Royaume du Maroc
Université Abdelmalek Essaâdi
Faculté des Sciences et Techniques – Tanger

Master AI & DATA Science

University Year: 2023-2024

BDD Assignment Report

Intituled :

SI Departement

Framed By :

Pr. Mostafa Ezziyyani

Du 4/12/2023 au 1/12/2023

Realised By :

Abdelmajid Benjelloun	H130407540	20000203
Mohammed Aachabi	N130119025	20001710

.

I. Introduction :

My colleague and I have undertaken a series of exercises focusing on database management within the context of an airline system. These exercises involved analyzing a given database schema, understanding the business rules associated with it, and addressing various queries and scenarios using SQL queries, procedures, triggers, and cursors.

To provide a structured and systematic approach to our solutions, we have utilized several modeling techniques, including use case diagrams, sequence diagrams, and class diagrams. These diagrams offer visual representations of the interactions, processes, and structure of the database system, aiding in our understanding and communication of the solutions developed.

In this report, we present our solutions to the exercises, detailing our approach, rationale, and the methodologies employed. Each exercise is accompanied by relevant SQL queries and explanations, demonstrating our proficiency in database management principles and techniques.

Through these exercises, we have aimed to enhance our skills in database design, implementation, and query optimization, while also gaining practical insights into the complexities of managing data within the aviation industry. Our collaborative efforts have enabled us to tackle diverse scenarios, apply theoretical knowledge to practical problems, and develop effective solutions within a database management context.

We now present our findings and solutions, showcasing our analytical abilities and problem-solving skills in the domain of airline database management.

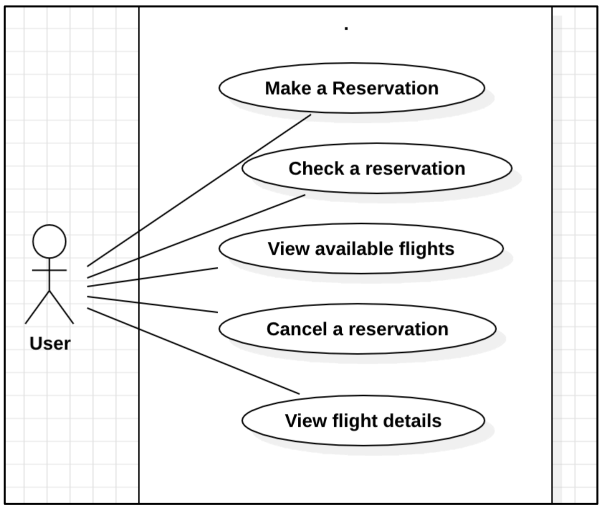
II. Conception Phase:

In this second chapter, we transition from the conceptualization phase to the practical implementation of our database solutions. Building upon the foundation established in the previous chapter, we delve into the intricacies of database design and query execution. Our focus shifts towards translating conceptual models into tangible database structures and executing queries to extract meaningful insights from the data.

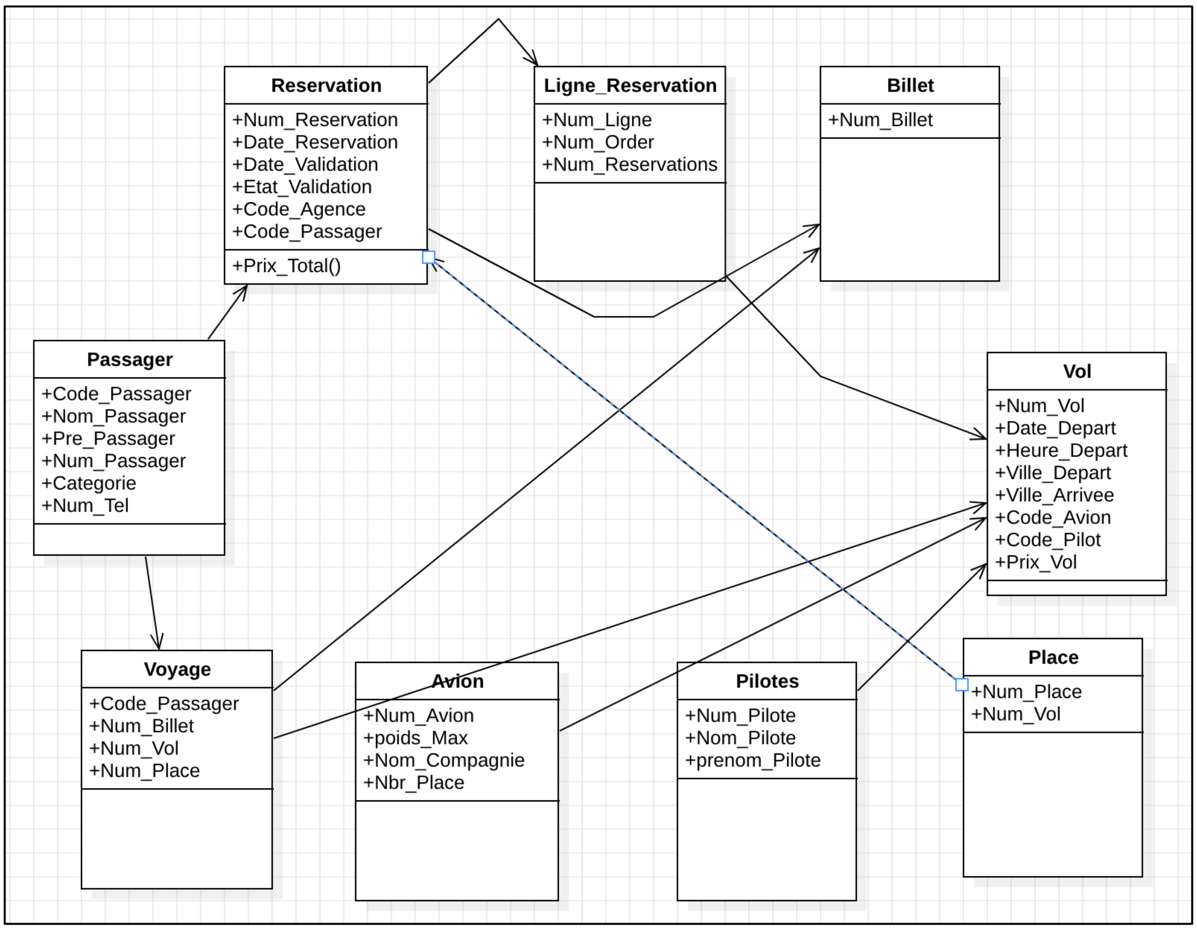
This chapter serves as a bridge between theoretical concepts and practical application, showcasing our ability to transform design blueprints into functional database systems. Through a series of exercises and scenarios, we demonstrate our proficiency in SQL query formulation, database manipulation, and optimization techniques.

Join us as we navigate through the implementation phase, presenting our approach, methodologies, and solutions to real-world database challenges. Through hands-on experimentation and problem-solving, we aim to refine our skills and deepen our understanding of database management principles.

we outline the key interactions between users and our airline management system. By defining clear and concise use cases, we aim to capture the core functionalities and user actions essential for system operation and usability. Through this analysis, we lay the groundwork for designing intuitive interfaces and implementing robust backend functionalities.



In this section, we present a class diagram depicting the structure and relationships of entities within our airline management system. The diagram provides a comprehensive overview of the system's architecture, including entities such as passengers, reservations, flights, aircraft, and pilots. Through this visual representation, we aim to elucidate the interdependencies between entities and facilitate a deeper understanding of the system's design and functionality.


Reservation
+Num_Reservation
+Date_Reservation
+Date_Validation
+Etat_Validation
+Code_Agence
+Code_Passager
+Prix_Total()
Ligne_Reservation
+Num_Ligne
+Num_Order
+Num_Reservations
Billet
+Num_Billet
Passager
+Code_Passager
+Nom_Passager
+Pre_Passager
+Num_Passager
+Categorie
+Num_Tel
Vol
+Num_Vol
+Date_Depart
+Heure_Depart
+Ville_Depart
+Ville_Arrivee
+Code_Avion
+Code_Pilot
+Prix_Vol
Voyage
+Code_Passager
+Num_Billet
+Num_Vol
+Num_Place
Avion
+Num_Avion
+poids_Max
+Nom_Compagnie
+Nbr_Place
Pilotes
+Num_Pilote
+Nom_Pilote
+prenom_Pilote
Place
+Num_Place
+Num_Vol

III. Exercise Solutions and Analysis.

Execice 1:

This algorithm defines a stored procedure in SQL Server to find all integers less than a given number whose digits sum up to 6. It stores the results, including the counts of even and odd digits, in a temporary table. Utilizing nested loops, it iterates through each integer, calculates the digit sums, and inserts qualifying integers into the temporary table. Finally, it returns the results from the temporary table and drops it once the operation is complete, providing an efficient solution for the specified task.

```
CREATE PROCEDURE dbo.Ex1_Done
(
    @nombre INT
)
AS
BEGIN

    CREATE TABLE #Resultats
    (
        Entier INT,
        ChiffresPairs INT,
        ChiffresImpairs INT
    );

    DECLARE @i INT = 1;

    WHILE @i < @nombre
    BEGIN
        DECLARE @chaine VARCHAR(10) = CAST(@i AS VARCHAR(10));
        DECLARE @longueur INT = LEN(@chaine);
        DECLARE @j INT = 1;
        DECLARE @somme INT = 0;
        DECLARE @pairs INT = 0;
        DECLARE @impairs INT = 0;
        WHILE @j <= @longueur
        BEGIN
            DECLARE @chiffre INT = CAST(SUBSTRING(@chaine, @j, 1) AS INT);
            SET @somme = @somme + @chiffre;

            IF @chiffre % 2 = 0
                SET @pairs = @pairs + 1;
            ELSE
                SET @impairs = @impairs + 1;

            SET @j = @j + 1;
        END
        IF @somme = 6
            INSERT INTO #Resultats (Entier, ChiffresPairs, ChiffresImpairs)
            VALUES (@i, @pairs, @impairs)
        SET @i = @i + 1;
    END
    SELECT * FROM #Resultats;
    DROP TABLE #Resultats;
END
GO
```

We have executed our programme using :

```
EXEC Ex1_Done @nombre = 100;
```

Executing the stored procedure Ex1_Done with the parameter @nombre set to 100 results in the creation of a temporary table. This table comprises three columns: 'Entier' (Integer), 'Chiffres Paires' (Even Digits), and 'Chiffres Impairs' (Odd Digits). The table is populated with numbers up to 100, where each row represents a number along with the count of its even and odd digits.

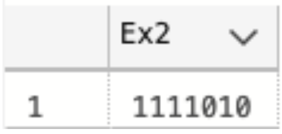
Exercice 2:

In this algorithm, we introduce a stored function named Ex2 designed to calculate the binary representation of an integer input. The function takes an integer value as input and returns a string representing its binary equivalent. By employing a systematic approach, the function iteratively divides the input integer by 2, capturing the remainders at each step to construct the binary string.

```
CREATE FUNCTION Ex2
(
    @input INT
)
RETURNS NVARCHAR(MAX)
AS
BEGIN
    DECLARE @binaryString NVARCHAR(MAX) = '';
    DECLARE @remainder INT;

    IF @input = 0
        RETURN '0';

    WHILE @input > 0
    BEGIN
        SET @remainder = @input % 2;
        SET @binaryString = CONVERT(NVARCHAR(1), @remainder) + @binaryString;
        SET @input = @input / 2;
    END

    RETURN @binaryString;
END
GO
```

The result after executing the programme

```sql
SELECT dbo.Ex2(122) AS Ex2;
```



```
   Ex2
1  1111010
```

Exercice 3:

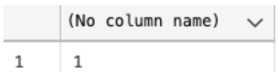
In this task, we aim to develop a stored function in SQL Server that determines whether a given string is a palindrome or not. A palindrome is defined as a sequence of characters that reads the same forwards and backwards, disregarding spaces, punctuation, and capitalization. For instance, "TOTOT" and "TOUSUOT" are both examples of palindromes.

```sql
CREATE FUNCTION IsPalindrome(@str VARCHAR(255)) RETURNS BIT
AS
BEGIN
    DECLARE @len INT;
    DECLARE @i INT;
    DECLARE @isPal BIT;

    SET @len = LEN(@str);
    SET @i = 1;
    SET @isPal = 1;

    WHILE @i <= @len / 2
    BEGIN
        IF SUBSTRING(@str, @i, 1) != SUBSTRING(@str, @len - @i + 1, 1)
        BEGIN
            SET @isPal = 0;
            BREAK;
        END;

        SET @i = @i + 1;
    END;

    RETURN @isPal;
END;
```

To Execute the programme :

```sql
SELECT dbo.IsPalindrome('TOTOT');
```

The programme gives 1 if the string is a palindrom else the output is 0



```
   (No column name)
1  1
```

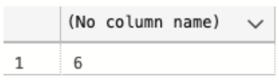
Exercice 4

In this task, we're tasked with writing a stored function that counts the number of words in a given string of characters.

```sql
CREATE FUNCTION CompterMots(@chaine VARCHAR(MAX)) RETURNS INT
AS
BEGIN
    DECLARE @nombreMots INT = 0;
    DECLARE @position INT = 1;
    DECLARE @longueur INT = LEN(@chaine);
    DECLARE @caracterePrecedent CHAR(1);

    WHILE @position <= @longueur
    BEGIN
        IF @caracterePrecedent IS NULL OR @caracterePrecedent IN (' ', CHAR(9),
CHAR(10), CHAR(13))
        BEGIN
            IF SUBSTRING(@chaine, @position, 1) NOT IN (' ', CHAR(9), CHAR(10),
CHAR(13))
            BEGIN
                SET @nombreMots = @nombreMots + 1;
            END;
        END;

        SET @caracterePrecedent = SUBSTRING(@chaine, @position, 1);
        SET @position = @position + 1;
    END;

    RETURN @nombreMots;
END;
```

To Execute the programme:

```sql
SELECT dbo.CompterMots('Master intelligence artificielle et data science');
```



	(No column name)
1	6

Exercice 5:

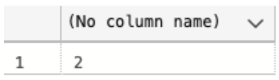
In this task, we are going to create a stored function that counts the number of occurrences of a given string within another string of characters.

```
 CREATE FUNCTION CompterOccurences(@chaine VARCHAR(MAX), @motCherche VARCHAR(MAX))
RETURNS INT
AS
BEGIN
    DECLARE @index INT = 1;
    DECLARE @count INT = 0;
    DECLARE @lenChaine INT = LEN(@chaine);
    DECLARE @lenMot INT = LEN(@motCherche);

    WHILE @index <= @lenChaine - @lenMot + 1
    BEGIN
        IF SUBSTRING(@chaine, @index, @lenMot) = @motCherche
        BEGIN
            SET @count = @count + 1;
            SET @index = @index + @lenMot - 1;
        END;
        SET @index = @index + 1;
    END;

    RETURN @count;
END;
```

To Execute the programme:

```
SELECT dbo.CompterOccurences('abdelmajidabd', 'abd');
```

The second column shows the number of occurrences of the 2nd name:



	(No column name)
1	2

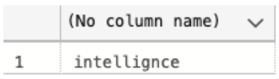
Exercice 6:

In this task, we are required to create a stored function that finds the longest word in a given string of characters.

```
CREATE FUNCTION FindLongestWord(@inputString VARCHAR(MAX)) RETURNS VARCHAR(MAX)
AS
BEGIN
    DECLARE @longestWord VARCHAR(MAX) = '';
    DECLARE @words TABLE (word VARCHAR(MAX));

    -- Insert each word into the temporary table
    INSERT INTO @words (word)
    SELECT value
    FROM STRING_SPLIT(@inputString, ' ');

    -- Select the longest word
    SELECT TOP 1 @longestWord = word
    FROM @words
    ORDER BY LEN(word) DESC;

    RETURN @longestWord;
END;
GO
```

To Execute the programme:

```
SELECT dbo.FindLongestWord('Master Artificial intellignce and data science');
```



	(No column name)
1	intellignce

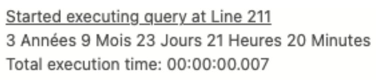
Exercice 7:

In this task, we're tasked with creating a stored procedure that displays a given number of minutes X in the format: "AA Années MM Mois JJJ Jours HH Heures MM Minutes", without using any built-in predefined functions.

```
CREATE PROCEDURE DisplayMinutesX
    @minutes INT
AS
BEGIN
    DECLARE @years INT, @months INT, @days INT, @hours INT;

    SET @years = @minutes / (60 * 24 * 365);
    SET @minutes = @minutes % (60 * 24 * 365);

    SET @months = @minutes / (60 * 24 * 30);
    SET @minutes = @minutes % (60 * 24 * 30);

    SET @days = @minutes / (60 * 24);
    SET @minutes = @minutes % (60 * 24);

    SET @hours = @minutes / 60;
    SET @minutes = @minutes % 60;

    PRINT CAST(@years AS VARCHAR(5)) + ' Années ' +
          CAST(@months AS VARCHAR(5)) + ' Mois ' +
          CAST(@days AS VARCHAR(5)) + ' Jours ' +
          CAST(@hours AS VARCHAR(5)) + ' Heures ' +
          CAST(@minutes AS VARCHAR(5)) + ' Minutes';
END;
```

To Execute the programme:

```
EXEC DisplayMinutesX @minutes = 2000000;
```



```
Started executing query at Line 211
3 Années 9 Mois 23 Jours 21 Heures 20 Minutes
Total execution time: 00:00:00.007
```

Exercice 8:

In this task, we're tasked with creating a stored procedure that creates the Vols table. The Vols table represents flights and is an essential part of our airline management system. We'll ensure all necessary constraints are applied to maintain data integrity, including structural and referential integrity constraints.

```
CREATE PROCEDURE CreateVolsTable
AS
BEGIN
    SET NOCOUNT ON;

    IF OBJECT_ID('Vols', 'U') IS NOT NULL
    BEGIN
        PRINT 'The Vols table already exists.';
        RETURN;
    END

    CREATE TABLE Vols (
        Num_Vol INT PRIMARY KEY,
        Date_Depart DATE,
        Heure_Depart TIME,
        Ville_Depart VARCHAR(100),
        Ville_Arrivee VARCHAR(100),
        Code_Avion INT,
        Code_Pilote INT,
        Prix_Vol DECIMAL(10, 2),
        CONSTRAINT FK_Avion FOREIGN KEY (Code_Avion) REFERENCES Avions(Num_Avion),
        CONSTRAINT FK_Pilote FOREIGN KEY (Code_Pilote) REFERENCES
Pilotes(Num_Pilote)
    );

    PRINT 'The Vols table has been created successfully.';
END;
```

To Execute the programme:

```
EXEC CreateVolsTable;
```

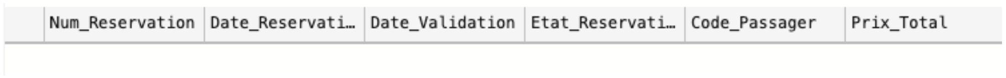
Exercice 9:

In this task, we aim to create a stored procedure that displays all non-validated reservations for a specific date. This stored procedure will help us efficiently manage reservations in our system, allowing us to identify and address any outstanding reservations.

```
CREATE PROCEDURE DisplayNonValidatedReservations
    @targetDate DATE
AS
BEGIN
    SET NOCOUNT ON;
    SELECT *
    FROM Reservations
    WHERE Date_Validation IS NULL
    AND Date_Reservation = @targetDate;
END;
```

To Execute the programme:

```
EXEC DisplayNonValidatedReservations @targetDate = '2024-03-15';
```



Num_Reservation	Date_Reservati…	Date_Validation	Etat_Reservati…	Code_Passager	Prix_Total

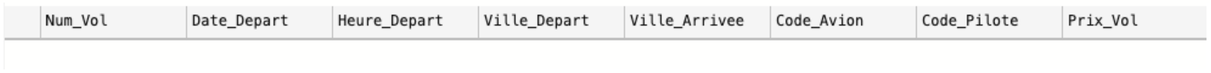
Exercice 10:

This stored procedure, named DisplayFlightInformation, is designed to fetch and display all information related to a given flight. It aims to provide comprehensive details about a specific flight, facilitating efficient retrieval and management of flight-related data.

```
CREATE PROCEDURE DisplayFlightInformation
    @flightNumber INT
AS
BEGIN
    SET NOCOUNT ON;

    SELECT *
    FROM Vols
    WHERE Num_Vol = @flightNumber;
END;
```

To Execute the programme:

```
EXEC DisplayFlightInformation @flightNumber = 123;
```



Num_Vol	Date_Depart	Heure_Depart	Ville_Depart	Ville_Arrivee	Code_Avion	Code_Pilote	Prix_Vol

Exercie 11:

To fulfill the task of displaying all information about a given flight, including details about pilots, departure city/time, arrival city/time, and details of any layovers, we will create a stored procedure.

```
CREATE PROCEDURE DisplayFlightInformation
    @flightNumber INT
AS
BEGIN
    SET NOCOUNT ON;

    SELECT V.Num_Vol,
```

```sql
           V.Date_Depart,
           V.Heure_Depart,
           V.Ville_Depart,
           V.Ville_Arrivee,
           PD.Nom_Pilote AS Nom_Pilote_Depart,
           PD.Prenom_Pilote AS Prenom_Pilote_Depart,
           PA.Nom_Pilote AS Nom_Pilote_Arrivee,
           PA.Prenom_Pilote AS Prenom_Pilote_Arrivee,
           E.Ville AS Ville_Escale,
           E.Heure_Arrivee AS Heure_Arrivee_Escale,
           E.Duree_Escale
    FROM Vols V
    LEFT JOIN Pilotes PD ON V.Code_Pilote = PD.Num_Pilote
    LEFT JOIN Pilotes PA ON V.Code_Pilote = PA.Num_Pilote
    LEFT JOIN Escales E ON V.Num_Vol = E.Num_Vol
    WHERE V.Num_Vol = @flightNumber;
END;
GO
EXEC DisplayFlightInformation @flightNumber = 123;
```

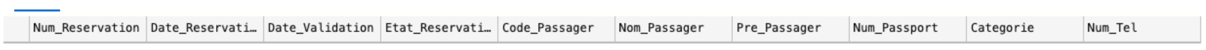
Exercice 12:

To display all information about a validated reservation (ticket), we will create a stored procedure.

```sql
CREATE PROCEDURE DisplayValidatedReservation
    @reservationNumber INT
AS
BEGIN
    SET NOCOUNT ON;

    SELECT R.Num_Reservation,
           R.Date_Reservation,
           R.Date_Validation,
           R.Etat_Reservation,
           P.Code_Passager,
           P.Nom_Passager,
           P.Pre_Passager,
           P.Num_Passport,
           P.Categorie,
           P.Num_Tel
    FROM Reservations R
    INNER JOIN Passagers P ON R.Code_Passager = P.Code_Passager
    WHERE R.Num_Reservation = @reservationNumber
          AND R.Date_Validation IS NOT NULL;
END;
```

To Execute the programme:

```sql
EXEC DisplayValidatedReservation @reservationNumber = 123;
```



Num_Reservation	Date_Reservati…	Date_Validation	Etat_Reservati…	Code_Passager	Nom_Passager	Pre_Passager	Num_Passport	Categorie	Num_Tel

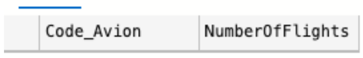
Exercice 13:

To display the number of flights for each airplane in descending order, we will create a stored procedure.

```
CREATE PROCEDURE DisplayNumberOfFlightsPerAirplane
AS
BEGIN
    SET NOCOUNT ON;

    SELECT V.Code_Avion, COUNT(*) AS NumberOfFlights
    FROM Vols V
    GROUP BY V.Code_Avion
    ORDER BY NumberOfFlights DESC;
END;
```

To Execute the programme:

```
EXEC DisplayNumberOfFlightsPerAirplane;
```



Code_Avion	NumberOfFlights

Exercice 14:

To calculate the number of trips for a given passenger, we will create a stored function.

```
CREATE FUNCTION CalculateNumberOfTripsForPassenger
(
    @passengerCode INT
)
RETURNS INT
AS
BEGIN
    DECLARE @numberOfTrips INT;

    SELECT @numberOfTrips = COUNT(*)
    FROM Voyages
    WHERE Code_Passager = @passengerCode;

    RETURN @numberOfTrips;
END;
```

To Execute the programme:

```
DECLARE @passengerCode INT = 123; -- Replace 123 with the desired passenger code

SELECT dbo.CalculateNumberOfTripsForPassenger1(@passengerCode) AS NumberOfTrips;
```

Exercice 15:

```
CREATE FUNCTION CalculateFlightCost1
(
    @flightNumber INT
)
RETURNS DECIMAL(10, 2)
AS
BEGIN
    DECLARE @cost DECIMAL(10, 2);

    -- Calculate cost based on airplane
    DECLARE @airplaneCost DECIMAL(10, 2);
    SELECT @airplaneCost = AVG(Prix_Vol) FROM Vols WHERE Num_Vol = @flightNumber;

    -- Return the total cost
    RETURN @airplaneCost;
END;
```

To Execute the programme:

```
DECLARE @flightNumber INT = 123; -- Replace 123 with the desired flight number
DECLARE @flightCost DECIMAL(10, 2);

SELECT @flightCost = dbo.CalculateFlightCost1(@flightNumber);

PRINT 'The estimated cost of flight ' + CAST(@flightNumber AS VARCHAR(10)) + ' is
$' + CAST(@flightCost AS VARCHAR(20));
```