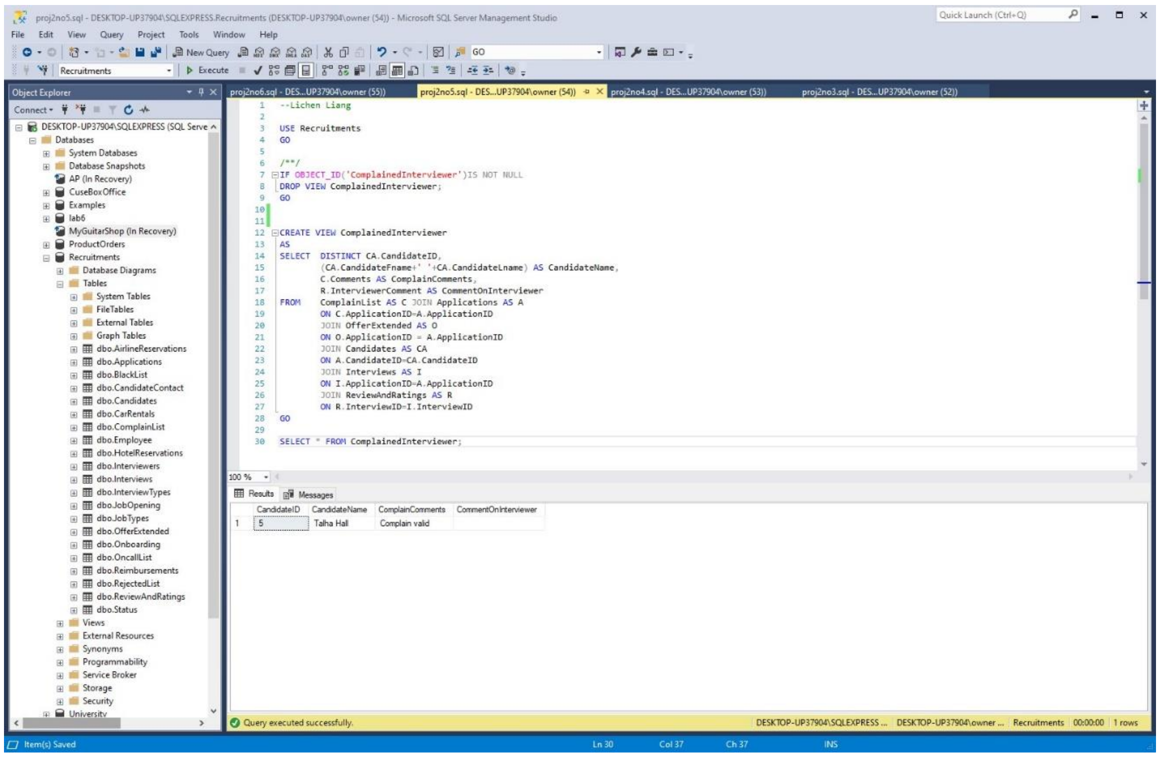
Query executed successfully.

Figure 5. Interviewer and his/her average rating

3. In this scenario, I want to find the candidate who have filed a complaint, and eventually given an offer for a job. The candidate might leave any review on the interviewer who rejected his/her application which can be useful for evaluating the interviewer's quality.

```sql
USE Recruitments
GO

IF OBJECT_ID('ComplainedInterviewer')IS NOT NULL
DROP VIEW ComplainedInterviewer;
GO


CREATE VIEW ComplainedInterviewer
AS
SELECT DISTINCT CA.CandidateID,
              (CA.CandidateFname+' '+CA.CandidateLname) AS CandidateName,
              C.Comments AS ComplainComments,
              R.InterviewerComment AS CommentOnInterviewer
FROM   ComplainList AS C JOIN Applications AS A
       ON C.ApplicationID=A.ApplicationID
       JOIN OfferExtended AS O
       ON O.ApplicationID = A.ApplicationID
       JOIN Candidates AS CA
       ON A.CandidateID=CA.CandidateID
       JOIN Interviews AS I
       ON I.ApplicationID=A.ApplicationID
       JOIN ReviewAndRatings AS R
       ON R.InterviewID=I.InterviewID
GO

SELECT * FROM ComplainedInterviewer;
```

Again, create the view using same method as before. However, we want to select DISTINCT candidates who left the review. The query returns candidate ID, his/her full name concatenated, the comments in the complaint that was filed, and the review on the interviewer he/she gave. The tables are again joined with application ID, candidate ID, and interview ID.



```sql
--Lichen Liang

USE Recruitments
GO

/**/
IF OBJECT_ID('ComplainedInterviewer')IS NOT NULL
DROP VIEW ComplainedInterviewer;
GO


CREATE VIEW ComplainedInterviewer
AS
SELECT  DISTINCT CA.CandidateID,
        (CA.CandidateFname+' '+CA.CandidateLname) AS CandidateName,
        C.Comments AS ComplainComments,
        R.InterviewerComment AS CommentOnInterviewer
FROM    ComplainList AS C JOIN Applications AS A
        ON C.ApplicationID=A.ApplicationID
        JOIN OfferExtended AS O
        ON O.ApplicationID = A.ApplicationID
        JOIN Candidates AS CA
        ON A.CandidateID=CA.CandidateID
        JOIN Interviews AS I
        ON I.ApplicationID=A.ApplicationID
        JOIN ReviewAndRatings AS R
        ON R.InterviewID=I.InterviewID
GO

SELECT * FROM ComplainedInterviewer;
```

	CandidateID	CandidateName	ComplainComments	CommentOnInterviewer
1	5	Talha Hall	Complain valid	

```
Query executed successfully.
```

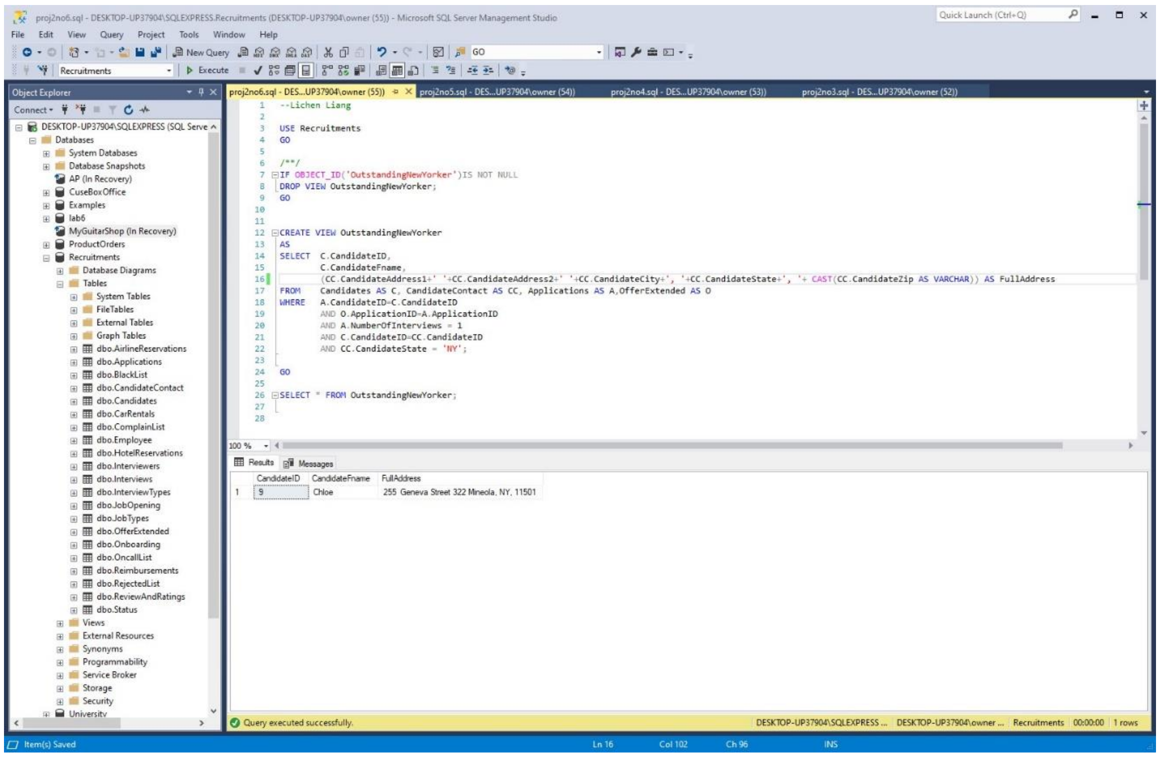
Figure 6. The review for the interviewer

Unfortunately, the candidate did not leave any review for the interviewer who rejected him/her.

4. In this scenario, I want to know who the outstanding New Yorker is. That is, a candidate who has been given an offer for a position in the first and only interview. He/she must be in New York State. Return the ID, first name and the full address concatenated.

```
USE Recruitments
GO


IF OBJECT_ID('OutstandingNewYorker')IS NOT NULL
DROP VIEW OutstandingNewYorker;
GO


CREATE VIEW OutstandingNewYorker
AS
SELECT C.CandidateID,
       C.CandidateFname,
       (CC.CandidateAddress1+' '+CC.CandidateAddress2+'
       '+CC.CandidateCity+', '+CC.CandidateState+', '+ CAST(CC.CandidateZip
       AS VARCHAR)) AS FullAddress
FROM   Candidates AS C, CandidateContact AS CC, Applications AS
A,OfferExtended AS O
WHERE  A.CandidateID=C.CandidateID
       AND O.ApplicationID=A.ApplicationID
       AND A.NumberOfInterviews = 1
       AND C.CandidateID=CC.CandidateID
       AND CC.CandidateState = 'NY';

GO

SELECT * FROM OutstandingNewYorker;
```

Create the view using same method as before. The returned columns are concatenated, and zip code must be character value using CAST(). Then set the conditions in the WHERE clause which also links the tables together. Number of interviews must be 1 as mentioned and must be in NY state.



```sql
--Lichen Liang

USE Recruitments
GO

/**/
IF OBJECT_ID('OutstandingNewYorker')IS NOT NULL
DROP VIEW OutstandingNewYorker;
GO


CREATE VIEW OutstandingNewYorker
AS
SELECT  C.CandidateID,
        C.CandidateFname,
        (CC.CandidateAddress1+' '+CC.CandidateAddress2+' '+CC.CandidateCity+', '+CC.CandidateState+', '+ CAST(CC.CandidateZip AS VARCHAR)) AS FullAddress
FROM    Candidates AS C, CandidateContact AS CC, Applications AS A,OfferExtended AS O
WHERE   A.CandidateID=C.CandidateID
        AND O.ApplicationID=A.ApplicationID
        AND A.NumberOfInterviews = 1
        AND C.CandidateID=CC.CandidateID
        AND CC.CandidateState = 'NY';

GO

SELECT * FROM OutstandingNewYorker;
```

	CandidateID	CandidateFname	FullAddress
1	9	Chloe	255 Geneva Street 322 Mineola, NY, 11501

```
Query executed successfully.
```

Figure 7. The outstanding New Yorker

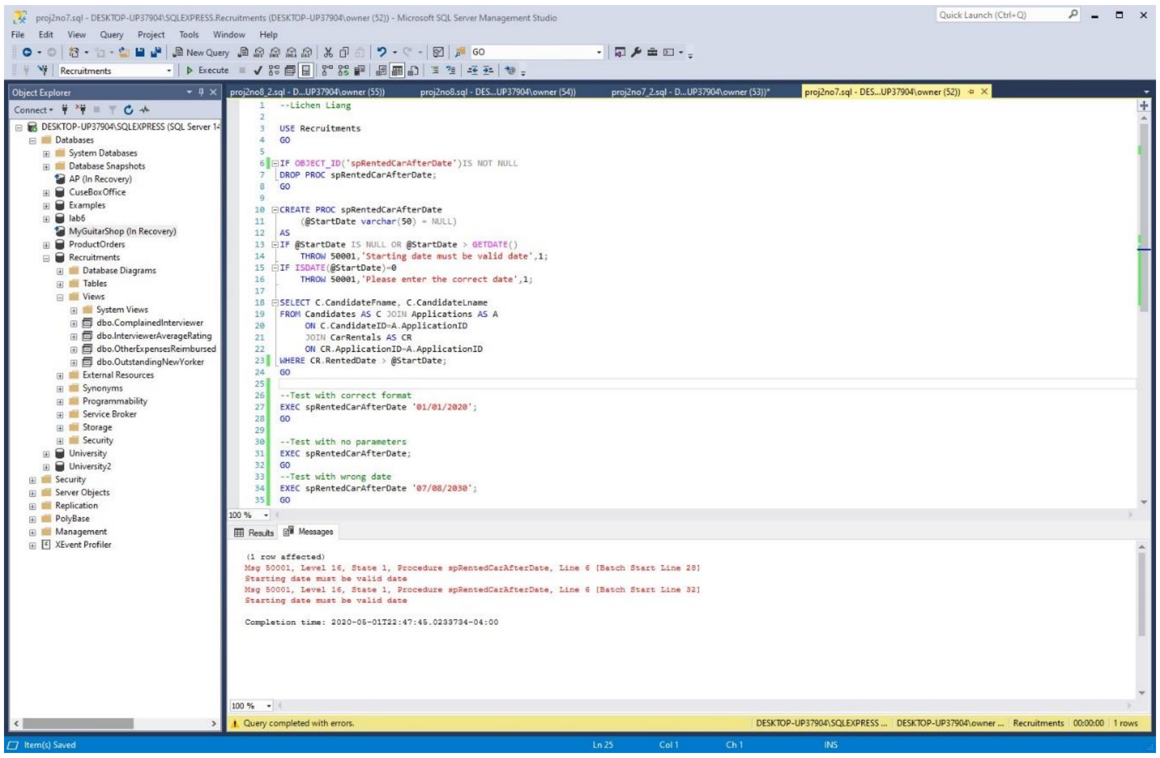
Stored Procedure

In this part, I created two stored procedures to return some results with entered parameters. Stored procedure is fast because it is precompiled in the database.

1. In the first one, I want to find the person who rented car after a specific date entered by me. If the date entered was wrong or missing, it should raise an error.

```
USE Recruitments
GO

IF OBJECT_ID('spRentedCarAfterDate')IS NOT NULL
DROP PROC spRentedCarAfterDate;
GO

CREATE PROC spRentedCarAfterDate
       (@StartDate varchar(50) = NULL)
AS
IF @StartDate IS NULL OR @StartDate > GETDATE()
       THROW 50001,'Starting date must be valid date',1;
IF ISDATE(@StartDate)=0
       THROW 50001,'Please enter the correct date',1;

SELECT C.CandidateFname, C.CandidateLname
FROM Candidates AS C JOIN Applications AS A
        ON C.CandidateID=A.ApplicationID
        JOIN CarRentals AS CR
        ON CR.ApplicationID=A.ApplicationID
WHERE CR.RentedDate > @StartDate;
GO

--Test with correct format
EXEC spRentedCarAfterDate '01/01/2020';
GO

--Test with no parameters
EXEC spRentedCarAfterDate;
GO
--Test with wrong date
EXEC spRentedCarAfterDate '07/08/2030';
GO
```

Check if the procedure exists first, if so, drop it. Then create a procedure using CREAT PROC..AS. Also specify the parameter and its corresponding datatype, and default values, if any. I used two IF statement to check whether parameters are valid, if not, THROW and error. The conditions are the date must be not null, must not be in the future, and the date entered is in correct format of a date. The SELECT statement returns the candidate name who rented car after the date I specified. The condition is checked in the WHERE clause and the tables are linked using JOIN..ON. The procedure is called using EXEC. The first test is correct, and the second and third should raise an error.

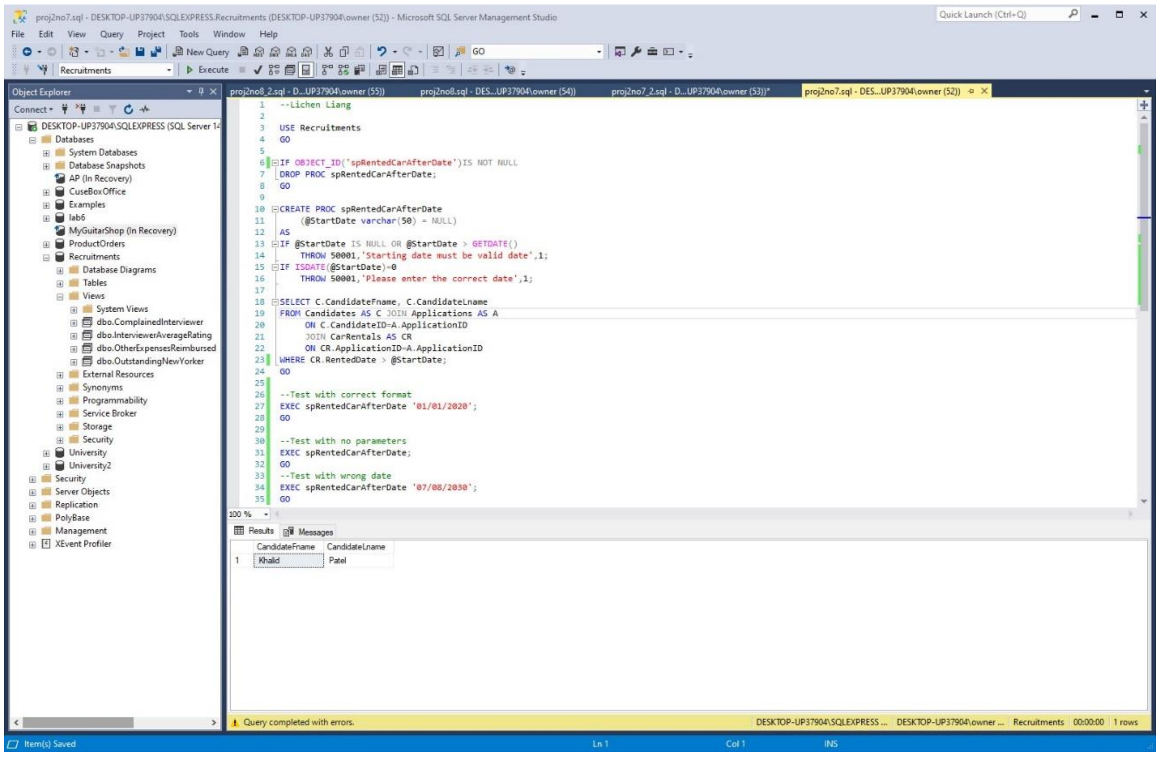


```sql
--Lichen Liang

USE Recruitments
GO

IF OBJECT_ID('spRentedCarAfterDate')IS NOT NULL
DROP PROC spRentedCarAfterDate;
GO

CREATE PROC spRentedCarAfterDate
    (@StartDate varchar(50) = NULL)
AS
IF @StartDate IS NULL OR @StartDate > GETDATE()
    THROW 50001,'Starting date must be valid date',1;
IF ISDATE(@StartDate)=0
    THROW 50001,'Please enter the correct date',1;

SELECT C.CandidateFname, C.CandidateLname
FROM Candidates AS C JOIN Applications AS A
     ON C.CandidateID=A.ApplicationID
     JOIN CarRentals AS CR
     ON CR.ApplicationID=A.ApplicationID
WHERE CR.RentedDate > @StartDate;
GO

--Test with correct format
EXEC spRentedCarAfterDate '01/01/2020';
GO

--Test with no parameters
EXEC spRentedCarAfterDate;
GO
--Test with wrong date
EXEC spRentedCarAfterDate '07/08/2030';
GO
```

```
(1 row affected)
Msg 50001, Level 16, State 1, Procedure spRentedCarAfterDate, Line 6 [Batch Start Line 28]
Starting date must be valid date
Msg 50001, Level 16, State 1, Procedure spRentedCarAfterDate, Line 6 [Batch Start Line 32]
Starting date must be valid date

Completion time: 2020-05-01T22:47:45.0233734-04:00
```

Query completed with errors.

Figure 8. One was successful and two errors was raised



```sql
--Lichen Liang

USE Recruitments
GO

IF OBJECT_ID('spRentedCarAfterDate')IS NOT NULL
DROP PROC spRentedCarAfterDate;
GO

CREATE PROC spRentedCarAfterDate
    (@StartDate varchar(50) = NULL)
AS
IF @StartDate IS NULL OR @StartDate > GETDATE()
    THROW 50001,'Starting date must be valid date',1;
IF ISDATE(@StartDate)=0
    THROW 50001,'Please enter the correct date',1;

SELECT C.CandidateFname, C.CandidateLname
FROM Candidates AS C JOIN Applications AS A
     ON C.CandidateID=A.ApplicationID
     JOIN CarRentals AS CR
     ON CR.ApplicationID=A.ApplicationID
WHERE CR.RentedDate > @StartDate;
GO

--Test with correct format
EXEC spRentedCarAfterDate '01/01/2020';
GO

--Test with no parameters
EXEC spRentedCarAfterDate;
GO
--Test with wrong date
EXEC spRentedCarAfterDate '07/08/2030';
GO
```

	CandidateFname	CandidateLname
1	Khalid	Patel

Query completed with errors.

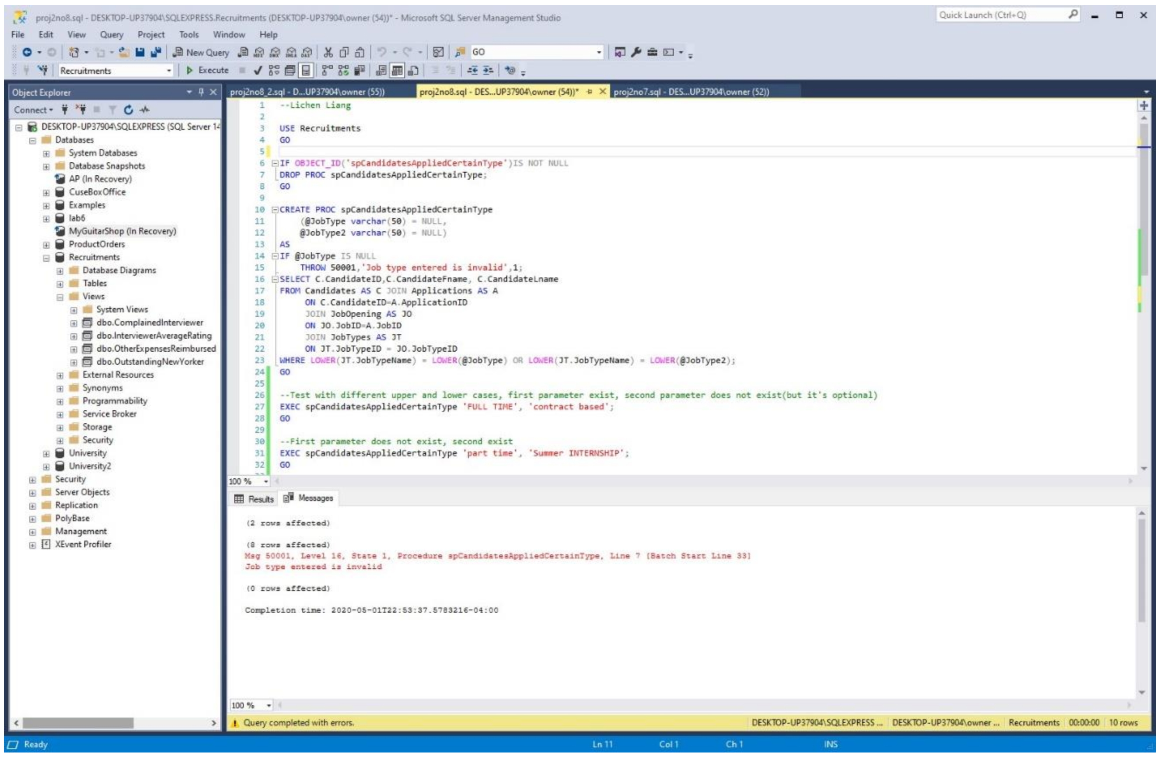
Figure 9. The candidate who rented car in 2020

This is expected result because he is the only one who reimbursed and had on-site interview.

2. In this part, I want to find all candidates who applied for certain type of job position. For example, internship, full time, etc. I want to have two to choose from at the same time, but I should at least enter one. Also, I want my parameters to be case insensitive.

```
USE Recruitments
GO

IF OBJECT_ID('spCandidatesAppliedCertainType')IS NOT NULL
DROP PROC spCandidatesAppliedCertainType;
GO

CREATE PROC spCandidatesAppliedCertainType
       (@JobType varchar(50) = NULL,
       @JobType2 varchar(50) = NULL)
AS
IF @JobType IS NULL
       THROW 50001,'Job type entered is invalid',1;
SELECT C.CandidateID,C.CandidateFname, C.CandidateLname
FROM Candidates AS C JOIN Applications AS A
        ON C.CandidateID=A.ApplicationID
        JOIN JobOpening AS JO
        ON JO.JobID=A.JobID
        JOIN JobTypes AS JT
        ON JT.JobTypeID = JO.JobTypeID
WHERE LOWER(JT.JobTypeName) = LOWER(@JobType) OR LOWER(JT.JobTypeName) =
LOWER(@JobType2);
GO

--Test with different upper and lower cases, first parameter exist, second
parameter does not exist(but it's optional)
EXEC spCandidatesAppliedCertainType 'FULL TIME', 'contract based';
GO

--First parameter does not exist, second exist
EXEC spCandidatesAppliedCertainType 'part time', 'Summer INTERNSHIP';
GO

--No parameter
EXEC spCandidatesAppliedCertainType ;
GO

--both parameter does not exist, empty table
EXEC spCandidatesAppliedCertainType 'part time', 'winter intern';
GO
```

Checking and creating the procedure like part 1. Define the parameter name, datatype, and default values. Since I want at most two parameter and at least one, I should check for the first to be not null in the IF statement, otherwise raise error using THROW. Select and join the tables as usual. The parameters should be case insensitive so I used LOWER() to make them all lower case in the WHERE clause, where I set conditions that job types should match. I had four testing cases using EXEC. The testing comments are in the code above.

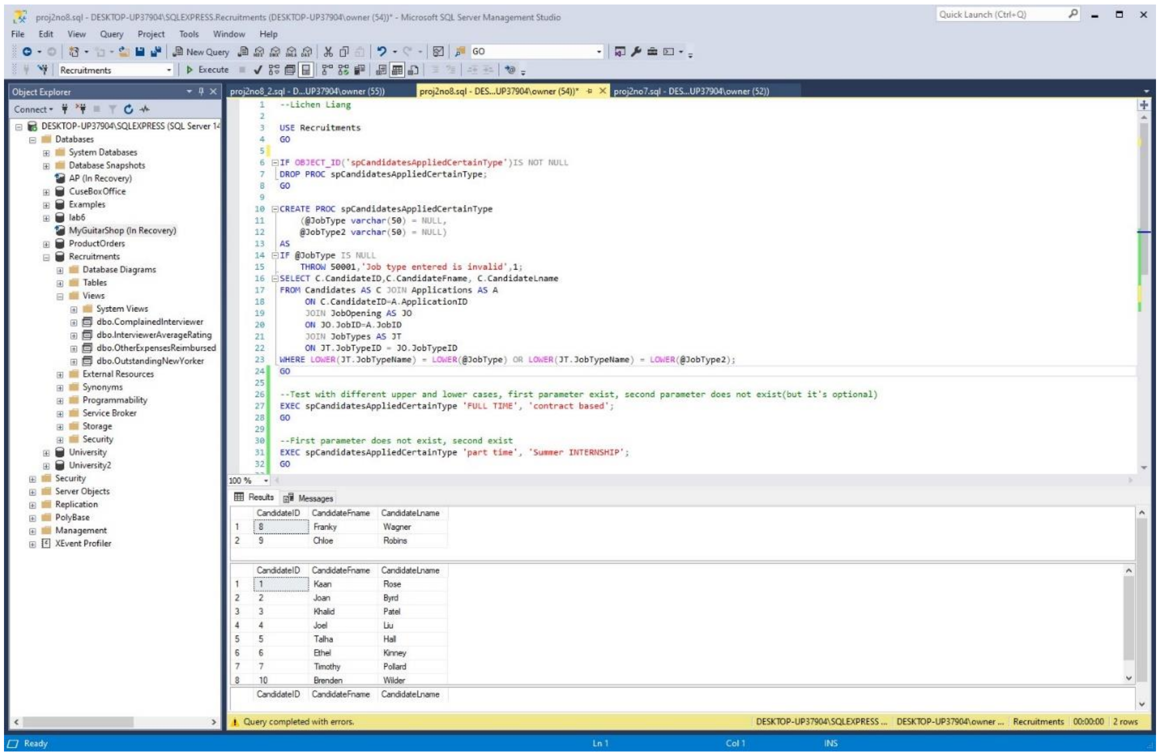


```sql
--Lichen Liang

USE Recruitments
GO

IF OBJECT_ID('spCandidatesAppliedCertainType')IS NOT NULL
DROP PROC spCandidatesAppliedCertainType;
GO

CREATE PROC spCandidatesAppliedCertainType
    (@JobType varchar(50) = NULL,
    @JobType2 varchar(50) = NULL)
AS
IF @JobType IS NULL
    THROW 50001,'Job type entered is invalid',1;
SELECT C.CandidateID,C.CandidateFname, C.CandidateLname
FROM Candidates AS C JOIN Applications AS A
     ON C.CandidateID=A.ApplicationID
     JOIN JobOpening AS JO
     ON JO.JobID=A.JobID
     JOIN JobTypes AS JT
     ON JT.JobTypeID = JO.JobTypeID
WHERE LOWER(JT.JobTypeName) = LOWER(@JobType) OR LOWER(JT.JobTypeName) = LOWER(@JobType2);
GO

--Test with different upper and lower cases, first parameter exist, second parameter does not exist(but it's optional)
EXEC spCandidatesAppliedCertainType 'FULL TIME', 'contract based';
GO

--First parameter does not exist, second exist
EXEC spCandidatesAppliedCertainType 'part time', 'Summer INTERNSHIP';
GO
```

```
(2 rows affected)

(8 rows affected)
Msg 50001, Level 16, State 1, Procedure spCandidatesAppliedCertainType, Line 7 [Batch Start Line 33]
Job type entered is invalid

(0 rows affected)

Completion time: 2020-05-01T22:53:37.5783216-04:00
```

Query completed with errors.

Figure 10. Three successful and one raised error



```sql
--Lichen Liang

USE Recruitments
GO

IF OBJECT_ID('spCandidatesAppliedCertainType')IS NOT NULL
DROP PROC spCandidatesAppliedCertainType;
GO

CREATE PROC spCandidatesAppliedCertainType
    (@JobType varchar(50) = NULL,
    @JobType2 varchar(50) = NULL)
AS
IF @JobType IS NULL
    THROW 50001,'Job type entered is invalid',1;
SELECT C.CandidateID,C.CandidateFname, C.CandidateLname
FROM Candidates AS C JOIN Applications AS A
     ON C.CandidateID=A.ApplicationID
     JOIN JobOpening AS JO
     ON JO.JobID=A.JobID
     JOIN JobTypes AS JT
     ON JT.JobTypeID = JO.JobTypeID
WHERE LOWER(JT.JobTypeName) = LOWER(@JobType) OR LOWER(JT.JobTypeName) = LOWER(@JobType2);
GO

--Test with different upper and lower cases, first parameter exist, second parameter does not exist(but it's optional)
EXEC spCandidatesAppliedCertainType 'FULL TIME', 'contract based';
GO

--First parameter does not exist, second exist
EXEC spCandidatesAppliedCertainType 'part time', 'Summer INTERNSHIP';
GO
```

	CandidateID	CandidateFname	CandidateLname
1	8	Franky	Wagner
2	9	Chloe	Robins

	CandidateID	CandidateFname	CandidateLname
1	1	Kaan	Rose
2	2	Joan	Byrd
3	3	Khalid	Patel
4	4	Joel	Liu
5	5	Talha	Hall
6	6	Ethel	Kinney
7	7	Timothy	Pollard
8	10	Brenden	Wilder

	CandidateID	CandidateFname	CandidateLname

Query completed with errors.

Figure 11. Results returned, with one empty table

As expected in the comments, one error due to missing parameter and one empty table.

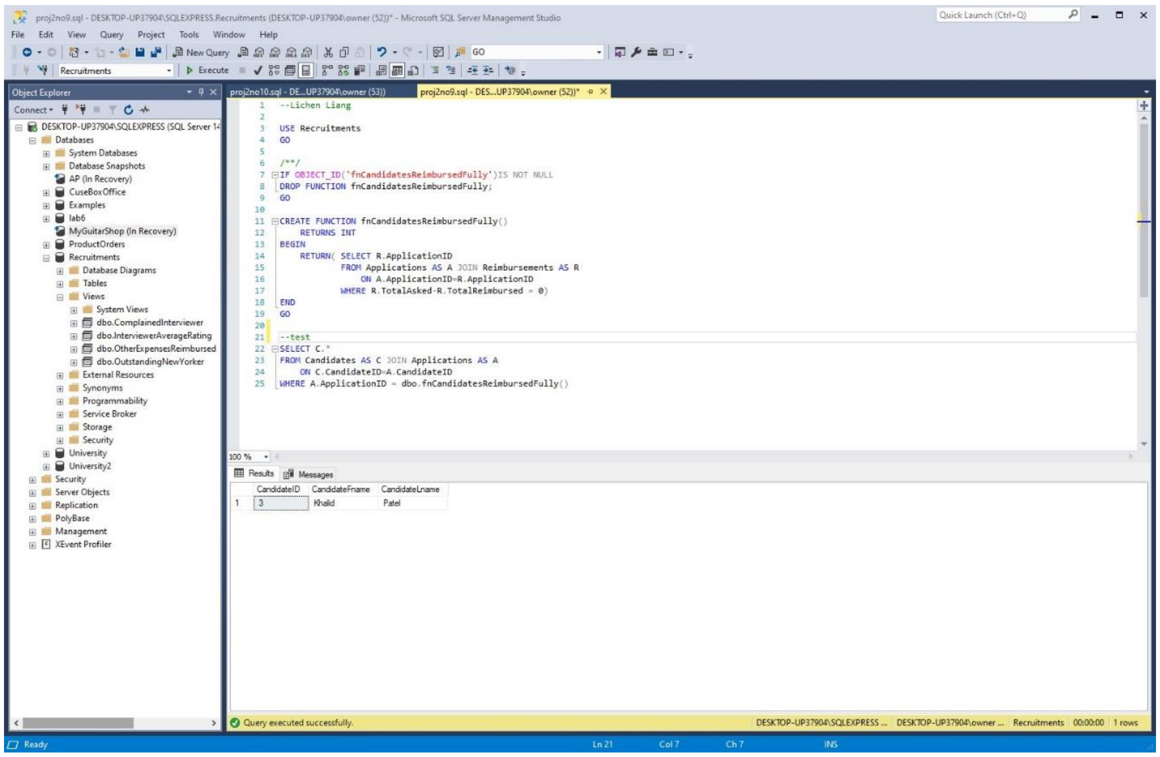
Functions

In this part I created two functions, with and without parameter. Then I called them in the select statement to return the correct results

1. For the first function, I want to find the candidates who got full reimbursement. That is, asked for X amount and got X amount.

```
USE Recruitments
GO

IF OBJECT_ID('fnCandidatesReimbursedFully')IS NOT NULL
DROP FUNCTION fnCandidatesReimbursedFully;
GO

CREATE FUNCTION fnCandidatesReimbursedFully()
       RETURNS INT
BEGIN
       RETURN(SELECT R.ApplicationID
              FROM Applications AS A JOIN Reimbursements AS R
                     ON A.ApplicationID=R.ApplicationID
              WHERE R.TotalAsked-R.TotalReimbursed = 0)
END
GO

--test
SELECT C.*
FROM Candidates AS C JOIN Applications AS A
       ON C.CandidateID=A.CandidateID
WHERE A.ApplicationID = dbo.fnCandidatesReimbursedFully()
```

Check if function exist, like before. Create a function using CREATE FUNCTION. Use RETURN INT to make sure it only returns integer value. Then BEGIN..RETURN the query that satisfy the ask. The difference between total asked and total reimbursed should be zero for full reimbursement and followed by END. To test the function, I returned all columns of candidates table such that the application id should match the result from the function in the WHERE clause.



```sql
--Lichen Liang

USE Recruitments
GO

/**/
IF OBJECT_ID('fnCandidatesReimbursedFully')IS NOT NULL
DROP FUNCTION fnCandidatesReimbursedFully;
GO

CREATE FUNCTION fnCandidatesReimbursedFully()
    RETURNS INT
BEGIN
    RETURN( SELECT R.ApplicationID
            FROM Applications AS A JOIN Reimbursements AS R
                ON A.ApplicationID=R.ApplicationID
            WHERE R.TotalAsked-R.TotalReimbursed = 0)
END
GO

--test
SELECT C.*
FROM Candidates AS C JOIN Applications AS A
    ON C.CandidateID=A.CandidateID
WHERE A.ApplicationID = dbo.fnCandidatesReimbursedFully()
```

	CandidateID	CandidateFname	CandidateLname
1	3	Khalid	Patel

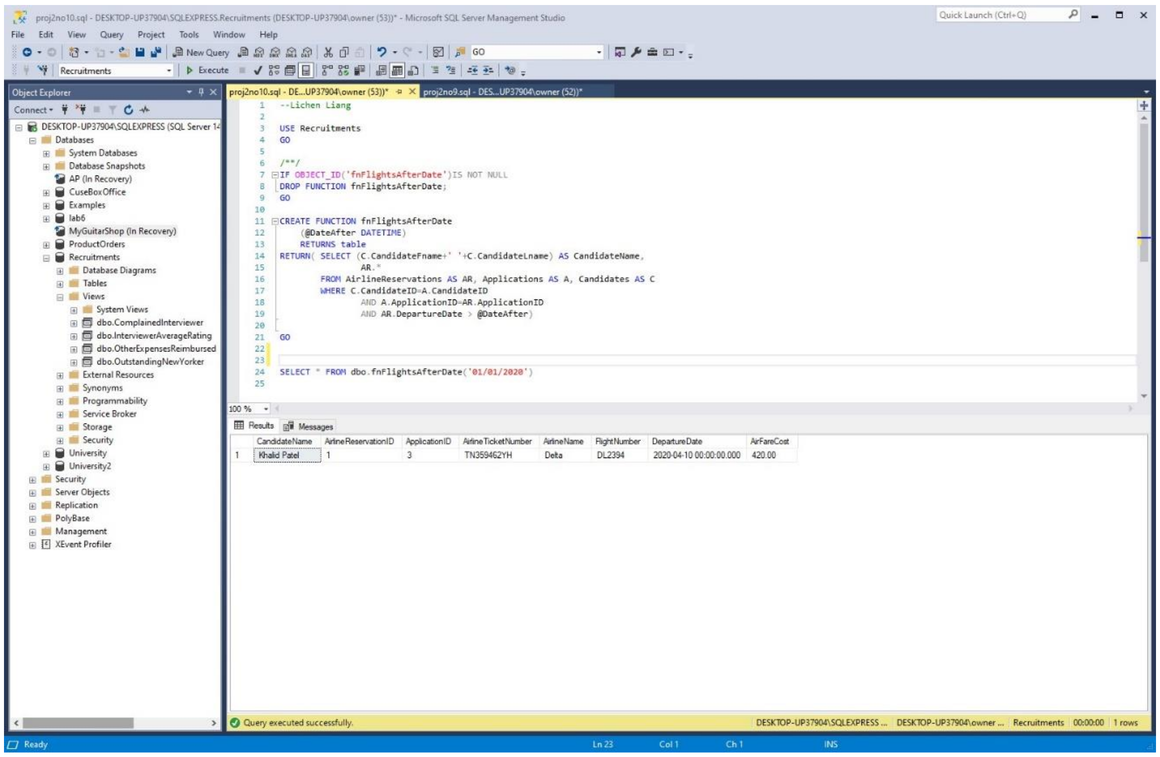
```
Query executed successfully.
```

Figure 12. Candidate who got full reimbursement

2. In this part, I want to find the candidate and any information on the flight if a candidate had a flight after a certain date. This date is set in the parameters.

```
USE Recruitments
GO

/**/
IF OBJECT_ID('fnFlightsAfterDate')IS NOT NULL
DROP FUNCTION fnFlightsAfterDate;
GO

CREATE FUNCTION fnFlightsAfterDate
       (@DateAfter DATETIME)
       RETURNS table
RETURN(SELECT (C.CandidateFname+' '+C.CandidateLname) AS CandidateName,
       AR.*
       FROM AirlineReservations AS AR, Applications AS A, Candidates AS C
       WHERE C.CandidateID=A.CandidateID
              AND A.ApplicationID=AR.ApplicationID
              AND AR.DepartureDate > @DateAfter)

GO

SELECT * FROM dbo.fnFlightsAfterDate('01/01/2020')
```

Like before, check for existence, then create function. Also set the parameter name and data type, and this function should return a whole table. The query selects the candidate name and all flight information. The condition departure date should be after the date specified. Then simply select everything from the function with the parameter specified.



```sql
--Lichen Liang

USE Recruitments
GO

/**/
IF OBJECT_ID('fnFlightsAfterDate')IS NOT NULL
DROP FUNCTION fnFlightsAfterDate;
GO

CREATE FUNCTION fnFlightsAfterDate
    (@DateAfter DATETIME)
    RETURNS table
RETURN( SELECT (C.CandidateFname+' '+C.CandidateLname) AS CandidateName,
                AR.*
        FROM AirlineReservations AS AR, Applications AS A, Candidates AS C
        WHERE C.CandidateID=A.CandidateID
                AND A.ApplicationID=AR.ApplicationID
                AND AR.DepartureDate > @DateAfter)

GO


SELECT * FROM dbo.fnFlightsAfterDate('01/01/2020')
```

```
   CandidateName  AirlineReservationID  ApplicationID  AirlineTicketNumber  AirlineName  FlightNumber  DepartureDate            AirFareCost
1  Khalid Patel   1                     3              TN359462YH           Delta        DL2394        2020-04-10 00:00:00.000  420.00
```

Query executed successfully.

Figure 13. The candidate and his flight information

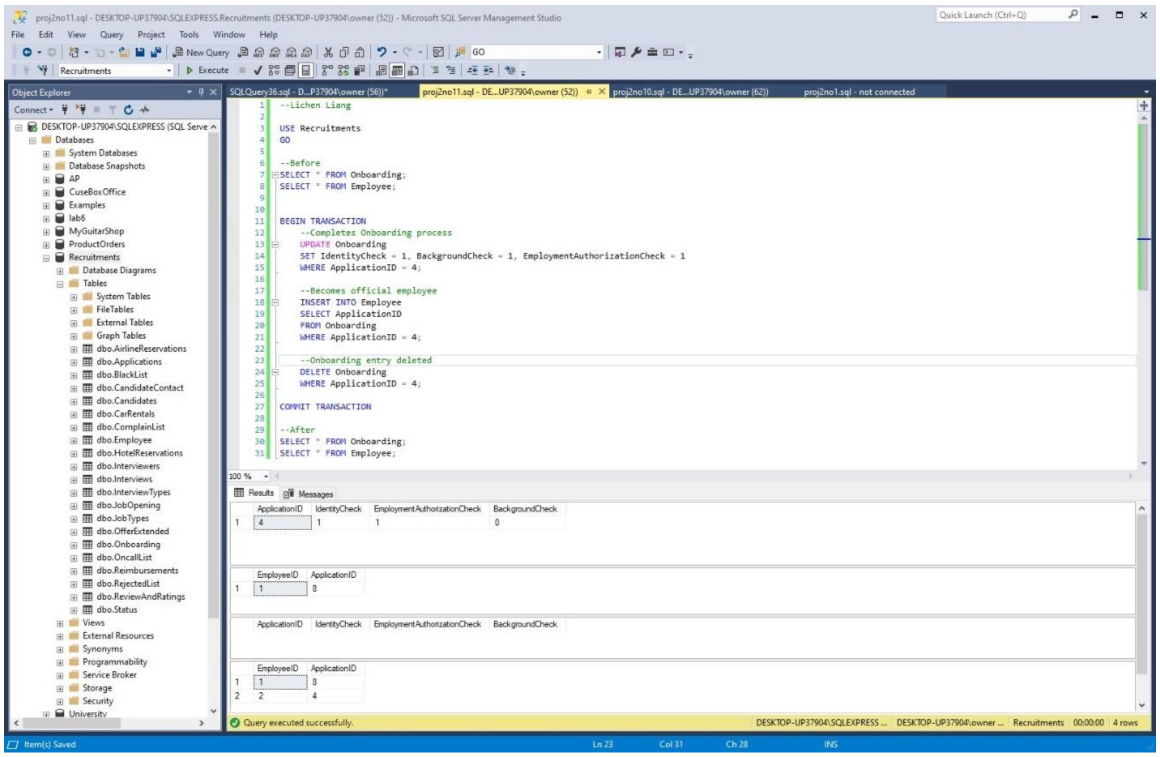
Transaction and Trigger

I created two transactions and one with a trigger. Transactions are used to prevent synchronizations issues.

1. The first transaction completes the onboarding process from candidate 4, therefore check all the columns in the onboarding table, add the application to employees table, and delete the entry from onboarding table.

```
USE Recruitments
GO

--Before
SELECT * FROM Onboarding;
SELECT * FROM Employee;


BEGIN TRANSACTION
       --Completes Onboarding process
       UPDATE Onboarding
       SET IdentityCheck = 1, BackgroundCheck = 1,
       EmploymentAuthorizationCheck = 1
       WHERE ApplicationID = 4;

       --Becomes official employee
       INSERT INTO Employee
       SELECT ApplicationID
       FROM Onboarding
       WHERE ApplicationID = 4;

       --Onboarding entry deleted
       DELETE Onboarding
       WHERE ApplicationID = 4;

COMMIT TRANSACTION

--After
SELECT * FROM Onboarding;
SELECT * FROM Employee;
```

Used BEGIN TRANSACTION and COMMIT TRANSACTION. In between, first UPDATE the onboarding table, then INSERT INTO the employee table, and DELETE the entry in the onboarding table. All of them set conditions in the WHERE clause of that application ID.



```sql
--Lichen Liang

USE Recruitments
GO

--Before
SELECT * FROM Onboarding;
SELECT * FROM Employee;


BEGIN TRANSACTION
    --Completes Onboarding process
    UPDATE Onboarding
    SET IdentityCheck = 1, BackgroundCheck = 1, EmploymentAuthorizationCheck = 1
    WHERE ApplicationID = 4;

    --Becomes official employee
    INSERT INTO Employee
    SELECT ApplicationID
    FROM Onboarding
    WHERE ApplicationID = 4;

    --Onboarding entry deleted
    DELETE Onboarding
    WHERE ApplicationID = 4;

COMMIT TRANSACTION

--After
SELECT * FROM Onboarding;
SELECT * FROM Employee;
```

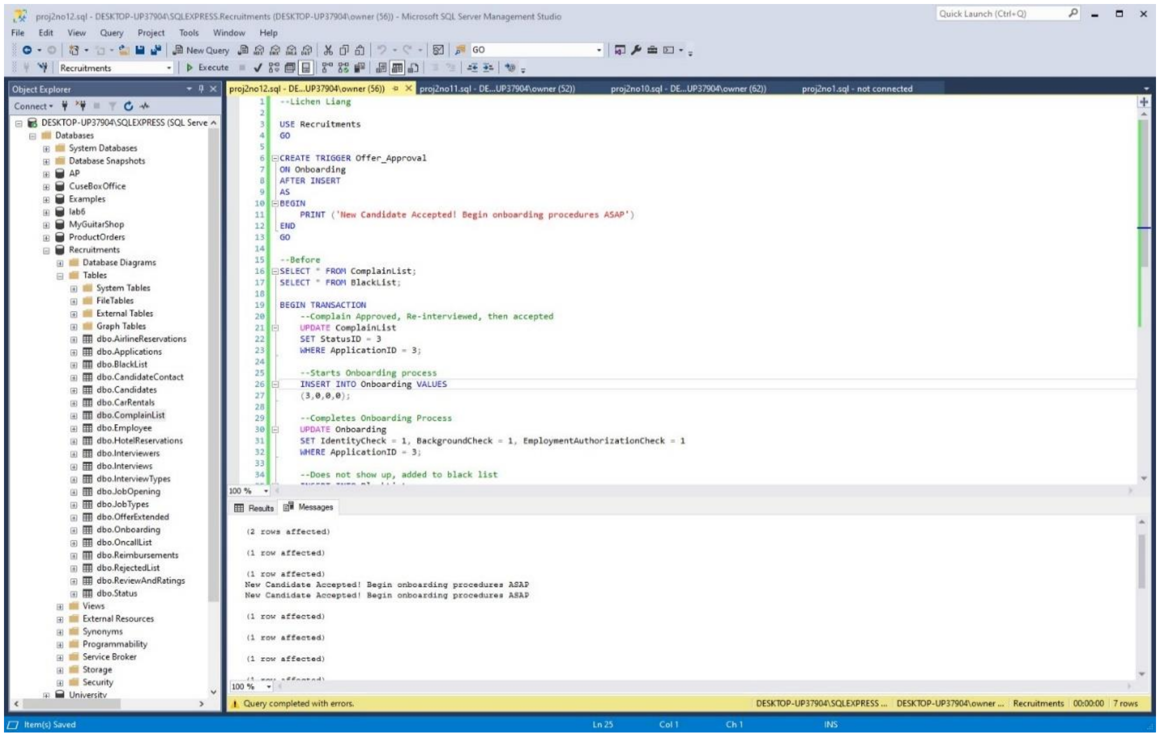
```
   ApplicationID  IdentityCheck  EmploymentAuthorizationCheck  BackgroundCheck
1  4              1              1                             0

   EmployeeID  ApplicationID
1  1           8

   ApplicationID  IdentityCheck  EmploymentAuthorizationCheck  BackgroundCheck

   EmployeeID  ApplicationID
1  1           8
2  2           4

Query executed successfully.
```

Figure 14. Onboarding successful, becomes employee

2. In this transaction, I want the candidate in the complain to re-interview, pass the interview into on-boarding, completes it and does not show up, and eventually end up in the blacklist. Also, use a trigger when a new entry is added to the onboarding table to speed up the onboarding process.

```
USE Recruitments
GO

CREATE TRIGGER Offer_Approval
ON Onboarding
AFTER INSERT
AS
BEGIN
       PRINT ('New Candidate Accepted! Begin onboarding procedures ASAP')
END
GO

--Before
SELECT * FROM ComplainList;
SELECT * FROM BlackList;

BEGIN TRANSACTION
       --Complain Approved, Re-interviewed, then accepted
       UPDATE ComplainList
       SET StatusID = 3
       WHERE ApplicationID = 3;

       --Starts Onboarding process
       INSERT INTO Onboarding VALUES
       (3,0,0,0);

       --Completes Onboarding Process
       UPDATE Onboarding
       SET IdentityCheck = 1, BackgroundCheck = 1,
       EmploymentAuthorizationCheck = 1
       WHERE ApplicationID = 3;

       --Does not show up, added to black list
       INSERT INTO BlackList
       SELECT ApplicationID
       FROM Onboarding
       WHERE ApplicationID = 3;

       --Delete entry from onboarding process
       DELETE Onboarding
       WHERE ApplicationID = 3;

COMMIT TRANSACTION

--After
SELECT * FROM ComplainList;
SELECT * FROM BlackList;
```

Create the trigger using CREATE TRIGGER..ON onboarding table AFTER INSERT into this table and PRINT a message to notify the user to speed up the process. Then BEGIN TRANSACTION that does the job described. UPDATE the status in complain list to complain approved, which result in a re-interview that he passed. Then add this application to the onboarding table with nothing being check at first. This will trigger the trigger. Update it to be checked and to be an employee. He did not show up, so add his application to the blacklist and delete from the onboarding table.

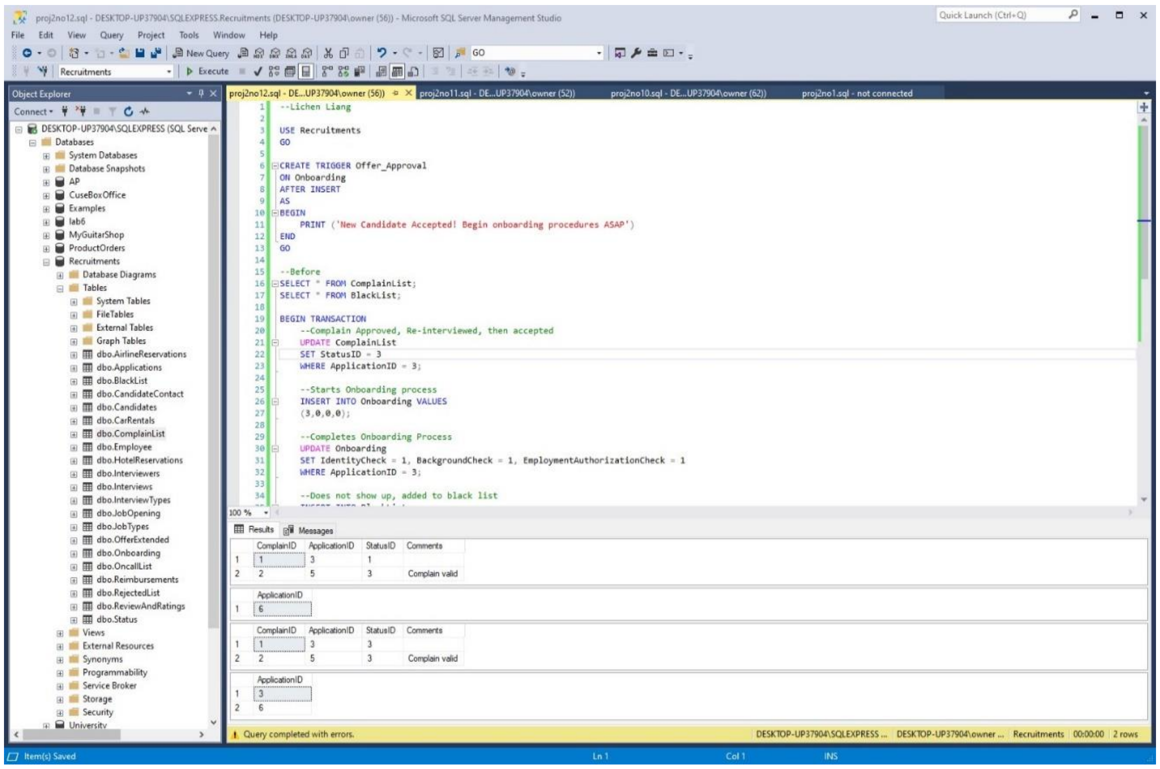


```sql
--Lichen Liang

USE Recruitments
GO

CREATE TRIGGER Offer_Approval
ON Onboarding
AFTER INSERT
AS
BEGIN
    PRINT ('New Candidate Accepted! Begin onboarding procedures ASAP')
END
GO

--Before
SELECT * FROM ComplainList;
SELECT * FROM BlackList;

BEGIN TRANSACTION
    --Complain Approved, Re-interviewed, then accepted
    UPDATE ComplainList
    SET StatusID = 3
    WHERE ApplicationID = 3;

    --Starts Onboarding process
    INSERT INTO Onboarding VALUES
    (3,0,0,0);

    --Completes Onboarding Process
    UPDATE Onboarding
    SET IdentityCheck = 1, BackgroundCheck = 1, EmploymentAuthorizationCheck = 1
    WHERE ApplicationID = 3;

    --Does not show up, added to black list
```

```
(2 rows affected)

(1 row affected)

(1 row affected)
New Candidate Accepted! Begin onboarding procedures ASAP
New Candidate Accepted! Begin onboarding procedures ASAP

(1 row affected)

(1 row affected)

(1 row affected)
```

Query completed with errors.

Figure 15. Transaction and the trigger during the transaction



```sql
--Lichen Liang

USE Recruitments
GO

CREATE TRIGGER Offer_Approval
ON Onboarding
AFTER INSERT
AS
BEGIN
    PRINT ('New Candidate Accepted! Begin onboarding procedures ASAP')
END
GO

--Before
SELECT * FROM ComplainList;
SELECT * FROM BlackList;

BEGIN TRANSACTION
    --Complain Approved, Re-interviewed, then accepted
    UPDATE ComplainList
    SET StatusID = 3
    WHERE ApplicationID = 3;

    --Starts Onboarding process
    INSERT INTO Onboarding VALUES
    (3,0,0,0);

    --Completes Onboarding Process
    UPDATE Onboarding
    SET IdentityCheck = 1, BackgroundCheck = 1, EmploymentAuthorizationCheck = 1
    WHERE ApplicationID = 3;

    --Does not show up, added to black list
```

	ComplainID	ApplicationID	StatusID	Comments
1	1	3	1	
2	2	5	3	Complain valid

	ApplicationID
1	6

	ComplainID	ApplicationID	StatusID	Comments
1	1	3	3	
2	2	5	3	Complain valid

	ApplicationID
1	3
2	6

```
Query completed with errors.    DESKTOP-UP37904\SQLEXPRESS ...  DESKTOP-UP37904\owner ...  Recruitments  00:00:00  2 rows
```

Figure 16. The change in the complain list and blacklist. The onboarding table had change, but had no difference before and after the transaction

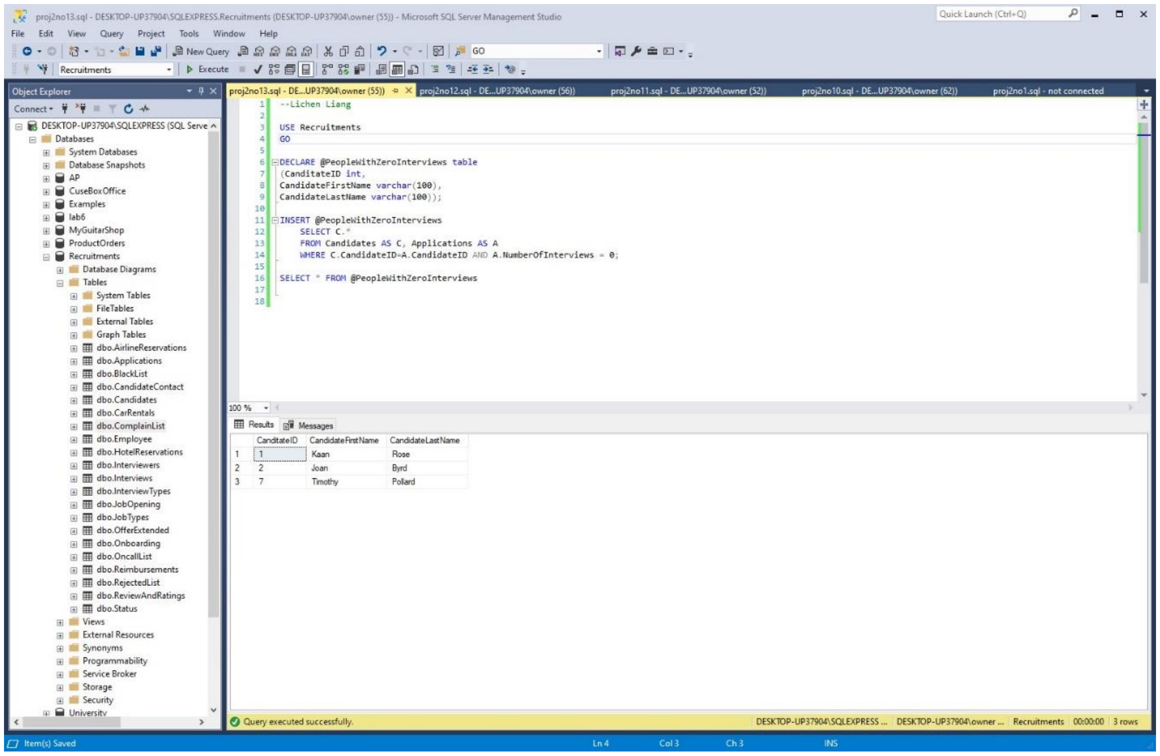
Script

I created two scripts to run, with one of them creating a role and logins with different security levels. The code used to create the database is also a script.

1. First script, I want to find the people who got rejected immediately and had no interviews.

```sql
USE Recruitments
GO

DECLARE @PeopleWithZeroInterviews table
(CanditateID int,
CandidateFirstName varchar(100),
CandidateLastName varchar(100));

INSERT @PeopleWithZeroInterviews
       SELECT C.*
       FROM Candidates AS C, Applications AS A
       WHERE C.CandidateID=A.CandidateID AND A.NumberOfInterviews = 0;

SELECT * FROM @PeopleWithZeroInterviews
```

Use DECLARE to declare a variable that is a table, and its columns and datatype. Then insert into this table by selecting the candidates whose interview times is zero. This will return the people who are rejected at the very beginning.



```sql
--Lichen Liang

USE Recruitments
GO

DECLARE @PeopleWithZeroInterviews table
(CanditateID int,
CandidateFirstName varchar(100),
CandidateLastName varchar(100));

INSERT @PeopleWithZeroInterviews
    SELECT C.*
    FROM Candidates AS C, Applications AS A
    WHERE C.CandidateID=A.CandidateID AND A.NumberOfInterviews = 0;

SELECT * FROM @PeopleWithZeroInterviews
```

	CanditateID	CandidateFirstName	CandidateLastName
1	1	Kaan	Rose
2	2	Joan	Byrd
3	7	Timothy	Pollard

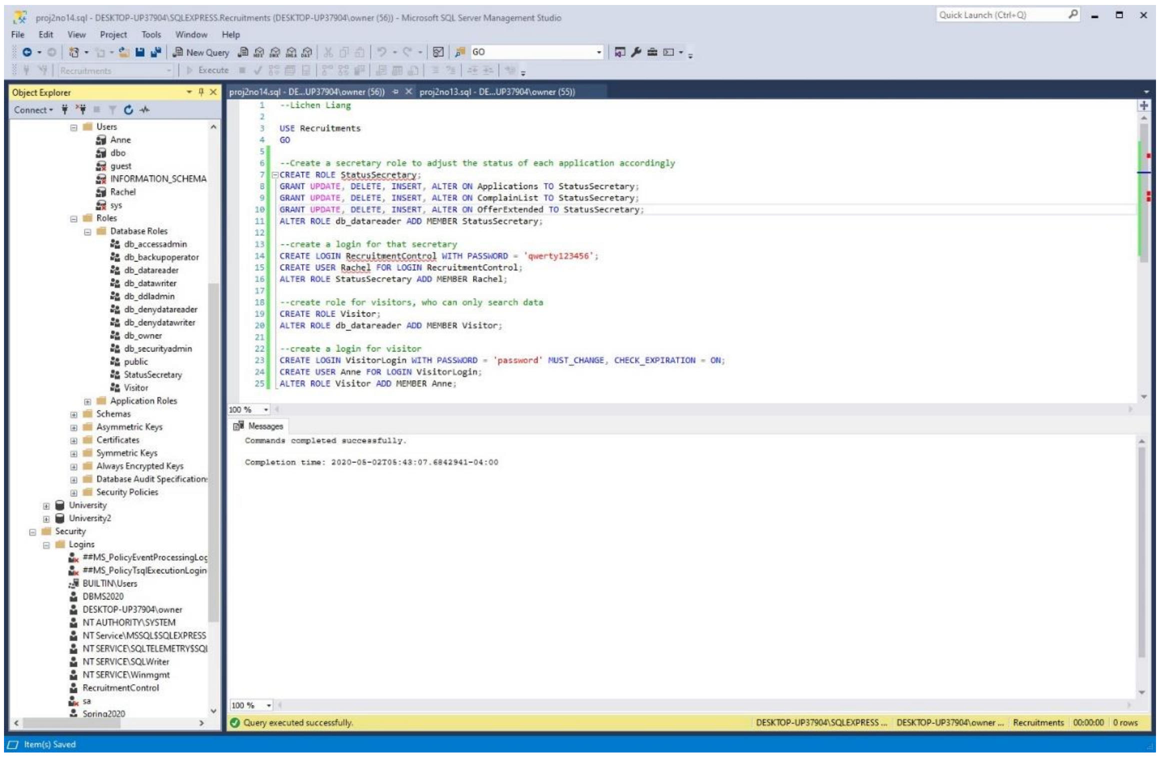
Query executed successfully.

Figure 17. The results match the table used for describing the candidates

2. In this script, I created roles, logins, and assigned users to them. There is a secretary role who can change the data in the database, especially for status change. Also, there is a visitor role who can only query the data and must change the password upon creation.

```
USE Recruitments
GO

--Create a secretary role to adjust the status of each application
accordingly
CREATE ROLE StatusSecretary;
GRANT UPDATE, DELETE, INSERT, ALTER ON Applications TO StatusSecretary;
GRANT UPDATE, DELETE, INSERT, ALTER ON ComplainList TO StatusSecretary;
GRANT UPDATE, DELETE, INSERT, ALTER ON OfferExtended TO StatusSecretary;
ALTER ROLE db_datareader ADD MEMBER StatusSecretary;

--create a login for that secretary
CREATE LOGIN RecruitmentControl WITH PASSWORD = 'qwerty123456';
CREATE USER Rachel FOR LOGIN RecruitmentControl;
ALTER ROLE StatusSecretary ADD MEMBER Rachel;

--create role for visitors, who can only search data
CREATE ROLE Visitor;
ALTER ROLE db_datareader ADD MEMBER Visitor;

--create a login for visitor
CREATE LOGIN VisitorLogin WITH PASSWORD = 'password' MUST_CHANGE,
CHECK_EXPIRATION = ON;
CREATE USER Anne FOR LOGIN VisitorLogin;
ALTER ROLE Visitor ADD MEMBER Anne;
```

Create new roles using CREATE ROLE, then use GRANT..ON..TO to give permissions roles on the tables. Use ALTER ROLE..ADD MEMBER to give the role with query permission. Then CREATE LOGIN…WITH PASSWORD to create a login. Also CREATE USER to create a user FOR LOGIN we just created. The second part is the same, but less permissions and must change password upon creating user using MUST_CHANGE and CHECK_EXPIRATION must be on.



```sql
--Lichen Liang

USE Recruitments
GO

--Create a secretary role to adjust the status of each application accordingly
CREATE ROLE StatusSecretary;
GRANT UPDATE, DELETE, INSERT, ALTER ON Applications TO StatusSecretary;
GRANT UPDATE, DELETE, INSERT, ALTER ON ComplainList TO StatusSecretary;
GRANT UPDATE, DELETE, INSERT, ALTER ON OfferExtended TO StatusSecretary;
ALTER ROLE db_datareader ADD MEMBER StatusSecretary;

--create a login for that secretary
CREATE LOGIN RecruitmentControl WITH PASSWORD = 'qwerty123456';
CREATE USER Rachel FOR LOGIN RecruitmentControl;
ALTER ROLE StatusSecretary ADD MEMBER Rachel;

--create role for visitors, who can only search data
CREATE ROLE Visitor;
ALTER ROLE db_datareader ADD MEMBER Visitor;

--create a login for visitor
CREATE LOGIN VisitorLogin WITH PASSWORD = 'password' MUST_CHANGE, CHECK_EXPIRATION = ON;
CREATE USER Anne FOR LOGIN VisitorLogin;
ALTER ROLE Visitor ADD MEMBER Anne;
```

```
Commands completed successfully.

Completion time: 2020-05-02T05:43:07.6842941-04:00
```

Figure 18. Roles, login, user being created

Conclusion and Future Improvement

Although this database mirrors a real word example, I think there is still room of improvement. There are also many tables that can belong to other databases: Car rental, employee, etc. These tables can be expanded and redesigned to create a bigger network of databases. The design may also affect the speed, but since the data amount is not high, speed factor is undeterminable so far.

Remarks

In this project, we mainly practiced on concepts learned after the first project: views, stored procedures, functions, scripts, and transactions. We also designed and implemented the database using scripts. With the help of realistic data, we can test multiple different scenarios which have been successful. I think this is a great hands-on practice to learn SQL. The only downside is that inserting data is very time consuming with repetitive work.