**n2..Description**

**Objective:**  
  
To work with the alter table.

**Concept Explanation:**

The **ALTER** statement is used to modify the existing database. It allows you to make changes to the structure, properties, or constraints of an object without dropping and recreating it

**Concept Implementation:**

1. The ALTER TABLE statement is used to modify the flight\_Details table by adding a new column called total\_duration of type INT (integer). This modification expands the table's structure to include information about the duration of flights.
2. The ALTER TABLE statement is used to remove the Email column from the passenger table with the DROP COLUMN clause. This action permanently eliminates the Email column and its associated data from the table.

**Description:**

Refer to the given schema. Assume that the Passengers table has already been created.

1. Write a query to create a new column named ‘total\_duration‘ as an integer in the Flight\_details table.

2. Write a query to drop the Email column from the passengers table.

**Note: Evaluate only the respective DDL query to get the desired result.**

***SAMPLE:  
<Line a..1>  : Your SQL CREATE Statement (do not use semicolon - ; to terminate)  
<Line a..n>  : GO***

***Answer:***

**Description**

**Objective:**  
  
To work with the alter table.

**Concept Explanation:**

The **ALTER** statement is used to modify the existing database. It allows you to make changes to the structure, properties, or constraints of an object without dropping and recreating it

**Concept Implementation:**

1. The ALTER TABLE statement is used to modify the flight\_Details table by adding a new column called total\_duration of type INT (integer). This modification expands the table's structure to include information about the duration of flights.
2. The ALTER TABLE statement is used to remove the Email column from the passenger table with the DROP COLUMN clause. This action permanently eliminates the Email column and its associated data from the table.

**Description:**

Refer to the given schema. Assume that the Passengers table has already been created.

1. Write a query to create a new column named ‘total\_duration‘ as an integer in the Flight\_details table.

2. Write a query to drop the Email column from the passengers table.

**Note: Evaluate only the respective DDL query to get the desired result.**

***SAMPLE:  
<Line a..1>  : Your SQL CREATE Statement (do not use semicolon - ; to terminate)  
<Line a..n>  : GO***

***Answer:***

***Alter table flight\_detail***

***Add total\_duration int;***

***Alter table passangers***

***Drop column Email;***

***3..Description***

***Objective:***

***To work with the select statement with the where clause and aggregate function.***

***Concept Explanation:***

***The SELECT statement is used to retrieve data from one or more tables in a database. It also retrieves the particular data under specific conditions.***

***Concept Implementation:***

***The Select Statement is used to retrieve the flight data by counting the number of flights for each source that contains the letter 'o'. It then groups these counts by the flight sources and orders them in descending order based on the source names. The GO statement is used to separate batches of SQL commands for execution.***

***Description:***

***Write a query to display the flight source and the number of flights taking off from each source, if and only if the flight source contains 'o' in it. Give an alias name for the number of flights as 'NO\_OF\_FLIGHTS'.***

***Sort the records based on the flight source in descending order.***

***(Note: The column name should be written as in the schema diagram and an alias name should be written as in the description. Data is case sensitive.)***

***Use GO as the terminator, GO signals the end of a batch of Transact-SQL statements, in MS SQL Server.***

***Answer:***

***Select Flight\_source, count(\*) as NO\_OF\_FLIGHTS***

***From Flight\_details***

***Where Flight\_source like ‘%o%’***

***Group by flight\_source***

***Order by Flight\_source desc***

***Go***

***4..* *Description***

***Objective:***

***To work with the select statement with pivot.***

***Concept Explanation:***

***The SELECT statement is used to retrieve data from one or more tables in a database. It allows you to specify which columns you want to retrieve and apply filtering or sorting conditions.***

***The PIVOT operation can be used to transform rows of data into columns, providing a more structured and summarized view of the data.***

***Concept Implementation:***

***The PIVOT operator transforms data from the Payment table, specifically counting occurrences of different pay\_mode values ('Cash' and 'Card'). The PivotTable used to count each pay\_mode category ('Cash' and 'Card') across different pay\_date values.***

***Description:***

***Write a query that performs a pivot operation on the payment table to calculate the number of payments done using 'Cash' and 'Card' modes based on the pay date.***

***Use:Pivot***

***Sample Output:***

***pay\_date     Cash     Card***

***2024-02-10      2           1***

***2024-03-12      1            0***

***Use GO as the terminator, GO signals the end of a batch of Transact-SQL statements, in MS SQL Server.***

***Answer:***

***SELECT pay\_date, Cash, Card***

***FROM (***

***SELECT pay\_date, pay\_mode***

***FROM Payment***

***) AS SourceTable***

***PIVOT (***

***COUNT(pay\_mode)***

***FOR pay\_mode IN ([Cash], [Card])***

***) AS PivotTable***

***GO***

***5..* *Description***

***Objective:***

***To work with the select statement with the where clause, scalar function and join.***

***Concept Explanation:***

***The SELECT statement is used to retrieve data from one or more tables in a database. It also retrieves the particular data under specific conditions.***

***Concept Implementation:***

***The Select statement concatenates passenger names with their addresses for flights booked on weekends. It joins passengers, booking\_details, booking, and flight\_details tables based on IDs. The date function extracts the day of the week from flight\_date, filtering for weekends (Saturday and Sunday).***

***Description:***

***Write a query to display the details of all passengers who fly on the weekend, as shown in the sample output.***

***Sort the records based on the Passengers\_List in descending order.***

***(Hint: Saturday and Sunday are weekends for flight date.)***

***SAMPLE DATA:***

***Passengers\_List***

***pass\_name hails from address***

***SAMPLE OUTPUT:***

***Passengers\_List***

***Callie hails from San Diego***

***((Note: The column name should be written as in the schema diagram and an alias name should be written as in the description. Data is case sensitive.)***

***Use GO as the terminator, GO signals the end of a batch of Transact-SQL statements, in MS SQL Server.***

***Answer:***

***SELECT***

***CONCAT(passengers.pass\_name, ' hails from ', passengers.address) AS Passengers\_List***

***FROM passengers***

***JOIN booking\_details ON passengers.passenger\_id = booking\_details.passenger\_id***

***JOIN booking ON booking\_details.booking\_id = booking.booking\_id***

***JOIN flight\_details ON booking.flight\_id = flight\_details.flight\_id***

***WHERE DATEPART(WEEKDAY, flight\_details.flight\_date) IN (7, 1) -- 7 = Saturday, 1 = Sunday***

***ORDER BY Passengers\_List DESC***

***GO***

***6..* *Description***

***Objective:***

***To work with the select statement with the where clause, aggregate function, scalar function and join.***

***Concept Explanation:***

***The SELECT statement is used to retrieve data from one or more tables in a database. It also retrieves the particular data under specific conditions.***

***Concept Implementation:***

***The Select statement retrieves flight names, determines flight times based on months, and counts passengers for flights excluding Boston. It joins tables Flight\_details, booking, and booking\_details using their IDs. Results are grouped by flight details and filtered for passenger counts less than 2, ordered alphabetically by flight name.***

***Description:***

***Write a query to display the flight name, flight time and the number of passengers of all the flights that are onboarding other than 'Boston' with the total number of passengers being less than 2.***

***Give an alias name to the flight time and the number of passengers as 'Flight\_Time' and 'Passenger\_Count'.***

***Sort the details based on the flight name in ascending order.***

***Flight time is based on the flight date.***

* ***If flights are onboarding from January to June then the flight time is '09:00:00 AM'.***
* ***If flights are onboarding from July to December then the flight time is '09:00:00 PM'.***

***((Note: The column name should be written as in the schema diagram and an alias name should be written as in the description. Data is case sensitive.)***

***Use GO as the terminator, GO signals the end of a batch of Transact-SQL statements, in MS SQL Server.***

***Answers:***

***SELECT***

***flight\_details.FLIGHT\_NAME,***

***CASE***

***WHEN MONTH(flight\_details.FLIGHT\_DATE) BETWEEN 1 AND 6 THEN '09:00:00 AM'***

***ELSE '09:00:00 PM'***

***END AS Flight\_Time,***

***COUNT(booking\_details.passenger\_id) AS Passenger\_Count***

***FROM flight\_details***

***JOIN booking ON flight\_details.FLIGHT\_ID = booking.FLIGHT\_ID***

***JOIN booking\_details ON booking.BOOKING\_ID = booking\_details.BOOKING\_ID***

***WHERE flight\_details.FLIGHT\_SOURCE <> 'Boston'***

***GROUP BY flight\_details.FLIGHT\_NAME, flight\_details.FLIGHT\_DATE***

***HAVING COUNT(booking\_details.passenger\_id) < 2***

***ORDER BY flight\_details.FLIGHT\_NAME ASC***

***GO***

***7..* *Description***

***Objective:***

***To work with the select statement with rank function.***

***Concept Explanation:***

***The SELECT statement is used to retrieve data from one or more tables in a database. The RANK() function is a window function that assigns a rank to each row within a partition of a result set based on the specified ordering criteria.***

***Concept Implementation:***

***The SELECT statement retrieves car details (CARNAME, MODEL, SALEUNITS) and assigns a ranking (SCORE) for cars based on the descending order of SALEUNITS. It filters sales data to include only April 2020 records, presenting a ranked view of car sales for that month and year.***

***Description:***

***Use the following Schema table for the query.***

***Table Name : CAR\_SALES***

|  |  |
| --- | --- |
| ***Attribute*** | ***Datatype*** |
| ***ID*** | ***Int*** |
| ***CARNAME*** | ***Varchar(50)*** |
| ***MODEL*** | ***Varchar(50)*** |
| ***DATEOFSALES*** | ***Date*** |
| ***SALEUNITS*** | ***Int*** |
| ***UNITPRICE*** | ***Decimal(7,2)*** |

***Display the car name, model, sale units and score in April'2020. Rank the cars and models based on the sale units by naming it as SCORE.***

***(Hint:   Use a RANK function.)***

***Sample output is shown for representation purpose only.***

|  |  |  |  |
| --- | --- | --- | --- |
| ***CARNAME*** | ***MODEL*** | ***SALEUNITS*** | ***SCORE*** |
| ***Porsche*** | ***928*** | ***791*** | ***1*** |
| ***Mitsubishi*** | ***Outlander*** | ***540*** | ***2*** |
| ***Oldsmobile*** | ***Bravada*** | ***480*** | ***3*** |

***(Note: The column name should be written as in the schema diagram and an alias name should be written as in the description. Data is case sensitive.)***

***Use GO as the terminator, GO signals the end of a batch of Transact-SQL statements, in MS SQL Server.***

***Answer:***

***SELECT***

***CARNAME,***

***MODEL,***

***SALEUNITS,***

***RANK() OVER (ORDER BY SALEUNITS DESC) AS SCORE***

***FROM CAR\_SALES***

***WHERE DATEOFSALES BETWEEN '2020-04-01' AND '2020-04-30'***

***GO***

***8…Description***

***Objective:***

***To work with the select statement with CTE and join.***

***Concept Explanation:***

***1. The SELECT statement is used to retrieve data from one or more tables in a database. It allows you to specify which columns you want to retrieve and apply filtering or sorting conditions.***

***2. CTEs are temporary result sets that you can define within a query and reuse multiple times. They help simplify complex queries, improve readability, and avoid code duplication. CTEs are defined using the WITH clause.***

***3. Joins are used to combine rows from two or more tables based on a related column between them. Common types of joins include INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL JOIN.***

***Concept Implementation:***

***A common table expression (CTE) named Flight\_Booking calculates the total amount(Total\_Amount) for each flight (Flight\_name) by joining the Flight\_details and booking tables. Then, it selects the top 5 flight names (Flight\_name) from the Flight\_Booking CTE, ordering them by the total amount (Total\_Amount) in descending order.***

***Description:  
  
Define a CTE named 'Flight\_Booking' which has 2 columns listed below: flight name and Total\_Amount. This common table expression 'Flight\_Booking' selects the flight's name and the total amount for each flight.  
Write a query using CTE to display top 5 flight name who have highest Total\_Amount.***

***((Note: The column name should be written as in the schema diagram and an alias name should be written as in the description. Data is case sensitive.)***

***Use GO as the terminator, GO signals the end of a batch of Transact-SQL statements, in MS SQL Server.***

***Answer:***

***WITH Flight\_Booking AS (***

***SELECT***

***flight\_details.FLIGHT\_NAME,***

***SUM(booking.AMOUNT) AS Total\_Amount***

***FROM flight\_details***

***JOIN booking ON flight\_details.FLIGHT\_ID = booking.FLIGHT\_ID***

***GROUP BY flight\_details.FLIGHT\_NAME***

***)***

***SELECT TOP 5 FLIGHT\_NAME, Total\_Amount***

***FROM Flight\_Booking***

***ORDER BY Total\_Amount DESC***

***GO***

***9...* *Description***

***Objective:***

***To work with the merge statement.***

***Concept Explanation:***

***The MERGE statement is used to perform multiple operations (INSERT, UPDATE, DELETE) in a single statement based on a specified condition. It's particularly useful for synchronizing data between two tables or merging changes from a source table into a target table***

***Concept Implementation:***

***A merge operation between two tables (updated\_doctor and doctor) based on matching doctor\_id values. It updates updated\_doctor records if the dr\_last\_name differs between the tables. If there's no match in updated\_doctor, it inserts a new record from the Doctor. The OUTPUT clause captures the action taken ($action), along with relevant columns, from the affected records after the merge operation.***

***Description:***

***Write a query using a simple MERGE statement to perform UPDATE and INSERT operations on the doctor and updated\_doctor tables. This merge statement does the following actions:***

***1. The doctor table acts as the source, and updated\_doctor acts as the target.(Updated\_doctor and doctor table will have same table structure.)***

***2. Both tables are merged using the doctor\_id attribute.***

***3. When doctor\_id gets matched in both tables and if dr\_last\_name differs, the UPDATE dr\_last\_name of target is updated with the source record, and the result is displayed with the action name as action, source\_doctor\_id, source\_dr\_first\_name, source\_dr\_middle\_name, and source\_dr\_last\_name.***

***4. When the doctor\_id does not exist in the target table, then INSERT source\_doctor\_id, source\_dr\_first\_name, source\_dr\_middle\_name, and source\_dr\_last\_name of the source record into the target table, and the result is displayed with the action name as action, source\_doctor\_id, source\_dr\_first\_name, source\_dr\_middle\_name, and source\_dr\_last\_name.***

***Hint: Use the OUTPUT clause that returns one of three values for each row: 'INSERT', 'UPDATE', or 'DELETE', according to the action that was performed on that row.***

***Use GO as the terminator, GO signals the end of a batch of Transact-SQL statements, in MS SQL Server.***

***Answer:***

***MERGE INTO updated\_doctor AS target***

***USING doctor AS source***

***ON target.doctor\_id = source.doctor\_id***

***WHEN MATCHED AND target.dr\_last\_name <> source.dr\_last\_name***

***THEN UPDATE SET target.dr\_last\_name = source.dr\_last\_name***

***WHEN NOT MATCHED BY TARGET***

***THEN INSERT (doctor\_id, dr\_first\_name, dr\_middle\_name, dr\_last\_name)***

***VALUES (source.doctor\_id, source.dr\_first\_name, source.dr\_middle\_name, source.dr\_last\_name)***

***OUTPUT $action AS action,***

***source.doctor\_id AS source\_doctor\_id,***

***source.dr\_first\_name AS source\_dr\_first\_name,***

***source.dr\_middle\_name AS source\_dr\_middle\_name,***

***source.dr\_last\_name AS source\_dr\_last\_name;***

***GO***

***10….* *Description***

***Objective:***

***To work with the view.***

***Concept Explanation:***

***A view is a virtual table that is based on the result set of a SELECT query. It doesn't store data itself but provides a way to present data from one or more tables in a structured format.***

***Concept Implementation:***

***A view named Flight\_Booking combines data from the flight\_details and booking tables. The view includes columns for Flight\_id, Flight\_name, Flight\_source, Flight\_destination, Booking\_id, and Booking\_date, providing a consolidated view of flight and booking information. This view simplifies querying by presenting relevant data from multiple tables in a single virtual table-like structure.***

***Description:  
Write a query to create a view named 'Flight\_Booking' with the columns flight id, flight name, flight source, flight destination, booking id and booking date of all the flights.***

***(Note: The column name should be written as in the schema diagram and an alias name should be written as in the description. Data is case sensitive.)***

***Use GO as the terminator, GO signals the end of a batch of Transact-SQL statements, in MS SQL Server.***

***Answer:***

***CREATE VIEW Flight\_Booking AS***

***SELECT***

***flight\_details.FLIGHT\_ID,***

***flight\_details.FLIGHT\_NAME,***

***flight\_details.FLIGHT\_SOURCE,***

***flight\_details.FLIGHT\_DESTINATION,***

***booking.BOOKING\_ID,***

***booking.BOOKING\_DATE***

***FROM flight\_details***

***JOIN booking ON flight\_details.FLIGHT\_ID = booking.FLIGHT\_ID***

***GO***

***11…* *Description***

***Objective:***

***To work with the view.***

***Concept Explanation:***

***A view is a virtual table that is based on the result set of a SELECT query. It doesn't store data itself but provides a way to present data from one or more tables in a structured format.***

***Concept Implementation:***

***The DROP VIEW statement is used to delete the view named pass\_details from the database.***

***Description:***

***Write a query to remove the view named 'pass\_details' that is already created.***

***Use GO as the terminator, GO signals the end of a batch of Transact-SQL statements, in MS SQL Server.***

***Answer:*** ***DROP VIEW pass\_details***

***GO***

***12….* *Description***

***Objective:***

***To work with the function.***

***Concept Explanation:***

***Functions are reusable blocks of code that perform specific tasks or calculations. It encapsulates logic and can accept inputs (arguments) and produce outputs (return values). The functions can be used to manipulate data, perform calculations, or return specific results based on given parameters.***

***Concept Implementation:***

***A function named flight\_update that returns flight information. It inserts data into the @flight\_info table variable based on the flight\_details table and the UPDATE statement attempts to modify the flight\_name column in the @flight\_info table variable. Specifically, it sets the flight\_name to 'Qatar Airways' for rows where the flight\_destination is 'Chicago' and returns a table variable as a result.***

***Description:***

***Write a user-defined, multi-statement, table-valued function named 'flight\_update' that returns a table variable as a result of an action performed by the function 'flight\_update'.***

***In this function logic, try to insert the flight name, flight source and flight destination columns only into the defined table variable and then update the flight name to 'Qatar Airways' for the flight destination 'Chicago'.***

***Hints :***

***1. Data is case sensitive.***

***2. Table variables must be explicitly declared and defined.***

***3. Column name and number of columns in the table variable should be the same as that of the inserting columns.***

***Use GO as the terminator, GO signals the end of a batch of Transact-SQL statements, in MS SQL Server.***

***Answer:***

***CREATE FUNCTION flight\_update()***

***RETURNS @flight\_info TABLE (***

***flight\_name NVARCHAR(100),***

***flight\_source NVARCHAR(100),***

***flight\_destination NVARCHAR(100)***

***)***

***AS***

***BEGIN***

***-- Insert data into the table variable from the flight\_details table***

***INSERT INTO @flight\_info (flight\_name, flight\_source, flight\_destination)***

***SELECT flight\_name, flight\_source, flight\_destination***

***FROM flight\_details;***

***-- Update the flight\_name to 'Qatar Airways' for rows where flight\_destination is 'Chicago'***

***UPDATE @flight\_info***

***SET flight\_name = 'Qatar Airways'***

***WHERE flight\_destination = 'Chicago';***

***-- Return the table variable***

***RETURN;***

***END;***

***GO***

***13…* *Description***

***Objective:***

***To work with the function.***

***Concept Explanation:***

***Functions are reusable blocks of code that perform specific tasks or calculations. It encapsulates logic and can accept inputs (arguments) and produce outputs (return values). The functions can be used to manipulate data, perform calculations, or return specific results based on given parameters.***

***Concept Implementation:***

***A function named FlightDeck returns a result set containing all columns from the Booking\_details table. The function uses the RETURN statement to return the result set obtained from selecting all columns from the Booking\_details table.***

***Description:***

***Write a user-defined inline table-valued function named 'FlightDeck' that returns the table variable as a result of an action performed by the FlightDeck function and this function should return the whole table Booking\_Details.***

***Use GO as the terminator, GO signals the end of a batch of Transact-SQL statements, in MS SQL Server.***

***Answer:***

***CREATE FUNCTION FlightDeck()***

***RETURNS TABLE***

***AS***

***RETURN***

***(***

***SELECT \****

***FROM Booking\_Details***

***);***

***GO***

***14…* *Description***

***Objective:***

***To work with the procedures.***

***Concept Explanation:***

***Stored procedures are precompiled sets of SQL statements stored in a database. They encapsulate logic to perform specific tasks like querying data or updating record***

***Concept Implementation:***

***The stored procedure named bookingAmount takes a flight ID as input and calculates the total booking amount for that flight. The @total\_amount parameter is an output variable that holds the calculated sum of amounts from the booking table based on the specified flight ID and adjusts it by adding 10% to the total amount.***

***Description:***

***Create a stored procedure named bookingAmount, that takes the flight Id as input and returns the total amount booked for the given flight id. Then, it adjusts the total amount by adding 10% to the total amount.***

***The requirement is as given below:***

***Procedure Name: bookingAmount***

***Input Parameter: flightId (as varchar(20))***

***Output Parameter:total\_amount (as int)***

***Use GO as the terminator, GO signals the end of a batch of Transact-SQL statements, in MS SQL Server.***

***Answer:***

***CREATE PROCEDURE bookingAmount***

***@flightId VARCHAR(20),***

***@total\_amount INT OUTPUT***

***AS***

***BEGIN***

***-- Calculate the total booking amount for the given flight ID***

***SELECT @total\_amount = SUM(amount)***

***FROM booking***

***WHERE flight\_id = @flightId;***

***-- Adjust the total amount by adding 10%***

***SET @total\_amount = @total\_amount + (@total\_amount \* 0.10);***

***END;***

***GO***

***15…* *Description***

***Objective:***

***To work with the procedure using a cursor.***

***Concept Explanation:***

***Stored procedures are precompiled sets of SQL statements stored in a database. They encapsulate logic to perform specific tasks like querying data or updating record***

***Concept Implementation:***

***The passengerDetails stored procedure retrieves passenger names based on a given address as input parameter. It uses a cursor to iterate through passenger names matching the provided address and print them in upper case. The cursor fetches data until there are no more matching records.***

***Description:***

***Create a user-defined stored procedure named passengerDetails that accepts the address as input and prints the passenger name from the given address as given in the sample output.***

***Print the passenger’s name in uppercase.***

***Sort the records based on the passenger name in descending order.***

***The requirement is as given below:***

***Procedure  Name: passengerDetails***

***Input Parameter: add varchar(50)***

***Output Parameter: NA***

***Write the code to create the procedure.***

***Sample Input:***

***passengerDetails('Boston');***

***Sample Output:***

***Passengers names are:  
EDWIN  
RIYA  
NEHA***

***Use GO as the terminator, GO signals the end of a batch of Transact-SQL statements, in MS SQL Server.***

***Answer:***

***CREATE PROCEDURE passengerDetails***

***@add VARCHAR(50)***

***AS***

***BEGIN***

***DECLARE @passengerName VARCHAR(100);***

***-- Declare a cursor to select passenger names based on the given address***

***DECLARE passenger\_cursor CURSOR FOR***

***SELECT UPPER(passenger\_name)***

***FROM passengers***

***WHERE address = @add***

***ORDER BY passenger\_name DESC;***

***-- Open the cursor***

***OPEN passenger\_cursor;***

***-- Fetch the first row***

***FETCH NEXT FROM passenger\_cursor INTO @passengerName;***

***PRINT 'Passengers names are:';***

***-- Loop through the cursor and fetch each row***

***WHILE @@FETCH\_STATUS = 0***

***BEGIN***

***-- Print the passenger name in uppercase***

***PRINT @passengerName;***

***-- Fetch the next row***

***FETCH NEXT FROM passenger\_cursor INTO @passengerName;***

***END;***

***-- Close and deallocate the cursor***

***CLOSE passenger\_cursor;***

***DEALLOCATE passenger\_cursor;***

***END;***

***GO***

***16….***

***Description***

***Objective:***

***To work with the procedure using a cursor.***

***Concept Explanation:***

***Stored procedures are precompiled sets of SQL statements stored in a database. They encapsulate logic to perform specific tasks like querying data or updating record***

***Concept Implementation:***

***The findBookings stored procedure retrieves booking details based on a provided booking ID. It uses a cursor to iterate through passenger names, booking dates, and flight dates associated with the given booking ID, calculating the days between booking and flight. If no bookings are found, it prints 'No bookings made'. Cursors should be used judiciously due to potential performance impacts on large datasets.***

***Description:***

***Create a user-defined stored procedure named findBookings that accepts the booking Id as input and prints the passenger name and total days as given in the sample output. If no such booking id is present, print "No bookings made".  Use cursor.***

***Sort the records based on the passenger name in descending order.***

***The requirement is as given below:***

***Procedure  Name: findBookings***

***Input Parameter: nBooking\_Id varchar(10)***

***Write the code to create the procedure.***

***Sample  Input:***

***findBookings(B861);***

***Sample Data:***

***Passenger Name purchased the flight ticket before total days***

***Sample  Output:***

***Donna purchased the flight ticket before 230 days***

***(Total days : is the number of days between the booking date and the flight date.)***

***(Hint:Use the Cast function. Data is case sensitive. Use Cursor.)***

***Use GO as the terminator, GO signals the end of a batch of Transact-SQL statements, in MS SQL Server.***

***Answer:*** ***CREATE PROCEDURE findBookings***

***@nBooking\_Id VARCHAR(10)***

***AS***

***BEGIN***

***DECLARE @passengerName VARCHAR(100);***

***DECLARE @bookingDate DATE;***

***DECLARE @flightDate DATE;***

***DECLARE @totalDays INT;***

***-- Declare a cursor to select passenger names, booking dates, and flight dates based on the given booking ID***

***DECLARE booking\_cursor CURSOR FOR***

***SELECT passenger\_name, booking\_date, flight\_date***

***FROM bookings***

***WHERE booking\_id = @nBooking\_Id***

***ORDER BY passenger\_name DESC;***

***-- Open the cursor***

***OPEN booking\_cursor;***

***-- Fetch the first row***

***FETCH NEXT FROM booking\_cursor INTO @passengerName, @bookingDate, @flightDate;***

***-- Check if any rows were returned***

***IF @@FETCH\_STATUS = -1***

***BEGIN***

***PRINT 'No bookings made';***

***RETURN;***

***END***

***-- Loop through the cursor and fetch each row***

***WHILE @@FETCH\_STATUS = 0***

***BEGIN***

***-- Calculate the total days between booking date and flight date***

***SET @totalDays = DATEDIFF(DAY, @bookingDate, @flightDate);***

***-- Print the passenger name and total days***

***PRINT @passengerName + ' purchased the flight ticket before ' + CAST(@totalDays AS VARCHAR) + ' days';***

***-- Fetch the next row***

***FETCH NEXT FROM booking\_cursor INTO @passengerName, @bookingDate, @flightDate;***

***END;***

***-- Close and deallocate the cursor***

***CLOSE booking\_cursor;***

***DEALLOCATE booking\_cursor;***

***END;***

***GO***

* ***17…***

***March Booking - CursorEarned XP : -/3***

* ***73%***

***March Booking - Cursor***

***Description***

***Objective:***

***To work with the cursor.***

***Concept Explanation:***

***A cursor is a database object that allows you to retrieve and manipulate data one row at a time. It's often used when you need to perform operations that are row-based***

***Concept Implementation:***

***The SQL query uses a cursor to fetch and print passenger names, flight names, and booking dates for bookings made in March. The cursor iterates through the result set obtained from joining passengers, booking\_details, booking, and flight\_details tables based on the condition of booking dates in March. Cursors should be used judiciously due to potential performance impacts on large datasets.***

***Description:***

***Refer to the given schema. Create a cursor that feeds data to a report.***

***Display the passenger name, the flight name, and the booking date of all the flights that were booked in the month of March using cursors.***

***Please refer to the sample output:***

***Emma,Qatar Airways,2023-03-20***

***Jeba,Qatar Airways,2019-03-20***

***Hint:***

***Use the cast function. Data is case sensitive.***

***Use GO as the terminator, GO signals the end of a batch of Transact-SQL statements, in MS SQL Server.***

***Answer:***

***-- Declare the cursor***

***DECLARE @PassengerName NVARCHAR(100), @FlightName NVARCHAR(100), @BookingDate DATE;***

***DECLARE MarchBookingCursor CURSOR FOR***

***SELECT p.PassengerName, f.FlightName, b.BookingDate***

***FROM passengers p***

***JOIN booking\_details bd ON p.PassengerID = bd.PassengerID***

***JOIN booking b ON bd.BookingID = b.BookingID***

***JOIN flight\_details f ON b.FlightID = f.FlightID***

***WHERE MONTH(b.BookingDate) = 3;***

***-- Open the cursor***

***OPEN MarchBookingCursor;***

***-- Fetch the first row***

***FETCH NEXT FROM MarchBookingCursor INTO @PassengerName, @FlightName, @BookingDate;***

***-- Loop through the result set***

***WHILE @@FETCH\_STATUS = 0***

***BEGIN***

***-- Print the current row***

***PRINT @PassengerName + ',' + @FlightName + ',' + CAST(@BookingDate AS NVARCHAR(10));***

***-- Fetch the next row***

***FETCH NEXT FROM MarchBookingCursor INTO @PassengerName, @FlightName, @BookingDate;***

***END;***

***-- Close and deallocate the cursor***

***CLOSE MarchBookingCursor;***

***DEALLOCATE MarchBookingCursor;***

***GO***

***18…* *Description***

***Objective:***

***To work with the trigger.***

***Concept Explanation:***

***Triggers are database objects used to automatically perform actions, such as inserting, updating, or deleting data in response to specified events, like before or after an operation on a table.***

***Concept Implementation:***

***The after\_insert\_passengers trigger fires after inserting a new record into the passengers table. It calculates the total passenger count and updates it with a descriptive message. Subsequently, it inserts an audit record into the passenger\_audit table, capturing the updated passenger count and the action performed.***

***Description:***

***Create a trigger named 'after\_insert\_passengers' on the passengers table that is triggered whenever a new record is inserted into the passengers table. This trigger will store the number of passengers currently available and action into the audit\_action attribute of the 'passenger\_audit' table, after inserting the new record into the passengers table. The action name in the affected log table passenger\_audit is 'A new record has been added'.***

***Sample Output format:***

|  |  |
| --- | --- |
| ***passenger\_count*** | ***audit\_action*** |
| ***12 passengers are present at this time*** | ***A new record has been added*** |

***The table structure for the passenger\_audit:***

|  |  |  |
| --- | --- | --- |
| ***column\_name*** | ***data type*** | ***constraint*** |
| ***passenger\_count*** | ***varchar(100)*** | ***not null*** |
| ***audit\_action*** | ***varchar(100)*** | ***not null*** |

***(Hint: Data is case sensitive. Assume that the 'passenger\_audit' table has been already created.)***

***Use GO as the terminator, GO signals the end of a batch of Transact-SQL statements, in MS SQL Server.***

***Answer;***

***CREATE TRIGGER after\_insert\_passengers***

***ON passengers***

***AFTER INSERT***

***AS***

***BEGIN***

***DECLARE @passenger\_count INT;***

***DECLARE @audit\_action VARCHAR(100);***

***-- Calculate the total number of passengers***

***SELECT @passenger\_count = COUNT(\*) FROM passengers;***

***-- Create the descriptive message***

***SET @audit\_action = CAST(@passenger\_count AS VARCHAR(100)) + ' passengers are present at this time';***

***-- Insert the audit record***

***INSERT INTO passenger\_audit (passenger\_count, audit\_action)***

***VALUES (@audit\_action, 'A new record has been added');***

***END;***

***GO***

***19…* *Description***

***Objective:***

***To work with the trigger.***

***Concept Explanation:***

***Triggers are database objects used to automatically perform actions, such as inserting, updating, or deleting data in response to specified events, like before or after an operation on a table.***

***Concept Implementation:***

***The trigger named trigger\_flight\_update on the flight\_details table fires after an update operation. It retrieves the updated flight\_id and flight\_name from the inserted pseudo table and checks if the flight\_seat column was updated. If so, it sets the variable to 'Updated Flight Seat' and inserts a corresponding record into the flight\_log\_history table.***

***Description:***

***Create a trigger named 'trigger\_flight\_update' that is triggered whenever the flight table is updated.***

***This trigger will insert the flight id, flight name and action into the table 'flight\_log\_history' after the updation of the flight seat in the flight\_details table.***

***The action name in the affected log table flight\_log\_history is 'Updated Flight Seat'.***

***Trigger name : trigger\_flight\_update***

***Table name : flight\_log\_history***

***Field names : flight\_id, flight\_name, action***

***Action  : 'Updated Flight Seat'.***

***The table structure of flight\_log\_history is as follows:***

|  |  |
| --- | --- |
| ***Column name*** | ***Data type*** |
| ***Flight\_id*** | ***varchar(20)*** |
| ***Flight\_name*** | ***varchar(20)*** |
| ***action*** | ***varchar(100)*** |

***Sample Output:***

|  |  |  |
| --- | --- | --- |
| ***Flight\_id*** | ***Flight\_name*** | ***action*** |
| ***F20*** | ***Jetblue Airlines*** | ***Updated Flight Seat*** |

***(Hint: Data is case sensitive. Assume that the 'flight\_log\_history' table has been already created.)***

***Use GO as the terminator, GO signals the end of a batch of Transact-SQL statements, in MS SQL Server.***

***Answer:***

***20….***

***Description***

***Objective:***

***To work with the function with the exception handling.***

***Concept Explanation:***

1. ***Functions are reusable blocks of code that perform specific tasks or calculations. It encapsulates logic and can accept inputs (arguments) and produce outputs (return values). The functions can be used to manipulate data, perform calculations, or return specific results based on given parameters.***
2. ***Exception handling deals with managing and responding to unexpected or exceptional situations that may occur during the execution of a query.***

***Concept Implementation:***

***A function named find\_passenger\_detail that takes a @passid parameter and returns a varchar(60) value. The function retrieves the passenger's name and booking date by joining the booking, booking\_details, and passengers tables based on the provided passenger ID. If no records are found for the given passenger ID, the function returns 'Passenger Not Available'. Otherwise, it returns the concatenated string of the passenger's name and bookingdate.***

***Description:***

***Write an SQL function named Find\_passenger\_detail() that consumes the passenger id varchar(50) as input and returns the varchar(60) for passenger name along with the booking date concatenated with a space between them. If a passenger doesn't exist for the input id, then it returns an error message as 'Passenger Not Available'.***

***Sample Output:***

***John booked on 2020-04-21***

***Use GO as the terminator, GO signals the end of a batch of Transact-SQL statements, in MS SQL Server.***

***Answer:***

***CREATE FUNCTION Find\_passenger\_detail(@passid VARCHAR(50))***

***RETURNS VARCHAR(60)***

***AS***

***BEGIN***

***DECLARE @result VARCHAR(60);***

***SELECT @result = p.name + ' booked on ' + CONVERT(VARCHAR, b.bookingdate, 120)***

***FROM booking b***

***JOIN booking\_details bd ON b.bookingid = bd.bookingid***

***JOIN passengers p ON bd.passengerid = p.passengerid***

***WHERE p.passengerid = @passid;***

***IF @result IS NULL***

***BEGIN***

***SET @result = 'Passenger Not Available';***

***END***

***RETURN @result;***

***END;***

***GO***

***21…Description***

***Objective:***

***To work with the procedure with exception handling.***

***Concept Explanation:***

***Exception handling deals with managing and responding to unexpected or exceptional situations that may occur during the execution of a query.***

***Concept Implementation:***

***A stored procedure named delete\_Flight\_Details that accepts a @flightid parameter. It attempts to delete records from the flight\_Details table where the flight\_id matches the input parameter. If the deletion is unsuccessful due to child records existing in the Booking table, it catches the error and inserts an error message into the flight\_error\_log table. The BEGIN TRY and BEGIN CATCH blocks are used for error handling, enhancing the robustness of the procedure.***

***Description:***

***Create a procedure named 'delete\_Flight\_Details' that accepts one input named flightid of type varchar(20). This procedure is used to do the following operations:***

***1. If the flightid (i.e., flight\_id) passed as input does not have any child records in the booking table, then delete the flight with the given flight id.***

***2. If the flightid (i.e., flight\_id) passed as input has any child records available in the booking table, then insert an error\_msg of type varchar(200), 'Deletion failed. The flight cannot be removed because it has existing bookings' into the flight\_error\_log table.***

***USE: TRY...CATCH BLOCK***

***The table structure of flight\_error\_log is as follows:***

|  |  |
| --- | --- |
| ***Column name*** | ***Data type*** |
| ***Error\_msg*** | ***varchar(200)*** |

***(Hint: Assume that the 'flight\_error\_log' table has already been created.)***

***Use GO as the terminator, GO signals the end of a batch of Transact-SQL statements, in MS SQL Server.***

***CREATE PROCEDURE delete\_Flight\_Details***

***@flightid VARCHAR(20)***

***AS***

***BEGIN***

***BEGIN TRY***

***-- Check if there are any child records in the booking table***

***IF NOT EXISTS (SELECT 1 FROM booking WHERE flightid = @flightid)***

***BEGIN***

***-- Delete the flight details***

***DELETE FROM flight\_details WHERE flightid = @flightid;***

***-- Optionally, delete the flight itself if needed***

***DELETE FROM flights WHERE flightid = @flightid;***

***PRINT 'Flight details deleted successfully.';***

***END***

***ELSE***

***BEGIN***

***-- Insert error message into flight\_error\_log table***

***INSERT INTO flight\_error\_log (Error\_msg)***

***VALUES ('Deletion failed. The flight cannot be removed because it has existing bookings');***

***END***

***END TRY***

***BEGIN CATCH***

***-- Handle any unexpected errors***

***INSERT INTO flight\_error\_log (Error\_msg)***

***VALUES ('An unexpected error occurred: ' + ERROR\_MESSAGE());***

***END CATCH***

***END;***

***GO***

***22…***

***Description***

***Objective:***

***To work with the procedure and transaction control language.***

***Concept Explanation:***

1. ***Stored procedures are precompiled sets of SQL statements stored in a database. They encapsulate logic to perform specific tasks like querying data or updating records.***
2. ***Transaction Control Language (TCL) in SQL is used to manage transactions within a database. Transactions are units of work that involve one or more SQL operations, and TCL statements help control the outcome of transactions.***

***Concept Implementation:***

***The stored procedure UpdatePassengerInfo starts a transaction to ensure that the data updates are consistent. First, it updates the email of a passenger with ID 'P4' to 'welcome@hotmail.com'. Then, it creates a savepoint named SavePoint1 before proceeding with updating another passenger's email to 'hello@hotmail.com' with ID 'P40'. If no rows are affected during this second update, it rolls back to the savepoint SavePoint1 to maintain data integrity. However, if rows are updated, it commits the transaction successfully and prints a confirmation message indicating the successful transaction commit.***

***Description:***

***Create a stored procedure named updatePassengerInfo to update the email addresses of passengers. It operates in a transactional context to ensure data integrity.***

* ***The procedure should begin by initiating a transaction using BEGIN TRANSACTION.***
* ***The transaction attempts to update the email address of a passenger with the ID 'P4' to 'welcome@hotmail.com' using an UPDATE statement.***
* ***A savepoint named SavePoint1 is set using SAVE TRANSACTION. This savepoint allows for the creation of a point in the transaction that can be rolled back to if needed.***
* ***Next, another UPDATE statement is executed to update the email address of a passenger with the ID 'P40' to 'hello@hotmail.com'.***
* ***After the second, UPDATE statement, the procedure checks the @@ROWCOUNT system variable to determine if any rows were affected by the update. If no rows were updated (indicating that there was no passenger with the ID 'P40'), the transaction is rolled back to the savepoint SavePoint1 and it should print 'The rollback to SavePoint1 was initiated as there were no changes made to any rows.'. Otherwise, if at least one row was updated, the transaction is committed, and print 'The completion of the transaction was successful.'.***

***Use GO as the terminator, GO signals the end of a batch of Transact-SQL statements, in MS SQL Server.***

***Answer:***

***CREATE PROCEDURE updatePassengerInfo***

***AS***

***BEGIN***

***BEGIN TRANSACTION;***

***-- Update email for passenger with ID 'P4'***

***UPDATE passengers***

***SET email = 'welcome@hotmail.com'***

***WHERE passengerid = 'P4';***

***-- Create a savepoint***

***SAVE TRANSACTION SavePoint1;***

***-- Update email for passenger with ID 'P40'***

***UPDATE passengers***

***SET email = 'hello@hotmail.com'***

***WHERE passengerid = 'P40';***

***-- Check if any rows were affected***

***IF @@ROWCOUNT = 0***

***BEGIN***

***-- Rollback to savepoint if no rows were updated***

***ROLLBACK TRANSACTION SavePoint1;***

***PRINT 'The rollback to SavePoint1 was initiated as there were no changes made to any rows.';***

***END***

***ELSE***

***BEGIN***

***-- Commit the transaction if rows were updated***

***COMMIT TRANSACTION;***

***PRINT 'The completion of the transaction was successful.';***

***END***

***END;***

***GO***