ADVANCE DATA BASE ASSESSMENT REPORT

Marvis Osazee Osazuwa

Title: Airport Ticketing System Database Design

1. Introduction

This report presents the design and implementation of a relational database system for managing flight ticketing and e-boarding at an airport. Acting as a database consultant, I was tasked with creating a system that handles reservations, ticket issuance, employees, passengers, flights, baggage, and additional services.

The solution is built using Microsoft SQL Server and developed entirely with T-SQL. It includes normalized tables (up to 3NF), constraints, views, stored procedures, triggers, and user-defined functions. This report outlines the design process, key SQL implementations, and considerations for data integrity, concurrency, security, and backup.

2. Database Design Process

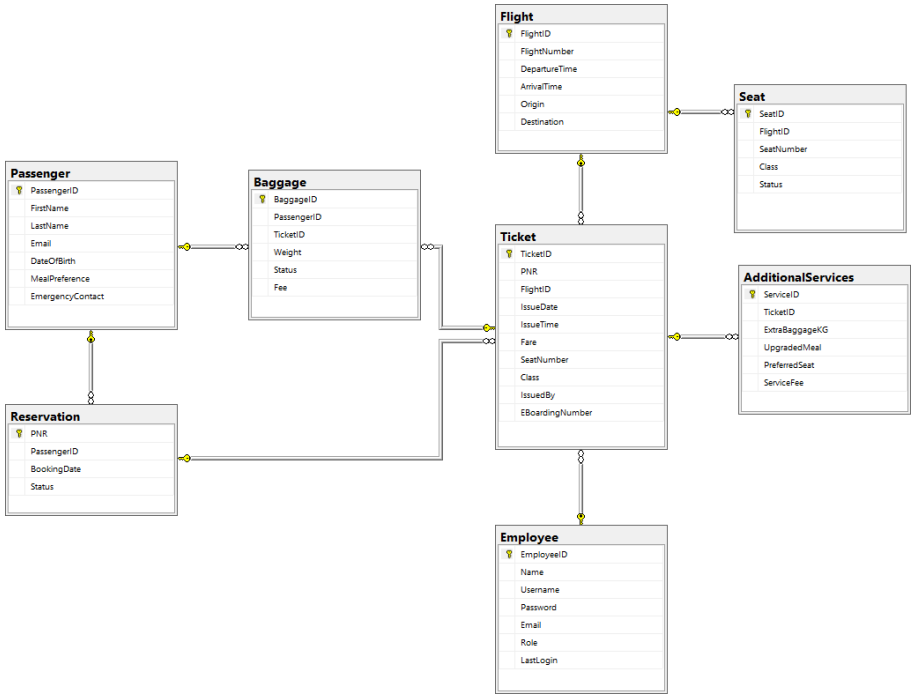
2.1 Database Normalization

I have designed a normalized database for the airport ticketing system based on the client requirements. The design follows 3rd Normal Form (3NF) to eliminate redundancy and ensure data integrity. Here's my approach:

1. **First Normal Form (1NF)**: Ensured each table has a primary key and all attributes contain atomic values
2. **Second Normal Form (2NF)**: Removed partial dependencies by separating entities into different tables
3. **Third Normal Form (3NF)**: Removed transitive dependencies to ensure non-key attributes depend only on the primary key.

2.2 Database Schema

Entity-Relationship Diagram



The ER (Entity-Relationship) diagram visually represents the structure and relationships of data within a database. Foreign keys in the ER diagram enable the joining of tables and retrieval of related data, ensuring consistency and meaningful connections across different entities.

2.3 Tables Structure

The figures below show the structure of tables in the database.

```
-- Create the database
CREATE DATABASE AirWaveExpressTicketingSystem;
GO

USE AirWaveExpressTicketingSystem;
GO

 -- Create the Employee table
CREATE TABLE Employee (
    EmployeeID VARCHAR(10) PRIMARY KEY,
    Name NVARCHAR(100) NOT NULL,
    Username NVARCHAR(50) NOT NULL UNIQUE,
    Password NVARCHAR(100) NOT NULL,
    Email NVARCHAR(100) NOT NULL UNIQUE,
    Role NVARCHAR(20) NOT NULL CHECK (Role IN ('Ticketing Staff', 'Ticketing Supervisor')),
    LastLogin DATETIME NULL
);
GO

 -- Create a new sequence for generating unique EmployeeID
CREATE SEQUENCE EmployeeSeq
    START WITH 200  -- Start from 200 to avoid conflict with existing EmployeeID
    INCREMENT BY 1;
GO
 -- Create a new trigger to generate the EmployeeID using the sequence
CREATE TRIGGER trgGenerateEmployeeID
ON Employee
INSTEAD OF INSERT
AS
BEGIN
    DECLARE @NextID INT;

    -- Get the next value from the sequence
    SELECT @NextID = NEXT VALUE FOR EmployeeSeq;

    -- Insert the new employee with the generated EmployeeID (prefix 'EM' + sequence number)
    INSERT INTO Employee (EmployeeID, Username, Password, Name, Email, Role, LastLogin)
    SELECT 'EM' + RIGHT('000' + CAST(@NextID AS VARCHAR(10)), 3), Username, Password, Name, Email, Role, LastLogin
    FROM INSERTED;
END;
GO
 -- Create Passenger Table
CREATE TABLE Passenger (
    PassengerID INT PRIMARY KEY IDENTITY(2000,1),
    FirstName NVARCHAR(50) NOT NULL,
    LastName NVARCHAR(50) NOT NULL,
    Email NVARCHAR(100) NOT NULL UNIQUE,
    DateOfBirth DATE NOT NULL,
    MealPreference NVARCHAR(20) CHECK (MealPreference IN ('Vegetarian', 'Non-Vegetarian')),
    EmergencyContact NVARCHAR(50) NULL
);
GO
```

```
CREATE TABLE Flight (
    FlightID INT PRIMARY KEY IDENTITY(3000,1),
   FlightNumber VARCHAR(10),
    DepartureTime DATETIME NOT NULL,
    ArrivalTime DATETIME NOT NULL,
    Origin NVARCHAR(50) NOT NULL,
    Destination NVARCHAR(50) NOT NULL,
    CONSTRAINT CHK_ArrivalAfterDeparture CHECK (ArrivalTime > DepartureTime)
);
GO

 -- Create sequence to generate numeric part of FlightNumber
CREATE SEQUENCE FlightSeq
    START WITH 100  -- Starting value for FlightNumber
    INCREMENT BY 1; -- Increment by 1
GO

 -- Create trigger to automatically generate FlightNumber based on FlightSeq
CREATE TRIGGER trgGenerateFlightNumber
ON Flight
INSTEAD OF INSERT
AS
BEGIN
    DECLARE @NextID INT;

    -- Get the next value from the sequence
    SELECT @NextID = NEXT VALUE FOR FlightSeq;

    -- Insert the record with the generated FlightNumber
    INSERT INTO Flight (FlightNumber, DepartureTime, ArrivalTime, Origin, Destination)
    SELECT 'AW' + RIGHT('000' + CAST(@NextID AS VARCHAR(3)), 3), DepartureTime, ArrivalTime, Origin, Destination
    FROM INSERTED;
END;
GO

 -- Create Reservation Table
CREATE TABLE Reservation (
    PNR CHAR(10) PRIMARY KEY,
    PassengerID INT NOT NULL,
    BookingDate DATETIME NOT NULL,
    Status NVARCHAR(20) NOT NULL CHECK (Status IN ('Confirmed', 'Pending', 'Cancelled')),
    FOREIGN KEY (PassengerID) REFERENCES Passenger(PassengerID)
);
GO
```

```sql
-- Create Ticket Table
CREATE TABLE Ticket (
    TicketID INT PRIMARY KEY IDENTITY(4000,1),
    PNR CHAR(10) NOT NULL,
    FlightID INT NOT NULL,
    IssueDate DATE NOT NULL DEFAULT GETDATE(),
    IssueTime TIME NOT NULL DEFAULT CONVERT(TIME, GETDATE()),
    Fare DECIMAL(10,2) NOT NULL,
    SeatNumber NVARCHAR(10) NULL,
    Class NVARCHAR(20) NOT NULL CHECK (Class IN ('Economy', 'Business', 'FirstClass')),
    IssuedBy VARCHAR(10) NOT NULL,
    EBoardingNumber NVARCHAR(20) UNIQUE NOT NULL,
    FOREIGN KEY (PNR) REFERENCES Reservation(PNR),
    FOREIGN KEY (FlightID) REFERENCES Flight(FlightID),
    FOREIGN KEY (IssuedBy) REFERENCES Employee(EmployeeID)
);
GO

-- AdditionalServices Table
CREATE TABLE AdditionalServices (
    ServiceID INT PRIMARY KEY IDENTITY(5000,1),
    TicketID INT NOT NULL,
    ExtraBaggageKG DECIMAL(5,2) NULL DEFAULT 0,
    UpgradedMeal BIT NULL DEFAULT 0,
    PreferredSeat BIT NULL DEFAULT 0,
    ServiceFee DECIMAL(10,2) NOT NULL DEFAULT 0,
    FOREIGN KEY (TicketID) REFERENCES Ticket(TicketID)
);
GO

-- Create Baggage Table
CREATE TABLE Baggage (
    BaggageID VARCHAR(15) PRIMARY KEY,  -- Alphanumeric BaggageID (BG + PassengerID)
    PassengerID INT NOT NULL,           -- Foreign Key referencing PassengerID
    TicketID INT NOT NULL,
    Weight DECIMAL(5,2) NOT NULL,
    Status NVARCHAR(20) NOT NULL CHECK (Status IN ('CheckedIn', 'Loaded')),
    Fee DECIMAL(10,2) NOT NULL,
    FOREIGN KEY (PassengerID) REFERENCES Passenger(PassengerID),  -- Foreign key constraint
    FOREIGN KEY (TicketID) REFERENCES Ticket(TicketID)            -- Assuming Ticket table exists
);
GO

-- Create trigger to generate BaggageID
CREATE TRIGGER trgGenerateBaggageID
ON Baggage
INSTEAD OF INSERT
AS
BEGIN
    DECLARE @PassengerID INT;

    -- Get the PassengerID from the inserted row
    SELECT @PassengerID = PassengerID FROM INSERTED;

    -- Insert the record with the generated BaggageID (BG + PassengerID)
    INSERT INTO Baggage (BaggageID, PassengerID, TicketID, Weight, Status, Fee)
    SELECT 'BG' + CAST(@PassengerID AS VARCHAR(10)), PassengerID, TicketID, Weight, Status, Fee
    FROM INSERTED;
END;
GO
```

2.4 Justification of Design Decisions

1. **Employee Table**
   a. EmployeeID is auto generated with a trigger and sequence, giving IDs like "EM200"
   b. Only two roles are allowed ‘Ticketing Staff’ or ‘Ticketing Supervisor’
   c. Both Username and Email must be unique to avoid duplicates
   d. Passwords are stored as text for now but should be hashed in a real system
   e. LastLogin is optional and tracks the last time the staff logged in
2. **Passenger Table**
   a. PassengerID starts from 2000 and increases automatically
   b. Email must be unique—no two passengers can share the same email

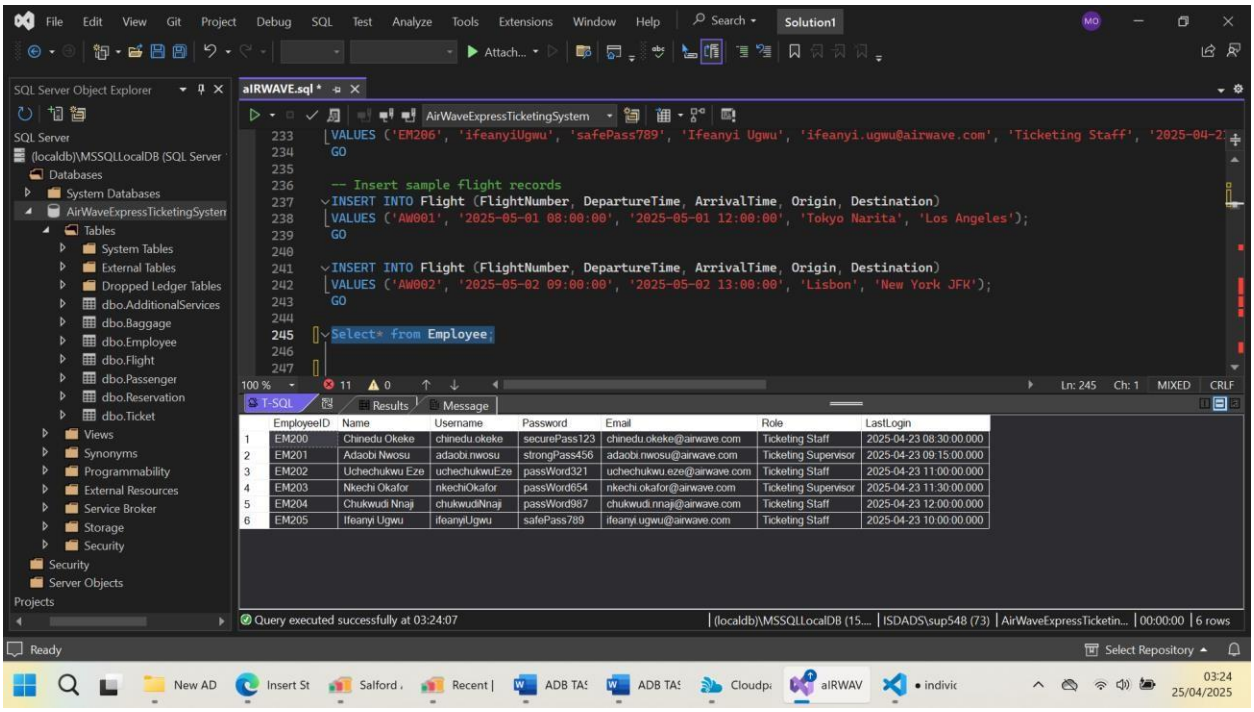
c. Meal preferences are optional and limited to ‘Vegetarian’ or ‘Non-Vegetarian’
d. Emergency contact info is also optional

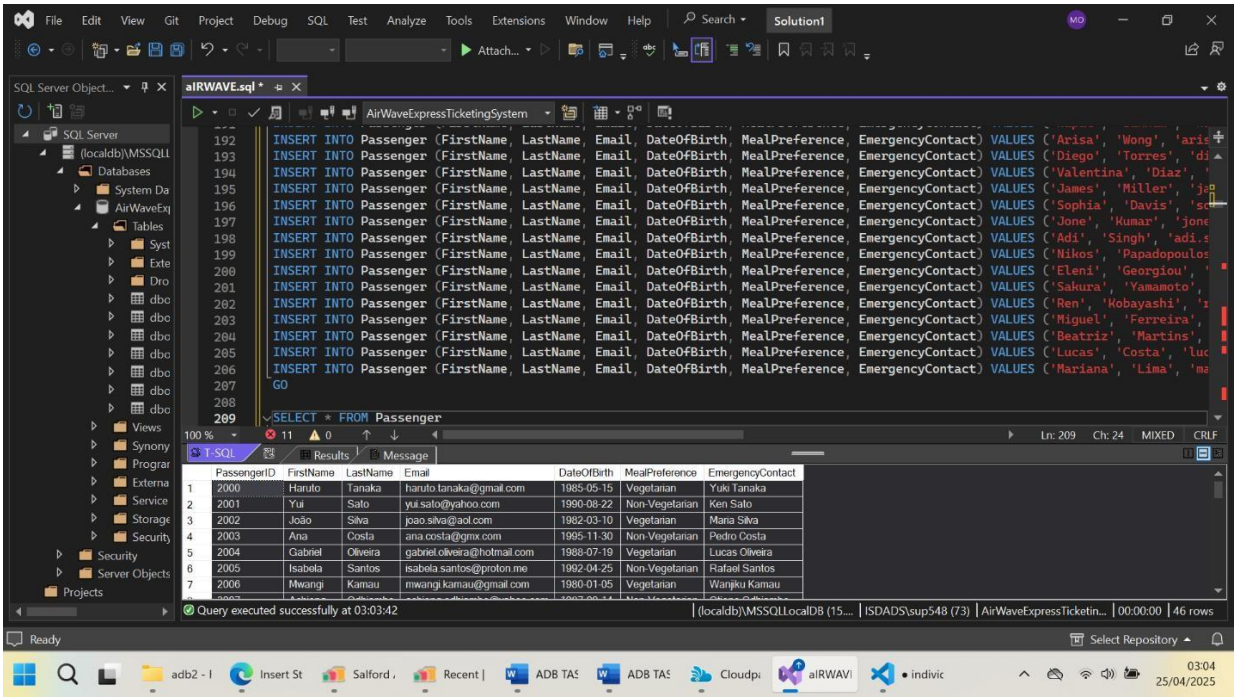
3. **Flight Table**
   a. Each flight gets a unique ID starting from 3000
   b. The system ensures that ArrivalTime is always after DepartureTime
   c. Flight numbers are auto created using a trigger and look like "AW101", "AW102", etc.
4. **Reservation Table**
   a. PNR (Passenger Name Record) is used as the primary key—it’s a unique code
   b. Status is limited to ‘Confirmed’, ‘Pending’, or ‘Cancelled’
   c. Each reservation is linked to a specific passenger
5. **Ticket Table**
   a. TicketID auto-increments from 4000
   b. IssueDate and IssueTime are recorded automatically when the ticket is created
   c. SeatNumber is optional if a seat hasn’t been selected
   d. E-Boarding numbers must be unique for each ticket
   e. Ticket Class is restricted to ‘Economy’, ‘Business’, or ‘FirstClass’
   f. Each ticket is tied to a reservation, flight, and issuing employee
6. **Additional Services Table**
   a. Tracks extra services linked to a ticket
   b. Includes yes/no options like upgraded meals or preferred seats (default is ‘No’)
   c. Service fees are calculated, starting at 0 by default
7. **Baggage Table**
   a. BaggageID is auto generated with a trigger and looks like "BG2000" for PassengerID 2000
   b. Baggage status can only be 'CheckedIn' or 'Loaded'
   c. Weight is stored with decimal accuracy
   d. Each baggage entry is linked to both a passenger and a ticket
8. **Seat Table**
   a. SeatID auto-increments from 1
   b. Each seat is linked to a specific flight via FlightID
   c. Class is limited to ‘Economy’, ‘Business’, or ‘FirstClass’
   d. Seat status can be ‘Available’, ‘Reserved’, or ‘Occupied’—default is ‘Available’
   e. No two seats can have the same number on the same flight (enforced with a unique constraint)

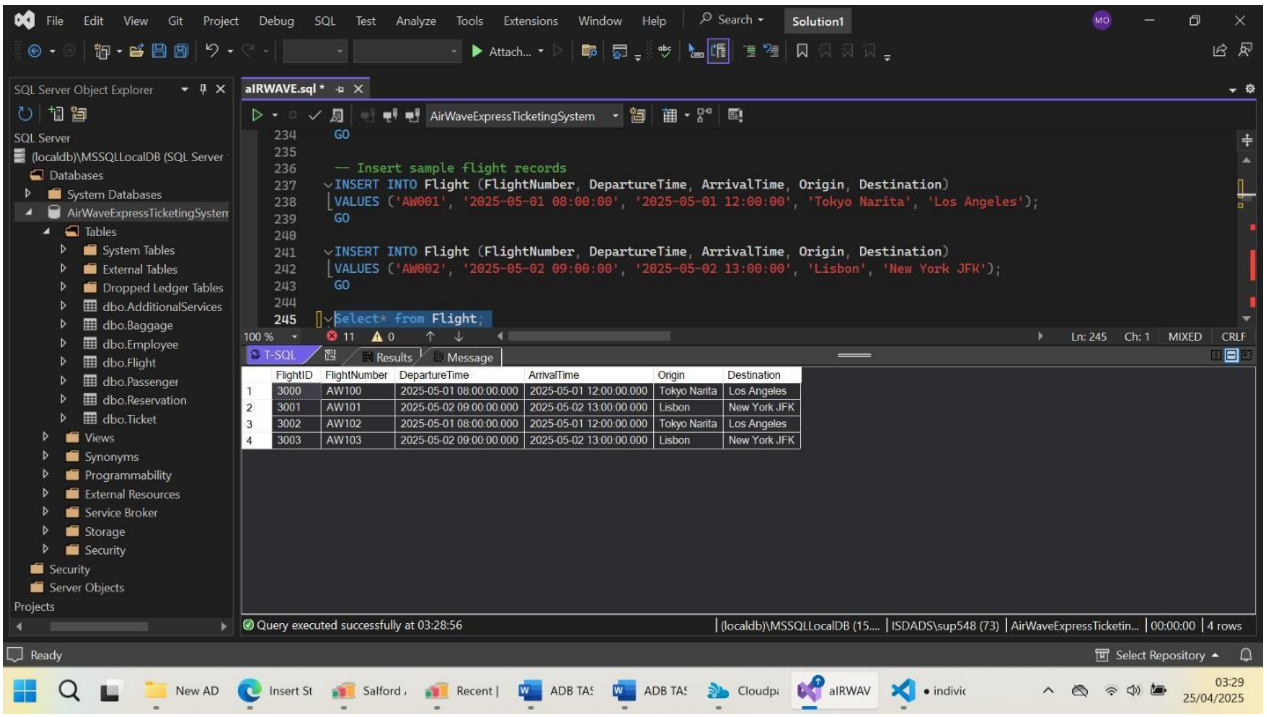
2.5 Data Population

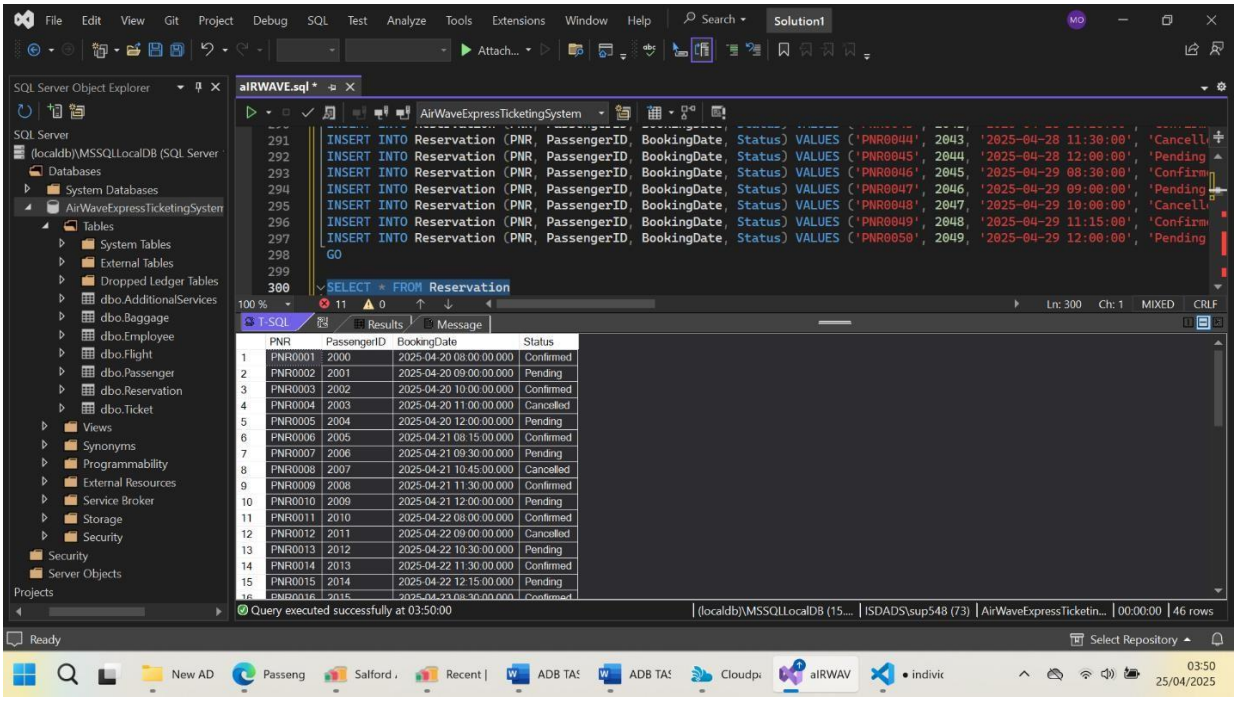
The figures below show the population of the database and display of the respective tables. It was populated with data of 5 employees, 50 passengers and 2 flights that

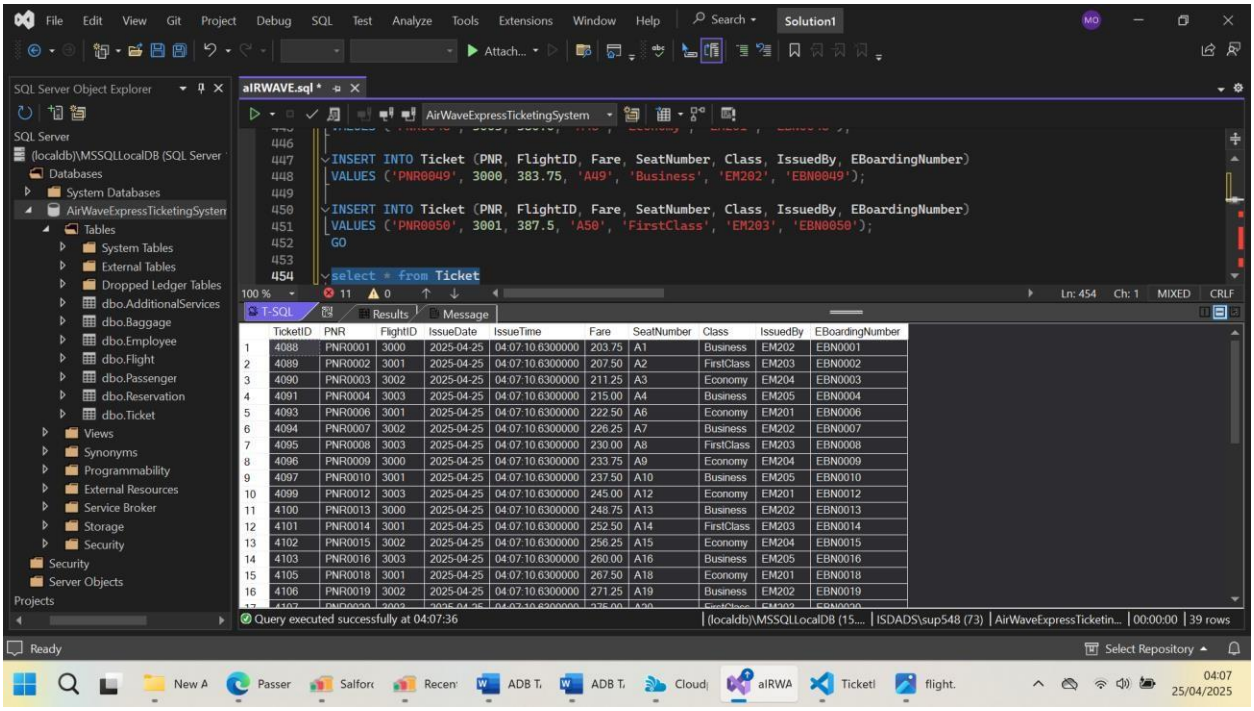
servedas basis for the data in other tables.

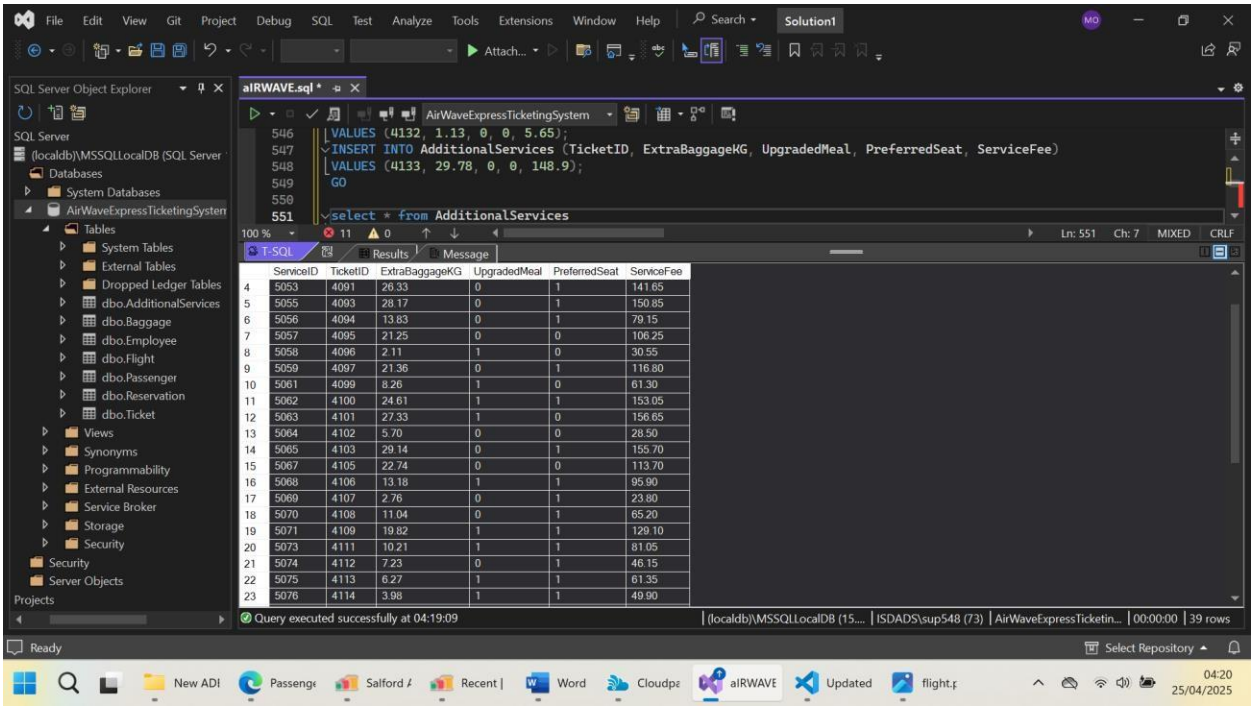


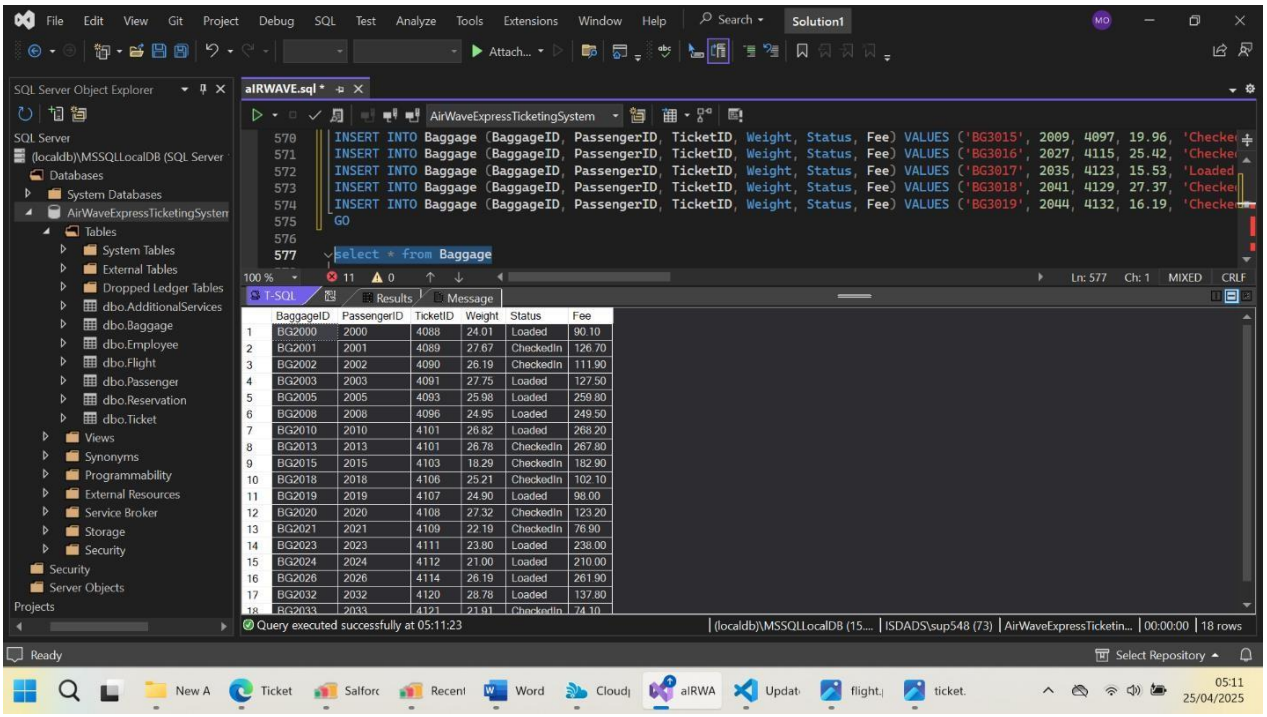



File Edit View Git Project Debug SQL Test Analyze Tools Extensions Window Help
Search
Solution1
Attach...
SQL Server Object Explorer
aIRWAVE.sql *
AirWaveExpressTicketingSystem
234 GO
235
236 -- Insert sample flight records
237 INSERT INTO Flight (FlightNumber, DepartureTime, ArrivalTime, Origin, Destination)
238 VALUES ('AW001', '2025-05-01 08:00:00', '2025-05-01 12:00:00', 'Tokyo Narita', 'Los Angeles');
239 GO
240
241 INSERT INTO Flight (FlightNumber, DepartureTime, ArrivalTime, Origin, Destination)
242 VALUES ('AW002', '2025-05-02 09:00:00', '2025-05-02 13:00:00', 'Lisbon', 'New York JFK');
243 GO
244
245 Select* from Flight;
Ln: 245 Ch: 1 MIXED CRLF
T-SQL
Results
Message
FlightID FlightNumber DepartureTime ArrivalTime Origin Destination
1 3000 AW100 2025-05-01 08:00:00.000 2025-05-01 12:00:00.000 Tokyo Narita Los Angeles
2 3001 AW101 2025-05-02 09:00:00.000 2025-05-02 13:00:00.000 Lisbon New York JFK
3 3002 AW102 2025-05-01 08:00:00.000 2025-05-01 12:00:00.000 Tokyo Narita Los Angeles
4 3003 AW103 2025-05-02 09:00:00.000 2025-05-02 13:00:00.000 Lisbon New York JFK
Query executed successfully at 03:28:56
(localdb)\MSSQLLocalDB (15.... | ISDADS\sup548 (73) | AirWaveExpressTicketin... | 00:00:00 | 4 rows
Ready
Select Repository
03:29
25/04/2025


File Edit View Git Project Debug SQL Test Analyze Tools Extensions Window Help
Search
Solution1
Attach...
SQL Server Object Explorer
aIRWAVE.sql *
AirWaveExpressTicketingSystem
291 INSERT INTO Reservation (PNR, PassengerID, BookingDate, Status) VALUES ('PNR0044', 2043, '2025-04-28 11:30:00', 'Cancell
292 INSERT INTO Reservation (PNR, PassengerID, BookingDate, Status) VALUES ('PNR0045', 2044, '2025-04-28 12:00:00', 'Pending
293 INSERT INTO Reservation (PNR, PassengerID, BookingDate, Status) VALUES ('PNR0046', 2045, '2025-04-29 08:30:00', 'Confirm
294 INSERT INTO Reservation (PNR, PassengerID, BookingDate, Status) VALUES ('PNR0047', 2046, '2025-04-29 09:00:00', 'Pending
295 INSERT INTO Reservation (PNR, PassengerID, BookingDate, Status) VALUES ('PNR0048', 2047, '2025-04-29 10:00:00', 'Cancell
296 INSERT INTO Reservation (PNR, PassengerID, BookingDate, Status) VALUES ('PNR0049', 2048, '2025-04-29 11:15:00', 'Confirm
297 INSERT INTO Reservation (PNR, PassengerID, BookingDate, Status) VALUES ('PNR0050', 2049, '2025-04-29 12:00:00', 'Pending
298 GO
299
300 SELECT * FROM Reservation
Ln: 300 Ch: 1 MIXED CRLF
T-SQL
Results
Message
PNR PassengerID BookingDate Status
1 PNR0001 2000 2025-04-20 08:00:00.000 Confirmed
2 PNR0002 2001 2025-04-20 09:00:00.000 Pending
3 PNR0003 2002 2025-04-20 10:00:00.000 Confirmed
4 PNR0004 2003 2025-04-20 11:00:00.000 Cancelled
5 PNR0005 2004 2025-04-20 12:00:00.000 Pending
6 PNR0006 2005 2025-04-21 08:15:00.000 Confirmed
7 PNR0007 2006 2025-04-21 09:30:00.000 Pending
8 PNR0008 2007 2025-04-21 10:45:00.000 Cancelled
9 PNR0009 2008 2025-04-21 11:30:00.000 Confirmed
10 PNR0010 2009 2025-04-21 12:00:00.000 Pending
11 PNR0011 2010 2025-04-22 08:00:00.000 Confirmed
12 PNR0012 2011 2025-04-22 09:00:00.000 Cancelled
13 PNR0013 2012 2025-04-22 10:30:00.000 Pending
14 PNR0014 2013 2025-04-22 11:30:00.000 Confirmed
15 PNR0015 2014 2025-04-22 12:15:00.000 Pending
Query executed successfully at 03:50:00
(localdb)\MSSQLLocalDB (15.... | ISDADS\sup548 (73) | AirWaveExpressTicketin... | 00:00:00 | 46 rows
Ready
Select Repository
03:50
25/04/2025

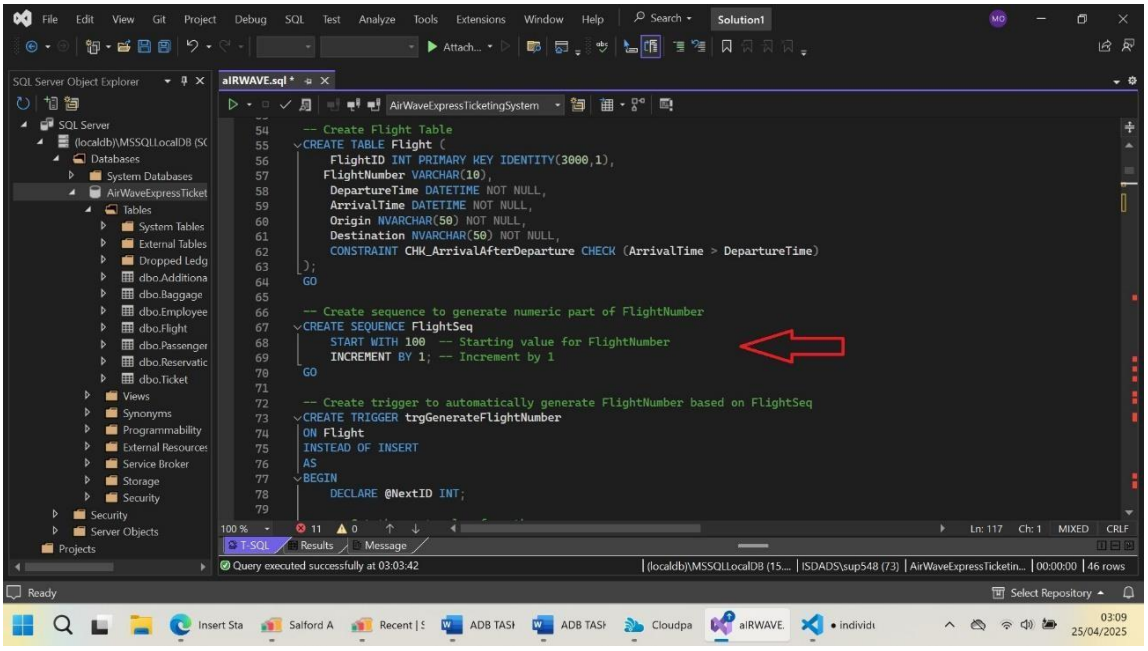

File Edit View Git Project Debug SQL Test Analyze Tools Extensions Window Help Search Solution1
SQL Server Object Explorer
aIRWAVE.sql *
AirWaveExpressTicketingSystem
INSERT INTO Ticket (PNR, FlightID, Fare, SeatNumber, Class, IssuedBy, EBoardingNumber)
VALUES ('PNR0049', 3000, 383.75, 'A49', 'Business', 'EM202', 'EBN0049');
INSERT INTO Ticket (PNR, FlightID, Fare, SeatNumber, Class, IssuedBy, EBoardingNumber)
VALUES ('PNR0050', 3001, 387.5, 'A50', 'FirstClass', 'EM203', 'EBN0050');
GO
select * from Ticket
T-SQL Results Message
TicketID PNR FlightID IssueDate IssueTime Fare SeatNumber Class IssuedBy EBoardingNumber
4088 PNR0001 3000 2025-04-25 04:07:10.6300000 203.75 A1 Business EM202 EBN0001
4089 PNR0002 3001 2025-04-25 04:07:10.6300000 207.50 A2 FirstClass EM203 EBN0002
4090 PNR0003 3002 2025-04-25 04:07:10.6300000 211.25 A3 Economy EM204 EBN0003
4091 PNR0004 3003 2025-04-25 04:07:10.6300000 215.00 A4 Business EM205 EBN0004
4093 PNR0006 3001 2025-04-25 04:07:10.6300000 222.50 A6 Economy EM201 EBN0006
4094 PNR0007 3002 2025-04-25 04:07:10.6300000 226.25 A7 Business EM202 EBN0007
4095 PNR0008 3003 2025-04-25 04:07:10.6300000 230.00 A8 FirstClass EM203 EBN0008
4096 PNR0009 3000 2025-04-25 04:07:10.6300000 233.75 A9 Economy EM204 EBN0009
4097 PNR0010 3001 2025-04-25 04:07:10.6300000 237.50 A10 Business EM205 EBN0010
4099 PNR0012 3003 2025-04-25 04:07:10.6300000 245.00 A12 Economy EM201 EBN0012
4100 PNR0013 3000 2025-04-25 04:07:10.6300000 248.75 A13 Business EM202 EBN0013
4101 PNR0014 3001 2025-04-25 04:07:10.6300000 252.50 A14 FirstClass EM203 EBN0014
4102 PNR0015 3002 2025-04-25 04:07:10.6300000 256.25 A15 Economy EM204 EBN0015
4103 PNR0016 3003 2025-04-25 04:07:10.6300000 260.00 A16 Business EM205 EBN0016
4105 PNR0018 3001 2025-04-25 04:07:10.6300000 267.50 A18 Economy EM201 EBN0018
4106 PNR0019 3002 2025-04-25 04:07:10.6300000 271.25 A19 Business EM202 EBN0019
Query executed successfully at 04:07:36
(localdb)\MSSQLLocalDB (15.... ISDADS\sup548 (73) AirWaveExpressTicketin... 00:00:00 39 rows
Ready
Select Repository
04:07
25/04/2025


File Edit View Git Project Debug SQL Test Analyze Tools Extensions Window Help Search Solution1
SQL Server Object Explorer
aIRWAVE.sql *
AirWaveExpressTicketingSystem
VALUES (4132, 1.13, 0, 0, 5.65);
INSERT INTO AdditionalServices (TicketID, ExtraBaggageKG, UpgradedMeal, PreferredSeat, ServiceFee)
VALUES (4133, 29.78, 0, 0, 148.9);
GO
select * from AdditionalServices
T-SQL Results Message
ServiceID TicketID ExtraBaggageKG UpgradedMeal PreferredSeat ServiceFee
5053 4091 26.33 0 1 141.65
5055 4093 28.17 0 1 150.85
5056 4094 13.83 0 1 79.15
5057 4095 21.25 0 0 106.25
5058 4096 2.11 1 0 30.55
5059 4097 21.36 0 1 116.80
5061 4099 8.26 1 0 61.30
5062 4100 24.61 1 1 153.05
5063 4101 27.33 1 0 156.65
5064 4102 5.70 0 0 28.50
5065 4103 29.14 0 1 155.70
5067 4105 22.74 0 0 113.70
5068 4106 13.18 1 1 95.90
5069 4107 2.76 0 1 23.80
5070 4108 11.04 0 1 65.20
5071 4109 19.82 1 1 129.10
5073 4111 10.21 1 1 81.05
5074 4112 7.23 0 1 46.15
5075 4113 6.27 1 1 61.35
5076 4114 3.98 1 1 49.90
Query executed successfully at 04:19:09
(localdb)\MSSQLLocalDB (15.... ISDADS\sup548 (73) AirWaveExpressTicketin... 00:00:00 39 rows
Ready
Select Repository
04:20
25/04/2025



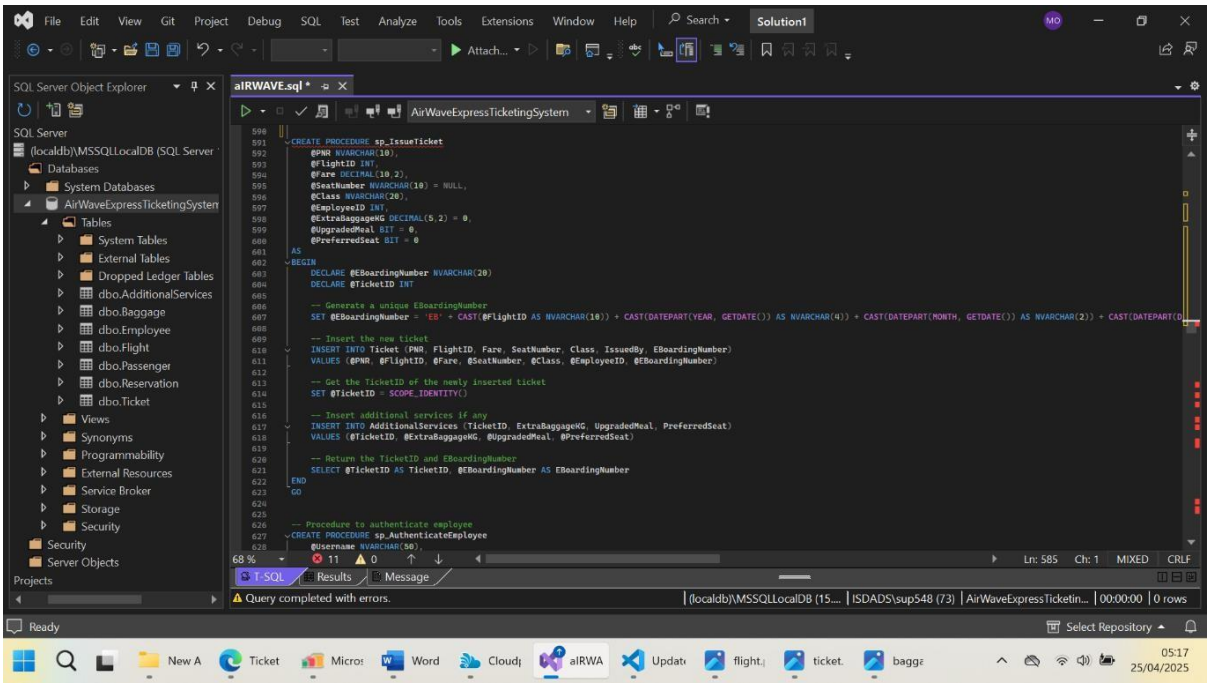
2.6 Database Objects

- *Sequence*



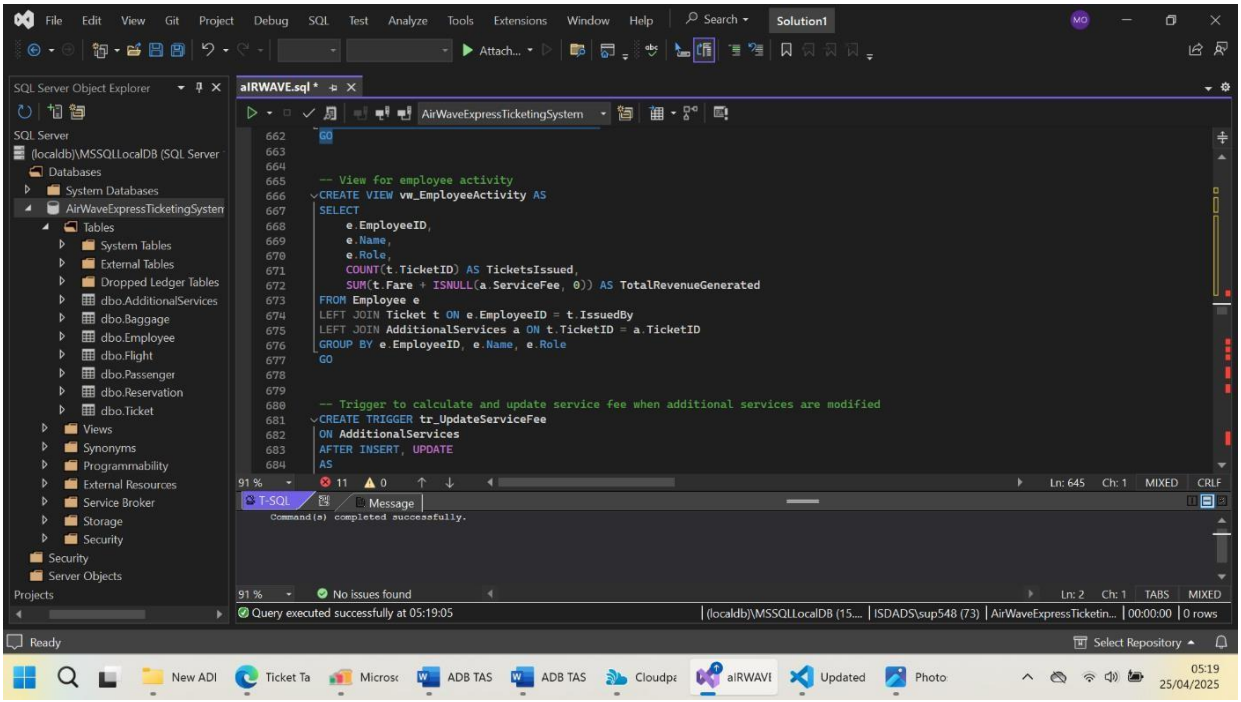
This sequence generates incremental numbers used in a trigger to automatically create formatted FlightNumber values like AW100, AW101.

- *Stored Procedures*



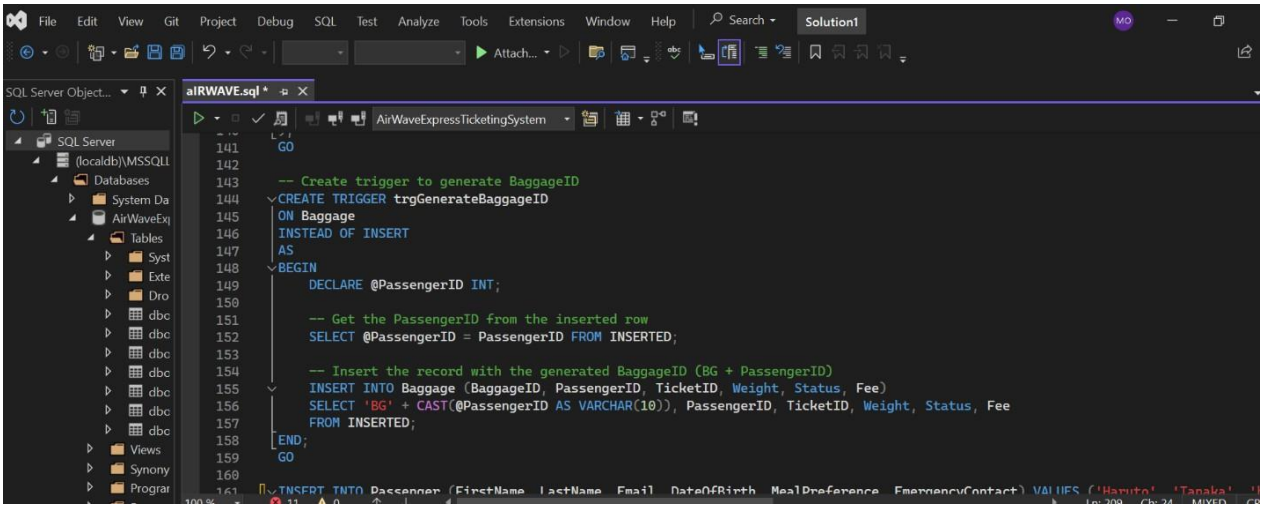
This is the procedure to issue a new ticket.

- *Views*



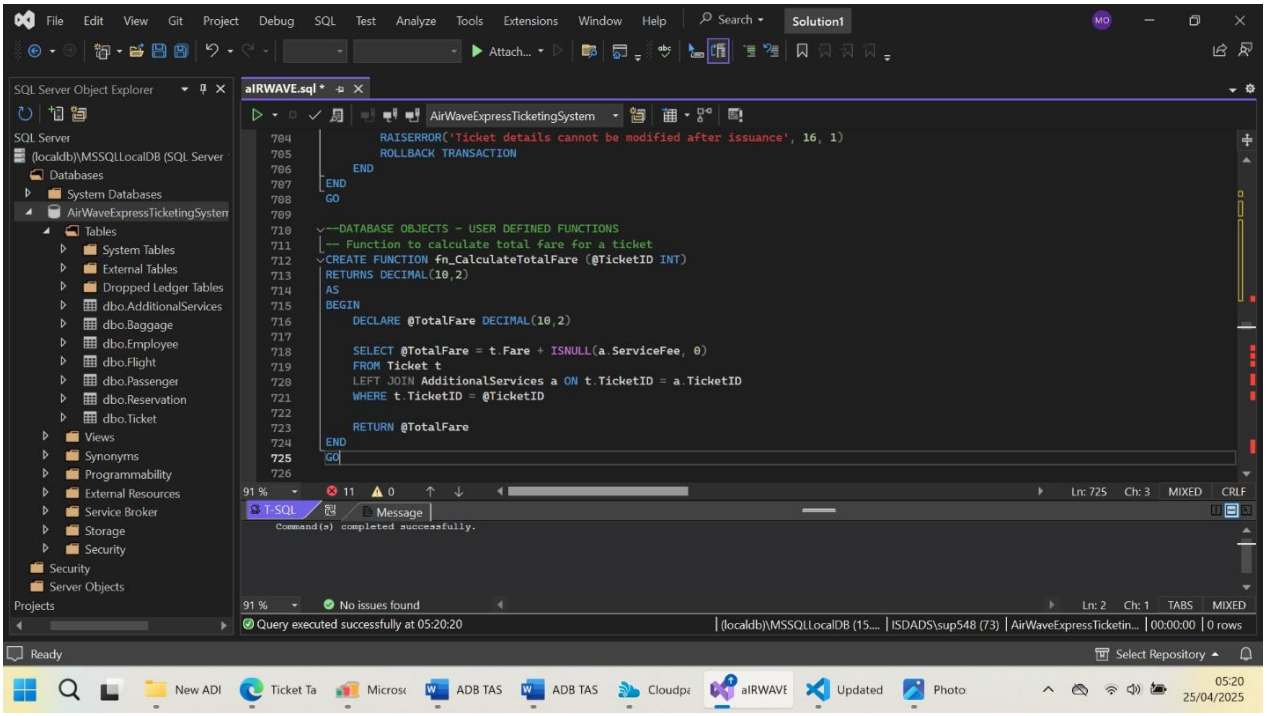
View on employee activity.

- *Triggers*



Trigger to generate Baggage ID by adding 'BG' to the passengerID.

- *User-Defined Functions*



This function is to calculate total fare for a ticket

2.7 Data Integrity and Security Considerations

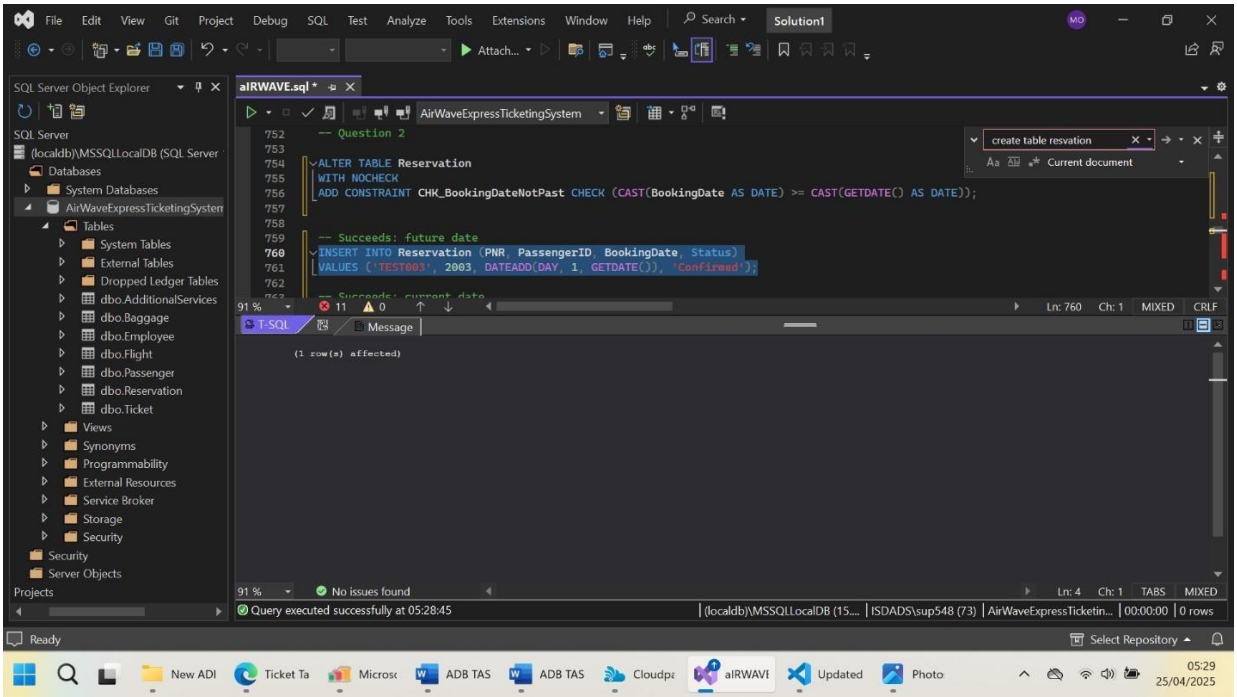
1. **Constraints:**
   1.1. CHECK constraints ensure valid values for roles, statuses, and meal preferences
   1.2. UNIQUE constraints prevent duplicate usernames, emails, and boarding numbers
   1.3. FOREIGN KEY constraints maintain referential integrity
2. **Security:**
   2.1. Authentication stored procedure handles employee login
   2.2. In production, passwords would be hashed and salted
   2.3. Role-based access would be implemented at application level
3. **Transaction Management:**
   3.1. Critical operations like ticket issuance are wrapped in transactions
   3.2. Triggers help maintain data consistency

3 Implementation of T-SQL Requirements

3.1 Q2. Adding Constraint for Reservation Date Validation

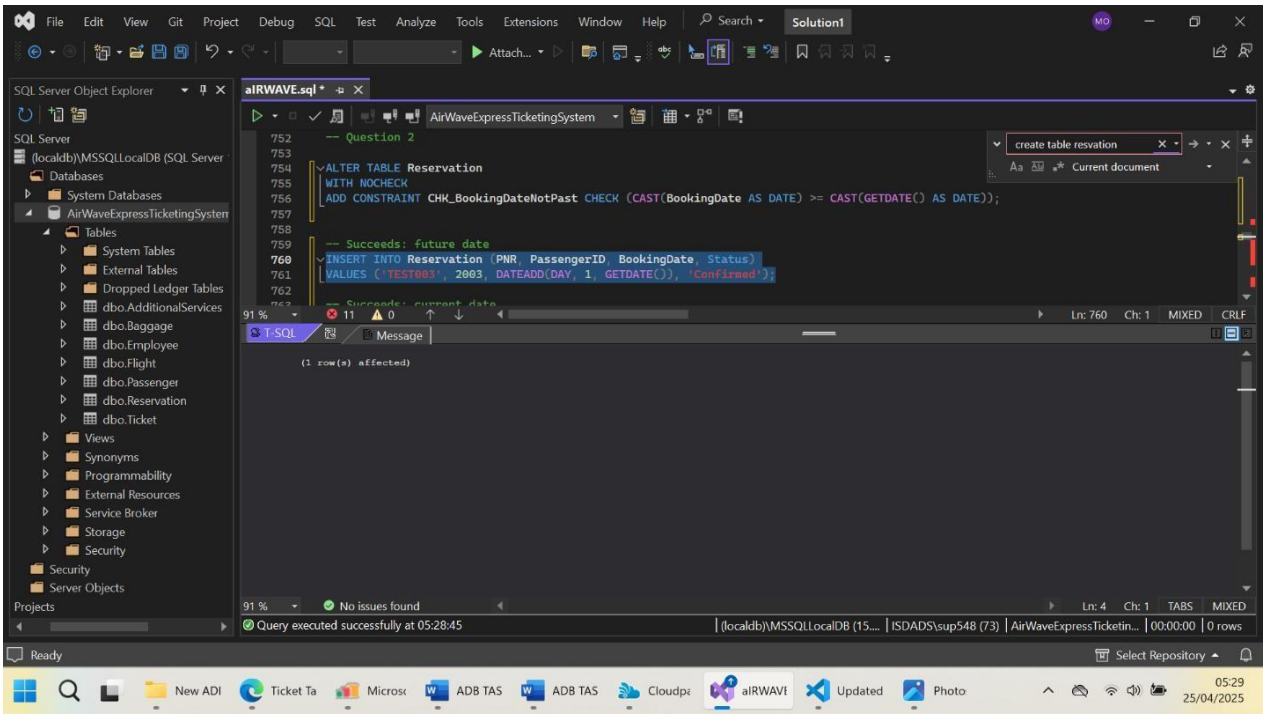
I added a constraint to ensure that reservation dates cannot be in the past. This is an important business rule to maintain data integrity in the ticketing system.

- *Implementation*

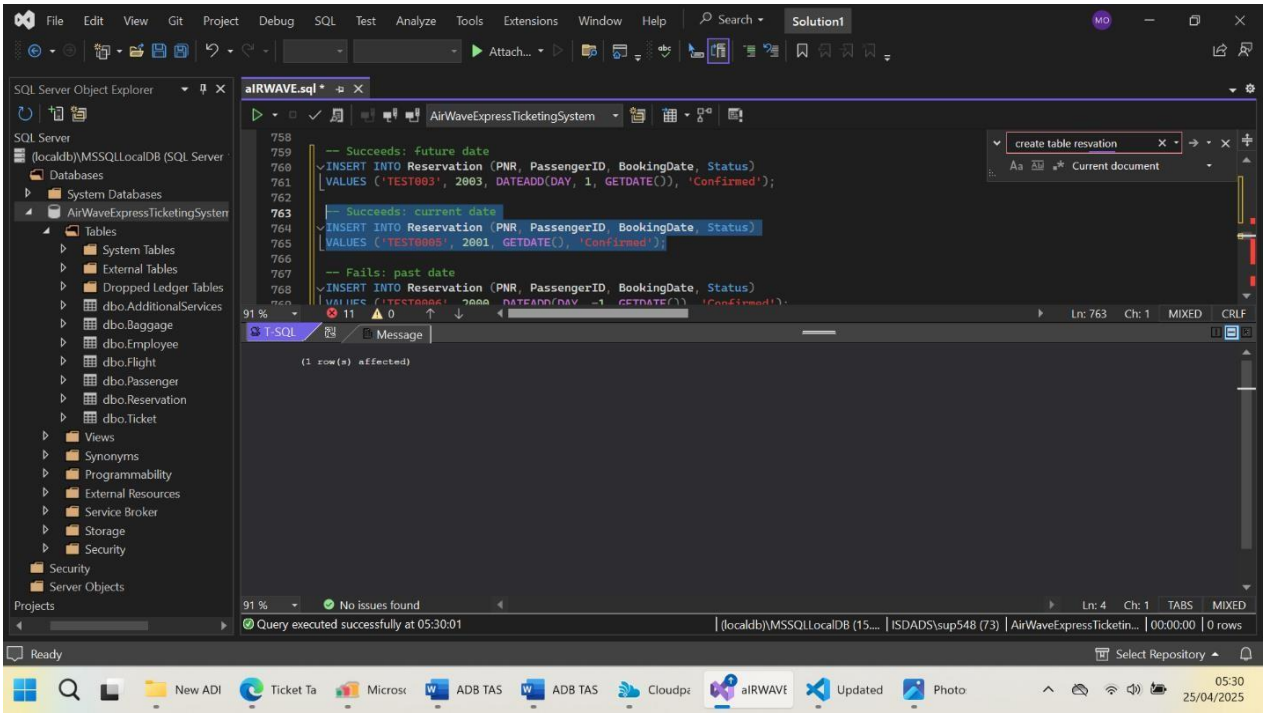


- *Testing the Constraint*

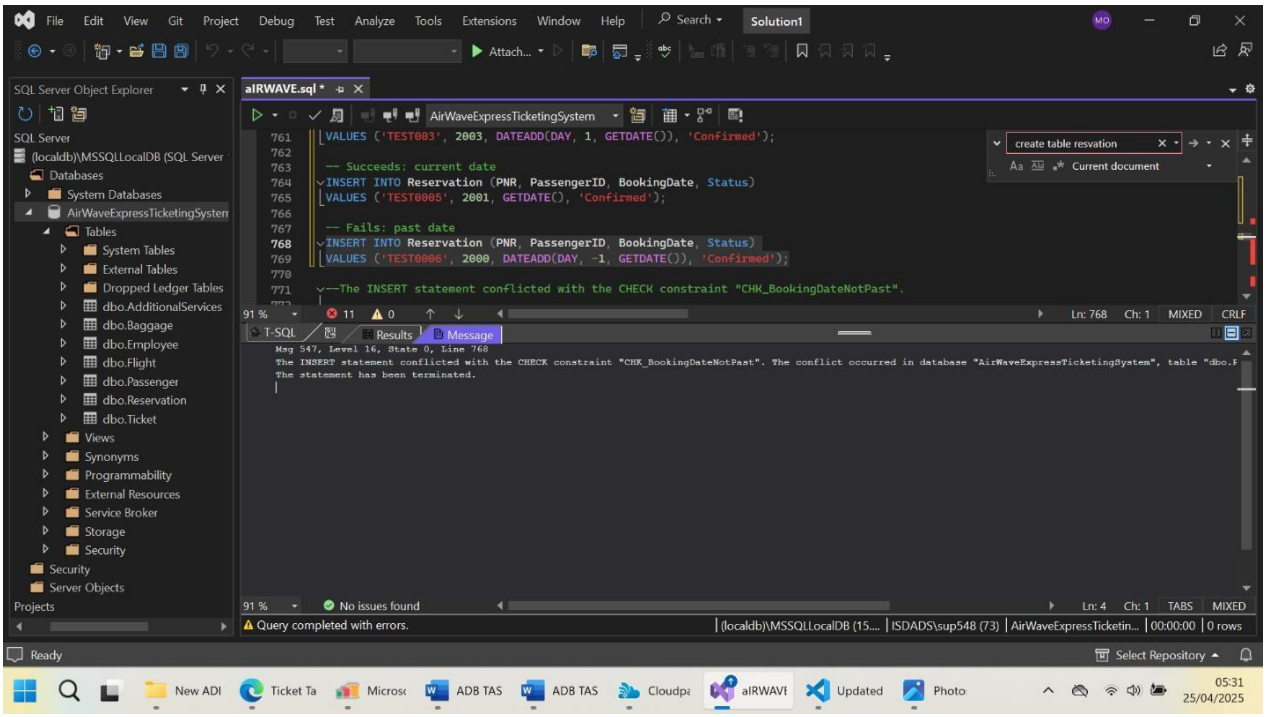
Let's verify the constraint works properly:



This is successful (future date)



This is successful (current date)



-- This fails (past date)

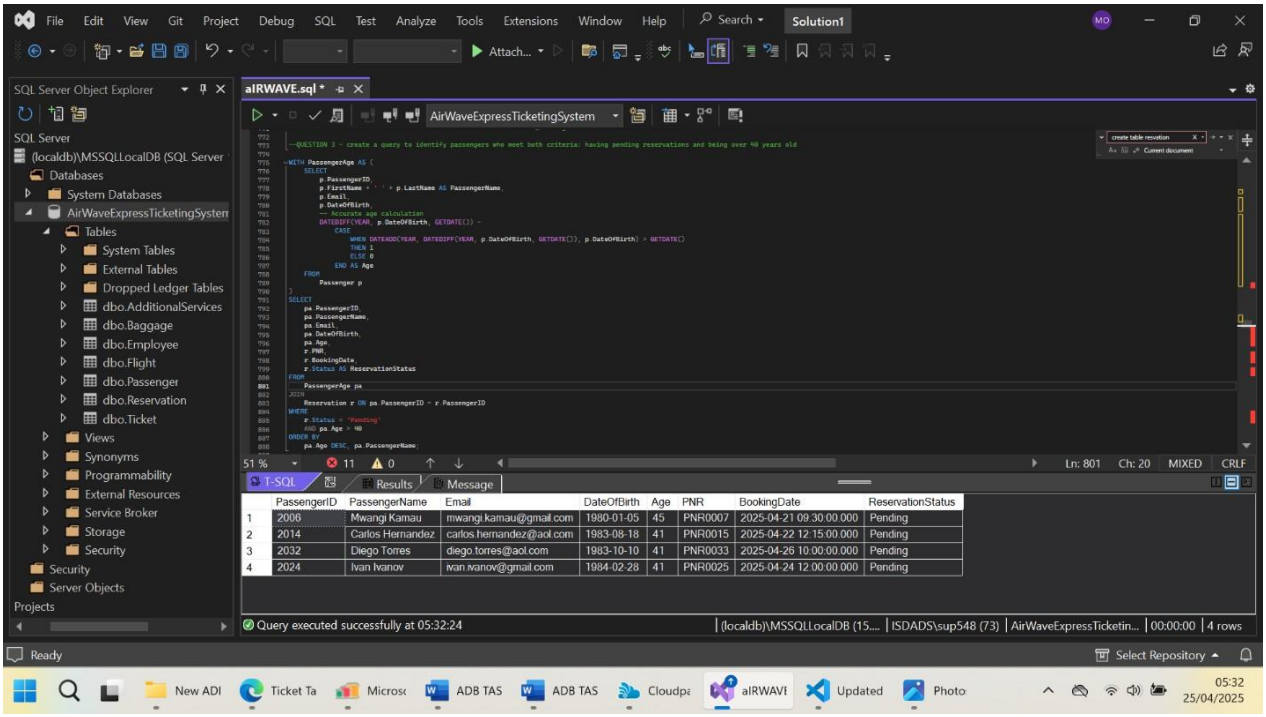
The third insert should fail with an error like:

The INSERT statement conflicted with the CHECK constraint "CHK_BookingDateNotPast".

3.2 Q3. Identifying Passengers with Pending Reservations and Age > 40 Years

I created a query to identify passengers who meet both criteria: having pending reservations and being over 40 years old.

- *Implementation*

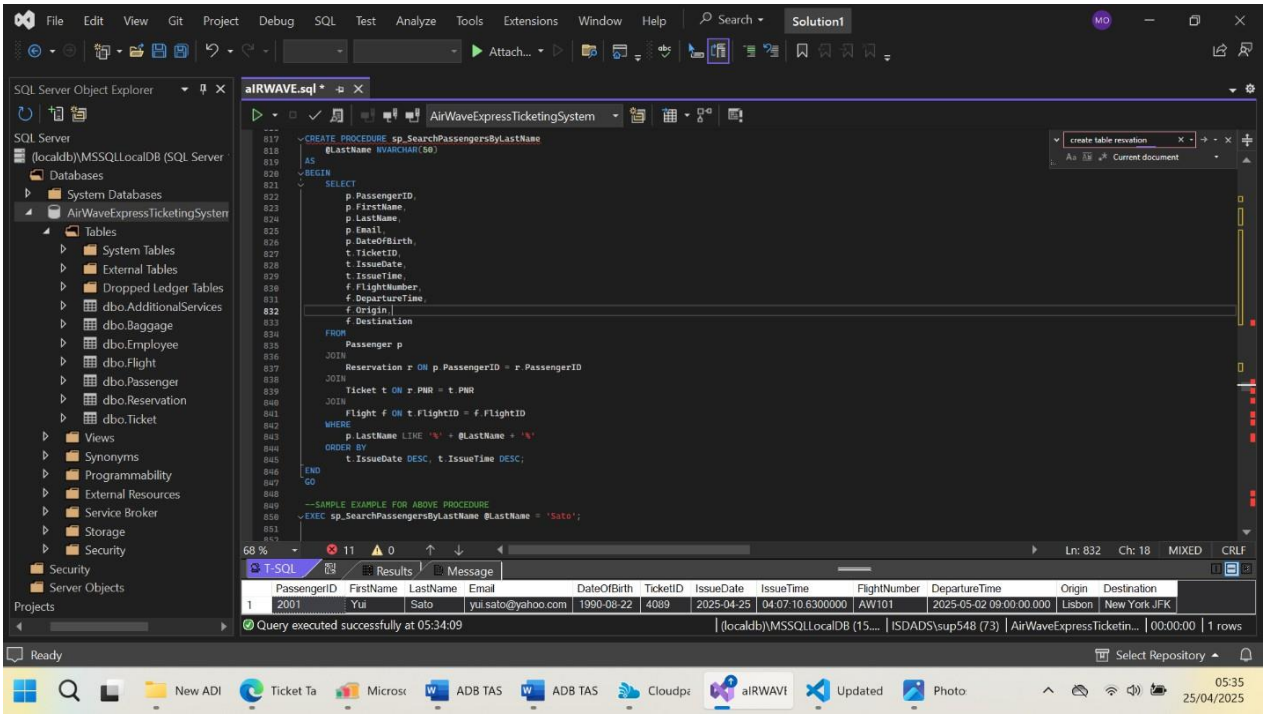


The result table from the query shows the passengers' name Mwangi Kamau, Carlos Hernandez, Diego Torres and Ivan Ivanov. Their birth years are 1980/01/05, 1983/08/18,1983/10/10, 1984/02/28 respectively.

3.3 Q4. Stored Procedures and Functions for Ticketing System

Here are the implementations for the required database operations:

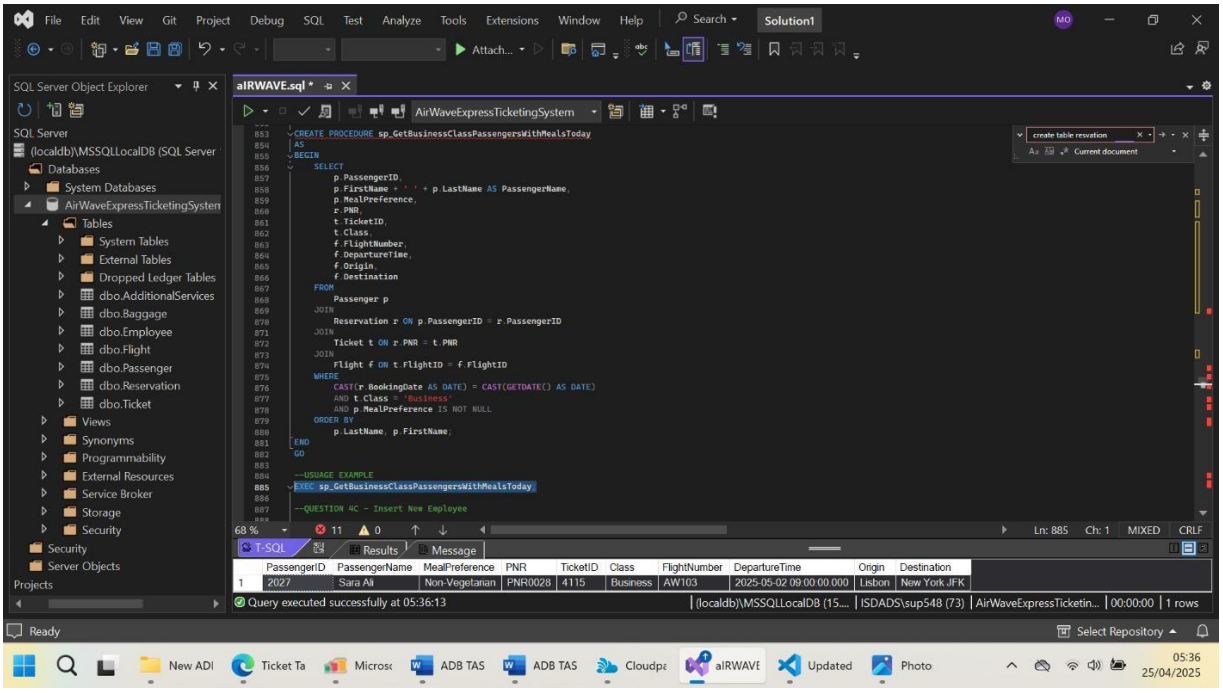
- *Search Passengers by Last Name*



Usage Example:

```
EXEC sp_SearchPassengersByLastName @LastName = 'Sato';
```

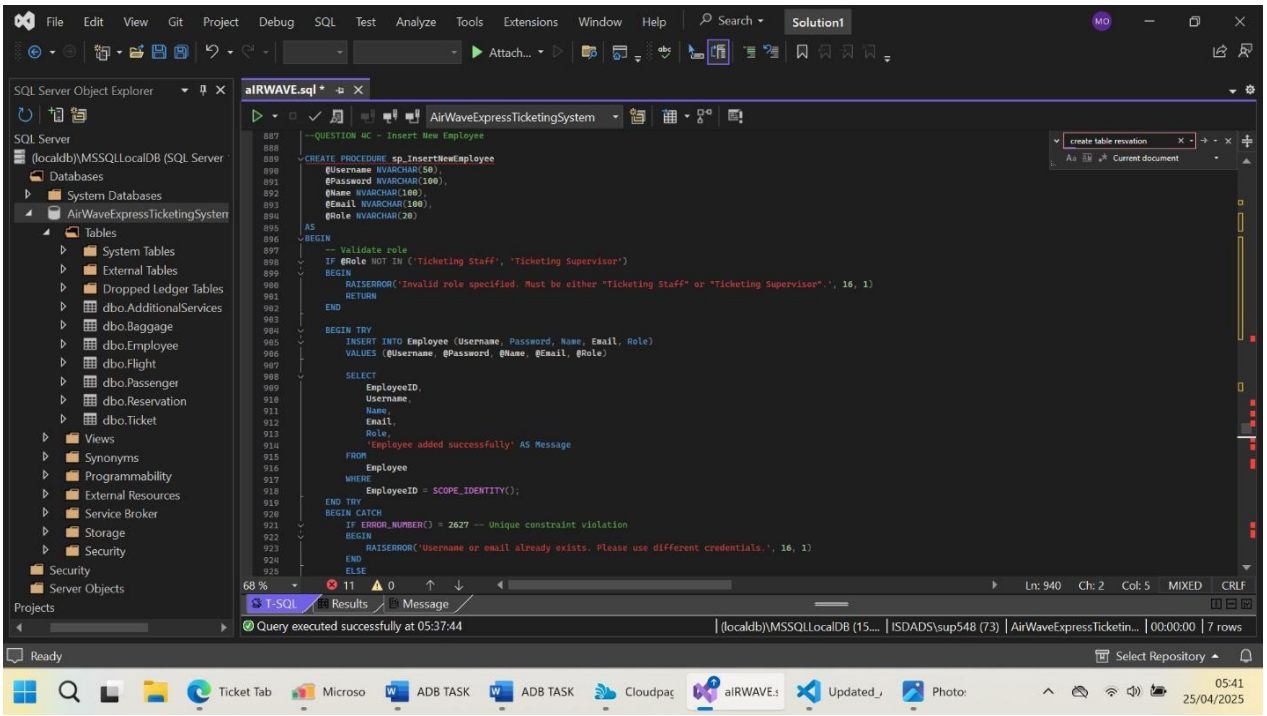
- *List Business Class Passengers with Meal Requirements (Today's Reservations)*

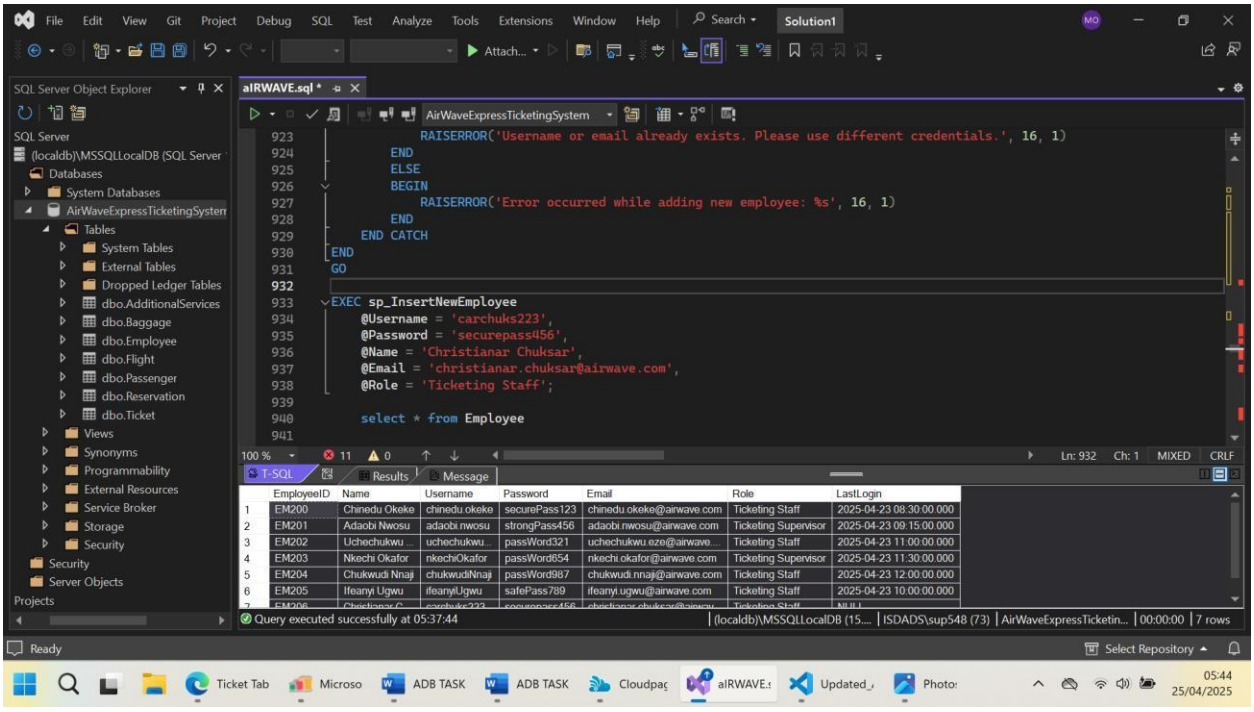


Usage Example:

```
EXEC sp_GetBusinessClassPassengersWithMealsToday;
```

- *Insert New Employee*

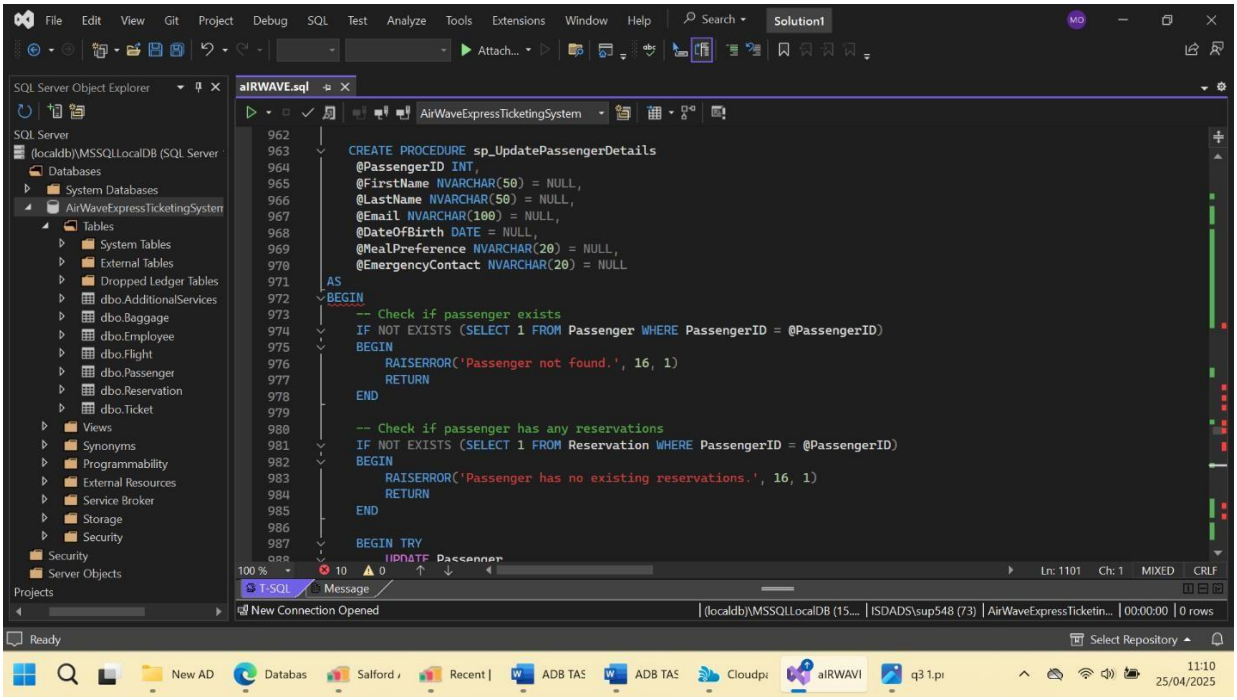


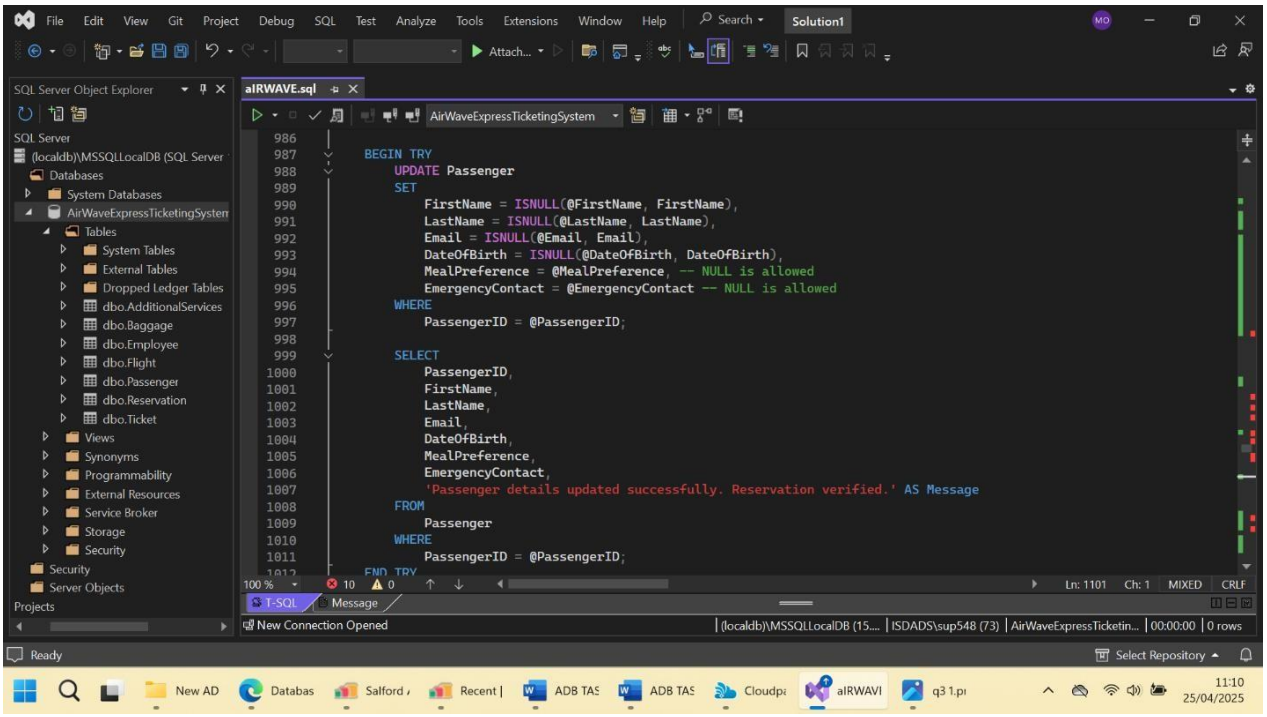


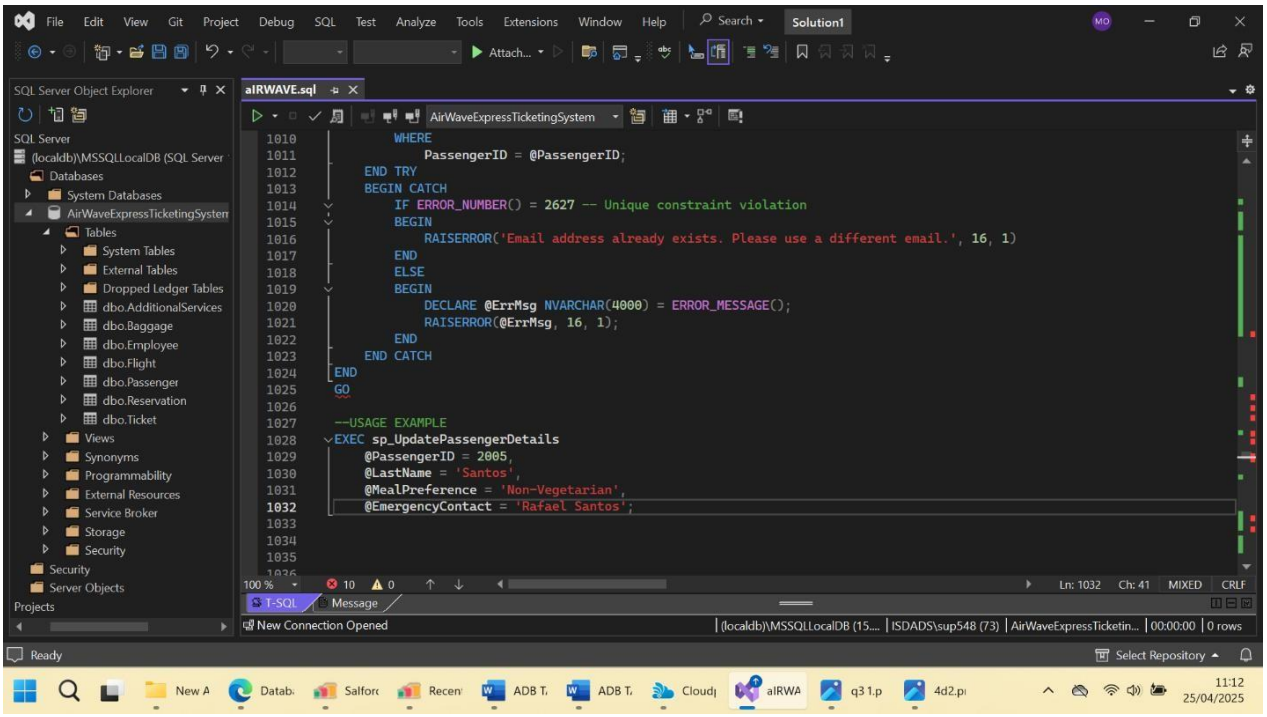
Usage Example:

```
EXEC sp_InsertNewEmployee
    @Username = 'carchuks223',
    @Password = 'securepass456',
    @Name = 'Christianar Chuksar',
    @Email = 'christianar.chuksar@airwave.com',
    @Role = 'Ticketing Staff';
```

- *d) Update Passenger Details*



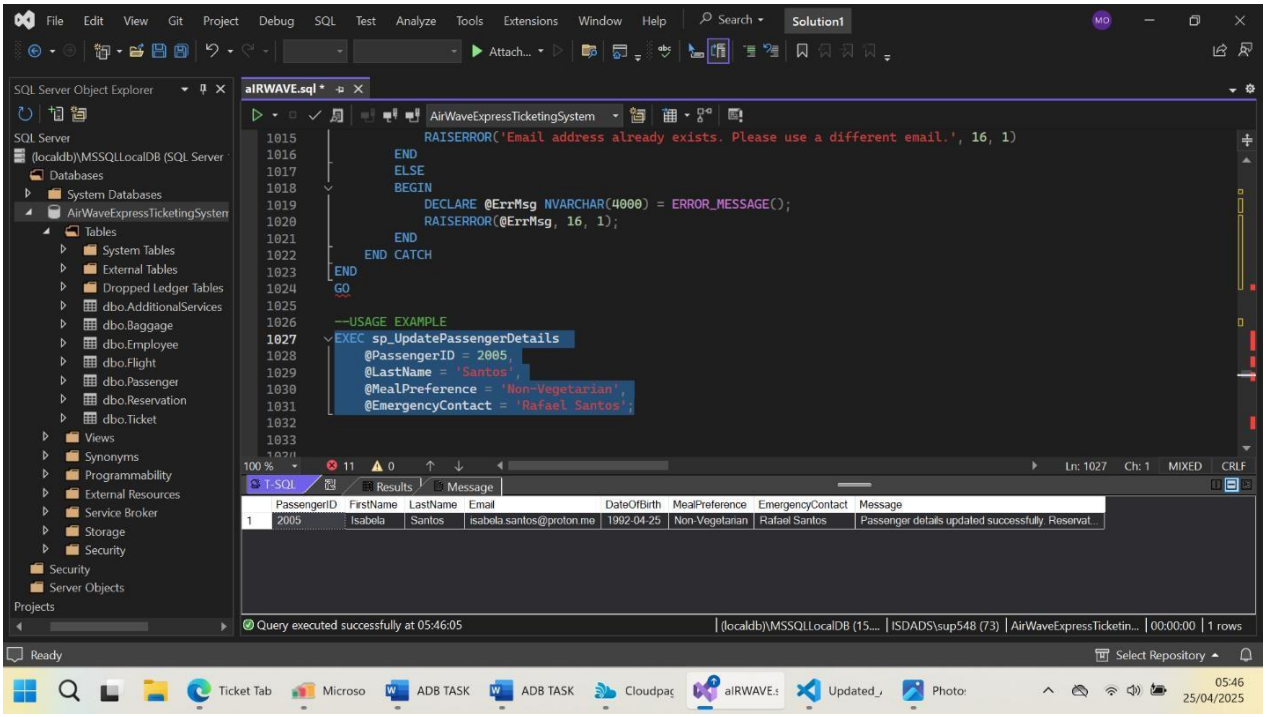




USAGE EXAMPLE

```
EXEC sp_UpdatePassengerDetails
  @PassengerID = 2005,
  @LastName = 'Santos',
  @MealPreference = 'Non-Vegetarian',
```

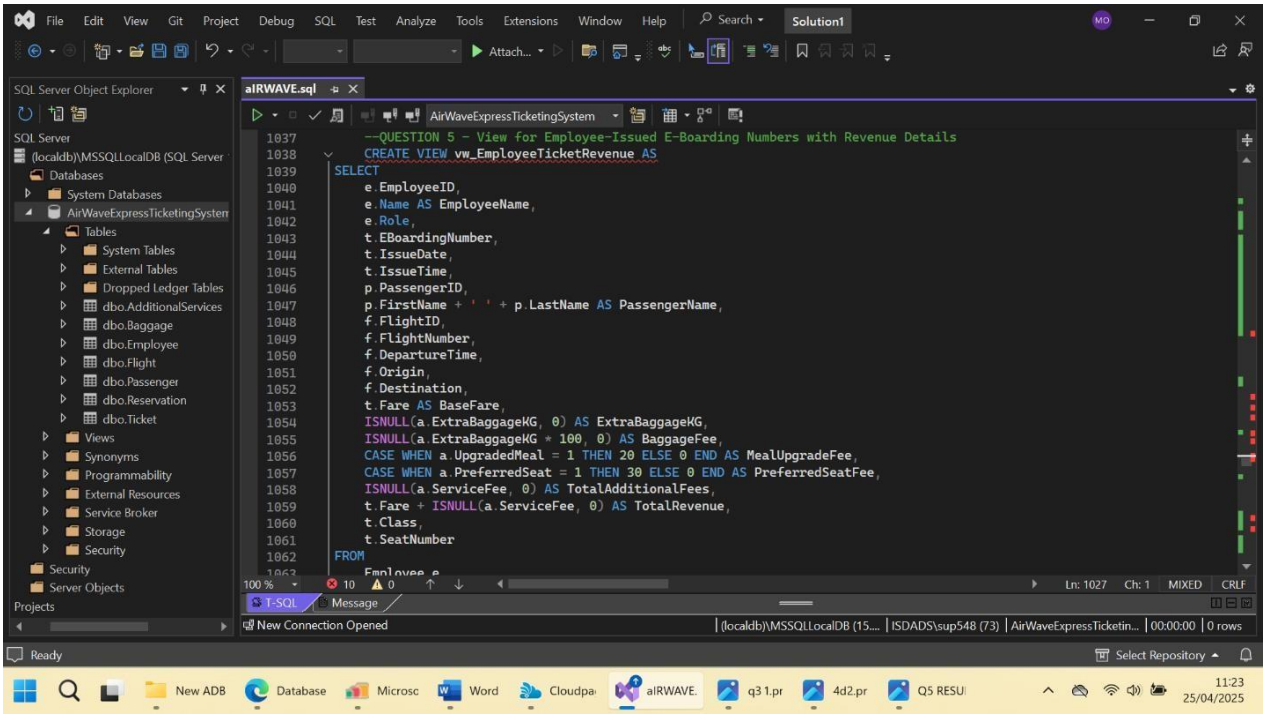
@EmergencyContact = 'Rafael Santos';

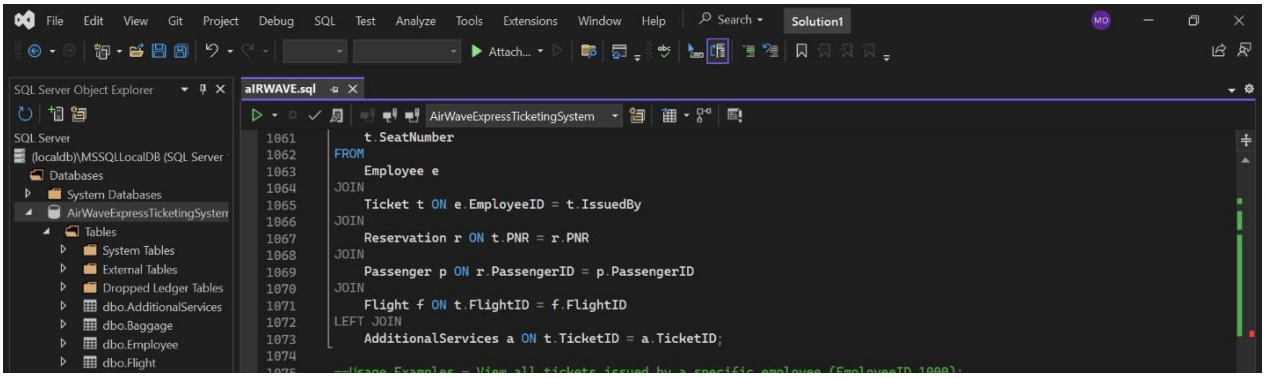


3.4 Q5. View for Employee-Issued E-Boarding Numbers with Revenue Details

I created a comprehensive view that shows all e-boarding numbers issued by a specific employee, along with the revenue details for each ticket, including base fare and additional service fees.

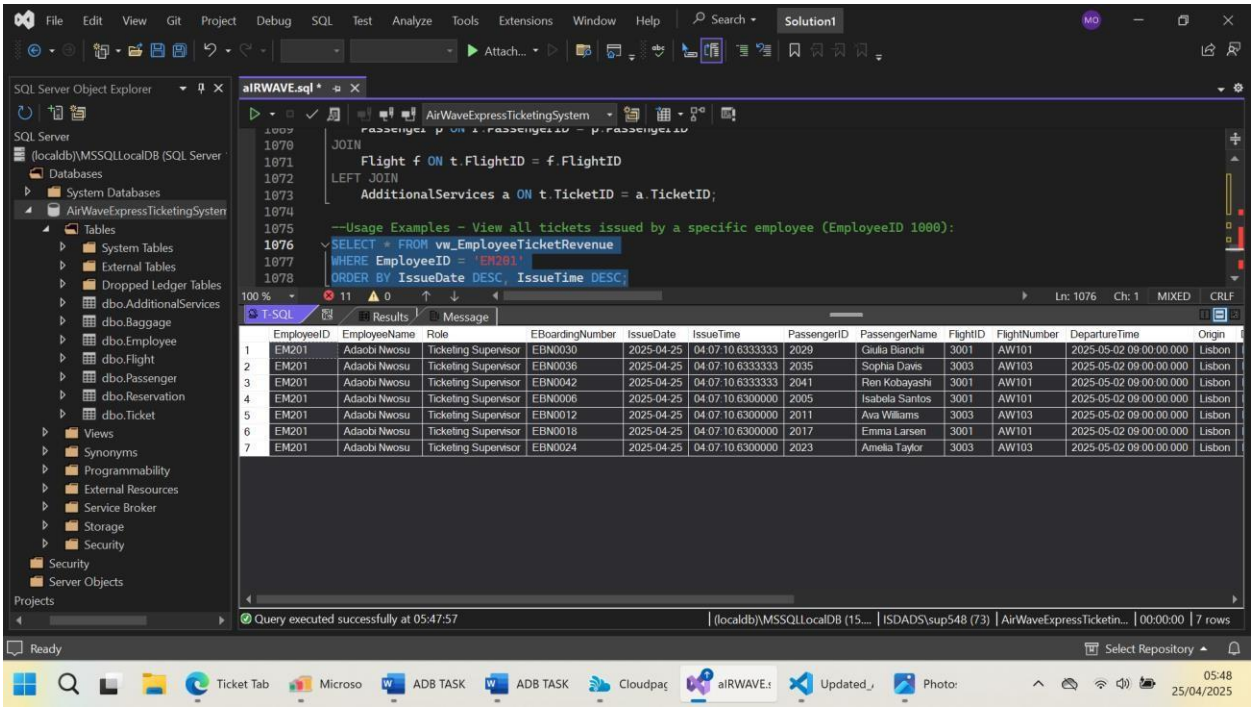
- *Implementation*



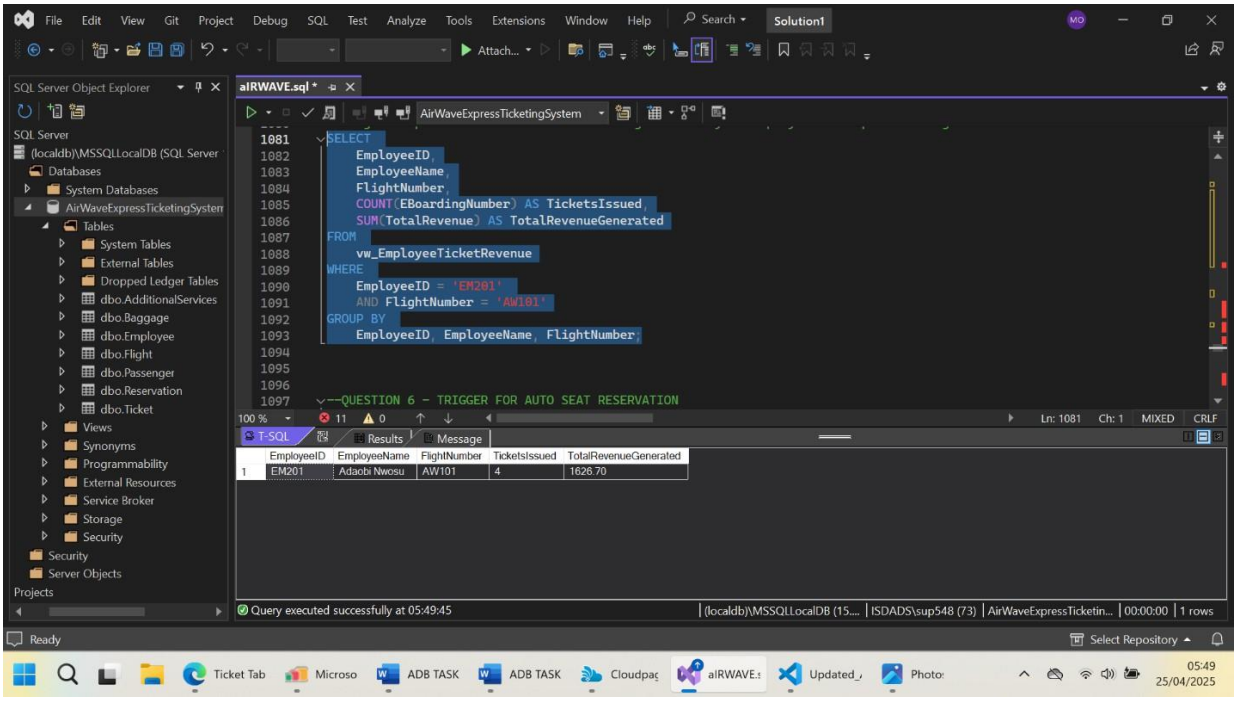


Usage Examples

- View all tickets issued by a specific employee (EmployeeID: EM201):



- Calculate total revenue generated by an employee on a specific flight:

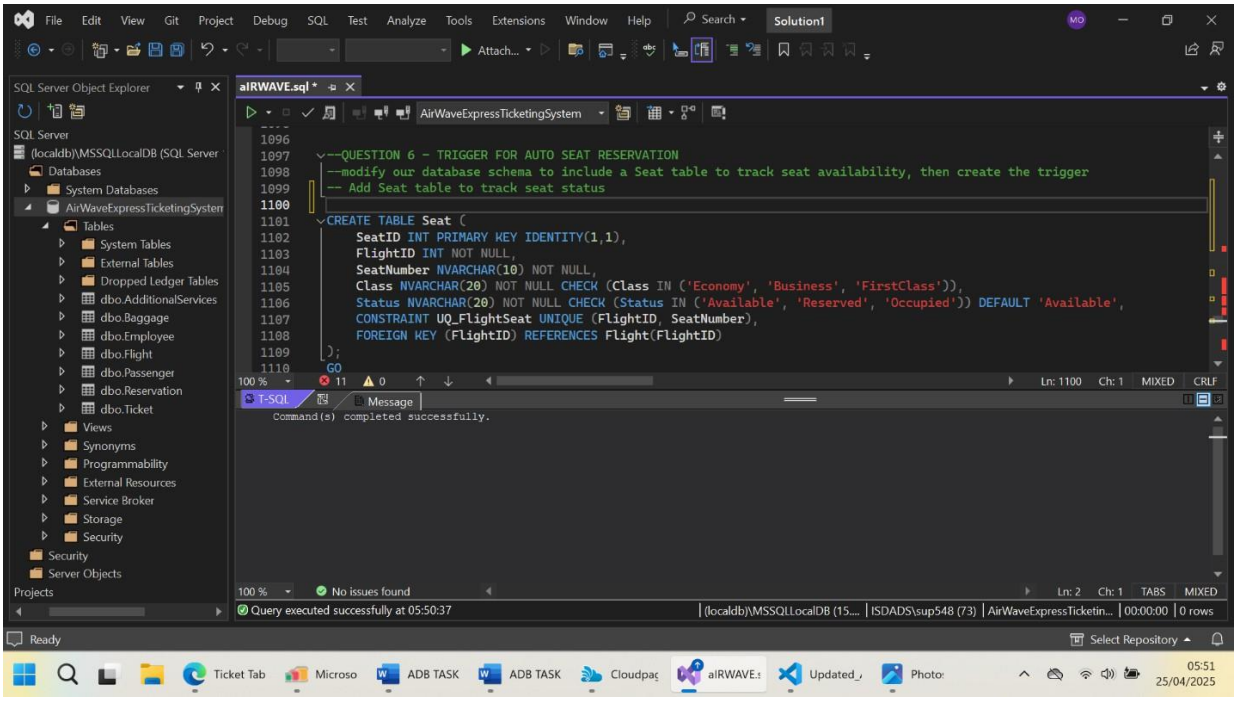


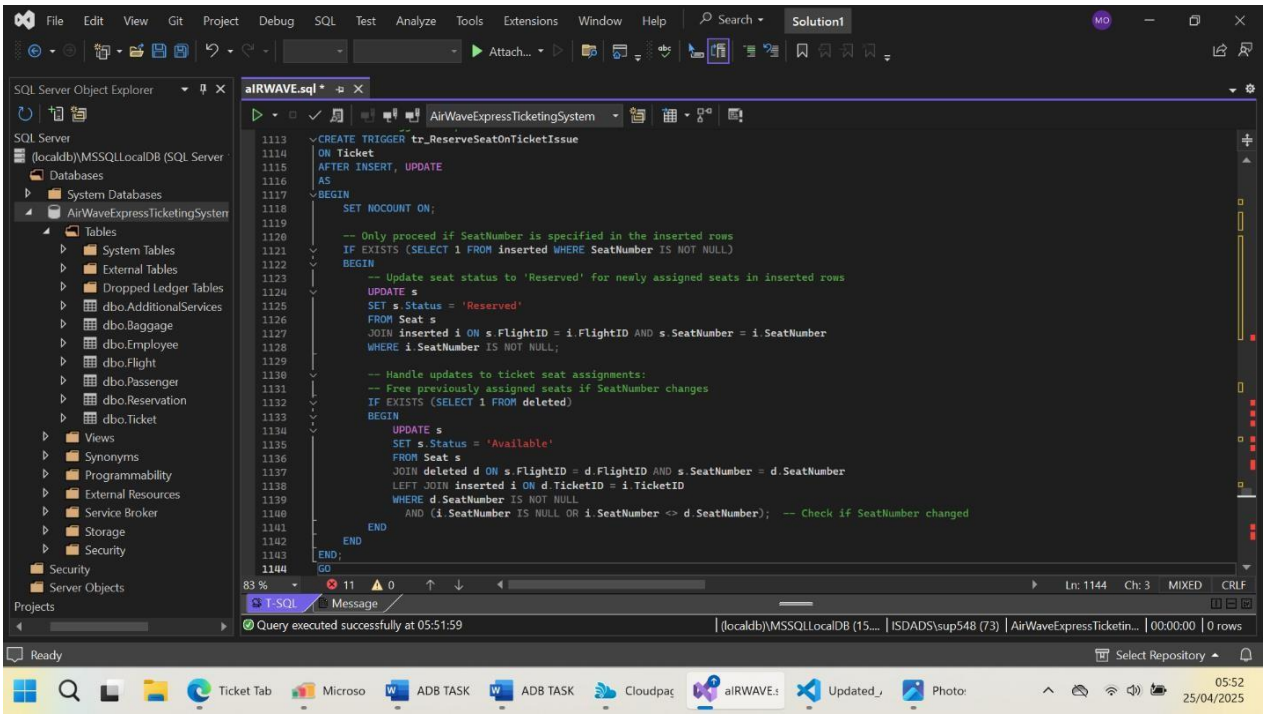
3.5 Q6. Trigger for Automatic Seat Reservation Update

I created a trigger that automatically updates the seat status to "Reserved" when a ticket is issued with a seat assignment. This ensures seat allocations are properly tracked in real time.

Implementation

First, we need to modify our database schema to include a `Seat` table to track seat availability, then create the trigger:

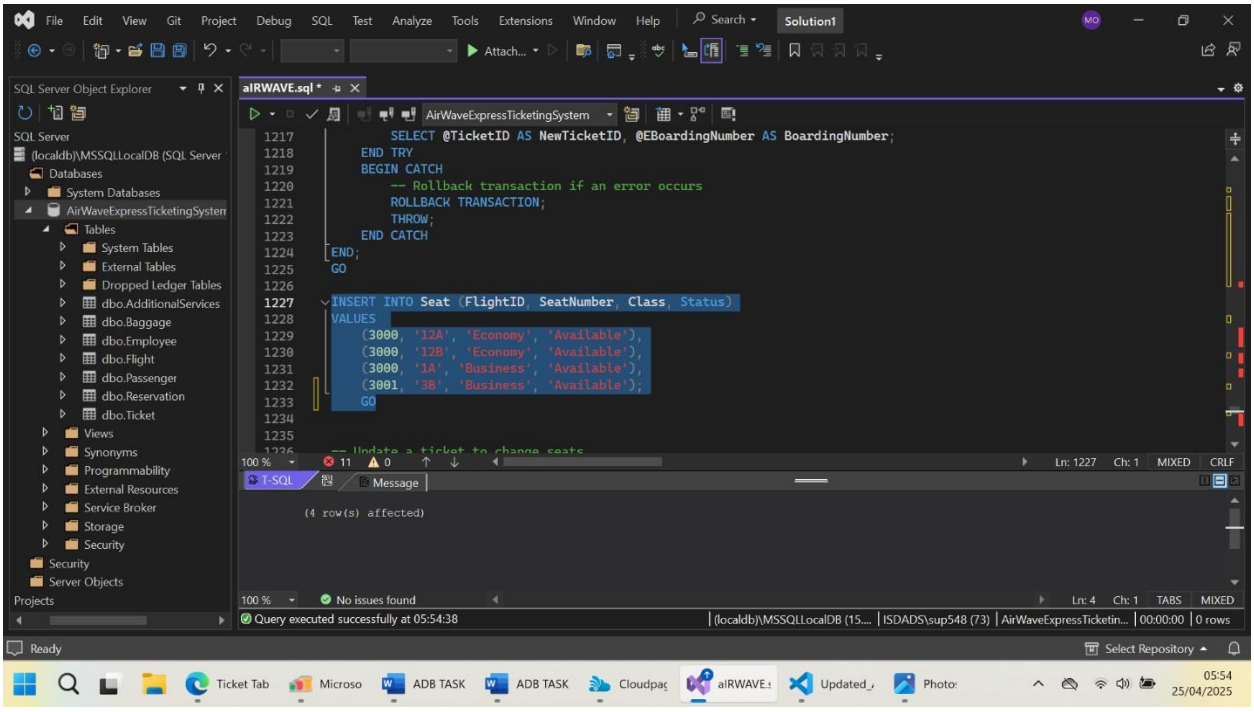




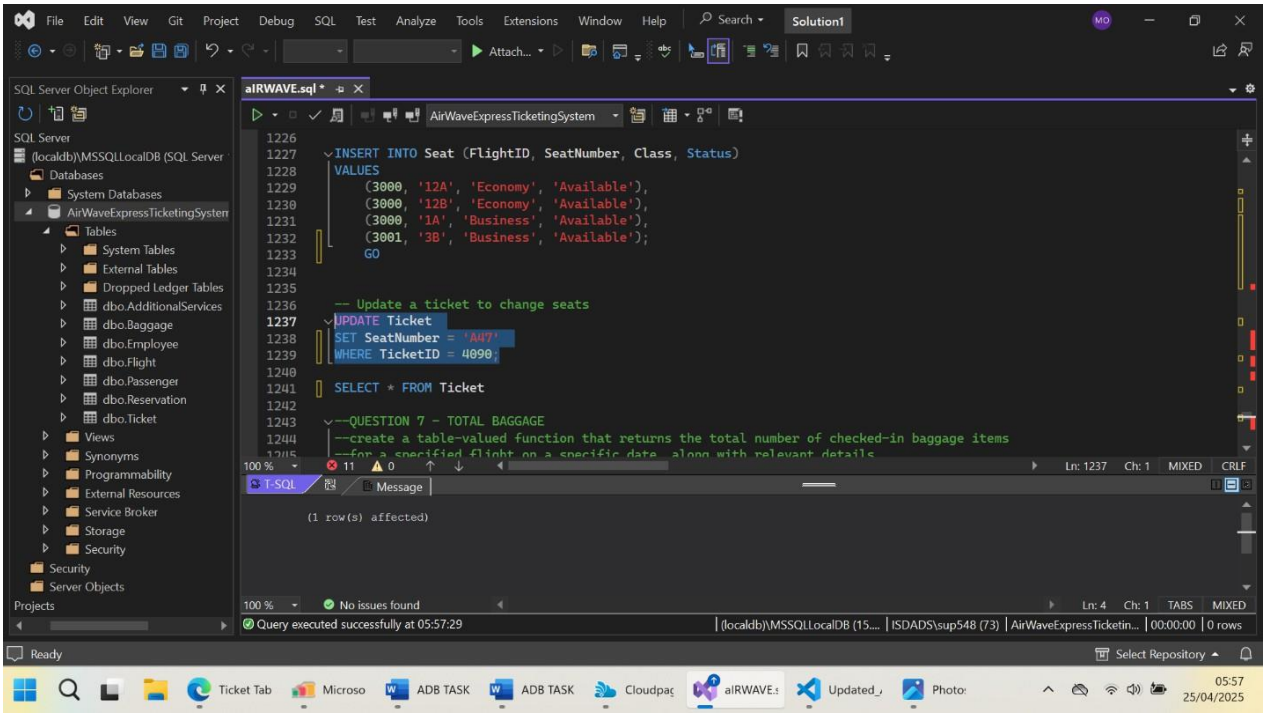
- *Testing the Trigger*

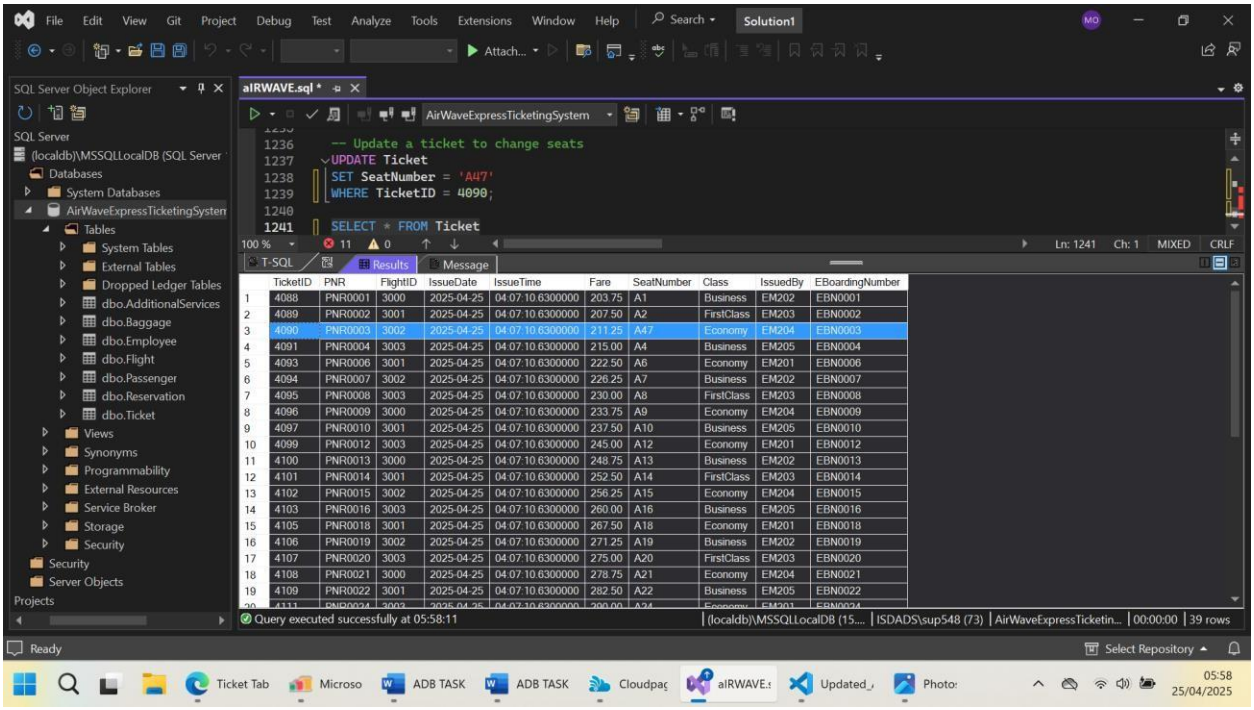
Let's test the functionality:

- First, populate some seats for our test flight



- Update a ticket to change seats



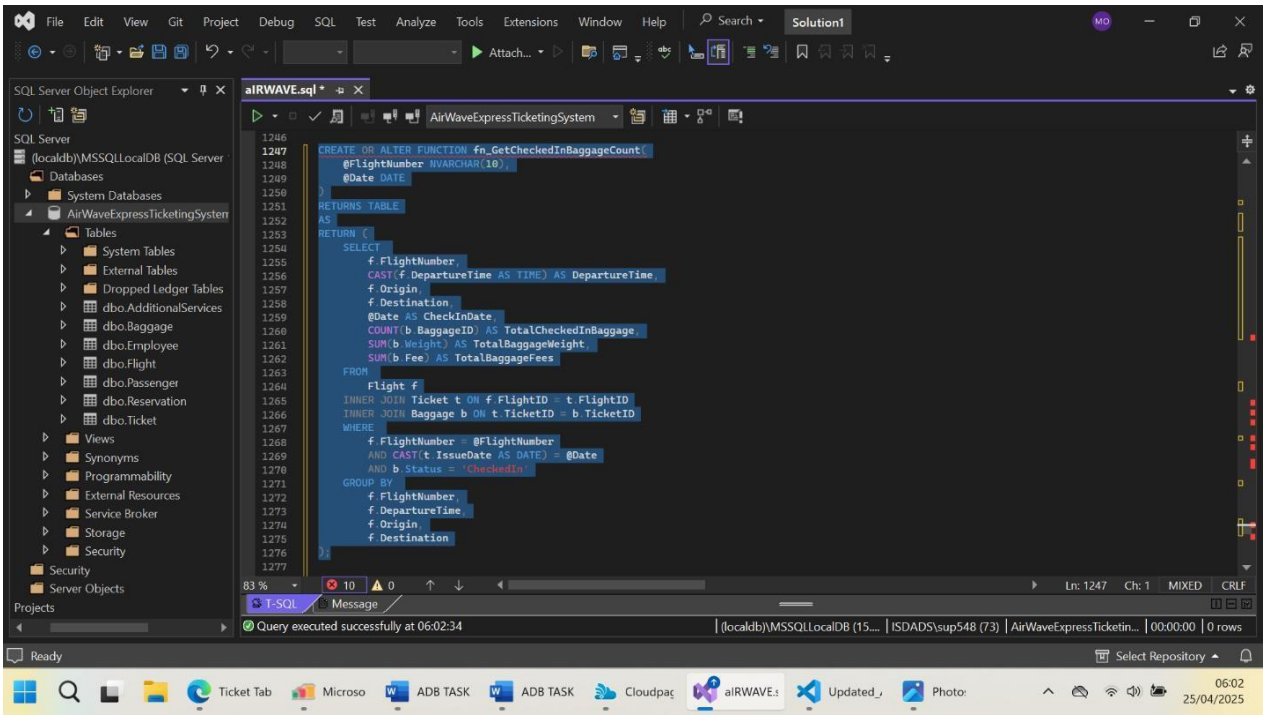


This solution provides a robust mechanism for managing seat allocations automatically whenever tickets are issued or modified, ensuring data consistency and preventing seat conflicts.

3.6 Q7. Function to Identify Checked-In Baggage for a Specific Flight and Date

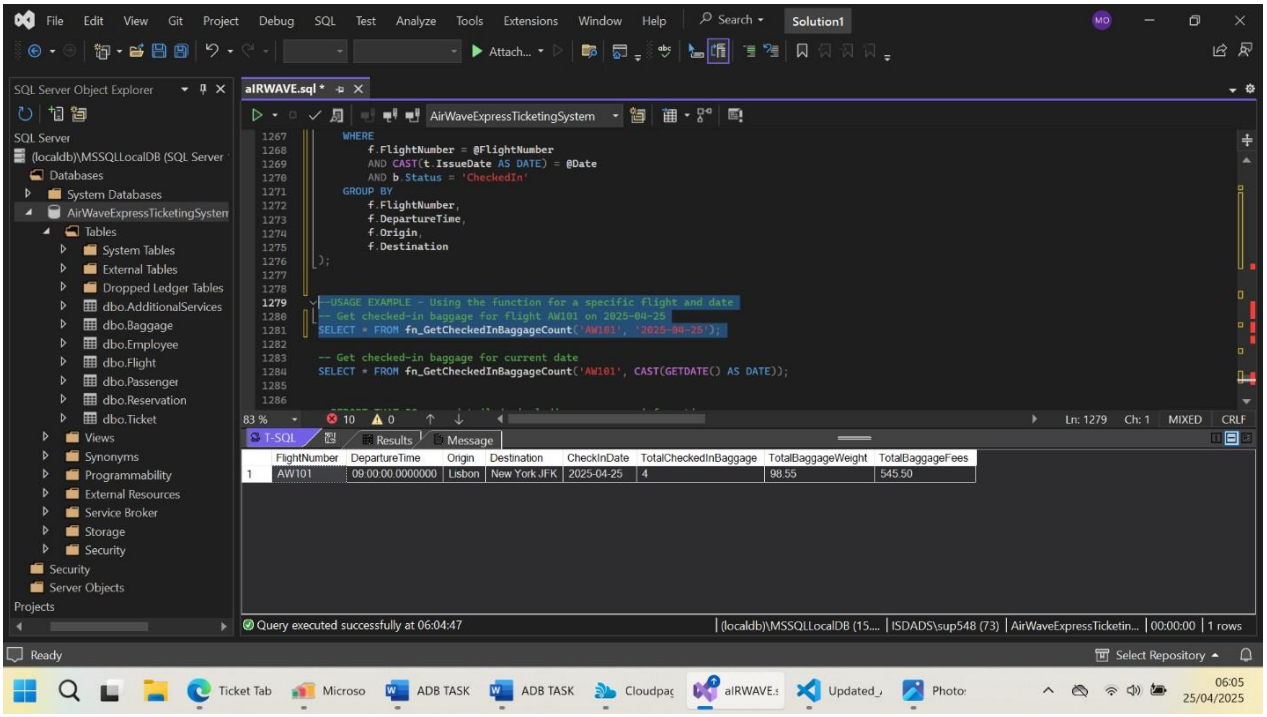
I created a table-valued function that returns the total number of checked-in baggage items for a specified flight on a specific date, along with relevant details.

- *Implementation*

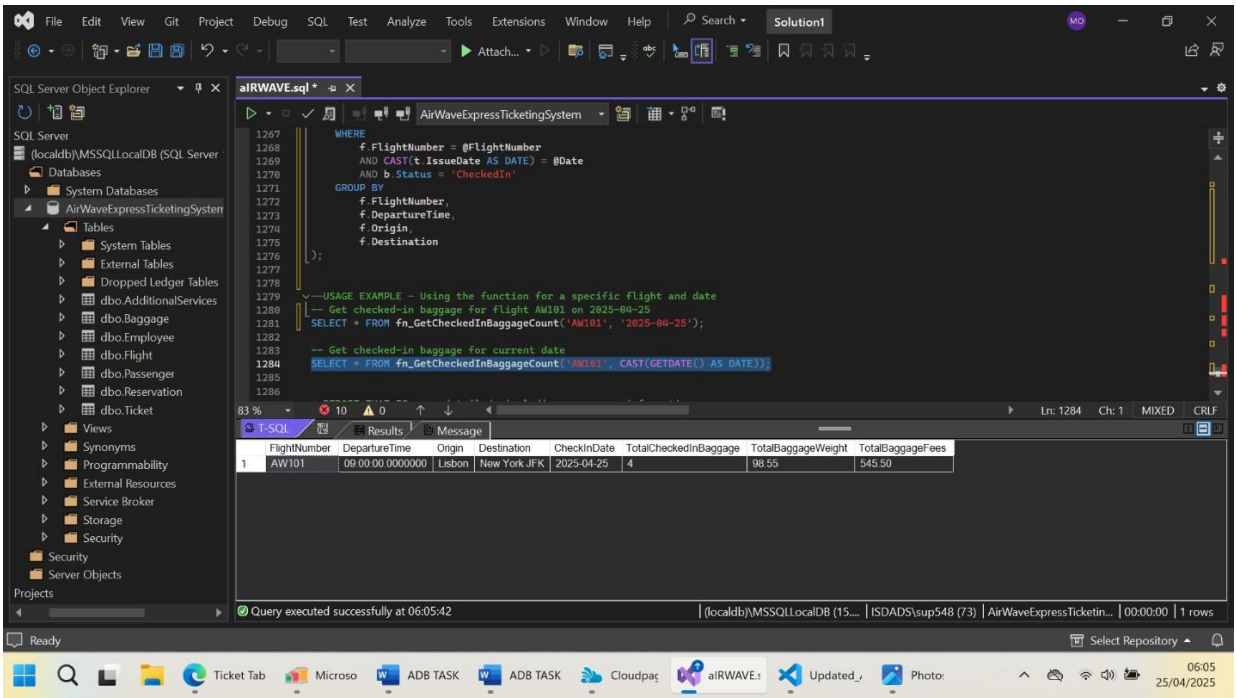


Usage Examples
Using the function for a specific flight and date:

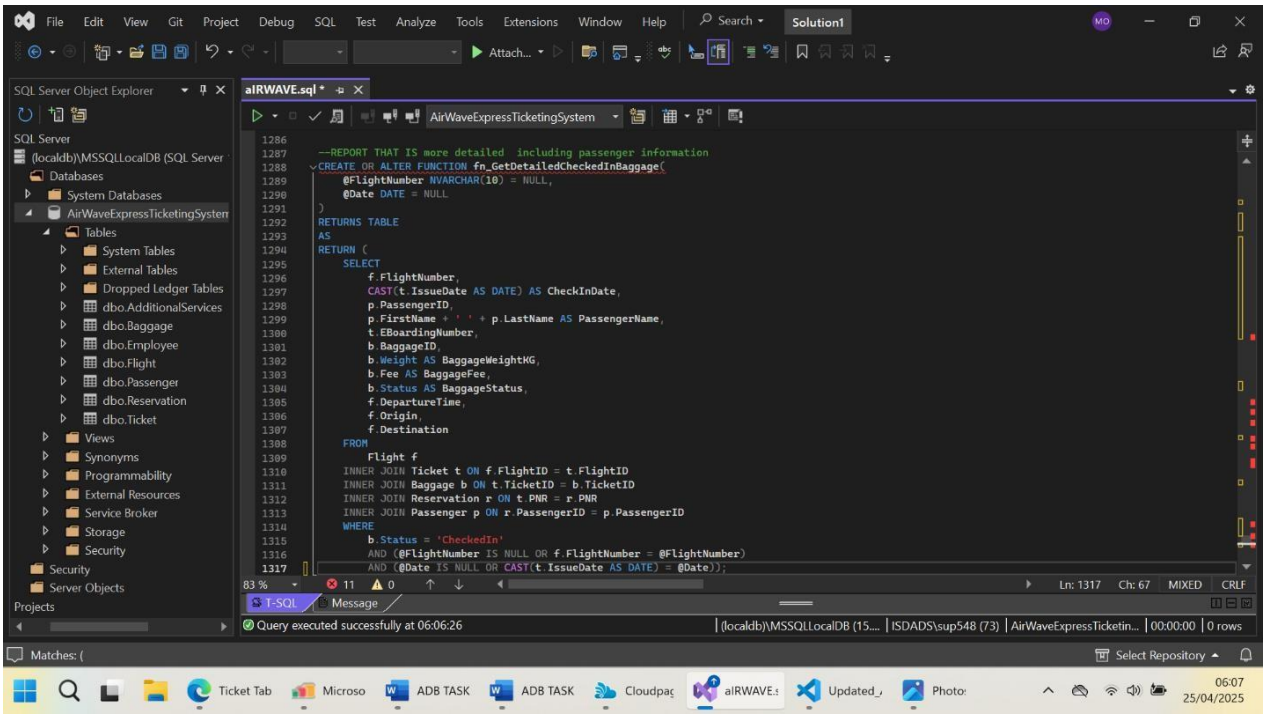
Get checked-in baggage for flight AW101 on April 25, 2025

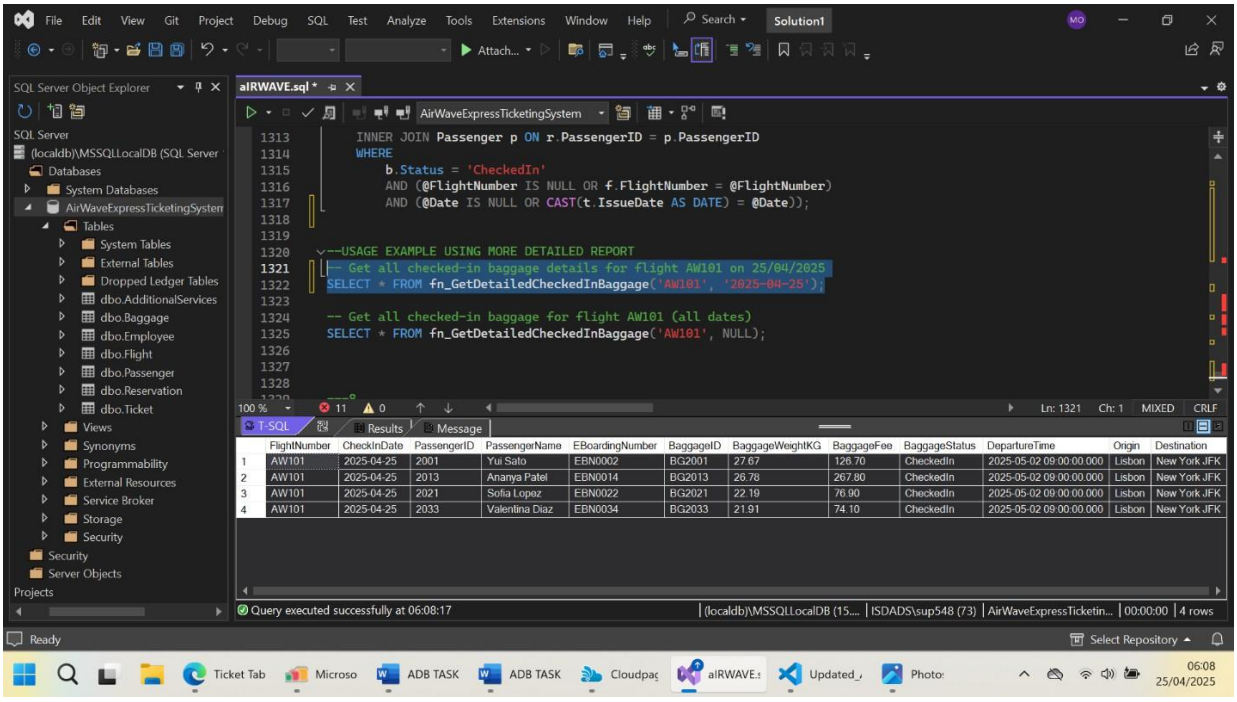


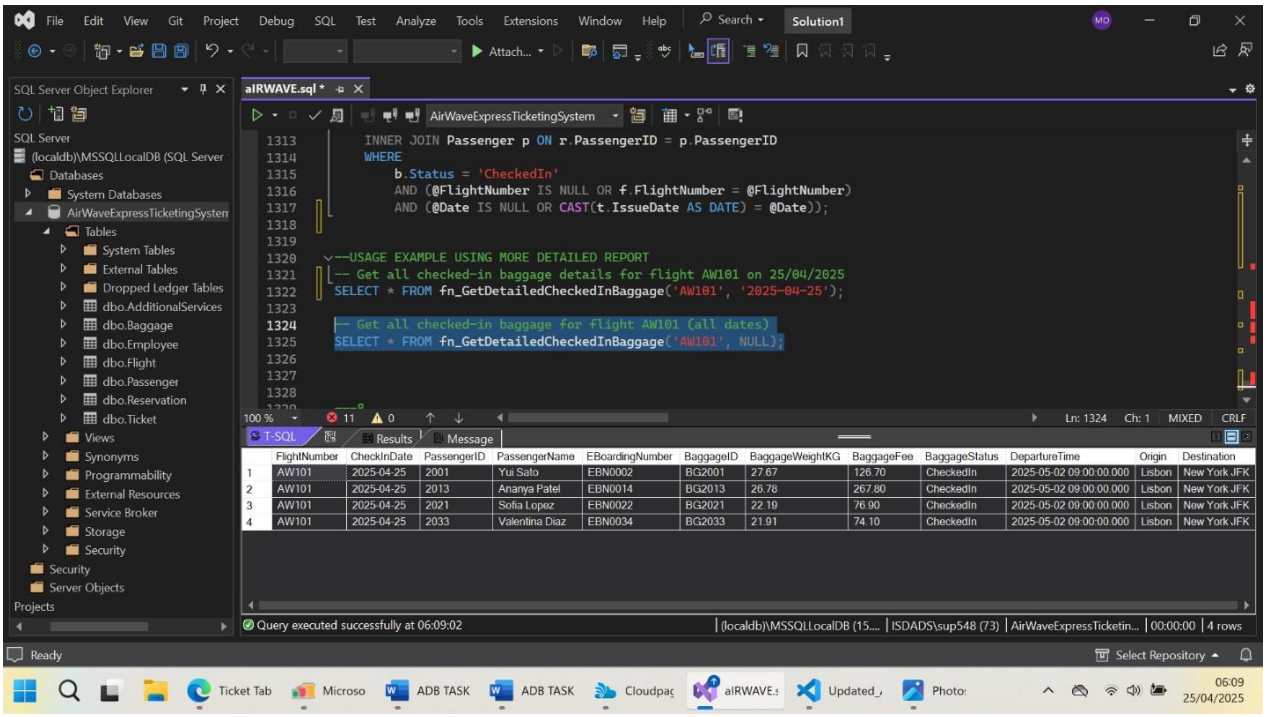
Get checked-in baggage for current date



- *Enhanced Version with Passenger Details*



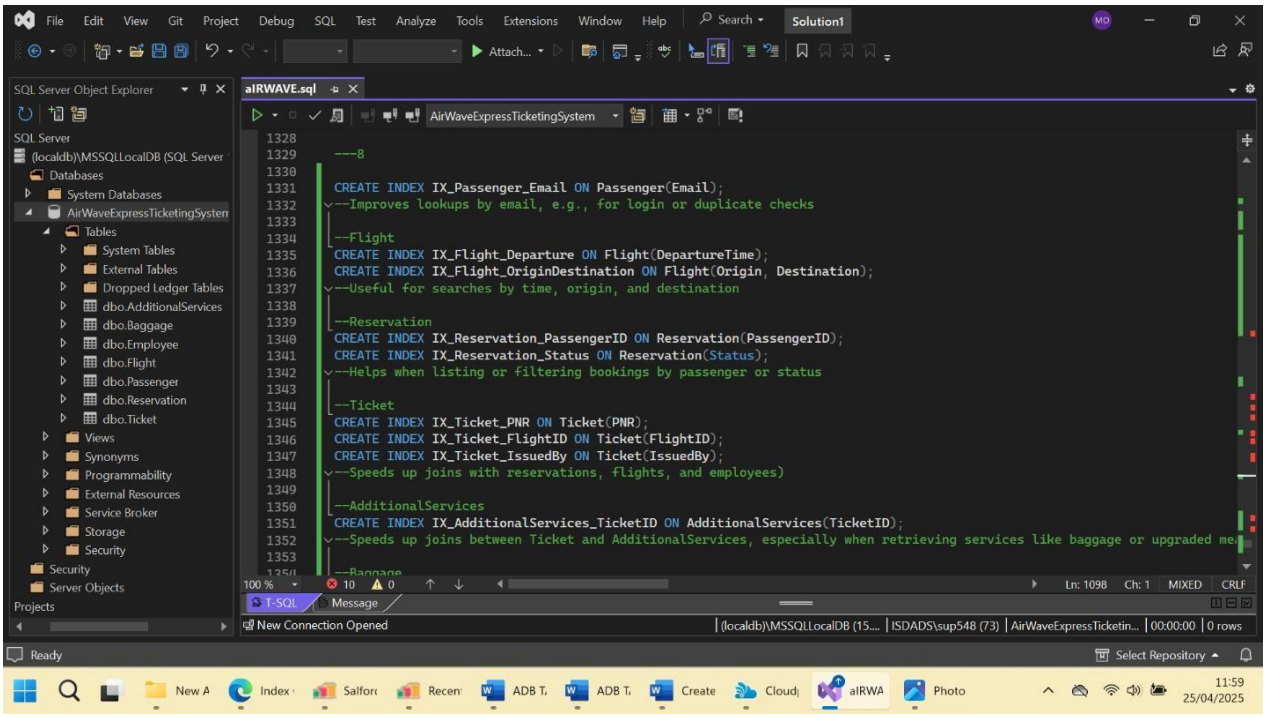


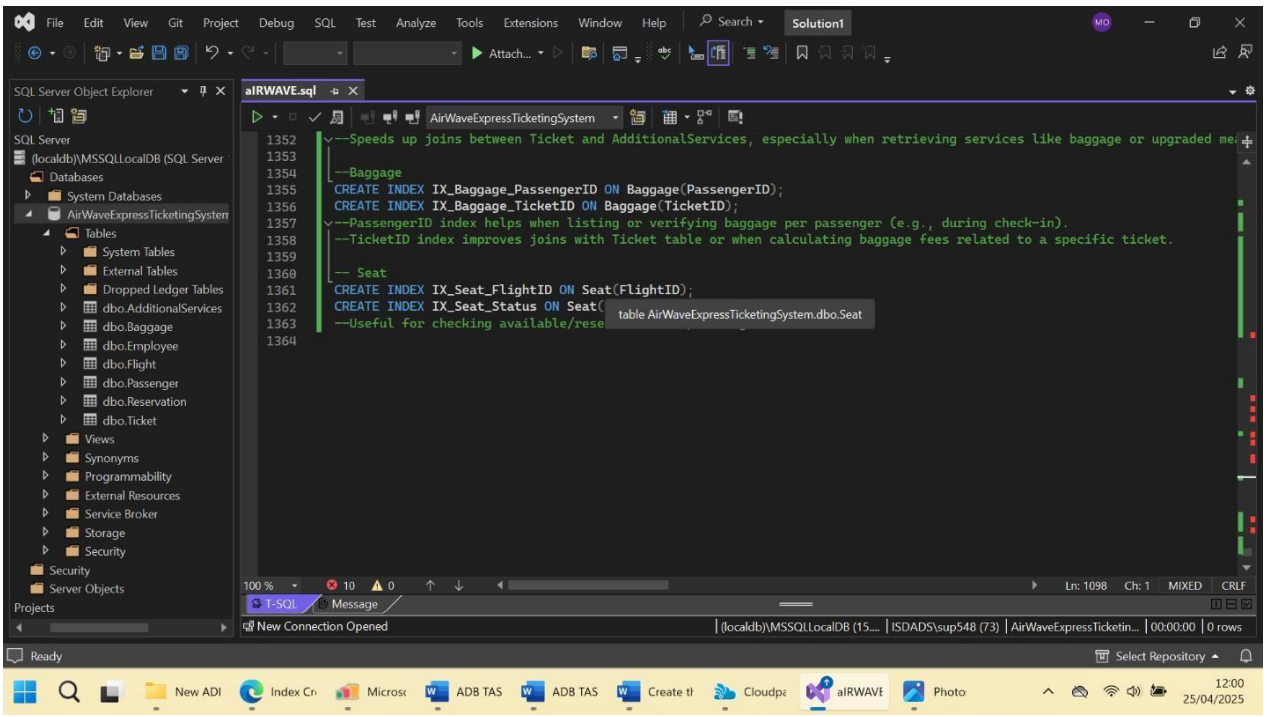


This solution gives the ticketing system complete visibility into baggage check-in operations for any flight on any date, with the flexibility to view summary statistics or detailed passenger-level information.

3.7 Q8. Additional Database Objects for Enhanced Airport Ticketing System

I created these indexes to make the system faster and more efficient, especially for the kinds of searches and actions users will do most often—like looking up a passenger by email during login, finding flights based on time and route, or checking a passenger's reservations and baggage. By adding indexes to the columns most frequently used in searches, filters, and joins, I've helped the database respond more quickly and reduced the load during common tasks. It's all about making the experience smoother for both users and the system.





4. Additional Suggestions

To make the system truly ready for everyday use, a few extra touches would go a long way like hashing passwords for safety, using parameterized queries to guard against SQL

injection, and adding clear error messages to help with troubleshooting. Making sure commonly searched fields are indexed and using transactions when needed can also help things run more smoothly. Simple checks, like catching duplicate emails or validating roles, help keep the data clean and user-friendly.

5. Final Thoughts

All in all, this database does a solid job of meeting the needs of an airport ticketing system. It's built with care, keeps things secure and fast, and follows best practices to make sure it runs reliably now and scales well in the future.