



**CT069-3-3-DBS**

**DATABASE SECURITY**

**APU3F2211CS(DA)**

**HAND OUT DATE : 13 May 2023**

**HAND IN DATE : 30 July 2023, 12 Midnight**

---

**Group Number: 12**

**Group Members:**

<b>Name</b>	<b>Student Number</b>
<b>Ferdian Marcel</b>	<b>TP058072</b>
<b>Ferdinand Wilson</b>	<b>TP062635</b>
<b>Marcell Agung Wahyudi</b>	<b>TP058650</b>
<b>Michael Henry</b>	<b>TP058088</b>

## Table of Contents

Workload Matrix.....	5
1. Relational Database with Integrity and Redundancy Controls.....	6
<b>1.1 Group Work (Data Dictionary) .....</b>	<b>6</b>
<b>Table name: Equipment.....</b>	<b>6</b>
<b>Table name: Transaction .....</b>	<b>8</b>
<b>Table name: Membership .....</b>	<b>9</b>
<b>Table name: OrderItem.....</b>	<b>11</b>
<b>Table name: Category.....</b>	<b>13</b>
<b>1.2 Individual Work – Part A (10 Marks).....</b>	<b>14</b>
<b>1.2.1 Ferdian Marcel [TP058072] .....</b>	<b>14</b>
<b>1.2.2 Ferdinand Wilson [TP062635].....</b>	<b>15</b>
<b>1.2.3 Marcell Agung Wahyudi [TP058650] .....</b>	<b>17</b>
<b>1.2.3 Michael Henry [TP058088] .....</b>	<b>19</b>
<b>1.3 Individual Work – Part B (10 Marks).....</b>	<b>21</b>
<b>1.3.1 Ferdian Marcel [TP058072] .....</b>	<b>21</b>
<b>1.3.2 Ferdinand Wilson [TP062635].....</b>	<b>23</b>
<b>1.3.3 Marcell Agung Wahyudi [TP058650] .....</b>	<b>25</b>
<b>1.3.4 Michael Henry [TP058088] .....</b>	<b>28</b>
2 Database Encryption .....	33
<b>2.1 Group Work.....</b>	<b>33</b>
<b>2.1.1 Decrypted Member Detail Views.....</b>	<b>33</b>
<b>2.1.2 Hidden Member Detail Views.....</b>	<b>35</b>
<b>2.1.3 Database Encryption/Backup Encryption .....</b>	<b>36</b>
3 User Permission Management .....	38
<b>3.1 Authorization Matrix.....</b>	<b>38</b>
<b>3.2 Individual Work (10 Marks) .....</b>	<b>40</b>
<b>3.2.1 Ferdian Marcel [TP058072] .....</b>	<b>40</b>
<b>3.2.2 Ferdinand Wilson [TP062635].....</b>	<b>40</b>
<b>3.2.3 Marcell Agung Wahyudi [TP058650] .....</b>	<b>42</b>
<b>3.2.4 Michael Henry [TP058088] .....</b>	<b>44</b>
4 Database Auditing.....	45
<b>4.1.1 Login/Logout.....</b>	<b>45</b>
<b>4.1.2 Database Structural Changes.....</b>	<b>47</b>
<b>4.1.3 Data Changes .....</b>	<b>48</b>
<b>4.1.4 User Permission Changes.....</b>	<b>49</b>
5 References.....	50

## Table of Figures

Figure 1 Code snippet for Transaction and Transaction View .....	14
Figure 2 Populate Transaction Table .....	14
Figure 3 Last 20 Days Transaction Query and Output .....	14
Figure 4 Code snippet for Equipment and Equipment View .....	15
Figure 5 Code snippet for Category .....	15
Figure 6 Populate Category table .....	15
Figure 7 Populate Equipment table .....	16
Figure 8 Last 7 Days Transaction Query and Output .....	16
Figure 9 Member Table and View Creation .....	17
Figure 10 Member Decrypted Details View Creation .....	17
Figure 11 Populate Member Table .....	18
Figure 12 Last 30 Days Transaction Query .....	18
Figure 13 Last 30 Days Transaction Output .....	18
Figure 14 Order Item Table and View Creation .....	19
Figure 15 Populate Order Item Table .....	19
Figure 16 Last 15 Days Transaction Query and Output .....	20
Figure 17 User Active or Inactive Trigger .....	21
Figure 18 Examples of the Trigger .....	21
Figure 19 Example of Trigger .....	22
Figure 20 Delete Prevention Trigger .....	23
Figure 21 Testing the Delete Prevention Trigger .....	23
Figure 22 Output for the Member table after implementing the Delete Prevention Trigger .....	24
Figure 23 Return Item Trigger .....	25
Figure 24 Return Item Trigger User Testing .....	25
Figure 25 Return Item Trigger User Testing [Exceeding Return Time] .....	26
Figure 26 Transaction from User '1001' .....	26
Figure 27 Member Unable to Return [Exceed Return Time] .....	26
Figure 28 Return Item Success .....	27
Figure 29 [Equipment] Table Before Returning Item .....	27
Figure 30 [Equipment] Table After Returning Item .....	27
Figure 31 Equipment Quantity Update Trigger .....	28
Figure 32 Insert into Transaction .....	28
Figure 33 Transaction Table .....	29
Figure 34 Member Decrypted Details View .....	33
Figure 35 Member Decrypted Views Output .....	33
Figure 36 Row Level Security Implementation .....	34
Figure 37 Row Level Security User Testing .....	34
Figure 38 Member Hidden Details View .....	35
Figure 39 Granting Member Hidden Details to Management .....	35
Figure 40 Granting Member Hidden Details to Store Clerk .....	35
Figure 41 Member Hidden Details Output .....	35
Figure 42 Create a new Master Key and Certificate for the backup .....	36
Figure 43 Alter the Database to turn on the Encryption .....	36
Figure 44 Backup the database .....	37
Figure 45 Store Clerk Login, User, and Role Creation .....	40
Figure 46 Grant CONTROL and GRANT privileges .....	40
Figure 47 Give role to the user .....	40

Figure 48 Member Login, User, and Role Creation .....	40
Figure 49 Grant Permission to Member.....	41
Figure 50 Member Role add Member.....	41
Figure 51 Member successfully view Transaction table and the decrypted Member tables .....	41
Figure 52 Implementation of member update their own details with Row-Level security.....	42
Figure 53 Management Login, User, and Role Creation .....	42
Figure 54 Grant Permission to ManagementRole.....	43
Figure 55 Management Testing .....	43
Figure 56 Management Testing [Denied Access].....	43
Figure 57 Database Administrator Login, User, and Role Creation with Grant Permission to Database Administrator.....	44
Figure 58 Database Administrator User Testing .....	44
Figure 59 Database Administrator User Testing Output.....	45
Figure 60 Database Administrator User Testing .....	45
Figure 61 Database Administrator View Member Table [Denied Access] .....	45
Figure 62 Login/Logout Auditing.....	45

**Workload Matrix**

Put down the percentage contribution by each member.

<b>Group Task</b>	<b>&lt;Ferdian Marcel&gt;</b>	<b>&lt;Ferdinand Wilson&gt;</b>	<b>&lt;Marcell Agung&gt;</b>	<b>&lt;Michael Henry&gt;</b>
Relational Database with Integrity and Redundancy Controls	30%	25%	20%	25%
User Administration and Authorization Matrix	15%	35%	15%	35%
Database Auditing	15%	35%	15%	35%
Database Encryption	40%	5%	50%	5%

## 1. Relational Database with Integrity and Redundancy Controls

### 1.1 Group Work (Data Dictionary)

**Table name:** Equipment

Column Name	Data Type (Length)	Default Value	Note	Owner
Product Code	Integer (20)	-	PK (Primary Key)	<ul style="list-style-type: none"><li>• Member (View Data)</li><li>• Store Clerk (Add, View, Remove Data)</li><li>• Management (View Data)</li><li>• DBA (Control Data)</li></ul>
Equipment Name	Varchar (100)	-		<ul style="list-style-type: none"><li>• Member (View Data)</li><li>• Management (View Data)</li><li>• Store Clerk (Add, View, Remove Data)</li><li>• DBA (Control Data)</li></ul>
Price Per Unit	Float (50)	-		<ul style="list-style-type: none"><li>• Member (View Data)</li><li>• Management (View Data)</li></ul>

				<ul style="list-style-type: none"> <li>• Store Clerk (Add, View, Remove Data)</li> <li>• DBA (Control Data)</li> </ul>
Category ID	Bigint	-	FK (Foreign Key)	<ul style="list-style-type: none"> <li>• Member (View Data)</li> <li>• Management (View Data)</li> <li>• Store Clerk (Add, View, Remove Data)</li> <li>• DBA (Control Data)</li> </ul>
Quantity In Stock	Integer (20)	0		<ul style="list-style-type: none"> <li>• Member (View Data)</li> <li>• Store Clerk (Add, View, Remove Data)</li> <li>• Management (View Data)</li> <li>• DBA (Control Data)</li> </ul>
Producing Country	Varchar (100)	-		<ul style="list-style-type: none"> <li>• Member (View Data)</li> <li>• Management (View Data)</li> <li>• Store Clerk (Add, View, Remove Data)</li> </ul>

				<ul style="list-style-type: none"> <li>• DBA (Control Data)</li> </ul>
--	--	--	--	--

**Table name:** Transaction

Column Name	Data Type (Length)	Default Value	Note	Owner
Product Code	Integer (20)	-	FK (Foreign Key)	<ul style="list-style-type: none"> <li>• Store Clerks (View Data)</li> <li>• DBA (Control Data)</li> <li>• Management (View Data)</li> </ul>
Member ID	Integer (50)	-	FK (Foreign Key)	<ul style="list-style-type: none"> <li>• Store Clerks (View Data)</li> <li>• DBA (Control Data)</li> <li>• Management (View Data)</li> </ul>
Transaction Code	Integer (20)	-	PK (Primary Key)	<ul style="list-style-type: none"> <li>• Store Clerks (View Data)</li> <li>• DBA (Control Data)</li> <li>• Management (View Data)</li> </ul>
Transaction Date	Date	-		<ul style="list-style-type: none"> <li>• Store Clerks (View Data)</li> <li>• DBA (Control Data)</li> </ul>



				<ul style="list-style-type: none"> <li>• Management (View Data)</li> </ul>
Items Purchase	Varchar (50)	-		<ul style="list-style-type: none"> <li>• Store Clerks (Add, View Data)</li> <li>• DBA (Control Data)</li> <li>• Management (View Data)</li> </ul>
Quantity Purchase	Integer (10)	-		<ul style="list-style-type: none"> <li>• Store Clerks (Add, View Data)</li> <li>• DBA (Control Data)</li> <li>• Management (View Data)</li> </ul>
Total Before Discount	Decimal (7, 2)	-		<ul style="list-style-type: none"> <li>• Store Clerks (Add, View Data)</li> <li>• DBA (Control Data)</li> <li>• Management (View Data)</li> </ul>
Total After Discount	Decimal (7, 2)	-		<ul style="list-style-type: none"> <li>• Store Clerks (Add, View Data)</li> <li>• DBA (Control Data)</li> <li>• Management (View Data)</li> </ul>

**Table name:** Membership

Column Name	Data Type (Length)	Default Value	Note	Owner
Member ID	Integer (100)	-	PK (Primary Key),	<ul style="list-style-type: none"><li>• Store Clerks (Add Data)</li><li>• DBA (Control Data)</li><li>• Management (View Data)</li></ul>
National registration ID or Passport Number	Varbinary (max)	-	Confidential	<ul style="list-style-type: none"><li>• Members (Edit)</li><li>• Store Clerks (Add Data)</li></ul>
Name	Varchar (100)	-		<ul style="list-style-type: none"><li>• Members (Edit)</li><li>• Store Clerks (Add Data, Update Data)</li><li>• Management (View Data)</li><li>• DBA (Control Data)</li></ul>
Address	Varbinary (max)	-	Confidential	<ul style="list-style-type: none"><li>• Members (Edit)</li><li>• Store Clerks (Add Data, Update Data)</li></ul>

Phone Number	Varbinary (max)	-		<ul style="list-style-type: none"><li>• Members (Edit)</li><li>• Store Clerks (Add Data, Update Data)</li><li>• Management (View Data)</li><li>• DBA (Control Data)</li></ul>
Member Status (Active, Expired)	Varchar (20)	-		<ul style="list-style-type: none"><li>• Clerks (Add Data, Update Data)</li><li>• Management (View Data)</li><li>• DBA (Control Data)</li></ul>
Login ID	Varbinary (max)	-	Confidential	<ul style="list-style-type: none"><li>• Members (Edit)</li><li>• Store Clerks (Add Data)</li></ul>
Login Password	Varbinary (max)	-	Confidential	<ul style="list-style-type: none"><li>• Members (Edit)</li><li>• Store Clerks (Add Data)</li></ul>

**Table name:** OrderItem

Column Name	Data Type (Length)	Default Value	Note	Owner
-------------	-----------------------	---------------	------	-------

Product Code	Bigint	-	FK (Foreign Key)	<ul style="list-style-type: none"><li>• DBA (Control Data)</li><li>• Management (View Data)</li><li>• Store Clerk (Add, View, Remove Data)</li></ul>
Order Code	Integer	-	PK (Primary Key)	<ul style="list-style-type: none"><li>• DBA (Control Data)</li><li>• Management (View Data)</li><li>• Store Clerk (Add, View, Remove Data)</li></ul>
Order Date	Date	-		<ul style="list-style-type: none"><li>• DBA (Control Data)</li><li>• Management (View Data)</li><li>• Store Clerk (Add, View, Remove Data)</li></ul>
Quantity Purchase	Integer	-		<ul style="list-style-type: none"><li>• DBA (Control Data)</li><li>• Management (View Data)</li><li>• Store Clerk (Add, View, Remove Data)</li></ul>

**Table name:** Category

Column Name	Data Type (Length)	Default Value	Note	Owner
Category	Bigint	-	PK (Primary Key)	<ul style="list-style-type: none"><li>• DBA (Control Data)</li><li>• Management (View Data)</li><li>• Store Clerk (Add, View, Remove Data)</li></ul>
Category Name	Varchar (100)	-		<ul style="list-style-type: none"><li>• DBA (Control Data)</li><li>• Management (View Data)</li><li>• Store Clerk (Add, View, Remove Data)</li></ul>
Discount	Decimal (7, 2)	-		<ul style="list-style-type: none"><li>• DBA (Control Data)</li><li>• Management (View Data)</li><li>• Store Clerk (Add, View, Remove Data)</li></ul>

## 1.2 Individual Work – Part A (10 Marks)

### 1.2.1 Ferdian Marcel [TP058072]

#### 1.2.1.1 SQL queries that create tables of Transaction and Transaction views

The code snippet below creates tables and views as planned in the data dictionary.

```
CREATE TABLE [Transaction] (  
    ProductCode BIGINT,  
    FOREIGN KEY (ProductCode) REFERENCES Equipment(ProductCode),  
    MemberID BIGINT,  
    FOREIGN KEY (MemberID) REFERENCES Member(MemberID),  
    TransactionCode INT PRIMARY KEY,  
    TransactionDate DATE,  
    QuantityPurchase INT,  
    TotalBeforeDiscount DECIMAL(7,2) NULL,  
    TotalAfterDiscount DECIMAL(7,2) NULL  
);  
  
CREATE VIEW TransactionDetails AS  
SELECT  
    TransactionCode,  
    ProductCode,  
    MemberID,  
    TransactionDate,  
    QuantityPurchase,  
    TotalBeforeDiscount,  
    TotalAfterDiscount  
FROM [Transaction];
```

Figure 1 Code snippet for Transaction and Transaction View

#### 1.2.1.2 SQL queries that populate the table

```
INSERT INTO [Transaction] (ProductCode, MemberID, TransactionCode, TransactionDate, QuantityPurchase)  
VALUES (3001, 1001, 1, '2023-07-08', 5),  
       (3002, 1002, 2, '2023-07-09', 10),  
       (3003, 1003, 3, '2023-07-23', 15),  
       (3004, 1004, 4, '2023-07-26', 35),  
       (3005, 1005, 5, '2023-07-27', 40);
```

Figure 2 Populate Transaction Table

#### 1.2.1.3 SQL query/queries that can produce details of transactions that happen in the last n days where n = {1,2,...., 7}

User may change the @Days variable integer to change to the user needs.

```
DECLARE @Days INT;  
SET @Days = 20; --Modify last N days  
  
SELECT * FROM [Transaction]  
WHERE TransactionDate >= DATEADD(DAY, -@Days, GETDATE());
```

Figure 3 Last 20 Days Transaction Query and Output

## 1.2.2 Ferdinand Wilson [TP062635]

### 1.2.2.1 SQL queries that create tables and views

The code snippet below creates tables and views as planned in the data dictionary for Equipment.

```
--i) Tables and Views creation
CREATE TABLE [Equipment] (
    ProductCode BIGINT PRIMARY KEY,
    EquipmentName VARCHAR(100),
    PricePerUnit FLOAT(50),
    Category VARCHAR(50),
    QuantityInStock INT DEFAULT 0,
    ProducingCountry VARCHAR(100)
);

CREATE VIEW [EquipmentView] AS
SELECT
    ProductCode,
    EquipmentName,
    PricePerUnit,
    Category,
    QuantityInStock,
    ProducingCountry
FROM Equipment;
SELECT * FROM [EquipmentView]
```

Figure 4 Code snippet for Equipment and Equipment View

Next, the code snippet below is for creating the table category in the data dictionary.

```
CREATE TABLE Category (
    CategoryID BIGINT PRIMARY KEY,
    CategoryName VARCHAR(100),
    Discount DECIMAL(7,2)
);
```

Figure 5 Code snippet for Category

### 1.2.2.2 SQL Queries that populate the tables

Populate Category Table:

```
INSERT INTO [Category] (CategoryID, CategoryName, Discount)
VALUES (2001, 'Ball', 0.1),
(2002, 'Racket', 0.2),
(2003, 'Sportswear', 0.3),
(2004, 'Sneakers', 0.4),
(2005, 'Bicycle', 0.5);
```

Figure 6 Populate Category table

Populate Equipment Table:

```
INSERT INTO [Equipment] (ProductCode, EquipmentName, PricePerUnit, CategoryID, QuantityInStock, ProducingCountry)
VALUES (3001, 'Basketball Ball', 25.99, 2001, 50, 'USA'),
       (3002, 'Tennis Racket', 89.50, 2002, 30, 'China'),
       (3003, 'Cricket Bat', 75.75, 2002, 20, 'India'),
       (3004, 'Volleyball Ball', 19.99, 2001, 40, 'Brazil'),
       (3005, 'Badminton Racket', 45.25, 2002, 25, 'Malaysia');
```

Figure 7 Populate Equipment table

### 1.2.2.3 SQL query/queries that can produce details of transactions that happen in the last n days where $n = \{1, 2, \dots, 7\}$

```
DECLARE @Days INT;
SET @Days = 7; --Modify last N days

SELECT * FROM [Transaction]
WHERE TransactionDate >= DATEADD(DAY, -@Days, GETDATE());
```

Results Messages

ProductCode	MemberID	TransactionCode	TransactionDate	QuantityPurchase	TotalBeforeDiscount	TotalAfterDiscount
-------------	----------	-----------------	-----------------	------------------	---------------------	--------------------

Figure 8 Last 7 Days Transaction Query and Output

The user here can change the @Days variable integer to change to the user needs. In this case, the user set the @Days to 7 which is a week as to produce details of transaction from a week prior to this date. The output of the transaction is blank, as there is no data for transaction from 7 days prior.



### 1.2.3 Marcell Agung Wahyudi [TP058650]

#### 1.2.3.1 SQL queries that create tables and views

The code snippet below creates tables and views as planned in the data dictionary for member.

```
CREATE TABLE [Member] (  
    MemberID BIGINT PRIMARY KEY,  
    NationalIDOrPassportNumber VARCHAR(max),  
    [Name] VARCHAR(100),  
    [Address] VARCHAR(max),  
    PhoneNumber VARCHAR(20),  
    MemberStatus VARCHAR(20),  
    LoginID VARCHAR(max),  
    LoginPW VARBINARY(max)  
);  
  
CREATE VIEW MemberDetails AS  
SELECT  
    MemberID,  
    NationalIDOrPassportNumber,  
    [Name],  
    [Address],  
    PhoneNumber,  
    MemberStatus,  
    LoginID,  
    LoginPW  
FROM Member;
```

Figure 9 Member Table and View Creation

The next view is for member's decrypted details, for other roles who are denied access to view members' crucial information.

```
CREATE VIEW dbo.memberDecryptedDetails AS  
SELECT  
    MemberID,  
    CONVERT(VARCHAR(MAX), DecryptByCert(Cert_ID('Cert1'), NationalIDOrPassportNumber)) AS NationalIDOrPassportNumber,  
    [Name],  
    CONVERT(VARCHAR(MAX), DecryptByCert(Cert_ID('Cert1'), [Address])) AS [Address],  
    [PhoneNumber],  
    [MemberStatus],  
    CONVERT(VARCHAR(MAX), DecryptByCert(Cert_ID('Cert1'), LoginID)) AS LoginID,  
    CONVERT(VARCHAR(MAX), DecryptByCert(Cert_ID('Cert1'), LoginPW)) AS LoginPW  
FROM [Member];
```

Figure 10 Member Decrypted Details View Creation

#### 1.2.3.2 SQL Queries that populate the tables

Populate Member Table (with encryption):

```

INSERT INTO Member (MemberID, NationalIDOrPassportNumber, [Name], [Address], PhoneNumber, MemberStatus, LoginID, LoginPW)
VALUES
(1001, EncryptByCert(Cert_ID('Cert1'),'A123456789'), 'Madel Agung', EncryptByCert(Cert_ID('Cert1'),'123 Main St, City A'),
'069-1310', 'Active', EncryptByCert(Cert_ID('Cert1'),'macel'), EncryptByCert(Cert_ID('Cert1'),'qwerty')),
(1002, EncryptByCert(Cert_ID('Cert1'),'B987654321'), 'Michael Henry', EncryptByCert(Cert_ID('Cert1'),'456 Elm St, City B'),
'069-5678', 'Active', EncryptByCert(Cert_ID('Cert1'),'michenny'), EncryptByCert(Cert_ID('Cert1'),'qwerty')),
(1003, EncryptByCert(Cert_ID('Cert1'),'C246813579'), 'Ferdian Marcel', EncryptByCert(Cert_ID('Cert1'),'789 Oak St, City C'),
'069-9876', 'Inactive', EncryptByCert(Cert_ID('Cert1'),'wenwen'), EncryptByCert(Cert_ID('Cert1'),'qwerty')),
(1004, EncryptByCert(Cert_ID('Cert1'),'D135792468'), 'Ferdinand Wilson', EncryptByCert(Cert_ID('Cert1'),'321 Maple St, City D'),
'069-4567', 'Active', EncryptByCert(Cert_ID('Cert1'),'mingming'), EncryptByCert(Cert_ID('Cert1'),'qwerty')),
(1005, EncryptByCert(Cert_ID('Cert1'),'E864209753'), 'Celine Taydey', EncryptByCert(Cert_ID('Cert1'),'654 Birch St, City E'),
'069-0317', 'Active', EncryptByCert(Cert_ID('Cert1'),'ctaydey'), EncryptByCert(Cert_ID('Cert1'),'qwerty'));

```

Figure 11 Populate Member Table

### 1.2.3.3 SQL query/queries that can produce details of transactions that happen in the last n days where $n = \{1, 2, \dots, 7\}$

User may change the @Days variable integer to change to the user needs.

```

DECLARE @Days INT;
SET @Days = 30; --Modify last N days

SELECT * FROM [Transaction]
WHERE TransactionDate >= DATEADD(DAY, -@Days, GETDATE());

```

Figure 12 Last 30 Days Transaction Query

Output:

	ProductCode	MemberID	TransactionCode	TransactionDate	QuantityPurchase	TotalBeforeDiscount	TotalAfterDiscount
1	3003	1003	3	2023-07-23	15	NULL	NULL
2	3004	1004	4	2023-07-26	35	NULL	NULL
3	3005	1005	5	2023-07-27	40	NULL	NULL

Figure 13 Last 30 Days Transaction Output

### 1.2.3 Michael Henry [TP058088]

#### 1.2.3.1 SQL queries that create tables and views

The code snippet below creates tables and views as planned in the data dictionary.

```
CREATE TABLE OrderItem (  
    ProductCode BIGINT,  
    FOREIGN KEY (ProductCode) REFERENCES Equipment(ProductCode),  
    OrderCode INT PRIMARY KEY,  
    OrderDate DATE,  
    QuantityPurchase INT,  
);  
  
CREATE VIEW OrderDetails AS  
SELECT  
    ProductCode,  
    OrderCode,  
    OrderDate,  
    QuantityPurchase  
FROM [OrderItem];
```

Figure 14 Order Item Table and View Creation

#### 1.2.3.2 SQL Queries that populate the tables

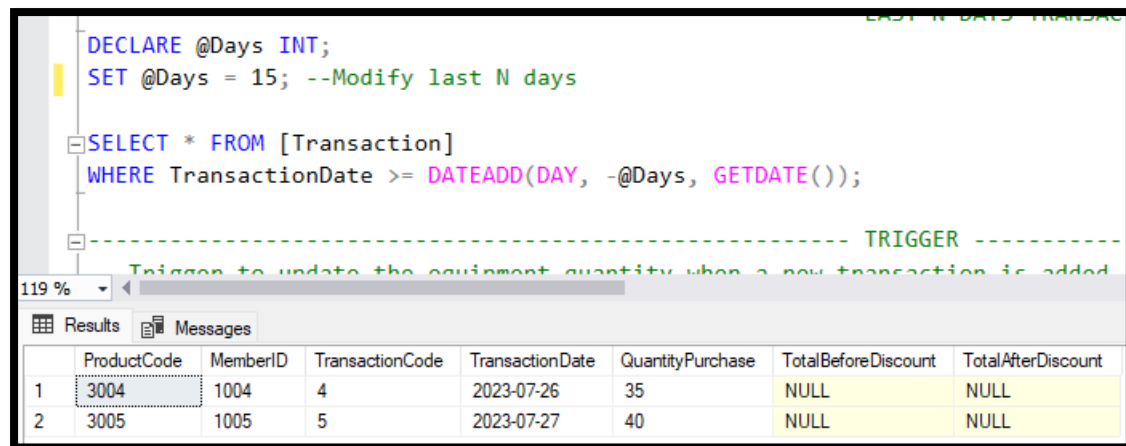
Populate OrderItem Table (with encryption):

```
INSERT INTO [OrderItem] (ProductCode, OrderCode, OrderDate, QuantityPurchase)  
VALUES (3001, 1, '2023-07-18', 5),  
       (3002, 2, '2023-07-19', 10),  
       (3003, 3, '2023-07-20', 15),  
       (3004, 4, '2023-07-21', 35),  
       (3005, 5, '2023-07-22', 40);
```

Figure 15 Populate Order Item Table

#### 1.2.3.3 SQL query/queries that can produce details of transactions that happen in the last n days where $n = \{1, 2, \dots, 7\}$

User may change the @Days variable integer to change to the user needs, in here it is 15 days.



The screenshot shows a SQL query editor with the following code:

```
DECLARE @Days INT;  
SET @Days = 15; --Modify last N days  
  
SELECT * FROM [Transaction]  
WHERE TransactionDate >= DATEADD(DAY, -@Days, GETDATE());
```

Below the query editor, there is a tab labeled "Results" which displays the output of the query. The output is a table with 8 columns: ProductCode, MemberID, TransactionCode, TransactionDate, QuantityPurchase, TotalBeforeDiscount, and TotalAfterDiscount. The table contains two rows of data.

	ProductCode	MemberID	TransactionCode	TransactionDate	QuantityPurchase	TotalBeforeDiscount	TotalAfterDiscount
1	3004	1004	4	2023-07-26	35	NULL	NULL
2	3005	1005	5	2023-07-27	40	NULL	NULL

Figure 16 Last 15 Days Transaction Query and Output

### 1.3 Individual Work – Part B (10 Marks)

#### 1.3.1 Ferdian Marcel [TP058072]

##### User Active or Inactive Trigger

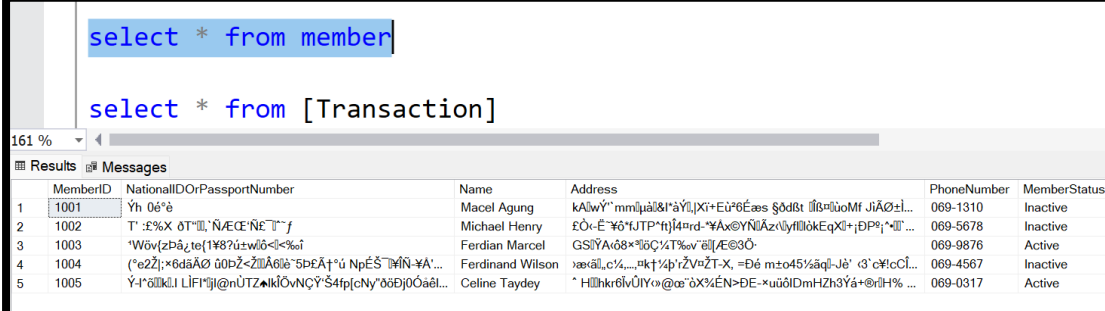
```
CREATE OR ALTER TRIGGER USER_STATUS_TRIGGER
ON [TRANSACTION]
AFTER INSERT
AS
BEGIN
    UPDATE [Member]
    SET MemberStatus = 'Inactive'
    WHERE MemberID IN (SELECT MemberID FROM inserted)
    AND DATEDIFF(month,
    (SELECT MAX(TransactionDate) FROM [TRANSACTION] WHERE MemberID = [Member].MemberID)
    , GETDATE()) > 1;

    UPDATE [Member]
    SET MemberStatus = 'Active'
    WHERE MemberID IN (SELECT MemberID FROM inserted)
    AND DATEDIFF(month,
    (SELECT MAX(TransactionDate) FROM [TRANSACTION] WHERE MemberID = [Member].MemberID)
    , GETDATE()) <= 1;
END;
```

Figure 17 User Active or Inactive Trigger

As we can see from the figure 17, it writes a query about the user active and inactive trigger. So basically, will trigger member table (MemberStatus) will change to their transaction, if they had a transaction between 1 month, it will change into Active, but if they did not do any transaction in entire month, it will change into Inactive.

For example,



	MemberID	NationalIDOrPassportNumber	Name	Address	PhoneNumber	MemberStatus
1	1001	Yh 0é*è	Macel Agung	kAlwY" mmijpâ&lt;*âYl,Xi+Eû6Êæss \$ôd8t ðß=ûoMf JiÂ0±l...	069-1310	Inactive
2	1002	T' :E%X ôT"ll' N'EC'E'NE"l"~f	Michael Henry	ÉÔ<É-¥ô*fJTP*fj)I4*rd-¥Ax@YñlÄZvÛyfllo&kEqXl+ËP*;*ll'...	069-5678	Inactive
3	1003	*Wôv(zbâ¿te{1¥87û±wô<@<%i	Ferdian Marcel	GSlY'Aô8*%ôC%4T%w'âll/E@3Ô	069-9876	Active
4	1004	(*e2Zj;*ôdâÂ0 ô0bZ<ZlilÂ0lë"5pEÄ†*û NpÉS"lHlN-¥A'...	Ferdinand Wilson	æxâll,c/%.....*k†*p'iZVvZT-X, =Ðé m±o45%âqll-Jè' 'ô' c#icCl...	069-4567	Inactive
5	1005	Y-†ôllklJ LIF†*j @nÛTZ•klôvNÇY'S4fp[cNy'ôôE)j0ôâel...	Celine Taydey	* Hllhkr6lvÜlYc@æ†ôX%EN>DE-æuôlDmH3Yá+@vIH%...	069-0317	Active

Figure 18 Examples of the Trigger

In this example, I will use number 4 which the MemberID 1004 it said the MemberStatus is Inactive because he did not do any transaction on entire months.

```

INSERT INTO [Transaction] (ProductCode, MemberID, Trans
VALUES (3001, 1004, 8, '2023-08-01', 5)

select * from member

select * from [Transaction]

```

95 %

Results Messages

	MemberID	NationalIDOrPassportNumber	Name	Address	PhoneNumber	MemberStatus
1	1001	Yh 06*è	Macel Agung	kAiwY" mmqù&I*àYl,Xi+Eù?6Éæes \$ôd8t 0B°ùoMf JiÄ0±l...	069-1310	Inactive
2	1002	T' :E%X 8T"III, NÆCE'ÑE"U"~f	Michael Henry	ÈÖ<E'¥ô"JUTP"tj]4°rd-¥Ax@YÑÜÄz<üÿfll6kEqX0+jDP9i"4II"...	069-5678	Inactive
3	1003	'WöV(zbâ¿te{1¥8?ú±wû&<I<%ö	Ferdian Marcel	GSiYAcó8×%úÇ¼T%sv'ëllÆ@3Ö.	069-9876	Active
4	1004	(°eZJl:×6daÄ0 ú0pZ<ZIIÄ6üë"5pEÄ†*ú NpÉŠ"MIÑ-¥A'...	Ferdinand Wilson	ææäü,c¼,...,°k†¼p'zV°ZT-X, =Ðé m±o45%äqil-Jé' '3' c¥lcÖl...	069-4567	Active
5	1005	Y+*8ükl.I LIF*ij)@nÜTZ•klÖvNÇY'S4fp[cNy'ôöDj0Óäêl...	Celine Taydey	* Hllhkr6lvÜIY°@ce"òX%ÉN>DE->uüöIdmHzh3Ýä+@rH4% ...	069-0317	Active

	ProductCode	MemberID	TransactionCode	TransactionDate	QuantityPurchase	TotalBeforeDiscount	TotalAfterDiscount
1	3001	1001	1	2020-07-01	5	NULL	NULL
2	3002	1002	2	2020-07-02	10	NULL	NULL
3	3003	1003	3	2020-07-23	15	NULL	NULL
4	3004	1004	4	2020-07-26	35	NULL	NULL
5	3005	1005	5	2020-07-27	40	NULL	NULL
6	3003	1005	6	2020-07-26	3	NULL	NULL
7	3003	1005	7	2023-08-01	3	NULL	NULL
8	3001	1004	8	2023-08-01	5	NULL	NULL
9	3001	1003	10	2023-08-01	3	NULL	NULL

Figure 19 Example of Trigger

As we can see that when I insert the new transaction on member 1004 on number 8 on transaction table, the status of the Member will change into Active.

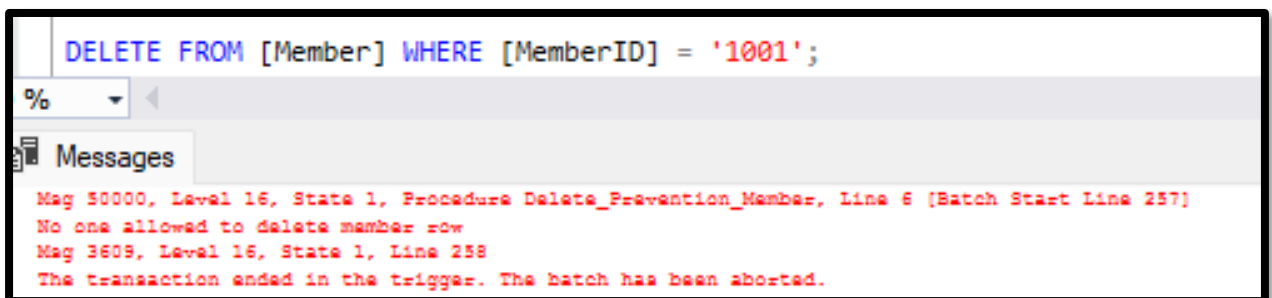
### 1.3.2 Ferdinand Wilson [TP062635]

#### Delete Prevention Trigger

```
-- Avoid accidental deletion in row from member table [Ferdinand Wilson]
--DROP Trigger [Delete_Prevention_Member]
CREATE TRIGGER [Delete_Prevention_Member]
ON [Member]
INSTEAD OF DELETE
AS
BEGIN
    RAISERROR('No one allowed to delete member row', 16,1);
    ROLLBACK;
END
```

Figure 20 Delete Prevention Trigger

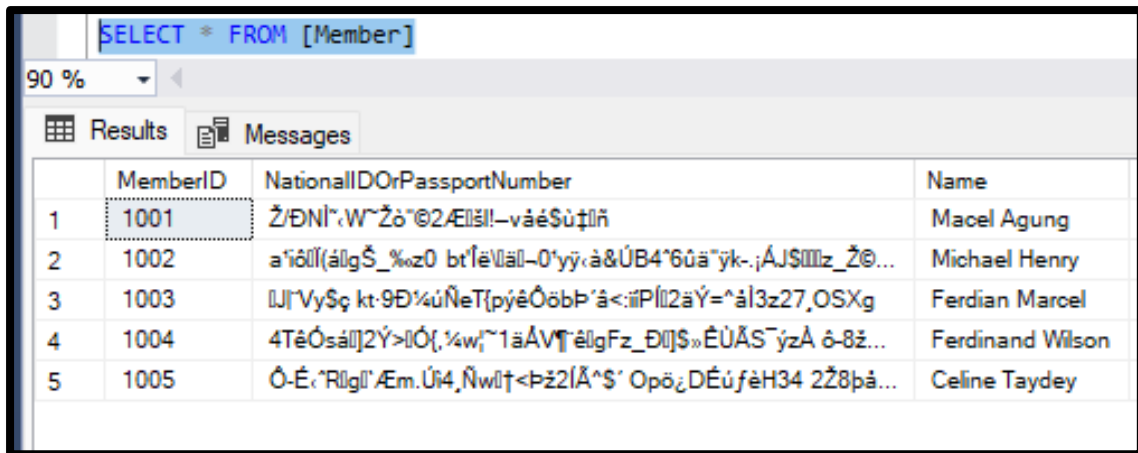
The code in the above figure is showing the Trigger function of row deletion from the member table. The Trigger is going to prevent the unauthorized accesses in deleting the row from the member table. The code that is provided is going to raise an error notification for the user when they want to delete the row from the member table. The “RAISERROR” function is used for showing the error message called “No one allowed to delete member row” when running a delete function.



The screenshot shows a SQL query window with the following text: `DELETE FROM [Member] WHERE [MemberID] = '1001';`. Below the query window, the 'Messages' pane displays the following error message:   
Msg 50000, Level 16, State 1, Procedure Delete\_Prevention\_Member, Line 6 [Batch Start Line 257]  
No one allowed to delete member row  
Msg 3609, Level 16, State 1, Line 258  
The transaction ended in the trigger. The batch has been aborted.

Figure 21 Testing the Delete Prevention Trigger

The code snippet above is used for testing the new Delete Prevention Trigger that is implemented in the system. In this case, the user here wants to delete the row which has '1001' from the member table, which is not allowed. In return, the system raises an error message showing that the user is not able to delete the table.



	MemberID	NationalIDOrPassportNumber	Name
1	1001	Ž/ĐNl'·W~Žô"©2Ællšll-váé\$ú†llñ	Macel Agung
2	1002	a'iôll(állgŠ_‰z0 bt'fēVāll-0'yŷ·à&ÚB4'6ûā"ŷk-.jÁJ\$lllz_Ž@...	Michael Henry
3	1003	llj'Vy\$ç kt-9Đ'¼úŇeT{pýéÔöbP'â<:šPll2āŸ=^āl3z27,OSXg	Ferdian Marcel
4	1004	4TéÔsáll2Ÿ>llÓ[,¼w'~1āAVŋ'élIgFz_Đll)\$»ÉÜĀS~ŷzĀ ô-8ž...	Ferdinand Wilson
5	1005	Ô-É·Rllgl'Æm.Úi4,Ňwll†<Pž2lĀ^\$' Opö¿DÉúfèH34 2Ž8pá...	Celine Taydey

Figure 22 Output for the Member table after implementing the Delete Prevention Trigger

To test the Trigger further, the select all function is used as to view that the Trigger is working fine. In a nutshell, the row which contains member id of '1001' still exists with the other details untouched and unbothered. In conclusion, the Trigger is working perfectly for the data base system of the project team.



### 1.3.3 Marcell Agung Wahyudi [TP058650]

#### Return Item Trigger

```
--Return Item Trigger [Marcell Agung W]
DROP TRIGGER tr_TransferTransaction
CREATE TRIGGER tr_TransferTransaction
ON [Transaction]
AFTER DELETE
AS
BEGIN
    DECLARE @ProductCode BIGINT, @TransactionCode INT, @QuantityPurchase INT, @TransactionDate Date;

    SELECT @ProductCode = ProductCode,
           @TransactionCode = TransactionCode,
           @QuantityPurchase = QuantityPurchase,
           @TransactionDate = TransactionDate
    FROM DELETED;

    -- Check if the [Member] is executing the trigger and the TransactionDate is 3 days from now
    IF IS_ROLEMEMBER('MemberRole') = 1 AND GETDATE() - 3 <= @TransactionDate AND @TransactionDate <= GETDATE()
    BEGIN
        -- Update the quantity in the [Equipment] table
        UPDATE [Equipment]
        SET QuantityInStock = QuantityInStock + @QuantityPurchase
        WHERE ProductCode = @ProductCode;

        -- Delete the transaction from the [Transaction] table
        DELETE FROM [Transaction] WHERE TransactionCode = @TransactionCode;
    END
    ELSE
    BEGIN
        RAISERROR('Transaction %d cannot be deleted. It has exceeded the return time OR Executing as Non-Member Role.', 16, 1, @TransactionCode);
        ROLLBACK; -- Rollback the DELETE operation to prevent deletion
    END
END;
```

Figure 23 Return Item Trigger

The code above is a return item trigger, meaning when a member deletes an item in their transaction table, the trigger will add the amount purchased back to the equipment table, but this does not always come true, there is a specific condition, which is if the executing role is 'Member' and also if the TransactionDate is three days from current date, if these two conditions are not met, an error message will be printed out saying "Transaction cannot be deleted. It has exceeded the return time OR executing as Non-Member Role." This is to ensure accidental deletion by other roles are not possible. The following is the examples of the possible outputs:

- a. Executing as Management:

```
Execute as User = 'Marcell Agung W'
SELECT * FROM [Transaction] WHERE TransactionCode = 11;
DELETE FROM [Transaction] WHERE TransactionCode = 11;
```

	ProductCode	MemberID	TransactionCode	TransactionDate	ItemsPurchase	QuantityPurchase
1	3001	1001	11	2023-08-05	Basketball Ball	11

Figure 24 Return Item Trigger User Testing

Although TransactionCode = 11 is able to be returned since it is in range for returning time (3 days from GETDATE()).

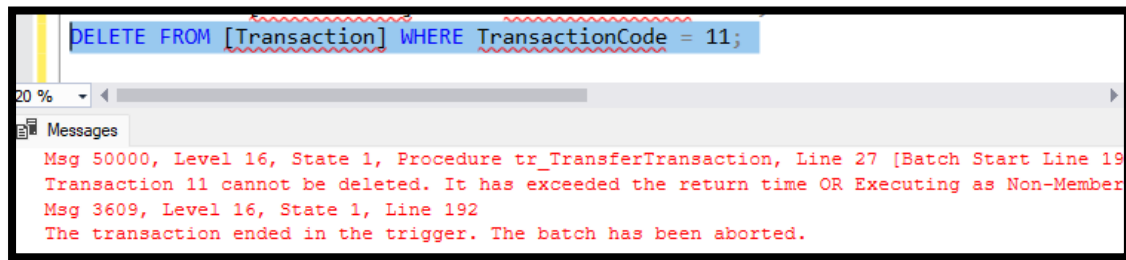


Figure 25 Return Item Trigger User Testing [Exceeding Return Time]

It is unable to be deleted since the executing user is not a member, rather a management role.

b. Executing as Member (Out of return time):

Below are the transactions by member '1001'

The screenshot shows a SQL query window with the following text:

```
--[MemberRole]
EXECUTE AS USER = '1001'
SELECT * FROM [Transaction]
```

Below the query window, the Results pane displays the following table:

	ProductCode	MemberID	TransactionCode	TransactionDate	ItemsPurchase	QuantityPurchase
1	300	1001	1	2023-08-02	Basketball Ball	1
2	3001	1001	2	2023-08-04	Basketball Ball	1
3	3001	1001	11	2023-08-05	Basketball Ball	11

Figure 26 Transaction from User '1001'

If the member were to the transaction with [TransactionCode] = 1, they are not able to because the transaction date is not between 3 days the current date(8/6/2023) at the time documented.

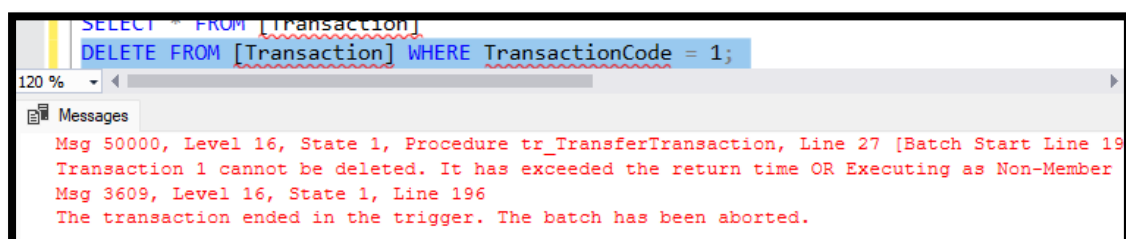


Figure 27 Member Unable to Return [Exceed Return Time]

c. Executing as Member (In Return Time):

However, if the date is in between 3 days of the TransactionDate (TransactionCode = 11):

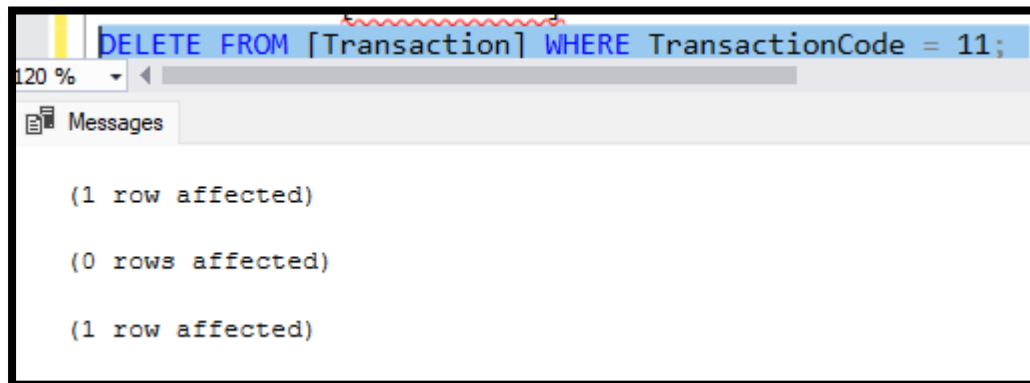


Figure 28 Return Item Success

The deletion is success.

[Equipment] Table before returning the item:

The screenshot shows a SQL query window with the command: `SELECT * FROM Equipment`. Below the query, the Results pane displays the following data:

	ProductCode	EquipmentName	PricePerUnit	Category	QuantityInStock	ProducingCountry
1	3001	Basketball Ball	25.99	Balls	73	USA
2	3002	Tennis Racket	89.5	Rackets	32	China
3	3003	Cricket Bat	75.75	Bats	20	India
4	3004	Volleyball Ball	19.99	Balls	40	Brazil
5	3005	Badminton Racket	45.25	Rackets	25	Malaysia

Figure 29 [Equipment] Table Before Returning Item

[Equipment] Table after returning the item:

The screenshot shows a SQL query window with the command: `SELECT * FROM Equipment`. Below the query, the Results pane displays the following data:

	ProductCode	EquipmentName	PricePerUnit	Category	QuantityInStock	ProducingCountry
1	3001	Basketball Ball	25.99	Balls	84	USA
2	3002	Tennis Racket	89.5	Rackets	32	China
3	3003	Cricket Bat	75.75	Bats	20	India
4	3004	Volleyball Ball	19.99	Balls	40	Brazil
5	3005	Badminton Racket	45.25	Rackets	25	Malaysia

Figure 30 [Equipment] Table After Returning Item

It is seen that the current quantity in stock is 11 in difference, just as the quantity previously purchased by the member.

### 1.3.4 Michael Henry [TP058088]

#### Equipment Quantity Update Trigger (Transaction)

```
-- Trigger to update the equipment quantity when a new transaction is added
CREATE OR ALTER TRIGGER [Equipment_Sold]
ON [Transaction]
AFTER INSERT
AS
BEGIN
    DECLARE @quantitypurchase INT, @productcode BIGINT, @quantityinstock INT
    SELECT @quantitypurchase=QuantityPurchase, @productcode=ProductCode
    FROM inserted

    SELECT @quantityinstock=QuantityInStock
    FROM Equipment
    WHERE ProductCode=@productcode

    SET @quantityinstock = (@quantityinstock - @quantitypurchase)

    UPDATE Equipment
    SET QuantityInStock = @quantityinstock
    WHERE ProductCode=@productcode
END
```

Figure 31 Equipment Quantity Update Trigger

The above SQL trigger depicts an automated procedure in which the equipment table is updated when a new transaction is added. This method is based on the assumption of a continually accessible supply of equipment. The trigger enables the deduction of equipment quantity based on the specifics of the new transaction for each incoming transaction. This dynamic approach ensures that equipment records are correct and up to date, in accordance with the changing transactional scenario. This trigger accelerates data maintenance and improves overall system performance by smoothly integrating amount revisions into the transaction process, all while retaining the assumption of continually accessible equipment quantities.

```
INSERT INTO [Transaction] (ProductCode, MemberID, TransactionCode, TransactionDate, QuantityPurchase)
VALUES (3003, 1002, 6, '2023-08-05', 3)
```

Figure 32 Insert into Transaction

	ProductCode	EquipmentName	PricePerUnit	CategoryID	QuantityInStock	ProducingCountry
1	3001	Basketball Ball	25,99	2001	50	USA
2	3002	Tennis Racket	89,5	2002	30	China
3	3003	Cricket Bat	75,75	2002	20	India
4	3004	Volleyball Ball	19,99	2001	40	Brazil
5	3005	Badminton Racket	45,25	2002	25	Malaysia

Figure 33 Transaction Table

	ProductCode	EquipmentName	PricePerUnit	CategoryID	QuantityInStock	ProducingCountry
1	3001	Basketball Ball	25,99	2001	50	USA
2	3002	Tennis Racket	89,5	2002	30	China
3	3003	Cricket Bat	75,75	2002	17	India
4	3004	Volleyball Ball	19,99	2001	40	Brazil
5	3005	Badminton Racket	45,25	2002	25	Malaysia

	ProductCode	MemberID	TransactionCode	TransactionDate	QuantityPurchase
1	3001	1001	1	2023-07-08	5
2	3002	1002	2	2023-07-09	10
3	3003	1003	3	2023-07-23	15
4	3004	1004	4	2023-07-26	35
5	3005	1005	5	2023-07-27	40
6	3003	1002	6	2023-08-05	3

As seen in the result, when a new transaction is added to the transaction table, involving the purchase of a product labelled 3003 and totalling three purchases, the quantity of the equipment table decreased from 20 to 17 units.

### Equipment Quantity Update Trigger (Item Order)

```
-- Trigger to update the equipment quantity when a new order item is added
CREATE OR ALTER TRIGGER [Equipment_Bought]
ON [OrderItem]
AFTER INSERT
AS
BEGIN
    DECLARE @quantitypurchase INT, @productcode BIGINT, @quantityinstock INT
    SELECT @quantitypurchase=QuantityPurchase, @productcode=ProductCode
    FROM inserted

    SELECT @quantityinstock=QuantityInStock
    FROM Equipment
    WHERE ProductCode=@productcode

    SET @quantityinstock = (@quantityinstock + @quantitypurchase)

    UPDATE Equipment
    SET QuantityInStock = @quantityinstock
    WHERE ProductCode=@productcode
END
```

The trigger described above, like the one mentioned previously, modifies the equipment number within the designated equipment table. This operation is triggered

when a new order is added to the order item database. Any addition of an order triggers an automatic modification in the equipment quantity, ensuring precise and synchronized records. This streamlined approach not only improves data consistency but also optimizes operational efficiency, continuing the trigger's former duty of keeping records up to date within the system.

```
INSERT INTO [OrderItem] (ProductCode, OrderCode, OrderDate, QuantityPurchase)
VALUES (3003, 6, '2023-08-05', 10)
```

	ProductCode	EquipmentName	PricePerUnit	CategoryID	QuantityInStock	ProducingCountry
1	3001	Basketball Ball	25,99	2001	50	USA
2	3002	Tennis Racket	89,5	2002	30	China
3	3003	Cricket Bat	75,75	2002	17	India
4	3004	Volleyball Ball	19,99	2001	40	Brazil
5	3005	Badminton Racket	45,25	2002	25	Malaysia

	ProductCode	EquipmentName	PricePerUnit	CategoryID	QuantityInStock	ProducingCountry
1	3001	Basketball Ball	25,99	2001	50	USA
2	3002	Tennis Racket	89,5	2002	30	China
3	3003	Cricket Bat	75,75	2002	27	India
4	3004	Volleyball Ball	19,99	2001	40	Brazil
5	3005	Badminton Racket	45,25	2002	25	Malaysia

	ProductCode	OrderCode	OrderDate	QuantityPurchase
1	3001	1	2023-07-18	5
2	3002	2	2023-07-19	10
3	3003	3	2023-07-20	15
4	3004	4	2023-07-21	35
5	3005	5	2023-07-22	40
6	3003	6	2023-08-05	10

The shown output shows that when new order data with a product code of 3003 and a quantity of 10 units is introduced into the order item database, a corresponding update is triggered in the quantity field of the equipment table to appropriately reflect the change. The initial equipment quantity with product code of 3003 in the equipment table is 17 and after the insertion it turns into 27.

## Tax and Discount Trigger

```
-- Tax and discount trigger [Michael Henry]
CREATE OR ALTER TRIGGER [Final_Price_Calculation]
ON [Transaction]
AFTER INSERT, UPDATE
AS
BEGIN
    DECLARE @transactioncode BIGINT, @quantitypurchase INT,
            @productcode BIGINT, @producingcountry VARCHAR(100),
            @priceperunit FLOAT(50), @categoryid BIGINT, @discount DECIMAL(7,2),
            @totalbeforediscount DECIMAL(7,2), @totalafterdiscount DECIMAL(7,2)
    SELECT @quantitypurchase=QuantityPurchase, @productcode=ProductCode, @transactioncode=TransactionCode
    FROM inserted

    SELECT @categoryid=CategoryID, @producingcountry=ProducingCountry, @priceperunit=PricePerUnit
    FROM Equipment
    WHERE ProductCode=@productcode

    SELECT @discount=Discount
    FROM Category
    WHERE @categoryid=CategoryID
    SET @totalbeforediscount =
        CASE
            WHEN @producingcountry != 'Malaysia' THEN (@quantitypurchase * @priceperunit) * 1.1
            ELSE @quantitypurchase * @priceperunit
        END;
    SET @totalafterdiscount = @totalbeforediscount * (1 - @discount)

    UPDATE [Transaction]
    SET TotalBeforeDiscount = @totalbeforediscount, TotalAfterDiscount = @totalafterdiscount
    WHERE TransactionCode=@transactioncode
END
```

The tax and discount trigger computes transaction pricing post-tax and post-discount with automated precision. When a new transaction is entered, the trigger begins by referencing the equipment table to gather essential details such as country and price. Concurrently, it searches the category table for discount information. Provided with this information, the trigger methodically calculates the final transaction price. The starting price is the pure transaction total, free of any tax or discount factors. Following that, the original amount is taxed and discounted, resulting in the ultimate transaction price. The trigger conducts a symphonic symphony of data from several tables, delivering correct post-processing statistics that account for both tax and discount effects. This technique exhibits the incorporation of dynamic pricing aspects into transactions, improving the financial accuracy and operational efficiency of the system.

```
INSERT INTO [Transaction] (ProductCode, MemberID, TransactionCode, TransactionDate, QuantityPurchase)
VALUES (3003, 1002, 7, '2023-08-05', 10)
```

	ProductCode	MemberID	TransactionCode	TransactionDate	QuantityPurchase	TotalBeforeDiscount	TotalAfterDiscount
1	3001	1001	1	2023-07-08	5	NULL	NULL
2	3002	1002	2	2023-07-09	10	NULL	NULL
3	3003	1003	3	2023-07-23	15	NULL	NULL
4	3004	1004	4	2023-07-26	35	NULL	NULL
5	3005	1005	5	2023-07-27	40	NULL	NULL
6	3003	1002	6	2023-08-05	3	NULL	NULL
7	3003	1002	7	2023-08-05	10	833.25	666.60

The displayed result above is an example of an automated process in which the addition of a new transaction causes automatic computations of pre-discount and post-discount totals. The sum after taxation is shown in the "Total Before Discount" column, whereas the figure after taxation and discount application is shown in the "Total After Discount" column. The discount value comes from the category table's "Discount" column, while the tax value comes from the equipment table's "Country" column. This streamlined solution harmonizes data from different sources, allowing for precise and dynamic pricing estimates that improve transactional clarity while taking discount and tax implications into account.



## 2 Database Encryption

### 2.1 Group Work

#### 2.1.1 Decrypted Member Detail Views

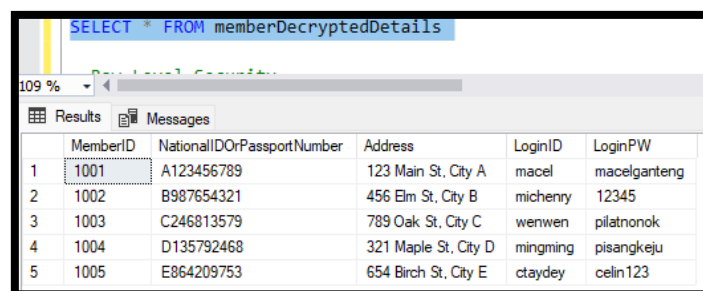
Create a view that will show member full details including automatically decrypting the encrypted values if run by a user in the Members role. This view should be accessible by Members only. It must also show only the users' own details only (implement row level security).

Firstly, a data view with the decrypted member details is created:

```
CREATE VIEW dbo.memberDecryptedDetails AS
SELECT
    MemberID,
    CONVERT(VARCHAR(MAX), DecryptByCert(Cert_ID('Cert1'), NationalIDOrPassportNumber)) AS NationalIDOrPassportNumber,
    CONVERT(VARCHAR(MAX), DecryptByCert(Cert_ID('Cert1'), [Address])) AS [Address],
    CONVERT(VARCHAR(MAX), DecryptByCert(Cert_ID('Cert1'), LoginID)) AS LoginID,
    CONVERT(VARCHAR(MAX), DecryptByCert(Cert_ID('Cert1'), LoginPW)) AS LoginPW
FROM [Member];
```

Figure 34 Member Decrypted Details View

The output are as follows:



	MemberID	NationalIDOrPassportNumber	Address	LoginID	LoginPW
1	1001	A123456789	123 Main St, City A	macel	macelganteng
2	1002	B987654321	456 Elm St, City B	michenry	12345
3	1003	C246813579	789 Oak St, City C	wenwen	pilatnonok
4	1004	D135792468	321 Maple St, City D	mingming	pisangkeju
5	1005	E864209753	654 Birch St, City E	ctaydey	celin123

Figure 35 Member Decrypted Views Output

In order to limit privilege for the members to see only their own data, row-level security is implemented, the following code implements row-level security for Member role:

```

-- 1. fn_securitypredicate function
CREATE FUNCTION Security.fn_securitypredicate
    (@MemberID AS nvarchar(100))
RETURNS TABLE
WITH SCHEMABINDING
AS
RETURN
    SELECT
        MemberID AS fn_securitypredicate_result
    FROM dbo.[Member]
    WHERE
        (IS_ROLEMEMBER('MemberRole') = 1 AND @MemberID = USER_NAME())
        OR IS_ROLEMEMBER('MemberRole') = 0; -- Return all rows for other roles

```

Figure 36 Row Level Security Implementation

The code above will return the MemberID which will be used in the later security policy as input to only show the data in which the current user is executing as, if the current user's role is not 'MemberRole', the function will not work, meaning that higher status roles, such as store clerks and management can see the full Member table.

Example, if the user '1001' executes `SELECT * FROM memberDecryptedDetails` and `SELECT * FROM [Transaction]`, the only information shown is about the MemberID with '1001':

```

EXECUTE AS USER = '1001'
SELECT * FROM memberDecryptedDetails
SELECT * FROM [Transaction]
REVERT

```

	MemberID	NationalIDOrPassportNumber	Name	Address	PhoneNumber	MemberStatus	LoginID	LoginPW
1	1001	A123456789	Macel Agung	123 Main St, City A	069-1310	Active	macel	qwerty

	ProductCode	MemberID	TransactionCode	TransactionDate	ItemsPurchase	QuantityPurchase
1	3001	1001	1	2023-07-15	Basketball Ball	2
2	3002	1001	2	2023-07-15	Tennis Racket	1

Figure 37 Row Level Security User Testing

### 2.1.2 Hidden Member Detail Views

Create a view that will show member full details and hides any encrypted values. This view can be accessed by Store Clerks and Management only.

```
CREATE VIEW memberHiddenDetails AS  
SELECT  
    MemberID,  
    [Name],  
    PhoneNumber,  
    MemberStatus  
FROM [Member];
```

Figure 38 Member Hidden Details View

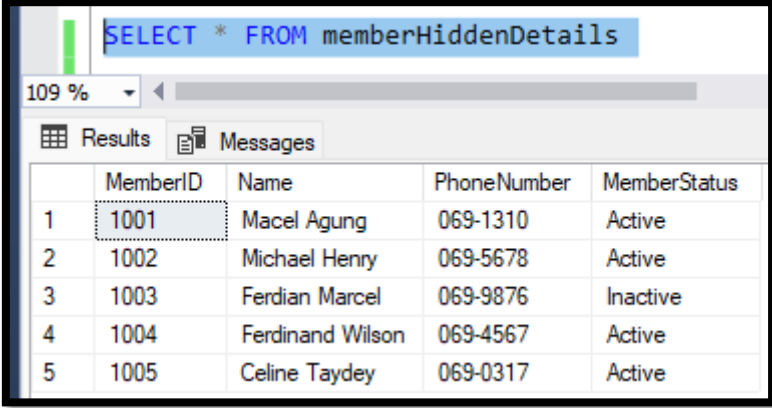
```
GRANT SELECT ON memberHiddenDetails TO [ManagementRole]
```

Figure 39 Granting Member Hidden Details to Management

```
GRANT SELECT ON [memberHiddenDetails] TO [Store Clerk]
```

Figure 40 Granting Member Hidden Details to Store Clerk

The output are as follows:



	MemberID	Name	PhoneNumber	MemberStatus
1	1001	Macel Agung	069-1310	Active
2	1002	Michael Henry	069-5678	Active
3	1003	Ferdian Marcel	069-9876	Inactive
4	1004	Ferdinand Wilson	069-4567	Active
5	1005	Celine Taydey	069-0317	Active

Figure 41 Member Hidden Details Output

The code ensures that it will only select columns that are not encrypted.

### Database Encryption/Backup Encryption

In this part, we work to do the encryption for the database backup.

```
USE master
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'qwert'
SELECT * FROM sys.symmetric_keys

USE master
CREATE CERTIFICATE CertMasterDB
WITH SUBJECT = 'CertmasterDB'
SELECT * FROM sys.certificates

USE DBS_Assignment
CREATE DATABASE ENCRYPTION KEY
    WITH ALGORITHM = AES_128
    ENCRYPTION BY SERVER CERTIFICATE CertMasterDB;
go
```

*Figure 42 Create a new Master Key and Certificate for the backup*

In this database, we use the new master key and new certificate to back up the database. The master key is protected by an Encryption by a password 'qwert' and the Certificate just a normal certificate "CertmasterDB". After that we create the encryption key with algorithm AES\_128 by the server of the certificate "CertMasterDB". After that we Alter the DB to turn on the encryption using this query below.

```
ALTER DATABASE DBS_Assignment SET ENCRYPTION ON;
```

*Figure 43 Alter the Database to turn on the Encryption*

After Creating a Master key, Certificate for the backup and alter it to turn on the Encryption, we backup the database into a disk in the laptop and we Backup the the “CertMasterDB” Certificate and the Private Key.

```
USE master
SELECT b.name AS [DB Name], a.encryption_state_desc, a.key_algorithm
FROM sys.dm_database_encryption_keys a
inner join sys.databases b on a.database_id = b.database_id
WHERE b.name = 'DBS_Assignment'

-- back up the DB --
BACKUP DATABASE DBS_Assignment
TO DISK = 'C:\Test\Backup\backup.bak'

BACKUP CERTIFICATE CertMasterDB
TO FILE = 'C:\Test\Backup\CertMasterDB.cert'
WITH PRIVATE KEY (
    FILE = 'C:\Test\Backup\CertMasterDB.key',
    ENCRYPTION BY PASSWORD = 'qwert'
);
Go

USE master; -- Switch to the master database context (you cannot
DROP DATABASE DBS_Assignment;

USE master;
RESTORE DATABASE DBS_Assignment
FROM DISK = 'C:\Test\Backup\backup.bak'
WITH MOVE 'DBS_Assignment' TO 'C:\Test\Backup\DBS_Assignment.mdf'
MOVE 'DBS_Assignment_Log' TO 'C:\Test\Backup\DBS_Assignment.ldf'
```

Figure 44 Backup the database

Before we do the restore of the backup we need to drop/delete the database first, after dropping or deleting the database, we can easily try to restore the database using the last 5-line query on the figure 44.

### 3 User Permission Management

#### 3.1 Authorization Matrix

Account	Type	Object(s)	Privilege(s)	Owner (grup kita yang nanti coding)
Member Role	SQL Role	Table: Equipment	SELECT	Ferdinand Wilson (TP062636)
		Column: Membership (National Registration ID or Passport Number, Name, Address, Phone Number, Login ID, Login Password)	UPDATE	
		Table: Transaction	SELECT	
Store Clerks	SQL Role	Table: Equipment	CONTROL	Ferdian Marcel (TP058072)
		Column: Transaction (Product Code, Member ID, Transaction Code, Transaction Date, Items Purchase, Quantity Purchase)	SELECT	
		Table: Membership	INSERT	

		Column: Membership (Name, Address, Phone Number, Member Status (Active, Expired))	UPDATE	
Database Administrators	SQL Role	Table: Transaction	CONTROL	Michael Henry
		Column: Membership (Member ID, Name, Member Status)	CONTROL	
		Table: Equipment	CONTROL	
Management	SQL Role	Database: DBS_Assignment	CONTROL, GRANT	Marcell Agung W

### 3.2 Individual Work (10 Marks)

#### 3.2.1 Ferdian Marcel [TP058072]

SQL Code for Management Role:

First, create the user.

```
CREATE LOGIN [ferdianmarcel] WITH PASSWORD = 'akuganteng'  
CREATE USER [ferdianmarcel] FOR LOGIN [ferdianmarcel]  
CREATE ROLE [Store Clerk]
```

Figure 45 Store Clerk Login, User, and Role Creation

When the Role and user has been created, grant the CONTROL and GRANT privileges.

```
GRANT CONTROL ON Equipment TO [Store Clerk]  
GRANT SELECT ON [Transaction] TO [Store Clerk]  
GRANT INSERT ON [Member] TO [Store Clerk]  
GRANT UPDATE ON [Member] (Name) TO [Store Clerk]  
GRANT UPDATE ON [Member] (Address) TO [Store Clerk]  
GRANT UPDATE ON [Member] (PhoneNumber) TO [Store Clerk]  
GRANT UPDATE ON [Member] (MemberStatus) TO [Store Clerk]  
GRANT SELECT ON [memberHiddenDetails] TO [Store Clerk]  
GRANT SELECT, INSERT, DELETE ON [OrderItem] TO [Store Clerk]  
GRANT SELECT, INSERT, DELETE ON [Category] TO [Store Clerk]
```

Figure 46 Grant CONTROL and GRANT privileges

Also, after grant the control to the store clerk, add the user to the role using Alter Role

```
ALTER ROLE [Store Clerk] ADD MEMBER [ferdianmarcel]
```

Figure 47 Give role to the user

#### 3.2.2 Ferdinand Wilson [TP062635]

```
--Member  
--DROP USER [1001]  
--DROP ROLE [MemberRole]  
CREATE LOGIN [1001] WITH PASSWORD = 'pisangkeju'  
CREATE USER [1001] FOR LOGIN [1001]  
CREATE ROLE [MemberRole]
```

Figure 48 Member Login, User, and Role Creation

To start with, the User Permission Management coding part can be done with the help of the authorization matrix that has been made previously, as to help the preparation of the coding part. Next, the login is created with the login id “1001” with the password called “pisangkeju” as to access as the user named. Following that, the role is being created for the member, called “MemberRole”.



```

GRANT SELECT ON [Equipment] TO [MemberRole]
GRANT SELECT, UPDATE ON [Member] TO [MemberRole]
DENY UPDATE ON [Member] (MemberID) TO [MemberRole]
GRANT DELETE, SELECT ON [Transaction] TO [MemberRole]
GRANT CONTROL ON CERTIFICATE::Cert1 to [MemberRole]
GRANT SELECT ON dbo.memberDecryptedDetails TO [MemberRole];

```

Figure 49 Grant Permission to Member

Furthermore, the member is going to be able to view the whole Equipment table, based on the authorization table that is made prior to this coding stage. Next, the member table can be viewed and updated by the member. Deny update is used for denying the member role from modifying the member ID in the member table. The grant deletes and select on transaction table is given for member role as well. Next, the grant control on the certificate is used for member role as a security function. Next, the members can view their own decrypted details.

```

ALTER ROLE [MemberRole] ADD MEMBER [1001]

```

Figure 50 Member Role add Member

The member '1001' is added to the member role list. In this list, there will be more than one member later.

```

--test
EXECUTE AS USER = '1001'
SELECT * FROM memberDecryptedDetails
SELECT * FROM [Transaction]
REVERT
--Member Update it's own details:

```

Results Messages

MemberID	NationalIDOrPassportNumber	Name	Address	PhoneNumber	MemberStatus	LoginID
1001	A123456789	Macel Agung	123 Main St, City A	069-1310	Active	macel

ProductCode	MemberID	TransactionCode	TransactionDate	QuantityPurchase	TotalBeforeDiscount	TotalAfterDiscoo
3001	1001	1	2023-07-08	5	NULL	NULL

Figure 51 Member successfully view Transaction table and the decrypted Member tables

The code is then tested for the user '1001' to view their own personal member details from the member table. Then, the details like the transaction(s) that is or are under the same member id, which is '1001', can also be seen.



```
--Member Update it's own details:
EXECUTE AS USER = '1001';
SELECT * FROM memberDecryptedDetails
UPDATE [Member] SET PhoneNumber = 1123
-- Update Address
UPDATE [Member] SET [Address] = EncryptByCert(Cert_ID('Cert1'),'12345')
-- Update NationalIDOrPassportNumber
UPDATE [Member] SET NationalIDOrPassportNumber = EncryptByCert(Cert_ID('Cert1'),'new_national_id')
-- Update Name
UPDATE [Member] SET [Name] = 'new_name'
-- Update LoginID
UPDATE [Member] SET LoginID = EncryptByCert(Cert_ID('Cert1'),'new_login_id')
-- Update LoginPW
UPDATE [Member] SET LoginPW = EncryptByCert(Cert_ID('Cert1'),'qwerty')
REVERT;

--SELECT dp.NAME AS principal name,
```

MemberID	NationalIDOrPassportNumber	Name	Address	PhoneNumber	MemberStatus	LoginID	LoginPW
1001	new_national_id	new_name	12345	1123	Active	new_login_id	qwerty

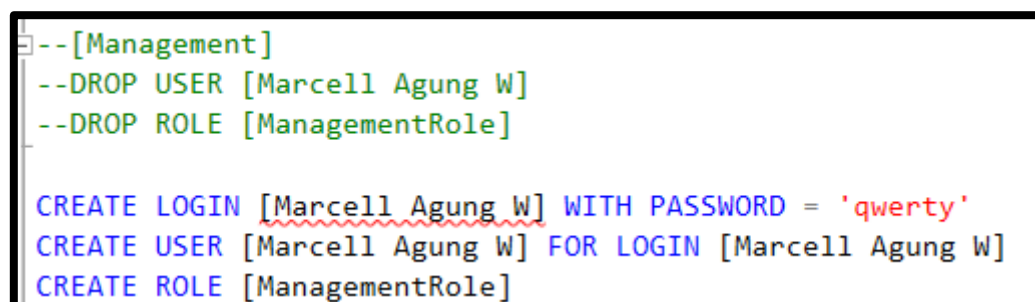
Figure 52 Implementation of member update their own details with Row-Level security

The code allows the member '1001' to update their own details. The output after the update is shown as well. The user '1001' is granted permission to update their own phone number, address, passport number or national id, name, login id, and finally their own login password.

### 3.2.3 Marcell Agung Wahyudi [TP058650]

SQL Code for Management Role:

Firstly, the user has to be created.



```
--[Management]
--DROP USER [Marcell Agung W]
--DROP ROLE [ManagementRole]

CREATE LOGIN [Marcell Agung W] WITH PASSWORD = 'qwerty'
CREATE USER [Marcell Agung W] FOR LOGIN [Marcell Agung W]
CREATE ROLE [ManagementRole]
```

Figure 53 Management Login, User, and Role Creation

Then, next the [Management] role is created and granted the CONTROL and GRANT privileges, also the user is added to the role.

```

GRANT CONTROL ON SCHEMA::dbo TO [ManagementRole] WITH GRANT OPTION;
DENY CONTROL ON [Member] TO [ManagementRole]
DENY SELECT ON [memberDecryptedDetails] TO [ManagementRole]
GRANT SELECT ON memberHiddenDetails TO [ManagementRole]
ALTER ROLE [ManagementRole] ADD MEMBER [Marcell Agung W]

```

Figure 54 Grant Permission to ManagementRole

Finally, to test the role:

```

--Testing
Execute as User = 'Marcell Agung W'
SELECT * FROM [memberDecryptedDetails] --Denied Access (Testing)
SELECT * FROM memberHiddenDetails ORDER BY MemberID
SELECT * FROM [TransactionDetails] Order By MemberID
Revert

```

119 %

Results Messages

	MemberID	Name	PhoneNumber	MemberStatus
1	1001	new_name	5555	Active
2	1002	Michael Henry	069-5678	Active
3	1003	Ferdian Marcel	069-9876	Inactive
4	1004	Ferdinand Wilson	069-4567	Active
5	1005	Celine Taydey	069-0317	Active

	TransactionCode	ProductCode	MemberID	TransactionDate	QuantityPurchase	TotalBeforeDiscount	TotalAfterDiscount
1	1	3001	1001	2023-07-08	5	NULL	NULL
2	2	3002	1002	2023-07-09	10	NULL	NULL
3	3	3003	1003	2023-07-23	15	NULL	NULL
4	4	3004	1004	2023-07-26	35	NULL	NULL
5	5	3005	1005	2023-07-27	40	NULL	NULL

Figure 55 Management Testing

```

--Testing
Execute as User = 'Marcell Agung W'
SELECT * FROM [memberDecryptedDetails] --Denied Access (Testing)
SELECT * FROM memberHiddenDetails ORDER BY MemberID
SELECT * FROM [Transaction] Order By MemberID
Revert

```

19 %

Messages

Msg 229, Level 14, State 5, Line 424  
The SELECT permission was denied on the object 'memberDecryptedDetails', database 'DBS\_Assignment', schema 'dbo'.

Figure 56 Management Testing [Denied Access]

As seen from the output, management is able to read member hidden details and transaction data view but unable to read to encrypted values.

### 3.2.4 Michael Henry [TP058088]

```
-- [Database Administrator]
--DROP USER [Michael Henry]
--DROP ROLE [Database Administrator]
CREATE LOGIN [Michael Henry] WITH PASSWORD = 'qwerty'
CREATE USER [Michael Henry] FOR LOGIN [Michael Henry]
CREATE ROLE [Database Administrator]
GRANT CONTROL ON [Equipment] TO [Database Administrator]
GRANT CONTROL ON [Transaction] TO [Database Administrator]
GRANT CONTROL ON [Member] TO [Database Administrator]
GRANT CONTROL ON [OrderItem] TO [Database Administrator]
GRANT CONTROL ON [Category] TO [Database Administrator]
DENY SELECT ON [Member] (NationalIDOrPassportNumber) TO [Database Administrator]
DENY SELECT ON [Member] (Address) TO [Database Administrator]
DENY SELECT ON [Member] (LoginID) TO [Database Administrator]
ALTER ROLE [Database Administrator] ADD MEMBER [Michael Henry]
```

Figure 57 Database Administrator Login, User, and Role Creation with Grant Permission to Database Administrator

The authorisation of the database administrator is closely linked to a pre-established authorization matrix. The scope of their privileges is defined by this matrix. The administrator, in particular, has the authority to perform Data Definition Language (DDL) queries across all databases. There is, however, one important caveat: access to confidential member data is prohibited. A series of procedures is taken to implement this authorisation setup. First, the administrator's login credentials, and user profile are built, and then a particular role is created. Following that, a granular granting process occurs, allocating control privileges within the scope of the role. Furthermore, customized select rights are methodically provided to certain columns. This stratified method guarantees that only pertinent information is available. Finally, the administrator is assigned to the appropriate role, completing the framework of their approved skills. This rigorous procedure ensures that permissions are not only aligned with the matrix, but also that data security and operational integrity are prioritized.

```
Execute as User = 'Michael Henry'
ALTER TABLE [Member] ADD [test] INT
SELECT MemberID, test FROM [Member]
Revert
```

Figure 58 Database Administrator User Testing

	MemberID	test
1	1001	NULL
2	1002	NULL
3	1003	NULL
4	1004	NULL
5	1005	NULL

Figure 59 Database Administrator User Testing Output

```
Execute as User = 'Michael Henry'
SELECT * FROM [Member]
Revert
```

Figure 60 Database Administrator User Testing

```
Msg 230, Level 14, State 1, Line 410
The SELECT permission was denied on the column 'NationalIDOrPassportNumber' of the object 'Member', database 'DBS_Assignment', schema 'dbo'.
Msg 230, Level 14, State 1, Line 410
The SELECT permission was denied on the column 'Address' of the object 'Member', database 'DBS_Assignment', schema 'dbo'.
Msg 230, Level 14, State 1, Line 410
The SELECT permission was denied on the column 'LoginID' of the object 'Member', database 'DBS_Assignment', schema 'dbo'.
Completion time: 2023-08-09T11:35:20.1321058+08:00
```

Figure 61 Database Adminsitrator View Member Table [Denied Access]

As shown in the output above, DBA role able to alter the member table but they cannot see the member confidential details. If they are trying to access the member table, the SQL will throw a permission denied message.

## 4 Database Auditing

### 4.1.1 Login/Logout

```
----- Login and Logout
-- ALTER SERVER AUDIT Login_Logout_Audit WITH (STATE = OFF)
-- DROP SERVER AUDIT Login_Logout_Audit
USE master
CREATE SERVER AUDIT Login_Logout_Audit TO FILE ( FILEPATH = 'D:\test' );
ALTER SERVER AUDIT Login_Logout_Audit WITH (STATE = ON) ;

-- ALTER SERVER AUDIT SPECIFICATION Login_Logout_Audit_Spec WITH (STATE = OFF)
-- DROP SERVER AUDIT SPECIFICATION Login_Logout_Audit_Spec
USE DBS_Assignment
CREATE SERVER AUDIT SPECIFICATION [Login_Logout_Audit_Spec]
FOR SERVER AUDIT [Login_Logout_Audit]
ADD (LOGOUT_GROUP),
ADD (SUCCESSFUL_LOGIN_GROUP),
ADD (FAILED_LOGIN_GROUP)
WITH (STATE = ON);
```

Figure 62 Login/Logout Auditing

The provided code snippet demonstrates how to set up an audit system to track both login and logout events in a server environment. This feature improves the system's security and accountability by meticulously monitoring user activities. Let's get into the specifics of this code:

The first step in starting this audit trail is to create a server audit called "Login\_Logout\_Audit." This audit is meticulously connected to a specific file path on the computer's drive. This link to a file system location guarantees that all relevant audit data is persistently maintained, allowing for further analysis and review.

Following the creation of the server audit, the establishment of a server audit specification is the next critical step. This specification outlines the audit's precise scope and characteristics. This standard delineates three distinct groups of events:

1. **LOGOUT\_GROUP:** This category contains events associated with user logouts. It includes a detailed record of logout operations, revealing when and by whom logout actions are initiated. This is critical for tracking user sessions and identifying illegal access attempts.
2. **SUCCESSFUL\_LOGIN\_GROUP:** This section is about successful login attempts. It meticulously records occasions in which users get permitted access to the system. This information is critical for maintaining accountability and monitoring user interactions to ensure that security standards are followed.
3. **FAILED\_LOGIN\_GROUP:** The third group is for unsuccessful login attempts. It logs incidents in which users' attempts to access the system are blocked due to invalid credentials or other circumstances. The extensive logging of failed login attempts can act as an early warning system for future security breaches, allowing administrators to take preventative measures to protect the system.

```
-- Read
DECLARE @AuditFilePath VARCHAR(8000);
Select @AuditFilePath = audit_file_path
From sys.dm_server_audit_status
where name = 'Data_Changes_Audit'

select action_id, event_time, database_name, database_principal_name, object_name, statement
from sys.fn_get_audit_file(@AuditFilePath,default,default)
where database_name = 'DBS_Assignment'
```

Following that, a critical procedure happens in order to access and analyse the audit data's contents. This includes declaring a variable to hold the exact file location linked with the audit records. Following that, there is a purposeful curation of specific

bits of information, with the emphasis on carefully selecting significant data points from the audit records. Once the file path is obtained, the following step is to carefully extract essential data pieces from the audit records. This procedure is distinguished by the careful selection of material that is relevant in the context of the audit's aim. Rather to overburdening the study with irrelevant details, the emphasis is squarely on recognizing and extracting significant insights.

	event_time	action_id	succeeded	server_principal_name	statement
1	2023-08-10 06:11:23.6401137	AUSC	1	LAPTOP-T5QFH3CM\User	
2	2023-08-10 06:12:05.7384868	LGIS	1	1001	-- network protocol: LPC set quoted_identifier ...
3	2023-08-10 06:12:05.8880300	LGIS	1	1001	-- network protocol: LPC set quoted_identifier ...
4	2023-08-10 06:12:05.9853780	LGIS	1	1001	-- network protocol: LPC set quoted_identifier ...
5	2023-08-10 06:12:05.9920252	LGO	1	1001	
6	2023-08-10 06:12:05.9930256	LGIS	1	1001	-- network protocol: LPC set quoted_identifier ...
7	2023-08-10 06:12:05.9961155	LGO	1	1001	
8	2023-08-10 06:12:05.9961155	LGO	1	1001	
9	2023-08-10 06:12:06.0040716	LGIS	1	1001	-- network protocol: LPC set quoted_identifier ...
10	2023-08-10 06:12:06.0292348	LGO	1	1001	

The output shows that the login and logout records were successfully retrieved. "1001" identifies a member, with "LGIS" indicating successful login and "LGO" signaling logout events.

#### 4.1.2 Database Structural Changes

```

----- Database Structural Changes
-- ALTER SERVER AUDIT Structural_Changes_Audit WITH (STATE = OFF)
-- DROP SERVER AUDIT Structural_Changes_Audit
USE master
CREATE SERVER AUDIT Structural_Changes_Audit TO FILE ( FILEPATH = 'D:\test' );
ALTER SERVER AUDIT Structural_Changes_Audit WITH (STATE = ON) ;

-- ALTER DATABASE AUDIT SPECIFICATION Structural_Changes_Audit_Spec WITH (STATE = OFF)
-- DROP DATABASE AUDIT SPECIFICATION Structural_Changes_Audit_Spec
USE DBS_Assignment
CREATE DATABASE AUDIT SPECIFICATION [Structural_Changes_Audit_Spec]
FOR SERVER AUDIT [Structural_Changes_Audit]
ADD (DATABASE_OBJECT_CHANGE_GROUP)
WITH (STATE = ON) ;

```

Same as before, the audit procedure begins with the addition of a server audit. Unlike previous cases, a database audit specification is used over a server specification audit. This customized specification includes the DATABASE\_OBJECT\_CHANGE\_GROUP, indicating an intentional focus on monitoring changes within database objects. By selecting this group, the audit becomes more sensitive to changes in database components. This sophisticated approach emphasizes the commitment to gathering exact and relevant data, boosting the system's security and integrity by tracking alterations with greater accuracy within the authorized database.



```
-- Read
DECLARE @AuditFilePath VARCHAR(8000);
Select @AuditFilePath = audit_file_path
From sys.dm_server_audit_status
where name = 'Structural_Changes_Audit'

select action_id, event_time, database_name, database_principal_name, object_name, statement
from sys.fn_get_audit_file(@AuditFilePath,default,default)
Where database_name = 'DBS_Assignment'
```

	action_id	event_time	database_name	database_principal_name	object_name	statement
1	AL	2023-08-10 06:18:02.3022335	DBS_Assignment	dbo	dbo	CREATE TABLE OrderItem1 ( ProductCode BIGINT, ...
2	CR	2023-08-10 07:44:57.9920578	DBS_Assignment	dbo	Security	CREATE SCHEMA Security;
3	AL	2023-08-10 07:45:07.1726183	DBS_Assignment	dbo	Security	CREATE FUNCTION Security.fn_securitypredicate (...
4	AL	2023-08-10 07:45:10.1201923	DBS_Assignment	dbo	dbo	CREATE SECURITY POLICY [SecurityPolicy_Member] ...
5	AL	2023-08-10 07:45:10.1242003	DBS_Assignment	dbo	dbo	CREATE SECURITY POLICY [SecurityPolicy_Transacti...

Same as before, to read the audit, a variable to hold the file path is created. Then, some selection of information is chosen from the audit file. Furthermore, as seen in the output, all database structure such as create and alter is recorded in the audit file.

#### 4.1.3 Data Changes

```
----- Data Changes Audit
-- ALTER SERVER AUDIT Data_Changes_Audit WITH (STATE = OFF)
-- DROP SERVER AUDIT Data_Changes_Audit
USE master
CREATE SERVER AUDIT Data_Changes_Audit TO FILE ( FILEPATH = 'D:\test' );
ALTER SERVER AUDIT Data_Changes_Audit WITH (STATE = ON) ;

-- ALTER DATABASE AUDIT SPECIFICATION Data_Changes_Audit_Spec WITH (STATE = OFF)
-- DROP DATABASE AUDIT SPECIFICATION Data_Changes_Audit_Spec
USE DBS_Assignment
CREATE DATABASE AUDIT SPECIFICATION Data_Changes_Audit_Spec
FOR SERVER AUDIT Data_Changes_Audit
ADD ( INSERT , UPDATE, DELETE, SELECT
ON DATABASE::DBS_Assignment BY PUBLIC)
WITH (STATE = ON) ;
```

The procedure used for implementing the data changes audit is similar to that of other database audits. Nonetheless, a distinguishing feature develops in the shape of audit specifications. This specification, designed for the data changes audit, cover a wide range of activities, including INSERT, UPDATE, DELETE, and SELECT. This specification covers the entire range of data manipulation, monitoring the insertion, alteration, removal, and retrieval of information from a database. The audit, by embracing these critical functions, carefully monitors every aspect of data dynamics, enhancing security, compliance, and accountability. This complex approach indicates a dedication to thorough control, preserving database integrity and protecting against illegal changes.



```
-- Read
DECLARE @AuditFilePath VARCHAR(8000);
Select @AuditFilePath = audit_file_path
From sys.dm_server_audit_status
where name = 'Data_Changes_Audit'

select action_id, event_time, database_name, database_principal_name, object_name, statement
from sys.fn_get_audit_file(@AuditFilePath,default,default)
where database_name = 'DBS_Assignment'
```

	action_id	event_time	database_name	database_principal_name	object_name	statement
1	SL	2023-08-10 06:05:54.8043285	DBS_Assignment	dbo	trigger_events	SELECT tr.name AS [Name], tr.object_id AS [ID], ...
2	SL	2023-08-10 06:05:54.8053296	DBS_Assignment	dbo	system_sql_modules	SELECT tr.name AS [Name], tr.object_id AS [ID], ...
3	SL	2023-08-10 06:05:54.8063417	DBS_Assignment	dbo	system_objects	SELECT tr.name AS [Name], tr.object_id AS [ID], ...
4	SL	2023-08-10 06:05:54.8063417	DBS_Assignment	dbo	sql_modules	SELECT tr.name AS [Name], tr.object_id AS [ID], ...
5	SL	2023-08-10 06:05:54.8063417	DBS_Assignment	dbo	assembly_modules	SELECT tr.name AS [Name], tr.object_id AS [ID], ...
6	SL	2023-08-10 06:05:54.8072953	DBS_Assignment	dbo	objects	SELECT tr.name AS [Name], tr.object_id AS [ID], ...
7	SL	2023-08-10 06:05:54.8072953	DBS_Assignment	dbo	tables	SELECT tr.name AS [Name], tr.object_id AS [ID], ...
8	SL	2023-08-10 06:05:55.0232947	DBS_Assignment	dbo	remote_data_archi...	INSERT INTO #tmp_extended_remote_data_arc...
9	SL	2023-08-10 06:05:55.0232947	DBS_Assignment	dbo	tables	INSERT INTO #tmp_extended_remote_data_arc...
10	SL	2023-08-10 06:05:59.9655920	DBS_Assignment	dbo	types	SELECT tbl.name AS [Name], tbl.object_id AS [ID]...

The procedure, as previously stated, remains unchanged. A variable is set up to store the file path in order to access the audit information. Following that, a purposeful extraction of certain data from the audit file occurs. Moreover, as seen in the output, some activities such as select, and insert is recorded in the audit file.

#### 4.1.4 User Permission Changes

```
----- User Permission Changes
-- ALTER SERVER AUDIT User_Permission_Audit WITH (STATE = OFF)
-- DROP SERVER AUDIT User_Permission_Audit
USE master
CREATE SERVER AUDIT User_Permission_Audit TO FILE ( FILEPATH = 'D:\test' );
ALTER SERVER AUDIT User_Permission_Audit WITH (STATE = ON) ;

-- ALTER DATABASE AUDIT SPECIFICATION User_Permission_Audit_Spec WITH (STATE = OFF)
-- DROP DATABASE AUDIT SPECIFICATION User_Permission_Audit_Spec
USE DBS_Assignment
CREATE DATABASE AUDIT SPECIFICATION [User_Permission_Audit_Spec]
FOR SERVER AUDIT [User_Permission_Audit]
ADD (SCHEMA_OBJECT_PERMISSION_CHANGE_GROUP)
WITH (STATE = ON) ;
```

The audit specification is adjusted to include the SCHEMA\_OBJECT\_PERMISSION\_CHANGES\_GROUP when auditing changes to user permissions. This option makes it easier to keep track of changes to schema-level object permissions. Notably, the selection for constructing the audit mechanism, including the formation of the server audit and the definition of the file path, is same across all three audits outlined before. This consistent approach emphasizes the methodical character of the auditing process, ensuring that each audit, while unique in

focus, follows to a defined framework that simplifies implementation and promotes cohesive audit record management.

```
-- Read
DECLARE @AuditFilePath VARCHAR(8000);
Select @AuditFilePath = audit_file_path
From sys.dm_server_audit_status
where name = 'User_Permission_Audit'

select event_time, database_name, database_principal_name, object_name, statement
from sys.fn_get_audit_file(@AuditFilePath,default,default)
where database_name = 'DBS_Assignment'
```

	event_time	database_name	database_principal_name	object_name	statement
1	2023-08-10 06:15:39.9548101	DBS_Assignment	dbo	Equipment	GRANT CONTROL ON [Equipment] TO [Database Adminis...
2	2023-08-10 06:15:39.9558698	DBS_Assignment	dbo	Transaction	GRANT CONTROL ON [Transaction] TO [Database Admini...
3	2023-08-10 06:15:39.9578528	DBS_Assignment	dbo	Member	GRANT CONTROL ON [Member] TO [Database Administra...
4	2023-08-10 06:15:39.9588575	DBS_Assignment	dbo	OrderItem	GRANT CONTROL ON [OrderItem] TO [Database Administ...
5	2023-08-10 06:15:39.9598592	DBS_Assignment	dbo	Category	GRANT CONTROL ON [Category] TO [Database Administr...
6	2023-08-10 06:15:39.9608219	DBS_Assignment	dbo	Member	DENY SELECT ON [Member] (NationalIDOrPassportNum...
7	2023-08-10 06:15:39.9618182	DBS_Assignment	dbo	Member	DENY SELECT ON [Member] (Address) TO [Database Ad...
8	2023-08-10 06:15:39.9634205	DBS_Assignment	dbo	Member	DENY SELECT ON [Member] (LoginID) TO [Database Ad...
9	2023-08-10 06:15:54.9495118	DBS_Assignment	dbo	Equipment	GRANT CONTROL ON [Equipment] TO [Database Adminis...
10	2023-08-10 06:15:54.9505116	DBS_Assignment	dbo	Transaction	GRANT CONTROL ON [Transaction] TO [Database Admini...

As seen from the output above, some action such as granting and denying permission to user is recorded in the audit file.

## 5 References

Backup Location: C:\College Assignment\BackupDatabase