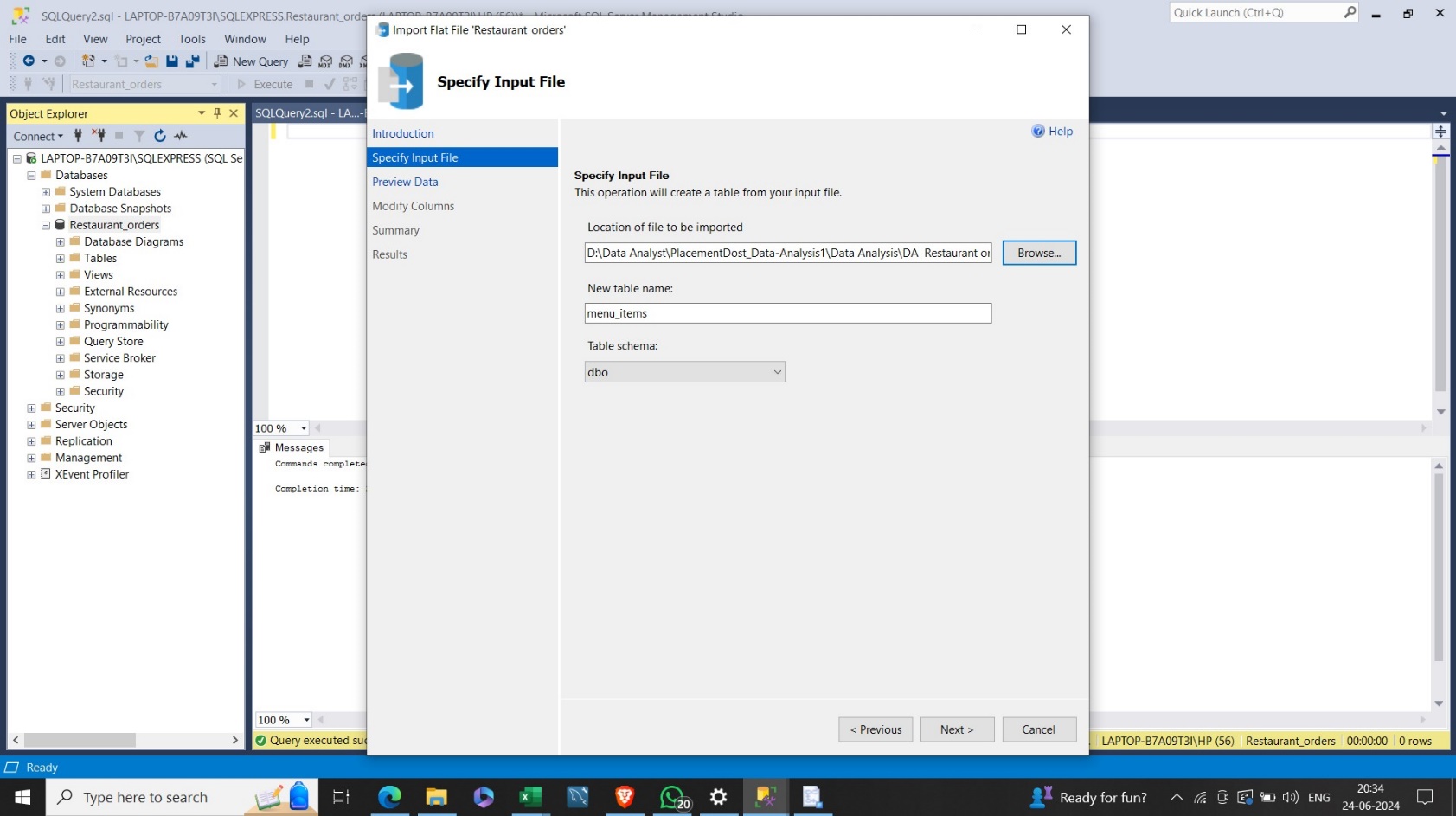
**RESTAURANT ORDERS ANALYSIS**

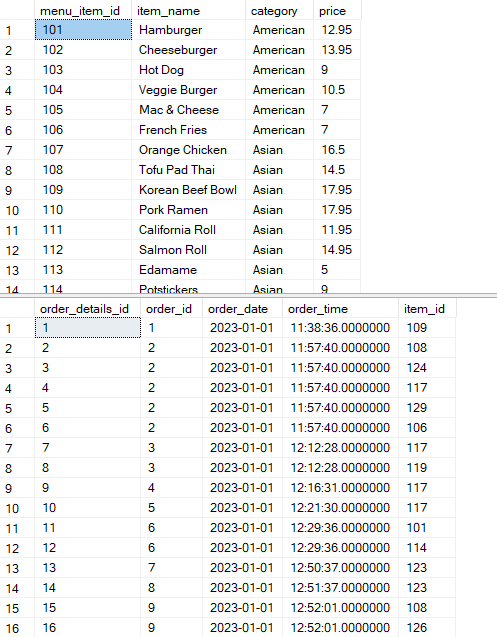
1. Convert Dataset to SQL Database: -

* Create SQL statements to define and populate tables for menu\_details and order\_details using the provided dataset.

Solution: - We have used the inbuilt feature of importing csv file in SSMS as it was not possible to insert every value one by one

Output:



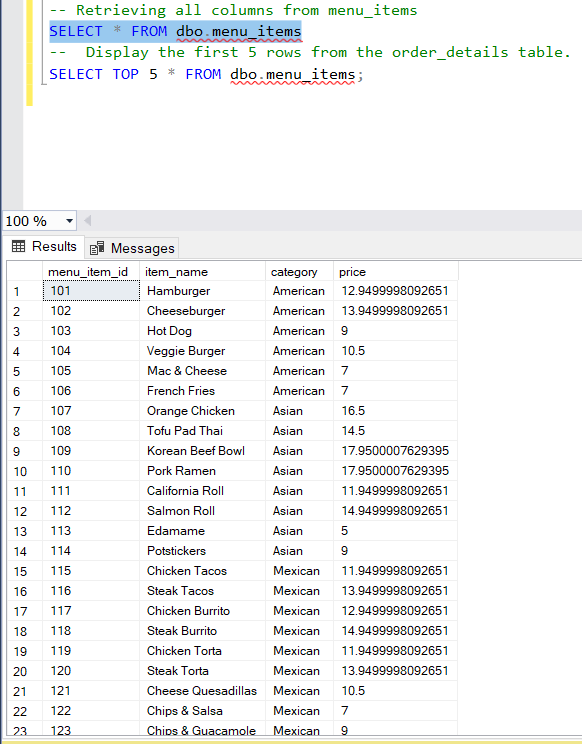


1. Basic SELECT Queries: -

* Retrieve all columns from the menu\_items table. –

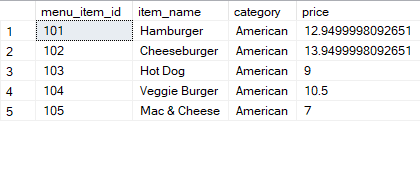
Output:-

Query : SELECT \* FROM dbo.menu\_items



* Display the first 5 rows from the order\_details table.

Solution : SELECT TOP 5 \* FROM dbo.menu\_items;



1. Filtering and Sorting: -

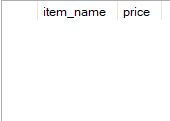
* Select the item\_name and price columns for items in the 'Main Course' category.
* Sort the result by price in descending order.

Solution: SELECT item\_name, price FROM dbo.menu\_items

WHERE category= 'Main Course'

ORDER BY price DESC

There is no ‘Main Course’ Category there for result is blank



1. Aggregate Functions: -

* Calculate the average price of menu items.

Solution: SELECT AVG(price) as avg\_price FROM dbo.menu\_items



* Find the total number of orders placed.

Solution: SELECT COUNT(DISTINCT(order\_details\_id)) as

total\_orders FROM order\_details



1. Joins: -

* Retrieve the item\_name, order\_date, and order\_time for all items in the order\_details table, including their respective menu item details

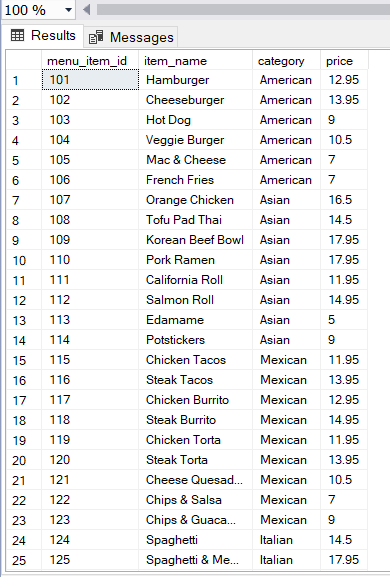
Solution:

SELECT item\_name, order\_date, order\_time,menu.menu\_item\_id as item\_id, category,price FROM order\_details AS orders

LEFT OUTER JOIN menu\_items AS menu

ON menu.menu\_item\_id= orders.item\_id

;



1. Subqueries: - List the menu items (item\_name) with a price greater than the average price of all menu items

Solution: SELECT item\_name FROM menu\_items

WHERE price> (SELECT AVG(price) FROM menu\_items);



1. Date and Time Functions

* Extract the month from the order\_date and count the number of orders placed in each month.

Solution:

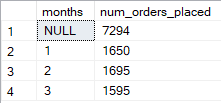
SELECT MONTH(order\_date) AS months,

COUNT(order\_details\_id) AS num\_orders\_placed

FROM order\_details

GROUP BY MONTH(order\_date)

ORDER BY 1;



1. Group By and Having

* Show the categories with the average price greater than $15.

Solution: SELECT category,AVG(price) AS avg\_price

FROM menu\_items

GROUP BY category

HAVING AVG(price)>15



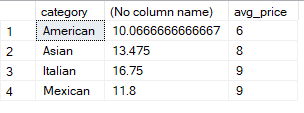
* Include the count of items in each category.

Solution:

SELECT category,AVG(price),COUNT(category) AS avg\_price

FROM menu\_items

GROUP BY category



1. Conditional Statements: -

* Display the item\_name and price, and indicate if the item is priced above $20 with a new column named 'Expensive'.

Solution:

SELECT item\_name,price,

CASE

WHEN price>20 THEN 'Yes'

ELSE 'No'

END AS Expenisve

FROM menu\_items



1. Data Modification - Update: -

* Update the price of the menu item with item\_id = 101 to $25

Solution:

UPDATE menu\_items

SET price=25 WHERE

menu\_item\_id = 101;

* To see the changes execute select query

SELECT \* FROM menu\_items

WHERE

menu\_item\_id = 101;



1. Data Modification - Insert: -

* Insert a new record into the menu\_items table for a dessert item.

Solution:

INSERT INTO menu\_items

VALUES(133,'Dessert','Sweet',2);

* To see the changes execute select query

SELECT \* FROM menu\_items

WHERE item\_name='Dessert'



1. Data Modification - Delete: -

* Delete all records from the order\_details table where the order\_id is less than 100.

Solution:

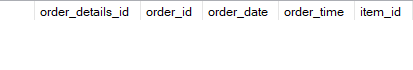
DELETE FROM order\_details

WHERE order\_id<100

* To recheck running select query and found no records with order\_id<100

SELECT \* FROM order\_details

WHERE order\_id<100



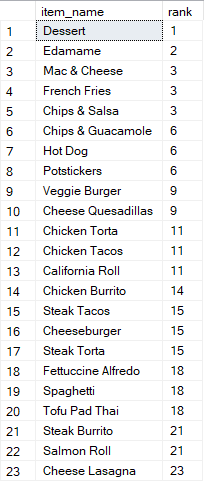
1. Window Functions - Rank:

* Rank menu items based on their prices, displaying the item\_name and its rank.

Solution: SELECT item\_name,

RANK() OVER(ORDER BY price) as "rank"

FROM menu\_items;



1. Window Functions - Lag and Lead:

* Display the item\_name and the price difference from the previous and next menu item.

Solution:

SELECT item\_name,price,

CASE

WHEN prev\_item\_price=0 THEN 0

ELSE

ABS(price-prev\_item\_price)

END as prev\_price\_diff,

CASE

WHEN next\_item\_price=0 THEN 0

ELSE

ABS(price-next\_item\_price)

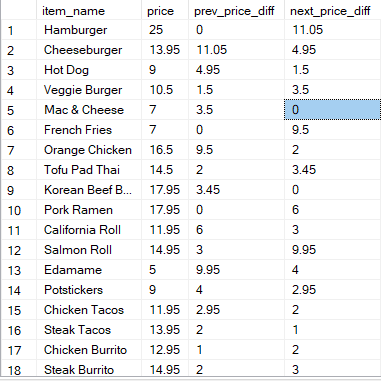
END as next\_price\_diff

FROM(SELECT item\_name,price,

LAG(price,1,0) OVER(ORDER BY menu\_item\_id) as prev\_item\_price,

LEAD(price,1,0) OVER(ORDER BY menu\_item\_id) as next\_item\_price

FROM menu\_items) as sub



1. Common Table Expressions (CTE): -

Create a CTE that lists menu items with prices above $15.

Use the CTE to retrieve the count of such items.

Solution:

WITH menu\_item AS

(SELECT \* FROM menu\_items

WHERE price>15)

SELECT COUNT(\*) as count\_above\_$15

FROM menu\_item



1. Advanced Joins: -

* Retrieve the order\_id, item\_name, and price for all orders with their respective menu item details.
* Include rows even if there is no matching menu item.

Solution:

SELECT order\_id,item\_name,price,menu\_item\_id,category

FROM order\_details as orders

FULL JOIN menu\_items as menu

ON orders.item\_id=menu.menu\_item\_id



1. Unpivot Data: -

* Unpivot the menu\_items table to show a list of menu item properties (item\_id, item\_name, category, price).

Solution:

SELECT

menu\_item\_id,

item\_name,

property,

value

FROM

(SELECT

menu\_item\_id,

item\_name,

CAST(category AS VARCHAR(50)) AS category,

CAST(price AS VARCHAR(50)) AS price

FROM

menu\_items

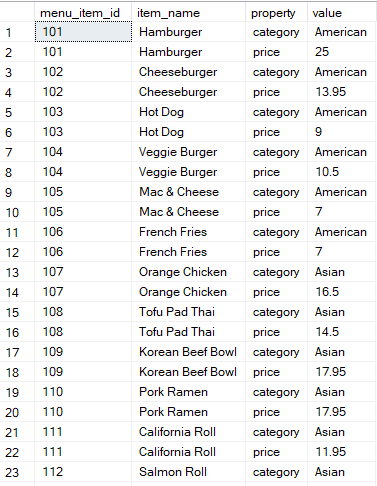
) AS source

UNPIVOT

(

value FOR property IN (category, price)

) AS unpvt;



1. Dynamic SQL: -

* Write a dynamic SQL query that allows users to filter menu items based on category and price range.

Solution:

DECLARE @Category NVARCHAR(50) = NULL;

DECLARE @MinPrice DECIMAL(5,2) = NULL;

DECLARE @MaxPrice DECIMAL(5,2) = NULL;

DECLARE @SQL NVARCHAR(MAX);

SET @SQL = N'

SELECT menu\_item\_id, item\_name, category, price

FROM menu\_items

WHERE 1=1';

IF @Category IS NOT NULL

BEGIN

SET @SQL = @SQL + N' AND category = @Category';

END

IF @MinPrice IS NOT NULL

BEGIN

SET @SQL = @SQL + N' AND price >= @MinPrice';

END

IF @MaxPrice IS NOT NULL

BEGIN

SET @SQL = @SQL + N' AND price <= @MaxPrice';

END

SET @SQL = @SQL + N' ORDER BY price';

PRINT @SQL; -- This will show you the constructed query

EXEC sp\_executesql @SQL,

N'@Category NVARCHAR(50), @MinPrice DECIMAL(5,2), @MaxPrice DECIMAL(5,2)',

@Category, @MinPrice, @MaxPrice;



1. Stored Procedure:

* Create a stored procedure that takes a menu category as input and returns the average price for that category.

Solution:

CREATE PROCEDURE avg\_price\_provider

@menu\_category VARCHAR(50)=NULL

AS

BEGIN

SELECT AVG(price) as avg\_price FROM menu\_items

WHERE category=@menu\_category;

END

EXEC avg\_price\_provider @menu\_category= 'Asian'



1. Triggers: -

* Design a trigger that updates a log table whenever a new order is inserted into the order\_details table.

Solution:

CREATE TRIGGER tr\_LogNewOrder

ON order\_details

AFTER INSERT

AS

BEGIN

INSERT INTO order\_log (order\_details\_id, order\_id, log\_date, action)

SELECT

i.order\_details\_id,

i.order\_id,

GETDATE(),

'New Order Inserted'

FROM

inserted i;

END

INSERT INTO order\_details (order\_details\_id, order\_id, order\_date, order\_time, item\_id)

VALUES (12235,5371, '2023-06-26', '14:30:00', 5001);

SELECT \* FROM order\_log;

