DBAM Semester Project Work

Airport Management System



**Contributors:**

Ambreen (F2021105009)

Abdullah Imran (F2021105135)

*This document will act as proof of work and also a guide so that seamless communication can take place between the group members, and so that we can track the overall actual progress in real-time.*

# 1. Choosing the topic:

The following topics were shared, almost equally by each person:

1. Customer Relationship Management (CRM)
2. E-Commerce System
3. Supply Chain Management
4. Human Capital or HR
5. Vendor Management System
6. Interior Designing
7. Railway Management System
8. Inventory Control Management
9. Airport and Passenger Management System
10. Hotels and Motels
11. Blood Donation Management

One to two topics were at common ground, and out of these, the Airport management system was chosen as the topic to be worked on.

# 2. Choosing DB software:

So we are going to choose Microsoft SQL Server Studio Management version 19 and here are the reasons for it:

1. We shall be using Visual Studio Code for building the front end and it is also a Microsoft-owned product, which means it will provide a better and seamless developmental experience for us.
2. This will also help us to integrate the front-end and the database in a better way.
3. There is also a personal preference for the software as we have learned the SQL on it.
4. There are also advanced-level security features in SSMS that will make our database more secure.
5. We can also easily tune the efficiency and overall performance of the software with a few tweaks, if necessary.

# 3. Choosing the Backend integration model:

# 4: Listing the entities:

1. Passenger (Passenger ID, Name, Contact, Passport/ID, **LuggageID**) *→ Make PassengerContact table during normalization*
2. Flight(Flight ID, Departure Time(+date), Arrival Time(+date), **TicketID**)
3. Airport(Airport ID, Name, Location, Airport\_Type (passenger, airforce, domestic/international), **FlightID, SecurityID, TrafficControlID, FuelingStationID, RunwayID**)
4. Ticket(Ticket ID, Ticket\_Type(Economy, First Class, Business Class), Price, Purchase Date, **PassengerID**)
5. Airline(Airline ID, Airline Name, Airline\_Type(AirBus, 777, etc, FuelCapacity), AirlineStatus(Banned, Operational, under watch, etc), **FlightID**)
6. Security(Security ID, Name, SecurityDepartment(Customs, Clearance, Airline), AllocatedArea(BagCheck, PassengerCheck, Immigration, etc.))
7. Pilot(Pilot ID, Name, HoursExperience, LicenseNo, AircraftAllowed(Aircrafts allowed to operate)
8. TrafficControl(TrafficControlID, Name, TowerNo, TowerName)
9. Luggage(LuggageID, Type(Carry-on/hand-carry, boarded), weight)
10. FuelingStation(FuelingStationID, Name, capacity, AircraftsAttended, FuelType(AvGas, JetType A/A1/B))
11. AirportRunway(RunwayID, Length, LightingSystems(Edge, End, Centre, PathIndicators))

*Bold ones highlight the foreign keys.*

**Explanation for some attributes:**

Lighting Systems are laid out to guide the airplane when it lands and flies, where to land, where to fly from, where to park, where are the boundaries of the runaway, and so on.

## 4.1: Building Relations:

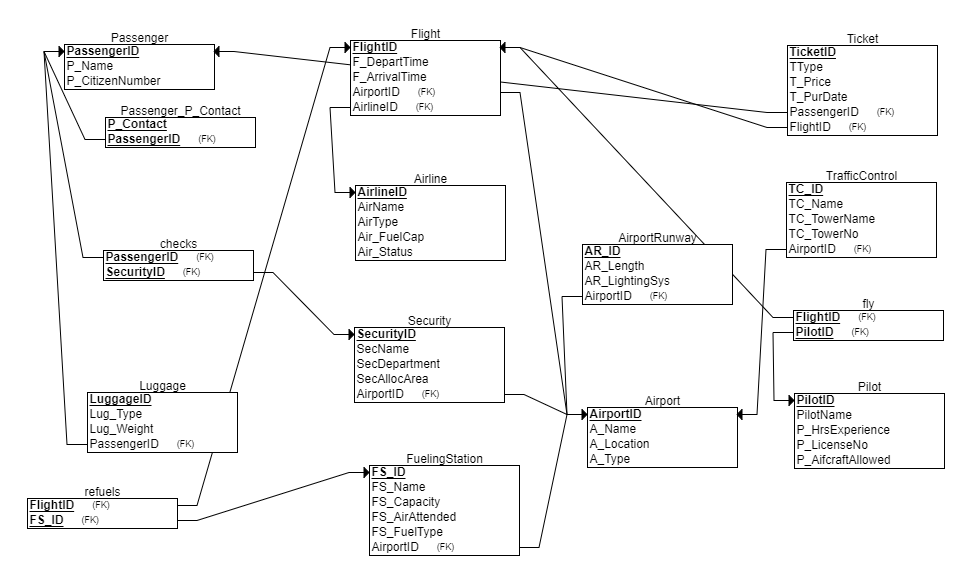
1. Passenger - Ticket: (One to Many)
2. Flight - Airport (One to Many)
3. Ticket - Flight (Many to One)
4. Flight - Airline (Many to One)
5. Security - Airport (Many to One)
6. Pilot-Flight (Many to Many ) [Make a PilotFlightAssociation table during normalization (PilotID, FlightID)]
7. TrafficControl - Airport (Many to One)
8. Luggage - Passenger (One to Many)
9. FuelingStation - Airport (Many to One)
10. Fueling Station - Flight (Many to Many) [Make FlightFuelingStationAssociation Table during normalization (FlightID, FuelingStationID)]
11. AirportRunway - Airport(Many to One)
12. Passenger - Security (Many to Many) [Make PassengerSecurityAssociation table during normalization (PassengerID, SecurityID, CheckDateTime, Status)]

## 4.2 Entity Relationship Diagram:

ERD (Yikes!)

## 

## 4.3: Relational Schema Diagram:



## 4.4: Normalization:

1. Passenger (Passenger ID, Name, Contact, Passport/ID, **LuggageID**) *→ Make PassengerContact table during normalization*
2. PassengerContact(P\_Contact, **PassengerID**)
3. Flight(Flight ID, Departure Time(+date), Arrival Time(+date), **TicketID**)
4. Airport(Airport ID, Name, Location, Airport\_Type (passenger, airforce, domestic/international), **FlightID, SecurityID, TrafficControlID, FuelingStationID, RunwayID**)
5. Ticket(Ticket ID, Ticket\_Type(Economy, First Class, Business Class), Price, Purchase Date, **PassengerID**)
6. Airline(Airline ID, Airline Name, Airline\_Type(AirBus, 777, etc, FuelCapacity), AirlineStatus(Banned, Operational, under watch, etc), **FlightID**)
7. Security(Security ID, Name, SecurityDepartment(Customs, Clearance, Airline), AllocatedArea(BagCheck, PassengerCheck, Immigration, etc.))
8. PassengerSecurityAssociation(PassengerID, SecurityID, CheckDateTime, Status)
9. Pilot(Pilot ID, Name, HoursExperience, LicenseNo, AircraftAllowed(Aircrafts allowed to operate)
10. PilotFlightAssociation(PilotID, FlightID)
11. TrafficControl(TrafficControlID, Name, TowerNo, TowerName)
12. Luggage(LuggageID, Type(Carry-on/hand-carry, boarded), weight)
13. FuelingStation(FuelingStationID, Name, capacity, AircraftsAttended, FuelType(AvGas, JetType A/A1/B))
14. FlightFuelingStationAssociation(FlightID, FuelingStationID)
15. AirportRunway(RunwayID, Length, LightingSystems(Edge, End, Centre, PathIndicators))

**Bold** represents the foreign key. Blue ones represent the table that has been created as a result of the normalization process.

# 5. Authorization:

The authorization will be based on the created tables and the users that will use this database. So we will first need to create the logins of those users whom we target as DB users.

## 5.1: User Login, roles, privileges, and authorization:

So, for simplicity purposes, we are going to create 4 types of user logins, one that will be used by the admins, one by the airport’s staff, one by the security, and the other for passengers or anyone outside the airport’s premises whom we want to target.

### 5.1.1: Admins:

-- Creating Admin Role:

CREATE LOGIN AdminLogin WITH PASSWORD = 'Airport321$%';

-- Creating Users who are going to use an admin login

CREATE USER CoAdmin1 FOR LOGIN AdminLogin;

CREATE USER CoAdmin2 FOR LOGIN AdminLogin;

CREATE USER AirportDBA FOR LOGIN AdminLogin;

-- Create DBARole with extensive privileges

CREATE ROLE DBARole;

-- Grant extensive privileges to DBARole

GRANT ALTER ANY ASSEMBLY TO DBARole;

GRANT ALTER ANY DATASPACE TO DBARole;

GRANT ALTER ANY FULLTEXT CATALOG TO DBARole;

GRANT ALTER ANY DATABASE TO DBARole;

GRANT CREATE ANY DATABASE TO DBARole;

GRANT CREATE PROCEDURE TO DBARole;

GRANT CREATE SCHEMA TO DBARole;

GRANT CREATE TABLE TO DBARole;

GRANT CREATE VIEW TO DBARole;

GRANT CREATE TRIGGER TO DBARole;

GRANT ALTER ANY LOGIN TO DBARole;

GRANT ALTER ANY USER TO DBARole;

GRANT VIEW DEFINITION TO DBARole;

GRANT CONNECT SQL TO DBARole;

GRANT EXECUTE TO DBARole;

GRANT SHOWPLAN TO DBARole;

GRANT VIEW SERVER STATE TO DBARole;

GRANT ALTER ANY CONNECTION TO DBARole;

-- Revoke permissions from public role (customize as necessary)

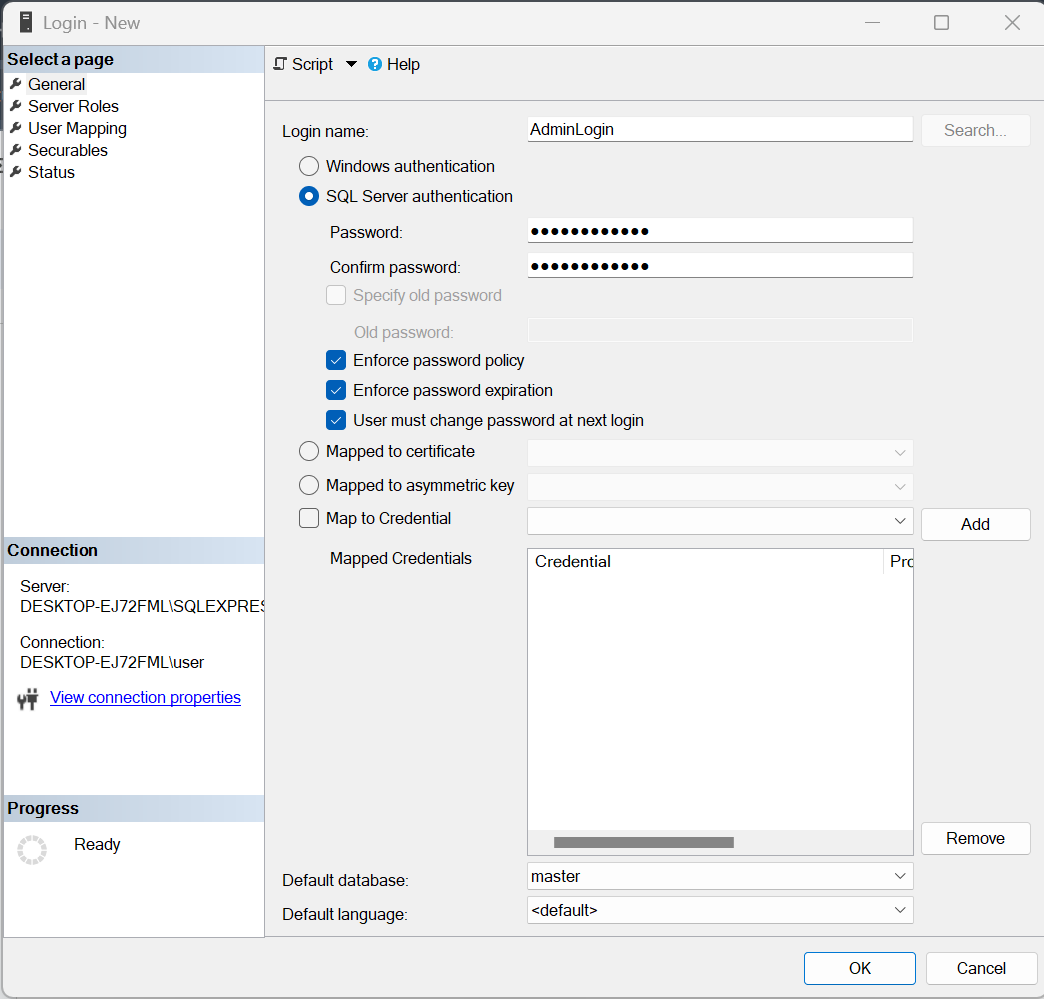
REVOKE CONNECT SQL FROM public;

-- Assign DBARole to users

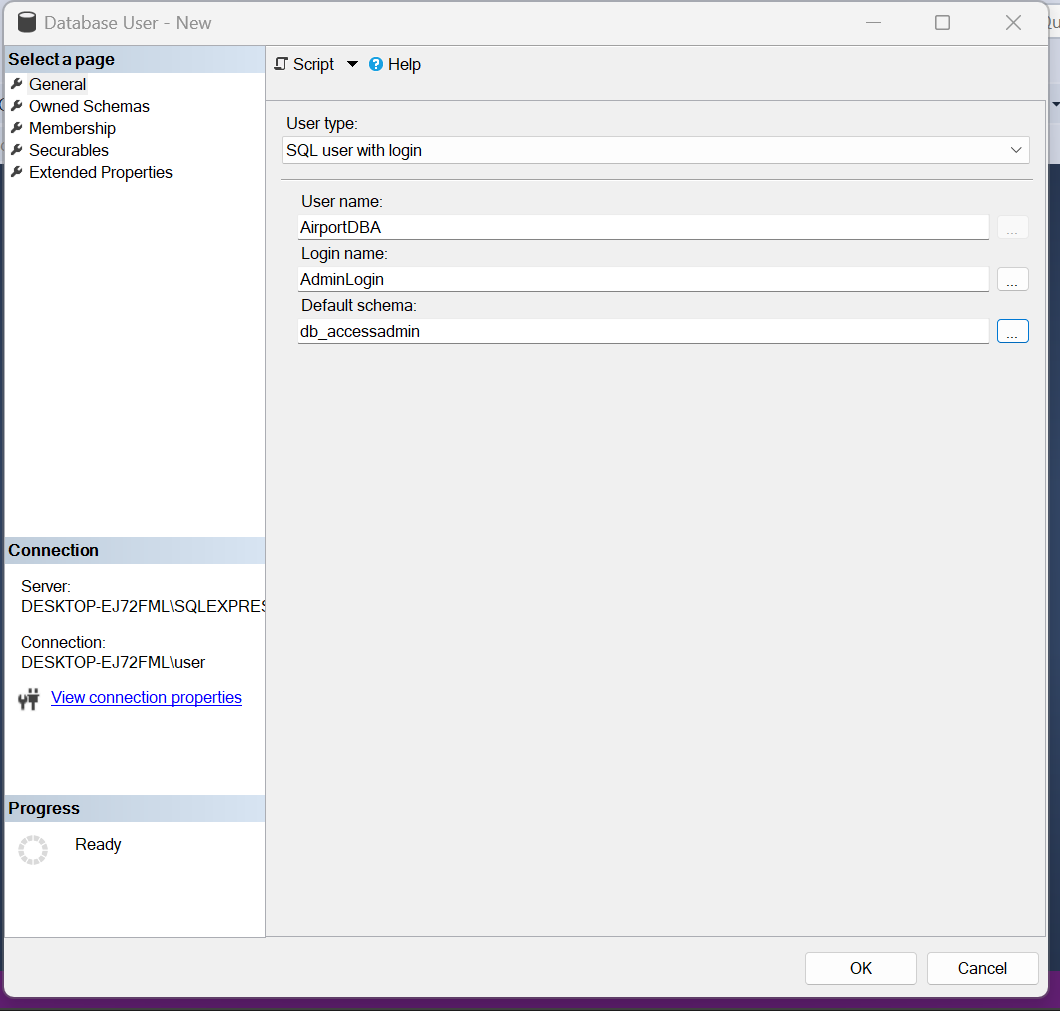
ALTER ROLE DBARole ADD MEMBER coadmin1;

ALTER ROLE DBARole ADD MEMBER coadmin2;

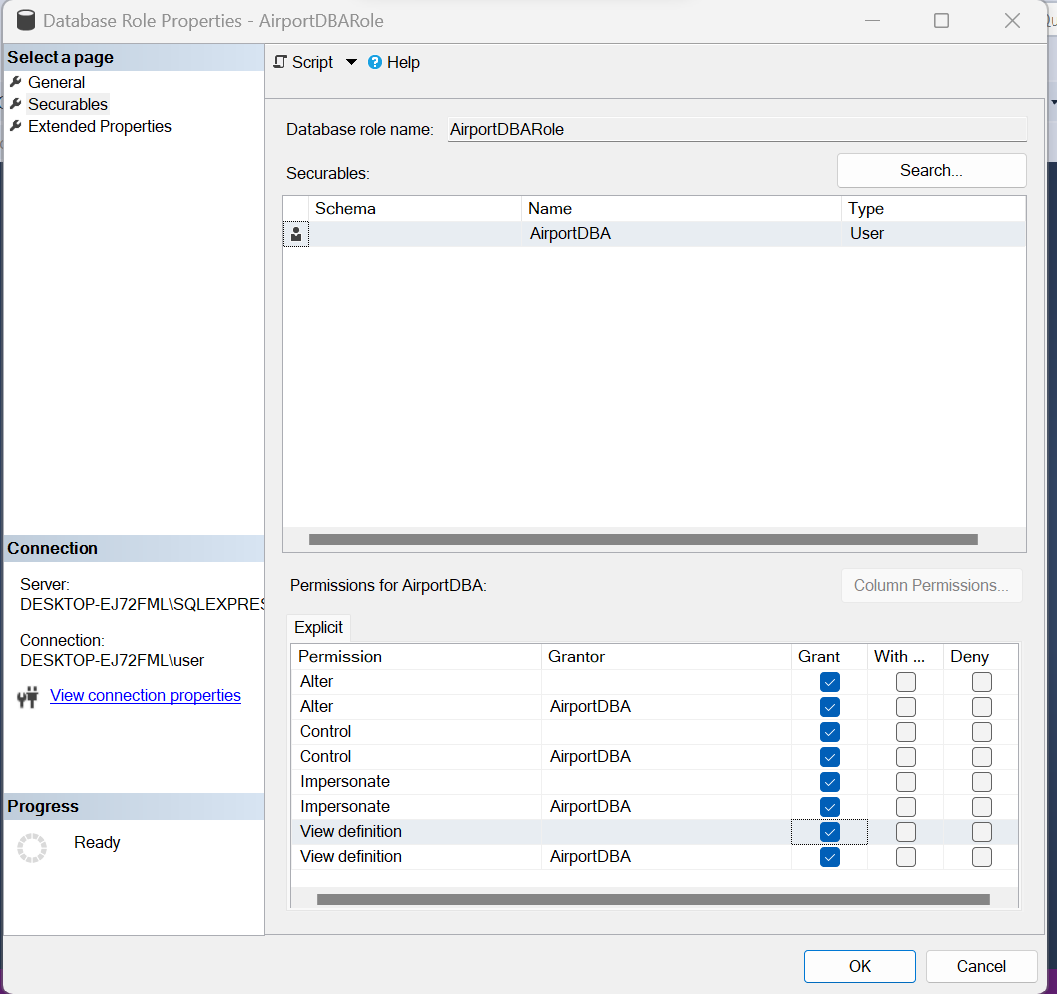
ALTER ROLE DBARole ADD MEMBER AirportDBA;

Creating Admin Login Via GUI:  
  


I was getting a lot of errors. Even after a zillion tries, I could not create multiple users for one login that would be accessing one database. So, I am invoking plan B here, and just creating one general user for this purpose called “AirportDBA”.



AirportDBARole for the AirportDBA:



### 5.1.2: Airport Staff:

-- Create AirportStaffLogin

CREATE LOGIN AirportStaffLogin WITH PASSWORD = 'AirportStaff987%$';

-- Create users for the AirportStaffLogin and associate them with roles

CREATE USER PilotUser FOR LOGIN AirportStaffLogin;

CREATE USER AirlineUser FOR LOGIN AirportStaffLogin;

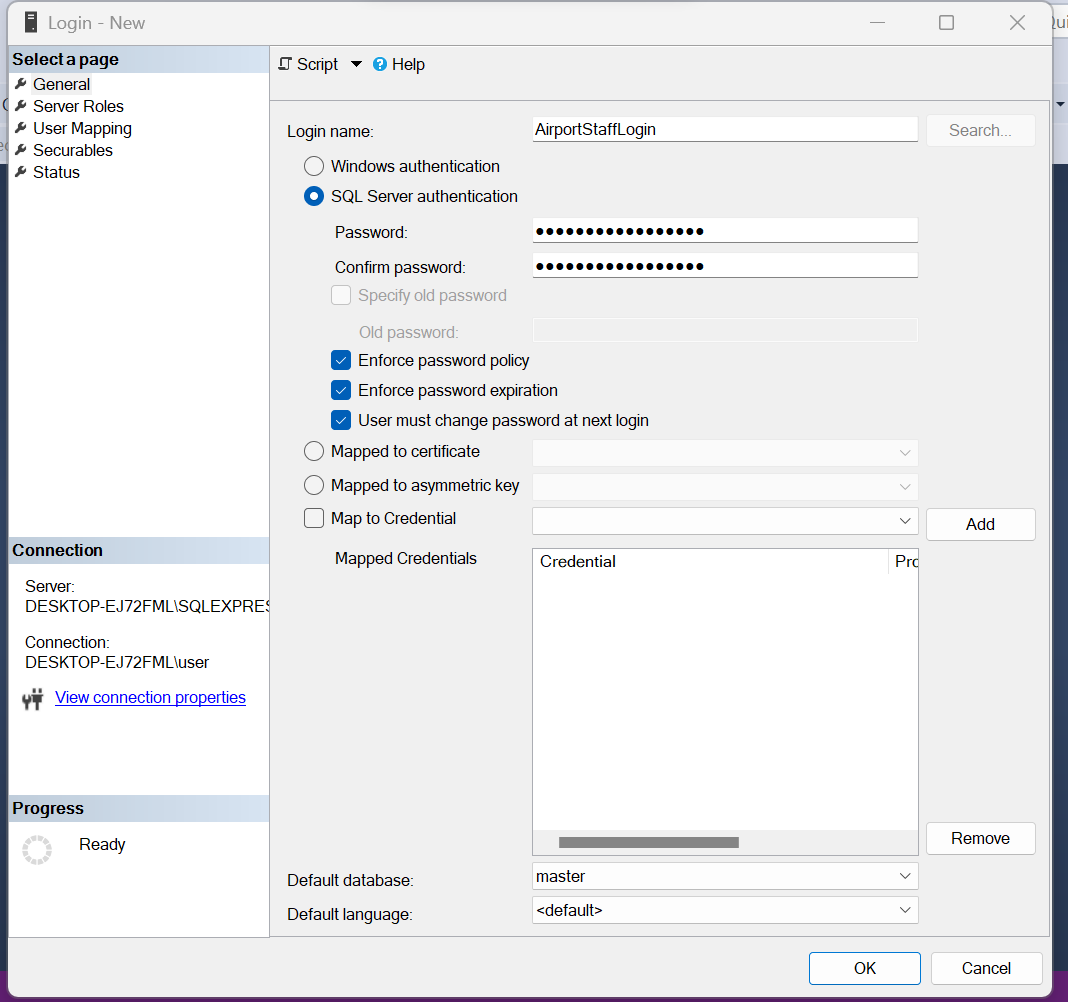
CREATE USER RunwayManagerUser FOR LOGIN AirportStaffLogin;

CREATE USER TrafficControlUser FOR LOGIN AirportStaffLogin;

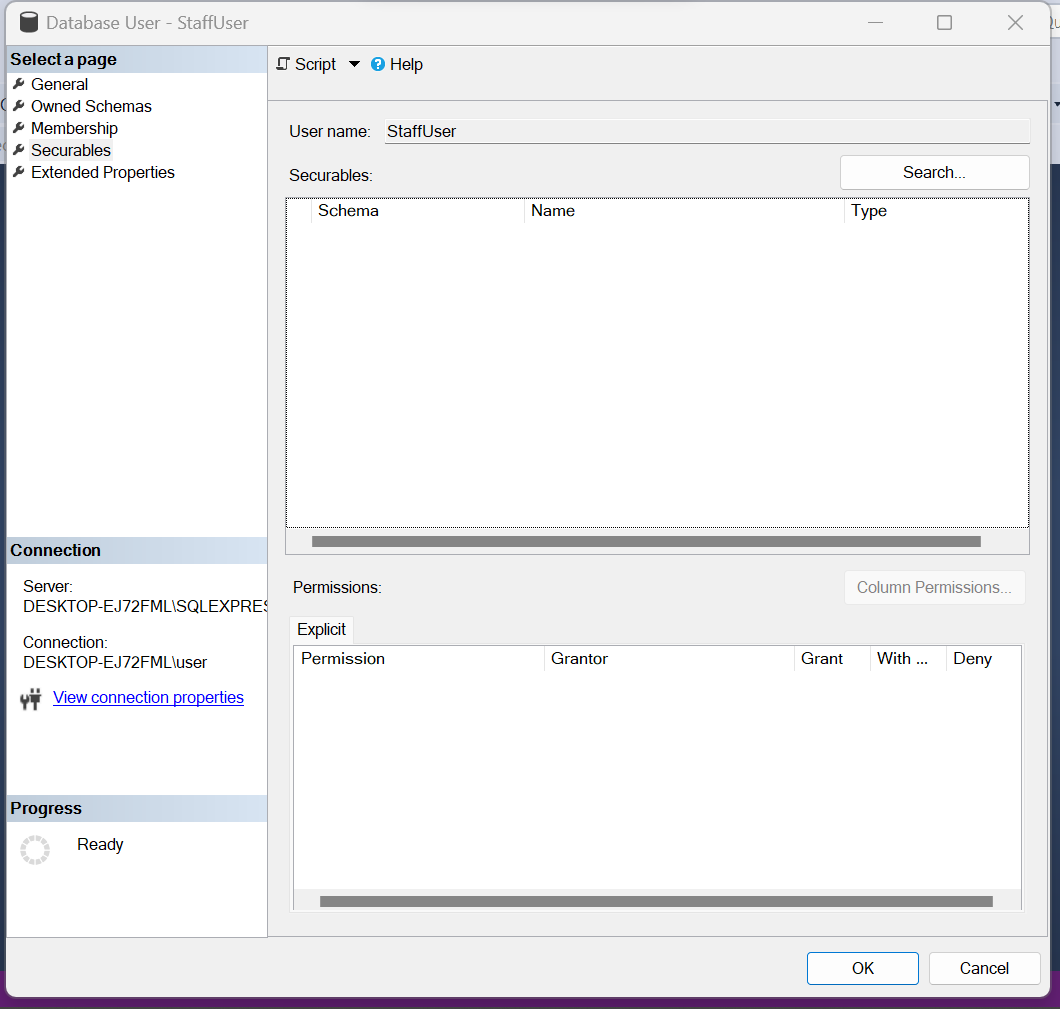
CREATE USER LuggageDepartmentUser FOR LOGIN AirportStaffLogin;

CREATE USER FuelingManagerUser FOR LOGIN AirportStaffLogin;

Creating a GUI login: AirportStaffLogin



Creating a user StaffUser for the AirportStaffLogin:

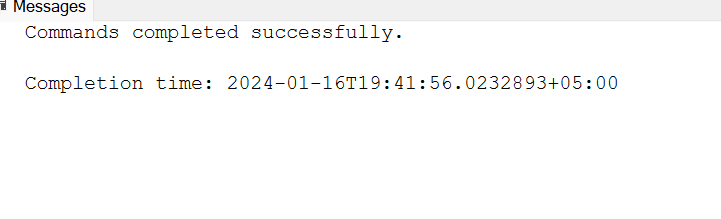


For ease, use the statements to create and assign the role:  
  
use AirportManagementSystem;

create role StaffRole;

GRANT SELECT, UPDATE, INSERT ON SCHEMA::dbo TO StaffRole;

ALTER ROLE StaffRole ADD MEMBER StaffUser;



#### 5.1.2.1: Airline:

-- Airline:

-- Grant privileges on tables

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Passenger TO AirlineRole;

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.PassengerContact TO AirlineRole;

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Flight TO AirlineRole;

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Ticket TO AirlineRole;

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Airline TO AirlineRole;

-- Grant privileges to create views

GRANT CREATE VIEW TO AirlineRole;

-- Grant SELECT privileges on additional tables

GRANT SELECT ON dbo.TrafficControl TO AirlineRole;

GRANT SELECT ON dbo.Pilot TO AirlineRole;

GRANT SELECT ON dbo.Security TO AirlineRole;

GRANT SELECT ON dbo.Airport TO AirlineRole;

-- Assign AirlineUser to AirlineRole

ALTER ROLE AirlineRole ADD MEMBER AirlineUser;

#### 5.1.2.2: Pilot

-- Pilot:

-- Create PilotRole

CREATE ROLE PilotRole;

-- Grant privileges on tables

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Pilot TO PilotRole;

GRANT UPDATE, SELECT ON dbo.PilotFlightAssociation TO PilotRole;

-- Grant SELECT privileges on additional tables

GRANT SELECT ON dbo.Airport TO PilotRole;

GRANT SELECT ON dbo.TrafficControl TO PilotRole;

GRANT SELECT ON dbo.Luggage TO PilotRole;

GRANT SELECT ON dbo.FuelingStation TO PilotRole;

GRANT SELECT ON dbo.AirportRunway TO PilotRole;

GRANT SELECT ON dbo.Security TO PilotRole;

– Granting the role to the Pilot.

ALTER ROLE PilotRole ADD MEMBER PilotUser;

#### 5.1.2.3: Runway Manager:

-- Runway:

-- Create RunwayManager role

CREATE ROLE RunwayManager;

-- Grant privileges on tables

GRANT INSERT, UPDATE, DELETE, CREATE VIEW ON dbo.AirportRunway TO RunwayManager;

-- Grant SELECT privileges on additional tables

GRANT SELECT ON dbo.Flight TO RunwayManager;

GRANT SELECT ON dbo.Airline TO RunwayManager;

GRANT SELECT ON dbo.TrafficControl TO RunwayManager;

GRANT SELECT ON dbo.Pilot TO RunwayManager;

-- Create RunwayManagerUser and assign to RunwayManager role

CREATE USER RunwayManagerUser FOR LOGIN AirportStaffLogin;

ALTER ROLE RunwayManager ADD MEMBER RunwayManagerUser;

-- Assign RunwayManagerUser to RunwayManager role

ALTER ROLE RunwayManager ADD MEMBER RunwayManagerUser;

#### 5.1.2.4: Traffic Control:

-- Traffic:

-- Create TrafficControlRole

CREATE ROLE TrafficControlRole;

-- Grant privileges on tables

GRANT INSERT, UPDATE, DELETE, CREATE VIEW ON dbo.TrafficControl TO TrafficControlRole;

-- Grant SELECT privileges on additional tables

GRANT SELECT ON dbo.Airline TO TrafficControlRole;

GRANT SELECT ON dbo.Flight TO TrafficControlRole;

GRANT SELECT ON dbo.Pilot TO TrafficControlRole;

ALTER ROLE TrafficControlRole ADD MEMBER TrafficControlUser;

#### 5.1.2.5: Luggage Department

-- Create LuggageDeptRole

CREATE ROLE LuggageDeptRole;

-- Grant privileges on tables

GRANT INSERT, UPDATE, DELETE, CREATE VIEW, ALTER, DROP ON dbo.Luggage TO LuggageDeptRole;

-- Grant SELECT privileges on additional tables

GRANT SELECT ON dbo.Airline TO LuggageDeptRole;

GRANT SELECT ON dbo.Flight TO LuggageDeptRole;

GRANT SELECT ON dbo.Passenger TO LuggageDeptRole;

GRANT SELECT ON dbo.Airport TO LuggageDeptRole;

ALTER ROLE LuggageDeptRole ADD MEMBER LuggageDepartmentUser;

#### 5.1.2.6: Fueling Station

-- Fueling Station:

-- Create FuelingDepartRole

CREATE ROLE FuelingDepartRole;

-- Grant privileges on tables

GRANT INSERT, UPDATE, DELETE, CREATE VIEW ON dbo.FuelingStation TO FuelingDepartRole;

-- Grant privileges on a specific table

GRANT UPDATE, SELECT ON dbo.FlightFuelingStationAssociation TO FuelingDepartRole;

-- Grant SELECT privileges on additional tables

GRANT SELECT ON dbo.Airport TO FuelingDepartRole;

GRANT SELECT ON dbo.Airline TO FuelingDepartRole;

GRANT SELECT ON dbo.Flight TO FuelingDepartRole;

ALTER ROLE FuelingDepartRole ADD MEMBER FuelingManagerUser;

#### 5.1.3: Security

-- Security:

-- Create SecurityLogin

CREATE LOGIN SecurityLogin WITH PASSWORD = 'SecurityPassword321%@';

-- Create SecurityUser

CREATE USER SecurityUser FOR LOGIN SecurityLogin;

-- Create SecurityRole

CREATE ROLE SecurityRole;

-- Grant all privileges on Security table

GRANT ALL ON dbo.Security TO SecurityRole;

-- Grant SELECT and UPDATE privileges on PassengerSecurityAssociation table

GRANT SELECT, UPDATE ON dbo.PassengerSecurityAssociation TO SecurityRole;

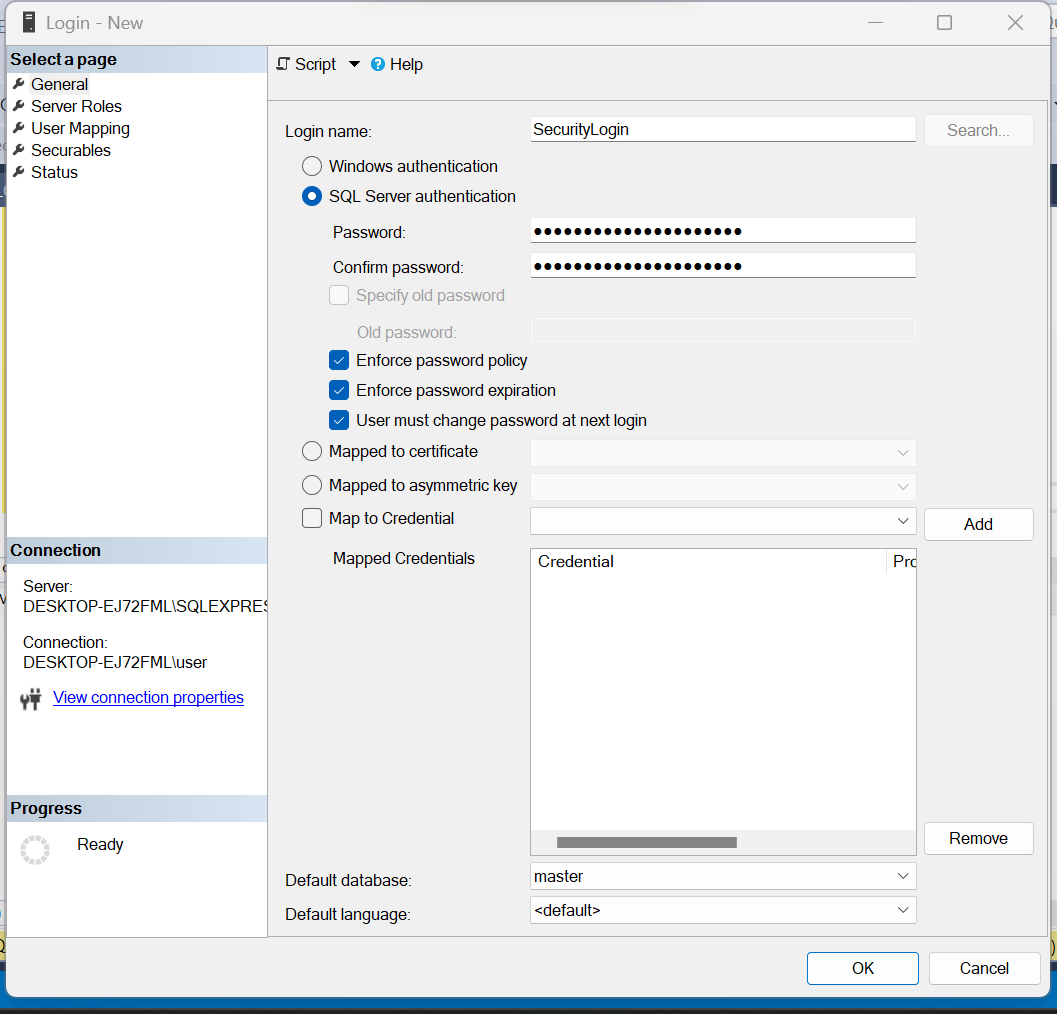
-- Grant SELECT privileges on all tables

GRANT SELECT ON SCHEMA::dbo TO SecurityRole;

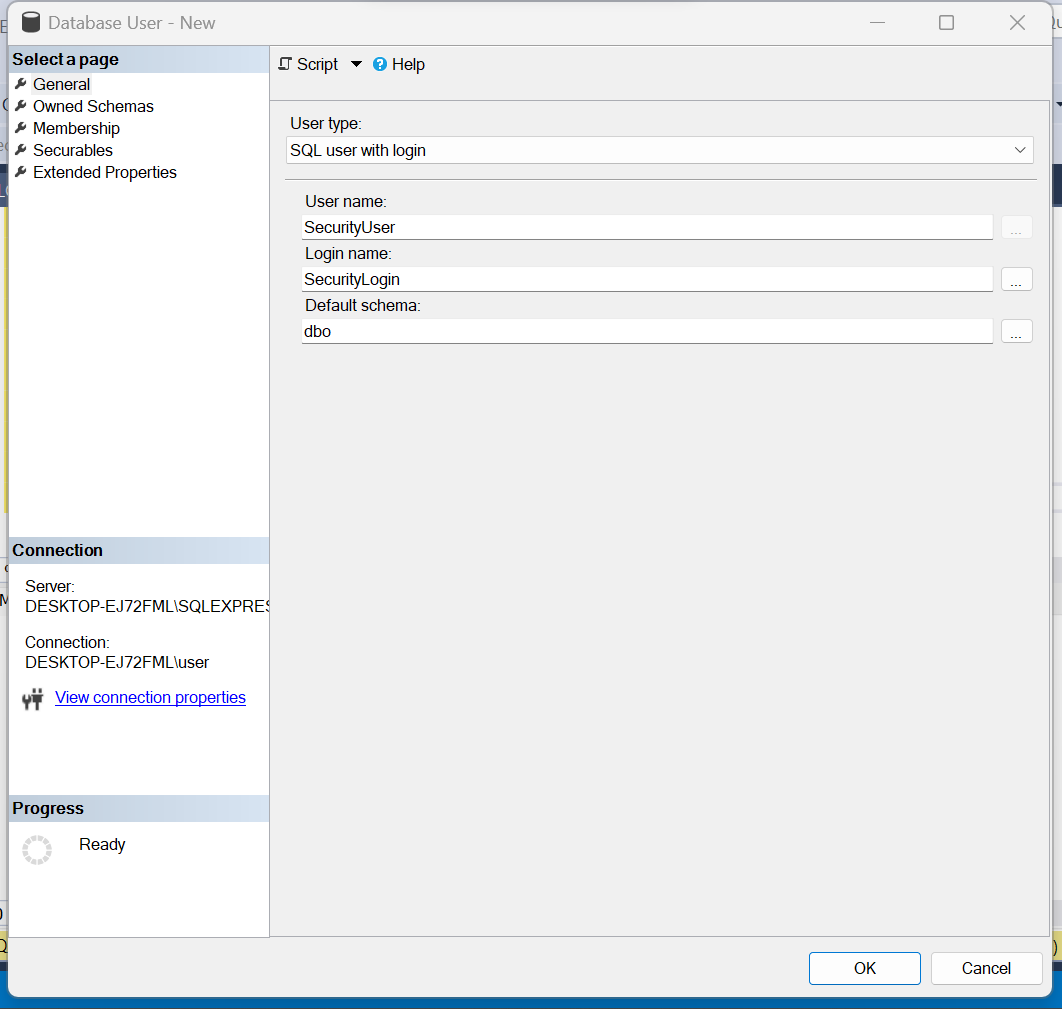
-- Assign SecurityRole to SecurityUser

ALTER ROLE SecurityRole ADD MEMBER SecurityUser;

Creating SecurityLogin using GUI:



Creating the SecurityUser for SecurityLogin:



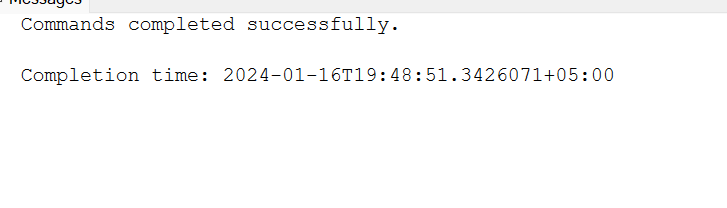
Creating the SecurityRole:

USE AirportManagementSystem;

CREATE ROLE SecurityRole;

GRANT SELECT ON SCHEMA::dbo TO SecurityRole;

ALTER ROLE SecurityRole ADD MEMBER SecurityUser;



### 5.1.4: Passenger:

-- Passenger:

-- Create PassengerLogin

CREATE LOGIN PassengerLogin WITH PASSWORD = 'PassengerPassword';

-- Create Passenger Users

CREATE USER PassengerUser1 FOR LOGIN PassengerLogin;

CREATE USER PassengerUser2 FOR LOGIN PassengerLogin;

CREATE USER PassengerUser3 FOR LOGIN PassengerLogin;

CREATE USER PassengerUser4 FOR LOGIN PassengerLogin;

CREATE USER PassengerUser5 FOR LOGIN PassengerLogin;

-- Create PassengerRole

CREATE ROLE PassengerRole;

-- Grant privileges on tables

GRANT INSERT, UPDATE, DELETE ON dbo.Passenger TO PassengerRole;

GRANT INSERT, UPDATE, DELETE ON dbo.PassengerContact TO PassengerRole;

GRANT INSERT, UPDATE, DELETE ON dbo.Luggage TO PassengerRole;

-- Grant SELECT and UPDATE privileges on Ticket table

GRANT SELECT, UPDATE ON dbo.Ticket TO PassengerRole;

-- Grant SELECT privileges on additional tables

GRANT SELECT ON dbo.Airline TO PassengerRole;

GRANT SELECT ON dbo.Flight TO PassengerRole;

GRANT SELECT ON dbo.Airport TO PassengerRole;

-- Assign PassengerRole to all five users

ALTER ROLE PassengerRole ADD MEMBER PassengerUser1;

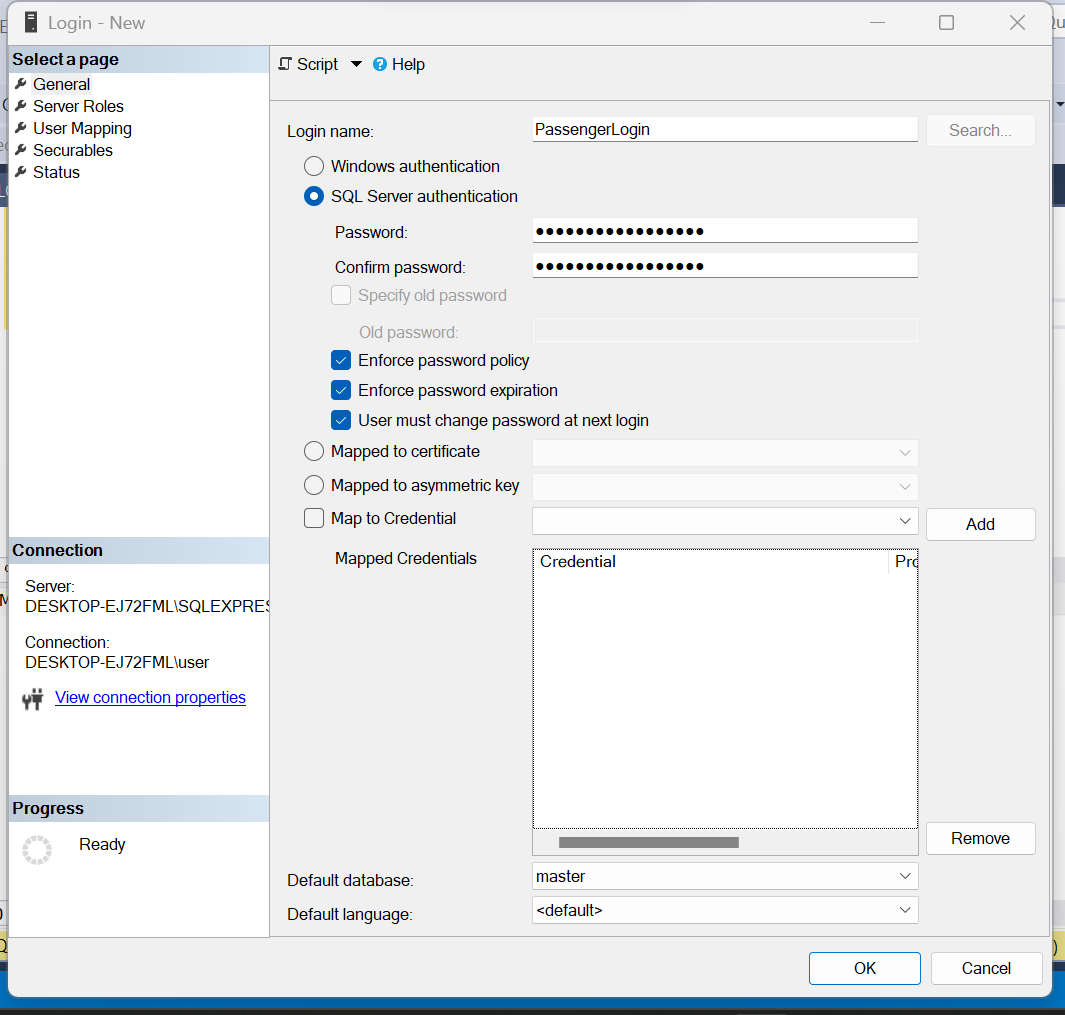
ALTER ROLE PassengerRole ADD MEMBER PassengerUser2;

ALTER ROLE PassengerRole ADD MEMBER PassengerUser3;

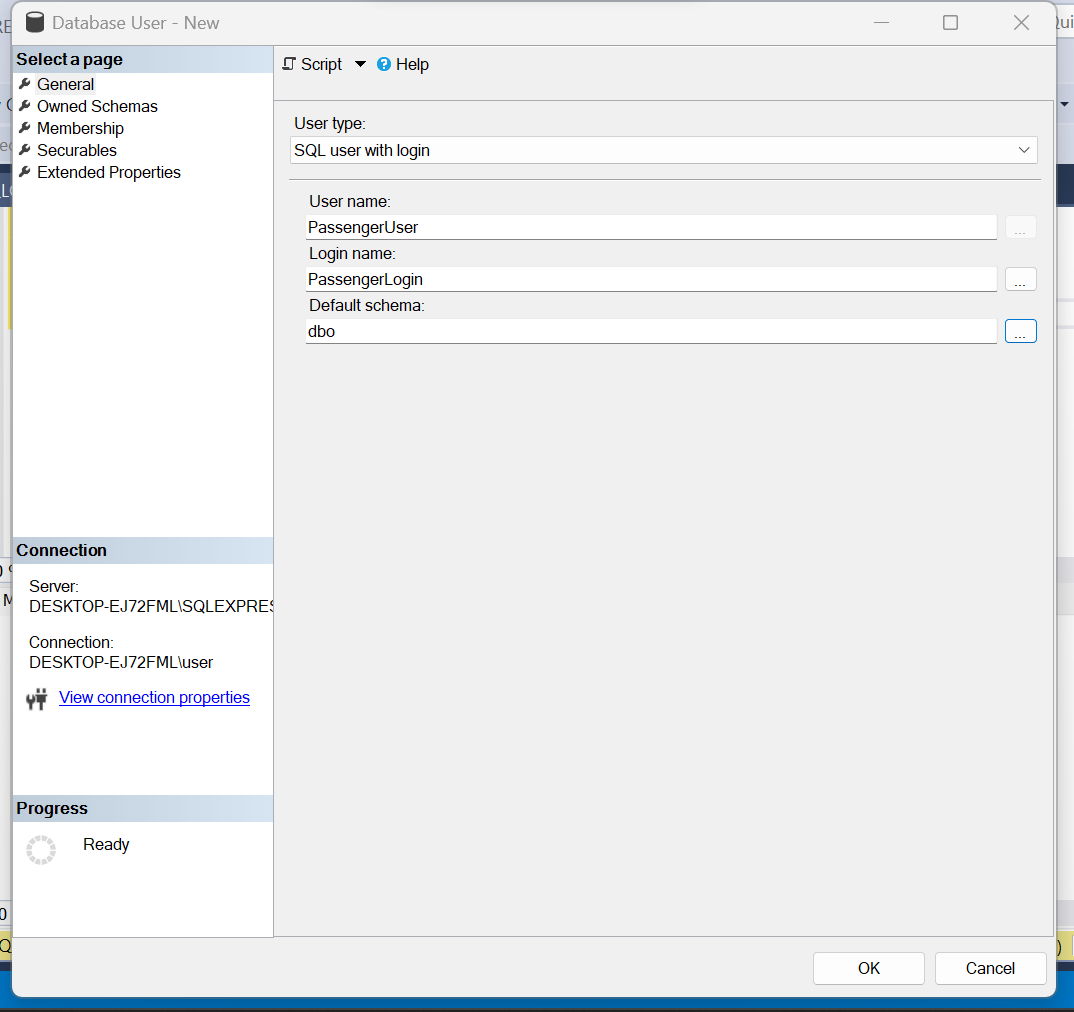
ALTER ROLE PassengerRole ADD MEMBER PassengerUser4;

ALTER ROLE PassengerRole ADD MEMBER PassengerUser5;

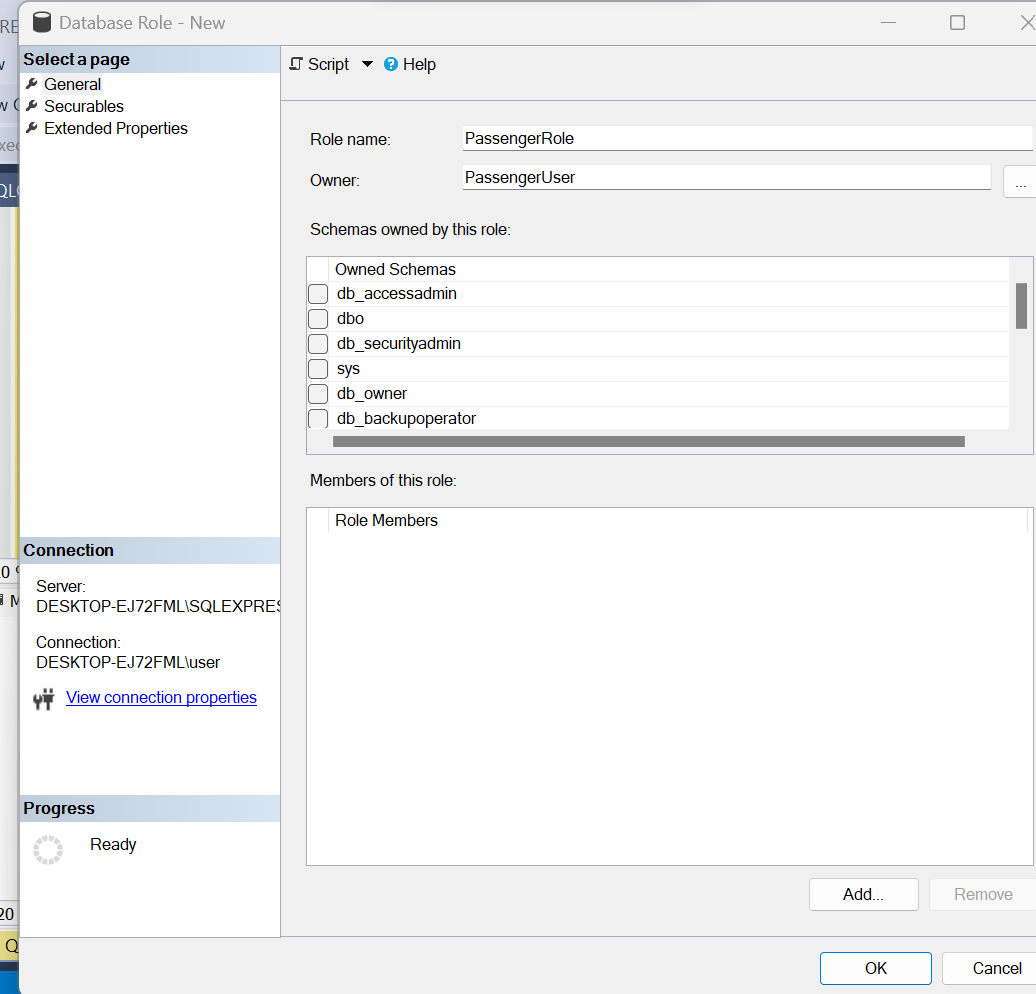
Creating Passenger Login:



Creating PassengerUser for PassengerLogin:



Creating PassengerRole:



# 6 Tables:

## 6.1: Creation & Data Population:

--Database Creation

CREATE DATABASE db\_AirportManagementSystem

GO

USE db\_AirportManagementSystem

GO

--Luggage Table

CREATE TABLE Luggage (

LuggageID INT PRIMARY KEY,

Type VARCHAR(255),

Weight DECIMAL(10, 2)

);

--Data Population

INSERT INTO Luggage (LuggageID, Type, Weight)

VALUES

(1, 'Carry-On', 12.5),

(2, 'Checked', 25.0),

(3, 'Carry-On', 15.0),

(4, 'Checked', 30.0),

(5, 'Carry-On', 10.0),

(6, 'Checked', 22.0),

(7, 'Carry-On', 18.0),

(8, 'Checked', 28.5),

(9, 'Carry-On', 14.0),

(10, 'Checked', 27.0),

(11, 'Carry-On', 11.5),

(12, 'Checked', 24.0),

(13, 'Carry-On', 16.0),

(14, 'Checked', 32.0),

(15, 'Carry-On', 13.5),

(16, 'Checked', 26.0),

(17, 'Carry-On', 20.0),

(18, 'Checked', 35.0),

(19, 'Carry-On', 17.0),

(20, 'Checked', 29.5)

select \* from Luggage

--Passenger Table

CREATE TABLE Passenger (

PassengerID INT PRIMARY KEY,

Name VARCHAR(255),

PassportID VARCHAR(255),

LuggageID INT,

FOREIGN KEY (LuggageID) REFERENCES Luggage(LuggageID)

);

--Data Population

INSERT INTO Passenger (PassengerID, Name, PassportID, LuggageID)

VALUES

(1, 'John Doe', 'AB123456', 1),

(2, 'Jane Smith', 'CD789012', 2),

(3, 'Alice Johnson', 'EF345678', 3),

(4, 'Bob Wilson', 'GH901234', 4),

(5, 'Emily Davis', 'IJ567890', 5),

(6, 'Michael Brown', 'KL123456', 6),

(7, 'Sophia Miller', 'MN789012', 7),

(8, 'William White', 'OP345678', 8),

(9, 'Olivia Harris', 'QR901234', 9),

(10, 'James Jones', 'ST567890', 10),

(11, 'Emma Taylor', 'UV123456', 11),

(12, 'Daniel Davis', 'WX789012', 12),

(13, 'Ava Wilson', 'YZ345678', 13),

(14, 'Matthew Miller', 'AA901234', 14),

(15, 'Grace White', 'BB567890', 15),

(16, 'Christopher Harris', 'CC123456', 16),

(17, 'Lily Jones', 'DD789012', 17),

(18, 'Andrew Taylor', 'EE345678', 18),

(19, 'Chloe Davis', 'FF901234', 19),

(20, 'Ryan Harris', 'GG567890', 20)

select \* from Passenger

--Passenger Contact Table

CREATE TABLE PassengerContact (

P\_Contact INT PRIMARY KEY,

PassengerID INT,

FOREIGN KEY (PassengerID) REFERENCES Passenger(PassengerID)

);

--Data Population

INSERT INTO PassengerContact (P\_Contact, PassengerID)

VALUES

(1, 1),

(2, 2),

(3, 3),

(4, 4),

(5, 5),

(6, 6),

(7, 7),

(8, 8),

(9, 9),

(10, 10),

(11, 11),

(12, 12),

(13, 13),

(14, 14),

(15, 15),

(16, 16),

(17, 17),

(18, 18),

(19, 19),

(20, 20)

select \* from PassengerContact

--Ticket Table

CREATE TABLE Ticket (

TicketID INT PRIMARY KEY,

TicketType VARCHAR(255),

Price DECIMAL(10, 2),

PurchaseDate DATE,

PassengerID INT,

FOREIGN KEY (PassengerID) REFERENCES Passenger(PassengerID)

);

--Data Population

INSERT INTO Ticket (TicketID, TicketType, Price, PurchaseDate, PassengerID)

VALUES

(1, 'Economy', 200.00, '2023-05-15', 1),

(2, 'Business', 400.00, '2023-05-16', 2),

(3, 'First Class', 600.00, '2023-05-17', 3),

(4, 'Economy', 250.00, '2023-05-18', 4),

(5, 'Business', 500.00, '2023-05-19', 5),

(6, 'First Class', 700.00, '2023-05-20', 6),

(7, 'Economy', 300.00, '2023-05-21', 7),

(8, 'Business', 550.00, '2023-05-22', 8),

(9, 'First Class', 750.00, '2023-05-23', 9),

(10, 'Economy', 220.00, '2023-05-24', 10),

(11, 'Business', 480.00, '2023-05-25', 11),

(12, 'First Class', 680.00, '2023-05-26', 12),

(13, 'Economy', 270.00, '2023-05-27', 13),

(14, 'Business', 520.00, '2023-05-28', 14),

(15, 'First Class', 720.00, '2023-05-29', 15),

(16, 'Economy', 320.00, '2023-05-30', 16),

(17, 'Business', 580.00, '2023-05-31', 17),

(18, 'First Class', 780.00, '2023-06-01', 18),

(19, 'Economy', 240.00, '2023-06-02', 19),

(20, 'Business', 460.00, '2023-06-03', 20);

select \* from Ticket

--Flight Table

CREATE TABLE Flight (

FlightID INT PRIMARY KEY,

DepartureTime DATETIME,

ArrivalTime DATETIME,

TicketID INT,

FOREIGN KEY (TicketID) REFERENCES Ticket(TicketID)

);

--Data Population

INSERT INTO Flight (FlightID, DepartureTime, ArrivalTime, TicketID)

VALUES

(1, '2024-01-09 10:00:00', '2024-01-09 12:00:00', 1),

(2, '2024-01-10 14:30:00', '2024-01-10 16:30:00', 2),

(3, '2024-01-11 08:00:00', '2024-01-11 10:00:00', 3),

(4, '2024-01-12 16:00:00', '2024-01-12 18:00:00', 4),

(5, '2024-01-13 12:30:00', '2024-01-13 14:30:00', 5),

(6, '2024-01-14 09:45:00', '2024-01-14 11:45:00', 6),

(7, '2024-01-15 17:30:00', '2024-01-15 19:30:00', 7),

(8, '2024-01-16 13:15:00', '2024-01-16 15:15:00', 8),

(9, '2024-01-17 11:00:00', '2024-01-17 13:00:00', 9),

(10, '2024-01-18 18:45:00', '2024-01-18 20:45:00', 10),

(11, '2024-01-19 14:20:00', '2024-01-19 16:20:00', 11),

(12, '2024-01-20 10:05:00', '2024-01-20 12:05:00', 12),

(13, '2024-01-21 15:45:00', '2024-01-21 17:45:00', 13),

(14, '2024-01-22 11:30:00', '2024-01-22 13:30:00', 14),

(15, '2024-01-23 19:15:00', '2024-01-23 21:15:00', 15),

(16, '2024-01-24 15:00:00', '2024-01-24 17:00:00', 16),

(17, '2024-01-25 10:35:00', '2024-01-25 12:35:00', 17),

(18, '2024-01-26 18:20:00', '2024-01-26 20:20:00', 18),

(19, '2024-01-27 14:05:00', '2024-01-27 16:05:00', 19),

(20, '2024-01-28 09:50:00', '2024-01-28 11:50:00', 20)

select \* from Flight

--Security Table

CREATE TABLE Security (

SecurityID INT PRIMARY KEY,

Name VARCHAR(255),

SecurityDepartment VARCHAR(255),

AllocatedArea VARCHAR(255)

);

--Data Population

INSERT INTO Security (SecurityID, Name, SecurityDepartment, AllocatedArea)

VALUES

(1, 'Security A', 'Department A', 'Area A'),

(2, 'Security B', 'Department B', 'Area B'),

(3, 'Security C', 'Department C', 'Area C'),

(4, 'Security D', 'Department D', 'Area D'),

(5, 'Security E', 'Department E', 'Area E'),

(6, 'Security F', 'Department F', 'Area F'),

(7, 'Security G', 'Department G', 'Area G'),

(8, 'Security H', 'Department H', 'Area H'),

(9, 'Security I', 'Department I', 'Area I'),

(10, 'Security J', 'Department J', 'Area J'),

(11, 'Security K', 'Department K', 'Area K'),

(12, 'Security L', 'Department L', 'Area L'),

(13, 'Security M', 'Department M', 'Area M'),

(14, 'Security N', 'Department N', 'Area N'),

(15, 'Security O', 'Department O', 'Area O'),

(16, 'Security P', 'Department P', 'Area P'),

(17, 'Security Q', 'Department Q', 'Area Q'),

(18, 'Security R', 'Department R', 'Area R'),

(19, 'Security S', 'Department S', 'Area S'),

(20, 'Security T', 'Department T', 'Area T');

select \* from Security

--Traffic Control Table

CREATE TABLE TrafficControl (

TrafficControlID INT PRIMARY KEY,

Name VARCHAR(255),

TowerNo INT,

TowerName VARCHAR(255)

);

--Data Population

INSERT INTO TrafficControl (TrafficControlID, Name, TowerNo, TowerName)

VALUES

(1, 'Control A', 1, 'Tower A'),

(2, 'Control B', 2, 'Tower B'),

(3, 'Control C', 3, 'Tower C'),

(4, 'Control D', 4, 'Tower D'),

(5, 'Control E', 5, 'Tower E'),

(6, 'Control F', 6, 'Tower F'),

(7, 'Control G', 7, 'Tower G'),

(8, 'Control H', 8, 'Tower H'),

(9, 'Control I', 9, 'Tower I'),

(10, 'Control J', 10, 'Tower J'),

(11, 'Control K', 11, 'Tower K'),

(12, 'Control L', 12, 'Tower L'),

(13, 'Control M', 13, 'Tower M'),

(14, 'Control N', 14, 'Tower N'),

(15, 'Control O', 15, 'Tower O'),

(16, 'Control P', 16, 'Tower P'),

(17, 'Control Q', 17, 'Tower Q'),

(18, 'Control R', 18, 'Tower R'),

(19, 'Control S', 19, 'Tower S'),

(20, 'Control T', 20, 'Tower T');

select \* from TrafficControl

--Feuling Station Table

CREATE TABLE FuelingStation (

FuelingStationID INT PRIMARY KEY,

Name VARCHAR(255),

Capacity INT,

AircraftsAttended INT,

FuelType VARCHAR(255)

);

--Data Population

INSERT INTO FuelingStation (FuelingStationID, Name, Capacity, AircraftsAttended, FuelType)

VALUES

(1, 'Fuel Station A', 50000, 5, 'Jet-A'),

(2, 'Fuel Station B', 60000, 6, 'AVGAS'),

(3, 'Fuel Station C', 70000, 7, 'Jet-A'),

(4, 'Fuel Station D', 80000, 8, 'AVGAS'),

(5, 'Fuel Station E', 90000, 9, 'Jet-A'),

(6, 'Fuel Station F', 100000, 10, 'AVGAS'),

(7, 'Fuel Station G', 110000, 11, 'Jet-A'),

(8, 'Fuel Station H', 120000, 12, 'AVGAS'),

(9, 'Fuel Station I', 130000, 13, 'Jet-A'),

(10, 'Fuel Station J', 140000, 14, 'AVGAS'),

(11, 'Fuel Station K', 150000, 15, 'Jet-A'),

(12, 'Fuel Station L', 160000, 16, 'AVGAS'),

(13, 'Fuel Station M', 170000, 17, 'Jet-A'),

(14, 'Fuel Station N', 180000, 18, 'AVGAS'),

(15, 'Fuel Station O', 190000, 19, 'Jet-A'),

(16, 'Fuel Station P', 200000, 20, 'AVGAS'),

(17, 'Fuel Station Q', 210000, 21, 'Jet-A'),

(18, 'Fuel Station R', 220000, 22, 'AVGAS'),

(19, 'Fuel Station S', 230000, 23, 'Jet-A'),

(20, 'Fuel Station T', 240000, 24, 'AVGAS');

select \* from FuelingStation

--Airport Runway Table

CREATE TABLE AirportRunway (

RunwayID INT PRIMARY KEY,

Length INT,

LightingSystems VARCHAR(255)

);

--Data Population

INSERT INTO AirportRunway (RunwayID, Length, LightingSystems)

VALUES

(1, 8000, 'High-Intensity'),

(2, 9000, 'Medium-Intensity'),

(3, 10000, 'Low-Intensity'),

(4, 8500, 'High-Intensity'),

(5, 9500, 'Medium-Intensity'),

(6, 11000, 'Low-Intensity'),

(7, 8200, 'High-Intensity'),

(8, 9300, 'Medium-Intensity'),

(9, 10500, 'Low-Intensity'),

(10, 8700, 'High-Intensity'),

(11, 9700, 'Medium-Intensity'),

(12, 11500, 'Low-Intensity'),

(13, 8800, 'High-Intensity'),

(14, 9200, 'Medium-Intensity'),

(15, 10700, 'Low-Intensity'),

(16, 8900, 'High-Intensity'),

(17, 9400, 'Medium-Intensity'),

(18, 11200, 'Low-Intensity'),

(19, 9200, 'High-Intensity'),

(20, 9900, 'Medium-Intensity');

select \* from AirportRunway

--Airport Table

CREATE TABLE Airport (

AirportID INT PRIMARY KEY,

Name VARCHAR(255),

Location VARCHAR(255),

AirportType VARCHAR(255),

FlightID INT,

SecurityID INT,

TrafficControlID INT,

FuelingStationID INT,

RunwayID INT,

FOREIGN KEY (FlightID) REFERENCES Flight(FlightID),

FOREIGN KEY (SecurityID) REFERENCES Security(SecurityID),

FOREIGN KEY (TrafficControlID) REFERENCES TrafficControl(TrafficControlID),

FOREIGN KEY (FuelingStationID) REFERENCES FuelingStation(FuelingStationID),

FOREIGN KEY (RunwayID) REFERENCES AirportRunway(RunwayID)

);

--Data Population

INSERT INTO Airport (AirportID, Name, Location, AirportType, FlightID, SecurityID, TrafficControlID, FuelingStationID, RunwayID)

VALUES

(1, 'International Airport', 'City A', 'International', 1, 1, 1, 1, 1),

(2, 'Domestic Airport', 'City B', 'Domestic', 2, 2, 2, 2, 2),

(3, 'Regional Airport', 'City C', 'Regional', 3, 3, 3, 3, 3),

(4, 'Private Airport', 'City D', 'Private', 4, 4, 4, 4, 4),

(5, 'Major Airport', 'City E', 'International', 5, 5, 5, 5, 5),

(6, 'City Airport', 'City F', 'Domestic', 6, 6, 6, 6, 6),

(7, 'Capital Airport', 'City G', 'International', 7, 7, 7, 7, 7),

(8, 'Skyport', 'City H', 'Domestic', 8, 8, 8, 8, 8),

(9, 'Aero Hub', 'City I', 'International', 9, 9, 9, 9, 9),

(10, 'Air Base', 'City J', 'Military', 10, 10, 10, 10, 10),

(11, 'Gateway Airport', 'City K', 'International', 11, 11, 11, 11, 11),

(12, 'Sunset Airfield', 'City L', 'Private', 12, 12, 12, 12, 12),

(13, 'Metro Airport', 'City M', 'Domestic', 13, 13, 13, 13, 13),

(14, 'Eagle Landing', 'City N', 'Regional', 14, 14, 14, 14, 14),

(15, 'Harbor Skyport', 'City O', 'International', 15, 15, 15, 15, 15),

(16, 'Silver Wings', 'City P', 'Domestic', 16, 16, 16, 16, 16),

(17, 'Panorama Airport', 'City Q', 'International', 17, 17, 17, 17, 17),

(18, 'Golden Fields', 'City R', 'Domestic', 18, 18, 18, 18, 18),

(19, 'Air Haven', 'City S', 'International', 19, 19, 19, 19, 19),

(20, 'Central Skyport', 'City T', 'Domestic', 20, 20, 20, 20, 20)

select \* from Airport

--Airline Table

CREATE TABLE Airline (

AirlineID INT PRIMARY KEY,

AirlineName VARCHAR(255),

AirlineType VARCHAR(255),

AirlineStatus VARCHAR(255),

FlightID INT,

FOREIGN KEY (FlightID) REFERENCES Flight(FlightID)

);

--Data Population

INSERT INTO Airline (AirlineID, AirlineName, AirlineType, AirlineStatus, FlightID)

VALUES

(1, 'Airline A', 'International', 'Active', 1),

(2, 'Airline B', 'Domestic', 'Active', 2),

(3, 'Airline C', 'Regional', 'Inactive', 3),

(4, 'Airline D', 'Private', 'Active', 4),

(5, 'Airline E', 'Domestic', 'Active', 5),

(6, 'Airline F', 'International', 'Inactive', 6),

(7, 'Airline G', 'Private', 'Active', 7),

(8, 'Airline H', 'Regional', 'Active', 8),

(9, 'Airline I', 'International', 'Inactive', 9),

(10, 'Airline J', 'Domestic', 'Active', 10),

(11, 'Airline K', 'Private', 'Active', 11),

(12, 'Airline L', 'International', 'Inactive', 12),

(13, 'Airline M', 'Regional', 'Active', 13),

(14, 'Airline N', 'Domestic', 'Active', 14),

(15, 'Airline O', 'Private', 'Active', 15),

(16, 'Airline P', 'International', 'Inactive', 16),

(17, 'Airline Q', 'Regional', 'Active', 17),

(18, 'Airline R', 'Domestic', 'Active', 18),

(19, 'Airline S', 'Private', 'Active', 19),

(20, 'Airline T', 'International', 'Inactive', 20);

select \* from Airline

--Passenger Security Association Table

CREATE TABLE PassengerSecurityAssociation (

PassengerID INT,

SecurityID INT,

CheckDateTime DATETIME,

Status VARCHAR(255),

PRIMARY KEY (PassengerID, SecurityID),

FOREIGN KEY (PassengerID) REFERENCES Passenger(PassengerID),

FOREIGN KEY (SecurityID) REFERENCES Security(SecurityID)

);

--Data Population

INSERT INTO PassengerSecurityAssociation (PassengerID, SecurityID, CheckDateTime, Status)

VALUES

(1, 1, '2023-05-15 08:00:00', 'Clear'),

(2, 2, '2023-05-16 10:30:00', 'Clear'),

(3, 3, '2023-05-17 12:45:00', 'Pending'),

(4, 4, '2023-05-18 14:15:00', 'Clear'),

(5, 5, '2023-05-19 16:30:00', 'Pending'),

(6, 6, '2023-05-20 18:45:00', 'Clear'),

(7, 7, '2023-05-21 20:00:00', 'Pending'),

(8, 8, '2023-05-22 22:15:00', 'Clear'),

(9, 9, '2023-05-23 23:30:00', 'Pending'),

(10, 10, '2023-05-24 01:45:00', 'Clear'),

(11, 11, '2023-05-25 03:00:00', 'Clear'),

(12, 12, '2023-05-26 05:15:00', 'Pending'),

(13, 13, '2023-05-27 07:30:00', 'Clear'),

(14, 14, '2023-05-28 09:45:00', 'Pending'),

(15, 15, '2023-05-29 11:00:00', 'Clear'),

(16, 16, '2023-05-30 13:15:00', 'Pending'),

(17, 17, '2023-05-31 15:30:00', 'Clear'),

(18, 18, '2023-06-01 17:45:00', 'Pending'),

(19, 19, '2023-06-02 20:00:00', 'Clear'),

(20, 20, '2023-06-03 22:15:00', 'Pending');

select \* from PassengerSecurityAssociation

--Pilot Table

CREATE TABLE Pilot (

PilotID INT PRIMARY KEY,

Name VARCHAR(255),

HoursExperience INT,

LicenseNo VARCHAR(255),

AircraftAllowed VARCHAR(255)

);

--Data Population

INSERT INTO Pilot (PilotID, Name, HoursExperience, LicenseNo, AircraftAllowed)

VALUES

(1, 'John Smith', 1500, 'PL12345', 'Boeing 737'),

(2, 'Jane Doe', 2000, 'PL67890', 'Airbus A320'),

(3, 'Michael Johnson', 1800, 'PL34567', 'Boeing 747'),

(4, 'Emily White', 1200, 'PL89012', 'Cessna 172'),

(5, 'David Brown', 2500, 'PL45678', 'Bombardier Challenger 300'),

(6, 'Emma Wilson', 1600, 'PL12346', 'Airbus A380'),

(7, 'Andrew Miller', 2200, 'PL67891', 'Boeing 777'),

(8, 'Olivia Davis', 1900, 'PL23456', 'Embraer E175'),

(9, 'William Taylor', 1400, 'PL78901', 'Cessna Citation X'),

(10, 'Sophia Anderson', 2600, 'PL34567', 'Gulfstream G650'),

(11, 'Liam Harris', 1700, 'PL89012', 'Airbus A330'),

(12, 'Ava Clark', 2100, 'PL12347', 'Boeing 787'),

(13, 'Daniel Martin', 1300, 'PL67892', 'Cessna 208 Caravan'),

(14, 'Isabella Hall', 2400, 'PL23457', 'Bombardier Global 6000'),

(15, 'Jackson Adams', 2000, 'PL78902', 'Embraer Phenom 300'),

(16, 'Mia Turner', 1800, 'PL34568', 'Airbus A350'),

(17, 'Logan Walker', 1500, 'PL90123', 'Boeing 767'),

(18, 'Sophie Baker', 1700, 'PL12348', 'Cirrus SR22'),

(19, 'Samuel Cooper', 2100, 'PL56789', 'Airbus A319'),

(20, 'Chloe Nelson', 2300, 'PL23458', 'Boeing 757');

select \* from Pilot

--Pilot Flight Association Table

CREATE TABLE PilotFlightAssociation (

PilotID INT,

FlightID INT,

PRIMARY KEY (PilotID, FlightID),

FOREIGN KEY (PilotID) REFERENCES Pilot(PilotID),

FOREIGN KEY (FlightID) REFERENCES Flight(FlightID)

);

--Data Population

INSERT INTO PilotFlightAssociation (PilotID, FlightID)

VALUES

(1, 1),

(2, 2),

(3, 3),

(4, 4),

(5, 5),

(6, 6),

(7, 7),

(8, 8),

(9, 9),

(10, 10),

(11, 11),

(12, 12),

(13, 13),

(14, 14),

(15, 15),

(16, 16),

(17, 17),

(18, 18),

(19, 19),

(20, 20);

select \* from PilotFlightAssociation

--Flight Fueling Station Association Table

CREATE TABLE FlightFuelingStationAssociation (

FlightID INT,

FuelingStationID INT,

PRIMARY KEY (FlightID, FuelingStationID),

FOREIGN KEY (FlightID) REFERENCES Flight(FlightID),

FOREIGN KEY (FuelingStationID) REFERENCES FuelingStation(FuelingStationID)

);

--Data Population

INSERT INTO FlightFuelingStationAssociation (FlightID, FuelingStationID)

VALUES

(1, 1),

(2, 2),

(3, 3),

(4, 4),

(5, 5),

(6, 6),

(7, 7),

(8, 8),

(9, 9),

(10, 10),

(11, 11),

(12, 12),

(13, 13),

(14, 14),

(15, 15),

(16, 16),

(17, 17),

(18, 18),

(19, 19),

(20, 20);

select \* from FlightFuelingStationAssociation

# 7: Backup & Recovery:

## 7.1: Backup and Recovery Strategy:

### 7.1.1: Full backup:

It is mandatory to be able to have a full backup within our reach so that we can recover our database in case of an emergency. For this, it is advised to take the backup after designing the database and then after each week on two separate locations or disks. One of these disks will be stored locally within the airport, while the other disk will be stored at a safe place, outside the airport, and it will be ensured that both disks have proper maintenance and updation with time, and will be kept in a secure location with limited access. These two disks will be of 100TB each.

### 7.1.2: Differential backups:

The airport’s database will need constant changes with time, maybe even daily. So it is advisable to take Differential backup every two days, mostly after regular office hours when the flights are less operating. Backup these data to both of the disk locations.

### 7.1.3: Transactional Backups:

Due to the constant need for data updation and maintenance, it would be advised to take daily log backups to have some sort of backup to trace back to during the recovery phase.

### 7.1.4: Full Recovery:

Let us think of a scenario where there is power turbulence in the airport, and because of that reason, the disks on which the data is stored are somewhat damaged. Within 15 minutes of the incident, the DBA at the airport must start the recovery process by taking a full recovery from the off-premises disk and locating other servers for this activity.

In the worst-case scenario, if a large-scale incident occurs such as a natural disaster, and the on-premises disk is damaged, we will still be getting our data in terms of recovery from the off-premises disk. The recovery time will be preferred over the availability of the data to the other users in this case.

### 7.1.5: Differential Recovery:

After taking the full recovery, we will take differential recovery (ies) after the last full backup to be able to have the latest possible data. It will be made sure that the downtime for the airport staff in this case does not occur so that they can access their data.

### 7.1.6: Transactional Recovery:

After the full and differential recovery processes, it will be highly advisable to first make the downtime zero and then continue with this recovery phase. This recovery phase will be executed in steps, to minimize the downtime. That time will be searched when we took the last differential recovery, and then the first transactional recovery phase will start, followed by the second one until the latest data is available.

### 7.1.7: Backups on Cloud:

As the system expands, it will also be necessary to store the important data on the cloud, so that important data can be accessed in case of a worst-case scenario or downtime from the cloud itself.

### 7.1.8: Data Retention timeframe:

It will be advised to finish all of the recovery process within 5-6 hours, in which the most time will be taken by the full recovery process, followed by the differential recovery and then the transactional recovery.

### 7.1.9: Updating the recovery & backup process:

As our system expands, it will be important for DBA to consider the drawbacks of the above backup and recovery phase and draw a new recovery plan for a better response to any incident.

### 7.1.10: Off-Premises Storage Considerations:

As our system expands, it will be important to have more than one off-site storage facility to have the recovery phase executed as quickly as possible. In the first phase, a new off-site facility will be located away from the city in a secure area for this purpose.

## 7.2: Backup Commands:

--First Copy of Full Database Backup

BACKUP DATABASE db\_AirportManagementSystem

TO DISK = 'C:\db\_AirportManagementSystem.bak'

WITH INIT;

--Second Copy of Full Database Backup

BACKUP DATABASE db\_AirportManagementSystem

TO DISK = 'C:\db\_AirportManagementSystem.bak1'

WITH INIT;

--Third Copy of Full Database Backup

BACKUP DATABASE db\_AirportManagementSystem

TO DISK = 'C:\db\_AirportManagementSystem.bak2'

WITH INIT;

--First Copy of Differential Backup

BACKUP DATABASE db\_AirportManagementSystem

TO DISK = 'C:\db\_AirportManagementSystem\_Diff.bak'

WITH DIFFERENTIAL;

--Second Copy of Differential Backup

BACKUP DATABASE db\_AirportManagementSystem

TO DISK = 'C:\db\_AirportManagementSystem\_Diff1.bak'

WITH DIFFERENTIAL;

--Transaction Log Backup

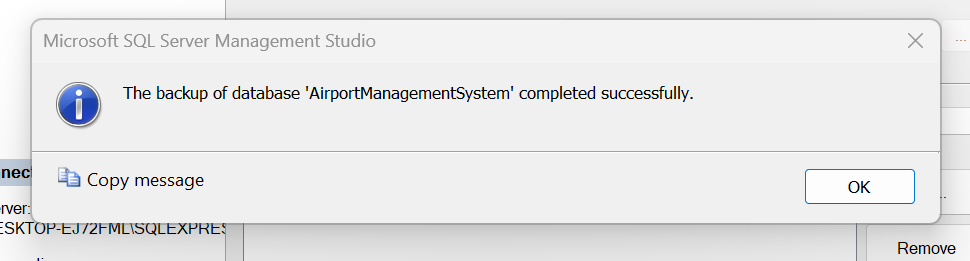
BACKUP LOG db\_AirportManagementSystem

TO DISK = 'C:\db\_AirportManagementSystem\_Log.trn';

Full backup:

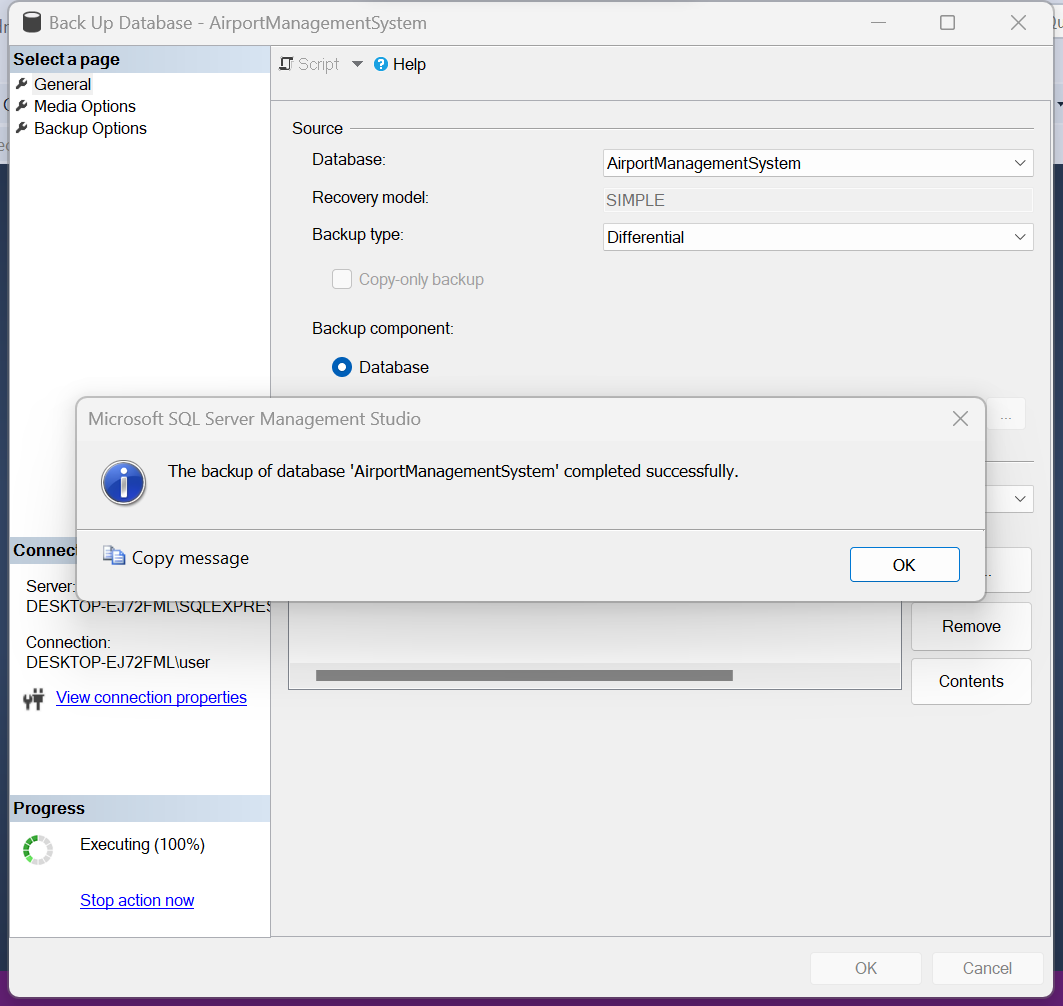
Backup location:

C:\Program Files\Microsoft SQL Server\MSSQL16.SQLEXPRESS\MSSQL\Backup\



Differential Backup:

Backup location: C:\Program Files\Microsoft SQL Server\MSSQL16.SQLEXPRESS\MSSQL\Backup\



# 7.3: Recovery commands:

--Recovery

drop db\_AirportManagementSystem

go

use master

select \* from tbl\_Employee

--Restoring Full Database Backup

RESTORE DATABASE db\_AirportManagementSystem

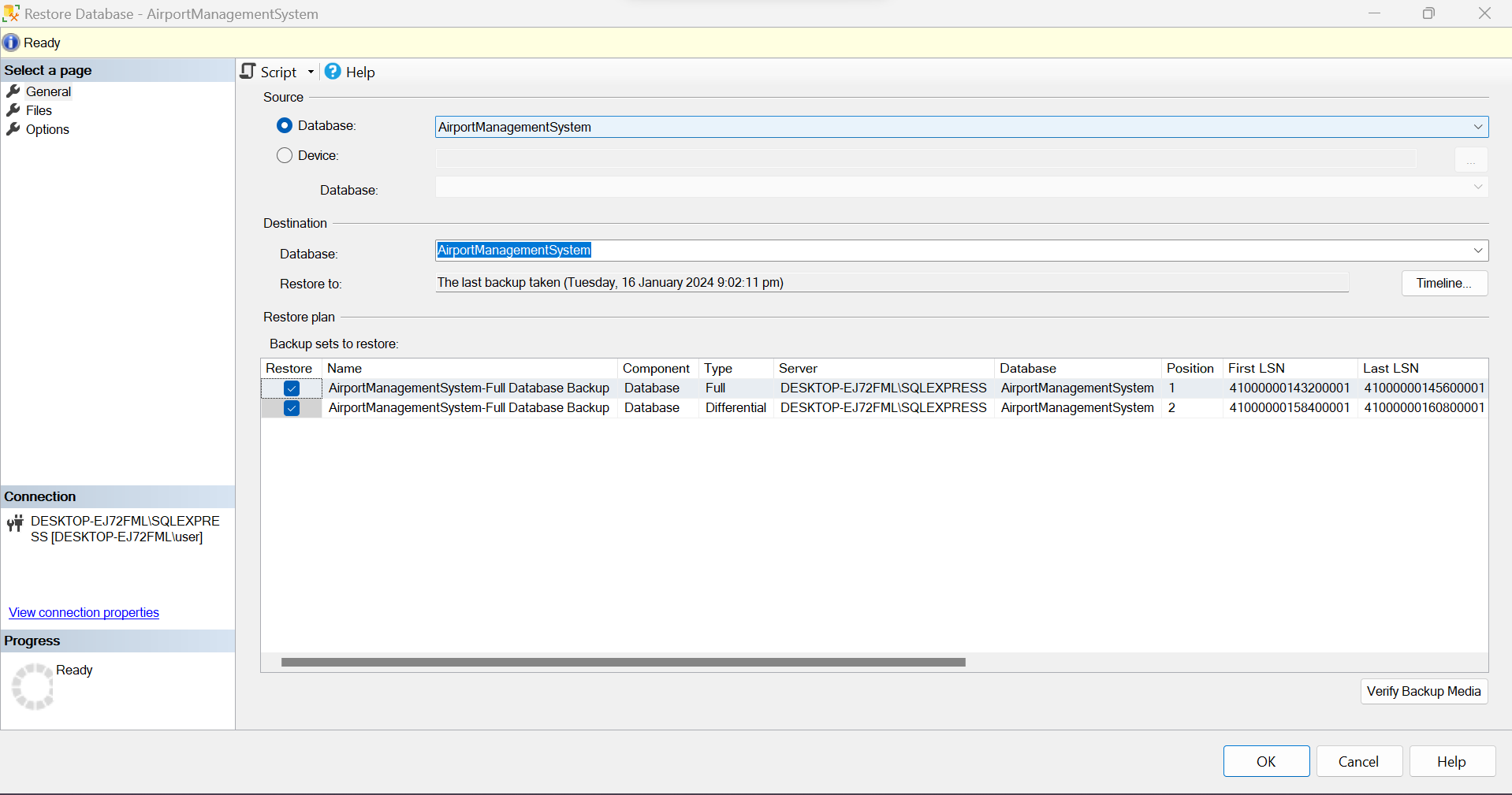
FROM DISK = 'C:\db\_AirportManagementSystemfull.bak';

--Restoring a Differential Backup

RESTORE DATABASE AirportManagementSystem

FROM DISK = 'C:\AirportManagementSystemfull-diff2.bak'

WITH RECOVERY;



# 8: Views & Joins:

## 8.1: Passenger View:

Creating a view that gets the passenger’s data, luggage, ticket, etc.

-- Create the view:

CREATE VIEW PassengerTicketView AS

SELECT

P.PassengerID,

P.Name AS PassengerName,

P.PassportID,

L.Type AS LuggageType,

L.Weight AS LuggageWeight,

T.TicketType,

T.Price AS TicketPrice,

T.PurchaseDate

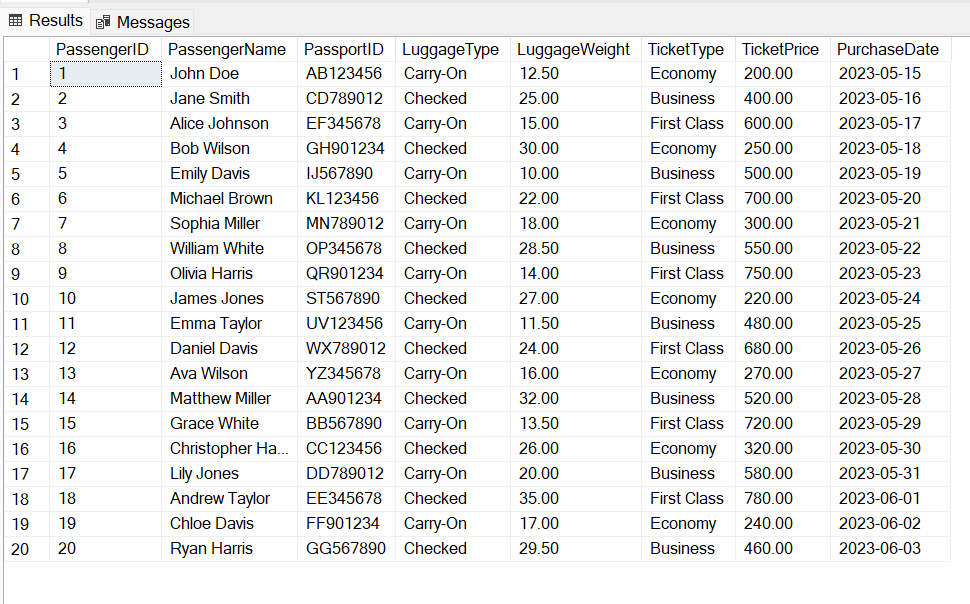
FROM

Passenger P

JOIN Luggage L ON P.LuggageID = L.LuggageID

JOIN Ticket T ON P.PassengerID = T.PassengerID;

SELECT \* FROM PassengerTicketView;



## 8.2: Pilot View:

Creating a view that displays the information about the Pilot, the airline they are associated with, and the flight that they are going to fly in their next shift:

-- Create the view

CREATE VIEW PilotScheduleView

AS

SELECT

P.PilotID,

P.Name AS PilotName,

P.HoursExperience,

P.LicenseNo,

P.AircraftAllowed,

AF.FlightID,

AF.DepartureTime AS NextShiftDepartureTime,

AF.ArrivalTime AS NextShiftArrivalTime,

A.AirlineName

FROM

Pilot AS P

JOIN

PilotFlightAssociation AS PFA ON P.PilotID = PFA.PilotID

JOIN

Flight AS F ON PFA.FlightID = F.FlightID

JOIN

Airline AS A ON F.FlightID = A.FlightID

CROSS APPLY (

SELECT TOP 1

AF2.FlightID,

AF2.DepartureTime,

AF2.ArrivalTime

FROM

PilotFlightAssociation AS PFA2

JOIN

Flight AS AF2 ON PFA2.FlightID = AF2.FlightID

WHERE

PFA2.PilotID = P.PilotID

AND AF2.DepartureTime > GETDATE() -- Next shift's departure time is in the future

ORDER BY

AF2.DepartureTime

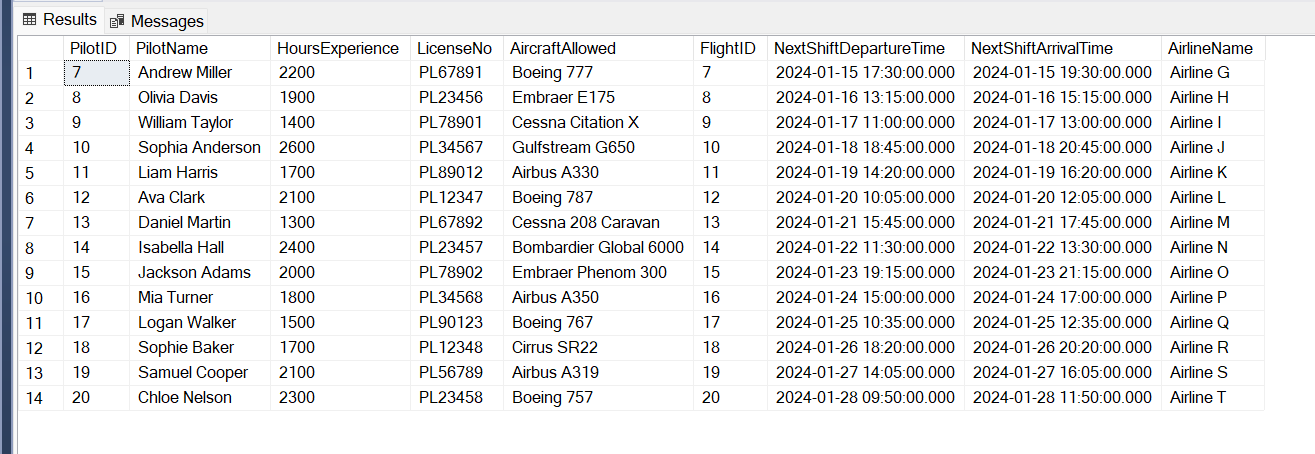
) AS AF

WHERE

AF.FlightID IS NOT NULL;

SELECT \* FROM PilotScheduleView;

Even though it is a very complex view, it will help the performance monitoring of the database.



## 8.3: Runway View:

Creating a view that checks which runway will be used by which flight and then will check if those runways are even free or not.

-- Create the view

CREATE VIEW RunwayOccupancyView

AS

SELECT

R.RunwayID,

R.Length,

R.LightingSystems,

F.FlightID,

F.DepartureTime,

F.ArrivalTime,

CASE

WHEN F.DepartureTime IS NOT NULL AND F.ArrivalTime IS NULL THEN 'Departure'

WHEN F.DepartureTime IS NULL AND F.ArrivalTime IS NOT NULL THEN 'Arrival'

ELSE 'Unknown'

END AS OperationType,

CASE

WHEN F.DepartureTime IS NOT NULL AND F.ArrivalTime IS NULL THEN 'Occupied'

WHEN F.DepartureTime IS NULL AND F.ArrivalTime IS NOT NULL THEN 'Occupied'

ELSE 'Free'

END AS RunwayStatus

FROM

AirportRunway AS R

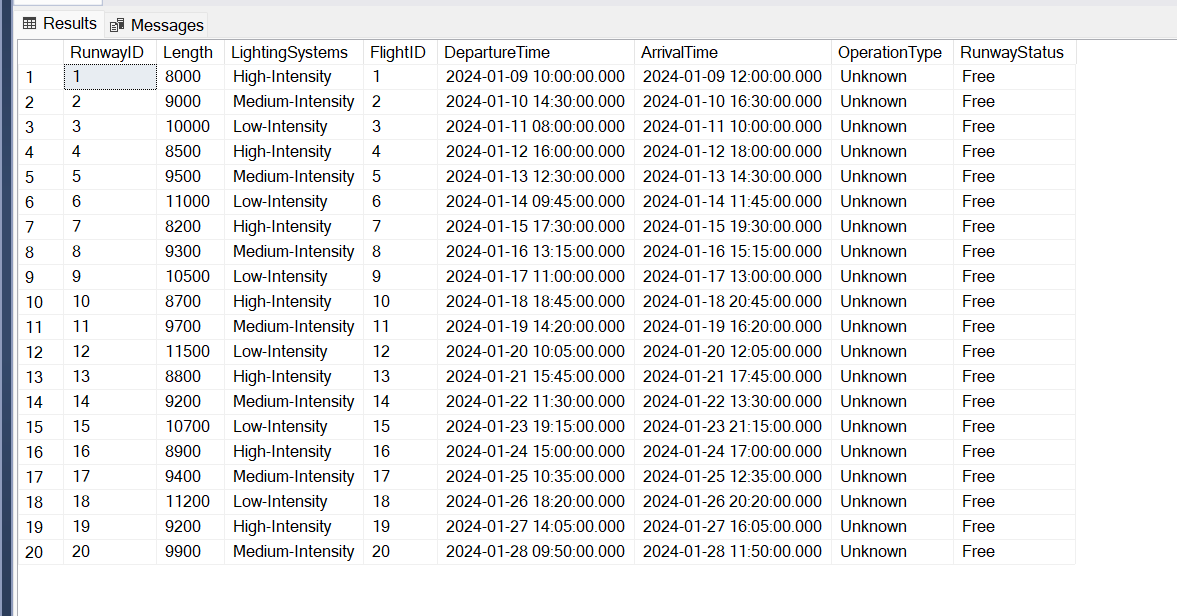
LEFT JOIN

Airport AS AP ON R.RunwayID = AP.RunwayID

LEFT JOIN

Flight AS F ON AP.FlightID = F.FlightID;

SELECT \* FROM RunwayOccupancyView;



## 8.4: Flight View:

A view that finds how many flights are operating on a particular day.

-- Create the view without dbo schema

CREATE VIEW FlightsOperatingOnDateView

AS

SELECT

FlightID,

DepartureTime,

ArrivalTime,

COUNT(\*) OVER (PARTITION BY CONVERT(DATE, DepartureTime)) AS FlightsOnDate

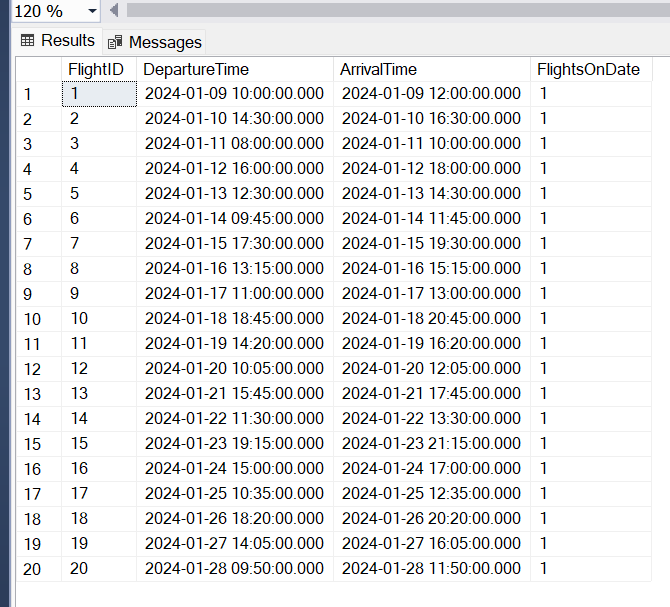
FROM

Flight;

-- Select data from the view without dbo schema

SELECT \* FROM FlightsOperatingOnDateView;

The COUNT(\*) OVER (PARTITION BY CONVERT(DATE, F.DepartureTime)) counts the number of flights for each date.



## 8.5: Fueling Station View:

-- Create the view without dbo schema

CREATE VIEW FuelingStationUsageView

AS

SELECT

FS.FuelingStationID,

FS.Name AS FuelingStationName,

COUNT(DISTINCT FSA.FlightID) AS FlightsAttended,

FS.FuelType AS FuelSoldType

FROM

FlightFuelingStationAssociation AS FSA

JOIN

FuelingStation AS FS ON FSA.FuelingStationID = FS.FuelingStationID

GROUP BY

FS.FuelingStationID,

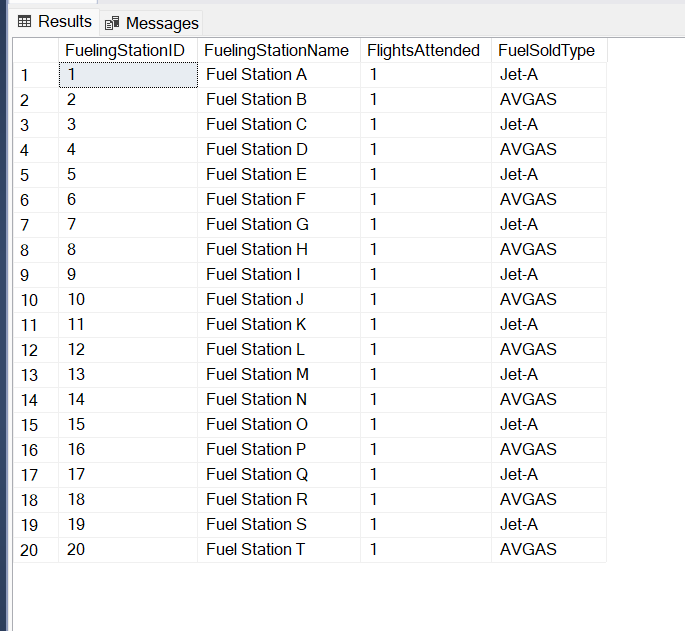
FS.Name,

FS.FuelType;

-- Select data from the view

SELECT \* FROM FuelingStationUsageView;

CONVERT(DATE, FSA.CheckDateTime) extracts the date part from the refueling check date and time.



# 9. Stored Procedures:

## 9.1: Weight Check Procedure:

--When a passenger increases the weight of his luggage greater

--than the limit set by the airport, the price of the ticket is increased

-- Create the procedure

CREATE PROCEDURE IncreaseTicketPriceOnExceedLimit

@passengerID INT,

@luggageWeight DECIMAL(10, 2),

@limit DECIMAL(10, 2),

@priceMultiplier DECIMAL(10, 2)

AS

BEGIN

-- Check if luggage weight exceeds the limit

IF @luggageWeight > @limit

BEGIN

-- Calculate the new ticket price

DECLARE @newPrice DECIMAL(10, 2);

SET @newPrice = (SELECT Price \* @priceMultiplier FROM Ticket WHERE PassengerID = @passengerID);

-- Update the ticket price

UPDATE Ticket

SET Price = @newPrice

WHERE PassengerID = @passengerID;

END

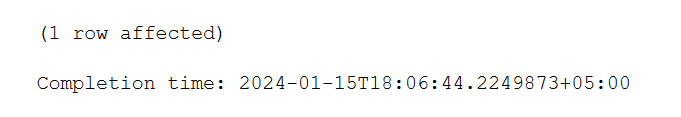
END;

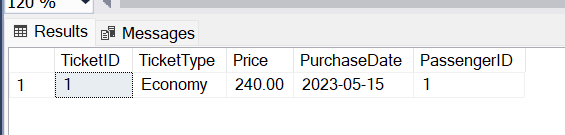
-- Execute the procedure

EXEC IncreaseTicketPriceOnExceedLimit @passengerID = 1, @luggageWeight = 37, @limit = 35, @priceMultiplier = 1.2;

-- Select data from the Ticket table

SELECT \* FROM Ticket WHERE PassengerID = 1;





## 9.2: Security Area Changed Procedure:

Sometimes when security personnel is on duty, they are ordered to change their locations for some reasons. We are assuming this scenario and implementing this:

-- Create the procedure

CREATE PROCEDURE UpdateSecurityArea

@securityID INT,

@newAllocatedArea VARCHAR(255)

AS

BEGIN

-- Update the allocated area for the specified security department

UPDATE Security

SET AllocatedArea = @newAllocatedArea

WHERE SecurityID = @securityID;

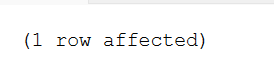
END;

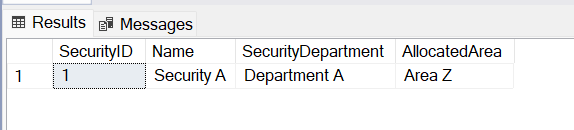
--Execute the procedure

EXEC UpdateSecurityArea @securityID = 1, @newAllocatedArea = 'Area Z';

– Select data from Security table to verify the update

SELECT \* FROM Security WHERE SecurityID = 1;





## 9.3: Print Details of First Five Passengers Procedure:

--Print the details of first five passengers

create procedure DetailsOfPassengers

as

begin

declare @counter int = 1

while @counter <= 5

begin

select \* from Passenger order by PassengerID

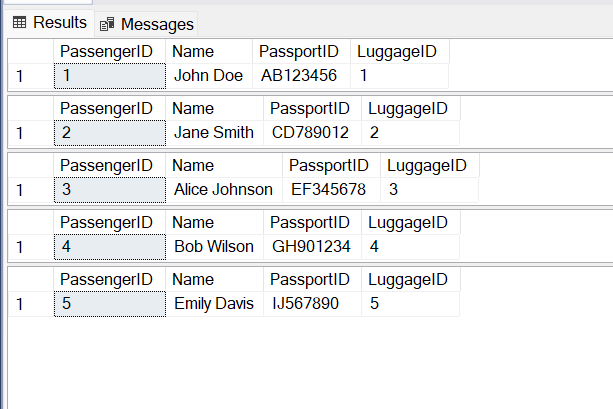
offset (@counter - 1) rows fetch next 1 rows only

set @counter = @counter + 1

end

end

exec DetailsOfPassengers



## 9.4: Show Information of Passengers & Their Luggage Procedure:

--Show the details of passengers and their luggage

create procedure PassengersAndLuggage

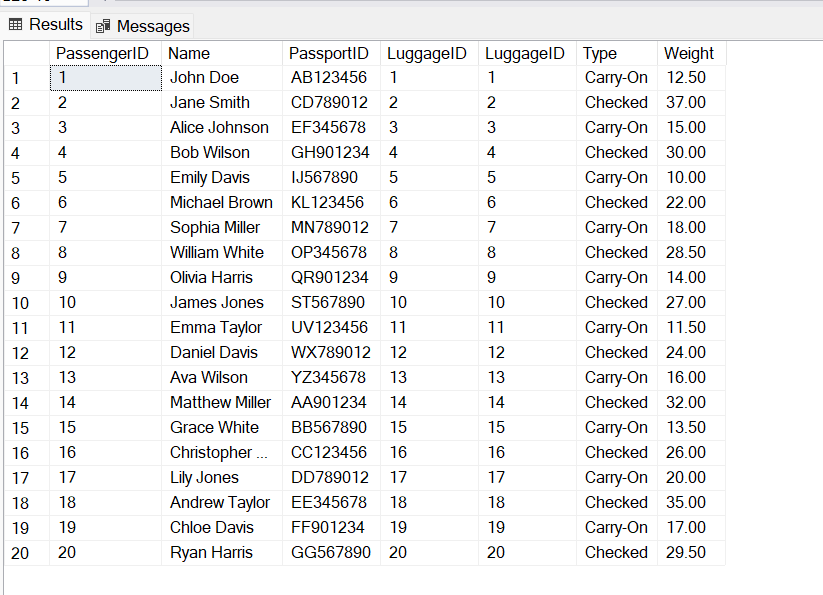
as

begin

select \* from Passenger as p inner join Luggage as l on p.LuggageID = l.LuggageID

end

exec PassengersAndLuggage



## 9.5: Number of Flights Booked Procedure:

--Show details of how many flights are already booked

create procedure FlightsBooked

@counter int output

as

begin

set NOCOUNT on

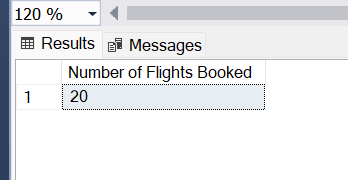
select @counter = count (\*) from Flight

select count (\*) as 'Number of Flights Booked' from Flight

end

declare @output int

exec FlightsBooked @counter = 1



## 9.6: Lighting System Check:

Check if the lighting systems are working fine. If not, then delay the flight.

-- Create the procedure

CREATE PROCEDURE CheckRunwayCondition

@flightID VARCHAR(50)

AS

BEGIN

DECLARE @runwayID INT;

DECLARE @departureTime DATETIME;

DECLARE @lightingSystemsCondition VARCHAR(255);

-- Get the runway ID and departure time for the specified flight

SELECT @runwayID = A.RunwayID, @departureTime = F.DepartureTime

FROM Flight AS F

JOIN Airport AS A ON F.FlightID = A.FlightID

WHERE F.FlightID = @flightID;

-- Get the condition of lighting systems for the specified runway

SELECT @lightingSystemsCondition = LightingSystems

FROM AirportRunway

WHERE RunwayID = @runwayID;

-- Check the condition and take action accordingly

IF @lightingSystemsCondition = 'Operational'

BEGIN

-- Allow the flight to operate

PRINT 'Flight ' + @flightID + ' can operate as scheduled.';

END

ELSE

BEGIN

-- Delay the flight

PRINT 'Flight ' + @flightID + ' is delayed due to runway lighting systems issues.';

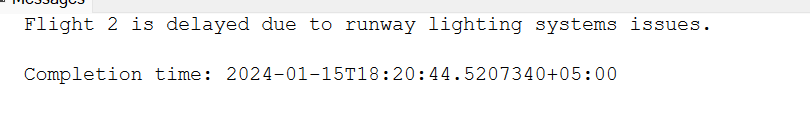
-- Add logic for further actions if needed, such as rescheduling or notifying passengers

END;

END;

-- Execute the procedure

EXEC CheckRunwayCondition @flightID = 2;



# 10. Triggers:

## 10.1: Passenger-Ticket Trigger:

--If the passenger wants to book an already booked ticket it generates the following trigger

create trigger BookATicket

on Ticket

for insert

as

begin

if ((select max(TicketID) from Ticket)<20)

begin

raiserror ('Error. This seat has already been book.',20,3)

end;

else

begin

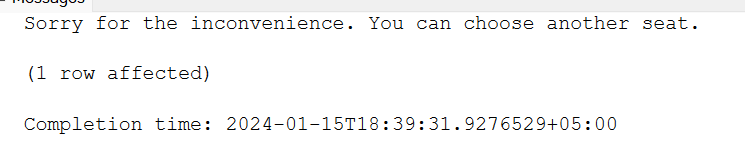
print 'Sorry for the inconvenience. You can choose another seat.'

end

end;

INSERT INTO Ticket (TicketID, TicketType, Price, PurchaseDate, PassengerID)

VALUES (21, 'Economy', 150.00, '2024-01-20', 1);



## 10.2: Ticket Upgradation Trigger:

-- A particular customer is trying to upgrade his flight from economy to business

CREATE TRIGGER UpgradeATicket

ON Ticket

INSTEAD OF DELETE

AS

BEGIN

PRINT 'Dear Passenger! Thank you for your request, your ticket is being upgraded.';

DECLARE @TicketID INT;

DECLARE @TicketType VARCHAR(255);

DECLARE @Price DECIMAL(10, 2);

DECLARE @PurchaseDate DATE;

DECLARE @PassengerID INT;

SELECT @TicketID = d.TicketID,

@TicketType = d.TicketType,

@Price = d.Price,

@PurchaseDate = d.PurchaseDate,

@PassengerID = d.PassengerID

FROM deleted d;

-- Output the updated ticket information

PRINT 'Dear Passenger! Following is your updated ticket information';

PRINT 'TicketID: ' + CAST(@TicketID AS VARCHAR(10));

PRINT 'TicketType: ' + @TicketType;

PRINT 'Price: ' + CAST(@Price AS VARCHAR(20));

PRINT 'PurchaseDate: ' + CAST(@PurchaseDate AS VARCHAR(20));

PRINT 'PassengerID: ' + CAST(@PassengerID AS VARCHAR(10));

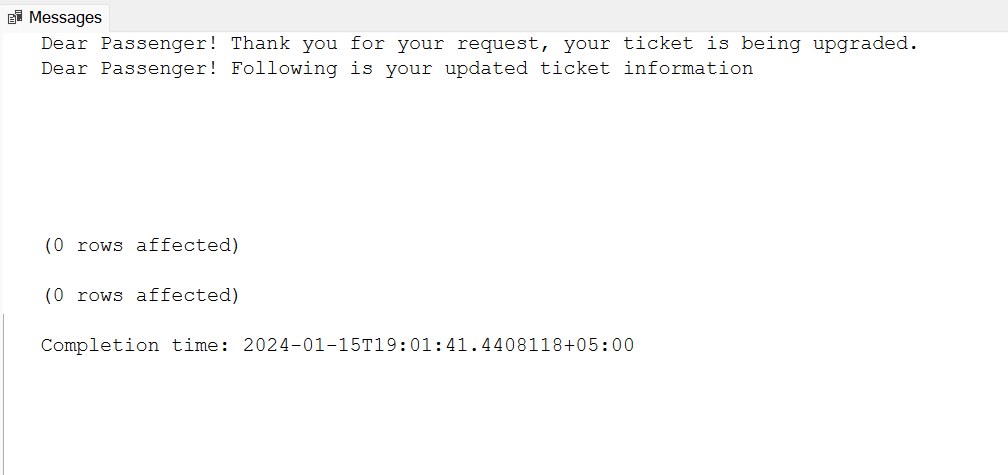
-- Perform the necessary updates

UPDATE Ticket SET TicketType = 'Business', Price = 600 WHERE TicketID = @TicketID;

END;

-- Execute the trigger by deleting a ticket (assuming TicketID = 1)

DELETE FROM Ticket WHERE TicketID = 1;



## 10.3: Flight Delay Trigger:

-- If there is a flight delay, the passengers are notified

CREATE TRIGGER FlightDelay

ON Flight

AFTER UPDATE

AS

BEGIN

-- Check if DepartureTime has been updated

IF UPDATE(DepartureTime)

BEGIN

DECLARE @FlightID VARCHAR(50);

DECLARE @NewDepartureTime DATETIME;

DECLARE @OldDepartureTime DATETIME;

-- Get the FlightID and the old and new DepartureTime values

SELECT @FlightID = i.FlightID, @NewDepartureTime = i.DepartureTime, @OldDepartureTime = d.DepartureTime

FROM inserted i

JOIN deleted d ON i.FlightID = d.FlightID;

-- Check if there's an actual delay

IF @NewDepartureTime > @OldDepartureTime

BEGIN

-- Update the DepartureTime for demonstration purposes

UPDATE Flight SET DepartureTime = DATEADD(HOUR, 2, @OldDepartureTime) WHERE FlightID = @FlightID;

-- Notify passengers about the delay

PRINT 'Dear Passengers! We would like to notify you that Flight ' + @FlightID + ' has been delayed by 2 hours. We appreciate your patience. Thank you!';

END;

END;

END;

-- Execute the trigger by simulating a flight delay (assuming FlightID = 10)

UPDATE Flight SET DepartureTime = '2024-01-18T22:45:00' WHERE FlightID = 10;

**Output:**

(1 row affected)

Dear Passengers! We would like to notify you that Flight 10 has been delayed by 2 hours. We appreciate your patience. Thank you!

(1 row affected)

Completion time: 2024-01-15T19:05:22.2326195+05:00

## 10.4: Luggage Weight Check Trigger:

-- If the weight of luggage exceeds the limit set by the airline, a trigger is generated

CREATE TRIGGER LuggageWeightCheck

ON Luggage

FOR INSERT

AS

BEGIN

IF ((SELECT Weight FROM INSERTED) > 35)

PRINT 'Dear Passenger! Your luggage weight exceeds the limit we allow, please either go for cargo option or pay for your extra luggage.'

ELSE

PRINT 'Dear Passenger! Your luggage has been taken in.'

END;

-- Execute the trigger by inserting a new luggage item

INSERT INTO Luggage (LuggageID, Type, Weight)

VALUES (22, 'Suitcase', 40);

**Output:**

Dear Passenger! Your luggage weight exceeds the limit we allow, please either go for cargo option or pay for your extra luggage.

(1 row affected)

Completion time: 2024-01-15T19:09:21.3392748+05:00

## 10.5: Fuel Monitoring Trigger:

-- A trigger is generated if the fuel in a plane exceeds the normal amount

create trigger FuelMonitoring

on FuelingStation

after insert

as

begin

if ((select max(Capacity) from FuelingStation) > 250000)

begin

print 'Error! The fuel capacity exceeds the normal amount, fix the issue before taking off'

end

else

begin

print 'Fuel capacity is within the normal range. Safe to proceed.'

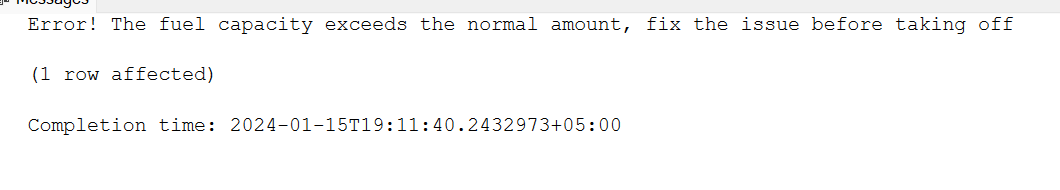
end

end;

-- Insert a new row into FuelingStation (assuming FuelingStationID = 22)

INSERT INTO FuelingStation (FuelingStationID, Name, Capacity, AircraftsAttended, FuelType)

VALUES (22, 'Fuel Station 1', 260000, 5, 'Jet-A');



# 11. Indexing:

## 11.1: Clustered Indexing:

Clustered indexes determine the physical order of the data in a particular database table. When a particular table has been indexed in this way, the data of that table has been arranged as per the key values of that index. This is done for quickly retrieving the data from a particular table, making our database search faster and more efficient.

In SQL Server Management Studio, the clustered index is automatically created on the primary key of the table as it sorts the data in a particular table in ascending order. We don’t need to create clustered indexes on every table if that table has a primary key. This is also true in the case of the tables consisting of composite primary keys as they are also automatically arranged on the disk. We can verify if the clustered index exists on our table via the following command:

-- Replace 'YourTable' with the actual table name

EXEC sp\_helpindex 'YourTable';

To see if the clustered index really works on our table, we can then execute the following command and manually see if the primary keys are arranged in ascending order.

-- Replace 'TABLE' with the actual table name

SELECT \* FROM TABLE;

## 11.2: Non-clustered Indexing:

In non-clustered indexing, the physical order of the data on the disk does not match the order of the index key. For this reason, a separate table or a structure is created to store the index, and it contains the list of references that point to the actual rows with data. Unlike clustered indexing, we can create non-clustered indexes on multiple rows of a table other than primary keys for faster retrieval of important data. This will speed up the execution of the entire database.

### 11.2.1: Indexing the flights:

You might have seen in the airports that the flights are arranged according to their departure or arrival times. Flights are being arranged via their departure time here.

-- Create a nonclustered index on the Flight table for DepartureTime

CREATE NONCLUSTERED INDEX IX\_NonClusteredFlightIndex

ON Flight (DepartureTime);

The flights are considered as departing if their departure time is not null.

### 11.2.2: Indexing the luggage:

We are ordering the luggage by its weight and not letting the data enter into the table if the luggage exceeds the total weight of 50 kg.

-- Create a nonclustered index on the Luggage table

CREATE NONCLUSTERED INDEX IX\_NonClusteredLuggageIndex

ON Luggage (Weight)

WHERE Weight <= 50;

### 11.2.3: Indexing the pilot:

We are going to order the pilot based on his/her experience flying.

-- Create a nonclustered index on the Pilot table

CREATE NONCLUSTERED INDEX IX\_NonClusteredPilotIndex

ON Pilot (HoursExperience);

### 11.2.4: Indexing the Ticket:

We are going to index the ticket based on its price.

-- Create a nonclustered index on the Ticket table

CREATE NONCLUSTERED INDEX IX\_NonClusteredTicketIndex

ON Ticket (Price);

### 11.2.5: Indexing the fueling station:

-- Create a nonclustered index on the FuelingStation table

CREATE NONCLUSTERED INDEX IX\_NonClusteredFuelingStationIndex

ON FuelingStation (Capacity, AircraftsAttended);

### 11.2.6: Executing the non-clustered indexes & verification:

To execute the non-clustered index, we can use the following command:

execute index\_name table\_name;

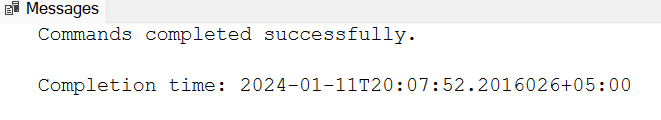
To verify if the nonclustered index has worked, execute the following command:

SELECT \* FROM table\_name;

# 12. Database Implementation:

This numbered point acts as a proof of the implementation of the database.

## 12.1: Database Creation:



## 12.2: Table Creation & Data Population:

(20 rows affected)

(20 rows affected)

(20 rows affected)

(20 rows affected)

(20 rows affected)

(20 rows affected)

(20 rows affected)

(20 rows affected)

(20 rows affected)

(20 rows affected)

(20 rows affected)

(20 rows affected)

(20 rows affected)

(20 rows affected)

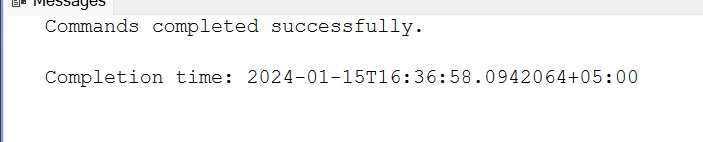
(20 rows affected)

Completion time: 2024-01-11T20:09:47.8818135+05:00

## 12.3: Indexes:

Implementing the indexes:

Status: All indexes have been implemented.



## 12.4: Views:

Status: Implemented. You can see the output in front of each view.

## 12.5: Procedures:

Status: Implemented. You can see the output in front of each procedure.

## 12.6: Triggers:

Status: Implemented. You can see the output in front of each trigger.

## 12.7: Authentication:

Status: Implemented. You can see the output in front of each authentication part.

# 13: Objectives:

The objective of this database includes efficiently handling passenger information, flight schedules, aircraft availability, pilot’s flying schedules, security checks, and the overall functionalities of an airport.

# 14: Scope of the database:

The scope of this system is to encompass diverse areas like passenger management, flight operations, security protocols, resource allocation, maintenance, tuning & reporting of the resources. It aims to integrate functions for effective & seamless airport operations.

# 15: Key Functionalities:

* **Passenger Management:**

It handles passenger management, ticketing & check-ins..

* **Flight Operations:**

It manages flight schedules, real-time flight tracking & pilot assignments.

* **Baggage Tracking:**

Ensuring accurate tracking & handling of passenger baggage.

* **Security Protocols:**

Implementing & monitoring security measures to ensure the safety of passengers & airport facilities.

* **Maintenance Tracking:**

Scheduling & tracking maintenance activities for airport equipment.

* **Communication:**

Facilitating communication between airport departments & external entities.

# 16. Performance & verification of database:

## 16.1: Relationship check:

-- Checking relationships of the table (subqueries)

SELECT

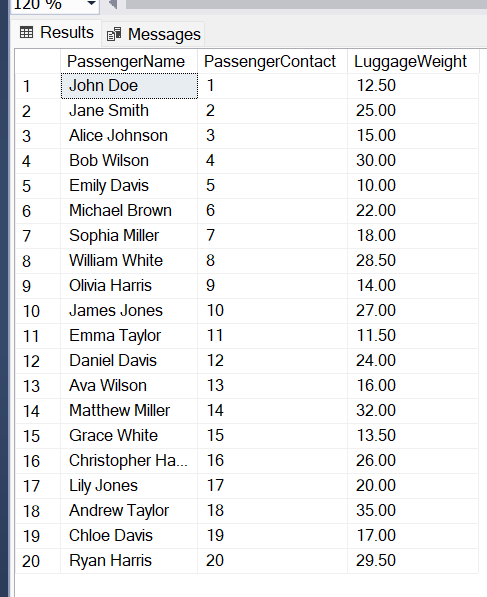
(SELECT Name FROM Passenger WHERE PassengerID = P.PassengerID) AS PassengerName,

(SELECT P\_Contact FROM PassengerContact WHERE PassengerID = P.PassengerID) AS PassengerContact,

(SELECT Weight FROM Luggage WHERE LuggageID = P.LuggageID) AS LuggageWeight

FROM

Passenger AS P;



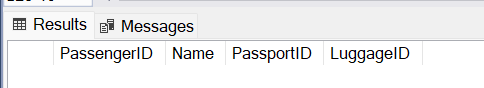
## 16.2: Checking the foreign key violation:

-- Subquery to check foreign key constraints for the Passenger table

SELECT \*

FROM Passenger

WHERE LuggageID IS NOT NULL AND LuggageID NOT IN (SELECT LuggageID FROM Luggage);



Returns nothing, which means that there is no foreign key violation.

## 16.3: Checking the logical errors:

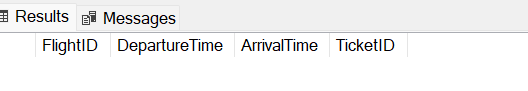
-- Checking logical errors:

### 16.3.1: Departure Time

SELECT \*

FROM Flight

WHERE DepartureTime > ArrivalTime;



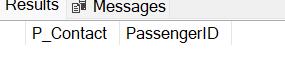
This means that there is no flight that departs later than it arrives.

### 16.3.2: Passenger Contact:

SELECT \*

FROM PassengerContact

WHERE PassengerID NOT IN (SELECT PassengerID FROM Passenger);



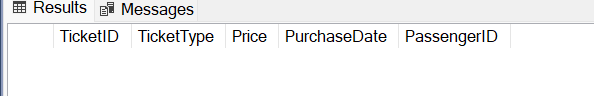
The Above query returns nothing, which means that there are only passengers in the PassengerContact table that exist in the Passenger table.

### 16.3.3: Passenger Ticket:

SELECT \*

FROM Ticket

WHERE PassengerID NOT IN (SELECT PassengerID FROM Passenger);



There is no such ticket of a passenger that does not exist in the passenger table.

## 16.4: Integrity check:

### 16.4.1: Pilot and Flight:

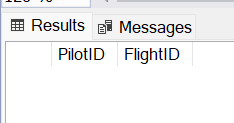
SELECT \*

FROM PilotFlightAssociation

WHERE PilotID NOT IN (SELECT PilotID FROM Pilot)

OR FlightID NOT IN (SELECT FlightID FROM Flight);

During the normalization process, the new table PilotFlightAssociation was made due to many-many relationships. We need to verify if the same Pilots and Flights exist in this table that were in the Pilot and Flight tables, which in our case is true.



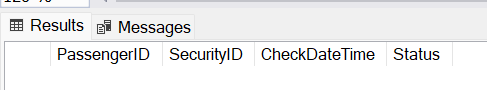
### 16.4.2: Security and Passenger:

SELECT \*

FROM PassengerSecurityAssociation

WHERE PassengerID NOT IN (SELECT PassengerID FROM Passenger)

OR SecurityID NOT IN (SELECT SecurityID FROM Security);

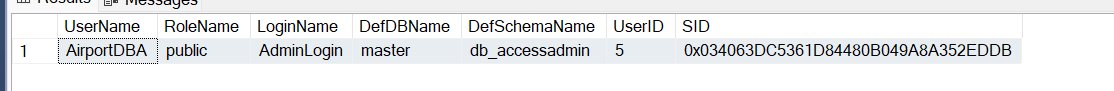


During the normalization process, the new table PassengerSecurityAssociation was made due to many-many relationships. We need to verify if the same Passengers and Security exist in this table that were in the Passenger and Security tables, which in our case is true.

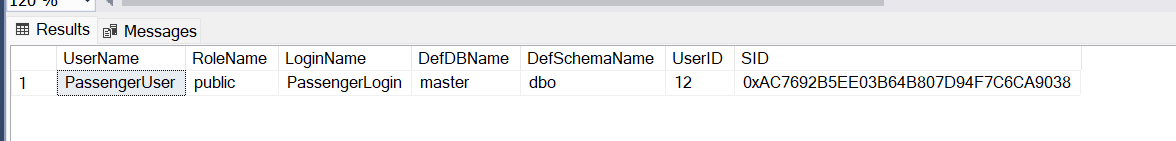
## 16.5: Verifying the users:

We will verify if the users exist and if they are allowed to do what we assigned them to do.

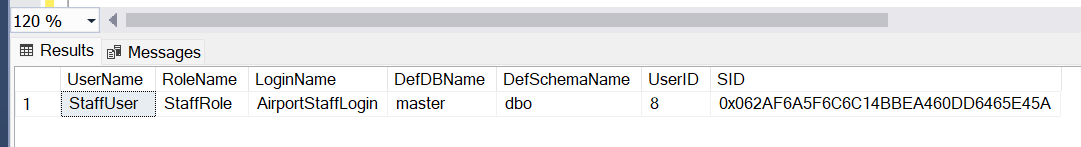
EXEC sp\_helpuser 'AirportDBA';



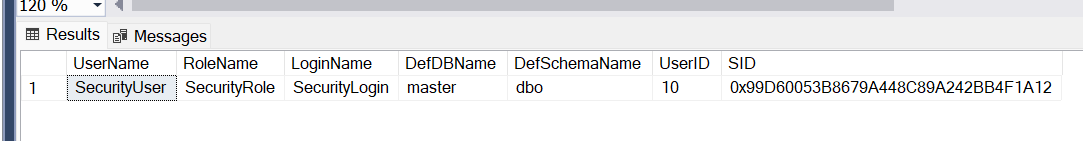
EXEC sp\_helpuser 'PassengerUser';



EXEC sp\_helpuser 'StaffUser';



EXEC sp\_helpuser 'SecurityUser';



### 16.5.1: Passenger:

USE AirportManagementSystem;

-- Passenger user doing the things on its end.

SELECT name

FROM sys.database\_principals

WHERE name = 'PassengerUser';

-- Passenger user getting his data:

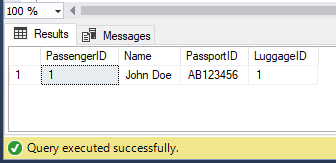
SELECT \* FROM Passenger WHERE ID = 1;

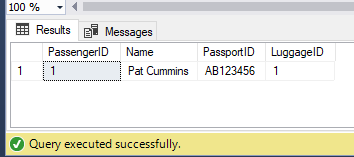
-- Passenger user updating his data:

UPDATE Passenger SET Name = 'Pat Cummins' WHERE PassengerID = 1;

-- Revert to the original user

REVERT;





### 16.5.2: Airport Staff:

--2:

USE AirportManagementSystem;

-- Airport Staff user connecting to DB:

EXECUTE AS USER = 'StaffUser';

-- Staff member getting his data:

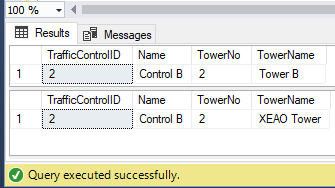
SELECT \* FROM TrafficControl WHERE TrafficControlID = 2;

-- Staff member updating the data:

UPDATE TrafficControl SET TowerName = 'XEAO Tower' WHERE TrafficControlID = 2;

-- Revert to the original user

REVERT;



### 16.5.3: Airport Staff user connecting to DB:

EXECUTE AS USER = 'AirportDBA';

-- Admin getting the data:

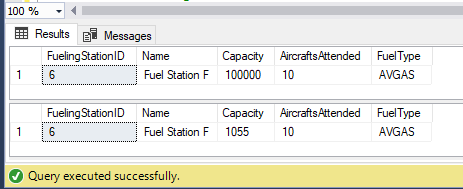
SELECT \* FROM FuelingStation WHERE FuelingStationID = 6;

-- Admin updating the capacity:

UPDATE FuelingStation SET Capacity = 1055 WHERE FuelingStationID = 6;

-- Revert to the original user

REVERT;



### 16.5.4: Security:

Security personnel connecting to DB:

EXECUTE AS USER = 'SecurityUser';

-- Security getting the data of Passenger:

SELECT \* FROM Passenger WHERE PassengerID = 7;

SELECT \* FROM Ticket WHERE TicketID = 7;

SELECT \* FROM Luggage WHERE LuggageID = 7;

-- Revert to the original user

REVERT;

