Advanced Data Management

ASSIGNMENT 2

Group Members

Kiu May Yee – 1211405279

Iymun Bashir – 1132702565

TABLE OF CONTENTS

Table Creation and Data Insertion

Dimension Tables

Customer Dimension Table - CustDim

```
create table CustDim (
Cust_ID INT PRIMARY key IDENTITY(1,1),
Cust_Name varchar(100) not null,
CustContact varchar(20) not null,
Cust_Pword varchar(40) not null)
```

Data Insertion

```
INSERT INTO CustDim (Cust_Name, CustContact, Cust_Pword)
VALUES ('Praset','(+66)8-12345678','PassW*12345'),
('Siti','(+60)14-3334567', 'Siti^12345'),
('DAvid','(+66)8-2346789', 'Dav!6789')
('Mary', '(+60}17-1234555', 'Abc#12345');
```

	Cust_ID	Cust_Name	CustContact	Cust_Pword
1	1	Praset	(+66)8-12345678	PassW*12345
2	2	Siti	(+60)14-3334567	Siti^12345
3	3	DAvid	(+66)8-2346789	Dav!6789
4	4	Mary	(+60}17-1234555	Abc#12345

Payment Dimension Table - PaymentDim

```
create table PaymentDim (
Payment_ID INT PRIMARY key IDENTITY(1,1),
Payment_Method varchar(100) not null,
Payment_Currency varchar(20) not null)
```

Data Insertion

```
INSERT INTO PaymentDim (Payment_Method, Payment_Currency)
VALUES
('Credit/Debit Card', 'RM'),
('Cash', 'RM'),
('E-wallet', 'RM'),
('Credit/Debit Card', 'THB'),
('Cash', 'THB'),
('E-wallet', 'THB');
```

	Payment_ID	Payment_Method	Payment_Currency
1	1	Credit/Debit Card	RM
2	2	Cash	RM
3	3	E-wallet	RM
4	4	Credit/Debit Card	THB
5	5	Cash	THB
6	6	E-wallet	THB

Driver Dimension Table - DriverDim

```
create table DriverDim (

Driver_ID INT PRIMARY key IDENTITY(1,1),

DriverName varchar(50) not null,

Driver_Contact varchar(20) not null,

Driver_PlateNum varchar(10) not null)
```

Data Insertion

```
INSERT INTO DriverDim (DriverName, Driver_Contact,
Driver_PlateNum)

VALUES

('Alex','(+60)14-3456789','ABC-234'),

('Anong', '(+66)8-34567890','b-7760'),

('AHmad', '(+60)13-5672345','VJ-7676')

('Ali','(+60)16-1234567', 'ABC-123')
```

	Driver_ID	DriverName	Driver_Contact	Driver_PlateNum
1	1	Alex	(+60)14-3456789	ABC-234
2	2	Anong	(+66)8-34567890	b-7760
3	3	AHmad	(+60)13-5672345	VJ-7676
4	4	Ali	(+60)16-1234567	ABC-123

Delivery Dimension Table - DeliveryAddressDim

```
create table DeliveryAddress_Dim (

del_addressID INT PRIMARY key IDENTITY(1,1),

delivery_state varchar(100),

delivery_city varchar(100),

delivery_street varchar(100),

delivery_zip varchar(10),

delivery_address varchar(200))
```

Data Insertion

```
INSERT INTO DeliveryAddress_Dim (delivery_state,delivery_city,
delivery_street,delivery_zip,delivery_address)

VALUES

('','','','', 'Municipality 1,JBang Bua
Thong,Nonthaburi,11110,Thailand'),

('Melaka','Ayer Keroh','Jalan Kota Fesyen 1','75450',''),

('Selangor','Sentul','Jalan Abu','12300',''),

('Pulau Pinang','Jelutong','Jalan Perak','10150','')
```

	del_addressID	delivery_state	delivery_city	delivery_street	delivery_address	delivery_zip
1	1				Municipality 1,JBang Bua Thong,Nonthaburi,11110,...	
2	2	Melaka	Ayer Keroh	Jalan Kota Fesyen 1		75450
3	3	Selangor	Sentul	Jalan Abu		12300
4	4	Pulau Pinang	Jelutong	Jalan Perak		10150

Restaurant Dimension Table - RestaurantDim

```
create table RestaurantDim (

restaurant_ID INT PRIMARY key IDENTITY(1,1),

restName varchar(100) not null,

restAddress varchar(100) not null

restTel varchar(100) not null),
```

Data Insertion

```
INSERT INTO RestaurantDim (restName,restAddress,restTel)

VALUES

('Mingle Cafe','51, Jln Damansara, Kuala Lumpur Sentral, 50490
Kuala Lumpur, Wilayah Persekutuan Kuala Lumpur','(+60)19-
2727755'),

('thursdvys', '22, Lorong Datuk Sulaiman 1, Taman Tun Dr
Ismail, 60000 Kuala Lumpur, Wilayah Persekutuan Kuala
Lumpur','(+60)3-77336722'),

('Floral Café at Napasorn','67 Chakkraphet Rd, Khwaeng Wang
Burapha Phirom, Khet Phra Nakhon, Krung Thep Maha Nakhon 10200,
Thailand','(+66)2 222 6895'),

('Dominos Pizza','599 Charoen Krung Rd, Pom Prap, Pom Prap
Sattru Phai, Bangkok 10100, Thailand','(+66)2-1614500'),

('Oriental Food Factory','57, Jalan Zainal Abidin, 10400 George
Town, Pulau Pinang','');
```

	restaurant_ID	restName	restAddress	restTel
1	1	Mingle Cafe	51, Jln Damansara, Kuala Lumpur Sentral, 50490 Kuala Lumpur, Wilayah Persekutuan Kuala Lumpur	(+60)19-2727755
2	2	thursdvys	22, Lorong Datuk Sulaiman 1, Taman Tun Dr Ismail, 60000 Kuala Lumpur, Wilayah Persekutuan Kuala Lumpur	(+60)3-77336722
3	3	Floral Café at Napasorn	67 Chakkraphet Rd, Khwaeng Wang Burapha Phirom, Khet Phra Nakhon, Krung Thep Maha Nakhon 10200, Thailand	(+66)2 222 6895
4	4	Dominos Pizza	599 Charoen Krung Rd, Pom Prap, Pom Prap Sattru Phai, Bangkok 10100, Thailand	(+66)2-1614500
5	5	Oriental Food Factory	57, Jalan Zainal Abidin, 10400 George Town, Pulau Pinang	

Menu Items Dimension Table - MenuItem_Dim

```
create table MenuItem_Dim (
MenuItem_ID INT PRIMARY key IDENTITY(1,1),
MenuItem_name varchar(100) not null,
MenuItem_Price varchar(100) not null)
```

Data Insertion

```
INSERT INTO MenuItem_Dim (MenuItem_name,MenuItem_Price)
VALUES
('Lemon Tart',100.0),
('Mingle Chicken Breakfast Burger',25.0),
('House Chicken Salad', 18.0),
('Hawaiian Pizza - Large', 300.0),
('Korean BBQ Chicken',12.90);
```

	MenuItem_ID	MenuItem_name	MenuItem_Price
1	1	Lemon Tart	100.0
2	2	Mingle Chicken Breakfast Burger	25.0
3	3	House Chicken Salad	18.0
4	4	Hawaiian Pizza - Large	300.0
5	5	Korean BBQ Chicken	12.90

Customer Review Dimension Table - CustomerReviewDim

```
create table CustomerReviewDim (
reviewID INT PRIMARY key IDENTITY(1,1),
cust_remakrs text,
cust_rating tinyint not null)
```

Data Insertion

```
INSERT INTO CustomerReviewDim (cust_remarks, cust_rating)
VALUES
('The lime taste is too strong.',3),
('Best burger in town.',5),
('Good big sized burger. Taste good.',5),
('Pizza is crispy and yummy.',5)
('',4);
```

	reviewID	cust_remarks	cust_rating
1	1	The lime taste is too strong.	3
2	2	Best burger in town.	5
3	3	Good big sized burger. Taste good.	5
4	4	Pizza is crispy and yummy.	5
5	5		4

Date Dimension Table - DateDim

```
Create table DateDim (
[DateKey] int PRIMARY key NOT NULL,
[DayOfMonth] VARCHAR(2),
[DayName] VARCHAR(9),
[WeekOfMonth] VARCHAR(1),
[Month] VARCHAR(2),
[MonthName] VARCHAR(9),
[Quarter] CHAR(1),
[QuarterName] VARCHAR(9),
[Year] CHAR(4)
```

Data Insertion

```
SELECT [@Order_Time] FROM OrderFact
DECLARE @EndDate DATETIME = '01/01/2022'
DECLARE
    @WeekOfMonth INT,
    @CurrentYear INT,
    @CurrentMonth INT,
    @CurrentQuarter INT
DECLARE @CurrentDate AS DATETIME = @Order_Time
SET @CurrentMonth = DATEPART(MM, @CurrentDate)
SET @CurrentYear = DATEPART(YY, @CurrentDate)
SET @CurrentQuarter = DATEPART(QQ, @CurrentDate)

WHILE @CurrentDate < @EndDate
```

```
INSERT INTO [DateDim]
SELECT
CONVERT (char(8),@CurrentDate,112) as DateKey,
@CurrentDate AS Date,
CONVERT (char(10),@CurrentDate,101) as FullDate,
DATEPART(DD, @CurrentDate) AS DayOfMonth,
DATENAME(DW, @CurrentDate) AS DayName,
DATEPART(WW, @CurrentDate) + 1 - DATEPART(WW, CONVERT(VARCHAR,
DATEPART(MM, @CurrentDate)) + '/1/' + CONVERT(VARCHAR,
DATEPART(YY, @CurrentDate))) AS WeekOfMonth,
(DATEDIFF(DD, DATEADD(QQ, DATEDIFF(QQ, 0, @CurrentDate), 0),
DATEPART(MM, @CurrentDate) AS Month,
DATENAME(MM, @CurrentDate) AS MonthName,
DATEPART(QQ, @CurrentDate) AS Quarter,

CASE DATEPART(QQ, @CurrentDate)
    WHEN 1 THEN 'First'
    WHEN 2 THEN 'Second'
    WHEN 3 THEN 'Third'
    WHEN 4 THEN 'Fourth'
END AS QuarterName,
DATEPART(YEAR, @CurrentDate) AS Year,

SET @CurrentDate = DATEADD(DD, 1, @Order_Time)
```

Fact Table

OrderFact

```
create table OrderFact (

Order_ID INT PRIMARY key IDENTITY(1,1),

Cust_ID INT, Payment_ID INT, restaurant_ID INT, MenuItem_ID INT,

DriverID INT, reviewID INT, del_addressID INT, ordertimeID INT,
deliverytimeID INT,

payment_time DATETIME, payment_status VARCHAR(20), Order_status
VARCHAR(50),

Order_amt INT, Order_Time DATETIME, food_qty INT, Order_currency
CHAR(20),



FOREIGN KEY (Cust_ID) REFERENCES  CustDim(Cust_ID),

FOREIGN KEY (Payment_ID) REFERENCES PaymentDim(Payment_ID),

FOREIGN KEY (restaurant_ID) REFERENCES RestaurantDim(restaurant_ID),

FOREIGN KEY (MenuItem_ID) REFERENCES MenuItem_Dim(MenuItem_ID),

FOREIGN KEY (DriverID) REFERENCES DriverDim(Driver_ID),

FOREIGN KEY (reviewID) REFERENCES CustomerReviewDim(reviewID),

FOREIGN KEY (del_addressID) REFERENCES
DeliveryAddress_Dim(del_addressID));
```

Data Insertion

```
INSERT INTO OrderFact
(Cust_ID,Payment_ID,restaurant_ID,MenuItem_ID,DriverID,reviewID
,del_addressID,ordertimeID,deliverytimeID,payment_time,payment_
status,Order_status,

Order_amt,Order_Time,food_qty,Order_currency)
```

```
VALUES

(1,4,4,4,2,4,1,'','','2022-05-23
15:05','complete','complete',600,'2022-05-23 15:00',2,'THB'),

(3,6,3,1,2,5,1,'','','2022-03-17
09:15','complete','complete',200,'2022-03-17 08:57',2,'THB'),

(2,1,1,2,4,2,3,'','','2022-04-09
17:45','complete','complete',25,'2022-04-09 17:30',1,'MY'),

(4,3,2,3,1,5,3,'','','2022-07-31
14:33','complete','complete',54,'2022-07-31 14:15',3,'MY'),

(2,3,5,5,4,5,4,'','','2022-08-30
19:30','complete','complete',12.90,'2022-08-30 19:17',1,'MY'),

(1,5,4,4,2,4,1,'','','2022-04-10
10:20','complete','complete',300,'2022-04-10 10:10',1,'THB'),

(3,6,4,4,2,4,1,'','','2022-06-22
19:10','complete','complete',600,'2022-06-22 19:01',2,'THB'),

(4,2,5,5,4,5,4,'','','2022-01-15
12:00','complete','complete',38.70,'2022-01-15 11:45',3,'MY')
```

	Order_ID	Cust_ID	Payment_ID	restaurant_ID	MenuItem_ID	DriverID	reviewID	del_addressID	ordertimeID	deliverytimeID	payment_time	payment_status	Order_status	Order_amt	Order_Time	food_qty	Order_currer
1	1	1	4	4	4	2	4	1	0	0	2022-05-23 15:05:00.000	complete	complete	600	2022-05-23 15:00:00.000	2	THB
2	2	2	1	1	2	4	2	3	0	0	2022-04-09 17:45:00.000	complete	complete	25	2022-04-09 17:30:00.000	1	MY
3	3	4	3	2	3	1	5	3	0	0	2022-07-31 14:33:00.000	complete	complete	54	2022-07-31 14:15:00.000	3	MY
4	4	2	3	5	5	4	5	4	0	0	2022-08-30 19:30:00.000	complete	complete	12	2022-08-30 19:17:00.000	1	MY
5	5	1	5	4	4	2	4	1	0	0	2022-04-10 10:20:00.000	complete	complete	300	2022-04-10 10:10:00.000	1	THB
6	6	3	6	4	4	2	4	1	0	0	2022-06-22 19:10:00.000	complete	complete	600	2022-06-22 19:01:00.000	2	THB
7	7	4	2	5	5	4	5	4	0	0	2022-01-15 12:00:00.000	complete	complete	38	2022-01-15 11:45:00.000	3	MY

Question 3 - Queries

a) Group by with rollup and order by clauses

This type of query shows hierarchical ways of aggregating results.

Answer 1

For our first answer, a combination of payment methods used in each country and in both countries is calculated.

- Payment method
- Payment method + each country /overall
- In overall (all payment method + all countries)

```
SELECT A.Order_currency as Country,B.Payment_Method
PaymentMethod, count(A.Cust_ID) Customer,SUM(A.Order_amt)
TotalOrderAmount

FROM OrderFact as A

inner join PaymentDim as B on A.Payment_ID = B.Payment_ID

  group by rollup ( [Order_currency],[Payment_Method])

  order by Payment_Method desc
```

Results | Messages

	Country	PaymentMethod	Customer	TotalOrderAmount
1	MY	E-wallet	2	66
2	THB	E-wallet	1	600
3	NULL	E-wallet	3	666
4	MY	Credit/Debit Card	1	25
5	THB	Credit/Debit Card	1	600
6	NULL	Credit/Debit Card	2	625
7	MY	Cash	1	38
8	THB	Cash	1	300
9	NULL	Cash	2	338
10	NULL	NULL	7	1629

Answer 2

For our second query, combination of payment methods used in each country and in both countries as calculated.

- Number of Order and Total of Order Amount generated by every driver + each country
- Number of Order and Total of Order Amount generated in each country
- Number of Order and Total of Order Amount generated in overall

```
SELECT A.Order_currency as Country,B.DriverName,
Count(A.Order_amt) TotalOrder, Sum(A.Order_amt)
TotalOrderAmount

FROM OrderFact as A

inner join DriverDim as B on A.Payment_ID = B.Driver_ID

  group by rollup ( [Order_currency],[DriverName])

  order by DriverName desc
```

	Country	DriverName	TotalOrder	TotalOrderAmount
1	MY	Anong	1	38
2	THB	Ali	1	600
3	MY	Alex	1	25
4	MY	AHmad	2	66
5	MY	NULL	4	129
6	THB	NULL	1	600
7	NULL	NULL	5	729

b) Group by with cube and order by clauses

Group by cube is used to show all the possible combinations of selected columns.

Answer 1

This query shows each payment method used in each country (by each customer), as well as the overall number of customers placing orders in Malaysia and Thailand respectively.

```
SELECT A.Order_currency as
Country,B.Payment_Method
PaymentMethod, count(A.Cust_ID)
Customer,SUM(A.Order_amt)
TotalOrderAmount

        FROM OrderFact as A

        inner join PaymentDim as B
on A.Payment_ID = B.Payment_ID

  group by cube ( [Payment_Method]
,[Order_currency])

  order by Payment_Method desc
```

Results | Messages

	Country	PaymentMethod	Customer	TotalOrderAmount
1	MY	E-wallet	2	66
2	THB	E-wallet	1	600
3	NULL	E-wallet	3	666
4	NULL	Credit/Debit Card	2	625
5	THB	Credit/Debit Card	1	600
6	MY	Credit/Debit Card	1	25
7	MY	Cash	1	38
8	THB	Cash	1	300
9	NULL	Cash	2	338
10	THB	NULL	3	1500
11	NULL	NULL	7	1629
12	MY	NULL	4	129

Answer 2

Besides from the available combinations shown from group by roll up, it also shows by

- Number of Order and Total of Order Amount generated by every driver

```
SELECT A.Order_currency as
Country,B.DriverName, Count(A.Order_amt)
TotalOrder, Sum(A.Order_amt)
TotalOrderAmount

FROM OrderFact as A

inner join DriverDim as B on A.Payment_ID
= B.Driver_ID

  group by cube (
[Order_currency],[DriverName])

  order by DriverName desc
```

	Country	DriverName	TotalOrder	TotalOrderAmount
1	MY	Anong	1	38
2	NULL	Anong	1	38
3	THB	Ali	1	600
4	NULL	Ali	1	600
5	MY	Alex	1	25
6	NULL	Alex	1	25
7	MY	AHmad	2	66
8	NULL	AHmad	2	66
9	NULL	NULL	5	729
10	MY	NULL	4	129
11	THB	NULL	1	600

c) Stored procedure

Stored procedures are prepared SQL code that is saved and able to be reused. They can perform one or more operations on the database and return any values.

Stored procedures are easy to modify as modifications apply to all related scripts. They are stored in the database data dictionary, making them available to all applications using the DBMS.

Stored Procedure - Answer 1

For our first answer, we created a stored procedure that can execute a reusable command to show the details of the order, in terms of the restaurant and driver details of each order.

```
create procedure getRestaurantdetails_withanOrderID
(@OrderID INT)
AS
BEGIN
SET NOCOUNT ON;


SELECT P.Order_ID,P.restaurant_ID,R.restName,R.restAddress,
from OrderFact P
INNER JOIN RestaurantDim R on P.restaurant_ID = R.restaurant_ID
where P.Order_ID = @OrderID
end
Go


EXEC getRestaurantdetails_withanOrderID 7
EXEC getRestaurantdetails_withanOrderID 1
```

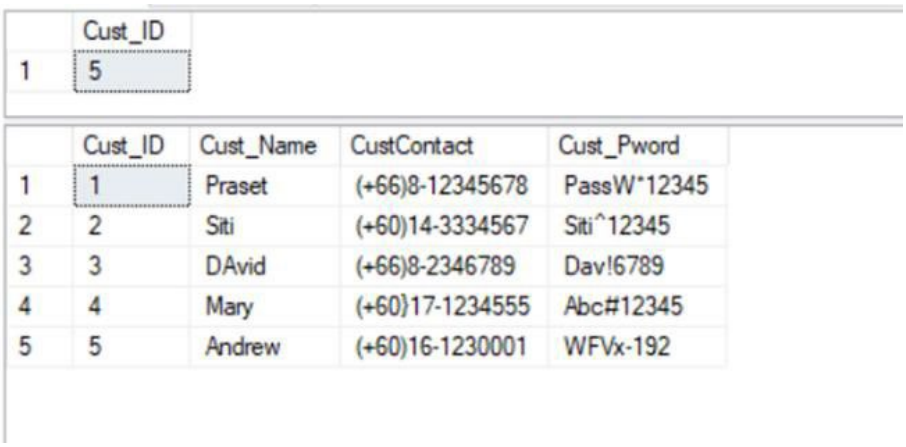
	Order_ID	restName	restAddress	DriverName	Driver_Contact	Order_status
1	7	Dominos Pizza	599 Charoen Krung Rd, Pom Prap, Pom Prap Sattru ...	Anong	(+66)8-34567890	Complete

	Order_ID	restName	restAddress	DriverName	Driver_Contact	Order_status
1	1	Dominos Pizza	599 Charoen Krung Rd, Pom Prap, Pom Prap Sattru ...	Anong	(+66)8-34567890	

Stored Procedure Answer 2

By using this stored procedure getcustomerID, we can insert new customer information and obtain an auto-generated customer ID.

```
GO

CREATE PROCEDURE [dbo].[getcustomerID]

(@Custname varchar(50),@CustPassword varchar(40),

@custContact varchar(20),

@Cust_ID int output)

AS

BEGIN

SET NOCOUNT ON

INSERT INTO CustDim (Cust_Name,Cust_Pword,CustContact) VALUES
(@Custname,@CustPassword,@custContact)

SELECT @Cust_ID= SCOPE_IDENTITY()

END



declare @Cust_ID INT, @Custname varchar(50),@CustPassword varchar(40),

@custContact varchar(20)



EXEC getcustomerID 'Andrew','WFVx-192','(+60)16-1230001', @Cust_ID
OUTPUT



SELECT @Cust_ID as Cust_ID

Select* from CustDim
```



	Cust_ID
1	5

	Cust_ID	Cust_Name	CustContact	Cust_Pword
1	1	Praset	(+66)8-12345678	PassW*12345
2	2	Siti	(+60)14-3334567	Siti^12345
3	3	DAvid	(+66)8-2346789	Dav!6789
4	4	Mary	(+60)17-1234555	Abc#12345
5	5	Andrew	(+60)16-1230001	WFVx-192

Stored Procedure Answer 3

Our date dimension table code also works as a stored procedure to generate a date dimension table that is usable in most any date warehouse.

The code has been changed into a more generalised form that can be tweaked as needed to use whatever data source that is needed.

Of note, @CurrentDate has to be set to the correct source attribute before data insertion is executed, the current code sets @CurrentDate to @StartDate which is set to a random date just to showcase the SP and for reference of how it works.

Create table

```
Create table DateDim (
[DateKey] int PRIMARY key NOT NULL,
[DayOfMonth] VARCHAR(2), -- Field will hold day number of Month
[DayName] VARCHAR(9), -- Contains name of the day, Sunday, Monday
[WeekOfMonth] VARCHAR(1), -- Week Number of Month
[Month] VARCHAR(2), --Number of the Month 1 to 12
[MonthName] VARCHAR(9), --January, February etc
```

	[Quarter] CHAR(1), -- First,Second.. [QuarterName] VARCHAR(9), [Year] CHAR(4)
Declarations	DECLARE @StartDate DATETIME = '01/01/2010' DECLARE @EndDate DATETIME = '01/01/2030' DECLARE @WeekOfMonth INT, @CurrentYear INT, @CurrentMonth INT, @CurrentQuarter INT DECLARE @CurrentDate AS DATETIME = @StartDate SET @CurrentMonth = DATEPART(MM, @CurrentDate) SET @CurrentYear = DATEPART(YY, @CurrentDate) SET @CurrentQuarter = DATEPART(QQ, @CurrentDate)
Condition	WHILE @CurrentDate < @EndDate
Data Insertion	INSERT INTO [DateDim] SELECT CONVERT (char(8),@CurrentDate,112) as DateKey, @CurrentDate AS Date, CONVERT (char(10),@CurrentDate,101) as FullDate, DATEPART(DD, @CurrentDate) AS DayOfMonth, DATENAME(DW, @CurrentDate) AS DayName, DATEPART(WW, @CurrentDate) + 1 - DATEPART(WW, CONVERT(VARCHAR, DATEPART(MM, @CurrentDate)) + '/1/' + CONVERT(VARCHAR,

```
DATEPART(YY, @CurrentDate))) AS WeekOfMonth,

(DATEDIFF(DD, DATEADD(QQ, DATEDIFF(QQ, 0, @CurrentDate), 0),

DATEPART(MM, @CurrentDate) AS Month,

DATENAME(MM, @CurrentDate) AS MonthName,

DATEPART(QQ, @CurrentDate) AS Quarter,


CASE DATEPART(QQ, @CurrentDate)

    WHEN 1 THEN 'First'

    WHEN 2 THEN 'Second'

    WHEN 3 THEN 'Third'

    WHEN 4 THEN 'Fourth'

END AS QuarterName,

DATEPART(YEAR, @CurrentDate) AS Year,


SET @CurrentDate = DATEADD(DD, 1, @ CurrentDate)
```

DateKey	DayOfMonth	DayName	WeekOfMonth	Month	MonthName	Quarter	QuarterName	Year
20100101	1	Friday	1	1	January	1	First	2010
20100102	2	Saturday	1	1	January	1	First	2010
20100103	3	Sunday	2	1	January	1	First	2010
20100104	4	Monday	2	1	January	1	First	2010
20100105	5	Tuesday	2	1	January	1	First	2010
20100106	6	Wednesday	2	1	January	1	First	2010
20100107	7	Thursday	2	1	January	1	First	2010
20100108	8	Friday	2	1	January	1	First	2010
20100109	9	Saturday	2	1	January	1	First	2010
20100110	10	Sunday	3	1	January	1	First	2010

d) Trigger

A trigger is essentially a form of stored procedure that reacts to a certain action made in the database. The main idea behind a trigger is that they always "trigger" once certain actions or conditions are met in the database.

Answer 1

For our first answer, we created a trigger that initiates once an order status has been updated. We have given two actions to show how the trigger can be differently executed and provide different operations.

```
go
create trigger OrderStatusUpdate on OrderFact
After Update
as
begin
set nocount on;
update OrderFact set Order_Time = GETDATE()
from OrderFact b
inner join inserted i on b.Order_ID = i.Order_ID
and i.Order_status ='Complete'
end
go
```

```sql
--%update an Order's payment time when the payment status is set to Complete

UPDATE OrderFact

SET payment_status='Complete'

WHERE Order_ID = 7;



--%update an Order's completion time when the order status is set to Complete

UPDATE OrderFact

SET Order_status='Complete'

WHERE Order_ID = 7;
```

	Order_ID	Cust_ID	Payment_ID	restaurant_ID	MenuItem_ID	DriverID	reviewID	del_addressID	delivery_time	payment_time	payment_status	Order_status	Order_amt	Order_Time	food_qty	Order_currency
1	1	1	4	4	4	2	4	1	NULL	1900-01-01 00:00:00.000			600	2022-09-06 07:39:53.420	2	THB
2	2	3	6	3	1	2	5	1	NULL	1900-01-01 00:00:00.000			200	2022-09-06 07:39:53.420	2	THB
3	3	2	1	1	2	4	2	3	NULL	1900-01-01 00:00:00.000			25	2022-09-06 07:39:53.420	1	MY
4	4	4	3	2	3	1	5	3	NULL	1900-01-01 00:00:00.000			54	2022-09-06 07:39:53.420	3	MY
5	5	2	3	5	5	4	5	4	NULL	1900-01-01 00:00:00.000			12	2022-09-06 07:39:53.420	1	MY
6	6	1	5	4	4	2	4	1	NULL	1900-01-01 00:00:00.000			300	2022-09-06 07:39:53.420	1	THB
7	7	3	6	4	4	2	4	1	NULL	2022-09-06 07:27:31.040	Complete	Complete	600	2022-09-06 07:39:53.420	2	THB
8	8	4	2	5	5	4	5	4	NULL	1900-01-01 00:00:00.000			38	2022-09-06 07:39:53.420	3	MY

Answer 2

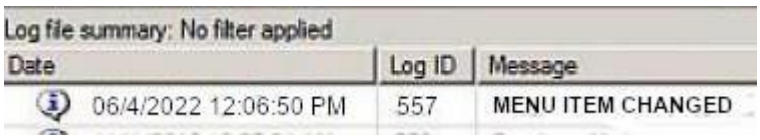
For our second answer, we created a trigger that logs any price change for a menu item that is higher than the old price currently set for the item.

```
CREATE TRIGGER log_price_increase

AFTER UPDATE ON Menu_Item

REFERENCING NEW AS N OLD AS O


FOR EACH ROW
WHEN (n.menu_item_price > o.menu_item_price)
BEGIN
    INSERT INTO menu_item_log (menu_item_id, log_date,
new_price, action)
       VALUES (:new.menu_itemno, SYSDATE, :new.price, 'NEW
PRICE');

END;
```

Answer 3

This query is a more generalised version of answer 2, whereby this trigger executes whenever the menu_item table is updated.

```
CREATE TRIGGER log_menu_update
AFTER UPDATE ON menu_item
BEGIN
    INSERT INTO menu_log (log_date, action)
        VALUES (SYSDATE, 'MENU ITEM CHANGED');
END;
```



e) Group by with grouping sets clauses

Group by grouping sets allows the user to set desired combinations of selected columns instead of all types of possibilities that are populated by group by cube.

Answer 1

In our first answer, only two grouping selection is selected:

- Payment method + Country

- Country

```
SELECT A.Order_currency as
Country,B.Payment_Method
PaymentMethod, count(A.Cust_ID)
Customer,SUM(A.Order_amt)
TotalOrderAmount

        FROM OrderFact as A

        inner join PaymentDim as
B on A.Payment_ID = B.Payment_ID

  group by cube (
[Payment_Method]
,[Order_currency])

  order by Payment_Method desc
```

	Country	PaymentMethod	Customer	TotalOrderAmount
1	MY	Cash	1	38
2	MY	Credit/Debit Card	1	25
3	MY	E-wallet	2	66
4	MY	NULL	4	129
5	THB	Cash	1	300
6	THB	Credit/Debit Card	1	600
7	THB	E-wallet	1	600
8	THB	NULL	3	1500
9	NULL	Cash	2	338
10	NULL	Credit/Debit Card	2	625
11	NULL	E-wallet	3	666

Answer 2

For our second answer, only two grouping selection is selected:

- Driver
- Country

```
SELECT A.Order_currency as
Country,B.DriverName,
Count(A.Order_amt) TotalOrder,
Sum(A.Order_amt)
TotalOrderAmount

  FROM OrderFact as A

  inner join DriverDim as B on
A.Payment_ID = B.Driver_ID

  group by grouping sets (
[Order_currency],[DriverName])

  order by DriverName desc
```

	Country	DriverName	TotalOrder	TotalOrderAmount
1	NULL	Anong	1	38
2	NULL	Ali	1	600
3	NULL	Alex	1	25
4	NULL	AHmad	2	66
5	MY	NULL	4	129
6	THB	NULL	1	600

Question 4 – Data Integration Issues

Data integration is about merging data from several heterogeneous sources. The fact that the sources are heterogenous means that there are going to be inevitable issues that can and will arise due to how the sources differ. From the simplest thing of a difference in a data type to larger differences such as in the source database design.

Entity Identification Problem

A potential issue that can arise from having to deal with heterogeneous sources is how we can 'match up' real-world entities from the data. For example, let's say we are importing data from two different databases - in one DB an entity is labelled as *student_ID* while in the other DB an entity is set as *student_num*, but both are referring to the same attribute. So, the issue here is how the data analyst who is implementing the data warehouse or the system itself is supposed to understand that these two entities are referring to the same attribute.

This issue is very common when dealing with heterogeneous sources, and the solution to the issue comes with proper schema integration.

Schema integration is used to merge two or more database schemas into a single schema that can store data from both the original databases. It aims to deal with the issue of entity identification by comparing the schemas of the different source databases and providing a single global schema that can be used to integrate the source data. A part of schema integration is to look at metadata of an attribute – attribute name, what it does in this scenario, its data type, accepted values, what rules the attribute follows? These types of questions help prevent errors in schema integration.

The possible conflicts that will be looked at to solve, including for DoorDelivery data integration, include:

- **Naming conflicts** – two schemas use different names to describe the same thing. It can also refer to a conflict whereby two schemas use the same name to describe different things.
- **Type conflicts** – a similar concept is represented differently, e.g., department is an entity type in one DB but an attribute in another.

- **Domain conflicts** – an attribute might have different domains in different schemas, e.g., weight is kg in one DB but pounds in another DB
- **Conflicts among constraints**

If we look at the assignment 1 source databases, an example of this type of issue would be *menu_item_code* from the Malaysian DB and *food_id* from the Thailand DB. Both are referring to the same attribute but are labelled differently and this is an example of a naming conflict.

One method of schema integration specifies the entities in each schema that represent the same real-world attribute so that it is much easier to integrate the source databases. Another way to perform schema integration would be modifying one specific schema to more closely match another schema, which can resolve some of the conflicts detailed earlier. So, modification is done mainly on one schema to confirm it to another.

Now no matter the exact way of performing the schema integration, arriving at one global schema that can be used for the data involves a significant level of human negotiation and involvement to settle conflicts and find the most logical and acceptable solution(s).

Redundancy

Redundancy is very common and one of the bigger issues during data integration. In short, redundant data is data that is no longer needed or even just unimportant data. Redundant data can also arise when attributes can be derived from other attributes in the dataset. Duplicate data also falls into redundancy.

An example of redundant data in the assignment DBs is the *food_type* attribute from the Thailand DB. Another example would be one database having an *age* attribute while the other dataset has the *date of birth*, making age redundant as it can be derived from the date of birth.

Dealing with redundant data can be done in several ways, such as the data analyser just knowing what data is not needed, manually checking for redundant data or by using correlation analysis.

Correlation analysis is basically analysing an attribute to look for its interdependency to other attributes, thus detecting the relationship between them. The higher the correlation between

two attributes, then one can be discarded as redundant. If the correlation is 0 then the attributes are independent and if it is negative then one attribute discourages the other, i.e., if the value of one attribute increases, then the value of the other decreases. This is all determined using the correlation analysis formula.

Data Conflict

Data conflict refers to data merged from different sources that do match in some way, which can cause an issue or conflict in the system. A very simple example would be attribute values that differ in different data sets, due to how said attributes were represented in each data set. A common attribute with this problem is a weight attribute, one attribute may be in pounds while the other in kilograms. Or an attribute related to price and how different data sets may use different currencies.

From the assignment 1 source databases, there is a data conflict with the price attributes as the ones in the Malaysian restaurant DB are using the Malaysia Ringgit currency while the Indonesian restaurant DB is using the Indonesian Rupiah currency.

Data Quality

Data quality is very important for data warehouses, as the quality of the data is directly linked to how good and effective the analysis from a data warehouse is. While the previous data integration issues detailed are a part of the quality of the data, there are specific quality criteria that the data needs to meet to be of high quality.

The dimensions of data quality typically include the following:

- **Completeness** – is the data missing anything or in an unstable state?
- **Consistency –** are values consistent across data sets?
- **Validity –** basically looking at the precision and rationality of data. Is the data formatted correctly?
- **Conformity –** are there specific formats that data values must conform to?
- **Accuracy –** do data objects accurately reflect the values that they are supposed to model from the "real world"?
- **Integrity –** is there any data missing from important relationships?

Lack of Planning and Manual Data Integration

While the previously mentioned data integration issues are all important and real, they all are looking at the source datasets themselves. Another aspect to look at with data integration is in just how the data integration is going to be performed.

This means having a plan on how the data is going to be properly integrated, a global schema created and which tools are going to be used to assist in the process. While manual data integration can be done (i.e., a data analyst manually goes through all the source data), this can take up valuable time and resources that otherwise could be saved if automated tools were used.

Question 5

We would not recommend DoorDelivery to implement a NoSQL database in the future. The basic reasoning behind this recommendation comes down to looking at what exactly a NoSQL is used for, and how the benefits of using a NoSQL database do not really work for DoorDelivery. Additionally, we looked at those benefits in isolation and found that they do not outweigh the cons of using a NoSQL database for the type of business DoorDelivery is.

First off, let's look at how and why NoSQL databases are generally used. Usually, a NoSQL database is considered when dealing with very huge volumes of data and in a very fast-paced environment where performance is a top priority. Moreover, NoSQL databases can more easily handle various structures of data, in particular unstructured data, as compared to SQL databases. And finally, NoSQL databases feature greater scalability than SQL databases such that one database can serve both transactional and analytical workloads.

Now on their own these are beneficial reasons to use a NoSQL database, but let's look at them in the context of the type of business DoorDelivery is and what they intend to use the database for.

The advantages of being able to better handle big data is not really relevant to DoorDelivery. The reality is that DoorDelivery is not anywhere near a big enough a business (with the number of restaurants it has) that it would generate truly large amounts of data. DoorDelivery will probably be dealing with gigabytes of data at its current size, which means a SQL database

will work just fine as a NoSQL database is looking at tera or even petabytes of data. For comparison, a truly global restaurant franchise company like McDonalds uses a SQL database (Amazon Redshift) and they have to handle terabytes of data.

Performance is not a top priority for DoorDelivery such that a NoSQL database would be required. DoorDelivery is not dealing with a situation such as handling a real-time flow of information or working with unstructured data that would require such high performance. And of note, it's not like SQL databases are that much "slower" or worse in performance than NoSQL databases, it is more about the context of that speed. The reality is that DoorDelivery will do fine with a SQL database and see well enough performance from a SQL database and data warehouse analysis.

On the topic of unstructured data, this advantage of NoSQL is not relevant to DoorDelivery as it will not be dealing with unstructured data. As such, a SQL database will work just fine for DoorDelivery and the type of data it will be storing and analysing.

Now NoSQL databases having greater scalability than SQL databases is definitely a relevant and clear benefit for DoorDelivery. But the question to be asked here is if scalability is such an important requirement for DoorDelivery that it alone would make using NoSQL the better choice. And the answer is that this single benefit does not make using a NoSQL database the correct choice over a SQL database.

First off, the vertical scaling of SQL databases is fine for DoorDelivery as it is not going to be dealing with very large and ever-changing data sets. Likewise, DoorDelivery will not be dealing with data sets that would actually require that much scaling in the first place. Also, while it would be good to have NoSQL databases that can function for both storage and analysis of data (while a SQL database would require separate data warehouse development), that one single benefit is not enough to warrant its use when every other potential benefit of a NoSQL database is not there for DoorDelivery.

We've looked at some of the bigger advantages to NoSQL databases and how they work in the context of DoorDelivery's business and requirements. Now, let's look at some of the

disadvantages of NoSQL databases and why a SQL database would be the better option in those situations.

When it comes to NoSQL databases, they lack the complex query functionality of SQL databases. So, while NoSQL is more flexible with being able to better store unstructured data, many of the standard queries of DBMS cannot be used. For DoorDelivery, it would be better to have access to the query functionality of SQL databases especially for the data warehouse analysis. Now this is not to say NoSQL databases cannot possibly handle these types of queries, but they would require custom code or configurations which means more development time and resources needed.

On the topic of development resources, NoSQL is also a much less mature language than SQL. NoSQL language lacks the almost endless online support, documentation and tutorials that SQL has. This means it is much easier to work with a SQL database and SQL language is incredibly efficient when it comes to querying data, manipulating and retrieving data from relational databases (all things useful for DoorDelivery). Furthermore, SQL being more common means it is easier and cheaper to find experienced SQL developers to work on the system.

NoSQL databases do not have a standard schema definition the way SQL databases do. NoSQL DBs feature dynamic schemas that are either key-value pairs, document-based, graph databases or wide-column stores depending on the requirements. This means that NoSQL databases are better suited for big data as flexibility is an important requirement which is fulfilled by their dynamic schema. This relates back to the earlier look at NoSQL databases and how they are better at working with large amounts of data. As explained before, this isn't really a relevant benefit for DoorDelivery.

Meanwhile, relational databases (SQL) have a fixed schema and use structured data. Now at first glance, it might seem that the fixed data structure of SQL may be limiting, but that fixed and standard definition makes SQL databases a better option for handling certain types of systems especially those that require multi-row transactions, e.g., payment, sales or reservation systems. Also, this fixed schema means SQL databases are more reliable as NoSQL databases sacrifice reliability for flexibility.

A final aspect to look at is in how SQL databases are much better at general data backup than NoSQL databases.

We've looked at some of the pros and cons of NoSQL/SQL databases, and how after applying the context of DoorDelivery's business a SQL database makes the most sense to use. To finish up on the reasoning on why we do not recommend a NoSQL database for DoorDelivery, let's look at just the data warehouse that DoorDelivery wants to implement and how SQL is much better suited to that task.

In a nutshell, SQL based tools for data warehousing are much better and more advanced than anything available to NoSQL today. It is simply much easier and cheaper to go with a SQL solution than it is a NoSQL, especially when looking at DoorDelivery's business and context.

NoSQL solutions are mostly about reading and writing data, and while something like MongoDB does have transformation capabilities, they pale in comparison to traditional SQL based tools and technology. Transformation and querying of data are very important for data warehouses as that is how data analysis is done. With SQL, that can be achieved and there are many tools/technologies available for that task. With NoSQL, you would need to write custom applications and tools to achieve the same results. This makes things much more complicated and would require more resources and time.

Finally, going back to the fact that DoorDelivery will be using structured data then SQL just makes the most sense to use for the data warehouse and databases.

In conclusion, while a NoSQL approach can work for DoorDelivery, it is much faster, easier, cheaper and more suitable to use a SQL database for its business and data warehouse.