1. Convert Dataset to SQL Database:

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

2. Basic SELECT Queries:

- Retrieve all columns from the menu\_items table.

*select \* from menu\_items*

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

*select top 5 \* from order\_details*

3. 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.

*select item\_name,price from menu\_items where category='Main Course' order by price desc*

4. Aggregate Functions:

- Calculate the average price of menu items.

*select AVG(price) as average\_price from menu\_items*

- Find the total number of orders placed.

*select count(order\_details\_id) as total\_place from order\_details*

5. Joins:

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

*select m.item\_name, o.order\_date, o.order\_time from menu\_items m left join order\_details o on m.menu\_item\_id = o.item\_id*

6. Subqueries:

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

*select item\_name from menu\_items where price > (select avg(price) from menu\_items)*

7. Date and Time Functions:

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

*select count(order\_id) as count\_order, month(order\_date) as month\_order from order\_details group by month(order\_date)*

8. Group By and Having:

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

*select category from menu\_items group by category having avg(price) > 15*

- Include the count of items in each category.

*select category, count(item\_name) as count\_item from menu\_items group by category*

9. Conditional Statements:

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

*select item\_name, price, case when price > 20 then 'yes' else 'no' end as expensive from menu\_items*

10. Data Modification - Update:

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

*update menu\_items set menu\_item\_id = 101 where price = 25*

11. Data Modification - Insert:

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

*insert into menu\_items (menu\_item\_id, item\_name, category, price) values (133, 'Dessert', 'American', 20)*

12. Data Modification - Delete:

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

*delete from order\_details where order\_id < 100*

13. Window Functions - Rank:

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

*select item\_name, price, rank() over(order by price desc) as price\_rank from menu\_items*

14. Window Functions - Lag and Lead:

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

*select item\_name, price, price - lag(price) over(order by price desc) as previous\_price, lead(price) over(order by price desc) - price as next\_price from menu\_items*

15. Common Table Expressions (CTE):

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

*with Expensive as (select item\_name, price from menu\_items where price > 15)*

*select \* from Expensive*

- Use the CTE to retrieve the count of such items.

*with Expensive as (select item\_name, price from menu\_items where price > 15)*

*select count(\*) as count\_items from Expensive*

16. Advanced Joins:

- Retrieve the order\_id, item\_name, and price for all orders with their respective menu item details.

*select order\_id, item\_name, price from menu\_items m left join order\_details o on m.menu\_item\_id = o.item\_id*

- Include rows even if there is no matching menu item.

*select order\_id, item\_name, price from order\_details o left join menu\_items m on m.menu\_item\_id = o.item\_id*

17. Unpivot Data:

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

*select menu\_item\_id, property, property\_value from (select menu\_item\_id, CAST(item\_name AS VARCHAR(25)) AS item\_name, CAST(category AS VARCHAR(25)) AS category, CAST(price AS VARCHAR(25)) AS price from menu\_items) as menu unpivot (property\_value for property in(item\_name, category, price)) as unpvt*

18. Dynamic SQL:

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

*declare*

*@categoryfilter varchar(25) = 'American',*

*@minprice decimal(10,2) = 15.00,*

*@maxprice decimal(10,2) = 30.00,*

*@sql nvarchar(max);*

*set @sql = N'select menu\_item\_id, item\_name, category, price from menu\_items*

*where category = @categoryfilter and price between @minprice and @maxprice'*

*exec sp\_executesql @sql, N'@categoryfilter varchar(25), @minprice decimal(10,2), @maxprice decimal(10,2)', @categoryfilter, @minprice, @maxprice;*

19. Stored Procedure:

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

*create procedure avg\_price\_by\_category*

*@categoryfilter varchar(25) = 'American'*

*as*

*begin*

*select avg(price) as avg\_price from menu\_items where category = @categoryfilter*

*end*

20. Triggers:

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

*create table order\_log (log\_id int identity primary key, order\_id int, log\_date datetime, log\_message varchar(100))*

*create trigger after\_insert\_order\_details*

*on order\_details*

*after insert*

*as*

*begin*

*declare @order\_id int;*

*select @order\_id = order\_id from inserted;*

*insert into order\_log (order\_id, log\_date, log\_message)*

*values (@order\_id, getdate(), 'New order inserted into order\_details');*

*end*

Advanced SQL (optional)

21. Recursive Common Table Expressions (CTE):

- Implement a recursive CTE to display the hierarchy of menu items with their subcategories.

*with menu\_cte as ( select category as parent\_name, null as child\_name, 0 as level from menu\_items where category is not null group by category*

*union all*

*select mi.category as parent\_name, mi.item\_name as child\_name, 1 as level from menu\_items mi where mi.category is not null)*

*select case when level = 0 then parent\_name else replicate(' ', level) + child\_name end as item\_hierarchy from menu\_cte order by parent\_name, level, child\_name;*

22. Temporal Tables:

- Design a temporal table structure to track changes in menu item prices over time.

*ALTER TABLE menu\_items ADD*

*valid\_from DATETIME2 GENERATED ALWAYS AS ROW START HIDDEN NOT NULL DEFAULT SYSUTCDATETIME(),*

*valid\_to DATETIME2 GENERATED ALWAYS AS ROW END HIDDEN NOT NULL DEFAULT CONVERT(DATETIME2, '9999-12-31 23:59:59.9999999'),*

*PERIOD FOR SYSTEM\_TIME (valid\_from, valid\_to);*

*ALTER TABLE menu\_items SET (SYSTEM\_VERSIONING = ON (HISTORY\_TABLE = dbo.menu\_items\_history))*

23. Database Transactions:

- Write a series of SQL statements within a transaction that ensures atomicity for updating menu item prices and inserting a new order.

*DECLARE @OrderId INT = (SELECT ISNULL(MAX(order\_id), 0) + 1 FROM order\_details);*

*DECLARE @OrderDetailId INT = (SELECT ISNULL(MAX(order\_details\_id), 0) + 1 FROM order\_details);*

*DECLARE @OrderDate DATE = CAST(GETDATE() AS DATE);*

*DECLARE @OrderTime TIME = CAST(GETDATE() AS TIME);*

*DECLARE @MenuItemId INT = 25;*

*DECLARE @NewPrice DECIMAL(10,2) = 18.99;*

*BEGIN TRANSACTION*

*BEGIN TRY*

*UPDATE menu\_items SET price = @NewPrice WHERE menu\_item\_id = @MenuItemId;*

*INSERT INTO order\_details (order\_details\_id, order\_id, order\_date, order\_time, item\_id) VALUES (@OrderDetailId, @OrderId, @OrderDate, @OrderTime, @MenuItemId);*

*COMMIT TRANSACTION;*

*END TRY*

*BEGIN CATCH*

*ROLLBACK TRANSACTION;*

*PRINT ERROR\_MESSAGE();*

*END CATCH*

24. Database Security:

- Create a role in the database and assign permissions to the role to restrict access to sensitive tables.

*IF NOT EXISTS (SELECT \* FROM sys.database\_principals WHERE name = 'SensitiveDataReader')*

*BEGIN CREATE ROLE SensitiveDataReader END*

*IF NOT EXISTS (SELECT \* FROM sys.database\_principals WHERE name = 'Laman')*

*BEGIN CREATE USER Laman WITHOUT LOGIN END*

*ALTER ROLE SensitiveDataReader ADD MEMBER Laman;*

*GRANT SELECT ON dbo.menu\_items TO SensitiveDataReader;*

*GRANT SELECT ON dbo.order\_details TO SensitiveDataReader;*

25. Advanced Indexing:

- Analyze the menu\_items table and suggest an appropriate index to improve the performance of queries involving category-based filtering.

*CREATE NONCLUSTERED INDEX idx\_menu\_items\_category ON menu\_items (category) INCLUDE (menu\_item\_id, item\_name, price)*