DBAM Semester Project Report

Airport Management System



**Contributors:**

*Ambreen (F2021105009)*

*Abdullah Imran (F2021105135)*

# 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:

We are going to choose the Django Framework as a backend to build a very simple front end on our end.

# 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.

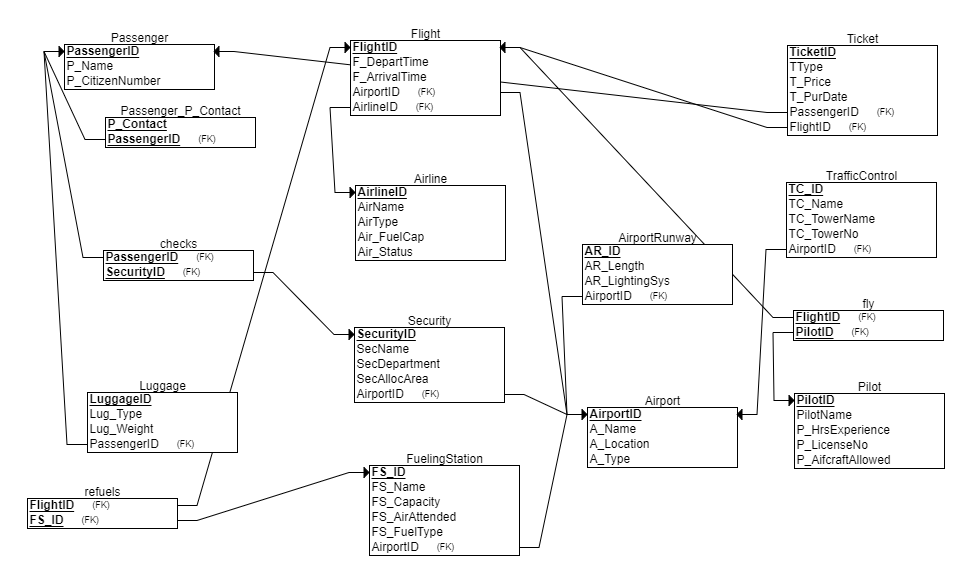
## 4.1: Building Relations:

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:

## 

## 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.

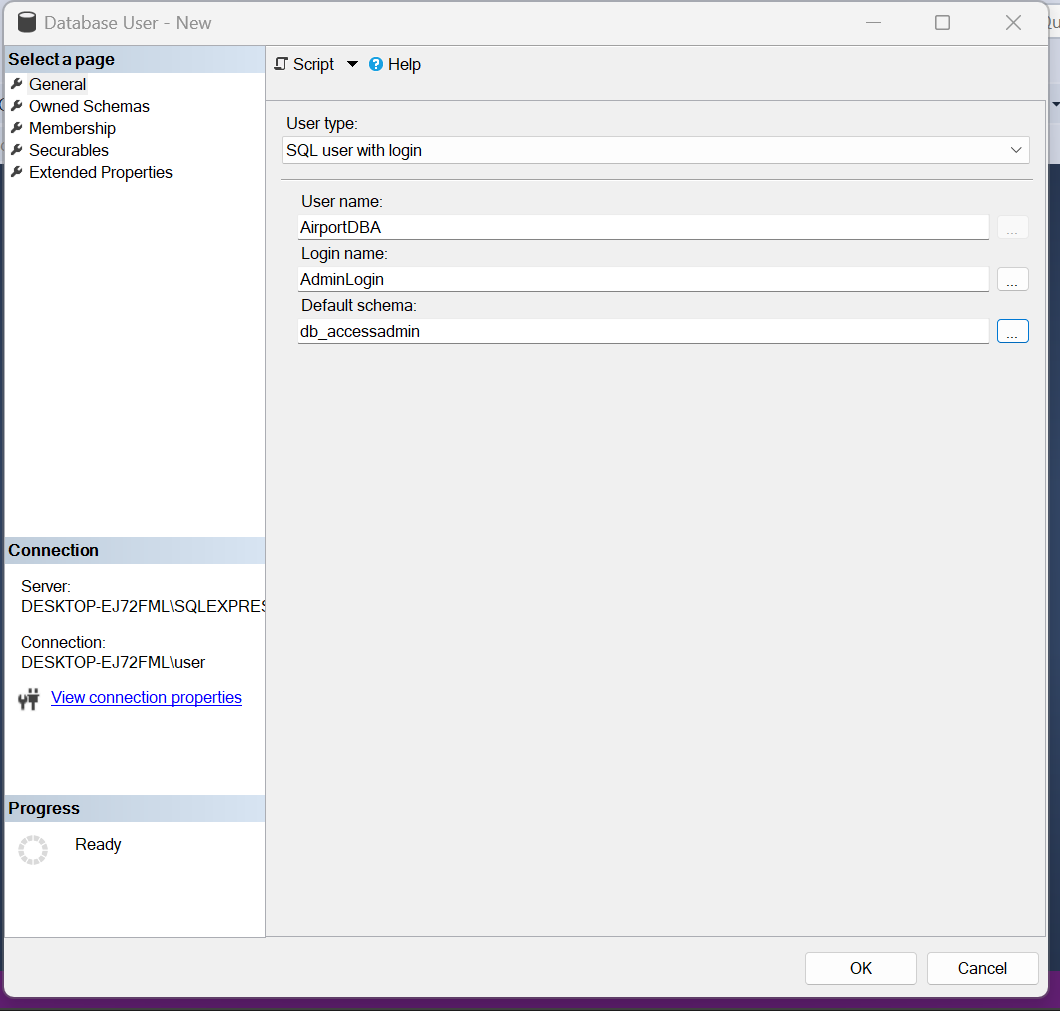
# 5. Authorization:

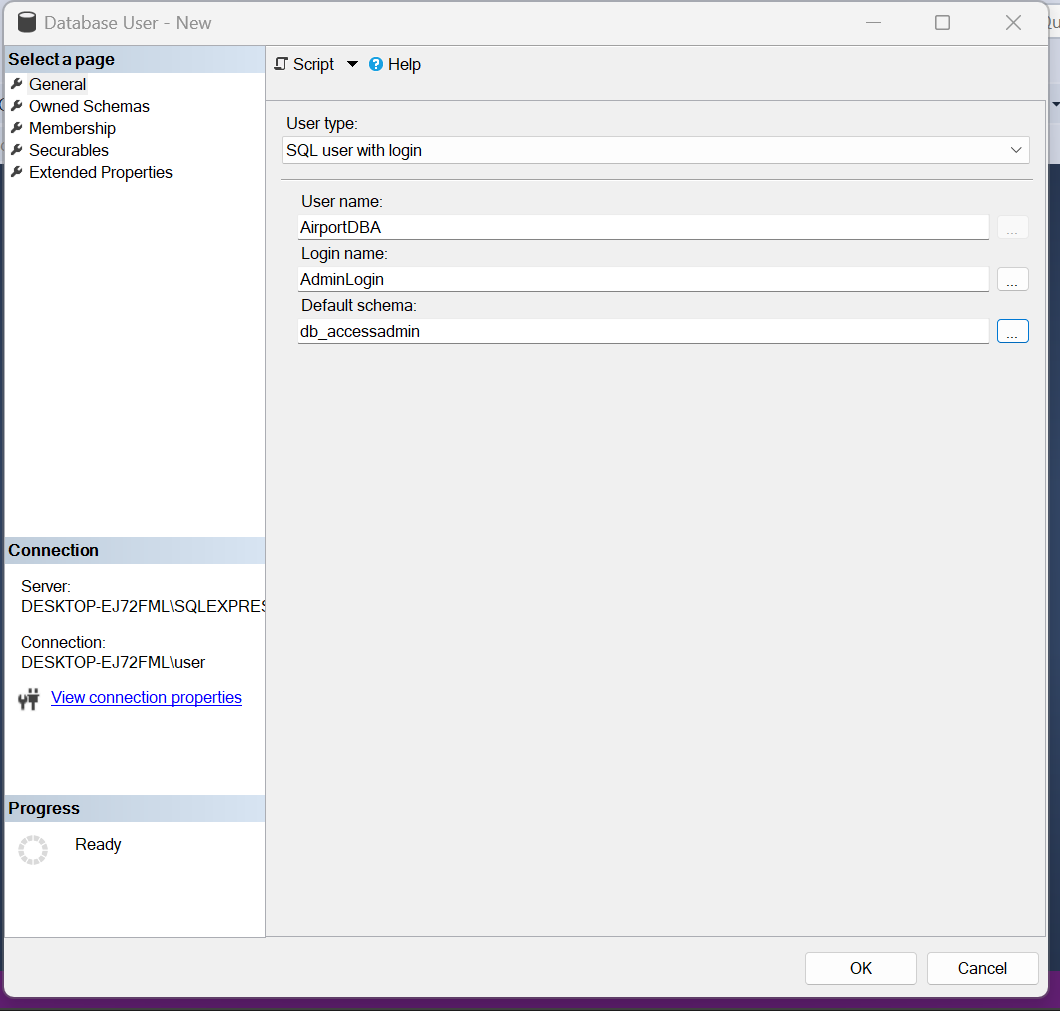
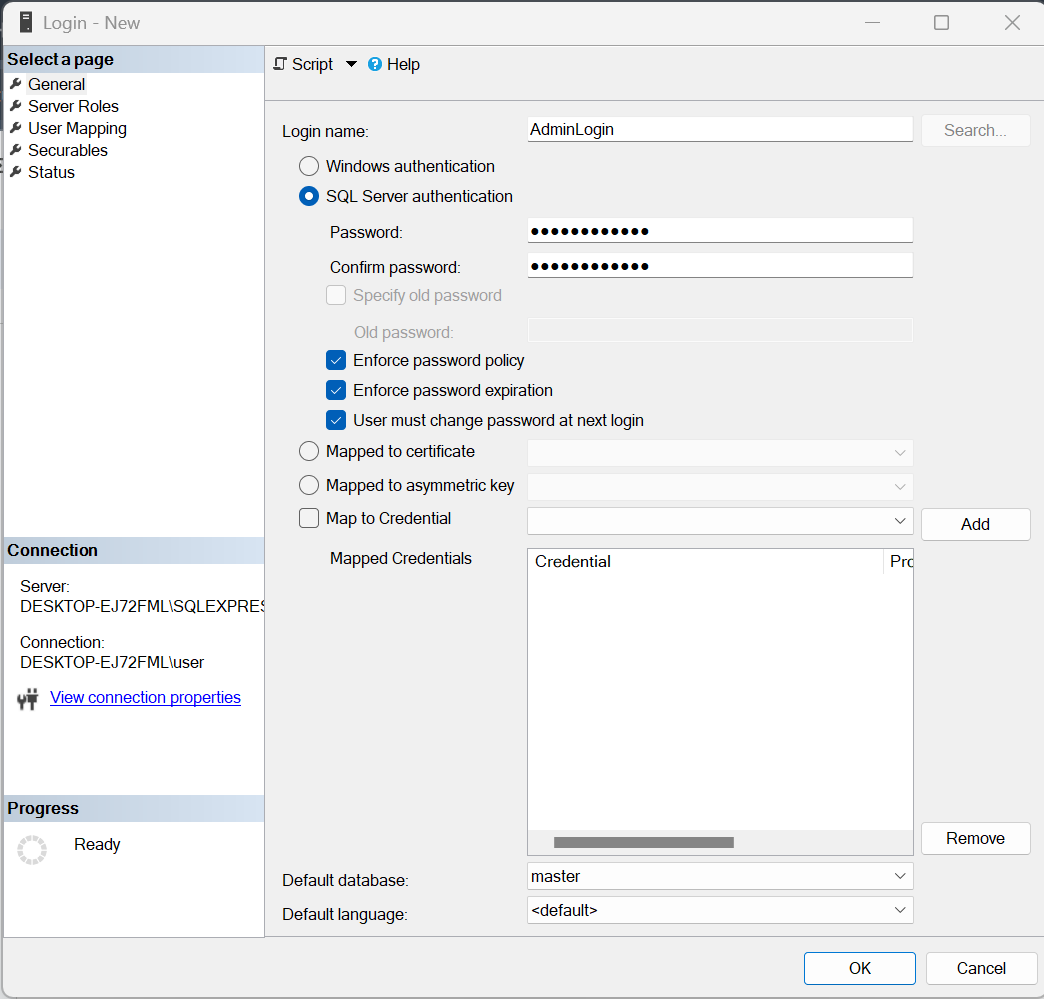
The authorization will be based on the created tables and the users that will use this database. So we will first need to create the logins of those users whom we target as DB users.

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

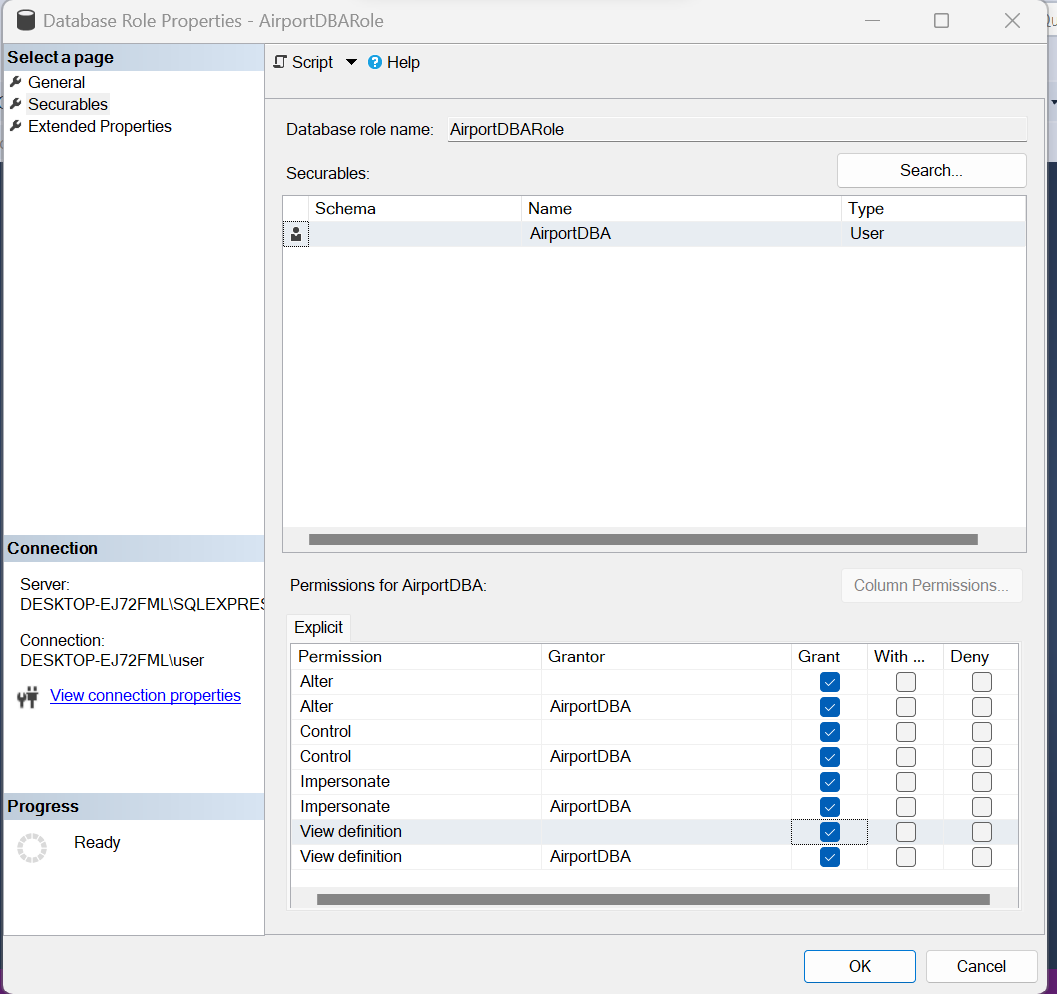
So, for simplicity purposes, we are going to create 4 types of user logins, one that will be used by the admins, one by the airport’s staff, one by the security, and the other for passengers or anyone outside the airport’s premises whom we want to target.

**5.1.1Admin Login & AirportDBA user:**

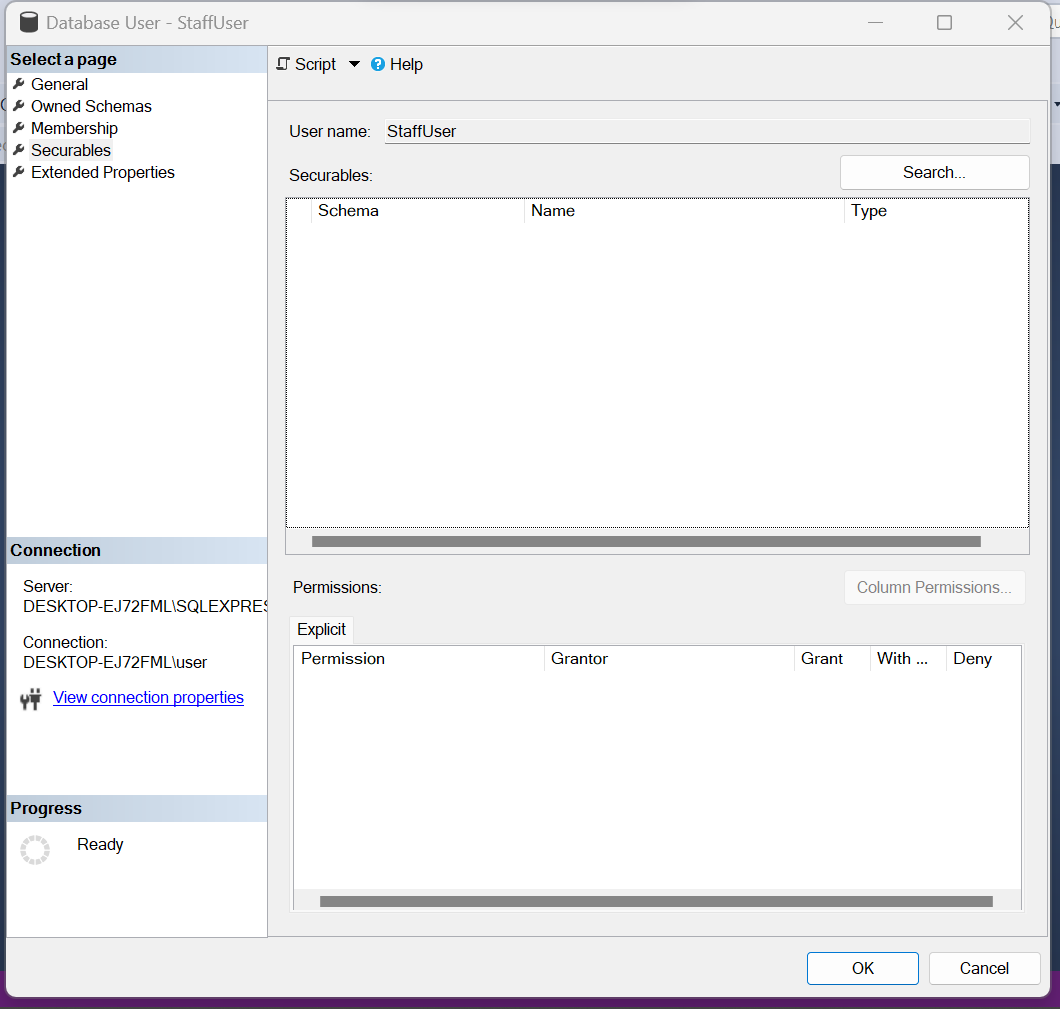


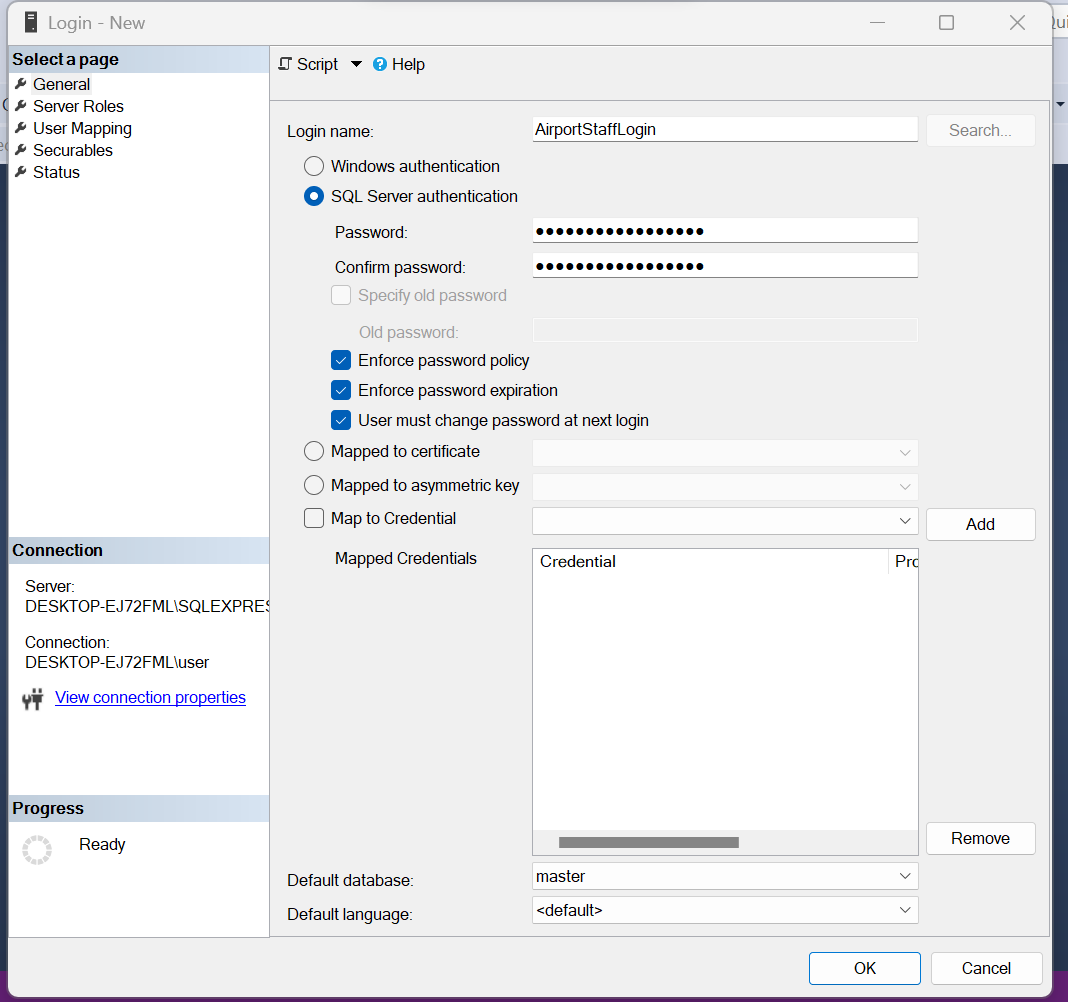


**DBARole:**



**AirportStaffLogin & StaffUser:**





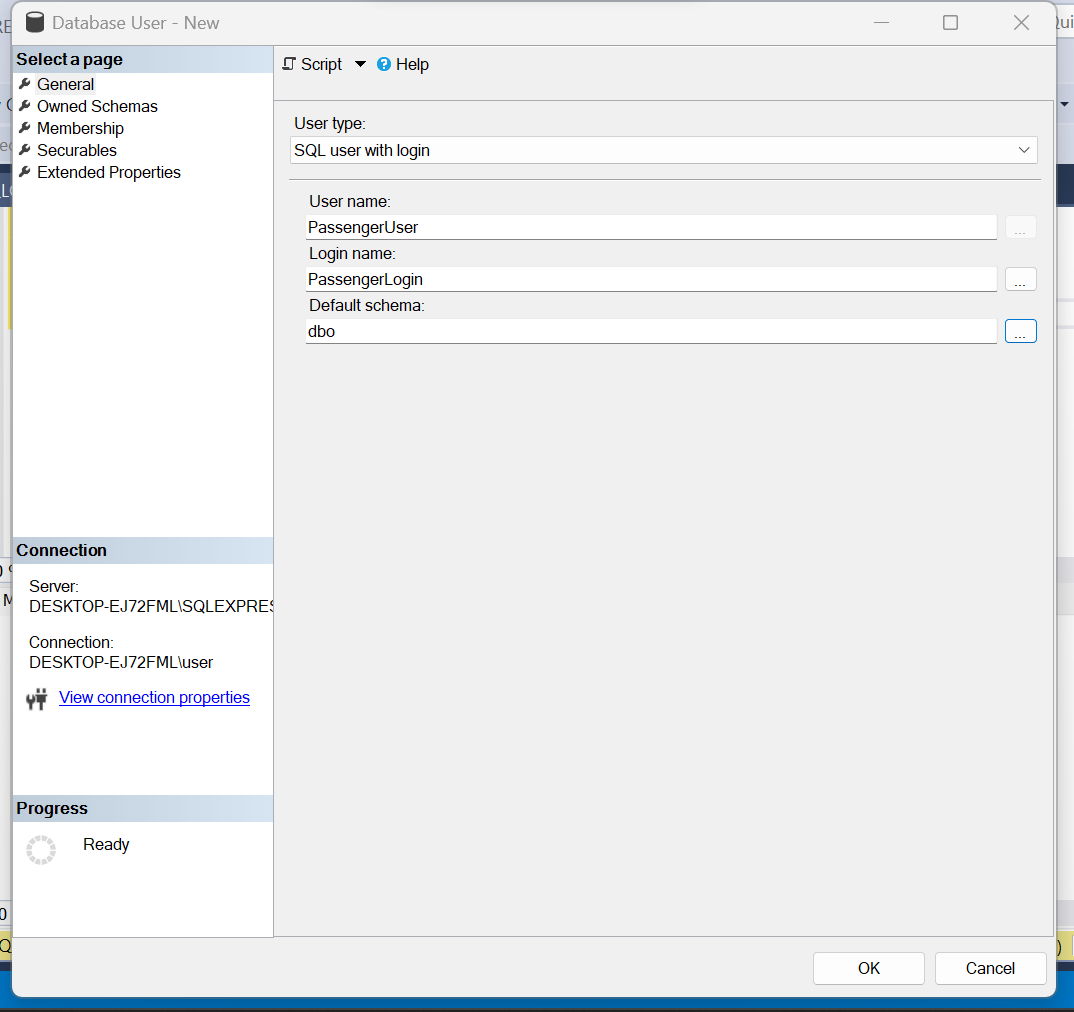
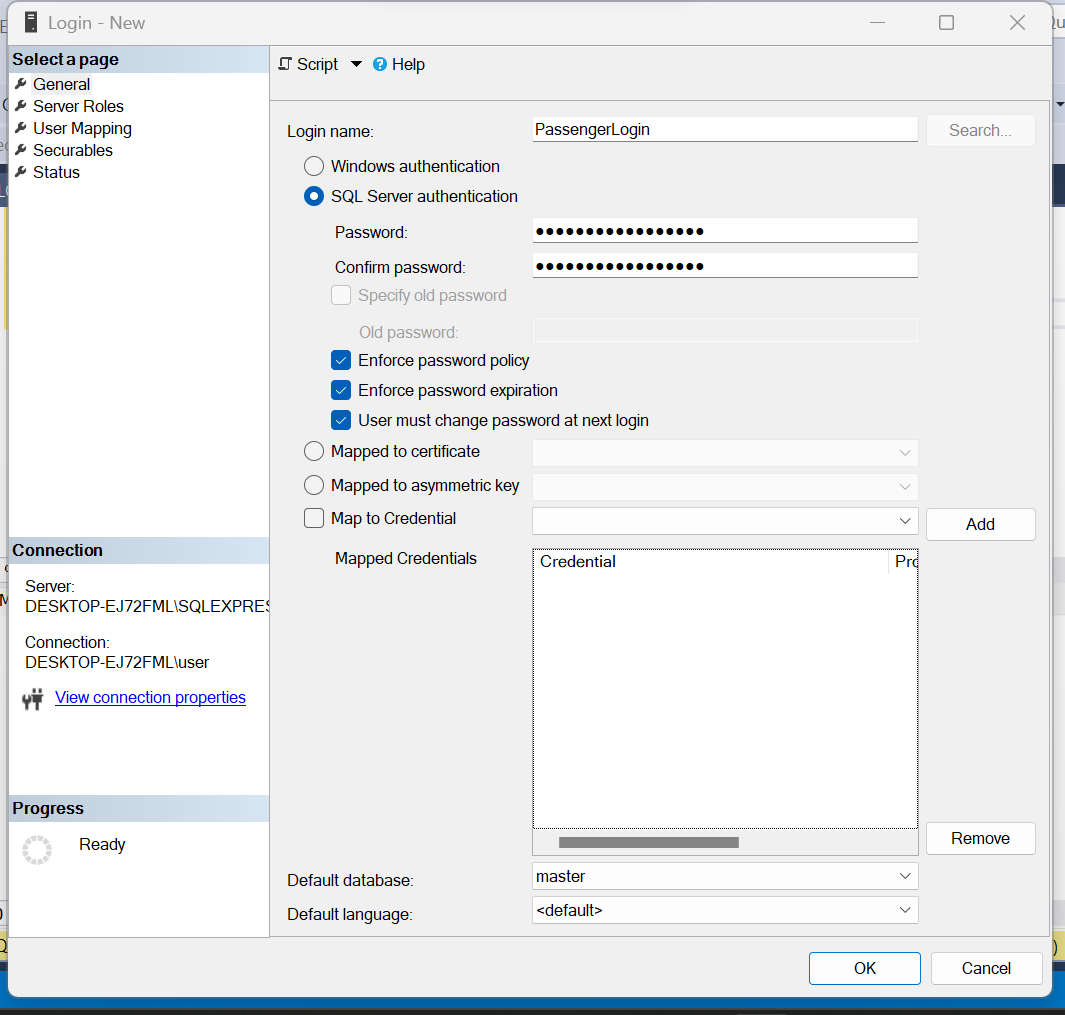
**StaffRole:**

create role StaffRole;

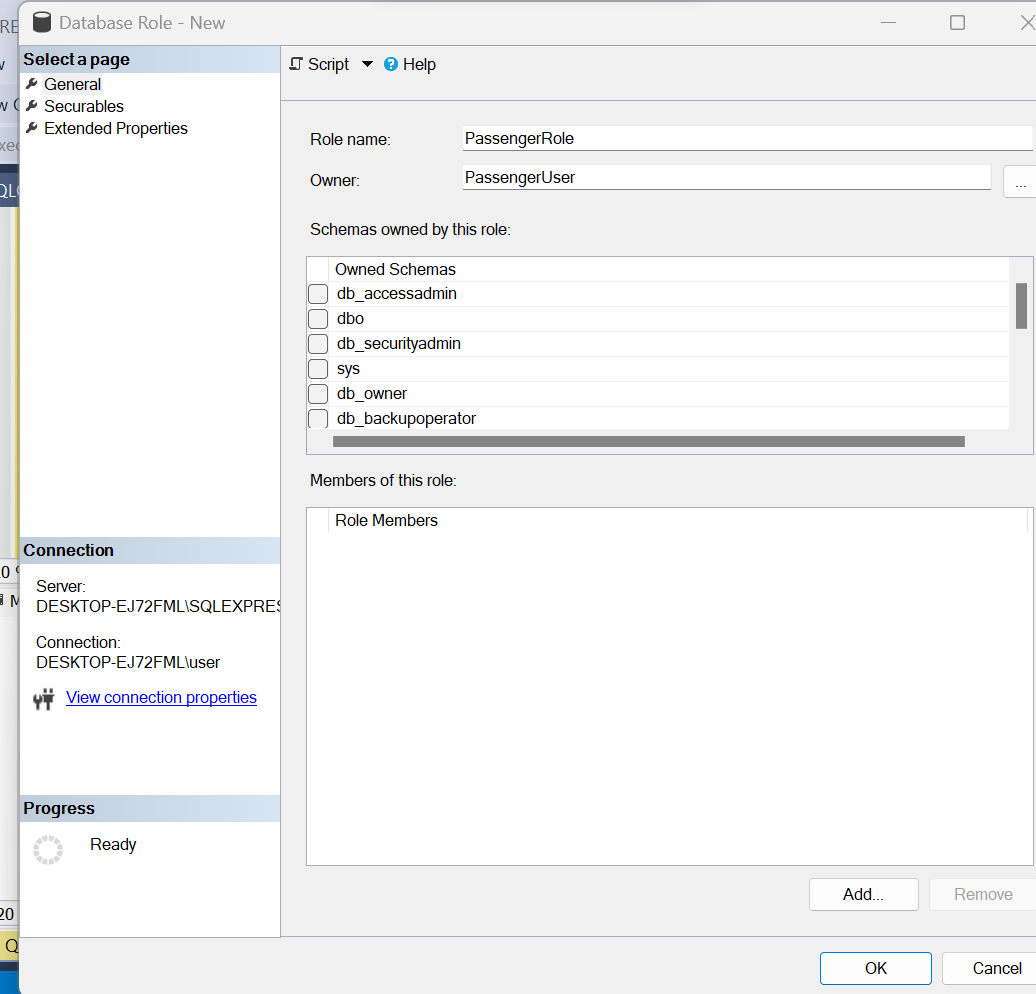
GRANT SELECT, UPDATE, INSERT ON SCHEMA::dbo TO StaffRole;

ALTER ROLE StaffRole ADD MEMBER StaffUser;

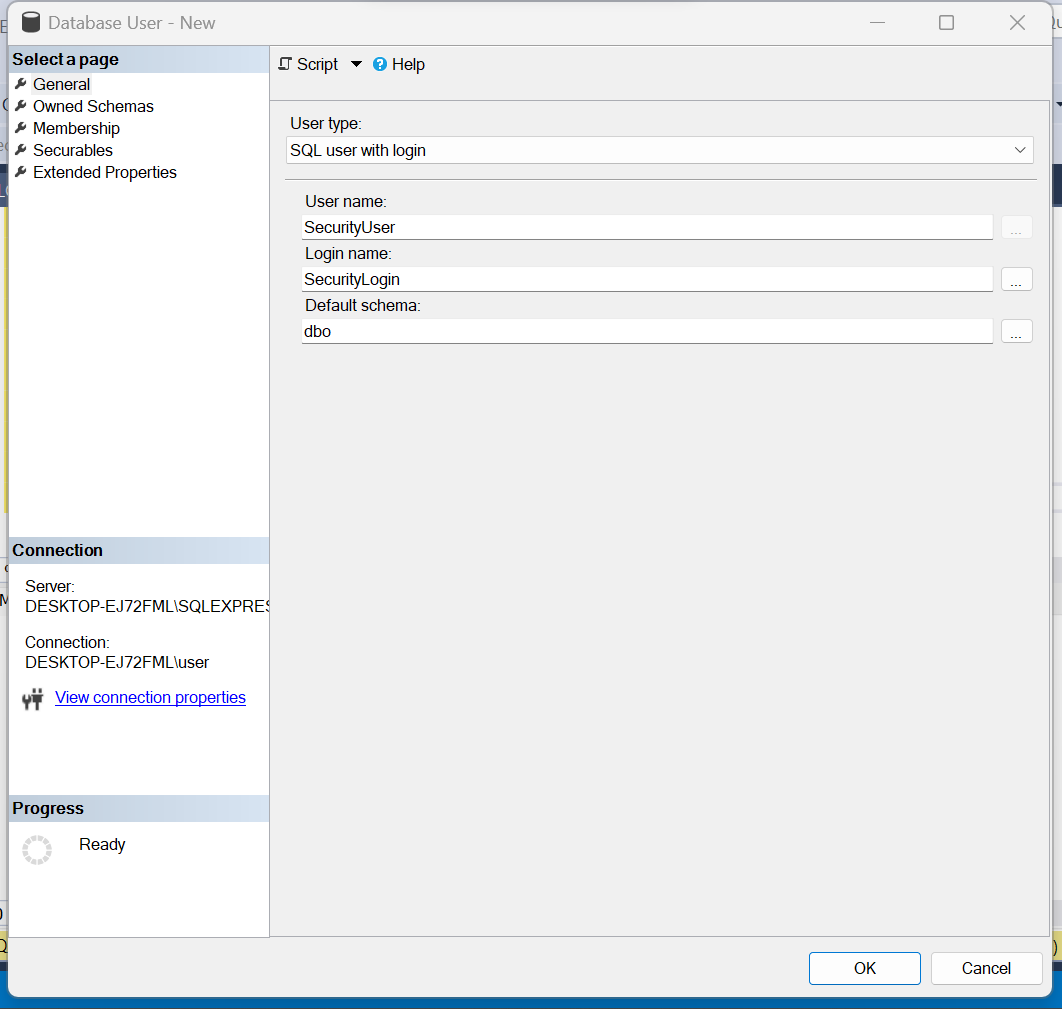
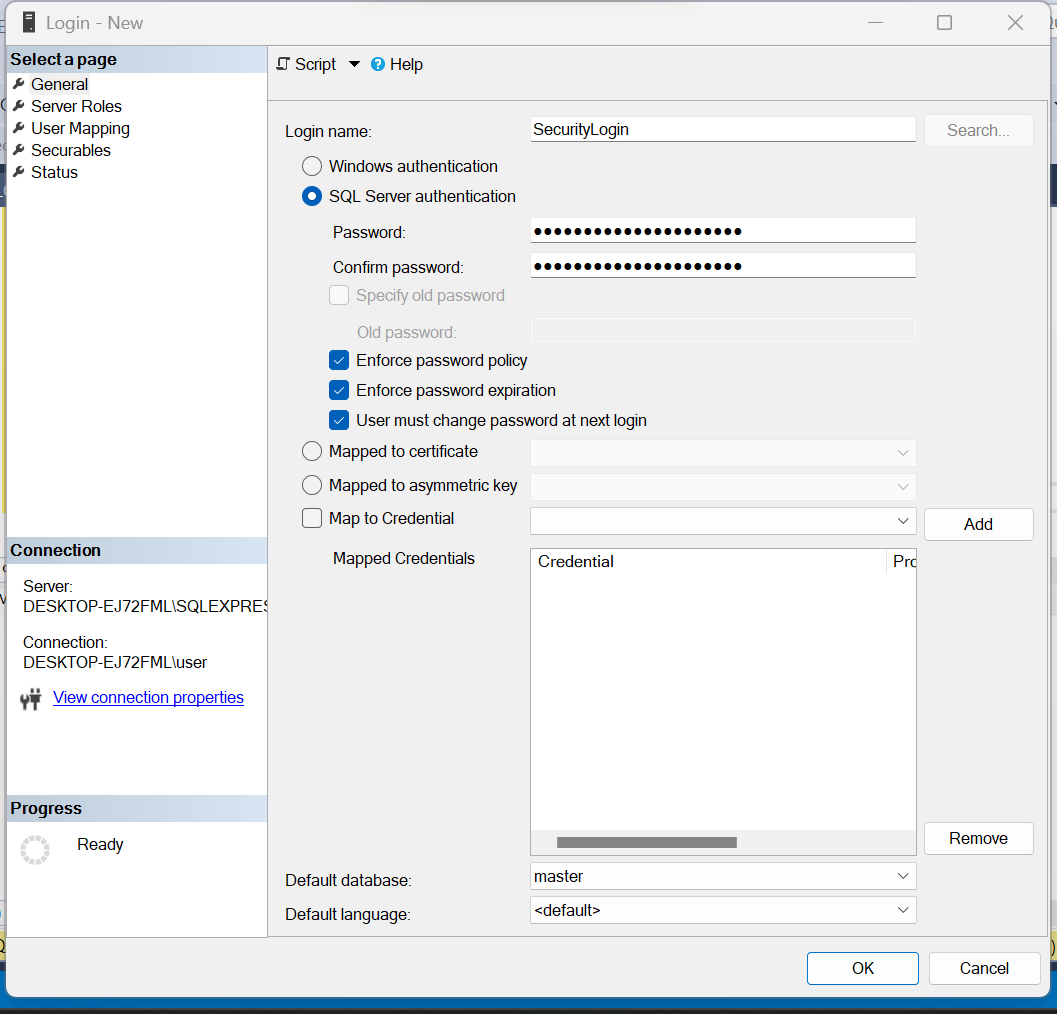
**PassengerLogin & PassengerUser:**



**PassengerRole:**



**SecurityLogin & SecurityUser (Pictures):**



**SecurityRole:**

CREATE ROLE SecurityRole;

GRANT SELECT ON SCHEMA::dbo TO SecurityRole;

ALTER ROLE SecurityRole ADD MEMBER SecurityUser;

# 6. Tables:

As per the given entities at the start, all of those tables have been created and each table consists of 20 different rows for data population purposes, keeping in mind the data integrity of the tables.

(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

# 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 processes 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:

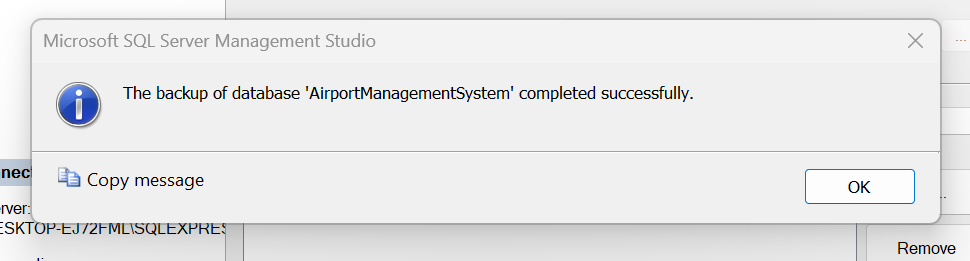
As our system expands, it will be important to have more than one off-site storage facility to have the recovery phase executed as quickly as possible. In the first phase, a new off-site facility will be located away from the city in a secure area for this purpose.

# 7.2: Backup process:

**Full backup:**

Backup location:

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



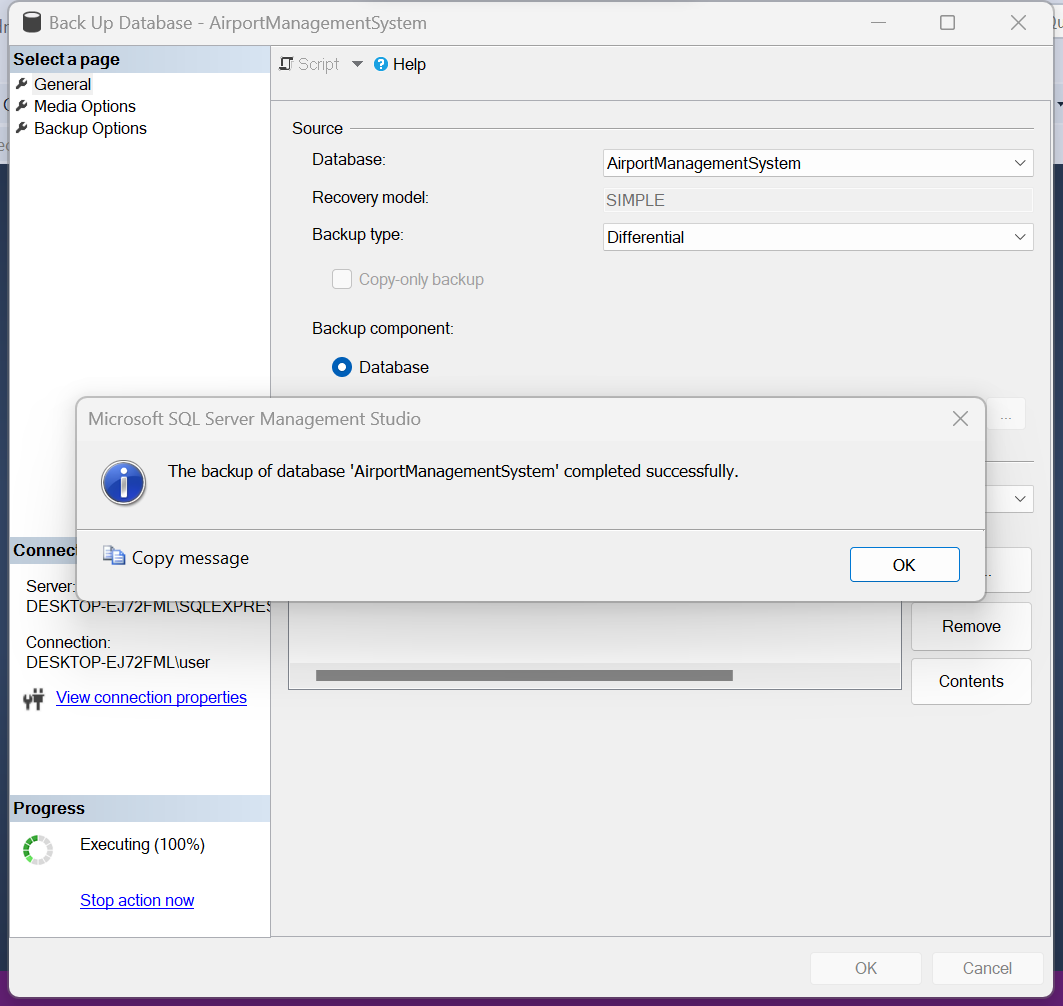


**Differential Backup:**

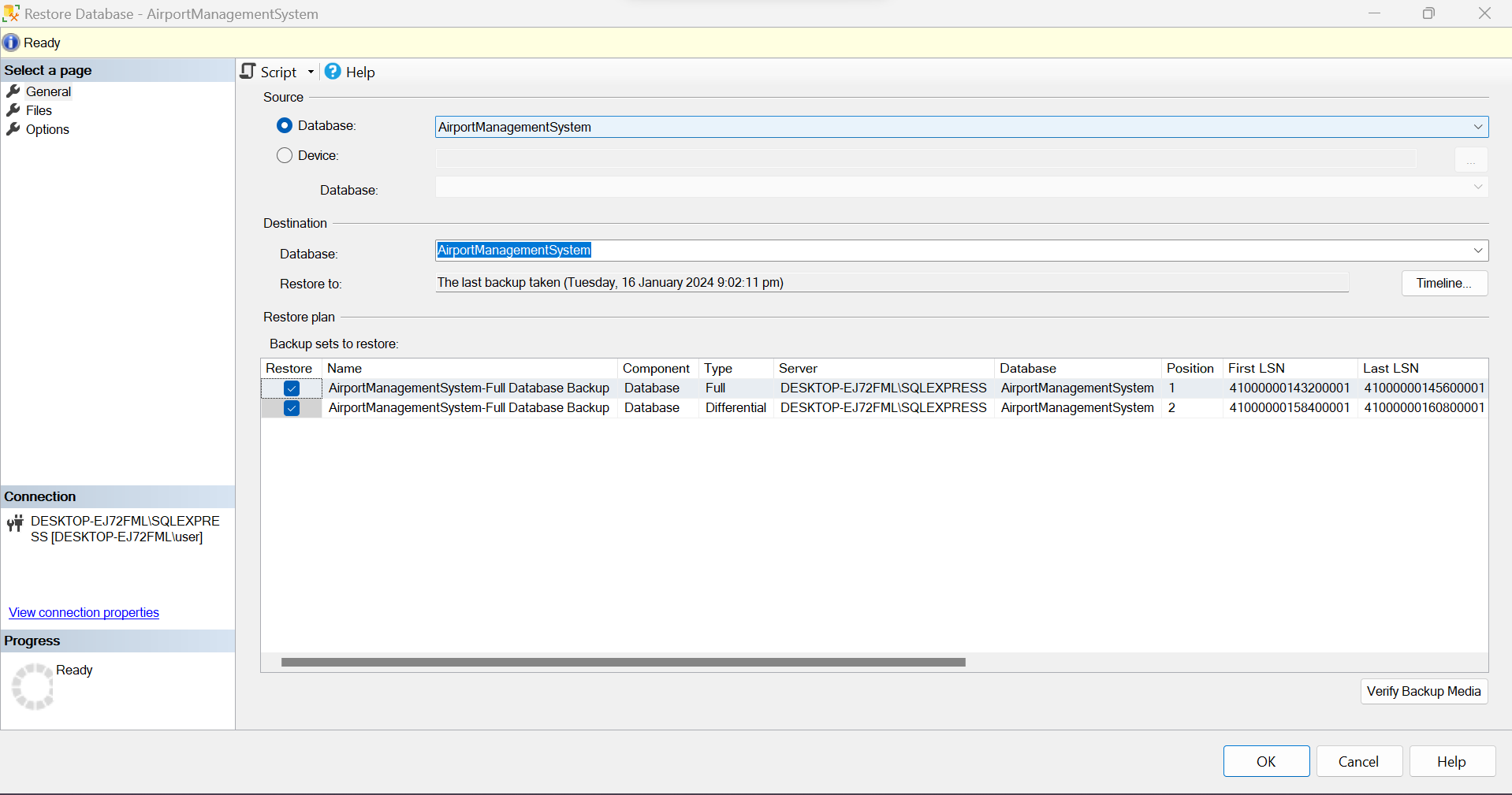
Backup location:

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





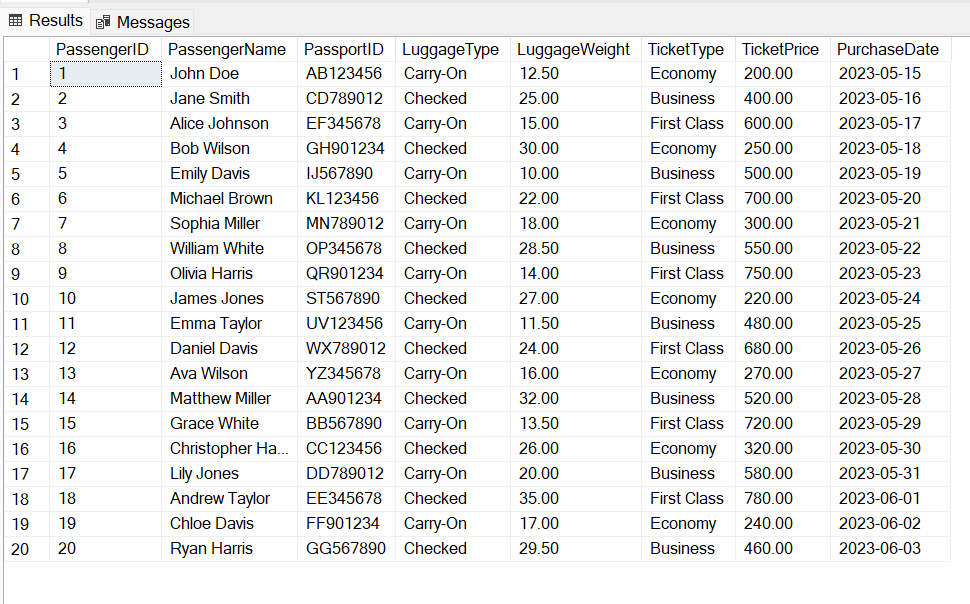
# 7.3: Recovery process:



# 8: Views & Joins:

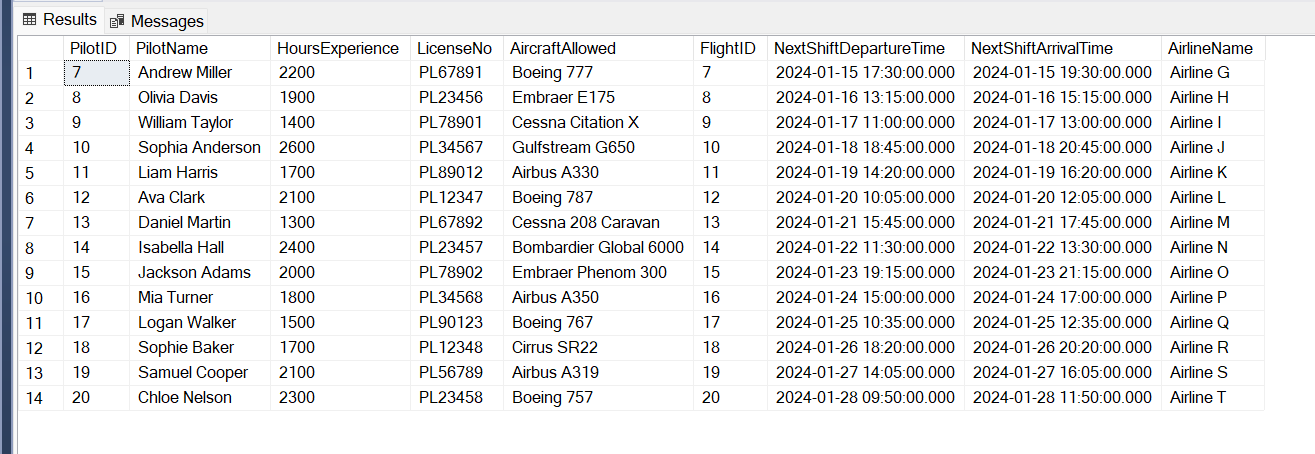
## 8.1: Passenger View:

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



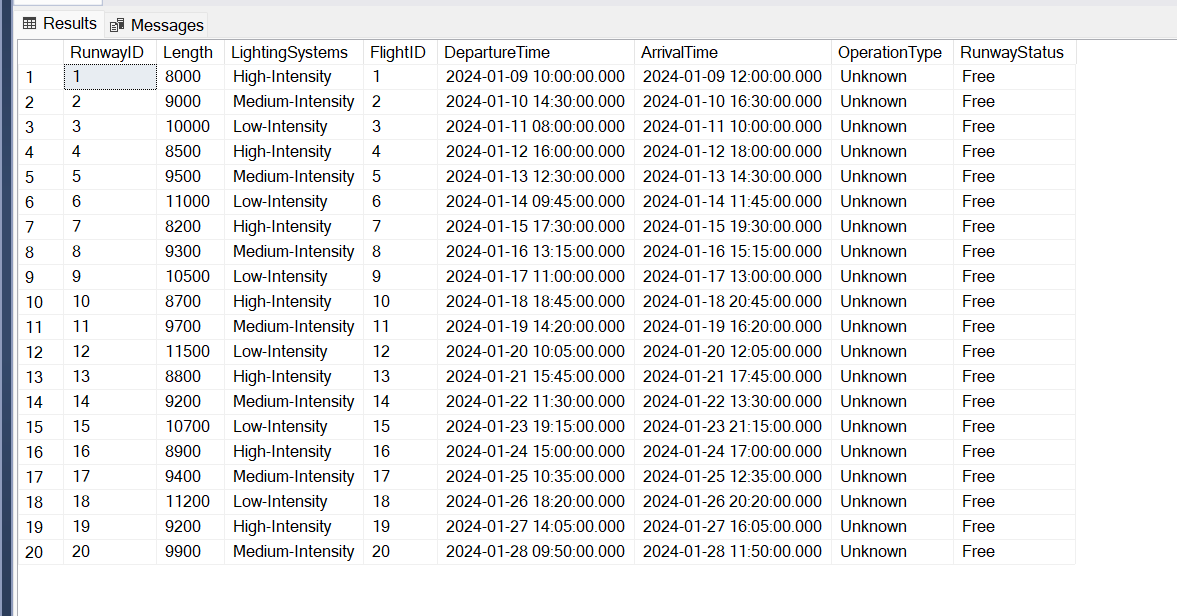
## 8.2: Pilot View:

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:



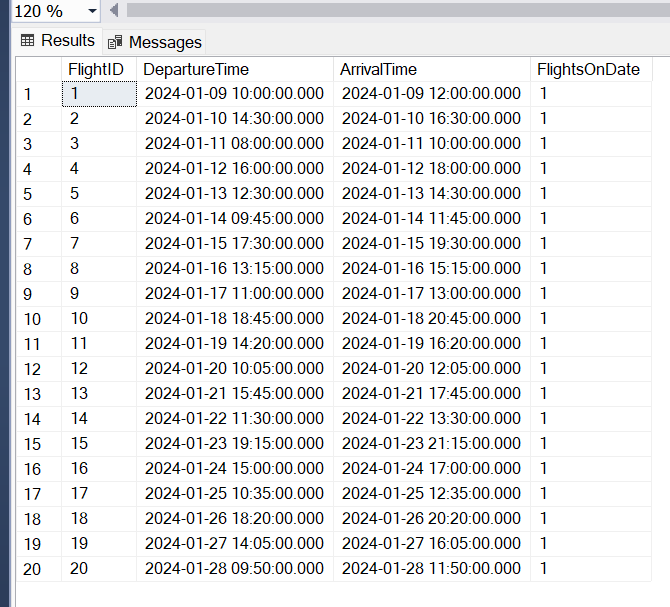
## 8.3: Runway View:

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.



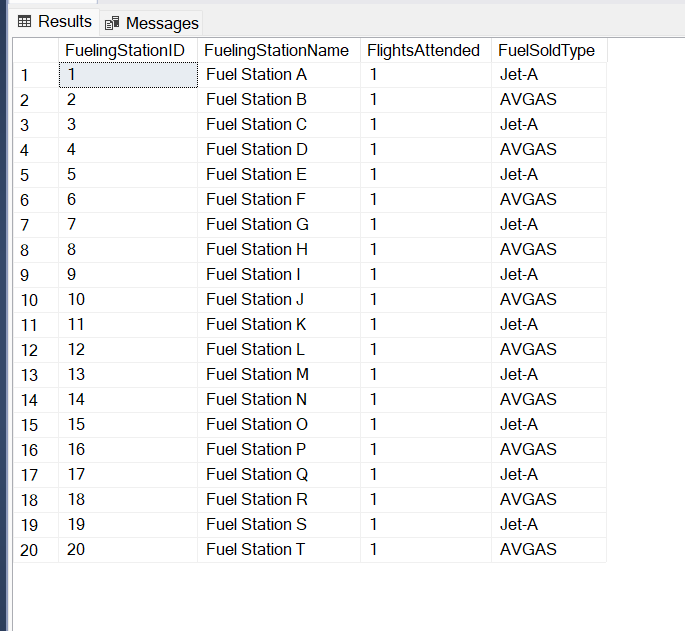
## 8.4: Flight View:

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



## 8.5: Fueling Station View:

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

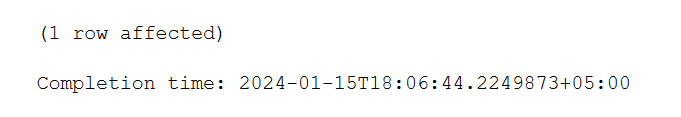


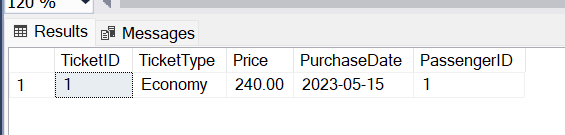
# 9. Stored Procedures:

## 9.1: Weight Check Procedure:

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

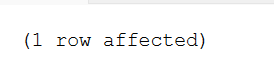
--than the limit set by the airport, the price of the ticket is increased

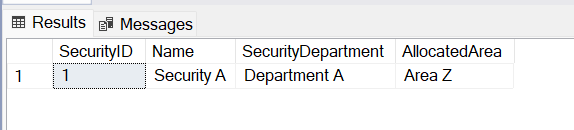




## 9.2: Security Area Changed Procedure:

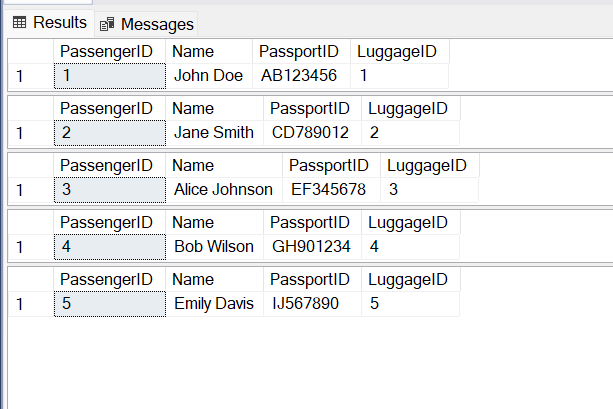
Sometimes when security personnel are on duty, they are ordered to change their locations for some reason. We are assuming this scenario and implementing this:





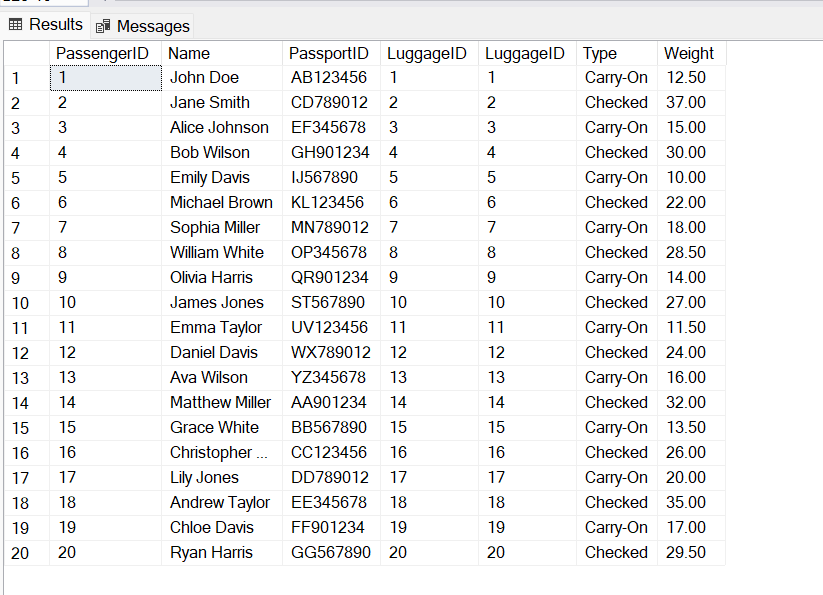
## 9.3: Print Details of First Five Passengers Procedure:

--Print the details of the first five passengers



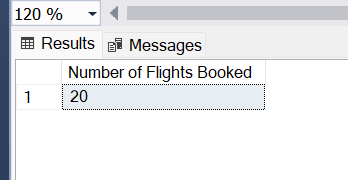
## 9.4: Show Information of Passengers & Their Luggage Procedure:

--Show the details of passengers and their luggage



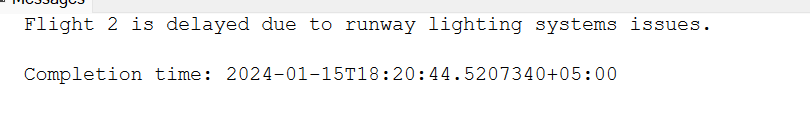
## 9.5: Number of Flights Booked Procedure:

--Show details of how many flights are already booked



## 9.6: Lighting System Check:

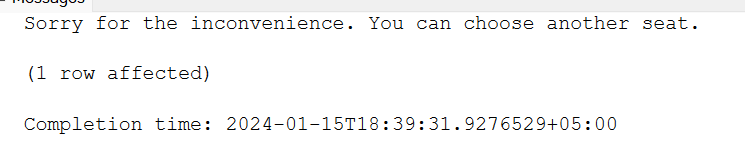
Check if the lighting systems are working fine. If not, then delay the flight.



# 10. Triggers:

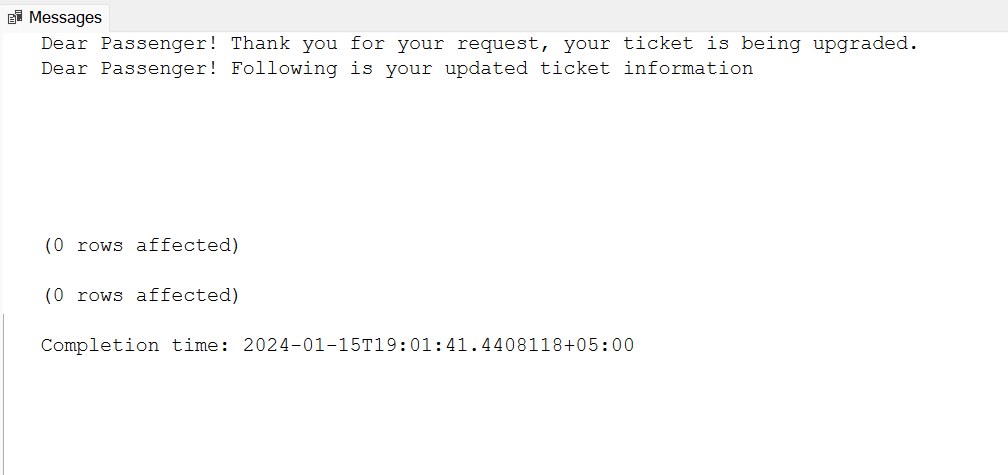
## 10.1: Passenger-Ticket Trigger:

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



## 10.2: Ticket Upgradation Trigger:

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



## 10.3: Flight Delay Trigger:

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

**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

**Output:**

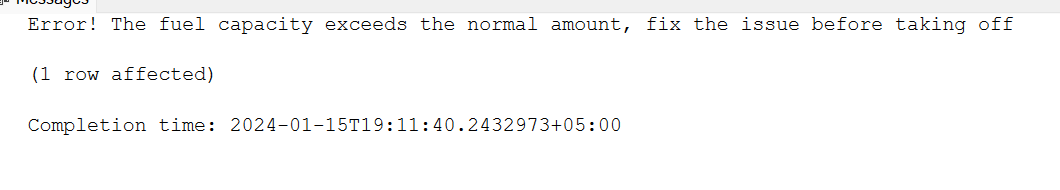
Dear Passenger! Your luggage weight exceeds the limit we allow, please either go for cargo option or pay for your extra luggage.

(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



# 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: 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.

# 13: 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.

# 14: 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.

# 15. Key interests:

It was our mutual consideration to do something that is slightly different, something out of the box. Hence, we went through various topics and then decided to pick one topic due to the following reasons:

* **Widely used:** Airport Management System is widely used throughout the world not only for passengers’ transportation but also for cargo and shipment purposes. It plays a pivotal role in the modern-day transportation industry and hence working on it would mean getting to know more about this particular industry.
* **Complex system:** When we develop the database system for the aviation industry, especially for the airports, we need to have this at the back of our mind that it will be working all the time, no matter what. This introduces some challenges that need to be kept in mind including the high data and server availability, making sure that the data integrity is ensured, and that the data is safe and difficult to access.
* **Generally interesting:** Airport Management System itself is an interesting subject, unlike the other topics that fade your interest as you work on it more and more.
* **Learning Opportunities:** I remember that when I was deciding on the entities, I looked at a lot of the possibilities and stuff that could identify an entity for our project. But we have obviously picked out a few of the entities that are the backbone for every airport to work as desired. Out of those, checking out how the luggage department, traffic control, fueling station, and runway departments work and what they do was probably one of the learning opportunities for me because as a passengers, we don’t really experience this firsthand.

# 16. Performance & verification of database:

## 16.1: Relationship check:

-- Checking relationships of the table (subqueries)

SELECT

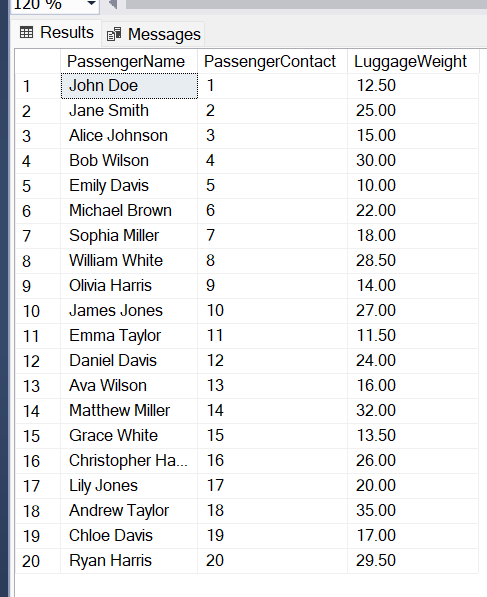
(SELECT Name FROM Passenger WHERE PassengerID = P.PassengerID) AS PassengerName,

(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;



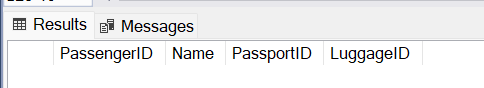
## 16.2: Checking the foreign key violation:

-- 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:

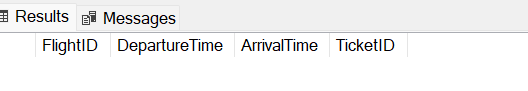
-- Checking logical errors:

### 16.3.1: Departure Time

SELECT \*

FROM Flight

WHERE DepartureTime > ArrivalTime;



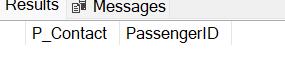
This means that there is no flight that departs later than it arrives.

### 16.3.2: Passenger Contact:

SELECT \*

FROM PassengerContact

WHERE PassengerID NOT IN (SELECT PassengerID FROM Passenger);



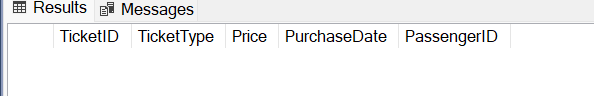
The Above query returns nothing, which means that there are only passengers in the PassengerContact table that exist in the Passenger table.

### 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:

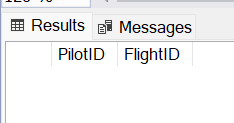
SELECT \*

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.



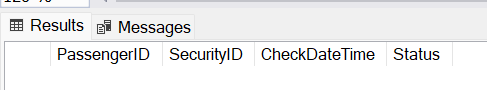
### 16.4.2: Security and Passenger:

SELECT \*

FROM PassengerSecurityAssociation

WHERE PassengerID NOT IN (SELECT PassengerID FROM Passenger)

OR SecurityID NOT IN (SELECT SecurityID FROM Security);

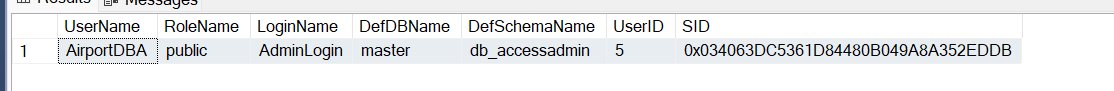


During the normalization process, the new table PassengerSecurityAssociation was made due to many-many relationships. We need to verify if the same Passengers and Security exist in this table that were in the Passenger and Security tables, which in our case is true.

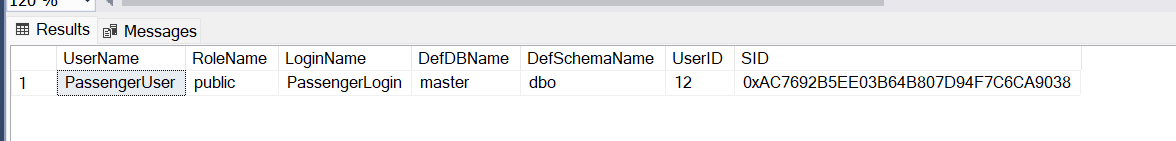
## 16.5: Verifying the users:

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

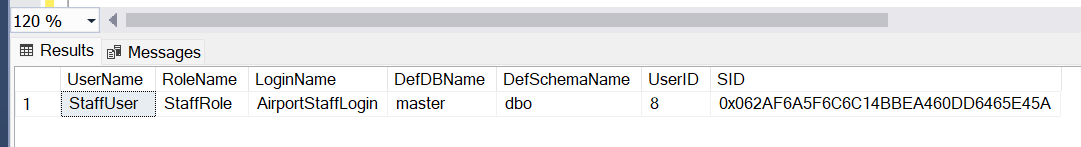
EXEC sp\_helpuser 'AirportDBA';



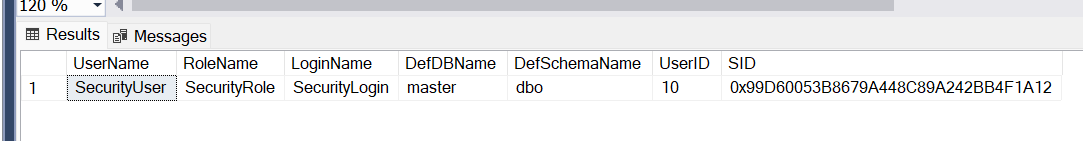
EXEC sp\_helpuser 'PassengerUser';



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:

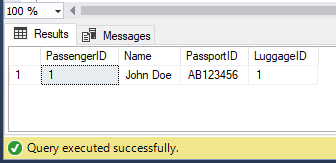
SELECT \* FROM Passenger WHERE ID = 1;

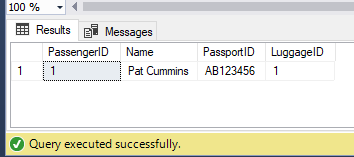
-- Passenger user updating his data:

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:

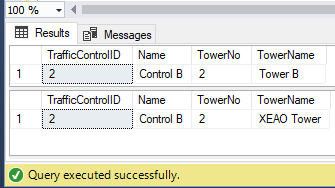
SELECT \* FROM TrafficControl WHERE TrafficControlID = 2;

-- 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:

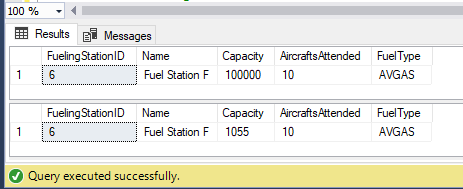
SELECT \* FROM FuelingStation WHERE FuelingStationID = 6;

-- 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;

