A · P · U

ASIA PACIFIC UNIVERSITY
OF TECHNOLOGY & INNOVATION

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

Intake code: **UCDF2309ICT(SE)**

Group Number: **Group 28**

Module Name: **Database Management**

Module Code: **022024-MHL**

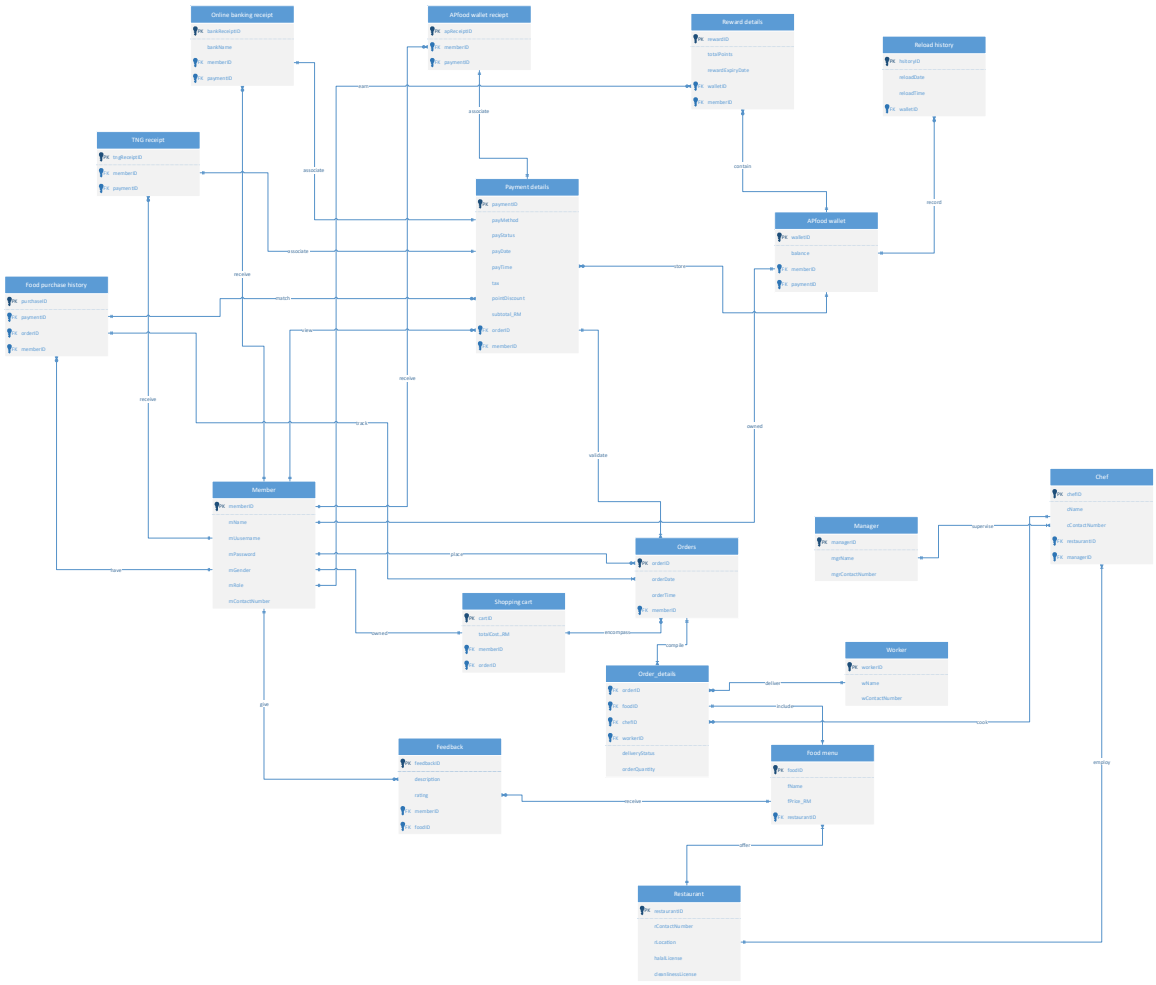
Lecturer: **Mr. Muhammad Huzaifa Ismail**

Group Member	**TP Number**
Ng Yvonne	TP076390
Lum Han Xun	TP076160
Heng Xin Hui	TP077232
Connie Puang Pei Qi	TP078412

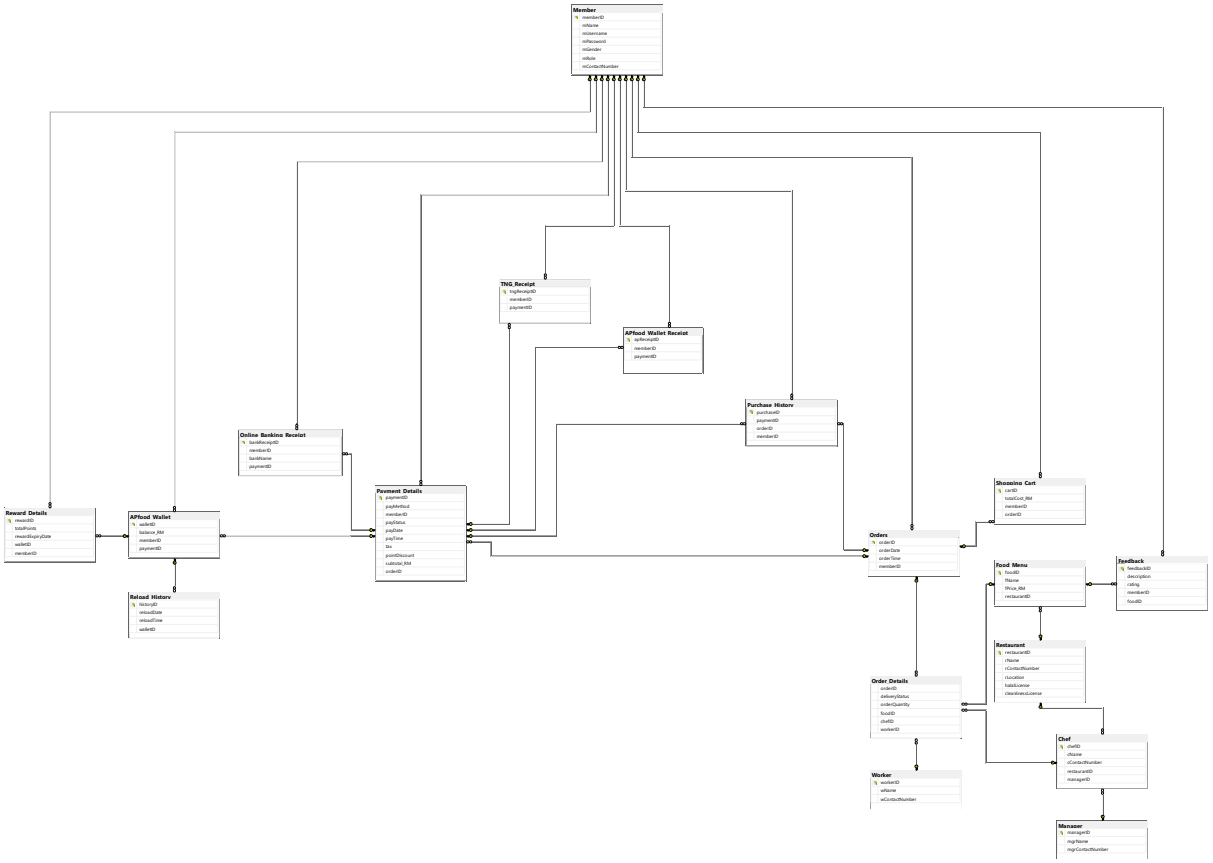
Table of Contents

1.0 Entity Relationship Diagram

Below is our new version of entity relationship diagram. Along the way we are completing this project, we discovered that the order version of ERD caused us unable retrieve some data from the tables. Thus, we had modified the 'Manager' table and added a bridge table: 'Order_Details'.



2.0 Database Diagram



3.0 Data Definition Language (DDL)

DDL is a prescribed language that include a set of SQL statements or commands to create, modify and remove database objects (Awati, 2022). The structure of database can be changed by the commands such as CREATE, ALTER, DROP, TRUNCATE and EXEC sp_rename (GeeksforGeeks, 2024). Below are the DDL that used in our database attached with respective results.

3.1 CREATE

Code block	**Explanation**
`CREATE DATABASE APU_Cafe;` `USE APU_Cafe;`	The 'CREATE' command is used to create a database for the APU Café Food Ordering System. Then, it has been used after the creation.
`CREATE TABLE Member (` `  memberID nvarchar(50) NOT NULL PRIMARY KEY,` `  mName nvarchar(50),` `  mUsername nvarchar(50),` `  mPassword nvarchar(50),` `  mGender nvarchar(10),` `  mRole nvarchar(50),` `  mContactNumber nvarchar(50)` `);`	Member table is created with its attributes and respective data types to store members' data. The attributes of this table including memberID, mName, mUsername, mPassword, mGender, mRole and mContactNumber.
`CREATE TABLE Restaurant (` `  restaurantID nvarchar(50) NOT NULL PRIMARY KEY,` `  rName nvarchar(50),` `  rContactNumber nvarchar(50),` `  rLocation nvarchar(50),` `  halalLicense nvarchar(50),` `  cleanlinessLicense nvarchar(50)` `);`	Restaurant table is created with its attributes and respective data types to store restaurants' data. The attributes of this table including restaurantID, rName, rContactNumber, rLocation, halalLicense and cleanlinessLicense.

`` CREATE TABLE Food_Menu (foodID nvarchar(50) NOT NULL PRIMARY KEY, fName nvarchar(50), fPrice_RM decimal(10,2), restaurantID nvarchar(50), FOREIGN KEY (restaurantID) REFERENCES Restaurant(restaurantID)); ```	Food_Menu table is created with its attributes and respective data types to store every food in each restaurant. This table contain one foreign key, which is restaurantID. The other attributes including foodID, fName and fPrice_RM.
`` CREATE TABLE Feedback (feedbackID nvarchar(50) NOT NULL PRIMARY KEY, description nvarchar(200), rating decimal(2,1), memberID nvarchar(50), foodID nvarchar(50), FOREIGN KEY (memberID) REFERENCES Member(memberID), FOREIGN KEY (foodID) REFERENCES Food_Menu(foodID)); ```	Feedback table is created with its attributes and respective data types to store every feedback of members. This table contain 2 foreign keys, which is memberID and foodID. The other attributes including feedbackID, description and rating.
`` CREATE TABLE Manager (managerID nvarchar(50) NOT NULL PRIMARY KEY, mgrName nvarchar(50), mgrContactNumber nvarchar(50)); ```	Manager table is created with its attributes and respective data types to store the manager's information. The attributes of this table including managerID, mgrName and mgrContactNumber.
`` CREATE TABLE Chef (chefID nvarchar(50) NOT NULL PRIMARY KEY, cName nvarchar(50), cContactNumber nvarchar(50), restaurantID nvarchar(50), managerID nvarchar(50), FOREIGN KEY (restaurantID) REFERENCES Restaurant(restaurantID), FOREIGN KEY (managerID) REFERENCES Manager(managerID)); ```	Chef table is created with its attributes and respective data types to store the chef's information. The attributes of this table including chefID, cName and cContactNumber. This table contain 2 foreign keys, which are restaurant ID and managerID.

`` CREATE TABLE Worker (workerID nvarchar(50) NOT NULL PRIMARY KEY, wName nvarchar(50), wContactNumber nvarchar(50)); ```	Worker table is created with its attributes and respective data types to store the worker's information. The attributes of this table including workerID, wName and wContactNumber.
`` CREATE TABLE Orders (orderID nvarchar(50) NOT NULL PRIMARY KEY, orderDate date, orderTime time, memberID nvarchar(50), FOREIGN KEY (memberID) REFERENCES Member(memberID)); ```	Orders table is created with its attributes and respective data types to store the orders' information. The attributes of this table including orderID, orderDate and orderTime. This table contain one foreign key, which is memberID.
`` CREATE TABLE Shopping_Cart (cartID nvarchar(50) NOT NULL PRIMARY KEY, totalCost_RM decimal(10,2), memberID nvarchar(50), orderID nvarchar(50), FOREIGN KEY (memberID) REFERENCES Member(memberID), FOREIGN KEY (orderID) REFERENCES Orders(orderID)); ```	Shopping_Cart table is created with its attributes and respective data types to store details of each cart. This table contain two foreign keys, which are memberID and orderID. The other attributes including cartID and totalCost_RM.
`` CREATE TABLE Order_Details (orderID nvarchar(50), deliveryStatus nvarchar(50), orderQuantity int, foodID nvarchar(50), chefID nvarchar(50), workerID nvarchar(50), FOREIGN KEY (orderID) REFERENCES Orders(orderID), FOREIGN KEY (foodID) REFERENCES Food_Menu(foodID), FOREIGN KEY (chefID) REFERENCES Chef(chefID), FOREIGN KEY (workerID) REFERENCES Worker(workerID)); ```	Order_Details table is created with its attributes and respective data types to store the details of every order in each restaurant. This table contain four foreign keys, which are orderID, foodID, chefID and workerID. The other attributes including deliveryStatus and orderQuantity

`` CREATE TABLE Payment_Details (paymentID nvarchar(50) NOT NULL PRIMARY KEY, payMethod nvarchar(50), memberID nvarchar(50), payStatus nvarchar(50), payDate date, payTime time, tax nvarchar(50), pointDiscount int, subtotal_RM decimal(10,2), orderID nvarchar(50), FOREIGN KEY (memberID) REFERENCES Member(memberID), FOREIGN KEY (orderID) REFERENCES Orders(orderID)); ```	Payment_Details table is created with its attributes and respective data types to store the details of every payment in each restaurant. This table contain two foreign keys, which are memberID and orderID. The other attributes including paymentID, payMethod, payStatus, payDate, payTime, tax, pointDiscount and subtotal_RM.
`` CREATE TABLE APfood_Wallet (walletID nvarchar(50) NOT NULL PRIMARY KEY, balance_RM decimal(10,2), memberID nvarchar(50), paymentID nvarchar(50), FOREIGN KEY (memberID) REFERENCES Member(memberID), FOREIGN KEY (paymentID) REFERENCES Payment_Details(paymentID)); ```	APfood_Wallet table is created with its attributes and respective data types to store the details of APfood_Wallet. This table contain two foreign keys, which are memberID and paymentID. The other attributes including walletID and balance_RM.
`` CREATE TABLE Reward_Details (rewardID nvarchar(50) NOT NULL PRIMARY KEY, totalPoints int, rewardExpiryDate date, walletID nvarchar(50), memberID nvarchar(50), FOREIGN KEY (memberID) REFERENCES Member(memberID), FOREIGN KEY (walletID) REFERENCES APfood_Wallet(walletID)); ```	The Reward_Details table is created with its attributes and respective data types to store the details of each reward. This table contain two foreign keys, which are memberID and walletID. The other attributes include rewardID, totalPoints and rewardExpiryDate.
`` CREATE TABLE Purchase_History (purchaseID nvarchar(50) NOT NULL PRIMARY KEY, paymentID nvarchar(50), orderID nvarchar(50), memberID nvarchar(50), FOREIGN KEY (paymentID) REFERENCES Payment_Details(paymentID), FOREIGN KEY (orderID) REFERENCES Orders(orderID), FOREIGN KEY (memberID) REFERENCES Member(memberID)); ```	The Purchase_History table is created with its attributes and respective data types to store the details of each purchase history. This table contain three foreign keys, which are paymentID, orderID and

	memberID. The other attributes include purchaseID.
`CREATE TABLE Reload_History (` `  historyID nvarchar(50) NOT NULL PRIMARY KEY,` `  reloadDate date,` `  reloadTime time,` `  walletID nvarchar(50),` `  FOREIGN KEY (walletID) REFERENCES APfood_Wallet(walletID)` `);`	The Reload_History table is created with its attributes and respective data types to store the details of each reload history. This table contain one foreign key, which is walletID. The other attributes include historyID, reloadDate and reloadTime.
`CREATE TABLE TNG_Receipt (` `  tngReceiptID nvarchar(50) NOT NULL PRIMARY KEY,` `  memberID nvarchar(50),` `  paymentID nvarchar(50),` `  FOREIGN KEY (memberID) REFERENCES Member(memberID),` `  FOREIGN KEY (paymentID) REFERENCES Payment_Details(paymentID)` `);`	The TNG_Receipt table is created with its attributes and respective data types to store the details of each TNG receipt. This table contain two foreign keys, which are memberID and paymentID. The other attributes include tngReceiptID.
`CREATE TABLE Online_Banking_Receipt (` `  bankReceiptID nvarchar(50) NOT NULL PRIMARY KEY,` `  memberID nvarchar(50),` `  bankName nvarchar(50),` `  paymentID nvarchar(50),` `  FOREIGN KEY (memberID) REFERENCES Member(memberID),` `  FOREIGN KEY (paymentID) REFERENCES Payment_Details(paymentID)` `);`	The Online_Banking_Receipt table is created with its attributes and respective data types to store the details of each online banking receipt. This table contain two foreign keys, which are memberID and paymentID. The other attributes include bankReceiptID and bankName.
`CREATE TABLE APfood_Wallet_Receipt (` `  apReceiptID nvarchar(50) NOT NULL PRIMARY KEY,` `  memberID nvarchar(50),` `  paymentID nvarchar(50),` `  FOREIGN KEY (memberID) REFERENCES Member(memberID),` `  FOREIGN KEY (paymentID) REFERENCES Payment_Details(paymentID)` `);`	The APfood_Wallet_Receipt table is created with its attributes and respective data types to store the details of each APfood wallet receipt. This table contain two foreign keys, which are memberID

	and paymentID. The other attributes include apReceiptID.

4.0 Data Manipulation Language (DML)

DML is the SQL commands that used to retrieve and manipulate the data in database, at the same time allows user to control access to database. SQL commands related to DML encompasses SELECT, INSERT, UPDATE, DELETE and so on (GeeksforGeeks, 2024). Below are the commands used in APU Café Food Ordering System.

4.1 INSERT

<table>
<tr><th>Code block</th><th>Explanation</th></tr>
<tr><td>

```
INSERT INTO Member VALUES
('M01','Harry','harry123','23h483ej','Male','Student','012-3345637'),
('M02','Matthew','5_matt_5','dj37dj9d','Male','Student','017-8051823'),
('M03','Evelyn','evelynnn','4372hd82','Female','Staff','012-4441016'),
('M04','Malfoy','888malfoy888','dihd8999','Male','Staff','018-8451584'),
('M05','Hermione','hErMiOnE','wdhd6dd','Female','Staff','011-8827320'),
('M06','Aurora','~aurora~','27dg7329','Female','Student','016-9207902'),
('M07','Johnson','john23','P@ssw0rd','Male','Lecturer','012-4835769'),
('M08','David','dave89','Sm1thP@ss','Male','Lecturer','018-5937042'),
('M09','Sarah','sarahlee7','L33s@rah','Female','Student','012-3289463'),
('M10','Turner','alex_turn','Turn3r!Pass','Male','Student','019-3785652');
```

Output table:

	memberID	mName	mUsername	mPassword	mGender	mRole	mContactNumber
1	M01	Harry	harry123	23h483ej	Male	Student	012-3345637
2	M02	Matthew	5_matt_5	dj37dj9d	Male	Student	017-8051823
3	M03	Evelyn	evelynnn	4372hd82	Female	Staff	012-4441016
4	M04	Malfoy	888malfoy888	dihd8999	Male	Staff	018-8451584
5	M05	Hermione	hErMiOnE	wdhd6dd	Female	Staff	011-8827320
6	M06	Aurora	~aurora~	27dg7329	Female	Student	016-9207902
7	M07	Johnson	john23	P@ssw0rd	Male	Lecturer	012-4835769
8	M08	David	dave89	Sm1thP@ss	Male	Lecturer	018-5937042
9	M09	Sarah	sarahlee7	L33s@rah	Female	Student	012-3289463
10	M10	Turner	alex_turn	Turn3r!Pass	Male	Student	019-3785652

</td><td>All related data is inserted into table Member.</td></tr>
<tr><td>

```
INSERT INTO Restaurant VALUES
('RT01','Savor Junction','03-2341722','south','HC28439','A'),
('RT02','Fizz and Sip Lounge','03-5930274','east','HC63826','A-'),
('RT03','Sweet Bliss Delight','03-2374920','west','HC90631','A+');
```

Output table:

	restaurantID	rName	rContactNumber	rLocation	halalLicense	cleanlinessLicense
1	RT01	Savor Junction	03-2341722	south	HC28439	A
2	RT02	Fizz and Sip Lounge	03-5930274	east	HC63826	A-
3	RT03	Sweet Bliss Delight	03-2374920	west	HC90631	A+

</td><td>All related data is inserted into table Restaurant.</td></tr>
</table>

```
INSERT INTO Food_Menu VALUES
('F01','Cheeseburger',4.50,'RT01'),
('F02','Molten lava cake',7.20,'RT03'),
('F03','Spaghetti',8.50,'RT01'),
('F04','Veggle wrap',6.00,'RT01'),
('F05','Iced lemon tea',3.50,'RT02'),
('F06','Green tea latte',6.90,'RT02'),
('F07','Fruit tart',4.30,'RT03'),
('F08','Americano',6.90,'RT02'),
('F09','Brownies',6.50,'RT03'),
('F10','Fried rice',7.00,'RT01'),
('F11','Pepperoni Pizza',10.00,'RT01'),
('F12','Sushi Combo',8.00,'RT01'),
('F13','Classic Mojito',3.00,'RT02'),
('F14','Strawberry Milkshake',3.00,'RT02'),
('F15','Tiramisu',4.50,'RT03');
```

Output table:

	foodID	fName	fPrice_RM	restaurantID
1	F01	Cheeseburger	4.50	RT01
2	F02	Molten lava cake	7.20	RT03
3	F03	Spaghetti	8.50	RT01
4	F04	Veggle wrap	6.00	RT01
5	F05	Iced lemon tea	3.50	RT02
6	F06	Green tea latte	6.90	RT02
7	F07	Fruit tart	4.30	RT03
8	F08	Americano	6.90	RT02
9	F09	Brownies	6.50	RT03
10	F10	Fried rice	7.00	RT01
11	F11	Pepperoni Pizza	10.00	RT01
12	F12	Sushi Combo	8.00	RT01
13	F13	Classic Mojito	3.00	RT02
14	F14	Strawberry Milkshake	3.00	RT02
15	F15	Tiramisu	4.50	RT03

All related data is inserted into table Food_Menu.

```
INSERT INTO Feedback VALUES
('E01','The taste is niceeee.',4.5,'M01','F05'),
('E02','The meat is undercooked!!!!!',2.0,'M01','F01'),
('E03','I wont be ordering this again.',1.5,'M02','F15'),
('E04','Definitely a must-try for pizza lovers!',4.9,'M02','F11'),
('E05','It is too sweet for me.',3.5,'M03','F02'),
('E06','The Sushi Combo was an absolute delight!',4.5,'M03','F12'),
('E07','The dish lacked flavor and had an unappealing texture,making it a disappointing choice.',2.0,'M03','F03'),
('E08','I hate cheese.',1.7,'M04','F03'),
('E09','The Americano was bold and invigorating,with a satisfyingly rich flavor profile.',4.5,'M04','F08'),
('E10','Highly recommended!',5.0,'M05','F04'),
('E11','It looks so cute and of course yummy.',4.0,'M05','F07'),
('E12','The food had a delightful medley of flavors and a pleasing texture that left a lasting impression.',5.0,'M06','F09'),
('E13','Delicious fried rice,well-seasoned and perfectly cooked.',4.8,'M07','F10'),
('E14','It is a perfect thirst-quencher on a warm summer day!',4.7,'M08','F13');
```

Output table:

	feedbackID	description	rating	memberID	foodID
1	E01	The taste is niceeee.	4.5	M01	F05
2	E02	The meat is undercooked!!!!!	2.0	M01	F01
3	E03	I wont be ordering this again.	1.5	M02	F15
4	E04	Definitely a must-try for pizza lovers!	4.9	M02	F11
5	E05	It is too sweet for me.	3.5	M03	F02
6	E06	The Sushi Combo was an absolute delight!	4.5	M03	F12
7	E07	The dish lacked flavor and had an unappealing te...	2.0	M03	F03
8	E08	I hate cheese.	1.7	M04	F03
9	E09	The Americano was bold and invigorating,with a s...	4.5	M04	F08
10	E10	Highly recommended!	5.0	M05	F04
11	E11	It looks so cute and of course yummy.	4.0	M05	F07
12	E12	The food had a delightful medley of flavors and a ...	5.0	M06	F09
13	E13	Delicious fried rice,well-seasoned and perfectly c...	4.8	M07	F10
14	E14	It is a perfect thirst-quencher on a warm summer ...	4.7	M08	F13

All related data is inserted into table Feedback.

`` INSERT INTO Manager VALUES ('MGR01','Cyrus','016-2858219'); ``` Output table: 		managerID	mgrName	mgrContactNumber
---	---	---	---
1	MGR01	Cyrus	016-2858219		All related data is inserted into table Manager.
`` INSERT INTO Chef VALUES ('C01','Marcus','018-4628293','RT01','MGR01'), ('C02','Nicholas','016-9106290','RT02','MGR01'), ('C03','Ryan','012-0385200','RT03','MGR01'); ``` Output table: 		chefID	cName	cContactNumber	restaurantID	managerID
---	---	---	---	---	---
1	C01	Marcus	018-4628293	RT01	MGR01
2	C02	Nicholas	016-9106290	RT02	MGR01
3	C03	Ryan	012-0385200	RT03	MGR01		All related data is inserted into table Chef.
`` INSERT INTO Worker VALUES ('W01','Jolin','012-3456789'), ('W02','Olivia','011-4752956'), ('W03','Jothan','014-9563850'); ``` Output table: 		workerID	wName	wContactNumber
---	---	---	---
1	W01	Jolin	012-3456789
2	W02	Olivia	011-4752956
3	W03	Jothan	014-9563850		All related data is inserted into table Worker.
`` INSERT INTO Orders VALUES ('OR-01','2023-01-05','13:08:45','M01'), ('OR-02','2023-02-16','16:28:52','M02'), ('OR-03','2023-03-08','10:51:38','M03'), ('OR-04','2023-03-11','14:13:25','M04'), ('OR-05','2023-05-17','09:49:01','M05'), ('OR-06','2023-05-29','15:07:43','M06'), ('OR-07','2023-06-02','12:01:44','M07'), ('OR-08','2023-06-02','12:33:41','M08'), ('OR-09','2023-06-13','11:27:31','M09'); ```	All related data is inserted into table Orders.

Output table:

	orderID	orderDate	orderTime	memberID
1	OR-01	2023-01-05	13:08:45	M01
2	OR-02	2023-02-16	16:28:52	M02
3	OR-03	2023-03-08	10:51:38	M03
4	OR-04	2023-03-11	14:13:25	M04
5	OR-05	2023-05-17	09:49:01	M05
6	OR-06	2023-05-29	15:07:43	M06
7	OR-07	2023-06-02	12:01:44	M07
8	OR-08	2023-06-02	12:33:41	M08
9	OR-09	2023-06-13	11:27:31	M09

```
INSERT INTO Shopping_Cart VALUES
('S01',8.00,'M01','OR-01'),
('S02',15.00,'M02','OR-02'),
('S03',36.70,'M03','OR-03'),
('S04',15.40,'M04','OR-04'),
('S05',14.80,'M05','OR-05'),
('S06',6.50,'M06','OR-06'),
('S07',7.00,'M07','OR-07'),
('S08',3.00,'M08','OR-08'),
('S09',10.00,'M09','OR-09'),
('S10',NULL,'M10',NULL);
```

All related data is inserted into table Shopping_Cart.

Output table:

	cartID	totalCost_RM	memberID	orderID
1	S01	8.00	M01	OR-01
2	S02	15.00	M02	OR-02
3	S03	36.70	M03	OR-03
4	S04	15.40	M04	OR-04
5	S05	14.80	M05	OR-05
6	S06	6.50	M06	OR-06
7	S07	7.00	M07	OR-07
8	S08	3.00	M08	OR-08
9	S09	10.00	M09	OR-09
10	S10	NULL	M10	NULL

```
INSERT INTO Order_Details VALUES
('OR-01','Out for delivery',1,'F05','C02','W01'),
('OR-01','In kitchen',2,'F01','C01','W02'),
('OR-02','Delivered',1,'F05','C02','W03'),
('OR-02','In kitchen',2,'F10','C01','W02'),
('OR-02','Unsuccessful',1,'F15','C03','W03'),
('OR-03','In kitchen',3,'F02','C03','W01'),
('OR-03','In kitchen',1,'F03','C01','W01'),
('OR-03','Out for delivery',1,'F12','C01','W03'),
('OR-03','Delivered',1,'F14','C02','W02'),
('OR-03','Unsuccessful',1,'F11','C01','W02'),
('OR-04','Delivered',2,'F03','C01','W01'),
('OR-04','Out for delivery',1,'F08','C02','W01'),
('OR-05','In kitchen',4,'F04','C01','W02'),
('OR-05','Unsuccessful',1,'F07','C03','W03'),
('OR-05','Out for delivery',1,'F15','C03','W03'),
('OR-06','Delivered',1,'F09','C03','W01'),
('OR-07','In kitchen',2,'F10','C01','W02'),
('OR-08','Out for delivery',1,'F13','C02','W03'),
('OR-09','Delivered',3,'F11','C01','W01');
```

Output table:

	orderID	deliveryStatus	orderQuantity	foodID	chefID	workerID
1	OR-01	Out for delivery	1	F05	C02	W01
2	OR-01	In kitchen	2	F01	C01	W02
3	OR-02	Delivered	1	F05	C02	W03
4	OR-02	In kitchen	2	F10	C01	W02
5	OR-02	Unsuccessful	1	F15	C03	W03
6	OR-03	In kitchen	3	F02	C03	W01
7	OR-03	In kitchen	1	F03	C01	W01
8	OR-03	Out for delivery	1	F12	C01	W03
9	OR-03	Delivered	1	F14	C02	W02
10	OR-03	Unsuccessful	1	F11	C01	W02
11	OR-04	Delivered	2	F03	C01	W01
12	OR-04	Out for delivery	1	F08	C02	W01
13	OR-05	In kitchen	4	F04	C01	W02
14	OR-05	Unsuccessful	1	F07	C03	W03
15	OR-05	Out for delivery	1	F15	C03	W03
16	OR-06	Delivered	1	F09	C03	W01
17	OR-07	In kitchen	2	F10	C01	W02
18	OR-08	Out for delivery	1	F13	C02	W03
19	OR-09	Delivered	3	F11	C01	W01

All related data is inserted into table Order_Details.

```
INSERT INTO Payment_Details VALUES
('PD01', 'TNG', 'M01', 'success', '2023-01-05', '13:14:30', '6%', 100, 7.48, 'OR-01'),
('PD02', 'Apfood wallet', 'M02', 'success', '2023-02-16', '16:50:11', '6%', 100, 14.90, 'OR-02'),
('PD03', 'Online banking', 'M03', 'success', '2023-03-08', '10:53:57', '6%', NULL, 38.90, 'OR-03'),
('PD04', 'Apfood wallet', 'M04', 'success', '2023-03-11', '17:30:39', '6%', 200, 14.32, 'OR-04'),
('PD05', 'Online banking', 'M05', 'success', '2023-05-17', '9:58:03', '6%', 500, 10.69, 'OR-05'),
('PD06', 'Apfood wallet', 'M06', 'success', '2023-05-29', '15:11:44', '6%', 100, 5.89, 'OR-06'),
('PD07', 'TNG', 'M07', 'success', '2023-06-02', '12:22:54', '6%', NULL, 7.42, 'OR-07'),
('PD08', 'Apfood wallet', 'M08', 'success', '2023-06-02', '12:38:42', '6%', NULL, 3.18, 'OR-08'),
('PD09', 'TNG', 'M09', 'success', '2023-06-13', '11:40:48', '6%', NULL, 10.60, 'OR-09');
```

All related data is inserted into table Payment_Details.

Output table:

	paymentID	payMethod	memberID	payStatus	payTime	tax	pointDiscount	subtotal_RM	orderID
1	PD01	TNG	M01	success	13:14:30	6%	100	7.48	OR-01
2	PD02	Apfood wallet	M02	success	16:50:11	6%	100	14.90	OR-02
3	PD03	Online banking	M03	success	10:53:57	6%	NULL	38.90	OR-03
4	PD04	Apfood wallet	M04	success	17:30:39	6%	200	14.32	OR-04
5	PD05	Online banking	M05	success	09:58:03	6%	500	10.69	OR-05
6	PD06	Apfood wallet	M06	success	15:11:44	6%	100	5.89	OR-06
7	PD07	TNG	M07	success	12:22:54	6%	NULL	7.42	OR-07
8	PD08	Apfood wallet	M08	success	12:38:42	6%	NULL	3.18	OR-08
9	PD09	TNG	M09	success	11:40:48	6%	NULL	10.60	OR-09

```
INSERT INTO APfood_Wallet VALUES
('WA01', 76.00, 'M01', NULL),
('WA02', 23.90, 'M02', 'PD02'),
('WA03', 18.30, 'M03', NULL),
('WA04', 50.00, 'M04', 'PD04'),
('WA05', 160.00, 'M05', NULL),
('WA06', 35.50, 'M06', 'PD06'),
('WA07', 78.00, 'M07', NULL),
('WA08', 67.00, 'M08', 'PD08'),
('WA09', 20.00, 'M09', NULL),
('WA10', 5.50, 'M10', NULL);
```

All related data is inserted into table APfood_Wallet.

Output table:

	walletID	balance_RM	memberID	paymentID
1	WA01	76.00	M01	NULL
2	WA02	23.90	M02	PD02
3	WA03	18.30	M03	NULL
4	WA04	50.00	M04	PD04
5	WA05	160.00	M05	NULL
6	WA06	35.50	M06	PD06
7	WA07	78.00	M07	NULL
8	WA08	67.00	M08	PD08
9	WA09	20.00	M09	NULL
10	WA10	5.50	M10	NULL

```
INSERT INTO Reward_Details VALUES
('R01', 50, '2025-01-05', 'WA01', 'M01'),
('R02', 200, '2025-02-16', 'WA02', 'M02'),
('R03', 42, '2025-03-08', 'WA03', 'M03'),
('R04', 35, '2025-03-11', 'WA04', 'M04'),
('R05', 120, '2025-05-17', 'WA05', 'M05'),
('R06', 351, '2025-05-29', 'WA06', 'M06'),
('R07', 224, '2025-06-02', 'WA07', 'M07'),
('R08', 78, '2025-06-02', 'WA08', 'M08'),
('R09', 90, '2025-06-13', 'WA09', 'M09'),
('R10', NULL, NULL, 'WA10', 'M10');
```

All related data is inserted into table Reward_Details.

Output table:

	rewardID	totalPoints	rewardExpiryDate	walletID	memberID
1	R01	50	2025-01-05	WA01	M01
2	R02	200	2025-02-16	WA02	M02
3	R03	42	2025-03-08	WA03	M03
4	R04	35	2025-03-11	WA04	M04
5	R05	120	2025-05-17	WA05	M05
6	R06	351	2025-05-29	WA06	M06
7	R07	224	2025-06-02	WA07	M07
8	R08	78	2025-06-02	WA08	M08
9	R09	90	2025-06-13	WA09	M09
10	R10	NULL	NULL	WA10	M10

```
INSERT INTO Purchase_History VALUES
('PH01', 'PD01', 'OR-01', 'M01'),
('PH02', 'PD02', 'OR-02', 'M02'),
('PH03', 'PD03', 'OR-03', 'M03'),
('PH04', 'PD04', 'OR-04', 'M04'),
('PH05', 'PD05', 'OR-05', 'M05'),
('PH06', 'PD06', 'OR-06', 'M06'),
('PH07', 'PD07', 'OR-07', 'M07'),
('PH08', 'PD08', 'OR-08', 'M08'),
('PH09', 'PD09', 'OR-09', 'M09');
```

All related data is inserted into table Purchase_History.

Output table:

	purchaseID	paymentID	orderID	memberID
1	PH01	PD01	OR-01	M01
2	PH02	PD02	OR-02	M02
3	PH03	PD03	OR-03	M03
4	PH04	PD04	OR-04	M04
5	PH05	PD05	OR-05	M05
6	PH06	PD06	OR-06	M06
7	PH07	PD07	OR-07	M07
8	PH08	PD08	OR-08	M08
9	PH09	PD09	OR-09	M09

```
INSERT INTO Reload_History VALUES
('RH01', '2022-12-29', '10:09:04', 'WA01'),
('RH02', '2023-02-10', '15:55:47', 'WA02'),
('RH03', '2023-03-13', '8:39:17', 'WA03'),
('RH04', '2023-03-20', '11:25:18', 'WA04'),
('RH05', '2023-04-27', '12:36:04', 'WA05'),
('RH06', '2023-05-10', '9:43:31', 'WA06'),
('RH07', '2023-05-18', '17:04:37', 'WA07'),
('RH08', '2023-06-29', '16:23:19', 'WA08'),
('RH09', '2023-07-01', '13:09:18', 'WA09'),
('RH10', NULL, NULL, 'WA10');
```

All related data is inserted into table Reload_History.

Output table:

	historyID	reloadDate	payTime	walletID
1	RH01	2022-12-29	10:09:04	WA01
2	RH02	2023-02-10	15:55:47	WA02
3	RH03	2023-03-13	08:39:17	WA03
4	RH04	2023-03-20	11:25:18	WA04
5	RH05	2023-04-27	12:36:04	WA05
6	RH06	2023-05-10	09:43:31	WA06
7	RH07	2023-05-18	17:04:37	WA07
8	RH08	2023-06-29	16:23:19	WA08
9	RH09	2023-07-01	13:09:18	WA09
10	RH10	NULL	NULL	WA10

```
INSERT INTO TNG_Receipt VALUES
('TNG01', 'M01', 'PD01'),
('TNG02', 'M06', 'PD07'),
('TNG03', 'M08', 'PD09');
```

All related data is inserted into table TNG_Receipt.

Output table:

	tngReceiptID	memberID	paymentID
1	TNG01	M01	PD01
2	TNG02	M06	PD07
3	TNG03	M08	PD09

```
INSERT INTO Online_Banking_Receipt VALUES
('OB01', 'M03', 'HongLeong Bank', 'PD03'),
('OB02', 'M05', 'CIMB Bank', 'PD05');
```

All related data is inserted into table Online_Banking_Receipt.

Output table:

	bankReceiptID	memberID	bankName	paymentID
1	OB01	M03	HongLeong Bank	PD03
2	OB02	M05	CIMB Bank	PD05

```
INSERT INTO Apfood_Wallet_Receipt VALUES
('AW01', 'M02', 'PD02'),
('AW02', 'M04', 'PD04'),
('AW03', 'M06', 'PD06'),
('AW04', 'M08', 'PD08');
```

All related data is inserted into table Apfood_Wallet_Receipt.

Output table:

	apReceiptID	memberID	paymentID
1	AW01	M02	PD02
2	AW02	M04	PD04
3	AW03	M06	PD06
4	AW04	M08	PD08

4.2 SELECT

Code block

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

```
SELECT Food_Menu.FoodID, Food_Menu.fName, Feedback.rating
FROM Food_Menu
INNER JOIN Feedback
ON Food_Menu.FoodID = Feedback.FoodID
WHERE rating = (SELECT MAX(rating) FROM Feedback);
```

Explanation:

The query above is used to retrieve food that has the highest rating. The 'SELECT' clause determine the columns that will be shown, which are foodID, fName and rating. The query 'FROM Food_Menu' indicates that Food_Menu is the primary table. 'INNER JOIN' is used to combine two tables and find the matched data only. The 'WHERE' clause is used for the requirements of the data to be executed. In this case, we used 'WHERE' clause to ensure only the maximum rating of food will be executed. Since the rating is from another table: Feedback, we retrieve the values of rating through subquery.

Executed Result:

	FoodID	fName	rating
1	F04	Veggle wrap	5.0
2	F09	Brownies	5.0

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

```
SELECT Member.memberID, Member.mName, COUNT(Feedback.feedbackID) AS total_feedback
FROM Member
LEFT JOIN Feedback
ON Member.memberID = Feedback.memberID
GROUP BY Member.memberID, Member.mName;
```

Explanation:

The query retrieves the member ID and member name from the 'Member' table and use the 'COUNT' function to calculate the total feedback count for each member. 'LEFT JOIN' is

employed to link the 'Member' and 'Feedback' tables based on the 'memberID' field. This ensures that all members are included in the output, even if they haven't submitted any feedback. The 'GROUP BY' clause is used to group the results by each member's ID and name, ensuring the feedback count is aggregated per member.

Executed Result:

	FoodID	fName	rating
1	F04	Veggle wrap	5.0
2	F09	Brownies	5.0

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

```
SELECT Member.memberID, Member.mName, COUNT(Orders.orderID) AS total_orders
FROM Member
LEFT JOIN Orders
ON Member.memberID = Orders.memberID
GROUP BY Member.memberID, Member.mName
HAVING COUNT(Orders.orderID) = 0;
```

Explanation:

The query retrieves the member ID and member name from the 'Member' table and calculates the total number of orders each member has placed using the 'COUNT' function. A 'LEFT JOIN' is used to combine the 'Member' and 'Orders' tables based on the 'memberID' field, ensuring all members are included in the result. 'GROUP BY' clause groups the results by each member's ID and name to aggregate the order count per member. The 'HAVING' clause filters the results to include only those members with a total order count of zero, thus identifying members who have not made any orders.

Executed Result:

	memberID	mName	total_orders
1	M10	Turner	0

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

```
SELECT Manager.managerID, Manager.mgrName AS Manager_Name, Chef.chefID, Chef.cName AS Chef_Name,
COUNT(Order_Details.foodID) AS Total_Meals_Ordered
FROM Order_Details
RIGHT JOIN Chef ON Order_Details.chefID = Chef.chefID
INNER JOIN Manager ON Chef.managerID = Manager.managerID
GROUP BY Manager.managerID, Manager.mgrName, Chef.chefID, Chef.cName
ORDER BY Manager.managerID, Chef.chefID;
```

Explanation:

The query retrieves the manager ID, manager name, chef ID, and chef name and the 'COUNT' function is used to count the number of food ordered that each chef has prepared under the supervision of each manager. The 'RIGHT JOIN' combines the 'Order_Details' and 'Chef' tables based on the 'chefID' field, ensuring all chefs are included in the result. The 'INNER JOIN' then links the 'Chef' table to the 'Manager' table using the 'managerID' field. The 'GROUP BY' clause groups the results by manager ID, manager name, chef ID, and chef name to ensure that the counts are aggregated per manager-chef pair. The 'ORDER BY' clause sorts the results by manager ID and chef ID for organized output.

Executed Result:

	managerID	Manager_Name	chefID	Chef_Name	Total_Meals_Ordered
1	MGR01	Cyrus	C01	Marcus	9
2	MGR01	Cyrus	C02	Nicholas	5
3	MGR01	Cyrus	C03	Ryan	5

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

```
SELECT Chef.chefID, Chef.cName AS cName, COUNT(Order_Details.orderID) AS total_meals
FROM Chef
LEFT JOIN Order_Details ON Chef.chefID = Order_Details.chefID
GROUP BY Chef.chefID, Chef.cName;
```

Explanation:

The SQL query is designed to find the total number of meals cooked by each chef. It selects the chef ID and chef name from the 'Chef' table and calculates the total number of meals cooked by each chef using the 'COUNT' function to count the order IDs associated with them in the 'Order_Details' table. A 'LEFT JOIN' combine the 'Chef' and 'Order_Details'

tables based on the ‘chefID’ field, ensuring all chefs are included in the result, even those who have not cooked any meals. ‘GROUP BY’ clause groups the results by chef ID and chef name to ensure that the meal counts are aggregated per chef.

Executed Result:

	chefID	cName	total_meals
1	C01	Marcus	9
2	C02	Nicholas	5
3	C03	Ryan	5

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

```
SELECT Food_Menu.FoodID, Food_Menu.fName, CAST(AVG(Feedback.rating) AS DECIMAL(10,2)) AS AverageRating
FROM Food_Menu
INNER JOIN Feedback
ON Food_Menu.FoodID = Feedback.FoodID
GROUP BY Food_Menu.FoodID, Food_Menu.fName
HAVING AVG(Feedback.rating) > (SELECT AVG(rating) FROM Feedback)
ORDER BY AverageRating DESC;
```

Explanation:

The query calculates the average of each food, group by food ID and food name to ensure that no multiple identical rating of same food item is calculated and retrieved. A ‘CAST’ function is applied, casting the average to a decimal with two decimal places for easy reading, and its value is labeled as ‘AverageRating’. The query then performs an inner join between the ‘Food_Menu’ and ‘Feedback’ table to retrive the data that matches both sides. The ‘HAVING’ clause is use to filter these grouped results, so that only those food items which average rating is higher than the overall average rating of all feedback ratings is returned. The results are ordered in decesding order based on the ‘AverageRating’ column to ensure foods with highest rating are listed first.

Executed Result:

	FoodID	fname	AverageRating
1	F04	Veggle wrap	5.00
2	F09	Brownies	5.00
3	F11	Pepperoni Pizza	4.90
4	F10	Fried rice	4.80
5	F13	Classic Mojito	4.70
6	F08	Americano	4.50
7	F12	Sushi Combo	4.50
8	F05	Iced lemon tea	4.50
9	F07	Fruit tart	4.00

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

```
SELECT Food_Menu.FoodID, Food_Menu.fName, Food_Menu.fPrice_RM, Order_Details.orderQuantity
FROM Food_Menu
INNER JOIN Order_Details
ON Food_Menu.FoodID = Order_Details.FoodID
WHERE Order_Details.orderQuantity IN(
    SELECT orderQuantity
    FROM(
        SELECT orderQuantity, DENSE_RANK() OVER (ORDER BY orderquantity DESC) AS orderQuantityrank
        FROM Order_Details
    ) AS subquery
    WHERE orderQuantityrank <= 3)
    ORDER BY orderQuantity DESC;
```

Explanation:

To find the top three bestselling food, ‘DENSE_RANK’ function is used to assign a unique rank to each row of data, and it will not skip values if multiple rows have the same value. Besides, it also orders the data in descending order. The main query then filters the results to select those food items with order quantity within the top three. The results are ordered by ‘orderquantity’ in descending order to ensure that the most frequently ordered items appear first.

Executed Result:

	FoodID	fname	fprice_RM	orderQuantity
1	F04	Veggle wrap	6.00	4
2	F02	Molten lava cake	7.20	3
3	F11	Pepperoni Pizza	10.00	3
4	F01	Cheeseburger	4.50	2
5	F10	Fried rice	7.00	2
6	F03	Spaghetti	8.50	2
7	F10	Fried rice	7.00	2

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

```
SELECT Member.memberID, Member.mName, Member.mRole
FROM Member
INNER JOIN Payment_Details
ON Member.memberID = Payment_Details.memberID
WHERE Member.memberID IN (
    SELECT TOP 3 memberID
    FROM Payment_Details
    GROUP BY memberID
    ORDER BY SUM(subtotal_RM)DESC)
GROUP BY Member.memberID, Member.mName, Member.mRole
ORDER BY SUM(subtotal_RM)DESC;
```

Explanation:

This query retrieves the top three members who have spent the most on the food orders. The subquery identifies the top three members by summing their total spending (the 'subtotal_RM' column), grouping by 'memberID' and ordering the result sets in descending order. The main query then filters the results to include only these top three members, group the result sets by 'memberID', 'mName' and 'mRole', and orders the final data sets by total spending in descending order.

Executed Result:

	memberID	mName	mRole
1	M03	Evelyn	Staff
2	M02	Matthew	Student
3	M04	Malfoy	Staff

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

```
SELECT COUNT(memberID) as Total_Gender, mGender FROM Member
GROUP BY mGender;
```

Explanation:

To display the result of how many members based on their genders, we used 'COUNT' to calculate the number of row of members and give the column name as 'Total_Gender'. The query also selects 'mGender' from Member table to show which gender each count corresponds to. The 'GROUP BY' clause group the rows in the Member table by 'mGender' column.

Executed Result:

	Total_Gender	mGender
1	4	Female
2	6	Male

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.

```
SELECT M.memberID, M.mRole, M.mContactNumber, F.foodID, F.fName, O.orderID, O.orderQuantity, O.deliveryStatus
FROM Member M
INNER JOIN Orders ON Orders.memberID = M.memberID
INNER JOIN Order_Details O ON O.orderID = Orders.orderID
INNER JOIN Food_Menu F ON F.foodID = O.FoodID
WHERE deliveryStatus IN (SELECT deliveryStatus FROM Order_Details WHERE deliveryStatus != 'Delivered');
```

Explanation:

The provided SQL query is sued to retrieve the details of members and their orders by filtering out which orders have not been delivered. It selects all the required columns from 3 different tables that have been joined and results only matched data between the tables to avoid data redundancy while the 'WHERE' clause here indicates only the food that have not been delivered will be selected.

Executed Result:

	memberID	mRole	mContactNumber	foodID	fName	orderID	orderQuantity	deliveryStatus
1	M01	Student	012-3345637	F05	Iced lemon tea	OR-01	1	Out for delivery
2	M01	Student	012-3345637	F01	Cheeseburger	OR-01	2	In kitchen
3	M02	Student	017-8051823	F10	Fried rice	OR-02	2	In kitchen
4	M02	Student	017-8051823	F15	Tiramisu	OR-02	1	Unsuccessful
5	M03	Staff	012-4441016	F02	Molten lava cake	OR-03	3	In kitchen
6	M03	Staff	012-4441016	F03	Spaghetti	OR-03	1	In kitchen
7	M03	Staff	012-4441016	F12	Sushi Combo	OR-03	1	Out for delivery
8	M03	Staff	012-4441016	F11	Pepperoni Pizza	OR-03	1	Unsuccessful
9	M04	Staff	018-8451584	F08	Americano	OR-04	1	Out for delivery
10	M05	Staff	011-8827320	F04	Veggle wrap	OR-05	4	In kitchen
11	M05	Staff	011-8827320	F07	Fruit tart	OR-05	1	Unsuccessful
12	M05	Staff	011-8827320	F15	Tiramisu	OR-05	1	Out for delivery
13	M07	Lecturer	012-4835769	F10	Fried rice	OR-07	2	In kitchen
14	M08	Lecturer	018-5937042	F13	Classic Mojito	OR-08	1	Out for delivery

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.

```
SELECT Member.memberID, Member.mName, Member.mRole, COUNT(Orders.orderID) AS total_orders
FROM Member
INNER JOIN Orders ON Member.memberID = Orders.memberID
GROUP BY Member.memberID, Member.mName, Member.mRole
HAVING COUNT(Orders.orderID) > 2;
```

Explanation:

The following SQL query is used to display members who made more than 2 orders in the list. Firstly, the member ID, member name and member role is selected because the list should display all the stated details. ‘COUNT’ is used to calculate all the order IDs to sum up the total amount of orders made by each member. Furthermore, an ‘INNER JOIN’ command is utilized to retrieve and merge the same data for ‘memberID’ in both tables, ‘Member’ and ‘Orders’ to ensure that there are no duplicate data for each member. Once both tables are joined, the member ID, member name and member role are grouped to produce an aggregated value of the total orders, followed by the ‘HAVING’ clause, where it is used to define aggregated values. In this case, more than 2 orders are an aggregated value. Hence, members who made 2 or less orders will be filtered out. The result does not display any value because no member has made 2 or more orders.

'Orders' table:

	orderID	orderDate	orderTime	memberID
1	OR-01	2023-01-05	13:08:45	M01
2	OR-02	2023-02-16	16:28:52	M02
3	OR-03	2023-03-08	10:51:38	M03
4	OR-04	2023-03-11	14:13:25	M04
5	OR-05	2023-05-17	09:49:01	M05
6	OR-06	2023-05-29	15:07:43	M06
7	OR-07	2023-06-02	12:01:44	M07
8	OR-08	2023-06-02	12:33:41	M08
9	OR-09	2023-06-13	11:27:31	M09

Executed Result:

memberID	mName	mRole	total_orders

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

```
SELECT YEAR(Payment_Details.payDate) AS order_year,
FORMAT(Payment_Details.payDate, 'MMMM') AS order_month,
SUM(Payment_Details.subtotal_RM) AS total_monthly_sales_RM
FROM Payment_Details
GROUP BY YEAR(Payment_Details.payDate), MONTH(Payment_Details.payDate), FORMAT(Payment_Details.payDate, 'MMMM')
ORDER BY YEAR(Payment_Details.payDate), MONTH(Payment_Details.payDate);
```

Explanation:

The query above is to calculate the monthly total sales made for the past year. 'payDate' in the table 'Payment_Details' is selected to display the year and the month of the sales. The 'FORMAT' clause is used to have the date set to a particular format. In this case, 'MMMM' represents the full name of a month. 'SUM' is to calculate the results of the total sales made for each month in RM. The year and full month name are then grouped and ordered according to the earliest month and year.

Executed Result:

	order_year	order_month	total_monthly_sales_RM
1	2023	January	7.48
2	2023	February	14.90
3	2023	March	53.22
4	2023	May	16.58
5	2023	June	21.20

5.0 References

Awati, R. (2022). Data definition language (DDL). https://www.techtarget.com/whatis/definition/Data-Definition-Language-DDL

GeeksforGeeks. (2024). SQL Commands | DDL, DQL, DML, DCL and TCL Commands. https://www.geeksforgeeks.org/sql-ddl-dql-dml-dcl-tcl-commands/#ddl-data-definition-language

AICT005-4-1 Database Systems – Workload Matrix

Part	Component	Student 1 Name: Ng Yvonne	Student 2 Name: Lum Han Xun	Student 3 Name: Heng Xin Hui	Student 4 Name: Connie Puang Pei Qi	Total
2	a. Database Schema	100%	0%	0%	0%	**100%**
2	b. SQL-Data Definition Language (DDL)	40%	20%	20%	20%	**100%**
2	c. SQL-Data Manipulation Language (DML)	30%	25%	23%	22%	**100%**

*Individual component is omitted because marks will be awarded separately
*Ignore student 4 column if there are only 3 members
*Total % of contribution of all members must sum up to 100%