A · P · U
ASIA PACIFIC UNIVERSITY
OF TECHNOLOGY & INNOVATION

Title : **Database Management Group Assignment Part 2**

Intake Code : **UCDF2309ICT(SE)**

Module Code : **AICT015-4-1-DBM**

Level of Study : **Diploma Semester 2**

Module Lecturer : **Mr. Muhammad Huzaifah bin Ismail**

Hand Out Date : **19 March 2024**

Submission Date : **26 May 2024**

Group Members :

No	Name	TP Number
1	Lim Chee Xuan	TP 075916
2	Paureen Tan Nie Nie	TP 075914
3	Phang Shea Wen	TP 075813

Table of Contents

1.0 Database Schema

1.1 Entity Relationship Diagram

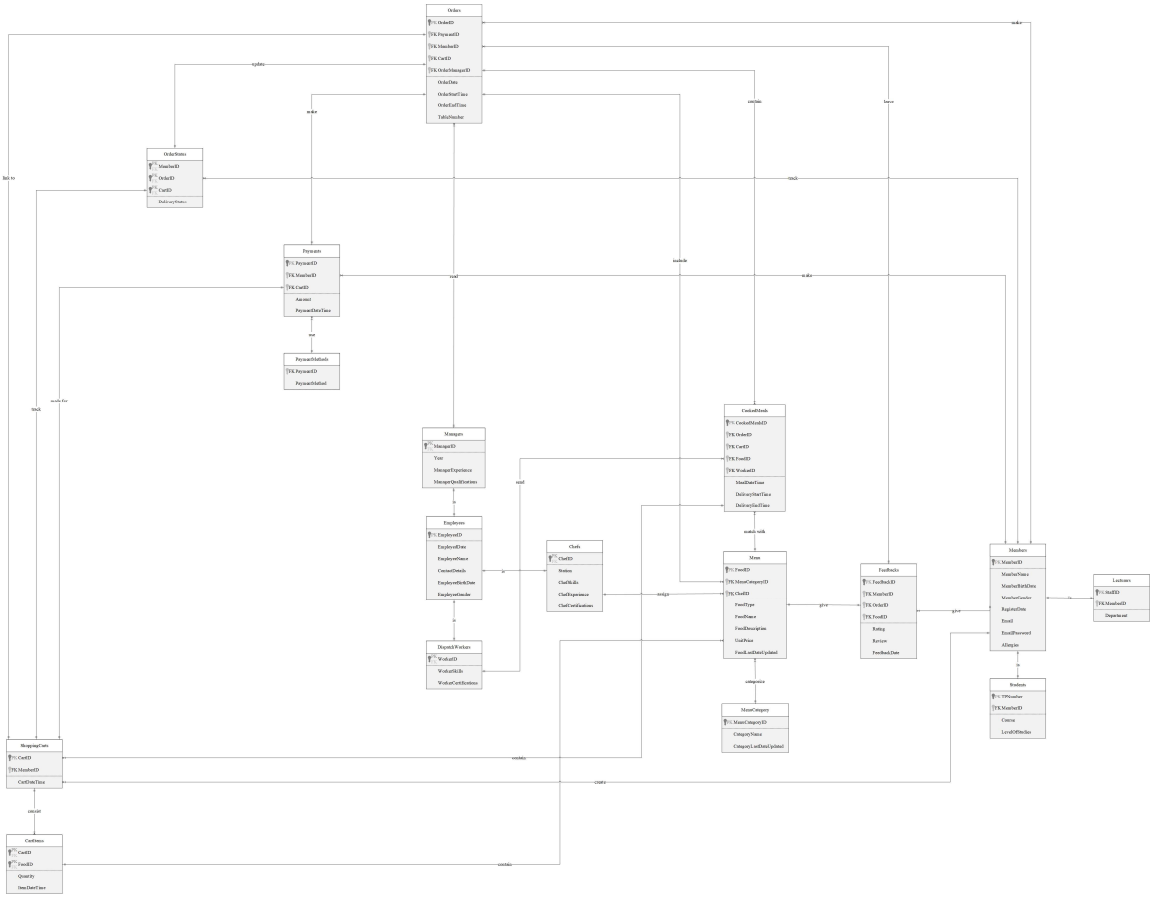
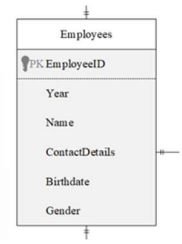
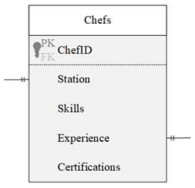
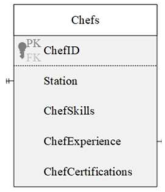
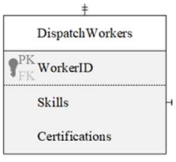
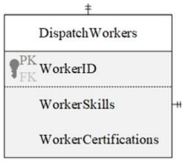
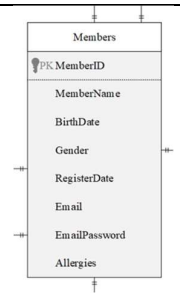
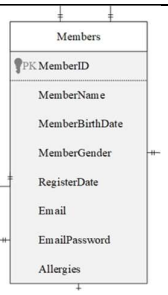
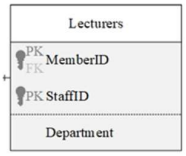
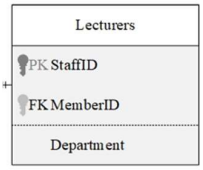
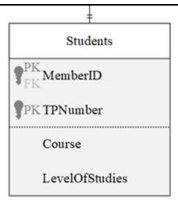
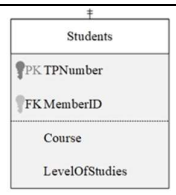
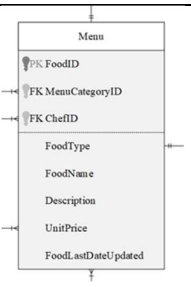
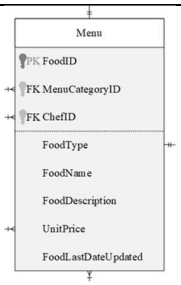
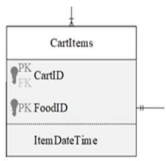
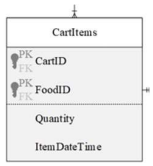
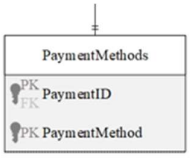
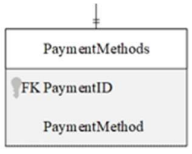


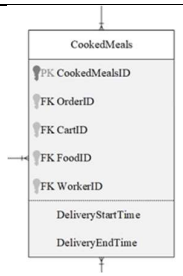
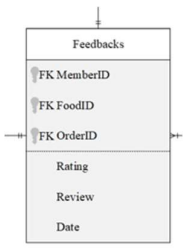
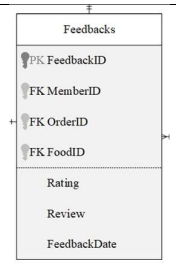
Diagram 1.1 APU Café Online Ordering System ERD

Updates from DBM Assignment Part 1, modification has been made to the Entity Relationship Diagram (ERD). Upon building our database environment in Microsoft SQL Server, a few necessary changes were made to the attributes name to ensure clarity and reduce redundancies; constraints enforced on some attributes were altered to enhance the database design; and new attributes were added for a more holistic data collection. *Diagram 1.1 APU Café Online Ordering System ERD* shows the modified ERD diagram, while *Table 1.1 Comparison between previous ERD and updated ERD* shows the comparison between the previous ERD diagram and the updated ERD.

Table 1.1 Comparison between previous ERD and updated ERD

Original ERD	New ERD	Explanation
		In the "Employees" entity: 1. Column named "Year" was renamed to "EmployedDate". 2. Column named "Name" was renamed to "EmployeeName". 3. Column named "Birthdate" was renamed to "EmployeeBirthDate". 4. Column named "Gender" was renamed to "EmployeeGender".
		In the "Managers" entity: 1. Constraint on "ManagerID" column was changed from a primary key to a combined primary key and foreign key. 2. Column named "Experience" was renamed to "ManagerExperience". 3. Column named "Qualifications" was renamed to "ManagerQualifications".
		In the "Chefs" entity: 1. Column named "Skills" was renamed to "ChefSkills". 2. Column named "Experience" was renamed to "ChefExperience". 3. Column named "Certifications" was renamed to "ChefCertifications".
		In the "DispatchWorkers" entity: 1. Column named "Skills" was renamed to "WorkerSkills". 2. Column named "Certifications" was renamed to "WorkerCertifications".

		In the "Employees" entity: 1. Column named "BirthDate" was renamed to "MemberBirthDate". 2. Column named "Gender" was renamed to "MemberGender".
		In the "Lecturers" entity: 1. Constraint on "MemberID" column was changed from a combined primary key and foreign key to a foreign key only.
		In the "Students" entity: 1. Constraint on "MemberID" column was changed from a combined primary key and foreign key to a foreign key only.
		In the "Menu" entity: 1. Column named "Description" was renamed to "FoodDescription"
		In the "CartItems" entity: 1. Constraint on "FoodID" column was changed from a primary key to a combined primary key and foreign key. 2. Column named "Quantity" is added.
		In the "PaymentMethods" entity: 1. Constraint on "PaymentID" column was changed from a combined primary key and foreign key to a foreign key only. 2. The foreign key constraint on "PaymentMethod" column was removed.

		In the "CookedMeals" entity: 1. Column named "MealDateTime" is added.
		In the "Feedbacks" entity: 1. Column named "FeedbackID" with a constraint of primary key is added. 2. Column named "Date" was renamed to "FeedbackDate".

1.2 Database Diagram

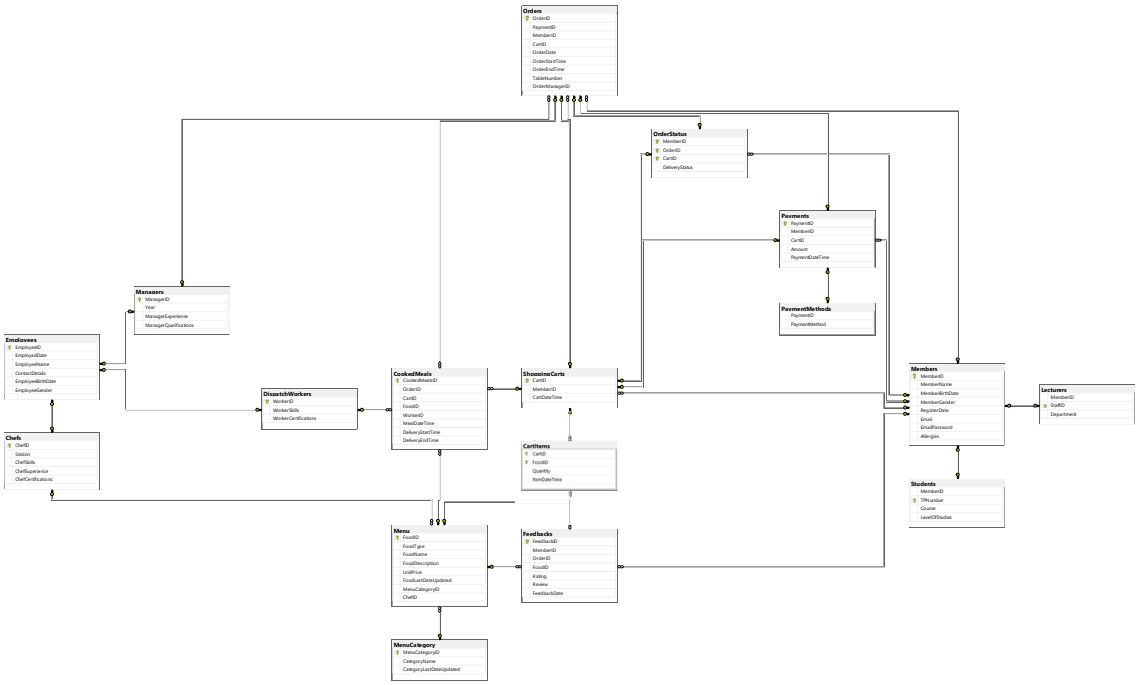


Diagram 1.2 APU Café Online Ordering System Database Diagram

2.0 SQL – Data Definition Language (DDL)

2.1 Summary

2.1.1 Overview of Entities in APU Café Online Ordering System

Table 2.1.1 Summary of the Table in APU Café Online Ordering System

Create Table Statement	Insert Data Statement	Results

```sql
CREATE TABLE Employees(
    EmployeeID NVARCHAR(10) NOT NULL PRIMARY KEY,
    EmployedDate DATETIME,
    EmployeeName NVARCHAR(100) NOT NULL,
    ContactDetails NVARCHAR(20),
    EmployeeBirthDate DATE,
    EmployeeGender NVARCHAR(20) CONSTRAINT validEmployeeGender CHECK(
        LOWER(EmployeeGender) IN('male', 'female', 'prefer not to say')
    )
);
```

```sql
INSERT INTO Employees VALUES
    ('MID01', '2004-02-01', 'Rowena.D', '0126455455', '1968-05-14', 'Female'),
    ('CFID01', '2004-07-11', 'Wesley.S', '0126533566', '1971-03-29', 'Male'),
    ('CFID02', '2004-09-23', 'Susana.R', '0173264595', '1974-08-21', 'Female'),
    ('CFID03', '2006-10-17', 'Otto.A', '0143962525', '1973-02-08', 'Male'),
    ('WID01', '2004-10-20', 'Elena.R', '0114888923', '1996-11-24', 'Female'),
    ('WID02', '2004-11-18', 'Rob.E', '0196587423', '1994-07-05', 'Male'),
    ('WID03', '2006-11-24', 'Joseph.B', '0162354815', '1998-10-09', 'Male');
```

	EmployeeID	EmployedDate	EmployeeName	ContactDetails	EmployeeBirthDate	EmployeeGender
1	CFID01	2004-07-11 00:00:00.000	Wesley.S	0126533566	1971-03-29	Male
2	CFID02	2004-09-23 00:00:00.000	Susana.R	0173264595	1974-08-21	Female
3	CFID03	2006-10-17 00:00:00.000	Otto.A	0143962525	1973-02-08	Male
4	MID01	2004-02-01 00:00:00.000	Rowena.D	0126455455	1968-05-14	Female
5	WID01	2004-10-20 00:00:00.000	Elena.R	0114888923	1996-11-24	Female
6	WID02	2004-11-18 00:00:00.000	Rob.E	0196587423	1994-07-05	Male
7	WID03	2006-11-24 00:00:00.000	Joseph.B	0162354815	1998-10-09	Male

```sql
CREATE TABLE Managers(
    ManagerID NVARCHAR(10) NOT NULL PRIMARY KEY FOREIGN KEY REFERENCES Employees(EmployeeID),
    Year DATE NOT NULL,
    ManagerExperience NVARCHAR(10),
    ManagerQualifications NVARCHAR(100)
);
```

```sql
INSERT INTO Managers VALUES
    ('MID01', '2004', '18 Years', 'Master of Business Administration (MBA)');
```

	ManagerID	Year	ManagerExperience	ManagerQualifications
1	MID01	2004-01-01	18 Years	Master of Business Administration (MBA)

```sql
CREATE TABLE Chefs(
    ChefID NVARCHAR(10) NOT NULL PRIMARY KEY FOREIGN KEY REFERENCES Employees(EmployeeID),
    Station NVARCHAR(50),
    ChefSkills NVARCHAR(300),
    ChefExperience NVARCHAR(10),
    ChefCertifications NVARCHAR(100)
);
```

```sql
INSERT INTO Chefs VALUES
    ('CFID01', 'Hot Kitchen', 'Cooking techniques like frying, stir-frying, and grilling', '9 Years', 'Culinary Arts Diploma'),
    ('CFID02', 'Beverage', 'Attention to detail for ensuring consistent drink quality and presentation', '5 Years', 'Food Safety and Sanitation Certification'),
    ('CFID03', 'Light Meal', 'Experience with preparing breakfast dishes', '7 Years', 'Basic Culinary Skills Certification');
```

	ChefID	Station	ChefSkills	ChefExperience	ChefCertifications
1	CFID01	Hot Kitchen	Cooking techniques like frying, stir-frying, and...	9 Years	Culinary Arts Diploma
2	CFID02	Beverage	Attention to detail for ensuring consistent drin...	5 Years	Food Safety and Sanitation Certification
3	CFID03	Light Meal	Experience with preparing breakfast dishes	7 Years	Basic Culinary Skills Certification

```sql
CREATE TABLE DispatchWorkers(
    WorkerID NVARCHAR(10) NOT NULL PRIMARY KEY FOREIGN KEY REFERENCES Employees(EmployeeID),
    WorkerSkills NVARCHAR(300),
    WorkerCertifications NVARCHAR(100)
);
```

```sql
INSERT INTO DispatchWorkers VALUES
    ('WID01', 'Problem-Solving Skills', 'Customer Service Certification'),
    ('WID02', 'Time Management and Organization', 'Delivery Dispatch Certification'),
    ('WID03', 'Communication Skills', 'Customer Service Certification');
```

	WorkerID	WorkerSkills	WorkerCertifications
1	WID01	Problem-Solving Skills	Customer Service Certification
2	WID02	Time Management and Organization	Delivery Dispatch Certification
3	WID03	Communication Skills	Customer Service Certification

```sql
CREATE TABLE Members(
    MemberID NVARCHAR(10) NOT NULL PRIMARY KEY,
    MemberName NVARCHAR(100) NOT NULL,
    MemberBirthDate DATE,
    MemberGender NVARCHAR(20) CONSTRAINT validMemberGender CHECK(
        LOWER(MemberGender) IN('male', 'female', 'prefer not to say')
    ),
    RegisterDate DATETIME,
    Email NVARCHAR(100),
    EmailPassword NVARCHAR(100),
    Allergies NVARCHAR(100)
);
```

```sql
INSERT INTO Members VALUES
    ('MB0001', 'Ali', '1994-10-08', 'Male', '2023-01-10 08:09:10.123', 'ali08@gmail.com', 'rduferie0', NULL),
    ('MB0002', 'Rainbow', '1999-05-14', 'Female', '2023-01-20 14:22:11.124', 'bow20@gmail.com', 'dA1! mRJ', 'Peanut'),
    ('MB0003', 'Sarah', '1965-03-20', 'Female', '2023-01-27 13:09:56.243', 's1714@gmail.com', 'S@r13', 'Seafood'),
    ('MB0004', 'Jason', '1985-02-14', 'Prefer not to say', '2023-02-15 10:12:12.453', 'j171n@gmail.com', 'JaSaN2003', NULL),
    ('MB0005', 'K.Vince', '2004-12-11', 'Male', '2023-02-21 16:44:04.444', 'kv0566@gmail.com', 'pP4#5jjC', 'Milk'),
    ('MB0006', 'MacHill', '1975-04-08', 'Male', '2023-02-24 4:44:04.448', 'mh0008@gmail.com', 'P@ssw673', 'Peach'),
    ('MB0007', 'J.Wenton', '1994-09-24', 'Prefer not to say', '2023-03-01 6:56:06.666', 'jw3687@gmail.com', 'JW!1994?', 'Banana'),
    ('MB0008', 'Micheal', '2005-06-09', 'Female', '2023-03-15 18:46:06.668', 'm1988@gmail.com', 'tD4!) Msi', NULL),
    ('MB0009', 'Shanan', '1977-11-11', 'Female', '2023-03-26 17:33:13.426', 'shan17@gmail.com', 'sMarTmE844', NULL),
    ('MB0010', 'S.Hanson', '1971-07-16', 'Male', '2023-03-05 15:23:45.231', 'sh1234@gmail.com', 'IdkWhat1971', 'Seafood');
```

	MemberID	MemberName	MemberBirthDate	MemberGender	RegisterDate	Email	EmailPassword	Allergies
1	MB0001	Ali	1994-10-08	Male	2023-01-10 08:09:10.123	ali08@gmail.com	rduferie0	NULL
2	MB0002	Rainbow	1999-05-14	Female	2023-01-20 14:22:11.123	bow20@gmail.com	dA1! mRJ	Peanut
3	MB0003	Sarah	1965-03-20	Female	2023-01-27 13:09:56.243	s1714@gmail.com	S@r13	Seafood
4	MB0004	Jason	1985-02-14	Prefer not to say	2023-02-15 10:12:12.453	j171n@gmail.com	JaSaN2003	NULL
5	MB0005	K.Vince	2004-12-11	Male	2023-02-21 16:44:04.443	kv0566@gmail.com	pP4#5jjC	Milk
6	MB0006	MacHill	1975-04-08	Male	2023-02-24 04:44:04.447	mh0008@gmail.com	P@ssw673	Peach
7	MB0007	J.Wenton	1994-09-24	Prefer not to say	2023-03-01 06:56:06.667	jw3687@gmail.com	JW!1994?	Banana
8	MB0008	Micheal	2005-06-09	Female	2023-03-15 18:46:06.667	m1988@gmail.com	tD4!) Msi	NULL
9	MB0009	Shanan	1977-11-11	Female	2023-03-26 17:33:13.427	shan17@gmail.com	sMarTmE844	NULL
10	MB0010	S.Hanson	1971-07-16	Male	2023-03-05 15:23:45.230	sh1234@gmail.com	IdkWhat1971	Seafood

```sql
CREATE TABLE Lecturers(
    MemberID NVARCHAR(10) FOREIGN KEY REFERENCES Members(MemberID) NULL UNIQUE,
    StaffID NVARCHAR(10) NOT NULL PRIMARY KEY,
    Department NVARCHAR(100)
);
```

```sql
INSERT INTO Lecturers VALUES
    ('MB0003', 'EMP100023', 'Academic Administration'),
    ('MB0004', 'EMP102054', 'Finance'),
    ('MB0006', 'EMP100889', 'School of Business'),
    ('MB0009', 'EMP107888', 'School of Technology'),
    ('MB0010', 'EMP104566', 'Human Resources'),
    (NULL, 'EMP103893', 'Psychology');
```

	MemberID	StaffID	Department
1	MB0003	EMP100023	Academic Administration
2	MB0006	EMP100889	School of Business
3	MB0004	EMP102054	Finance
4	NULL	EMP103893	Psychology
5	MB0010	EMP104566	Human Resources
6	MB0009	EMP107888	School of Technology

```sql
CREATE TABLE Students(
    MemberID nvarchar(10) FOREIGN KEY REFERENCES Members(MemberID) NULL UNIQUE,
    TPNumber nvarchar(10) NOT NULL PRIMARY KEY,
    Course nvarchar(100),
    LevelOfStudies nvarchar(100)
);
```

```sql
INSERT INTO Students VALUES
    ('MB0001', 'TP075966', 'Account', 'Master'),
    ('MB0002', 'TP054411', 'Software Engineering', 'PhD'),
    ('MB0005', 'TP045533', 'Design and Media', 'Diploma'),
    ('MB0007', 'TP062248', 'Data Informatics', 'Diploma'),
    ('MB0008', 'TP088422', 'Cybersecurity', 'Degree'),
    (NULL, 'TP645834', 'Artificial Intelligence', 'Degree');
```

	MemberID	TPNumber	Course	LevelOfStudies
1	MB0005	TP045533	Design and Media	Diploma
2	MB0002	TP054411	Software Engineering	PhD
3	MB0007	TP062248	Data Informatics	Diploma
4	MB0001	TP075966	Account	Master
5	MB0008	TP088422	Cybersecurity	Degree
6	NULL	TP645834	Artificial Intelligence	Degree

```sql
CREATE TABLE ShoppingCarts(
    CartID NVARCHAR(10) NOT NULL PRIMARY KEY,
    MemberID NVARCHAR(10) NOT NULL FOREIGN KEY REFERENCES Members(MemberID),
    CartDateTime DATETIME
);
```

```sql
INSERT INTO ShoppingCarts VALUES
    ('CID001', 'MB0005', '2023-03-15 11:29:00.123'),
    ('CID002', 'MB0002', '2023-03-20 13:02:10.453'),
    ('CID003', 'MB0006', '2023-03-30 12:07:09.001'),
    ('CID004', 'MB0008', '2023-07-02 11:44:34.907'),
    ('CID005', 'MB0010', '2023-07-09 12:59:10.291'),
    ('CID006', 'MB0007', '2023-07-18 12:12:37.241'),
    ('CID007', 'MB0003', '2024-06-06 11:37:02.874'),
    ('CID008', 'MB0001', '2024-06-06 12:16:23.542'),
    ('CID009', 'MB0005', '2024-06-06 12:27:13.379'),
    ('CID010', 'MB0007', '2024-06-06 13:02:43.124'),
    ('CID011', 'MB0007', '2024-06-06 13:41:29.431');
```

	CartID	MemberID	CartDateTime
1	CID001	MB0005	2023-03-15 11:29:00.123
2	CID002	MB0002	2023-03-20 13:02:10.453
3	CID003	MB0006	2023-03-30 12:07:09.000
4	CID004	MB0008	2023-07-02 11:44:34.907
5	CID005	MB0010	2023-07-09 12:59:10.290
6	CID006	MB0007	2023-07-18 12:12:37.240
7	CID007	MB0003	2024-06-06 11:37:02.873
8	CID008	MB0001	2024-06-06 12:16:23.543
9	CID009	MB0005	2024-06-06 12:27:13.380
10	CID010	MB0007	2024-06-06 13:02:43.123
11	CID011	MB0007	2024-06-06 13:41:29.430

```sql
CREATE TABLE MenuCategory(
    MenuCategoryID NVARCHAR(10) NOT NULL PRIMARY KEY,
    CategoryName NVARCHAR(50) NOT NULL UNIQUE,
    CategoryLastDateUpdated DATETIME
);
```

```sql
INSERT INTO MenuCategory VALUES
    ('MC01', 'Malay', '2023-01-15'),
    ('MC02', 'Chinese', '2023-01-15'),
    ('MC03', 'India', '2023-01-15'),
    ('MC04', 'Western', '2023-01-15'),
    ('MC05', 'Dessert', '2023-01-15'),
    ('MC06', 'Beverage', '2023-01-15');
```

	MenuCategoryID	CategoryName	CategoryLastDateUpdated
1	MC01	Malay	2023-01-15 00:00:00.000
2	MC02	Chinese	2023-01-15 00:00:00.000
3	MC03	India	2023-01-15 00:00:00.000
4	MC04	Western	2023-01-15 00:00:00.000
5	MC05	Dessert	2023-01-15 00:00:00.000
6	MC06	Beverage	2023-01-15 00:00:00.000

```sql
CREATE TABLE Menu(
    FoodID NVARCHAR(10) NOT NULL PRIMARY KEY,
    FoodType NVARCHAR(10),
    FoodName NVARCHAR(100) NOT NULL UNIQUE,
    FoodDescription NVARCHAR(MAX),
    UnitPrice DECIMAL(10,2),
    FoodLastDateUpdated DATETIME,
    MenuCategoryID NVARCHAR(10) FOREIGN KEY REFERENCES MenuCategory(MenuCategoryID),
    ChefID NVARCHAR(10) FOREIGN KEY REFERENCES Chefs(ChefID)
);
```

```sql
CREATE TABLE CartItems(
    CartID NVARCHAR(10) NOT NULL FOREIGN KEY REFERENCES ShoppingCarts(CartID),
    FoodID NVARCHAR(10) NOT NULL FOREIGN KEY REFERENCES MENU(FoodID),
    Quantity INT,
    ItemDateTime DATETIME,
    PRIMARY KEY (CartID, FoodID)
);
```

```sql
INSERT INTO CartItems VALUES
    ('CID001', 'F001', 2, '2023-03-15 11:30:31.010'),
    ('CID001', 'F009', 3, '2023-03-15 11:31:32.012'),
    ('CID001', 'F004', 1, '2023-03-15 11:31:50.210'),
    ('CID002', 'F006', 2, '2023-03-20 13:04:05.012'),
    ('CID002', 'F008', 4, '2023-03-20 13:05:01.002'),
    ('CID003', 'F007', 3, '2023-03-30 12:07:53.101'),
    ('CID003', 'F010', 2, '2023-03-30 12:08:32.124'),
    ('CID004', 'F002', 1, '2023-07-02 11:45:23.672'),
    ('CID005', 'F007', 3, '2023-07-09 13:00:43.001'),
    ('CID006', 'F006', 2, '2023-07-18 12:13:16.118'),
    ('CID006', 'F004', 1, '2023-07-18 12:13:50.001'),
    ('CID007', 'F001', 2, '2024-06-06 11:38:01.245'),
    ('CID008', 'F003', 1, '2024-06-06 12:17:08.224'),
    ('CID008', 'F009', 1, '2024-06-06 12:18:23.108'),
    ('CID009', 'F002', 3, '2024-06-06 12:28:16.139'),
    ('CID010', 'F007', 2, '2024-06-06 13:03:23.193'),
    ('CID011', 'F006', 2, '2024-06-06 13:42:30.537');
```

	CartID	FoodID	Quantity	ItemDateTime
1	CID001	F001	2	2023-03-15 11:30:31.010
2	CID001	F004	1	2023-03-15 11:31:50.210
3	CID001	F009	3	2023-03-15 11:31:32.013
4	CID002	F006	2	2023-03-20 13:04:05.013
5	CID002	F008	4	2023-03-20 13:05:01.003
6	CID003	F007	3	2023-03-30 12:07:53.100
7	CID003	F010	2	2023-03-30 12:08:32.123
8	CID004	F002	1	2023-07-02 11:45:23.673
9	CID005	F007	3	2023-07-09 13:00:43.000
10	CID006	F004	1	2023-07-18 12:13:50.000
11	CID006	F006	2	2023-07-18 12:13:16.117
12	CID007	F001	2	2024-06-06 11:38:01.247
13	CID008	F003	1	2024-06-06 12:17:08.223
14	CID008	F009	1	2024-06-06 12:18:23.107
15	CID009	F002	3	2024-06-06 12:28:16.140
16	CID010	F007	2	2024-06-06 13:03:23.193
17	CID011	F006	2	2024-06-06 13:42:30.537

```sql
CREATE TABLE Payments(
    PaymentID NVARCHAR(10) NOT NULL PRIMARY KEY,
    MemberID NVARCHAR(10)NOT NULL FOREIGN KEY REFERENCES Members(MemberID),
    CartID NVARCHAR(10) NOT NULL FOREIGN KEY REFERENCES ShoppingCarts(CartID) UNIQUE,
    Amount DECIMAL(10, 2),
    PaymentDateTime DATETIME
);
```

```sql
INSERT INTO Payments VALUES
    ('PYID001', 'MB0005', 'CID001', 16.50, '2023-03-15 11:40:23.600'),
    ('PYID002', 'MB0002', 'CID002', 32.60, '2023-03-20 13:10:09.080'),
    ('PYID003', 'MB0006', 'CID003', 31.00, '2023-03-30 12:13:52.700'),
    ('PYID004', 'MB0008', 'CID004', 5.00, '2023-07-02 11:50:53.006'),
    ('PYID005', 'MB0010', 'CID005', 27.00, '2023-07-09 13:06:46.430'),
    ('PYID006', 'MB0007', 'CID006', 21.50, '2023-07-18 13:16:13.230'),
    ('PYID007', 'MB0003', 'CID007', 6.00, '2024-06-06 11:45:37:913'),
    ('PYID008', 'MB0001', 'CID008', 3.50, '2024-06-06 12:24:53:519'),
    ('PYID009', 'MB0005', 'CID009', 15.00, '2024-06-06 12:36:17:802'),
    ('PYID010', 'MB0007', 'CID010', 18.00, '2024-06-06 13:11:21:643'),
    ('PYID011', 'MB0007', 'CID011', 17.00, '2024-06-06 13:50:34:561');
```

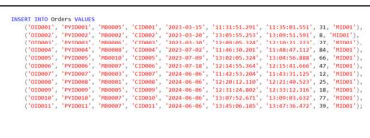
	PaymentID	MemberID	CartID	Amount	PaymentDateTime
1	PYID001	MB0005	CID001	16.50	2023-03-15 11:40:23.600
2	PYID002	MB0002	CID002	32.60	2023-03-20 13:10:09.080
3	PYID003	MB0006	CID003	31.00	2023-03-30 12:13:52.700
4	PYID004	MB0008	CID004	5.00	2023-07-02 11:50:53.007
5	PYID005	MB0010	CID005	27.00	2023-07-09 13:06:46.430
6	PYID006	MB0007	CID006	21.50	2023-07-18 13:16:13.230
7	PYID007	MB0003	CID007	6.00	2024-06-06 11:45:37.913
8	PYID008	MB0001	CID008	3.50	2024-06-06 12:24:53.520
9	PYID009	MB0005	CID009	15.00	2024-06-06 12:36:17.803
10	PYID010	MB0007	CID010	18.00	2024-06-06 13:11:21.643
11	PYID011	MB0007	CID011	17.00	2024-06-06 13:50:34.560

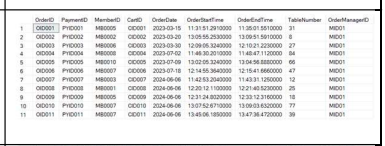
```sql
CREATE TABLE PaymentMethods(
    PaymentID NVARCHAR(10) NOT NULL FOREIGN KEY REFERENCES Payments(PaymentID) UNIQUE,
    PaymentMethod NVARCHAR(50)
);
```

```sql
INSERT INTO PaymentMethods VALUES
    ('PYID001', 'Pay at Counter'),
    ('PYID002', 'Online Banking'),
    ('PYID003', 'Pay at Counter'),
    ('PYID004', 'Online Banking'),
    ('PYID005', 'Online Banking'),
    ('PYID006', 'Pay at Counter'),
    ('PYID007', 'Online Banking'),
    ('PYID008', 'Pay at Counter'),
    ('PYID009', 'Pay at Counter'),
    ('PYID010', 'Online Banking'),
    ('PYID011', 'Online Banking');
```

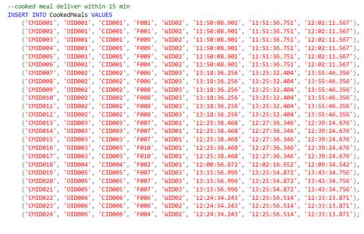
	PaymentID	PaymentMethod
1	PYID001	Pay at Counter
2	PYID002	Online Banking
3	PYID003	Pay at Counter
4	PYID004	Online Banking
5	PYID005	Online Banking
6	PYID006	Pay at Counter
7	PYID007	Online Banking
8	PYID008	Pay at Counter
9	PYID009	Pay at Counter
10	PYID010	Online Banking
11	PYID011	Online Banking

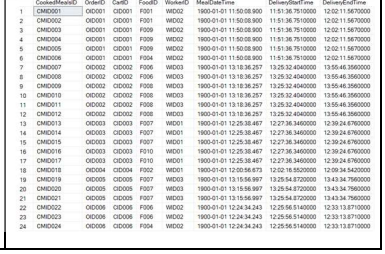
```sql
CREATE TABLE Orders(
    OrderID NVARCHAR(10) NOT NULL PRIMARY KEY,
    PaymentID NVARCHAR(10) NOT NULL FOREIGN KEY REFERENCES Payments(PaymentID) UNIQUE,
    MemberID NVARCHAR(10) NOT NULL FOREIGN KEY REFERENCES Members(MemberID),
    CartID NVARCHAR(10) NOT NULL FOREIGN KEY REFERENCES ShoppingCarts(CartID),
    OrderDate DATE,
    OrderStartTime TIME,
    OrderEndTime TIME,
    TableNumber NVARCHAR(10),
    OrderManagerID NVARCHAR(10) FOREIGN KEY REFERENCES Managers(ManagerID),
);
```





```sql
CREATE TABLE CookedMeals(
    CookedMealsID NVARCHAR(10) NOT NULL PRIMARY KEY,
    OrderID NVARCHAR(10) NOT NULL FOREIGN KEY REFERENCES Orders(OrderID),
    CartID NVARCHAR(10) NOT NULL FOREIGN KEY REFERENCES ShoppingCarts(CartID),
    FoodID NVARCHAR(10) NOT NULL FOREIGN KEY REFERENCES Menu(FoodID),
    WorkerID NVARCHAR(10) NOT NULL FOREIGN KEY REFERENCES DispatchWorkers(WorkerID),
    MealDateTime DATETIME,
    DeliveryStartTime TIME,
    DeliveryEndTime TIME,
);
```





```
INSERT INTO OrderStatus VALUES
    ('MB0005', 'OID001', 'CID001', 'Completed'),
    ('MB0002', 'OID002', 'CID002', 'Completed'),
    ('MB0006', 'OID003', 'CID003', 'Completed'),
    ('MB0008', 'OID004', 'CID004', 'Completed'),
    ('MB0010', 'OID005', 'CID005', 'Completed'),
    ('MB0007', 'OID006', 'CID006', 'Completed'),
    ('MB0003', 'OID007', 'CID007', 'Preparing'),
    ('MB0001', 'OID008', 'CID008', 'Read'),
    ('MB0005', 'OID009', 'CID009', 'Waiting'),
    ('MB0007', 'OID010', 'CID010', 'Confirmed'),
    ('MB0007', 'OID011', 'CID011', 'Pending');
```

	MemberID	OrderID	CartID	DeliveryStatus
1	MB0001	OID008	CID008	Read
2	MB0002	OID002	CID002	Completed
3	MB0003	OID007	CID007	Preparing
4	MB0005	OID001	CID001	Completed
5	MB0005	OID009	CID009	Waiting
6	MB0006	OID003	CID003	Completed
7	MB0007	OID006	CID006	Completed
8	MB0007	OID010	CID010	Confirmed
9	MB0007	OID011	CID011	Pending
10	MB0008	OID004	CID004	Completed
11	MB0010	OID005	CID005	Completed


```
CREATE TABLE Feedbacks(
    FeedbackID NVARCHAR(10) NOT NULL PRIMARY KEY,
    MemberID NVARCHAR(10) NOT NULL FOREIGN KEY REFERENCES Members(MemberID),
    OrderID NVARCHAR(10) NOT NULL FOREIGN KEY REFERENCES Orders(OrderID),
    FoodID NVARCHAR(10) NOT NULL FOREIGN KEY REFERENCES Menu(FoodID),
    Rating INT CONSTRAINT validRating CHECK(
        Rating BETWEEN 1 AND 5
    ),
    Review NVARCHAR(MAX),
    FeedbackDate DATE
);
```

	FeedbackID	MemberID	OrderID	FoodID	Rating	Review	FeedbackDate
1	FID001	MB0005	OID001	F001	5	The Nasi Lemak is very nice and tasty :P	2023-03-16
2	FID002	MB0005	OID001	F009	3	Just normal kopi	2023-03-16
3	FID003	MB0005	OID001	F004	2	The waffle is too sweet for me!	2023-03-16
4	FID004	MB0002	OID002	F006	1	Although the food is not bad, but the delivery is t...	2023-03-23
5	FID005	MB0002	OID002	F008	1	Although the food is not bad, but the delivery is t...	2023-03-23
6	FID006	MB0006	OID003	F007	4	The chicken chop is crispy and delicious :)	2023-03-31
7	FID007	MB0006	OID003	F010	4	The tea is okey.	2023-03-31
8	FID008	MB0008	OID004	F002	5	The nasi goreng is very delicious. I like the tast...	2023-07-05
9	FID009	MB0010	OID005	F007	2	Very disappointed as the delivery is too slow, a...	2023-07-12
10	FID010	MB0007	OID006	F006	4	The spaghetti is tasty and the portion is very wo...	2023-07-21
11	FID011	MB0007	OID006	F004	4	The waffle also taste good.	2023-07-21

2.1.2 List of Data Type in APU Café Online Ordering System

Table 2.1.2 Summary of Data Type for each attribute in APU Café Online Ordering System

Data Type	Explanation
NVARCHAR(number)	Store Unicode character data with a maximum length of "number" characters.
NVARCHAR(MAX)	Store data with a maximum length up to the database limit.
DATETIME	Store the data with a format of year, month, day, hour, minute, second and fraction of second.
DATE	Store the data with a format of year, month and day.
TIME	Store the data with a format of hour, minute and second.
DECIMAL(10,2)	Store the data in decimal with a total of 10 digits and 2 decimal places.
INT	Store a data with whole numbers.

2.2 Create Table Statement

Table 2.2.1 Database DBM_Assignment

SQL Statements	
`CREATE DATABASE DBM_Assignment;` `USE DBM_Assignment;`	
Explanation	
1	"`CREATE DATABASE DBM_Assignment`" statement - Used to create a new database which is named as "`DBM_Assignment`" within Database Management System (DBMS).
2	"`USE DBM_Assignment`" statement - Used to instructs the DBMS to start using a specific database, which is "`DBM_Assignment`".

Table 2.2.2 "Employees" Entity

SQL Statements	
`CREATE TABLE Employees(` `EmployeeID NVARCHAR(10) NOT NULL PRIMARY KEY,` `EmployedDate DATETIME,` `EmployeeName NVARCHAR(100) NOT NULL,` `ContactDetails NVARCHAR(20),` `EmployeeBirthDate DATE,` `EmployeeGender NVARCHAR(20) CONSTRAINT validEmployeeGender CHECK(` `LOWER(EmployeeGender) IN('male', 'female', 'prefer not to say')` `)` `);`	
Explanation	
1	"`CREATE TABLE Employees`" statement - Used to create a new entity within the database "`DBM_Assignment`" and named the entity as "`Employees`".
2	"`EmployeeID NVARCHAR(10) NOT NULL PRIMARY KEY`" statement - Defines the first column named "`EmployeeID`" (data type: `NVARCHAR(10)`) to store the Employee ID of each employee in the "`Employees`" entity. - "`NOT NULL`" is used to ensure that "`EmployeeID`" column does not contain null values, every row in this entity for the "`EmployeeID`" column must have a value. - "`PRIMARY KEY`" is used to designate "`EmployeeID`" as the primary key for the "`Employees`" entity. A primary key uniquely identifies each row and it cannot contain duplicate values in the "`Employees`" entity.

3	"`EmployedDate DATETIME`" statement - Defines the next column named "`EmployedDate`" (data type: `DATETIME`) to store the information of the employee's employed date.
4	"`EmployeeName NVARCHAR(100) NOT NULL`" statement - Defines another column named "`EmployeeName`" (data type: `NVARCHAR(100)`) to store employees' names. - "`NOT NULL`" is used to ensure a value is always provided for the employees' name.
5	"`ContactDetails NVARCHAR(20)`" statement - Defines a column named "`ContactDetails`" (data type: `NVARCHAR(20)`) to store the contact information of employees.
6	"`EmployeeBirthDate DATE`" statement - Defines a column named "`EmployeeBirthDate`" (data type: `DATE`) to store employees' birth dates.
7	"`EmployeeGender NVARCHAR(20) CONSTRAINT validEmployeeGender CHECK(LOWER(EmployeeGender) IN('male', 'female', 'prefer not to say'))`" statement - Defines the last column named "`EmployeeGender`" (data type: `NVARCHAR(20)`) to store employees' gender information. - "`CONSTRAINT validEmployeeGender CHECK(LOWER(EmployeeGender) IN('male', 'female', 'prefer not to say'))`" defines a constraint named "`validEmployeeGender`", which used the "`CHECK`" clause to ensure that the value stored in the "`EmployeeGender`" column must be either `'male'`, `'female'` or `'prefer not to say'`.

Table 2.2.3 "Managers" Entity

SQL Statements	
	
Explanation	
1	"`CREATE TABLE Managers`" statement - Used to create a new entity within the database "`DBM_Assignment`" and named the entity as "`Managers`".
2	"`ManagerID NVARCHAR(10) NOT NULL PRIMARY KEY FOREIGN KEY REFERENCES Employees(EmployeeID)`" statement - Defines the first column named "`ManagerID`" (data type: `NVARCHAR(10)`) to store the manager ID of the manager in the "`Managers`" entity. - "`NOT NULL`" is used to ensure that "`ManagerID`" column cannot be empty. - "`PRIMARY KEY`" is used to designate "`ManagerID`" as the primary key for the "`Managers`" entity. A primary key uniquely identifies each row and it cannot contain duplicate values in the "`Managers`" entity. - "`FOREIGN KEY REFERENCES Employees(EmployeeID)`" is used to establish a foreign key relationship, which ensures the value in the "`ManagerID`" column of the "`Managers`" entity must exist as valid IDs in the "`EmployeeID`" column of the "`Employees`" entity.
3	"`Year NOT NULL`" statement - Defines the next column named "`Year`" (data type: `DATE`) to store the information of the year became manager. - "`NOT NULL`" is used to ensure that "`Year`" column cannot be empty.
4	"`ManagerExperience NVARCHAR(10)`" statement - Defines another column named "`ManagerExperience`" (data type: `NVARCHAR(10)`) to store managers' experience.
5	"`ManagerQualifications NVARCHAR(100)`" statement - Defines a column named "`ManagerQualifications`" (data type: `NVARCHAR(100)`) to store the qualifications the manager has.

Table 2.2.4 "Chefs" Entity

SQL Statements	
	
Explanation	
1	"`CREATE TABLE Chefs`" statement - Used to create a new entity within the database "`DBM_Assignment`" and named the entity as "`Chefs`".
2	"`ChefID NVARCHAR(10) NOT NULL PRIMARY KEY FOREIGN KEY REFERENCES Employees(EmployeeID)`" statement - Defines the first column named "`ChefID`" (data type: `NVARCHAR(10)`) to store the chef ID of the chefs in the entity "`Chefs`". - "`NOT NULL`" is used to ensure that "`ChefID`" column cannot be empty. - "`PRIMARY KEY`" is used to designate "`ChefID`" as the primary key for the "`Chefs`" entity. A primary key uniquely identifies each row and it cannot contain duplicate values in the "`Chefs`" entity. - "`FOREIGN KEY REFERENCES Employees(EmployeeID)`" is used to establish a foreign key relationship, which ensures the value in the "`ChefID`" column of the "`Chefs`" entity must exist as valid IDs in the "`EmployeeID`" column of the "`Employees`" entity.
3	"`Station NVARCHAR(50)`" statement - Defines a column named "`Station`" (data type: `NVARCHAR(50)`) to store the chefs' work area.
4	"`ChefSkills NVARCHAR(300)`" statement - Defines a column named "`ChefSkills`" (data type: `NVARCHAR(300)`) to store the information of skills that the chef has.
5	"`ChefExperience NVARCHAR(10)`" statement - Defines another column named "`ChefExperience`" (data type: `NVARCHAR(10)`) to store chefs' experience.
6	"`ChefCertifications NVARCHAR(100)`" statement - Defines a column named "`ChefCertifications`" (data type: `NVARCHAR(100)`) to store the certifications the chef has.

Table 2.2.5 "DispatchWorkers" Entity

SQL Statements	
	
Explanation	
1	"`CREATE TABLE DispatchWorkers`" statement - Used to create a new entity within the database "`DBM_Assignment`" and named the entity as "`DispatchWorkers`".
2	"`WorkerID NVARCHAR(10) NOT NULL PRIMARY KEY FOREIGN KEY REFERENCES Employees(EmployeeID)`" statement - Defines the first column named "`WorkerID`" (data type: `NVARCHAR(10)`) to store the worker ID of the dispatch workers in the entity "`DispatchWorkers`". - "`NOT NULL`" is used to ensure that "`WorkerID`" column cannot be empty. - "`PRIMARY KEY`" is used to designate "`WorkerID`" as the primary key for the "`DispatchWorkers`" entity. A primary key uniquely identifies each row and it cannot contain duplicate values in the "`DispatchWorkers`" entity. - "`FOREIGN KEY REFERENCES Employees(EmployeeID)`" is used to establish a foreign key relationship, which ensures the value in the "`WorkerID`" column of the "`DispatchWorkers`" entity must exist as valid IDs in the "`EmployeeID`" column of the "`Employees`" entity.
3	"`WorkerSkills NVARCHAR(300)`" statement - Defines another column named "`WorkerSkills`" (data type: `NVARCHAR(300)`) to store the information of skills that the dispatch worker has.
4	"`WorkerCertifications NVARCHAR(100)`" statement - Defines a column named "`WorkerCertifications`" (data type: `NVARCHAR(100)`) to store the certifications the dispatch worker has.

Table 2.2.6 "Members" Entity

SQL Statements

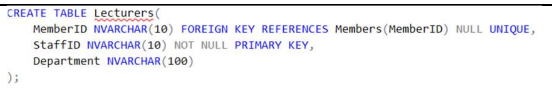
```
CREATE TABLE Members(
    MemberID NVARCHAR(10) NOT NULL PRIMARY KEY,
    MemberName NVARCHAR(100) NOT NULL,
    MemberBirthDate DATE,
    MemberGender NVARCHAR(20) CONSTRAINT validMemberGender CHECK(
        LOWER(MemberGender) IN('male', 'female', 'prefer not to say')
    ),
    RegisterDate DATETIME,
    Email NVARCHAR(100),
    EmailPassword NVARCHAR(100),
    Allergies NVARCHAR(100)
);
```

Explanation

1	"`CREATE TABLE Members`" statement - Used to create a new entity within the database "`DBM_Assignment`" and named the entity as "`Members`".
2	"`MemberID NVARCHAR(10) NOT NULL PRIMARY KEY`" statement - Defines the first column named "`MemberID`" (data type: `NVARCHAR(10)`) to store the member ID of the members in the "`Members`" entity. - "`NOT NULL`" is used to ensure that "`MemberID`" column cannot be empty. - "`PRIMARY KEY`" is used to designate "`MemberID`" as the primary key for the "`Members`" entity. A primary key uniquely identifies each row and it cannot contain duplicate values in the "`Members`" entity.
3	"`MemberBirthDate DATE`" statement - Defines a column named "`MemberBirthDate`" (data type: `DATE`) to store members' birth dates.
4	"`MemberGender NVARCHAR(20) CONSTRAINT validMemberGender CHECK(LOWER(MemberGender) IN('male', 'female', 'prefer not to say'))`" statement - Defines the last column named "`MemberGender`" (data type: `NVARCHAR(20)`) to store members' gender information in the "`Members`" entity. - "`CONSTRAINT validMemberGender CHECK(LOWER(MemberGender) IN('male', 'female', 'prefer not to say'))`" defines a constraint named "`validMemberGender`", which used the "`CHECK`" clause to ensure that the value stored in the "`MemberGender`" column must be either `'male'`, `'female'` or `'prefer not to say'`.

5	"`RegisterDate DATETIME`" statement - Defines a column named "`RegisterDate`" (data type: `DATETIME`) to store the information of the member's register date.
6	"`Email NVARCHAR(100)`" statement - Defines a column named "`Email`" (data type: `NVARCHAR(100)`) to store the members' email.
7	"`EmailPassword NVARCHAR(100)`" statement - Defines a column named "`EmailPassword`" (data type: `NVARCHAR(100)`) to store the password of the members' email.
8	"`Allergies NVARCHAR(100)`" statement - Defines the last column named "`Allergies`" (data type: `NVARCHAR(100)`) to store allergies the member may have.

Table 2.2.7 "Lecturers" Entity

SQL Statements	
	
Explanation	
1	"`CREATE TABLE Lecturers`" statement - Used to create a new entity within the database "`DBM_Assignment`" and named the entity as "`Lecturers`".
2	"`MemberID NVARCHAR(10) FOREIGN KEY REFERENCES Members(MemberID) NULL UNIQUE`" statement - Defines the first column named "`MemberID`" (data type: `NVARCHAR(10)`) to store the member ID of the cart in the "`Lecturers`" entity. - "`FOREIGN KEY REFERENCES Members(MemberID)`" is used to establish a foreign key relationship, which ensures the value in the "`MemberID`" column of the "`Lecturers`" entity must exist as valid member IDs in the "`MemberID`" column of the "`Members`" entity. - "`NULL`" allows the "`MemberID`" column to have empty values. - "`UNIQUE`" is used to ensure no duplicate member IDs exist in the "`Lecturers`" entity. Each lecturer must have a unique member ID.

3	"`StaffID NVARCHAR(10) NOT NULL PRIMARY KEY`" statement - Defines a column named "`StaffID`" (data type: `NVARCHAR(10)`) to store lecturers' ID. - "`NOT NULL`" is used to ensure a value is always provided for the staff's ID. - "`PRIMARY KEY`" is used to designate "`StaffID`" as the primary key for the "`Lecturers`" entity. A primary key uniquely identifies each row and it cannot contain duplicate values in the "`Lecturers`" entity.
4	"`Department NVARCHAR(100)`" statement - Defines a column named "`Department`" (data type: `NVARCHAR(100)`) to store the department that the lecturer belongs to.

Table 2.2.8 "Students" Entity

SQL Statements
`CREATE TABLE Students(` `MemberID nvarchar(10) FOREIGN KEY REFERENCES Members(MemberID) NULL UNIQUE,` `TPNumber nvarchar(10) NOT NULL PRIMARY KEY,` `Course nvarchar(100),` `LevelOfStudies nvarchar(100)` `);`
Explanation
1 "`CREATE TABLE Students`" statement - Used to create a new entity within the database "`DBM_Assignment`" and named the entity as "`Students`".
2 "`MemberID nvarchar(10) FOREIGN KEY REFERENCES Members(MemberID) NULL UNIQUE`" statement - Defines the first column named "`MemberID`" (data type: `NVARCHAR(10)`) to store the member ID of the students in the "`Students`" entity. - "`FOREIGN KEY REFERENCES Members(MemberID)`" is used to establish a foreign key relationship, which ensures the value in the "`MemberID`" column of the "`Students`" entity must exist as valid member IDs in the "`MemberID`" column of the "`Members`" entity. - "`NULL`" allows the "`MemberID`" column to have empty values. - "`UNIQUE`" is used to ensure no duplicate member IDs exist in the " `Students` " entity. Each student must have a unique member ID.

3	"`TPNumber nvarchar(10) NOT NULL PRIMARY KEY`" the statement - Defines a column named "`TPNumber`" (data type: `NVARCHAR(10)`) to store students' ID. - "`NOT NULL`" is used to ensure a value is always provided for the student's ID. - "`PRIMARY KEY`" is used to designate "`TPNumber`" as the primary key for the "`Students`" entity. A primary key uniquely identifies each row and it cannot contain duplicate values in the "`Students`" entity.
4	"`Course nvarchar(100)`" statement - Defines a column named "`Course`" (data type: `NVARCHAR(100)`) to store the course that the student belongs to.
5	"`LevelOfStudies nvarchar(100)`" statement - Defines the last column named "`LevelOfStudies`" (data type: `NVARCHAR(100)`) to store the student's level of study.

Table 2.2.9 "ShoppingCarts" Entity

SQL Statements	
``CREATE TABLE ShoppingCarts(CartID NVARCHAR(10) NOT NULL PRIMARY KEY, MemberID NVARCHAR(10) NOT NULL FOREIGN KEY REFERENCES Members(MemberID), CartDateTime DATETIME);```	
Explanation	
1	"`CREATE TABLE ShoppingCarts`" statement - Used to create a new entity within the `DBM_Assignment`" and named the entity as "`ShoppingCarts`".
2	"`CartID NVARCHAR(10) NOT NULL PRIMARY KEY`" statement - Defines the first column named "`CartID`" (data type: `NVARCHAR(10)`) to store the cart ID of the shopping carts in the entity "`ShoppingCarts`". - "`NOT NULL`" is used to ensure that "`CartID`" column cannot be empty. - "`PRIMARY KEY`" is used to designate "`CartID`" as the primary key for the "`ShoppingCarts`" entity. A primary key uniquely identifies each row and it cannot contain duplicate values in the "`ShoppingCarts`" entity.
3	"`MemberID NVARCHAR(10) NOT NULL FOREIGN KEY REFERENCES Members(MemberID)`" statement

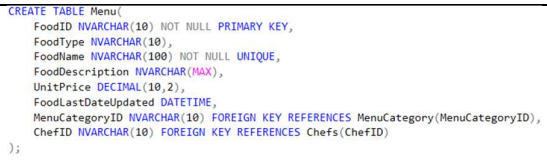
	- Defines a column named "`MemberID`" (data type: `NVARCHAR(10)`) to store the member ID of the cart in the entity "`ShoppingCarts`". - "`NOT NULL`" is used to ensure that "`MemberID`" column must have a value. - "`FOREIGN KEY REFERENCES Members(MemberID)`" is used to establish a foreign key relationship, which ensures the value in the "`MemberID`" column of the "`ShoppingCarts`" entity must exist as valid member IDs in the "`MemberID`" column of the "`Members`" entity.
4	"`CartDateTime DATETIME`" statement - Defines the last column named "`CartDateTime`" (data type: `DATETIME`) to store the information of the date and time that the carts created.

Table 2.2.10 "MenuCategory" Entity

SQL Statements	
`` CREATE TABLE MenuCategory(MenuCategoryID NVARCHAR(10) NOT NULL PRIMARY KEY, CategoryName NVARCHAR(50) NOT NULL UNIQUE, CategoryLastDateUpdated DATETIME); ```	
Explanation	
1	"`CREATE TABLE MenuCategory`" statement - Used to create a new entity within the database "`DBM_Assignment`" and named the entity as "`MenuCategory`".
2	"`MenuCategoryID NVARCHAR(10) NOT NULL PRIMARY KEY`" statement - Defines the first column named "`MenuCategoryID`" (data type: `NVARCHAR(10)`) to store the menu category ID of the menu in the entity "`MenuCategory`". - "`NOT NULL`" is used to ensure that "`MenuCategoryID`" column cannot be empty. - "`PRIMARY KEY`" is used to designate "`MenuCategoryID`" as the primary key for the "`MenuCategory`" entity. A primary key uniquely identifies each row and it cannot contain duplicate values in the "`MenuCategory`" entity.
3	"`CategoryName NVARCHAR(50) NOT NULL UNIQUE`" statement - Defines a column named "`CategoryName`" (data type: `NVARCHAR(10)`) to store category's names. - "`NOT NULL`" is used to ensure that "`CategoryName`" column must have a value.

	- "UNIQUE" is used to ensure no duplicate category name exist in the "MenuCategory" entity. Each category name must have a distinct name.
4	"CategoryLastDateUpdated DATETIME" statement - Defines a column named "CategoryLastDateUpdated" (data type: DATETIME) to store the information of the date and time that the category latest updated.

Table 2.2.11 "Menu" Entity

SQL Statements	
	
Explanation	
1	"CREATE TABLE Menu" statement - Used to create a new entity within the database "DBM_Assignment" and named the entity as "Menu".
2	"FoodID NVARCHAR(10) NOT NULL PRIMARY KEY" statement - Defines the first column named "FoodID" (data type: NVARCHAR(10)) to store the Food ID of the food in the entity "Menu". - "NOT NULL" is used to ensure that "FoodID" column must have a value. - "PRIMARY KEY" is used to designate "FoodID" as the primary key for the "Menu" entity. A primary key uniquely identifies each row and it cannot contain duplicate values in the "Menu" entity.
3	"FoodType NVARCHAR(10)" statement - Defines the next column named "FoodType" (data type: NVARCHAR(10)) to store the information of the type of foods.
4	"FoodName NVARCHAR(100) NOT NULL UNIQUE" statement - Defines a column named "FoodName" (data type: NVARCHAR(100)) to store foods' names. - "NOT NULL" is used to ensure that "FoodName" column must have a value. - "UNIQUE" is used to ensure no duplicate food names exist in the "Menu" entity. Each food name must have a distinct name.

5	"`FoodDescription NVARCHAR(MAX)`" statement - Defines a column named "`FoodDescription`" (data type: `NVARCHAR(MAX)`) to store the description of each food.
6	"`UnitPrice DECIMAL(10,2)`" statement - Defines a column named "`UnitPrice`" (data type: `DECIMAL(10,2)`) to store the unit price of each food.
7	"`FoodLastDateUpdated DATETIME`" statement - Defines a column named "`FoodLastDateUpdated`" (data type: `DATETIME`) to store the information about the food item was last modified.
8	"`MenuCategoryID NVARCHAR(10) FOREIGN KEY REFERENCES MenuCategory(MenuCategoryID)`" statement - Defines a column named "`MenuCategoryID`" (data type: `NVARCHAR(10)`) to store the category of each food in the "`Menu`" entity. - "`FOREIGN KEY REFERENCES MenuCategory(MenuCategoryID)`" is used to establish a foreign key relationship, which ensures the value in the "`MenuCategoryID`" column of the "`Menu`" entity must exist as valid category IDs in the "`MenuCategoryID`" column of the "`MenuCategory`" entity.
9	"`ChefID NVARCHAR(10) FOREIGN KEY REFERENCES Chefs(ChefID)`" statement - Defines the last column named "`ChefID`" (data type: `NVARCHAR(10)`) to store the category of each chefs' ID in the "`Menu`" entity. - "`FOREIGN KEY REFERENCES Chefs(ChefID)`" is used to establish a foreign key relationship, which ensures the value in the "`ChefID`" column of the "`Menu`" entity must exist as valid chef IDs in the "`ChefID`" column of the "`Chefs`" entity.

Table 2.2.12 "CartItems" Entity

SQL Statements	
	
Explanation	
1	"`CREATE TABLE CartItems`" statement - Used to create a new entity within the database "`DBM_Assignment`" and named the entity as "`CartItems`".
2	"`CartID NVARCHAR(10) NOT NULL FOREIGN KEY REFERENCES ShoppingCarts(CartID)`" statement - Defines the first column named "`CartID`" (data type: `NVARCHAR(10)`) to store the `CartID` of the cart in the entity "`CartItems`". - "`NOT NULL`" is used to ensure that "`CartID`" column cannot be empty. - "`FOREIGN KEY REFERENCES ShoppingCarts(CartID)`" is used to establish a foreign key relationship, which ensures the value in the "`CartID`" column of the "`CartItems`" entity must exist as valid cart IDs in the "`CartID`" column of the "`ShoppingCarts`" entity.
3	"`FoodID NVARCHAR(10) NOT NULL FOREIGN KEY REFERENCES MENU(FoodID)`" statement - Defines a column named "`FoodID`" (data type: `NVARCHAR(10)`) to store the ID of each food. - "`NOT NULL`" is used to ensure that "`FoodID`" column must have a value. - "`FOREIGN KEY REFERENCES Menu(FoodID)`" is used to establish a foreign key relationship, which ensures the value in the "`FoodID`" column of the "`CartItems`" entity must exist as valid food IDs in the "`FoodID`" column of the "`Menu`" entity.
4	"`Quantity INT`" statement - Defines a column named "`Quantity`" (data type: `INT`) to store the quantity of the food.
5	"`ItemDateTime DATETIME`" statement - Defines the last column named "`ItemDateTime`" (data type: `DATETIME`) to store the date and time when a particular item was added to the cart.

6	"`PRIMARY KEY (CartID, FoodID)`" statement - Defines a composite primary key for the entity, a combination of a specific "`CartID`" and a specific "`FoodID`" uniquely identifies each row in this "`CartItems`" entity.

Table 2.2.13 "Payments" Entity

SQL Statements	
`` CREATE TABLE Payments(PaymentID NVARCHAR(10) NOT NULL PRIMARY KEY, MemberID NVARCHAR(10)NOT NULL FOREIGN KEY REFERENCES Members(MemberID), CartID NVARCHAR(10) NOT NULL FOREIGN KEY REFERENCES ShoppingCarts(CartID) UNIQUE, Amount DECIMAL(10, 2), PaymentDateTime DATETIME); ```	
Explanation	
1	"`CREATE TABLE Payments`" statement - Used to create a new entity within the database "`DBM_Assignment`" and named the entity as "`Payments`".
2	"`PaymentID NVARCHAR(10) NOT NULL PRIMARY KEY`" statement - Defines the first column named "`PaymentID`" (data type: `NVARCHAR(10)`) to store the Payment ID of each payment transaction in the "`Payments`" entity. - "`NOT NULL`" is used to ensure that "`PaymentID`" column must have a value. - "`PRIMARY KEY`" is used to designate "`PaymentID`" as the primary key for the "`Payments`" entity. A primary key uniquely identifies each row and it cannot contain duplicate values in the "`Payments`" entity.
3	"`MemberID NVARCHAR(10)NOT NULL FOREIGN KEY REFERENCES Members(MemberID)`" statement - Defines a column named "`MemberID`" (data type: `NVARCHAR(10)`) to store the Member ID of each member in the "`Payments`" entity. - "`NOT NULL`" is used to ensure that "`MemberID`" column cannot be empty. - "`FOREIGN KEY REFERENCES Members(MemberID)`" is used to establish a foreign key relationship, which ensures the value in the "`MemberID`" column of the "`Payments`" entity must exist as valid member IDs in the "`MemberID`" column of the "`Members`" entity.

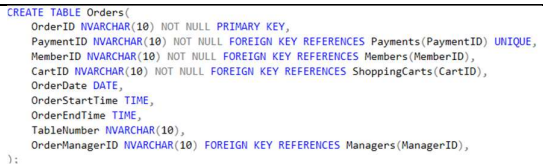
4	"`CartID NVARCHAR(10) NOT NULL FOREIGN KEY REFERENCES ShoppingCarts(CartID) UNIQUE`" statement - Defines a column named "`CartID`" (data type: `NVARCHAR(10)`) to store the Cart ID of each cart in the "`Payments`" entity. - "`NOT NULL`" is used to ensure that "`CartID`" column cannot be empty. - "`FOREIGN KEY REFERENCES ShoppingCarts(CartID)`" is used to establish a foreign key relationship, which ensures the value in the "`CartID`" column of the "`Payments`" entity must exist as valid cart IDs in the "`CartID`" column of the "`ShoppingCarts`" entity. - "`UNIQUE`" is used to ensure that a cart can only have one unique payment record based on cart ID at a time.
5	"`Amount DECIMAL(10, 2)`" statement - Defines a column named "`Amount`" (data type: `DECIMAL(10, 2)`) to store the payment amount.
6	"`PaymentDateTime DATETIME`" statement - Defines the last column named "`PaymentDateTime`" (data type: `DATETIME`) to store the date and time when the payment was made.

Table 2.2.14 "PaymentMethods" Entity

SQL Statements	
``CREATE TABLE PaymentMethods(PaymentID NVARCHAR(10) NOT NULL FOREIGN KEY REFERENCES Payments(PaymentID) UNIQUE, PaymentMethod NVARCHAR(50));```	
Explanation	
1	"`CREATE TABLE PaymentMethods`" statement - Used to create a new entity within the database "`DBM_Assignment`" and named the entity as "`PaymentMethods`".
2	"`PaymentID NVARCHAR(10) NOT NULL FOREIGN KEY REFERENCES Payments(PaymentID) UNIQUE`" statement - Defines the first column named "`PaymentID`" (data type: `NVARCHAR(10)`) to store the payment ID of each payment in the "`PaymentMethods`" entity. - "`NOT NULL`" is used to ensure that "`PaymentID`" column cannot be empty. - "`FOREIGN KEY REFERENCES Payments(PaymentID)`" is used to establish a foreign key relationship, which ensures the value in the "`PaymentID`" column of

	the "PaymentMethods" entity must exist as valid payment IDs in the "PaymentID" column of the "Payments" entity. - "UNIQUE" is used to ensure no duplicate payment IDs exist in the "PaymentMethods" entity. Each payment method must have a unique payment ID.
3	"PaymentMethod NVARCHAR(50)" statement - Defines the last column named "PaymentMethod" (data type: NVARCHAR(50)) to store the payment method of each payment.

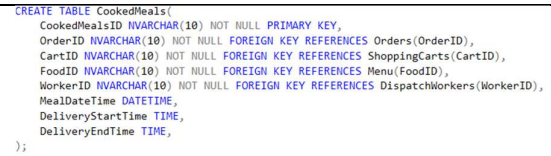
Table 2.2.15 "Orders" Entity

SQL Statements	
	
Explanation	
1	"CREATE TABLE Orders" statement - Used to create a new entity within the database "DBM_Assignment" and named the entity as "Orders".
2	"OrderID NVARCHAR(10) NOT NULL PRIMARY KEY" statement - Defines the first column named "OrderID" (data type: NVARCHAR(10)) to store the Order ID of each order in the "Orders" entity. - "NOT NULL" is used to ensure that "OrderID" column must have a value. - "PRIMARY KEY" is used to designate "OrderID" as the primary key for the "Orders" entity. A primary key uniquely identifies each row and it cannot contain duplicate values in the "Orders" entity.
3	"PaymentID NVARCHAR(10) NOT NULL FOREIGN KEY REFERENCES Payments(PaymentID) UNIQUE" statement - Defines a column named "PaymentID" (data type: NVARCHAR(10)) to store the Payment ID of each payment in the "Orders" entity. - "NOT NULL" is used to ensure that "PaymentID" column cannot be empty. - "FOREIGN KEY REFERENCES Payments(PaymentID)" is used to establish a foreign key relationship, which ensures the value in the "PaymentID" column of

	the "`Orders`" entity must exist as valid payment IDs in the "`PaymentID`" column of the "`Payments`" entity. - "`UNIQUE`" is used to ensure no duplicate payment IDs exist in the "`Orders`" entity. Each payment method must have a unique payment ID.
4	"`MemberID NVARCHAR(10) NOT NULL FOREIGN KEY REFERENCES Members(MemberID)`" statement - Defines a column named "`MemberID`" (data type: `NVARCHAR(10)`) to store the Member ID of each member in the "`Orders`" entity. - "`NOT NULL`" is used to ensure that "`MemberID`" column must have a value. - "`FOREIGN KEY REFERENCES Members(MemberID)`" is used to establish a foreign key relationship, which ensures the value in the "`MemberID`" column of the "`Orders`" entity must exist as valid member IDs in the "`MemberID`" column of the "`Members`" entity.
5	"`CartID NVARCHAR(10) NOT NULL FOREIGN KEY REFERENCES ShoppingCarts(CartID)`" statement - Defines a column named "`CartID`" (data type: `NVARCHAR(10)`) to store the Cart ID of each member in the "`Orders`" entity. - "`NOT NULL`" is used to ensure that "`CartID`" column must have a value. - "`FOREIGN KEY REFERENCES ShoppingCarts(CartID)`" is used to establish a foreign key relationship, which ensures the value in the "`CartID`" column of the "`Orders`" entity must exist as valid cart IDs in the "`CartID`" column of the "`ShoppingCarts`" entity.
6	"`OrderDate DATE`" statement - Defines a column named "`OrderDate`" (data type: `DATE`) to store the date the order was placed.
7	"`OrderStartTime TIME`" statement - Defines a column named "`OrderStartTime`" (data type: `TIME`) to store the start time the order was placed.
8	"`OrderEndTime TIME`" statement - Defines a column named "`OrderEndTime`" (data type: `TIME`) to store the end time the order was completed.

9	"`TableNumber NVARCHAR(10)`" statement - Defines a column named "`TableNumber`" (data type: `NVARCHAR(10)`) to store the table number of each order.
10	"`OrderManagerID NVARCHAR(10) FOREIGN KEY REFERENCES Managers(ManagerID)`" statement - Defines the last column named "`OrderManagerID`" (data type: `NVARCHAR(10)`) to store the Order Manager ID of the manager in the "`Orders`" entity. - "`FOREIGN KEY REFERENCES Managers(ManagerID)`" is used to establish a foreign key relationship, which ensures the value in the "`OrderManagerID`" column of the "`Orders`" entity must exist as valid manager IDs in the "`ManagerID`" column of the "`Managers`" entity.

Table 2.2.16 "CookedMeals" Entity

SQL Statements	
	
Explanation	
1	"`CREATE TABLE CookedMeals`" statement - Used to create a new entity within the database "`DBM_Assignment`" and named the entity as "`CookedMeals`".
2	"`CookedMealsID NVARCHAR(10) NOT NULL PRIMARY KEY`" statement - Defines the first column named "`CookedMealsID`" (data type: `NVARCHAR(10)`) to store the Cooked Meals ID of each cooked meal in the "`CookedMeals`" entity. - "`NOT NULL`" is used to ensure that "`CookedMealsID`" column must have a value. - "`PRIMARY KEY`" is used to designate "`CookedMealsID`" as the primary key for the "`CookedMeals`" entity. A primary key uniquely identifies each row and it cannot contain duplicate values in the "`CookedMeals`" entity.
3	"`OrderID NVARCHAR(10) NOT NULL FOREIGN KEY REFERENCES Orders(OrderID)`" statement

	- Defines a column named "`OrderID`" (data type: `NVARCHAR(10)`) to store the Order ID of the orders in the entity "`CookedMeals`". - "`NOT NULL`" is used to ensure that "`OrderID`" column must have a value. - "`FOREIGN KEY REFERENCES Orders (OrderID)`" is used to establish a foreign key relationship, which ensures the value in the "`OrderID`" column of the "`CookedMeals`" entity must exist as valid order IDs in the "`OrderID`" column of the "`Orders`" entity.
4	"`CartID NVARCHAR(10) NOT NULL FOREIGN KEY REFERENCES ShoppingCarts(CartID)`" statement - Defines a column named "`CartID`" (data type: `NVARCHAR(10)`) to store the Cart ID of the cart in the entity "`CookedMeals`". - "`NOT NULL`" is used to ensure that "`CartID`" column must have a value. - "`FOREIGN KEY REFERENCES ShoppingCarts(CartID)`" is used to establish a foreign key relationship, which ensures the value in the "`CartID`" column of the "`CookedMeals`" entity must exist as valid cart IDs in the "`CartID`" column of the "`ShoppingCarts`" entity.
5	"`FoodID NVARCHAR(10) NOT NULL FOREIGN KEY REFERENCES Menu(FoodID)`" statement - Defines a column named "`FoodID`" (data type: `NVARCHAR(10)`) to store the Food ID of the food in the entity "`CookedMeals`". - "`NOT NULL`" is used to ensure that "`FoodID`" column must have a value. - "`FOREIGN KEY REFERENCES Menu(FoodID)`" is used to establish a foreign key relationship, which ensures the value in the "`FoodID`" column of the "`CookedMeals`" entity must exist as valid food IDs in the "`FoodID`" column of the "`Menu`" entity.
6	"`WorkerID NVARCHAR(10) NOT NULL FOREIGN KEY REFERENCES DispatchWorkers(WorkerID)`" statement - Defines a column named "`WorkerID`" (data type: `NVARCHAR(10)`) to store the Worker ID of each worker in the "`CookedMeals`" entity. - "`NOT NULL`" is used to ensure that "`WorkerID`" column must have a value. - "`FOREIGN KEY REFERENCES DispatchWorkers(WorkerID)`" is used to establish a foreign key relationship, which ensures the value in the "`WorkerID`"

	column of the "`CookedMeals`" entity must exist as valid worker IDs in the "`WorkerID`" column of the "`DispatchWorkers`" entity.
7	"`MealDateTime DATETIME`" statement - Defines a column named "`MealDateTime`" (data type: `DATETIME`) to store the date and time the meal was prepared.
8	"`DeliveryStartTime TIME`" statement - Defines a column named "`DeliveryStartTime`" (data type: `TIME`) to store the time the delivery was started.
9	"`DeliveryEndTime TIME`" statement - Defines the last column named "`DeliveryEndTime`" (data type: `TIME`) to store the end time the delivery was completed.

Table 2.2.17 "OrderStatus" Entity

SQL Statements	
	
Explanation	
1	"`CREATE TABLE OrderStatus`" statement - Used to create a new entity within the database "`DBM_Assignment`" and named the entity as "`OrderStatus`".
2	"`MemberID NVARCHAR(10) NOT NULL FOREIGN KEY REFERENCES Members(MemberID)`" statement - Defines the first column named "`MemberID`" (data type: `NVARCHAR(10)`) to store the Member ID of each member in the entity "`OrderStatus`". - "`NOT NULL`" is used to ensure that "`MemberID`" column must have a value. - "`FOREIGN KEY REFERENCES Members(MemberID)`" is used to establish a foreign key relationship, which ensures the value in the "`MemberID`" column of the "`OrderStatus`" entity must exist as valid member IDs in the "`MemberID`" column of the "`Members`" entity.
3	"`OrderID NVARCHAR(10) NOT NULL FOREIGN KEY REFERENCES Orders(OrderID) UNIQUE`" statement - Defines a column named "`OrderID`" (data type: `NVARCHAR(10)`) to store the order ID of the orders in the entity "`OrderStatus`".

	- "`NOT NULL`" is used to ensure that "`OrderID`" column must have a value. - "`FOREIGN KEY REFERENCES Orders(OrderID)`" is used to establish a foreign key relationship, which ensures the value in the "`OrderID`" column of the "`OrderStatus`" entity must exist as valid order IDs in the "`OrderID`" column of the "`Orders`" entity. - "`UNIQUE`" is used to ensure no duplicate order IDs exist in the "`OrderStatus`" entity. Each order status must have a unique order ID.
4	"`CartID NVARCHAR(10) NOT NULL FOREIGN KEY REFERENCES ShoppingCarts(CartID) UNIQUE`" statement - Defines a column named "`CartID`" (data type: `NVARCHAR(10)`) to store the Cart ID of the cart in the entity "`OrderStatus`". - "`NOT NULL`" is used to ensure that "`MemberID`" column must have a value. - "`FOREIGN KEY REFERENCES ShoppingCarts(CartID)`" is used to establish a foreign key relationship, which ensures the value in the "`CartID`" column of the "`OrderStatus`" entity must exist as valid cart IDs in the "`CartID`" column of the "`ShoppingCarts`" entity. - "`UNIQUE`" is used to ensure no duplicate cart IDs exist in the "`OrderStatus`" entity. Each order status must have a unique cart ID.
5	"`DeliveryStatus NVARCHAR(20) CONSTRAINT validStatus CHECK(DeliveryStatus IN('Pending', 'Confirmed', 'Waiting', 'Read', 'Preparing', 'Completed'))`" statement - Defines the last column named "`DeliveryStatus`" (data type: `NVARCHAR(20)`) to store the status of the delivery in the "`OrderStatus`" entity. - "`CONSTRAINT validStatus CHECK(DeliveryStatus IN('Pending', 'Confirmed', 'Waiting', 'Read', 'Preparing', 'Completed'))`" defines a constraint named "`validStatus`", which used the "`CHECK`" clause to ensure that the value stored in the "`DeliveryStatus`" column must be either `'Pending', 'Confirmed', 'Waiting', 'Read', 'Preparing'` or `'Completed'`.

6	"`PRIMARY KEY (MemberID, OrderID, CartID)`" statement - Defines a composite primary key for the entity, a combination of a specific " `MemberID` ", a specific "`OrderID`" and a specific "`CartID`" uniquely identifies each row in this "`OrderStatus`" entity.

Table 2.2.18 "Feedbacks" Entity

SQL Statements	
`` CREATE TABLE Feedbacks(FeedbackID NVARCHAR(10) NOT NULL PRIMARY KEY, MemberID NVARCHAR(10) NOT NULL FOREIGN KEY REFERENCES Members(MemberID), OrderID NVARCHAR(10) NOT NULL FOREIGN KEY REFERENCES Orders(OrderID), FoodID NVARCHAR(10) NOT NULL FOREIGN KEY REFERENCES Menu(FoodID), Rating INT CONSTRAINT validRating CHECK(Rating BETWEEN 1 AND 5), Review NVARCHAR(MAX), FeedbackDate DATE); ```	
Explanation	
1	"`CREATE TABLE Feedbacks`" statement - Used to create a new entity within the database "`DBM_Assignment`" and named the entity as "`Feedbacks`".
2	"`FeedbackID NVARCHAR(10) NOT NULL PRIMARY KEY`" statement - Defines the first column named "`FeedbackID`" (data type: `NVARCHAR(10)`) to store the Feedback ID of each feedbacks in the "`Feedbacks`" entity. - "`NOT NULL`" is used to ensure that "`FeedbackID`" column must have a value. - "`PRIMARY KEY`" is used to designate "`FeedbackID`" as the primary key for the "`Feedbacks`" entity. A primary key uniquely identifies each row and it cannot contain duplicate values in the "`Feedbacks`" entity.
3	"`MemberID NVARCHAR(10) NOT NULL FOREIGN KEY REFERENCES Members(MemberID)`" statement - Defines a column of the "`Feedbacks`" entity named "`MemberID`" (data type: `NVARCHAR(10)`) to store the Member ID of each member in the entity "`Feedbacks`". - "`NOT NULL`" is used to ensure that "`MemberID`" column must have a value. - "`FOREIGN KEY REFERENCES Members(MemberID)`" is used to establish a foreign key relationship, which ensures the value in the "`MemberID`" column of the "`Feedbacks`" entity must exist as valid member IDs in the "`MemberID`" column of the "`Members`" entity.

4	"`OrderID NVARCHAR(10) NOT NULL FOREIGN KEY REFERENCES Orders(OrderID)`" statement - Defines a column named "`OrderID`" (data type: `NVARCHAR(10)`) to store the order ID of the orders in the "`Feedbacks`" entity. - "`NOT NULL`" is used to ensure that "`OrderID`" column must have a value. - "`FOREIGN KEY REFERENCES Orders(OrderID)`" is used to establish a foreign key relationship, which ensures the value in the "`OrderID`" column of the "`Feedbacks`" entity must exist as valid order IDs in the "`OrderID`" column of the "`Orders`" entity.
5	"`FoodID NVARCHAR(10) NOT NULL FOREIGN KEY REFERENCES Menu(FoodID)`" statement - Defines a column named "`FoodID`" (data type: `NVARCHAR(10)`) to store the Food ID of the food in the "`Feedbacks`" entity. - "`NOT NULL`" is used to ensure that "`FoodID`" column must have a value. - "`FOREIGN KEY REFERENCES Menu(FoodID)`" is used to establish a foreign key relationship, which ensures the value in the "`FoodID`" column of the "`Feedbacks`" entity must exist as valid food IDs in the "`FoodID`" column of the "`Menu`" entity.
6	"`Rating CONSTRAINT validRating CHECK(Rating BETWEEN 1 AND 5)`" statement - Defines a column named "`Rating`" (data type: `INT`) to store the ratings of the order in the "`Feedbacks`" entity. - "`CONSTRAINT validRating CHECK(Rating BETWEEN 1 AND 5)`" defines a constraint named "`validRating`", which used the "`CHECK`" clause to ensure that the integer value stored in the "`Rating`" column must be between 1 to 5.
7	"`Review NVARCHAR(MAX)`" statement - Defines a column named "`Review`" (data type: `NVARCHAR(MAX)`) to store the review of each order.
8	"`FeedbackDate DATE`" statement - Defines the last column named "`FeedbackDate`" (data type: `DATE`) to store the information of the date and time that the feedback created.

3.0 SQL – Data Manipulation Language (DML)

3.1 Insert Data

Table 3.1 Insert Data into Each Entity

<table>
<tr><th colspan="2">SQL Statements</th></tr>
<tr><td>Insert values into Employees entity:


```
INSERT INTO Employees VALUES
    ('MID01', '2004-02-01', 'Rowena.D', '0126455455', '1968-05-14', 'Female'),
    ('CFID01', '2004-07-11', 'Wesley.S', '0126533566', '1971-03-29', 'Male'),
    ('CFID02', '2004-09-23', 'Susana.R', '0173264595', '1974-08-21', 'Female'),
    ('CFID03', '2006-10-17', 'Otto.A', '0143962525', '1973-02-08', 'Male'),
    ('WID01', '2004-10-20', 'Elena.R', '0114888923', '1996-11-24', 'Female'),
    ('WID02', '2004-11-18', 'Rob.E', '0196587423', '1994-07-05', 'Male'),
    ('WID03', '2006-11-24', 'Joseph.B', '0162354815', '1998-10-09', 'Male');
```

</td><td>Insert values into Managers entity:


```
INSERT INTO Managers VALUES
    ('MID01', '2004', '18 Years', 'Master of Business Administration (MBA)');
```

</td></tr>
<tr><td>Insert values into Chefs entity:


```
INSERT INTO Chefs VALUES
    ('CFID01', 'Hot Kitchen', 'Cooking techniques like frying, stir-frying, and grilling', '9 Years', 'Culinary Arts Diploma'),
    ('CFID02', 'Beverage', 'Attention to detail for ensuring consistent drink quality and presentation', '5 Years', 'Food Safety and Sanitation Certification'),
    ('CFID03', 'Light Meal', 'Experience with preparing breakfast dishes', '7 Years', 'Basic Culinary Skills Certification');
```

</td><td>Insert values into DispatchWorkers entity:


```
INSERT INTO DispatchWorkers VALUES
    ('WID01', 'Problem-Solving Skills', 'Customer Service Certification'),
    ('WID02', 'Time Management and Organization', 'Delivery Dispatch Certification'),
    ('WID03', 'Communication Skills', 'Customer Service Certification');
```

</td></tr>
<tr><td>Insert values into Members entity:


```
INSERT INTO Members VALUES
    ('MB0001', 'Ali', '1994-10-08', 'Male', '2023-01-10 08:09:10.123', 'ali08@gmail.com', 'rduferie0', NULL),
    ('MB0002', 'Rainbow', '1999-05-14', 'Female', '2023-01-20 14:22:11.124', 'bow20@gmail.com', 'dAi! mR3', 'Peanut'),
    ('MB0003', 'Sarah', '1965-03-20', 'Female', '2023-01-27 13:09:56.243', 'si714@gmail.com', 'S@r13', 'Seafood'),
    ('MB0004', 'Jason', '1985-02-14', 'Prefer not to say', '2023-02-15 10:12:12.453', 'ji71n@gmail.com', 'JaSaN2003', NULL),
    ('MB0005', 'K.Vince', '2004-12-11', 'Male', '2023-02-21 16:44:04.444', 'kv0566@gmail.com', 'pP4#5jjC', 'Milk'),
    ('MB0006', 'MacHill', '1975-04-08', 'Male', '2023-02-24 4:44:04.448', 'ah0008@gmail.com', 'MBssw073', 'Peach'),
    ('MB0007', 'J.Wenton', '1994-09-24', 'Prefer not to say', '2023-03-01 6:56:06.666', 'jw368@gmail.com', '3WI19943', 'Banana'),
    ('MB0008', 'Micheal', '2005-06-09', 'Female', '2023-03-15 18:46:06.668', 'mi988@gmail.com', 'tOd!) Msi', NULL),
    ('MB0009', 'Shanan', '1977-11-11', 'Female', '2023-03-26 17:33:13.426', 'shan1@gmail.com', 'sMarTmE844', NULL),
    ('MB0010', 'S.Hanson', '1971-07-16', 'Male', '2023-03-05 15:23:45.231', 'sh123@gmail.com', 'IdkbAat1971', 'Seafood');
```

</td><td>Insert values into Lecturers entity:


```
INSERT INTO Lecturers VALUES
    ('MB0003', 'EMP100023', 'Academic Administration'),
    ('MB0004', 'EMP102054', 'Finance'),
    ('MB0006', 'EMP100889', 'School of Business'),
    ('MB0009', 'EMP107888', 'School of Technology'),
    ('MB0010', 'EMP104566', 'Human Resources'),
    (NULL, 'EMP103893', 'Psychology');
```

</td></tr>
<tr><td>Insert values into Students entity:


```
INSERT INTO Students VALUES
    ('MB0001', 'TP075966', 'Account', 'Master'),
    ('MB0002', 'TP054411', 'Software Engineering', 'PhD'),
    ('MB0005', 'TP045533', 'Design and Media', 'Diploma'),
    ('MB0007', 'TP062248', 'Data Informatics', 'Diploma'),
    ('MB0008', 'TP088422', 'Cybersecurity', 'Degree'),
    (NULL, 'TP645834', 'Artificial Intelligence', 'Degree');
```

</td><td>Insert values into ShoppingCarts entity:


```
INSERT INTO ShoppingCarts VALUES
    ('CID001', 'MB0005', '2023-03-15 11:29:00.123'),
    ('CID002', 'MB0002', '2023-03-20 13:02:10.453'),
    ('CID003', 'MB0006', '2023-03-30 12:07:09.001'),
    ('CID004', 'MB0008', '2023-07-02 11:44:34.907'),
    ('CID005', 'MB0010', '2023-07-09 12:59:10.291'),
    ('CID006', 'MB0007', '2023-07-18 12:12:37.241'),
    ('CID007', 'MB0003', '2024-06-06 11:37:02.874'),
    ('CID008', 'MB0001', '2024-06-06 12:16:23.542'),
    ('CID009', 'MB0005', '2024-06-06 12:27:13.379'),
    ('CID010', 'MB0009', '2024-06-06 13:02:43.124'),
    ('CID011', 'MB0007', '2024-06-06 13:41:29.431');
```

</td></tr>
<tr><td>Insert values into MenuCategory entity:


```
INSERT INTO MenuCategory VALUES
    ('MC01', 'Malay', '2023-01-15'),
    ('MC02', 'Chinese', '2023-01-15'),
    ('MC03', 'India', '2023-01-15'),
    ('MC04', 'Western', '2023-01-15'),
    ('MC05', 'Dessert', '2023-01-15'),
    ('MC06', 'Beverage', '2023-01-15');
```

</td><td>Insert values into Menu entity:


```
INSERT INTO Menu VALUES
    ('F001', 'Food', 'Nasi Lemak', 'Nasi lemak with Kacang, Ikan Bilis, Cucumber, Half Hard-Boiled Egg & Sambal.', 3.00, '2024-03-15', 'MC01', 'CFID01'),
    ('F002', 'Food', 'Nasi Goreng', 'Rice that is stir-fried with soy sauce, beaten egg, chopped meat, and vegetables.', 5.00, '2024-03-15', 'MC02', 'CFID01'),
    ('F003', 'Food', 'Roti Canai', 'A flatbread made from dough that is composed of fat, flour, and water.', 1.50, '2024-03-15', 'MC03', 'CFID01'),
    ('F004', 'Food', 'Waffles', 'Crisp raised cake baked in a waffle iron.', 4.50, '2024-03-15', 'MC05', 'CFID03'),
    ('F005', 'Food', 'Fried Noodle', 'Noodles stir-fried in cooking oil with garlic, onion, chicken, chili, and cabbages.', 6.00, '2024-03-15', 'MC02', 'CFID01'),
    ('F006', 'Food', 'Spaghetti Bolognese', 'Spaghetti with meat sauce made with minced beef, and tomatoes, served with Parmesan cheese.', 8.50, '2024-03-15', 'MC04', 'CFID01'),
    ('F007', 'Food', 'Chicken Chop', 'Marinated breaded boneless chicken that is fried to golden perfection, then drowned in a sauce.', 9.00, '2024-03-15', 'MC04', 'CFID01'),
    ('F008', 'Drink', 'Milk Tea', 'Tea with milk added.', 3.90, '2024-03-15', 'MC06', 'CFID02'),
    ('F009', 'Drink', 'Kopi', 'A beverage brewed from roasted coffee beans.', 2.00, '2024-03-15', 'MC06', 'CFID02'),
    ('F010', 'Drink', 'Tea', 'A beverage produced by steeping in freshly boiled water the tea pack.', 2.00, '2024-03-15', 'MC06', 'CFID02');
```

</td></tr>
</table>

Insert values into CartItems entity:

```
INSERT INTO CartItems VALUES
    ('CID001', 'F001', 2, '2023-03-15 11:30:31.010'),
    ('CID001', 'F009', 3, '2023-03-15 11:31:32.012'),
    ('CID001', 'F004', 1, '2023-03-15 11:31:50.210'),
    ('CID002', 'F006', 2, '2023-03-20 13:04:05.012'),
    ('CID002', 'F008', 4, '2023-03-20 13:05:01.002'),
    ('CID003', 'F007', 3, '2023-03-30 12:07:53.101'),
    ('CID003', 'F010', 2, '2023-03-30 12:08:32.124'),
    ('CID004', 'F002', 1, '2023-07-02 11:45:23.672'),
    ('CID005', 'F007', 3, '2023-07-09 13:00:43.001'),
    ('CID006', 'F006', 2, '2023-07-18 12:13:16.118'),
    ('CID006', 'F004', 1, '2023-07-18 12:13:50.001'),
    ('CID007', 'F001', 2, '2024-06-06 11:38:01.245'),
    ('CID008', 'F003', 1, '2024-06-06 12:17:08.224'),
    ('CID008', 'F009', 1, '2024-06-06 12:18:23.108'),
    ('CID009', 'F002', 3, '2024-06-06 12:28:16.139'),
    ('CID010', 'F007', 2, '2024-06-06 13:03:23.193'),
    ('CID011', 'F006', 2, '2024-06-06 13:42:30.537');
```

Insert values into Payments entity:

```
INSERT INTO Payments VALUES
    ('PYID001', 'MB0005', 'CID001', 16.50, '2023-03-15 11:40:23.600'),
    ('PYID002', 'MB0002', 'CID002', 32.60, '2023-03-20 13:10:09.080'),
    ('PYID003', 'MB0006', 'CID003', 31.00, '2023-03-30 12:13:52.700'),
    ('PYID004', 'MB0008', 'CID004', 5.00, '2023-07-02 11:50:53.006'),
    ('PYID005', 'MB0010', 'CID005', 27.00, '2023-07-09 13:06:46.430'),
    ('PYID006', 'MB0007', 'CID006', 21.50, '2023-07-18 13:16:13.230'),
    ('PYID007', 'MB0003', 'CID007', 6.00, '2024-06-06 11:45:37:913'),
    ('PYID008', 'MB0001', 'CID008', 3.50, '2024-06-06 12:24:53:519'),
    ('PYID009', 'MB0005', 'CID009', 15.00, '2024-06-06 12:36:17:802'),
    ('PYID010', 'MB0007', 'CID010', 18.00, '2024-06-06 13:11:21:643'),
    ('PYID011', 'MB0007', 'CID011', 17.00, '2024-06-06 13:50:34:561');
```

Insert values into PaymentMethods entity:

```
INSERT INTO PaymentMethods VALUES
    ('PYID001', 'Pay at Counter'),
    ('PYID002', 'Online Banking'),
    ('PYID003', 'Pay at Counter'),
    ('PYID004', 'Online Banking'),
    ('PYID005', 'Online Banking'),
    ('PYID006', 'Pay at Counter'),
    ('PYID007', 'Online Banking'),
    ('PYID008', 'Pay at Counter'),
    ('PYID009', 'Pay at Counter'),
    ('PYID010', 'Online Banking'),
    ('PYID011', 'Online Banking');
```

Insert values into Orders entity:

```
INSERT INTO Orders VALUES
    ('OID001', 'PYID001', 'MB0005', 'CID001', '2023-03-15', '11:31:51.291', '11:35:01.551', 31, 'MID01'),
    ('OID002', 'PYID002', 'MB0002', 'CID002', '2023-03-20', '13:05:55.253', '13:09:51.591', 8, 'MID01'),
    ('OID003', 'PYID003', 'MB0006', 'CID003', '2023-03-30', '12:09:05.324', '12:10:21.223', 27, 'MID01'),
    ('OID004', 'PYID004', 'MB0008', 'CID004', '2023-07-02', '11:46:30.201', '11:48:47.112', 84, 'MID01'),
    ('OID005', 'PYID005', 'MB0010', 'CID005', '2023-07-09', '13:02:05.324', '13:04:56.888', 66, 'MID01'),
    ('OID006', 'PYID006', 'MB0007', 'CID006', '2023-07-18', '12:14:55.364', '12:15:41.666', 47, 'MID01'),
    ('OID007', 'PYID007', 'MB0003', 'CID007', '2024-06-06', '11:42:53.204', '11:43:31.125', 12, 'MID01'),
    ('OID008', 'PYID008', 'MB0001', 'CID008', '2024-06-06', '12:20:12.110', '12:21:40.523', 25, 'MID01'),
    ('OID009', 'PYID009', 'MB0005', 'CID009', '2024-06-06', '12:31:24.802', '12:33:12.316', 18, 'MID01'),
    ('OID010', 'PYID010', 'MB0007', 'CID010', '2024-06-06', '13:07:52.671', '13:09:03.632', 77, 'MID01'),
    ('OID011', 'PYID011', 'MB0007', 'CID011', '2024-06-06', '13:45:06.185', '13:47:36.472', 39, 'MID01');
```

Insert values into CookedMeals entity:

```
--cooked meal deliver within 15 min
INSERT INTO CookedMeals VALUES
    ('CMID001', 'OID001', 'CID001', 'F001', 'WID02', '11:50:08.901', '11:51:36.751', '12:02:11.567'),
    ('CMID002', 'OID001', 'CID001', 'F001', 'WID02', '11:50:08.901', '11:51:36.751', '12:02:11.567'),
    ('CMID003', 'OID001', 'CID001', 'F009', 'WID02', '11:50:08.901', '11:51:36.751', '12:02:11.567'),
    ('CMID004', 'OID001', 'CID001', 'F009', 'WID02', '11:50:08.901', '11:51:36.751', '12:02:11.567'),
    ('CMID005', 'OID001', 'CID001', 'F009', 'WID02', '11:50:08.901', '11:51:36.751', '12:02:11.567'),
    ('CMID006', 'OID001', 'CID001', 'F004', 'WID02', '11:50:08.901', '11:51:36.751', '12:02:11.567'),
    ('CMID007', 'OID002', 'CID002', 'F006', 'WID03', '13:18:36.256', '13:25:32.404', '13:55:46.356'),
    ('CMID008', 'OID002', 'CID002', 'F006', 'WID03', '13:18:36.256', '13:25:32.404', '13:55:46.356'),
    ('CMID009', 'OID002', 'CID002', 'F008', 'WID03', '13:18:36.256', '13:25:32.404', '13:55:46.356'),
    ('CMID010', 'OID002', 'CID002', 'F008', 'WID03', '13:18:36.256', '13:25:32.404', '13:55:46.356'),
    ('CMID011', 'OID002', 'CID002', 'F008', 'WID03', '13:18:36.256', '13:25:32.404', '13:55:46.356'),
    ('CMID012', 'OID002', 'CID002', 'F008', 'WID03', '13:18:36.256', '13:25:32.404', '13:55:46.356'),
    ('CMID013', 'OID003', 'CID003', 'F007', 'WID01', '12:25:38.468', '12:27:36.346', '12:39:24.676'),
    ('CMID014', 'OID003', 'CID003', 'F007', 'WID01', '12:25:38.468', '12:27:36.346', '12:39:24.676'),
    ('CMID015', 'OID003', 'CID003', 'F007', 'WID01', '12:25:38.468', '12:27:36.346', '12:39:24.676'),
    ('CMID016', 'OID003', 'CID003', 'F010', 'WID01', '12:25:38.468', '12:27:36.346', '12:39:24.676'),
    ('CMID017', 'OID003', 'CID003', 'F010', 'WID01', '12:25:38.468', '12:27:36.346', '12:39:24.676'),
    ('CMID018', 'OID004', 'CID004', 'F002', 'WID01', '12:00:56.672', '12:02:16.552', '12:09:34.542'),
    ('CMID019', 'OID005', 'CID005', 'F007', 'WID03', '13:15:56.996', '13:25:54.872', '13:43:34.756'),
    ('CMID020', 'OID005', 'CID005', 'F007', 'WID03', '13:15:56.996', '13:25:54.872', '13:43:34.756'),
    ('CMID021', 'OID005', 'CID005', 'F007', 'WID03', '13:15:56.996', '13:25:54.872', '13:43:34.756'),
    ('CMID022', 'OID006', 'CID006', 'F006', 'WID02', '12:24:34.243', '12:25:56.514', '12:33:13.871'),
    ('CMID023', 'OID006', 'CID006', 'F006', 'WID02', '12:24:34.243', '12:25:56.514', '12:33:13.871'),
    ('CMID024', 'OID006', 'CID006', 'F004', 'WID02', '12:24:34.243', '12:25:56.514', '12:33:13.871');
```

Insert values into OrderStatus entity:

```
INSERT INTO OrderStatus VALUES
    ('MB0005', 'OID001', 'CID001', 'Completed'),
    ('MB0002', 'OID002', 'CID002', 'Completed'),
    ('MB0006', 'OID003', 'CID003', 'Completed'),
    ('MB0008', 'OID004', 'CID004', 'Completed'),
    ('MB0010', 'OID005', 'CID005', 'Completed'),
    ('MB0007', 'OID006', 'CID006', 'Completed'),
    ('MB0003', 'OID007', 'CID007', 'Preparing'),
    ('MB0001', 'OID008', 'CID008', 'Read'),
    ('MB0005', 'OID009', 'CID009', 'Waiting'),
    ('MB0009', 'OID010', 'CID010', 'Confirmed'),
    ('MB0007', 'OID011', 'CID011', 'Pending');
```

Insert values into Feedbacks entity:

```
INSERT INTO Feedbacks VALUES
    ('FID001', 'MB0005', 'OID001', 'F001', 5, 'The Nasi Lemak is very nice and tasty =P', '2023-03-16'),
    ('FID002', 'MB0005', 'OID001', 'F009', 3, 'Just normal kopi', '2023-03-16'),
    ('FID003', 'MB0005', 'OID001', 'F004', 2, 'The waffle is too sweet for me!', '2023-03-16'),
    ('FID004', 'MB0002', 'OID002', 'F006', 1, 'Although the food is not bad, but the delivery is too slow and the food has been cold already :(', '2023-03-23'),
    ('FID005', 'MB0002', 'OID002', 'F008', 1, 'Although the food is not bad, but the delivery is too slow and the food has been cold already :(', '2023-03-23'),
    ('FID006', 'MB0006', 'OID003', 'F007', 4, 'The chicken chop is crispy and delicious :)', '2023-03-31'),
    ('FID007', 'MB0006', 'OID003', 'F010', 4, 'The tea is okay.', '2023-03-31'),
    ('FID008', 'MB0008', 'OID004', 'F002', 5, 'The nasi goreng is very delicious, I like the taste of it!', '2023-07-05'),
    ('FID009', 'MB0010', 'OID005', 'F007', 2, 'Very disappointed as the delivery is too slow, and promised that only 15 minutes required for delivery.', '2023-07-12'),
    ('FID010', 'MB0007', 'OID006', 'F006', 4, 'The spaghetti is tasty and the portion is very worth for it =P', '2023-07-21'),
    ('FID011', 'MB0007', 'OID006', 'F004', 4, 'The waffle also taste good.', '2023-07-21');
```

3.2 Triggers

Table 3.2.1.1 "count_cookedMeals" After Insert Triggers

```
CREATE TRIGGER count_cookedMeals ON CookedMeals
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        DECLARE @mealID nvarchar(10)
        DECLARE @cartID nvarchar(10)
        DECLARE @foodID nvarchar(10)
        DECLARE @validn INT
        DECLARE @n INT
        SELECT @mealID = INSERTED.CookedMealsID,
        @cartID = INSERTED.CartID, @foodID = INSERTED.FoodID
        FROM INSERTED

        SELECT @validn = Quantity
        FROM CartItems
        WHERE CartID = @cartID AND FoodID = @foodID

        SELECT @n = COUNT(FoodID)
        FROM CookedMeals
        WHERE CartID = @cartID AND FoodID = @foodID

        IF @n > @validn
        BEGIN
            DECLARE @EM NVARCHAR(100)
            DECLARE @El INT
            SET @EM ='Exceed Quantity Ordered Error: CartID = %s, FoodID = %s'
            SET @El = ERROR_LINE()
            RAISERROR(@EM, 16, 1, @cartID, @foodID);

            DELETE FROM CookedMeals --delete wrongly inserted record
            WHERE CookedMealsID = @mealID
        END
    END TRY
    BEGIN CATCH
        DECLARE @ErrorMessage NVARCHAR(4000)
        DECLARE @ErrorSeverity INT
        DECLARE @ErrorState INT
        DECLARE @ErrorLine INT

        SELECT
            @ErrorMessage = ERROR_MESSAGE(),
            @ErrorSeverity = ERROR_SEVERITY(),
            @ErrorState = ERROR_STATE(),
            @ErrorLine = ERROR_LINE()

        PRINT('An Error Occured. ' + @ErrorMessage)
    END CATCH
END
```

"count_cookedMeals" trigger is a "AFTER INSERT" trigger is created and enforced on the "CookedMeals" entity; it runs automatically in the background after data has been inserted into the "CookedMeals" entity. The purpose of this trigger is to capture the number of cooked meals for an order, ensuring no meal is cooked more than the quantity that is ordered, enforcing data accuracies.

Explanation	
1.	"NOCOUNT" is set on to allow the trigger's code to run in the background, which suppresses the "(X rows affected)" message.
2.	"TRY...CATCH" block is applied.
3.	Under the "TRY" statement: (a) 5 variables are declared, (a) @mealID (data type: NVARCHAR(10)): to store the primary key value of "CookedMealsID" of the inserted row. (b) @cartID (data type: NVARCHAR(10)): to store the attribute value of "CartID" of the inserted row. (c) @foodID (data type: NVARCHAR(10)): to store the attribute value of "FoodID" of the inserted row. (d) @validn (data type: INT): to store the valid quantity ordered for a specific food ordered, which is the attribute value of "Quantity" from "CartItems"

entity, where the attribute value of "CartID" equates to @cartID and attribute value of "FoodID" equates to @foodID.

(e) @n INT: to store the current number of a specific food cooked for a specific order using "COUNT()", which is the count of rows from the "CookedMeals" entity where the attribute value of "CartID" equates to @cartID and attribute value of "FoodID" equates to @foodID, after inserting the new row.

(b) Once the variables declared are assigned with their values using the "SELECT" statement, a "IF" statement is applied for error handling, to determine whether the current number of a specific food for a specific order exceeds the required quantity. If so, a user-defined error will be created to dealt with this issue.

(c) Under the "IF" statement:

(i) The condition for this "IF" statement is if the value of @n variable is greater than the value of @validn variable. If this condition is satisfied, the code under this statement will run.

(ii) If the condition is satisfied, two variables are declared to create the user-defined error,

- @EM (data type: NVARCHAR(100)): to store the error message specified by the user, that is 'Exceed Quantity Ordered Error: CartID = %s, FoodID = %s'.
- @El (data type: INT): to store the line number where the error has arisen. "ERROR_LINE()" is a system built-in function that is used to track the error's line number.

(iii) Once the variables are assigned with their values, "RAISERROR()" function, which is a system built-in function is used to create a custom error. To introduce this user-defined error to the system, specific information about the error like the error message (@msg_str), error severity (@severity), and error state (@state) are mandatory to be passed to this function's parameters as an arguments, so that the system can store these relevant information about this error into the "sys.messages" catalog view, which is the central repository for storing both system-defined and user-defined errors' details. This allows the system to generate appropriate error message in accordance with the information provided.

	- Via "`RAISERROR(@EM, 16, 1, @cartID, @foodID)`", `@EM` variable will be passed as an argument to the `@msg_str` parameter, 16 will be passed as an argument to the `@severity` parameter, while 1 will be passed as an argument to the `@state` parameter in the "`RAISERROR()`" function. The remaining two variables, which are `@cartID` and `@foodID` are variables to be substituted into the `@msg_str` parameter via the two "`%s`" in the `@EM` variable, which serves as a placeholder for string values. (iv) Once the error is raised, the newly inserted row that satisfied the condition of the "`IF`" statement is said to be erroneous and will therefore be deleted from the "`CookedMeals`" entity where the attribute value of "CookedMealID" is equates to the `@mealID`.
4.	If the "`IF`" statement under the "`TRY`" block is satisfied, indicating there is an error arose in the "`TRY`" block, it throws an exception for the "`CATCH`" block to deal with. The "`TRY`" block will be terminated, and the control is passed to the successive group of statements enclosed in the "`CATCH`" block, and these statements will be executed to generate the error message.
5.	On the contrary, if the "`IF`" statement under the "`TRY`" block is not satisfied, indicating there is no errors, the "`TRY`" block will be completed successfully with no exception being thrown, and the control will be passed to the "`END CATCH`" statement, hence the statement enclosed in the "`CATCH`" block will not run, no error message will appear.
6.	Under the "`CATCH`" block: (a) 4 variables are declared, - `@ErrorMessage` (data type: `NVARCHAR(4000)`): to store the error message by retrieving the user-defined error message kept in the "`sys.messages`" catalog view via the "`ERROR_MESSAGE()`" function. - `@ErrorSeverity` (data type: `INT`): to store the error severity by referencing the user-specified error severity that is given through the "`RAISERROR()`" function via the "`ERROR_SEVERITY()`" system built-in function. - `@ErrorState` (data type: `INT`): to store the error state by referencing the user-specified error state that is given through the "`RAISERROR()`" function via the "`ERROR_STATE()`" system built-in function.

- `@ErrorLine` (data type: `INT`): to store the line number where the error has arisen via the "`ERROR_LINE()`" system built-in function.

(b) The `@ErrorMessage`, `@ErrorSeverity`, `@ErrorState`, and `@ErrorLine` will be assigned values via the system built-in functions mentioned and will be displayed using the "`SELECT`" statement.

(c) A string value `'An Error Occured :( '` concatenated with the user pre-defined `@ErrorMessage` in the "`TRY`" block will be displayed using the "`PRINT`" statement.

(d) The system will firstly display the user-defined error message, then display the system-generated error message that referenced the error details such as the error severity, error state, and error line number which is user-specified via "`RAISERROR()`".

***"count_cookedMeals"* Trigger Test Run**

"OrderID" : 'OID001' "CartID" : 'CID001'

"FoodID" : 'F001' "Quantity" : 2

Error 01: New record insertion exceeds the quantity of required row insertion for a specific order ID and food ID (exceeds the quantity of food ordered)

Table 3.2.1.2 "count_cookedMeals" After Insert Triggers Test Run

Insert correct number of records of cooked meal for food ordered
`INSERT INTO CookedMeals VALUES` `('CMID001', 'OID001', 'CID001', 'F001', 'WID02', '11:50:08.901', '11:51:36.751', '12:02:11.567'),` `('CMID002', 'OID001', 'CID001', 'F001', 'WID02', '11:50:08.901', '11:51:36.751', '12:02:11.567');`
Insert extra one record of cooked meal for the same food in the same order
`INSERT INTO CookedMeals VALUES` `('CMID003', 'OID001', 'CID001', 'F001', 'WID02', '11:50:08.901', '11:51:36.751', '12:02:11.567');`
Output
Messages An Error Occured :(Exceed Quantity Ordered Error: CartID = CID001, FoodID = F001 Msg 3616, Level 16, State 1, Line 388

Table 3.2.2.1 "valid_feedback" After Insert Trigger

```
CREATE TRIGGER valid_feedback ON Feedbacks
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON --surpress the 'X rows affected' message
    BEGIN TRY
        DECLARE @memberID NVARCHAR(10)
        DECLARE @orderID NVARCHAR(10)
        DECLARE @foodID NVARCHAR(10)
        DECLARE @feedbackID NVARCHAR(10)
        DECLARE @cartID NVARCHAR(10)
        DECLARE @n INT
        SELECT @feedbackID = INSERTED.FeedbackID, @memberID = INSERTED.MemberID,
        @orderID = INSERTED.OrderID, @foodID = INSERTED.FoodID
        FROM INSERTED

        --get cart id
        SELECT @cartID = O.CartID
        FROM Orders AS O
        WHERE O.MemberID = @memberID AND O.OrderID = @orderID

        --count the number of feedback given to the same food id by that member
        SELECT @n = COUNT(FB.FoodID)
        FROM Feedbacks AS FB
        WHERE FB.MemberID = @memberID AND FB.FoodID = @foodID

        DECLARE @EM NVARCHAR(100)
        DECLARE @El INT
        IF @orderID NOT IN(
            SELECT O.OrderID
            FROM Orders AS O
            WHERE O.MemberID = @memberID
        )
        OR @foodID NOT IN(
            SELECT CI.FoodID
            FROM CartItems AS CI
            WHERE CI.CartID = @cartID
        )
        BEGIN
            SET @EM = 'Invalid Order ID or FoodID Error: MemberID = %s, OrderID =%s, FoodID =%s'
            SET @El = ERROR_LINE()
            RAISERROR(@EM, 16, 1, @memberID, @orderID, @foodID);

            DELETE FROM Feedbacks
            WHERE FeedbackID = @feedbackID
        END

        ELSE IF @n > 1
        BEGIN
            SET @EM = 'One Member One Feedback Per Food Error: MemberID = %s, OrderID = %s, FoodID = %s'
            SET @El = ERROR_LINE()
            RAISERROR(@EM, 16, 1, @memberID, @orderID, @foodID);

            DELETE FROM Feedbacks
            WHERE FeedbackID = @feedbackID
        END
    END TRY
    BEGIN CATCH
        DECLARE @ErrorMessage NVARCHAR(4000)
        DECLARE @ErrorSeverity INT
        DECLARE @ErrorState INT
        DECLARE @ErrorLine INT

        SELECT
            @ErrorMessage = ERROR_MESSAGE(),
            @ErrorSeverity = ERROR_SEVERITY(),
            @ErrorState = ERROR_STATE(),
            @ErrorLine = ERROR_LINE();

        PRINT('An Error Occured :( ' + @ErrorMessage)
    END CATCH
END
```

"valid_feedback" trigger is a "AFTER INSERT" trigger is created and enforced on the "Feedbacks" entity; it runs automatically in the background after data has been inserted into the "Feedbacks" entity. The purpose of this trigger is to validate the order ID and food ID matches the mentioned member ID in the "Orders" and "CartItems" entities, to ensure that members can only provide one feedback to one specific food they ordered.

Explanations

1.	"NOCOUNT" is set on to allow the trigger's code to run in the background, which suppresses the "(X rows affected)" message.
2.	"TRY...CATCH" block is applied.
3.	Under the "TRY" statement: (a) 6 variables are declared, (i) @memberID (data type: NVARCHAR(10)): to store the attribute value of "MemberID" of the inserted row. (ii) @orderID (data type: NVARCHAR(10)): to store the attribute value of "OrderID" of the inserted row. (iii) @foodID (data type: NVARCHAR(10)): to store the attribute value of "FoodID" of the inserted row. (iv) @feedbackID (data type: NVARCHAR(10)): to store the primary key value of "FeedbackID" of the inserted row.

(v) `@cartID` (data type: `NVARCHAR(10)`): to store the attribute value of "`CartID`" of the inserted row.

(vi) `@n` (data type: `INT`): to store the current number of feedback given to a specific food ordered by a specific member using "`COUNT()`", which is the count of rows from the "`Feedbacks`" entity where the attribute value of "`MemberID`" equates to `@memberID` and attribute value of "`FoodID`" equates to `@foodID`, after inserting the new row; assuming that the "`FoodID`" matches the specified "`MemberID`" in the "`Orders`" and "`CartItems`" entities.

(b) Once the variables declared are assigned with their values using the "`SELECT`" statement, two variables are declared to create the user-defined error based on the conditions held by the "`IF...ELSE IF`" statement.

(c) A "`IF...ELSE IF`" statement is applied for error handling, to determine whether the "`FoodID`" given in that inserted row is indeed available in the "`OrderID`" made by that specified member; and to verify whether the number feedback given to a specific food ordered by a specific member exceeds 1. If so, a user-defined error will be created to dealt with each of these issues.

(d) Two variables are declared to create the user-defined error,

(i) `@EM` (data type: `NVARCHAR(100)`): to store the error message specified by the user.

(ii) `@El` (data type: `INT`): to store the line number where the error has arisen.

(e) Both variables will be assigned a different value based on what condition of the "`IF...ELSE IF`" is satisfied. If none of the condition is satisfied, no values will be assigned to these variables, therefore no error is defined.

(f) Under the "`IF`" statement:

(i) The condition for this "`IF`" statement is the value of `@orderID` is not one of the "`OrderID`" where the value of "`MemberID`" equates the value of `@memberID`, indicating the order is not made by the specified member; or the value of `@foodID` is not one of the "`FoodID`" where the value of "`CartID`" equates the value of `@cartID`, indicating the food is not in the shopping cart created by the specified member.

(ii) If the condition is satisfied, `@EM` and `@El` variables are assigned with their respective values.

- `@EM` is assigned with a user-defined error message of `'Invalid Order ID or FoodID Error: MemberID = %s, OrderID =%s, FoodID =%s'`.
- `@El` is assigned with an integer value via "`ERROR_LINE()`", which is a system built-in function that is used to track the error's line number.

(iii) Once the variables are assigned with their values, "`RAISERROR()`" function, which is a system built-in function is used to create a custom error. To introduce this user-defined error to the system, specific information about the error like the error message (`@msg_str`), error severity (`@severity`), and error state (`@state`) are mandatory to be passed to this function's parameters as an arguments, so that the system can store these relevant information about this error into the "`sys.messages`" catalog view, which is the central repository for storing both system-defined and user-defined errors' details. This allows the system to generate appropriate error message in accordance with the information provided.

- Via "`RAISERROR(@EM, 16, 1, @memberID, @orderID, @foodID)`", `@EM` variable will be passed as an argument to the `@msg_str` parameter, 16 will be passed as an argument to the `@severity` parameter, while 1 will be passed as an argument to the `@state` parameter in the "`RAISERROR()`" function. The remaining three variables, which are `@memberID`, `@orderID` and `@foodID` are to be substituted into the `@msg_str` parameter via the three "`%s`" in the `@EM` variable, which serves as a placeholder for string values.

(iv) Once the error is raised, the newly inserted row that satisfied the condition of the "`IF`" statement is said to be erroneous and will therefore be deleted from the "`Feedbacks`" entity where the attribute value of "`FeedbackID`" is equates to the `@feedbackID`.

(g) Under the "`ELSE IF`" statement:

(i) The condition for this "`ELSE IF`" statement is if the value of `@n` is more than 1, indicating there is more than one feedback given to a specific food ordered by a specified member.

(ii) If the condition is satisfied, `@EM` and `@El` variables are assigned with their respective values.

	- `@EM` is assigned with a user-defined error message of `'One Member One Feedback Per Food Error: MemberID = %s, OrderID = %s, FoodID = %s'`. - `@El` is assigned with an integer value via "`ERROR_LINE()`", which is a system built-in function that is used to track the error's line number. (iii) Once the variables are assigned with their values, "`RAISERROR()`" function, which is a system built-in function is used to create a custom error. To introduce this user-defined error to the system, specific information about the error like the error message (`@msg_str`), error severity (`@severity`), and error state (`@state`) are mandatory to be passed to this function's parameters as an arguments, so that the system can store these relevant information about this error into the "`sys.messages`" catalog view, which is the central repository for storing both system-defined and user-defined errors' details. This allows the system to generate appropriate error message in accordance with the information provided. - Via "`RAISERROR(@EM, 16, 1, @memberID, @orderID, @foodID)`", `@EM` variable will be passed as an argument to the `@msg_str` parameter, 16 will be passed as an argument to the `@severity` parameter, while 1 will be passed as an argument to the `@state` parameter in the "RAISERROR()" function. The remaining three variables, which are `@memberID`, `@orderID` and `@foodID` are to be substituted into the `@msg_str` parameter via the three "`%s`" in the `@EM` variable, which serves as a placeholder for string values. (iv) Once the error is raised, the newly inserted row that satisfied the condition of the "`ELSE IF`" statement is said to be erroneous and will therefore be deleted from the "`Feedbacks`" entity where the attribute value of "`FeedbackID`" is equates to the `@feedbackID`.
4.	If either the "`IF`" or "`ELSE IF`" statement under the "`TRY`" block is satisfied, indicating there is an error arose in the "`TRY`" block, the "`TRY`" block will be terminated and throws an exception for the "`CATCH`" block to deal with. The control is passed to the successive group of statements enclosed in the "`CATCH`" block, and these statements will be executed to generate the error message.

5.	On the contrary, if neither the "IF" or "ELSE IF statement under the "TRY" block is satisfied, indicating there is no errors, the "TRY" block will be completed successfully with no exception being thrown, and the control will be passed to the "END CATCH" statement, hence the statement enclosed in the "CATCH" block will not run, no error message will appear.
6.	Under the "CATCH" block: (a) 4 variables are declared, - @ErrorMessage (data type: NVARCHAR(4000)): to store the error message by retrieving the user-defined error message kept in the "sys.messages" catalog view via the "ERROR_MESSAGE()" function. - @ErrorSeverity (data type: INT): to store the error severity by referencing the user-specified error severity that is given through the "RAISERROR()" function via the "ERROR_SEVERITY()" system built-in function. - @ErrorState (data type: INT): to store the error state by referencing the user-specified error state that is given through the "RAISERROR()" function via the "ERROR_STATE()" system built-in function. - @ErrorLine (data type: INT): to store the line number where the error has arisen via the "ERROR_LINE()" system built-in function. (b) The @ErrorMessage, @ErrorSeverity, @ErrorState, and @ErrorLine will be assigned values via the system built-in functions mentioned and will be displayed using the "SELECT" statement. (c) A string value 'An Error Occured :(' concatenated with the user pre-defined @ErrorMessage in the "TRY" block will be displayed using the "PRINT" statement. (d) The system will firstly display the user-defined error message, then display the system-generated error message that referenced the error details such as the error severity, error state, and error line number which is user-specified via "RAISERROR()".

***"valid_feedback"* Trigger Test Run**

"MemberID" : 'MB005' "OrderID" : 'OID001'

"FoodID" : 'F001', 'F004', 'F009'

Error 01: New record insertion with incorrect order ID

Table 3.2.2.2 "valid_feedback" After Insert Triggers Test Run 01

Insert record with incorrect order ID
`INSERT INTO Feedbacks VALUES ('FID001', 'MB0005', 'OID002', 'F001', 5, 'The Nasi Lemak is very nice and tasty :P', '2023-03-16');`
Output
Messages An Error Occured :(Invalid Order ID or FoodID Error: MemberID = MB0005, OrderID =OID002, FoodID =F001 Msg 3616, Level 16, State 1, Line 395

Error 02: New record insertion with incorrect food ID

Table 3.2.2.3 "valid_feedback" After Insert Triggers Test Run 02

Insert record with incorrect food ID
`INSERT INTO Feedbacks VALUES ('FID001', 'MB0005', 'OID002', 'F005', 5, 'The Nasi Lemak is very nice and tasty :P', '2023-03-16');`
Output
Messages An Error Occured :(Invalid Order ID or FoodID Error: MemberID = MB0005, OrderID =OID002, FoodID =F005 Msg 3616, Level 16, State 1, Line 399

Error 03: New record insertion with the same member ID and food ID

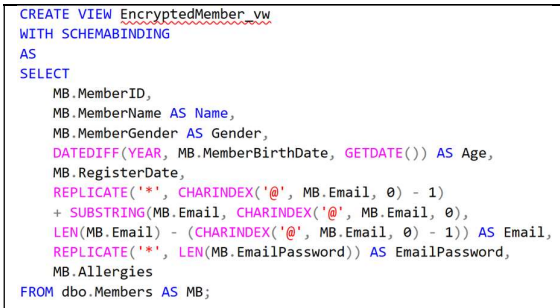
(One member giving two feedback for the same food)

Table 3.2.2.4 "valid_feedback" After Insert Triggers Test Run 03

Insert record for the first feedback by one specific member for a specific food
`INSERT INTO Feedbacks VALUES ('FID001', 'MB0005', 'OID001', 'F001', 5, 'The Nasi Lemak is very nice and tasty :P', '2023-03-16');`
Insert record for the second feedback by one specific member for a specific food
`INSERT INTO Feedbacks VALUES ('FID002', 'MB0005', 'OID001', 'F001', 4, 'The Nasi Lemak is very nice and tasty :D', '2023-04-16');`
Output
Messages An Error Occured :(One Member One Feedback Per Food Error: MemberID = MB0005, OrderID = OID001, FoodID = F001 Msg 3616, Level 16, State 1, Line 406

3.3 View

Table 3.3.1 "EncryptedMember_vw" View

 ``` CREATE VIEW EncryptedMember_vw WITH SCHEMABINDING AS SELECT MB.MemberID, MB.MemberName AS Name, MB.MemberGender AS Gender, DATEDIFF(YEAR, MB.MemberBirthDate, GETDATE()) AS Age, MB.RegisterDate, REPLICATE('*', CHARINDEX('@', MB.Email, 0) - 1) + SUBSTRING(MB.Email, CHARINDEX('@', MB.Email, 0), LEN(MB.Email) - (CHARINDEX('@', MB.Email, 0) - 1)) AS Email, REPLICATE('*', LEN(MB.EmailPassword)) AS EmailPassword, MB.Allergies FROM dbo.Members AS MB; ```	"EncryptedMember_vw" is a view that creates a virtual table from the actual entity, "Members" entity. The purpose of this view is to encrypt confidential information of the registered members such as the email and email password. This serves as a fair security safeguard that enables the user to create a virtual table for specified users with limited access control, those users can access the data via this view created without being granted with the permissions to directly access the underlying base tables.

Explanations	
1.	"SCHEMABINDING" is specified via "WITH SCHEMABINDING". (a) This view is bind to the schema of the underlying base entity, which is the "Members" entity. (b) By implementing "SCHEMABINDING", the underlying base entity is unable to be modified in a certain extend that would influence the view definition, establishing a dependency on the "Members" entity. So, if the entities participated in the view is to be altered, it will not be successful once it affects the view definition, and the Database engine will raise an error. (c) To make modifications to the Members" entity, the view definition is required to be first modified or dropped to eliminate the dependencies on the Members" entity. (d) If the Members" entity that participated in the view is to be dropped, the view definition must be dropped beforehand or modified so that "SCHEMABINDING" no longer exists to drop the Members" entity.
2.	"AS" statement is applied to specify the actions that are to be performed by the view. Under "AS" statement:

(a) "SELECT" statement builds the whole of the view definition. While using "SELECT" statement, referenced objects that must be within the same local database have to be written in two-parts name (schema.object), including views, entities, or user-defined functions; to avoid ambiguity.

(b) Hence, the following attributes are selected from the "dbo.Members" entity (aliased as MB) via the "SELECT" statement:

(i) "MB.MemberID"

(ii) "MB.MemberName" (aliased as Name)

(iii) "MB.MemberGender" (aliased as Gender)

(iv) "DATEDIFF(YEAR, MB.MemberBirthDate, GETDATE())"(aliased as Age)

- "DATEDIFF()" is a system built-in function that calculates the difference between two given dates as an integer, it returns the number of date and time boundaries crossed between two specified dates. Henceforth, the age can be obtained by calculating the difference in year between the birth date and the current date. Three arguments are required to be passed to this function:

(v) "MB.RegisterDate"

(vi) "REPLICATE('*', CHARINDEX('@', MB.Email, 0) - 1) + SUBSTRING(MB.Email, CHARINDEX('@', MB.Email, 0), LEN(MB.Email) - (CHARINDEX('@', MB.Email, 0) - 1))" (aliased as Email)

- Three system built-in functions are incorporated to provide an encrypted email, which are:

REPLICATE()	Repeats a character expression a specified amount of time. Two arguments are required to be passed to this function, which are "@expression" (data type: NVARCHAR(1)) as a string value that specified what expression to repeat; and "@expression" (data type: INT) as an integer value that specified the number of repetitions.
CHARINDEX()	Returns the starting position of the specified expression in character string as an integer value.

	Three arguments are required to be passed to this function, which are "@search_expression" (data type: NVARCHAR(1)) as a string value that specified the expression to search, "@expression_to_be_searched" (data type: NVARCHAR(1)) that specified where to search the expression from, and "@start_location" (data type: INT) that specified the starting index to begin the search.
SUBSTRING()	Returns part of a character, binary, text, or image expression. Three arguments are required to be passed to this function, which are "@expression" that specified what expression to extract a substring from, "@starting_position" (data type: INT) as an integer value that specified the starting index to extract the string as a substring, and "@length" (data type: INT) as an integer value that specified the length of the string from the starting index as the substring.

- The application of these three functions can be separated into two parts that are later concatenated via "+":

REPLICATE('*', CHARINDEX('@', MB.Email, 0) - 1)	
@expression NVARCHAR(1)	'*'
@expression INT	CHARINDEX('@', MB.Email, 0) - 1
	Concepts
	index of '@': 6 length of string before '@': 6 -1 = 5
Output 01	'*****'

`SUBSTRING(MB.Email, CHARINDEX('@', MB.Email, 0), LEN(MB.Email) - (CHARINDEX('@', MB.Email, 0) - 1))`”	
`@expression`	`MB.Email`
`@starting_position INT`	`CHARINDEX('@', MB.Email, 0)`
	Concepts
	index of `'@'`: 6
`@length INT`	`LEN(MB.Email) - (CHARINDEX('@', MB.Email, 0) - 1))`
	Concepts
	length of `'ali08@gmail.com'`: 15 length of string before `'@'`: 6 - 1 = 5 length of substring: 15 - 5 = 10
`Output 02`	`'@gmail.com'`

- Final Output: `'*****'` + `'@gmail.com'` = `'*****@gmail.com'`

(vii) “`REPLICATE('*', LEN(MB.EmailPassword))`” (aliased as `EmailPassword`)

- Two system built-in functions are incorporated to provide an encrypted email password, which are:

`REPLICATE()`	Refer table XX
`LEN()`	Returns the number of characters in a given string expression. One argument is required to be passed to this function, which is “`@expression`” (data type: `NVARCHAR(1)`) as a string value.

(e.g. `'rduferie0'`)

		`REPLICATE('*', LEN(MB.EmailPassword))`	
		`@expression NVARCHAR(1)`	`'*'`
		`@expression INT`	`LEN(MB.EmailPassword)`
			Concepts
			length of "`MB.EmailPassword`": 9
		`Final Output`	`'*********'`

(viii) "`MB.Allergies`"

Comparison between "`Members`" Entity and Virtual "`Members`" Entity

Table 3.3.2 Comparison Between "Members" Entity and "EncryptedMember_vw" View

"`Members`" Entity

Results Messages

	MemberID	MemberName	MemberBirthdate	MemberGender	RegisterDate	Email	EmailPassword	Allergies
1	MB0001	Ali	1994-10-08	Male	2023-01-10 08:09:10.123	ali08@gmail.com	rduferie0	NULL
2	MB0002	Rainbow	1999-05-14	Female	2023-01-20 14:22:11.123	bow20@gmail.com	dA1! mRJ	Peanut
3	MB0003	Sarah	1965-03-20	Female	2023-01-27 13:09:56.243	s1714@gmail.com	S@r13	Seafood
4	MB0004	Jason	1985-02-14	Prefer not to say	2023-02-15 10:12:12.453	j171n@gmail.com	JaSaN2003	NULL
5	MB0005	K.Vince	2004-12-11	Male	2023-02-21 16:44:04.443	kv0566@gmail.com	pP4#5jjC	Milk
6	MB0006	MacHill	1975-04-08	Male	2023-02-24 04:44:04.447	mh0008@gmail.com	P@ssw673	Peach
7	MB0007	J.Wenton	1994-09-24	Prefer not to say	2023-03-01 06:56:06.667	jw3687@gmail.com	JW!1994?	Banana
8	MB0008	Micheal	2005-06-09	Female	2023-03-15 18:46:06.667	m1988@gmail.com	tD4!} Msi	NULL
9	MB0009	Shanan	1977-11-11	Female	2023-03-26 17:33:13.427	shan17@gmail.com	sMarTmE844	NULL
10	MB0010	S.Hanson	1971-07-16	Male	2023-03-05 15:23:45.230	sh1234@gmail.com	ldkWhat1971	Seafood

Virtual "`Members`" Entity

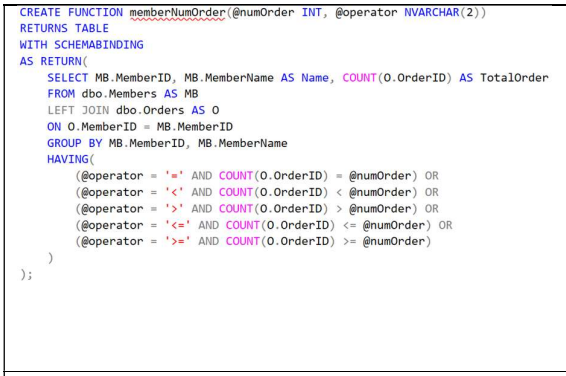
(From "`EncryptedMember_vw`" view)

Results Messages

	MemberID	Name	Gender	Age	RegisterDate	Email	EmailPassword	Allergies
1	MB0001	Ali	Male	30	2023-01-10 08:09:10.123	*****@gmail.com	*********	NULL
2	MB0002	Rainbow	Female	25	2023-01-20 14:22:11.123	*****@gmail.com	********	Peanut
3	MB0003	Sarah	Female	59	2023-01-27 13:09:56.243	*****@gmail.com	*****	Seafood
4	MB0004	Jason	Prefer not to say	39	2023-02-15 10:12:12.453	*****@gmail.com	*********	NULL
5	MB0005	K.Vince	Male	20	2023-02-21 16:44:04.443	******@gmail.com	********	Milk
6	MB0006	MacHill	Male	49	2023-02-24 04:44:04.447	******@gmail.com	********	Peach
7	MB0007	J.Wenton	Prefer not to say	30	2023-03-01 06:56:06.667	******@gmail.com	********	Banana
8	MB0008	Micheal	Female	19	2023-03-15 18:46:06.667	*****@gmail.com	*********	NULL
9	MB0009	Shanan	Female	47	2023-03-26 17:33:13.427	******@gmail.com	**********	NULL
10	MB0010	S.Hanson	Male	53	2023-03-05 15:23:45.230	******@gmail.com	**********	Seafood

3.4 User-Defined Functions

Table 3.4.1 memberNumOrder() Table-Valued Function

 ``` CREATE FUNCTION memberNumOrder(@numOrder INT, @operator NVARCHAR(2)) RETURNS TABLE WITH SCHEMABINDING AS RETURN(SELECT MB.MemberID, MB.MemberName AS Name, COUNT(O.OrderID) AS TotalOrder FROM dbo.Members AS MB LEFT JOIN dbo.Orders AS O ON O.MemberID = MB.MemberID GROUP BY MB.MemberID, MB.MemberName HAVING((@operator = '=' AND COUNT(O.OrderID) = @numOrder) OR (@operator = '<' AND COUNT(O.OrderID) < @numOrder) OR (@operator = '>' AND COUNT(O.OrderID) > @numOrder) OR (@operator = '<=' AND COUNT(O.OrderID) <= @numOrder) OR (@operator = '>=' AND COUNT(O.OrderID) >= @numOrder))); ```	"memberNumOrder()" is a user-defined table-valued function that displays a table showing members that made either equal to, less than, less than or equal to, more than, or more than or equal to a specified number of orders. This function returns the result in table.

Explanations	
1.	This function consists of two parameters: (a) "@numOrder" (data type: INT): specifying the number of orders made as integer value. (b) "@operator" (data type: NVARCHAR(2)): specifying the operators ranging from '=', '<', '<=', '>', and '>='.
2.	"SCHEMABINDING" is specified via "WITH SCHEMABINDING". (a) This function is bind to the schema of the underlying base entity, which is the "Members" entity. (b) By implementing "SCHEMABINDING", the underlying base entity is unable to be modified in a certain extend that would influence the function's definition, establishing a dependency on the "Members" entity. So, if the entities participated in the function is to be altered, it will not be successful once it affects the function's definition, and the Database engine will raise an error. (c) To make modifications to the Members" entity, the function's definition is required to be first modified or dropped to eliminate the dependencies on the Members" entity. (d) If the Members" entity that participated in the function is to be dropped, the function's definition must be dropped beforehand or modified so that "SCHEMABINDING" no longer exists to drop the Members" entity.
3.	"AS" statement is applied to specify the actions that are to be performed by the function. Under "AS" statement:

	(a) "SELECT" statement builds the whole of the function definition. While using "SELECT" statement, referenced objects that must be within the same local database have to be written in two-parts name (schema.object), including views, entities, or user-defined functions; to avoid ambiguity. (b) "DBM_Assignment_dbo.Members" entity (aliased as MB) is left joined with "DBM_Assignment.dbo.Orders" entity (aliased as O) via the "LEFT JOIN" statement, given that the value of "O.MemberID" in "DBM_Assignment_dbo.Members" entity is equates to the value of "MB.MemberID" in "DBM_Assignment_dbo.Orders" entity. All attributes in "DBM_Assignment_dbo.Members" entity and common attributes in both "DBM_Assignment_dbo.Members" and "DBM_Assignment_dbo.Orders" entity are combined in a single table. (c) Hence, the following attributes are selected from the "DBM_Assignment_dbo.Members" entity (aliased as MB) that is left joined with the "DBM_Assignment.dbo.Orders" entity (aliased as O) via the "SELECT" statement: (i) "MB.MemberID" (ii) "MB.MemberName" (aliased as Name) (iii) "COUNT(O.OrderID)" (aliased as TotalOrder) - "COUNT()" is a system built-in function that returns number of items in a group as an integer value, which indicates the number of order records grouped by each member, showing the total number of orders made by each member.
4.	A "HAVING" clause is applied to show results based on the arguments passed to the "@operator". The following are the condition of showing members with a specified range of number or orders made. (a) "@operator = '=' AND COUNT(O.OrderID) = @numOrder)": showing the members that made a total number of orders equates to the value of "@numOrder" variable. (b) "@operator = '<' AND COUNT(O.OrderID) < @numOrder)": showing the members that made a total number of orders that is less than the value of "@numOrder" variable.

	(c) "`@operator = '<='" AND COUNT(O.OrderID) = @numOrder)`": showing the members that made a total number of orders that is less than or equates to the value of "`@numOrder`" variable. (d) "`@operator = '>'" AND COUNT(O.OrderID) = @numOrder)`": showing the members that made a total number of orders that is more than the value of "`@numOrder`" variable. (e) "`@operator = '>='" AND COUNT(O.OrderID) = @numOrder)`": showing the members that made a total number of orders that is more than or equates to the value of "`@numOrder`" variable.

Table 3.4.2 studentOrStaff() Scalar-Valued Function

```
CREATE FUNCTION studentOrStaff(@memberID nvarchar(10))
RETURNS nvarchar(10)
AS
BEGIN
    DECLARE @role nvarchar(10)
    DECLARE @tpnumber nvarchar(10)
    DECLARE @staffID nvarchar(10)
    SET @tpnumber = (
        SELECT S.TPNumber
        FROM Members AS MB
        LEFT JOIN Students AS S
        ON S.MemberID = MB.MemberID
        LEFT JOIN Lecturers AS L
        ON L.MemberID = MB.MemberID
        WHERE MB.MemberID = @memberID
    );
    SET @staffID = (
        SELECT L.StaffID
        FROM Members AS MB
        LEFT JOIN Students AS S
        ON S.MemberID = MB.MemberID
        LEFT JOIN Lecturers AS L
        ON L.MemberID = MB.MemberID
        WHERE MB.MemberID = @memberID
    );

    IF (@tpnumber IS NOT NULL AND @staffID IS NULL)
    BEGIN
         SET @role = 'Student'
    END
    ELSE IF (@staffID IS NOT NULL AND @tpnumber IS NULL)
    BEGIN
         SET @role = 'Staff'
    END
    ELSE
    BEGIN
        SET @role = 'Unknown'
    END

    RETURN @role
END
```

"`studentOrStaff()`" is a user-defined scalar-valued function that differentiates the role of each members from the "`Members`" entity, whether the member is a student or a staff. This function returns a string value (data type: `NVARCHAR(10)`) of "Student" or "Staff".

Explanations	
1.	This function consists of one parameter: (a) "@memberID" (data type: NVARCHAR(10)): specifying the string value of "MemberID" attribute obtained from the "Members" entity.
2.	"AS" statement is applied to specify the actions that are to be performed by the function. Under "AS" statement: (a) 3 variables are declared, (i) "@role" (data type: NVARCHAR(10)): to store the string value of the role of the member with the specified "MemberID' attribute of value equates to "@memberID", that is either "Student" or "Staff". (ii) "@tpnumber" (data type: NVARCHAR(10)): to store the string value of the "TPNumber" attribute of the member with the specified "MemberID' attribute of value equates to "@memberID". The value can be "NULL". (iii) "@staffID" (data type: NVARCHAR(10)): to store the string value of the "StaffID" attribute of the member with the specified "MemberID" attribute of value equates to "@memberID". The value can be "NULL". (b) "SET" statement is used to assign values to "@tpnumber" and "@staffID" variables. (i) "@tpnumber" is set to the value of the "TPNumber" attribute of the member where the specified "MemberID' attribute of value equates to "@memberID", from the "Members" entity (aliased as MB) that is left joined with "Students" entity (aliased as S) on the value of "MemberID" on both entity are equal, and also left joined with "Lecturers" entity (aliased as L) on the value of "MemberID" on both entity are equal. "SELECT" statement is used to get the value. (ii) "@staffID" is set to the value of the "StaffID" attribute of the member where the specified "MemberID' attribute of value equates to "@memberID", from the "Members" entity (aliased as MB) that is left joined with "Students" entity (aliased as S) on the value of "MemberID" on both entity are equal, and also left joined with "Lecturers" entity (aliased as L) on the value of "MemberID" on both entity are equal. "SELECT" statement is used to get the value.

	(c) "IF...ELSE IF...ELSE" statement is used to determine whether the "@tpnumber" and "@staffID" variables are null or not null. (d) Under the "IF" statement: (i) The condition is when the value of "@tpnumber" is "NOT NULL" and "@staffID" variables is "NULL", the value of "@role" is set to 'Student', indicating that the member is student due to it having a not null TP number. (e) Under the "ELSE IF" statement: (i) The condition is when the value of "@staffID" is "NOT NULL" and "@tpnumber" variables is "NULL", the value of "@role" is set to 'Staff', indicating that the member is a staff due to it having a not null staff ID. (f) Under the "ELSE" statement: (i) The condition is when the value of both "@staffID" and "@tpnumber" variables is "NULL", the value of "@role" is set to 'Unknown', indicating that the role of the member cannot be identified.
3.	The value of "@role" is returned to the calling of the function as a string value via "RETURN" statement.

3.5 Select Queries

i. List the food(s) which has the highest rating. Show food id, food name and the rating.

Table 3.5.1 Select highest rating

SQL Statements	Executed Results

```
SELECT FB.FoodID, M.FoodName, AVG(Rating) AS Rating
FROM Feedbacks AS FB
INNER JOIN Menu AS M
ON FB.FoodID = M.FoodID
GROUP BY FB.FoodID, M.FoodName
ORDER BY AVG(Rating) DESC;
```

	FoodID	FoodName	Rating
1	F001	Nasi Lemak	5
2	F002	Nasi Goreng	5
3	F010	Tea	4
4	F009	Kopi	3
5	F007	Chicken Chop	3
6	F004	Waffles	3
7	F006	Spaghetti Bolognese	2
8	F008	Milk Tea	1

Explanation	
1.	"`SELECT FB.FoodID, M.FoodName, AVG(Rating) AS Rating`" - Selects three specific columns, which are "`FoodID`" from "`Feedbacks`" entity (aliased as "`FB`"), "`FoodName`" from "`Menu`" entity (aliased as "`M`") and the average "`Rating`" for each food (aliased as "`Rating`").
2.	"`FROM Feedbacks AS FB`" - The primary entity involved, which is "`Feedbacks`" entity (aliased as "`FB`").
3.	"`INNER JOIN Menu AS M`" - "`INNER JOIN`" between the "`Feedbacks`" entity (`FB`) and the "`Menu`" entity (`M`).
4.	"`ON FB.FoodID = M.FoodID`" - The entity joining condition is "`FB.FoodID = M.FoodID`", where only "`FoodID`" in "`Feedbacks`" entity match with "`FoodID`" in "`Menu`" entity.
5.	"`GROUP BY FB.FoodID, M.FoodName`" - "`GROUP`" the results by both "`FB.FoodID`" and "`M.FoodName`", ensures the average rating for each food is calculated.
6.	"`ORDER BY AVG(Rating) DESC`" - Sorts the results in "`DESC`" order based on the calculated "`AVG(Rating)`".

ii. Find the total number of feedback per member. Show member id, member name and total number of feedback per member.

Table 3.5.2 Select total number feedback per member

<table>
<tr><th>SQL Statements</th><th>Executed Results</th></tr>
<tr><td>

```
SELECT FB.MemberID, MB.MemberName AS Name, COUNT(FB.Rating) AS NumOfFeedbacks
FROM Feedbacks AS FB
INNER JOIN Members AS MB
ON FB.MemberID = MB.MemberID
GROUP BY FB.MemberID, MB.MemberName;
```

</td><td>

	MemberID	Name	NumOfFeedbacks
1	MB0002	Rainbow	2
2	MB0005	K.Vince	3
3	MB0006	MacHill	2
4	MB0007	J.Wenton	2
5	MB0008	Micheal	1
6	MB0010	S.Hanson	1

</td></tr>
<tr><td colspan="2">Explanation</td></tr>
<tr><td>1.</td><td>“<code>SELECT FB.MemberID, MB.MemberName AS Name, COUNT(FB.Rating) AS NumOfFeedbacks</code>”
- Selects three specific columns, which are “<code>MemberID</code>” from “<code>Feedbacks</code>” entity (aliased as “<code>FB</code>”), “<code>MemberName</code>” from “<code>Members</code>” entity (aliased as “<code>MB</code>”) and the counts of number of records in “<code>Rating</code>” from “<code>Feedbacks</code>” entity (aliased as “<code>FB</code>”) for each member (aliased as “<code>NumOfFeedbacks</code>”).</td></tr>
<tr><td>2.</td><td>“<code>FROM Feedbacks AS FB</code>”
- The primary entity involved, which is “<code>Feedbacks</code>” entity (aliased as “<code>FB</code>”).</td></tr>
<tr><td>3.</td><td>“<code>INNER JOIN Members AS MB</code>”
- “<code>INNER JOIN</code>” between the “<code>Feedbacks</code>” entity (<code>FB</code>) and the “<code>Members</code>” entity (<code>MB</code>).</td></tr>
<tr><td>4.</td><td>“<code>ON FB.MemberID = MB.MemberID</code>”
- The entity joining condition is “<code>FB.MemberID = MB.MemberID</code>”, where only “<code>MemberID</code>” in “<code>Feedbacks</code>” entity match with “<code>MemberID</code>” in “<code>Members</code>” entity.</td></tr>
<tr><td>5.</td><td>“<code>GROUP BY FB.MemberID, MB.MemberName</code>”
- “<code>GROUP</code>” the results by both “<code>FB.MemberID</code>” and “<code>MB.MemberName</code>”, ensures the total count of feedbacks for each member is calculated.</td></tr>
</table>

iii. Find members who have not made any orders. Show member id, member name and the total order.

Table 3.5.3 Select member who not made any orders

SQL Statements	Executed Results
`--memberNumOrder(@numOrder INT, @operator NVARCHAR(2))` `SELECT * FROM memberNumOrder(0, '=');`	MemberID | Name | TotalOrder 1 | MB0004 | Jason | 0 2 | MB0009 | Shanan | 0

Explanation	
1.	"`SELECT * FROM memberNumOrder(0, '=')`" - Calls the "`memberNumOrder`" function. - Passes two arguments o "`0`" is the number of orders to compare against. o "`'='`" is the comparison operator. - "`SELECT *`" retrieves all column returned by the function, which are "`MemberID`", "`Name`" and "`TotalOrder`" for members with zero order.

iv. Find the total number of food(meal) ordered by manager from each chef.

Table 3.5.1 Select total number of foods ordered by manager from each chef

SQL Statements	Executed Results
`--NumOfFood ordered by chef only, excluding order that is pending and just confirmed (not read)` `SELECT M.ChefID, EMP.EmployeeName AS Name, SUM(CI.Quantity) AS NumOfFood, O.OrderManagerID` `FROM Orders AS O` `INNER JOIN OrderStatus AS OS` `ON O.OrderID = OS.OrderID AND OS.DeliveryStatus NOT IN('Pending', 'Waiting', 'confirmed')` `INNER JOIN CartItems AS CI` `ON O.CartID = CI.CartID` `INNER JOIN Menu AS M` `ON CI.FoodID = M.FoodID` `INNER JOIN Employees AS EMP` `ON M.ChefID = EMP.EmployeeID` `GROUP BY M.ChefID, EMP.EmployeeName, O.OrderManagerID;`	ChefID | Name | NumOfFood | OrderManagerID 1 | CFID01 | Wesley.S | 19 | MID01 2 | CFID02 | Susana.R | 10 | MID01 3 | CFID03 | Otto.A | 2 | MID01

Explanation	
1.	"`SELECT M.ChefID, EMP.EmployeeName AS Name, SUM(CI.Quantity) AS NumOfFood, O.OrderManagerID`" - Selects four specific columns, which are "`ChefID`" from "`Menu`" entity (aliased as "`M`"), "`EmployeeName`" from "`Employees`" entity (aliased as "`EMP`"), the total number of "`Quantity`" for each food item from "`CartItems`" entity (aliased as "`CI`") and "`OrderManagerID`" from "`Orders`" entity (aliased as "`O`").

2.	"`FROM Orders AS O`" - The primary entity involved, which is "`Orders`" entity (aliased as "`O`").
3.	"`INNER JOIN OrderStatus AS OS`" - "`INNER JOIN`" between the "`Orders`" entity (`O`) and the "`OrderStatus`" entity (`OS`).
4.	"`ON O.OrderID = OS.OrderID AND OS.DeliveryStatus NOT IN('Pending', 'Waiting', 'Confirmed')`" - The entity joining condition is "`O.OrderID = OS.OrderID`", where only "`OrderID`" in "`Orders`" entity match with "`OrderID`" in "`OrderStatus`" entity "`AND`" filters for orders where the "`DeliveryStatus`" in "`OrderStatus`" entity is not `'Pending', 'Waiting',` or `'Confirmed'`.
5.	"`INNER JOIN CartItems AS CI`" - "`INNER JOIN`" between the "`Orders`" entity (`O`) and the "`CartItems`" entity (`CI`).
6.	"`ON O.CartID = CI.CartID`" - The entity joining condition is "`O.CartID = CI.CartID`", where only "`CartID`" in "`Orders`" entity match with "`CartID`" in "`CartItems`" entity.
7.	"`INNER JOIN Menu AS M`" - "`INNER JOIN`" between the "`CartItems`" entity (`CI`) and the "`Menu`" entity (`M`).
8.	"`ON CI.FoodID = M.FoodID`" - The join condition is "`CI.FoodID = M.FoodID`", where only "`FoodID`" in "`CartItems`" entity match with "`FoodID`" in "`Menu`" entity. The entity joining condition is "`CI.FoodID = M.FoodID`", where only "`FoodID`" in "`CartItems`" entity match with "`FoodID`" in "`Menu`" entity.
9.	"`INNER JOIN Employees AS EMP`" - "`INNER JOIN`" between the "`Menu`" entity (`M`) and the "`Employees`" entity (`EMP`).
10.	"`ON M.ChefID = EMP.EmployeeID`" - The join condition is "`M.ChefID = EMP.EmployeeID`", where only "`ChefID`" in "`Menu`" entity match with "`EmployeeID`" in "`Employees`" entity. The entity

	joining condition is "`M.ChefID = EMP.EmployeeID`", where only "`ChefID`" in "`Menu`" entity match with "`EmployeeID`" in "`Employees`" entity.
11.	"`GROUP BY M.ChefID, EMP.EmployeeName, O.OrderManagerID`" - "`GROUP`" the results by "`M.ChefID`", "`EMP.EmployeeName`" and "`O.OrderManagerID`", ensures the total quantity of cooked food ordered by manager from each chef is calculated.

v. Find the total number of food(meal) cooked by each chef. Show chef id, chef name, and number of meals cooked.

Table 3.5.4 Select total number of food cooked by each chef

SQL Statements	Executed Results

```
SELECT M.ChefID, EMP.EmployeeName AS Name, COUNT(CM.FoodID) AS NumOfFood
FROM CookedMeals AS CM
INNER JOIN Menu AS M
ON CM.FoodID = M.FoodID
INNER JOIN Employees AS EMP
ON M.ChefID = EMP.EmployeeID
GROUP BY M.ChefID, EMP.EmployeeName;
```

	ChefID	Name	NumOfFood
1	CFID01	Wesley.S	13
2	CFID02	Susana.R	9
3	CFID03	Otto.A	2

Explanation	
1.	"`SELECT M.ChefID, EMP.EmployeeName AS Name, COUNT(CM.FoodID) AS NumOfFood`" - Selects three specific columns, which are "`ChefID`" from "`Menu`" entity (aliased as "`M`"), "`EmployeeName`" from "`Employees`" entity (aliased as "`Name`") and the counts of number of entries in "`FoodID`" from "`CookedMeals`" entity (aliased as "`CM`") for each chef (aliased as "`NumOfFood`").Selects three specific columns, which are "`ChefID`" from "`Menu`" entity (aliased as "`M`"), "`EmployeeName`" from "`Employees`" entity (aliased as "`Name`") and the counts of number of records in "`FoodID`" from "`CookedMeals`" entity (aliased as "`CM`") for each chef (aliased as "`NumOfFood`").
2.	"`FROM CookedMeals AS CM`" - The primary entity involved, which is "`CookedMeals`" entity (aliased as "`CM`").
3.	"`INNER JOIN Menu AS M`" - "`INNER JOIN`" between the "`CookedMeals`" entity (`CM`) and the "`Menu`" entity (`M`).

4.	"`ON CM.FoodID = M.FoodID`" - The join condition is "`CM.FoodID = M.FoodID`", where only "`FoodID`" in "`CookedMeals`" entity match with "`FoodID`" in "`Menu`" entity.The entity joining condition is "`CM.FoodID = M.FoodID`", where only "`FoodID`" in "`CookedMeals`" entity match with "`FoodID`" in "`Menu`" entity.
5.	"`INNER JOIN Employees AS EMP`" - "`INNER JOIN`" between the "`Menu`" entity (`M`) and the "`Employees`" entity (`EMP`).
6.	"`ON M.ChefID = EMP.EmployeeID`" - The join condition is "`M.ChefID = EMP.EmployeeID`", where only "`ChefID`" in "`Menu`" entity match with "`EmployeeID`" in "`Employees`" entity.The entity joining condition is "`M.ChefID = EMP.EmployeeID`", where only "`ChefID`" in "`Menu`" entity match with "`EmployeeID`" in "`Employees`" entity.
7.	"`GROUP BY M.ChefID, EMP.EmployeeName`" - "`GROUP`" the results by both "`M.ChefID`" and "`EMP.EmployeeName`", ensures the total number of cooked meal for each chef is calculated.

vi. List all the food where its average rating is more than the average rating of all food.

Table 3.5.6 Select all food above average rating

SQL Statements	Executed Results

```
SELECT FB.FoodID, M.FoodName, AVG(FB.Rating) AS AverageRating
FROM Feedbacks AS FB
INNER JOIN Menu AS M
ON FB.FoodID = M.FoodID
GROUP BY FB.FoodID, M.FoodName
HAVING AVG(FB.Rating) > (
    SELECT AVG(Rating) FROM Feedbacks
);
```

	FoodID	FoodName	AverageRating
1	F001	Nasi Lemak	5
2	F002	Nasi Goreng	5
3	F010	Tea	4

Explanation	
1.	"`SELECT FB.FoodID, M.FoodName, AVG(FB.Rating) AS AverageRating`" - Selects three specific columns, which are "`FoodID`" from "`Feedbacks`" entity (aliased as "`FB`"), "`FoodName`" from "`Menu`" entity (aliased as "`M`") and the average "`Rating`" from "`Feedbacks`" entity (aliased as "`FB`") for each food (aliased as "`AverageRating`").

2.	"`FROM Feedbacks AS FB`" - The primary entity involved, which is "`Feedbacks`" entity (aliased as "`FB`").
3.	"`INNER JOIN Menu AS M`" - "`INNER JOIN`" between the "`Feedbacks`" entity (`FB`) and the "`Menu`" entity (`M`).
4.	"`ON FB.FoodID = M.FoodID`" - The join condition is "`FB.FoodID = M.FoodID`", where only "`FoodID`" in "`Feedbacks`" entity match with "`FoodID`" in "`Menu`" entity.The entity joining condition is "`FB.FoodID = M.FoodID`", where only "`FoodID`" in "`Feedbacks`" entity match with "`FoodID`" in "`Menu`" entity.
5.	"`GROUP BY FB.FoodID, M.FoodName`" - "`GROUP`" the results by both "`FB.FoodID`" and "`M.FoodName`", ensures the average rating of each food is calculated.
6.	"`HAVING AVG(FB.Rating) > (SELECT AVG(Rating) FROM Feedbacks)`" - Filters the results to include only foods where the "`AVG(FB.Rating)`" (average rating of each food) is "`>`" (greater than) the overall "`AVG(Rating)`" (average rating of all foods) from "`Feedbacks`" entity.

vii. Find the top 3 bestselling food(s). The list should include id, name, price and quantity sold.

Table 3.5.7 Select top 3 bestselling food

SQL Statements	Executed Results

```
SELECT TOP 3 CI.FoodID, M.FoodName, M.Price, SUM(CI.Quantity) AS QuantitySold
FROM CartItems AS CI
INNER JOIN Orders AS O
ON CI.CartID = O.CartID
INNER JOIN OrderStatus AS OS
ON O.OrderID = OS.OrderID AND OS.DeliveryStatus <> 'Pending'
INNER JOIN Menu AS M
ON CI.FoodID = M.FoodID
GROUP BY CI.FoodID, M.FoodName, M.Price
ORDER BY SUM(CI.Quantity) DESC, CI.FoodID ASC;
```

	FoodID	FoodName	Price	QuantitySold
1	F007	Chicken Chop	9.00	8
2	F001	Nasi Lemak	3.00	4
3	F002	Nasi Goreng	5.00	4

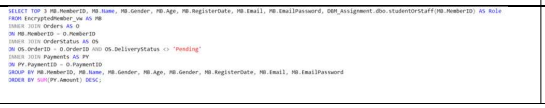
Explanation	
1.	"`SELECT TOP 3 CI.FoodID, M.FoodName, M.Price, SUM(CI.Quantity) AS QuantitySold`" - "`TOP 3`" limits the results to the top 3 rows. - Selects four specific columns, which are "`FoodID`" from "`CartItems`" entity (aliased as "`CI`"), "`FoodName`" from "`Menu`" entity (aliased as "`M`"), "`Price`"

	from "Menu" entity (aliased as "M") and the total number of "Quantity" from "CartItems" entity (aliased as "CI") for each food item (aliased as "QuantitySold").
2.	"FROM CartItems AS CI" - The primary entity involved, which is "CartItems" entity (aliased as "CI").
3.	"INNER JOIN Orders AS O" - "INNER JOIN" between the "CartItems" entity (CI) and the "Orders" entity (O).
4.	"ON CI.CartID = O.CartID" - The join condition is "CI.CartID = O.CartID", where only "CartID" in "CartItems" entity match with "CartID" in "Orders" entity.The entity joining condition is "CI.CartID = O.CartID", where only "CartID" in "CartItems" entity match with "CartID" in "Orders" entity.
5.	"INNER JOIN OrderStatus AS OS" - "INNER JOIN" between the "Orders" entity (O) and the "OrderStatus" entity (OS).
6.	"ON O.OrderID = OS.OrderID AND OS.DeliveryStatus <> 'Pending'" - The join condition is "O.OrderID = OS.OrderID", where only "OrderID" in "Orders" entity match with "OrderID" in "OrderStatus" entity "AND" filters for orders where the "DeliveryStatus" in "OrderStatus" entity is not 'Pending'. The entity joining condition is "O.OrderID = OS.OrderID", where only "OrderID" in "Orders" entity match with "OrderID" in "OrderStatus" entity "AND" filters for orders where the "DeliveryStatus" in "OrderStatus" entity is not 'Pending'.
7.	"INNER JOIN Menu AS M" - "INNER JOIN" between the "CartItems" entity (CI) and the "Menu" entity (M).
8.	"ON CI.FoodID = M.FoodID" - The entity joining condition is "CI.FoodID = M.FoodID", where only "FoodID" in "CartItems" entity match with "FoodID" in "Menu" entity.

9.	"`GROUP BY CI.FoodID, M.FoodName, M.Price`" - "`GROUP`" the results by "`CI.FoodID`", "`M.FoodName`" and "`M.Price`", ensures the total quantity sold for each food is calculated.
10.	"`ORDER BY SUM(CI.Quantity) DESC, CI.FoodID ASC`" - Orders the results by the "`SUM`" of "`CI.Quantity`" in "`DESC`". - If multiple food items have equal "`SUM(CI.Quantity)`", orders them by "`CI.FoodID`" in "`ASC`".

viii. Show the top 3 members who spent most on ordering food. List should include id and name and whether they student or staff.

Table 3.5.8 Select top 3 member who spent most on ordering food

SQL Statements	Executed Results
	

Explanation	
1.	"`SELECT TOP 3 MB.MemberID, MB.Name, MB.Gender, MB.Age, MB.RegisterDate, MB.Email, MB.EmailPassword, DBM_Assignment.dbo.studentOrStaff(MB.MemberID) AS Role`" - "`TOP 3`" limits the results to the top 3 rows. - Selects eight specific columns, which are "`MemberID`", "`Name`", "`Gender`", "`Age`", "`RegisterDate`", "`Email`", "`EmailPassword`" and "`DBM_Assignment.dbo.studentOrStaff(MB.MemberID)`" (aliased as "`Role`") from "`EncryptedMember_vw`" entity (aliased as "`MB`").
2.	"`FROM EncryptedMember_vw AS MB`" - The primary entity involved, which is the virtual table showed by the "`EncryptedMember_vw`" view (aliased as "`MB`").
3.	"`INNER JOIN Orders AS O`" - "`INNER JOIN`" between the virtual table of "`EncryptedMember_vw`" view (`MB`) and the "`Orders`" entity (`O`).

4.	"`ON MB.MemberID = O.MemberID`" - The entity joining condition is "`MB.MemberID = O.MemberID`", where only "`MemberID`" in virtual table of "`EncryptedMember_vw`" view match with "`MemberID`" in "`Orders`" entity.
5.	"`INNER JOIN OrderStatus AS OS`" - "`INNER JOIN`" between the "`Orders`" entity (`O`) and the "`OrderStatus`" entity (`OS`).
6.	"`ON OS.OrderID = O.OrderID AND OS.DeliveryStatus <> 'Pending'`" - The join condition is "`OS.OrderID = O.OrderID`", where only "`OrderID`" in "`OrderStatus`" entity match with "`OrderID`" in "`Orders`" entity "`AND`" filters for orders where the "`DeliveryStatus`" in "`OrderStatus`" entity is not `'Pending'`.
7.	"`INNER JOIN Payments AS PY`" - "`INNER JOIN`" between the "`Orders`" entity (`O`) and the "`Payments`" entity (`PY`).
8.	"`ON PY.PaymentID = O.PaymentID`" - The entity joining condition is "`PY.PaymentID = O.PaymentID`", where only "`PaymentID`" in "`Payments`" entity match with "`PaymentID`" in "`Orders`" entity.
9.	"`GROUP BY MB.MemberID, MB.Name, MB.Gender, MB.Age, MB.Gender, MB.RegisterDate, MB.Email, MB.EmailPassword`" - "`GROUP`" the results by "`MB.MemberID`", "`MB.Name`", "`MB.Gender`", "`MB.Age`", "`MB.Gender`", "`MB.RegisterDate`", "`MB.Email`" and "`MB.EmailPassword`", ensures the total amount spent for each member is calculated.
10.	"`ORDER BY SUM(PY.Amount) DESC`" - Orders the results by the "`SUM`" of "`PY.Amount`" in "`DESC`".

ix. Show the total members based on gender who are registered as members. List should include id, name, role(student/staff) and gender.

Table 3.5.9 Select total members based on gender who registered as members

SQL Statements	Executed Results

```
SELECT MB.MemberID, MB.MemberName AS Name, MB.MemberGender AS Gender,
DBM_Assignment.dbo.studentOrStaff(MB.MemberID) AS Role,
COUNT(MB.MemberGender) OVER (PARTITION BY MemberGender) AS TotalMemberGenderBased
FROM Members AS MB
ORDER BY Gender ASC, MemberID ASC;
```

	MemberID	Name	Gender	Role	TotalMemberGenderBased
1	MB0002	Rainbow	Female	Student	4
2	MB0003	Sarah	Female	Staff	4
3	MB0008	Micheal	Female	Student	4
4	MB0009	Shanan	Female	Staff	4
5	MB0010	S.Hanson	Male	Staff	4
6	MB0001	Ali	Male	Student	4
7	MB0005	K.Vince	Male	Student	4
8	MB0006	MacHill	Male	Staff	4
9	MB0007	J.Wenton	Prefer not to say	Student	2
10	MB0004	Jason	Prefer not to say	Staff	2

Explanation	
1.	"`SELECT MB.MemberID, MB.MemberName AS Name, MB.MemberGender AS Gender, DBM_Assignment.dbo.studentOrStaff(MB.MemberID) AS Role, COUNT(MB.MemberGender) OVER (PARTITION BY MemberGender) AS TotalMemberGenderBased`" - Selects five specific columns, which are "`MemberID`", "`MemberName`" (aliased as "`Name`"), "`MemberGender`" (aliased as "`Gender`"), "`DBM_Assignment.dbo.studentOrStaff(MB.MemberID)`" (aliased as "`Role`") that calls the "`studentOrStaff()`" function and passes the "`MB.MemberID`" as an argument; and the counts of member in each gender group in "`MemberGender`" (aliased as "`TotalMemberGenderBased`") is set into partitions, so it does not reduce the number of rows returned in the output, instead it shows the total number of members in each gender group in each row. ("`OVER (PARTITION BY MemberGender)`") from "`Members`" entity (aliased as "`MB`").
2.	"`FROM Members AS MB`" - The primary entity involved, which is "`Members`" entity (aliased as "`MB`").
3.	"`ORDER BY Gender ASC, MemberID ASC`" - Orders the results by the "`Gender`" in "`ASC`". - If multiple members have equal "`Gender`", orders them by "`MemberID`" in "`ASC`".

x. Show a list of ordered food which has not been delivered to members. The list should show member id, role(student/staff), contact number, food id, food name, quantity, date, and status of delivery.

Table 3.5.10 Select ordered food that is not delivered to members

<table>
<tr><th>SQL Statements</th><th>Executed Results</th></tr>
<tr><td>  </td><td> 
</td></tr>
<tr><td colspan="2">Explanation</td></tr>
<tr><td>1.</td><td>“<code>SELECT O.MemberID, DBM_Assignment.dbo.studentOrStaff(MB.MemberID) AS Role, MB.Name AS Name, MB.Email, MB.EmailPassword, CI.FoodID, M.FoodName, CI.Quantity, CI.ItemDateTime, OS.DeliveryStatus</code>”
- Selects ten specific columns, which are “<code>MemberID</code>” from “<code>Orders</code>” entity (aliased as “<code>O</code>”), “<code>DBM_Assignment.dbo.studentOrStaff(MB.MemberID)</code>” (aliased as “<code>Role</code>”) that calls the “<code>studentOrStaff()</code>” function and passes the “<code>MB.MemberID</code>” as an argument, “<code>Name</code>” (aliased as “<code>Name</code>”), “<code>Email</code>”, “<code>EmailPassword</code>” from the virtual table of “<code>EncryptedMember_vw</code>” view (aliased as “<code>MB</code>”), “<code>FoodID</code>” from “<code>CartItems</code>” entity (aliased as “<code>CI</code>”), “<code>FoodName</code>” from “<code>Menu</code>” entity (aliased as “<code>M</code>”), “<code>Quantity</code>”, “<code>ItemDateTime</code>” from “<code>CartItems</code>” entity (aliased as “<code>CI</code>”) and “<code>DeliveryStatus</code>” from “<code>OrderStatus</code>” entity (aliased as “<code>OS</code>”).</td></tr>
<tr><td>2.</td><td>“<code>FROM EncryptedMember_vw AS MB</code>”
- The primary entity involved, which is the virtual table showed by the “<code>EncryptedMember_vw</code>” view (aliased as “<code>MB</code>”).”)</td></tr>
<tr><td>3.</td><td>“<code>INNER JOIN Orders AS O</code>”
- “<code>INNER JOIN</code>” between the virtual table of “<code>EncryptedMember_vw</code>” view (<code>MB</code>) and the “<code>Orders</code>” entity (<code>O</code>).</td></tr>
</table>

4.	"`ON O.MemberID = MB.MemberID`" - The entity joining condition is "`O.MemberID = MB.MemberID`", where only "`MemberID`" in "`Orders`" entity match with "`MemberID`" in the virtual table of "`EncryptedMember_vw`" view.
5.	"`INNER JOIN OrderStatus AS OS`" - "`INNER JOIN`" between the "`Orders`" entity (`O`) and the "`OrderStatus`" entity (`OS`).
6.	"`ON O.OrderID = OS.OrderID AND OS.DeliveryStatus = 'Waiting'`" - The entity joining condition is "`O.OrderID = OS.OrderID`", where only "`OrderID`" in "`Orders`" entity match with "`OrderID`" in "`OrderStatus`" entity "`AND`" filters for orders where the "`DeliveryStatus`" in "`OrderStatus`" entity is equal to `'Waiting'`.
7.	"`INNER JOIN ShoppingCarts AS SC`" - "`INNER JOIN`" between the "`Orders`" entity (`O`) and the "`ShoppingCarts`" entity (`SC`).
8.	"`ON O.CartID = SC.CartID`" - The entity joining condition is "`O.CartID = SC.CartID`", where only "`CartID`" in "`Orders`" entity match with "`CartID`" in "`ShoppingCarts`" entity.
9.	"`INNER JOIN CartItems AS CI`" - "`INNER JOIN`" between the "`ShoppingCarts`" entity (`SC`) and the "`CartItems`" entity (`CI`).
10.	"`ON CI.CartID = SC.CartID`" - The entity joing condition is "`CI.CartID = SC.CartID`", where only "`CartID`" in "`CartItems`" entity match with "`CartID`" in "`ShoppingCarts`" entity.
11.	"`INNER JOIN Menu AS M`" - "`INNER JOIN`" between the "`CartItems`" entity (`CI`) and the "`Menu`" entity (`M`).

12.	"`ON CI.FoodID = M.FoodID`" - The entity joining condition is "`CI.FoodID = M.FoodID`", where only "`FoodID`" in "`CartItems`" entity match with "`FoodID`" in "`Menu`" entity.
13.	"`INNER JOIN CookedMeals AS CM`" - "`INNER JOIN`" between the "`CartItems`" entity (`CI`) and the "`CookedMeals`" entity (`CM`).
14.	"`ON CM.FoodID = CI.FoodID`" - The entity joining condition is "`CM.FoodID = CI.FoodID`", where only "`FoodID`" in "`CookedMeals`" entity match with "`FoodID`" in "`CartItems`" entity.
15.	"`WHERE CI.CartID <> CM.CartID OR (CI.CartID = CM.CartID AND CI.FoodID <> CM.FoodID)`" Filters the results to "`CI.CartID <> CM.CartID`" (`CI.CartID` is different to `CM.CartID`) "`OR`" "`(CI.CartID = CM.CartID AND CI.FoodID <> CM.FoodID)`" (`CI.CartID` is equal to `CM.CartID` "`AND`" `CI.FoodID` is different to `CM.FoodID`)

xi. Show a list of members who made more than 2 orders. The list should show their member id, name, and role(student/staff) and total orders.

Table 3.5.11 Select member who made more than 2 orders

SQL Statements	Executed Results
`--memberNumOrder(@numOrder INT, @operator NVARCHAR(2))` `SELECT MB.MemberID, MB.MemberName AS Name, DBM_Assignment.dbo.studentOrStaff(MB.MemberID) AS Role, MNO.TotalOrder` `FROM Members AS MB` `INNER JOIN memberNumOrder(2, '>') AS MNO` `ON MNO.MemberID = MB.MemberID;`	

	MemberID	Name	Role	TotalOrder
1	MB0007	J.Wenton	Student	3

Explanation	
1.	"`SELECT MB.MemberID, MB.MemberName AS Name, DBM_Assignment.dbo.studentOrStaff(MB.MemberID) AS Role, MNO.TotalOrder`" - Selects four specific columns, which are "`MemberID`", "`MemberName`" (aliased as "`Name`"), "`DBM_Assignment.dbo.studentOrStaff(MB.MemberID)`" (aliased as "`Role`") that calls the "`studentOrStaff()`" function and passes the

	"MB.MemberID" as an argument from "Members" entity (aliased as "MB") and "TotalOrder" from the function "memberNumOrder(2, '>')".
2.	"FROM Members AS MB" - The primary entity involved, which is "Members" entity (aliased as "MB").
3.	"INNER JOIN memberNumOrder(2, '>') AS MNO" - "INNER JOIN" between the "Members" entity (MB) and the tabular output of the "memberNumOrder(2, '>')" function (aliased as "MNO").
4.	"ON MNO.MemberID = MB.MemberID" - The entity joining condition is "MNO.MemberID = MB.MemberID", where only "MemberID" in the tabular output of the "memberNumOrder(2, '>')" function (aliased as "MNO") match with "MemberID" in "Members" entity.

xii. Find the monthly sales totals for the past year. The list should show order year, order month and total cost for that month.

Table 3.5.12 Select past year monthly sales totals

SQL Statements	Executed Results

```
SELECT YEAR(PY.PaymentDateTime) AS OrderYear, CASE MONTH(PY.PaymentDateTime)
    WHEN 1 THEN 'January'
    WHEN 2 THEN 'February'
    WHEN 3 THEN 'March'
    WHEN 4 THEN 'April'
    WHEN 5 THEN 'May'
    WHEN 6 THEN 'June'
    WHEN 7 THEN 'July'
    WHEN 8 THEN 'August'
    WHEN 9 THEN 'September'
    WHEN 10 THEN 'October'
    WHEN 11 THEN 'November'
    WHEN 12 THEN 'December'
END AS OrderMonth, SUM(PY.Amount) AS MonthlySales
FROM Payments AS PY
WHERE YEAR(PY.PaymentDateTime) = YEAR(GETDATE()) - 1
GROUP BY YEAR(PY.PaymentDateTime), MONTH(PY.PaymentDateTime);
```

	OrderYear	OrderMonth	MonthlySales
1	2023	March	80.10
2	2023	July	53.50

Explanation

1. "SELECT YEAR(PY.PaymentDateTime) AS OrderYear, CASE MONTH(PY.PaymentDateTime)

```
        WHEN 1 THEN 'January'
        WHEN 2 THEN 'February'
        WHEN 3 THEN 'March'
        WHEN 4 THEN 'April'
        WHEN 5 THEN 'May'
        WHEN 6 THEN 'June'
        WHEN 7 THEN 'July'
```

	WHEN 8 THEN 'August' WHEN 9 THEN 'September' WHEN 10 THEN 'October' WHEN 11 THEN 'November' WHEN 12 THEN 'December' END AS OrderMonth, SUM(PY.Amount) AS MonthlySales" - Selects three specific columns: - "YEAR()" extracted from "PaymentDateTime" (aliased as "OrderYear") - "CASE" statement is used to handle "WHEN...THEN" statements. Based on the number of the month extracted from "PaymentDateTime" (aliased as "OrderYear") via "MONTH()", it returns the output of the month in word as a string value. - "SUM()" of "Amount" from "Payments" entity (aliased as "PY") for each month (aliased by "MonthlySales").
2.	"FROM Payments AS PY" - The primary entity involved, which is "Payments" entity (aliased as "PY").
3.	"WHERE YEAR(PY.PaymentDateTime) = YEAR(GETDATE()) - 1" - Filters the results to include payments made only within the past year. - "YEAR(GETDATE())" retrieves the current year, "- 1" subtracts one from the current year to get the previous year.
4.	"GROUP BY YEAR(PY.PaymentDateTime), MONTH(PY.PaymentDateTime)" - "GROUP" the results by "YEAR(PY.PaymentDateTime)", "MONTH(PY.PaymentDateTime)", ensures the monthly sales for each combination of year and month is calculated.

4.0 Workload Matrix

Part	Component	Lim Chee Xuan	Paureen Tan Nie Nie	Phang Shea Wen	Total
2	a) Database Schema	$33\frac{1}{3}$ %	$33\frac{1}{3}$ %	$33\frac{1}{3}$ %	**100%**
2	b) SQL-Data Definition Language (DDL)	$33\frac{1}{3}$ %	$33\frac{1}{3}$ %	$33\frac{1}{3}$ %	**100%**
2	c) SQL-Data Manipulation Language (DML)	$33\frac{1}{3}$ %	$33\frac{1}{3}$ %	$33\frac{1}{3}$ %	**100%**