Hybrid Database System

Implementation of a Relational and an XML Database for Overdrive Car Dealers

Student:

Vinit Sawant (10527412)

Kumuditha Athukorala (10539070)

Srikanth Shilesh Pasam (10387794)

Teacher:

Dr. Shazia A Afzal

Course:

Advanced Databases (B9IS100)

Table of Contents

Index of Figures

Introduction

The Hybrid Database System report explains the working of a Database system for Overdrive car dealership. The Overdrive car dealership is a medium scale business that procures cars from various manufacturers based on customer orders. It has business partnerships with five different car manufacturers currently and plans on getting more on board. The company has a clear and structured hierarchy with various departments and levels within each. There is also an incentive system in place to motivate the sales associate's performance. Whenever a customer places an order for a car, the sales associate raises an order request within the system. This request gets forwarded to the concerned department, who then places an order for the same with the manufacturer. Once the order is completed successfully, the sales associate responsible for that sale will get an incentive. All these business functionalities require a centralized database management system in order to function efficiently.

Database

A database is a collection of information that is organized. This data can be accessed, managed, and updated as per the dealership requirements. The data is indexed, which makes it easier to find relevant information through queries (Rouse, no date).

Database Management System

The data in the database is managed by a Database Management System (DBMS). A DBMS serves as an interface between the database and the Overdrive car dealership, allowing them to create, manipulate, and store the data. DBMS is a software that facilitates oversight and control of databases. A few of the most popular DBMS available are MySQL, Microsoft SQL Server, Microsoft Access, Oracle Database, and dBASE (*What is a database?* no date).

Microsoft SQL

Overdrive uses Microsoft SQL Server to create a Hybrid database. The SQL Server is a relational database system from Microsoft. Apart from storing data, it also comprises of management system. There are numerous business applications of Microsoft SQL. Overdrive uses MS SQL Server to hold sensitive user information like personal details, purchase logs, manufacture details, and other confidential information. The system also allows for sharing data within the dealership with increased reliability. The SQL server is used to increase the speed with which data is processed,

allowing large operations to be performed with ease (*What is Microsoft SQL Server and What is it Used For?*, 2017).

Hybrid Database

The SQL Server used by Overdrive stores the data in the form of a Hybrid database. The database consists of Relational and XML data types.

Relational Database

A Relational database stores data in the form of tables. The database provides access to data points that are related to one another. Each row in the table is a record with a unique ID called the 'Key.' The columns hold the data attributes. The relational databases depict the logical data structure, which is different from the physical storage structure. Relational databases prioritize data accessibility and accuracy. This is accomplished in the form of data integrity rules. An example of this is where the relational database does not allow duplicate rows to be entered in the table. This helps prevent erroneous information from being stored in the database. The major application of Relational Database is in the place where data points relating to each other must be managed in a secure, rules-based, and consistent way (*What is a relational database?*, no date).

Overdrive chose Relation Database as one of its Hybrid database system components because of the following criteria:

- Data Accuracy Requirements – The business operates with data related to customers, car specifications, inventory management, and salaries of employees. All this requires accuracy in data management and avoid any kind of duplication.
- Scalability – Relational databases have good scalability in terms of anticipated growth and ability to produce mirrored database copies. These factors enable natural expansion of the Overdrive business in the future if needed.
- Concurrency – Overdrive dealers have multiple departments within the organization ranging from procurement, sales, finance, and HR. All these departments need to have access to the database simultaneously. This centralized management of the database allows for the smooth and efficient functioning of the business. Sales agents can place customer order requests, the procurement department can order for the required vehicles from the manufacturer, HR can recruit new sales agents, and the finance team handles the salaries of the employees along with their incentives based on the employee's individual sales record. Relational databases make this concurrent access and management of the database possible.

- Performance and Reliability Needs – The reliable performance of a system is a basic functionality expected out of any system. Overdrive dealers are highly dependent on the database management system for almost all business operations. In this regard, the relational database query response performance plays a crucial role.

XML

XML database is used for storing large volumes of data. It is a secure and persistent software system. Data in XML is queried using XQuery. XML's functionality is highlighted in situations where the data requirements structure varies within the same element.

Overdrive uses the Native XML database as a part of its hybrid database. This type of database is based on the container format. This form of hybrid database allows for specific values of XML data to be stored in relational columns while the rest can be stored in an XML column. The hybrid database yields better performance as we have better control over the indexes created in the relational columns and locking its characteristics.

Scope of the System

1. Overdrive car dealership employs a Hybrid Database of Relational and XML within an MS SQL Server database management system.
2. The database consists of 8 tables. Each table consists of various range of attributes like IDs of manufacturers, customers, employees, orders, manufacture details, customer details, order dates, names, and addresses, to name a few.
3. The car dealer stores the data about the manufacturers along with the models available in the manufacture and car model tables, respectively. Based on this information, current market trends, and estimations, certain cars are procured and stationed within the warehouse in advance.
4. The order details are tracked using the manufacture order table.
5. The car variant ordered from the manufacturer uses XML data type to enable flexibility in the details of the variant stored. As the parameters of the variant details can vary depending on the type of the car, using XML here has more relevance.
6. The list of procured cars is maintained in the inventory table, which highlights all the cars available and sold by Overdrive. While this shows the backend operation of the business, on the front end, the sales employees will invest the prospective customers.

7. Once a customer places an order, their details are recorded into the customer table. The customer address details are saved as an XML data type, which can be updated.
8. The customer order details get recorded into the customer order table.
9. If the order is completed successfully, then the sales employee will receive an incentive- based on their base salary, and the record is stored in the incentive table. This also helps track the performance of the employees.
10. The employee details are kept within the employee table.
11. The details of the order invoices from the manufacturer and the customer, along with the mode of payment is not included in the scope of this system.
12. The profits are generated and displayed using views. This is based on the price purchased from the manufacturer and the price at which it is sold to the customer.

Business Requirements

1. Details of all the orders placed by the customer.
2. Tracking the performance of the employees based on the cars sold.
3. A 10% incentive on the base salary should be provided to the employees for each car sold.
4. All the customer order details will be stored as an XML.
5. Generating a list of all cars belonging to a particular car type.
6. Editing the employee address whenever it is required.
7. Generating a list of cars based on the seating capacity.

Business Rules

1) Every customer should have a unique customer ID (PK constraint) in the format ‘OVDC1000000’.

```
-- Insert Customer

GO
CREATE SEQUENCE SEQ_CUSTOMER_ID START WITH 1000000 INCREMENT BY 1;

INSERT INTO Customer VALUES (CONCAT('OVDC',NEXT VALUE FOR SEQ_CUSTOMER_ID),'Chaminda Vass','','09411290901')
INSERT INTO Customer VALUES (CONCAT('OVDC',NEXT VALUE FOR SEQ_CUSTOMER_ID),'Virat Kohli','','09188134541')
INSERT INTO Customer VALUES (CONCAT('OVDC',NEXT VALUE FOR SEQ_CUSTOMER_ID),'VVS Laxman','','09190987609')
INSERT INTO Customer VALUES (CONCAT('OVDC',NEXT VALUE FOR SEQ_CUSTOMER_ID),'Sachin Tendulkar','','09112122211')
INSERT INTO Customer VALUES (CONCAT('OVDC',NEXT VALUE FOR SEQ_CUSTOMER_ID),'Wasim Akram','','09013454321')
INSERT INTO Customer VALUES (CONCAT('OVDC',NEXT VALUE FOR SEQ_CUSTOMER_ID),'Mohomad Hafeez','','09023097895')
```

Figure 1 Unique Customer ID

2) Every employee should have a unique employee ID (PK constraint) in the format ‘OVDE1000’.

```
GO
CREATE SEQUENCE SEQ_EMPLOYEE_ID START WITH 1000 INCREMENT BY 1;

INSERT INTO Employee VALUES (CONCAT('OVDE', NEXT VALUE FOR SEQ_EMPLOYEE_ID),'Manik Mahashabde','Salesman','','1993-11-06','2341897AS','20000')
INSERT INTO Employee VALUES (CONCAT('OVDE',NEXT VALUE FOR SEQ_EMPLOYEE_ID),'Kumuditha Athukorala','Salesman','','1990-03-05','1233198ER','10000')
INSERT INTO Employee VALUES (CONCAT('OVDE',NEXT VALUE FOR SEQ_EMPLOYEE_ID),'Vint Sawant','Salesman','','1994-09-11','3452789XC','15000')
INSERT INTO Employee VALUES (CONCAT('OVDE',NEXT VALUE FOR SEQ_EMPLOYEE_ID),'Srikanth Pasam','Salesman','','1993-12-01','8967845BD','20000')
INSERT INTO Employee VALUES (CONCAT('OVDE',NEXT VALUE FOR SEQ_EMPLOYEE_ID),'Deep Singh','Salesman','','1995-05-11','5674389AS','12000')
INSERT INTO Employee VALUES (CONCAT('OVDE',NEXT VALUE FOR SEQ_EMPLOYEE_ID),'Salman Bhatt','Salesman','','1996-06-23','4563098AS','18000')
INSERT INTO Employee VALUES (CONCAT('OVDE',NEXT VALUE FOR SEQ_EMPLOYEE_ID),'Salman Khan','Manager','','1986-07-23','4563098HS','80000')
```

Figure 2 Unique Employee ID

3) A customer order entry should have a unique order ID (PK) and should have a customer ID as well as an employee ID associated with it (FK constraints).
4) A manufacturer order entry should have a unique order ID (PK) and should have a car model ID as a foreign key.
5) An inventory data will have a unique inventory ID, status as 'SOLD,' or 'AVAILABLE,' customer order ID can be null in case of 'AVAILABLE' status.
6) Incentive value for the employee should be 10% on his basic salary for every car sold.

```
-- 02 After
CREATE TRIGGER TriggerAddIncentive
ON Incentive
AFTER INSERT
AS
BEGIN
DECLARE @incentive_id int
DECLARE @incentive_date date
DECLARE @employee_id varchar(50)
DECLARE @salary int


SELECT @incentive_id = i.incentive_id FROM inserted i
SELECT @incentive_date = i.incentive_date FROM inserted i
SELECT @employee_id = i.employee_id FROM inserted i

SELECT @salary = (SELECT employee_salary FROM Employee WHERE employee_id = @employee_id)
PRINT(@salary)

UPDATE Incentive
SET incentive_amount = dbo.fn_Calc_Incentive(@salary)
WHERE incentive_id = @incentive_id
PRINT ('Incentive Added')
END
-----------------------------------------------------------------
INSERT INTO Incentive (incentive_id,incentive_date, employee_id)
VALUES (NEXT VALUE FOR SEQ_INCENTIVE_ID,'2019-10-30','OVDE1005')
GO
```

Figure 3 Incentive Trigger

7) Maintaining a log table for the data inserted into the customer order table.

```
CREATE TRIGGER triggerAddCustomerOrder
ON Customer_Order
INSTEAD OF INSERT
AS
BEGIN
DECLARE @order_time date
DECLARE @customer_order_delivery_date date
DECLARE @customer_order_selling_price int
DECLARE @customer_order_date date
DECLARE @customer_id varchar(50)
DECLARE @employee_id varchar(50)
DECLARE @cur_val int = NEXT VALUE FOR SEQ_CUSTOMER_ORDER_ID

SELECT @order_time = GETDATE()
SELECT @customer_order_date =i.customer_order_date FROM inserted i
SELECT @customer_order_delivery_date=i.customer_order_delivery_date FROM inserted i
SELECT @customer_order_selling_price =i.customer_order_selling_price FROM inserted i
SELECT @customer_id = i.customer_id FROM inserted i
SELECT @employee_id = i.employee_id FROM inserted i
IF EXISTS (SELECT customer_order_id FROM Customer_Order WHERE customer_order_id = @cur_val)
    BEGIN
        PRINT('Error, Order already exists')
    END
ELSE
    BEGIN
        INSERT INTO Customer_Order VALUES (@cur_val,@customer_order_date,@customer_order_delivery_date,@customer_order_selling_price,@customer_id,@employee_id)
        PRINT('Order Added')
        INSERT INTO dbo.Customer_Order_Log VALUES (@cur_val, @order_time, @customer_id, @employee_id,@customer_order_selling_price)
        PRINT('Order Added to Log Table')
    END
END

INSERT INTO Customer_Order
(customer_order_date, customer_order_delivery_date, customer_order_selling_price, customer_id, employee_id)
VALUES ('2019-10-01','2019-10-20','100000','OVDC1000003','OVDE1005')
```

Figure 4 Log Table

Entity-Relationship Diagram

Figure 5 ER Diagram

Relational Schema in 3NF

3NF is a Normal Form used to normalize a database design. This helps reduce duplication of data and improves referential integrity. This is done by making sure that the entity is in second normal form and no non-prime attribute is transitively dependent on any key. By having the Overdrives relational schema in 3NF, we have achieved the following:

1. Eliminated undesirable data
2. Reduced need for restructuring
3. Made the data model more informative
4. Made the data model neutral to different kinds of statistics

The figure below shows the Relational Schema for Overdrive dealership in 3NF:

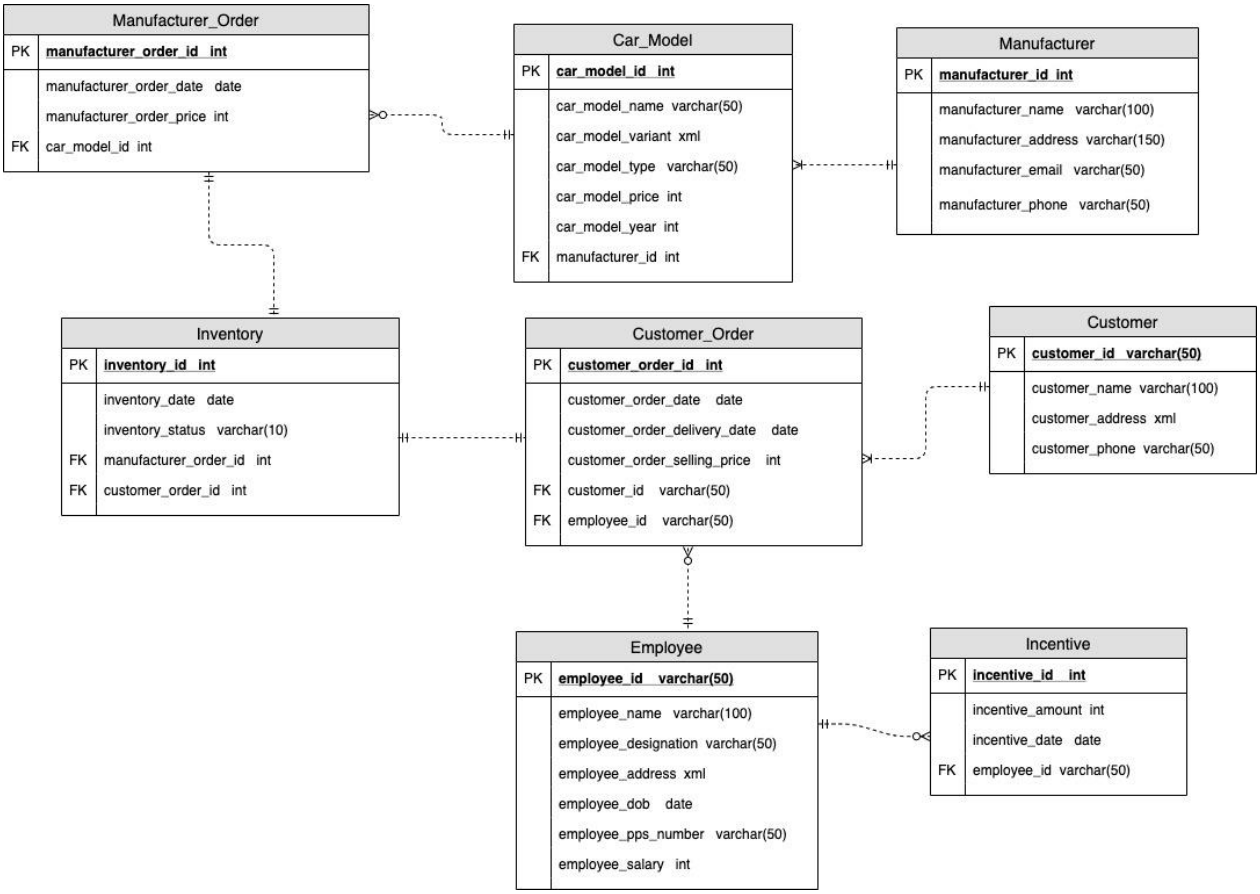


Figure 6 Relational Schema

The schema shows the hybrid database structure. While most of the attributes use relational structure, three of them are in the XML form. They are:

1. Customer address
2. Employee address
3. Car variant

The main reason behind using XML for these specific attributes is that they are semi-structured data. Address fields can vary widely based on the customer's location and type of residence like a building which will have a flat number and building name while individual residences will not have these fields. The other attribute is the car variant. This can vary depending on the category of the car. For example, a sports variant will have information regarding the car's acceleration while a family van will have information regarding the seating capacity. Because of these variations in data, XML will be effective over relation data structure here. This results in the Overdrives database becoming a hybrid database.

Implementation in SQL Server

Tables with Data Diagram

The data diagram for Overdrive car dealership is shown in the figure below:

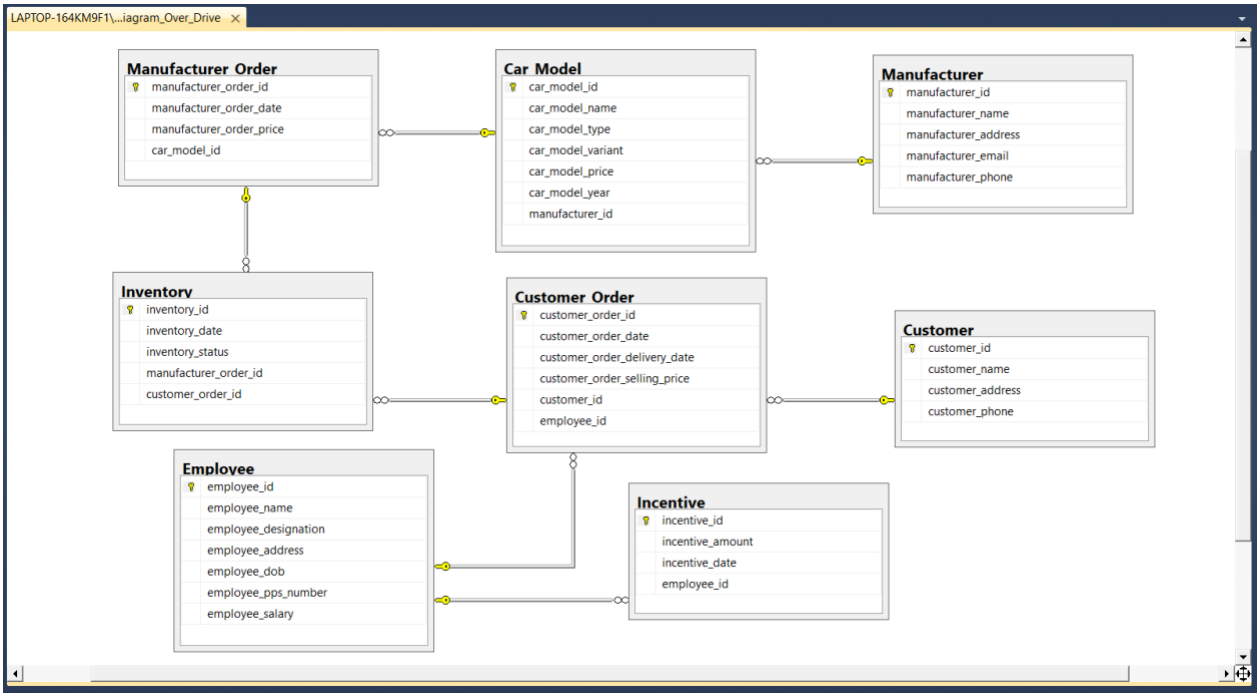


Figure 7 Data Diagram

According to the business requirements of the OverDrive database, it contains the data of vehicle manufactures, car models, customer, employee, and inventory. Also, it is capable of storing records of manufacturer order details, customer order details, and employee incentives. From the car dealer's perspective, it is mandatory to keep manufacturer records, car models, and manufacturer order details. The manufacturer table consists of all the car manufactures details and implemented manufacturer_id to identify a particular manufacturer uniquely. Car dealers keep the records of car models; there for the implemented table was Car_Model, which has car_model_id as the primary key. The relationship between the Manufacturer and Car_Model is one too many, and manufacturer_id is a foreign key of the Car_Model table. Therefore, foreign key constraints have been added to the Car_Model table. To keep the records of car dealer's order records, the Manufacturer_Order table is implemented, and the relationship between the Car_Model table and Manufacturer_Order is one to many because car dealers can place many orders based on one car model. Therefore car_model_id, which is the primary key of the Car_Model table presented in the Manufacturer_Order table as a foreign key. Foreign key constraints have been added to the Manufacturer_Order table. To represent the actual status of the cars which car dealer has

purchased and sold to the customers. The relationship between the Inventory and the Manufacturer_Order is one to many, and manufacturer_order_id and foreign key constrains has been added to the Inventory table. Based on the inventory records, the car dealer proceeds the customer orders. Assuming the one customer is purchasing one car at a time, the implemented relationship was one to one in between the Customer_Order table and Inventory table. To capture the customer orders based on car dealer inventory, the foreign key constraint has been added to the Inventory table, and customer_order_id is used as the foreign key in the Inventory table. A customer can place many orders; therefore, customer_id, which is the primary key of the Customer table, added as a foreign key to the Customer_Order table. Foreign key constraint has been added to the Customer_Order table. Employee table which holds the employee details has employee_id as primary key, and an employee is involved in a particular customer order. However, there can be zero or many customer order records related to an employee; therefore, Employee and Customer_Order table has zero to many relationships. Employee_id is a foreign key of the Customer_Order table, and foreign key constraint has been added to the Customer_Order table. Based on the number of orders employees are involved in, the car dealer is giving incentives. The incentive amount is 10% of the employee's salary. Employee table and Incentive table have one to many relationships; therefore, employee_id as the primary key is implemented as the foreign key in the Incentive table.

Stored Procedures

Stored procedures are used to group one or more logical SQL statements into one unit and are stored as objects in the database server. Usually, stored procedures are used when we want a certain code to be reused again and again. One can also pass values to the stored procedure to make it more useful and to get more detailed outputs from the same. Stored procedures resist SQL injection. Besides, stored procedures will reduce network traffic and increase performance (‘SQL Server Stored Procedures Tutorial’, no date).

For the OverDrive project, we have implemented our business requirements in terms of stored procedures.

Following are the seven business requirements that we had analyzed earlier and the screenshots of the same:

1. Use of JOIN between two or more tables as required –

   The stored procedure takes the input as a Customer Id and displays the details of all orders placed by the customer as well as the details of the customer. The output also displays the employee id associated who was involved in the selling of that car. This will be helpful for the management team to determine various entities involved in a successfully placed order.

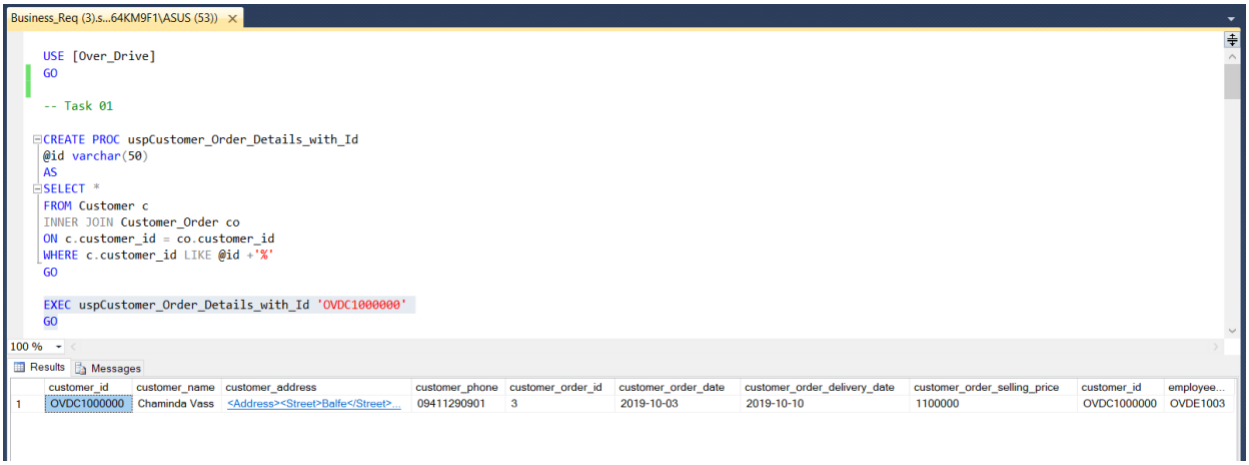


Figure 8 Stored Procedure 1

2. Use of GROUP BY with HAVING –

The stored procedure will take a cut-off price as input parameter and will display the list of all employees that were successful in selling the cars with cumulative prices above that cut-off price. So if the cut-off value is passed as 100000, then it will show all employees that have sold cars worth more than 100000. This will be helpful for the management of OverDrive to monitor the performance of the various employees.

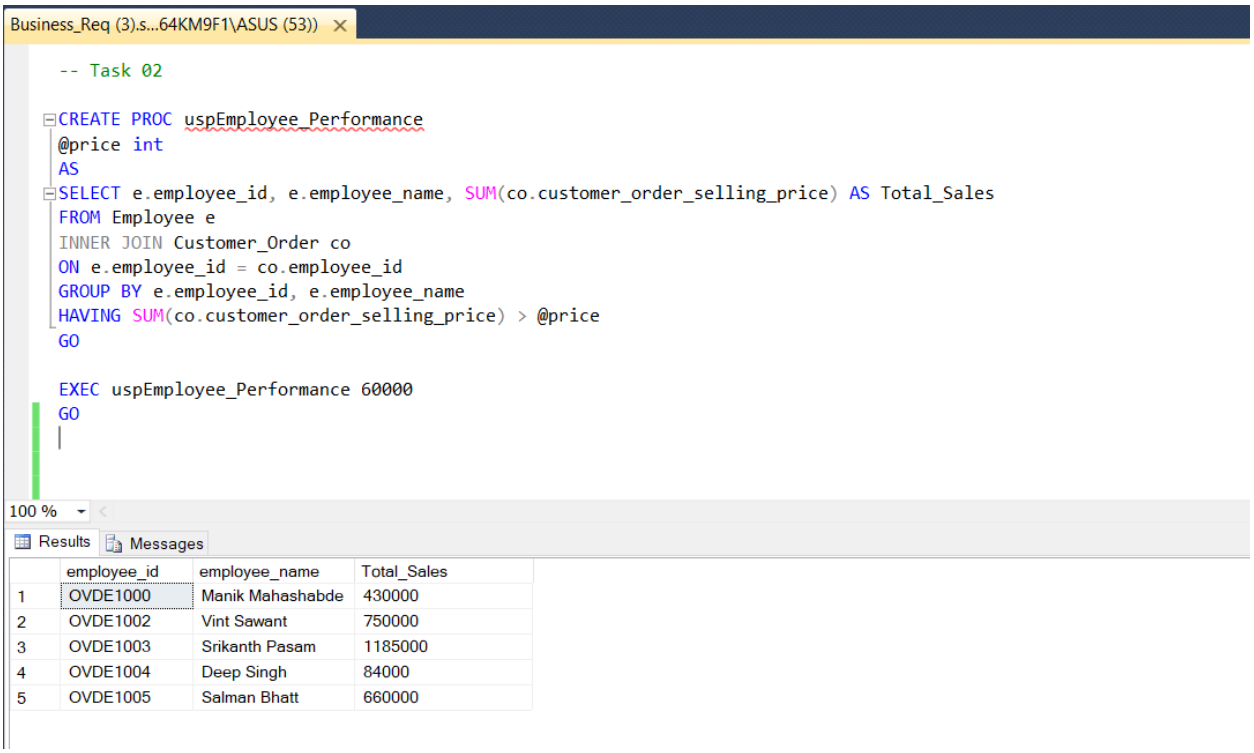


Figure 9 Stored Procedure 2

3. Use of custom SQL functions –
   The stored procedure will have a custom SQL function that will calculate the incentive value (bonus) by inputting the salary of the employee. The incentive value will be returned and will be 10 % of the employee's basic salary. This function will be used when giving incentives to employees after they are successful in completing an order from a customer. Such incentives are given in order to increase the morale of employees.

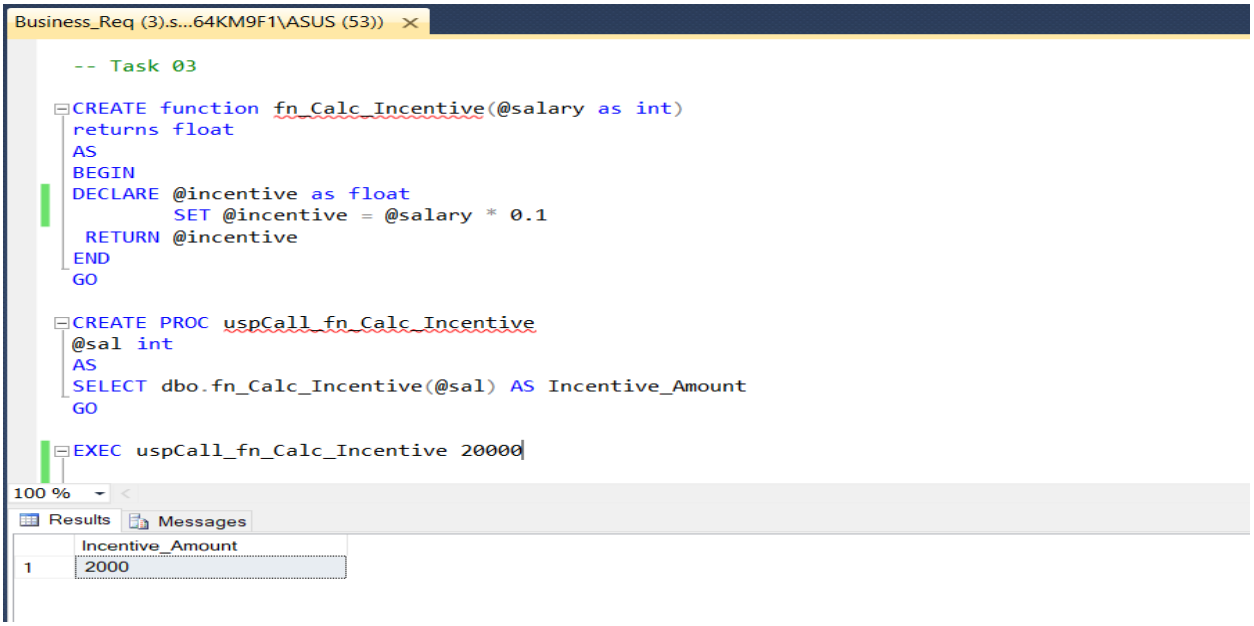


Figure 10 Stored Procedure 3

4. Developing XML with appropriate elements using relational fields –
   The stored procedure will develop an XML from the relational fields. The resultant XML will have details of customer id, car model and car manufacturer, customer selling price, and manufacturer order price. It will have an overall picture of the order fetched from various tables. This might be used to send the data in the form of XML to any other entities that might require it for audit purposes.

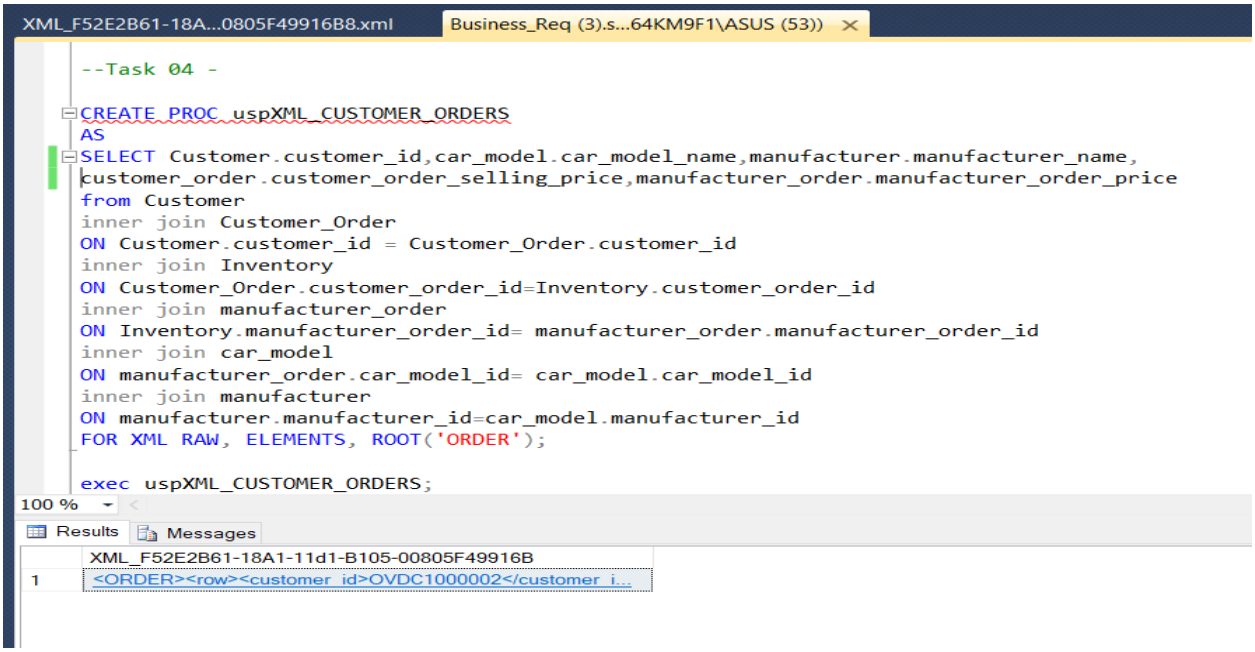


Figure 11 Stored Procedure 4.1

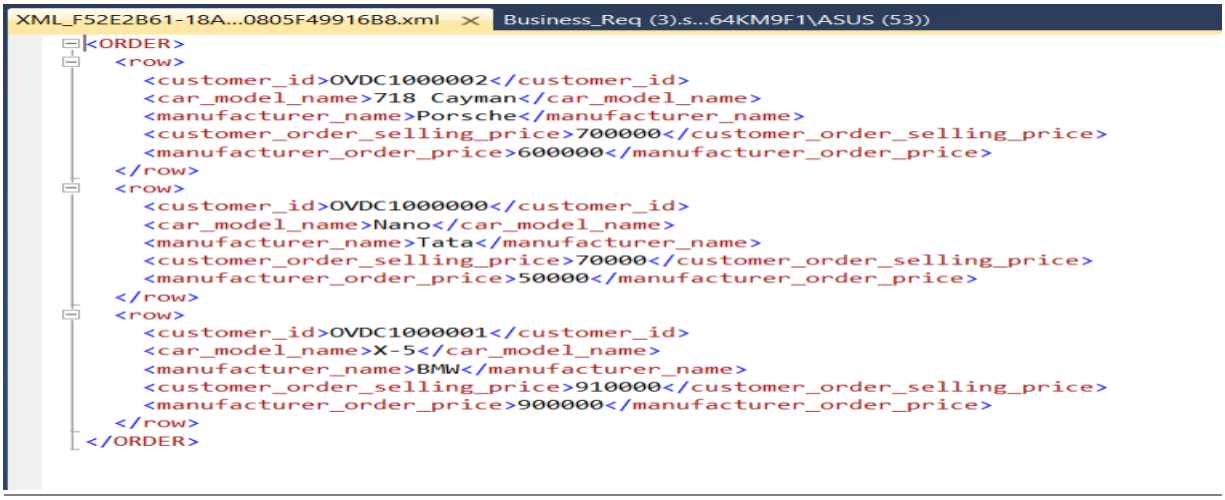


Figure 12 Stored Procedure 4.2

5. Retrieving data logically from a field with XML data type as well as data from fields from other data types –

   The stored procedure will take input as a car type, for example – Sports. The output will display all the cars that can be ordered through the OverDrive dealers. The output can be shown to the customers when they come with a specific intention of buying a particular car type. The output will have an XML data field as well, which will show the car specification.

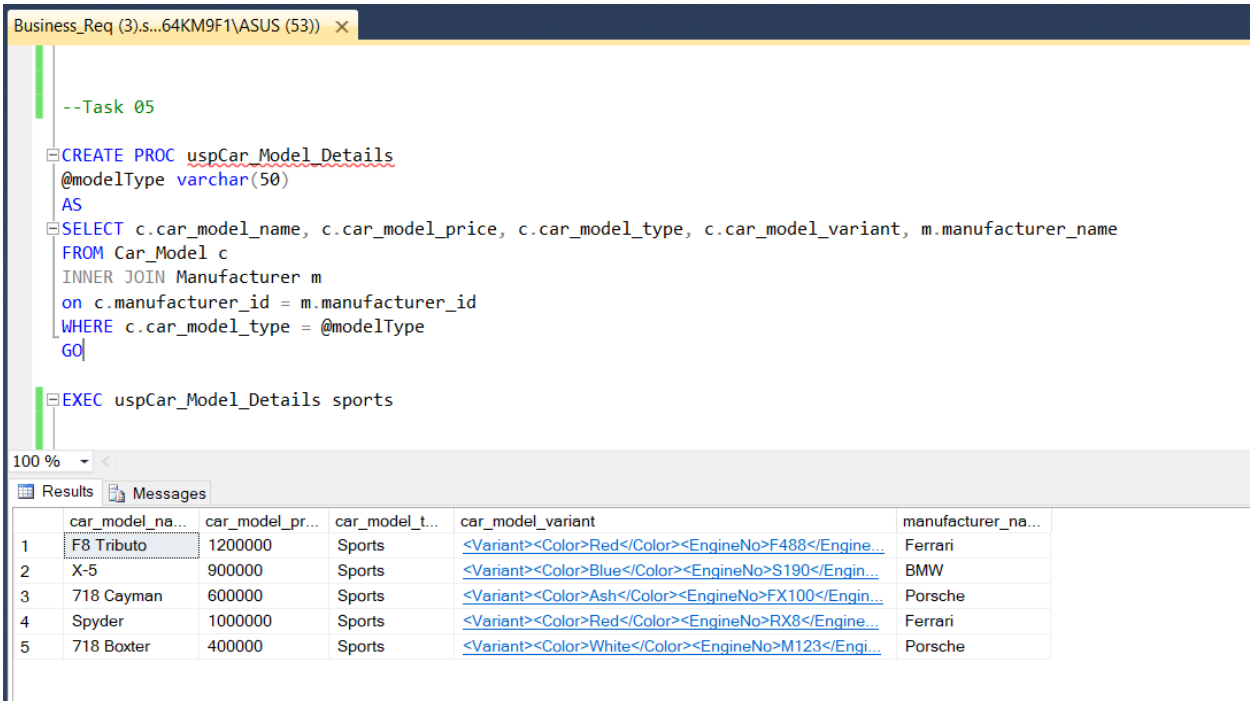


Figure 13 Stored Procedure 5

6. Modifying data in a field of XML data type –
   The stored procedure will be used to update one XML field, which is address in this case (street name). This will enable the OverDrive administration team to edit the employee address if there is a change in the same. The OverDrive aims to have updated details of all employees in all instances.

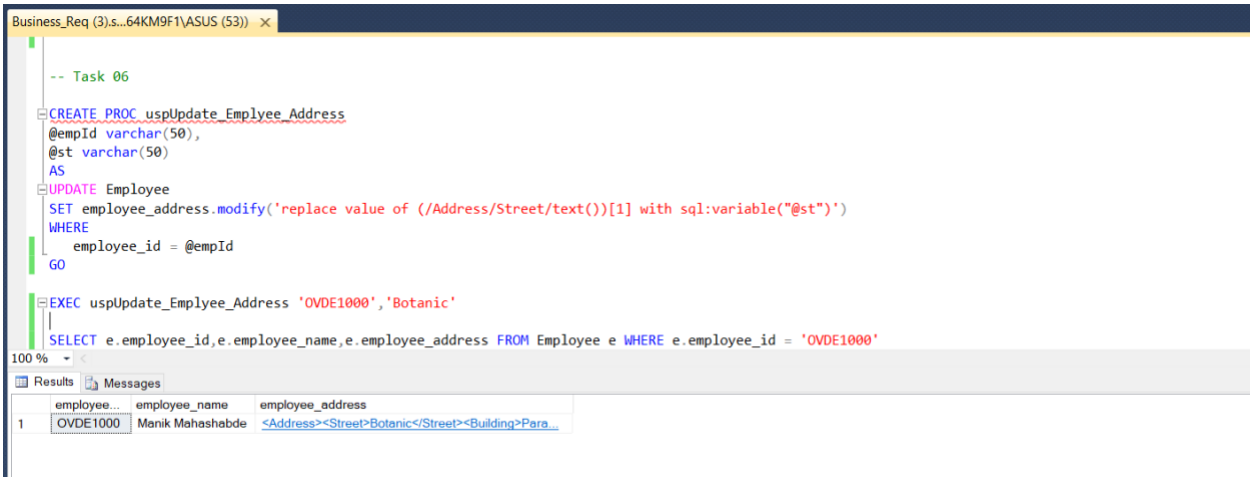


Figure 14 Stored Procedure 6

7. Searching data in a field of XML data type –
   The stored procedure will be used to search a particular input from the sales department for referring when a particular customer wants to see which all cars can be ordered, having a specified number of seats in them.

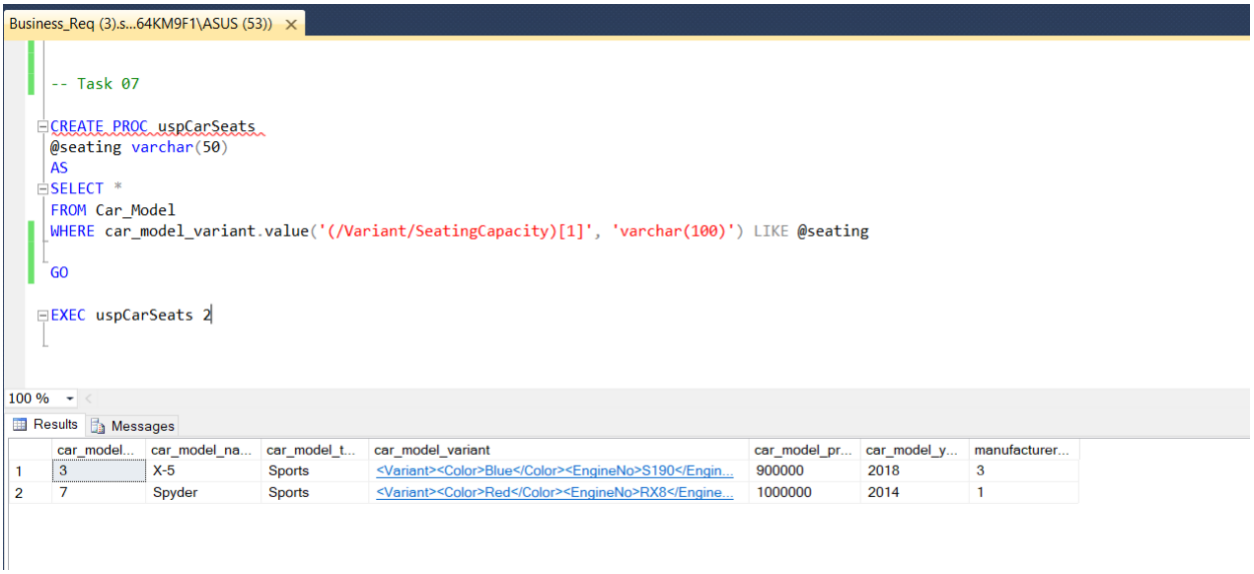


Figure 15 Stored Procedure 7

Triggers

Trigger is a kind of stored procedure that gets invoked automatically when a particular event takes place in a database. Triggers are used to maintain the referential integrity of data by changing the data in a systematic fashion. Triggers can be defined to run instead of or after DML (Data Manipulation Language) actions such as INSERT, UPDATE, and DELETE ('SQL Trigger | Student Database', 2018).

The OverDrive database implementation has two triggers implemented to enforce the business rules on the tables.

1. INSTEAD OF TRIGGER –

   Since the customer order table is the most important table to be maintained for the OverDrive dealers, and instead, the trigger has been implemented to back the data up inside a Customer order log table.

Figure 16 Trigger 1.1

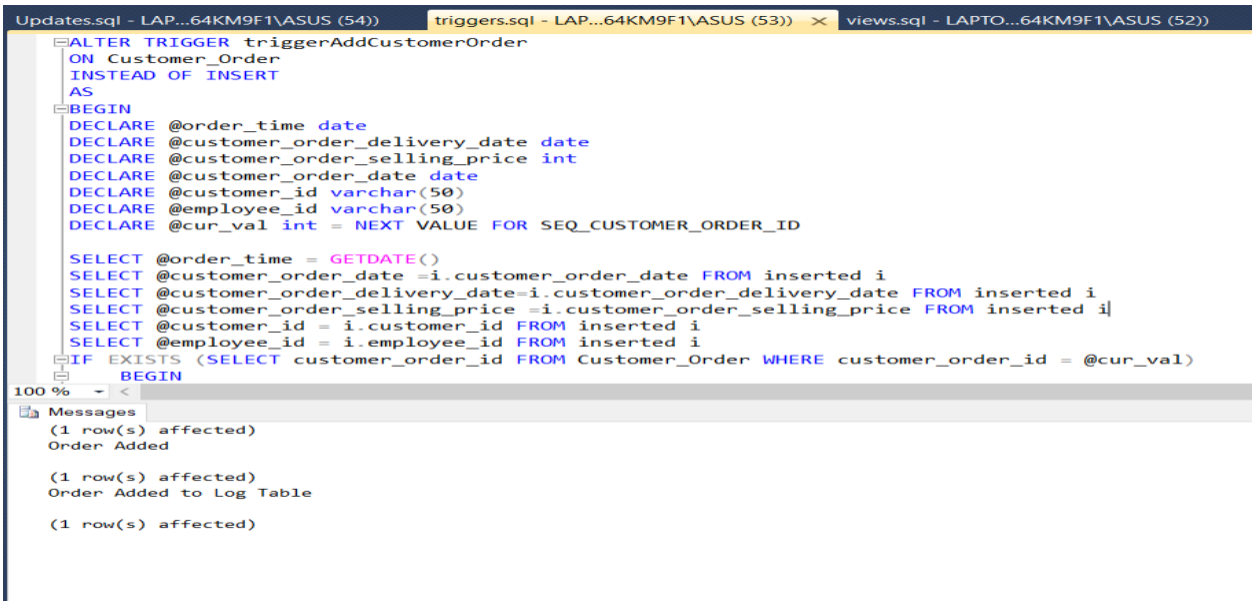
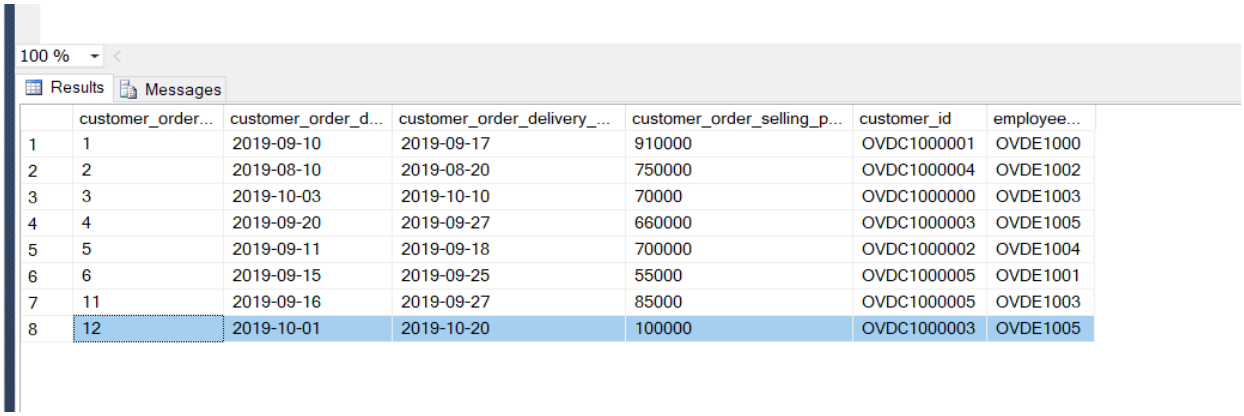


Figure 17 Trigger 1.2



100 %

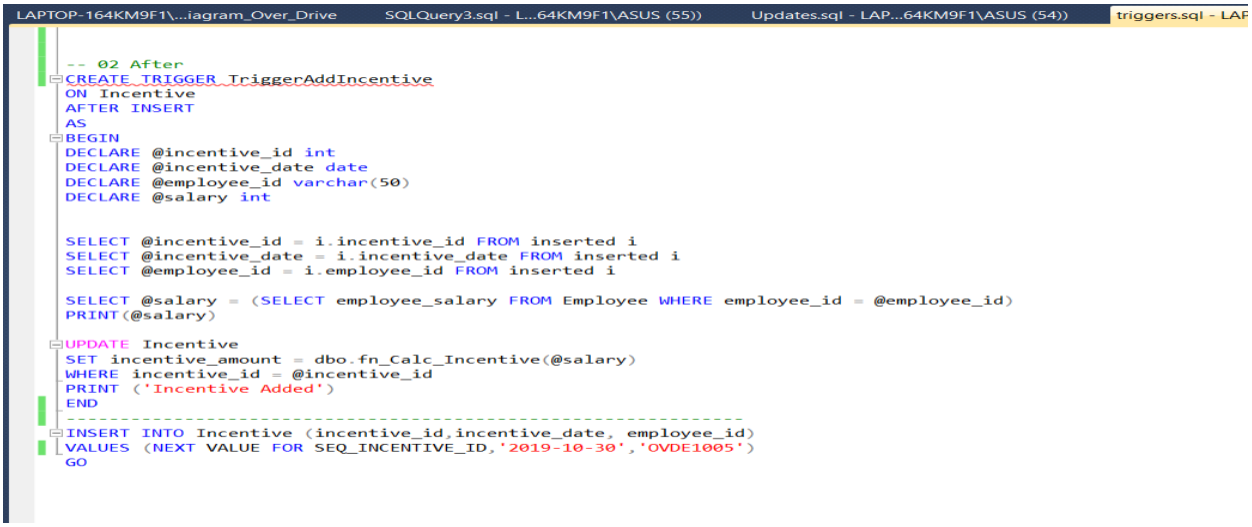
Results | Messages

	customer_order...	customer_order_d...	customer_order_delivery_...	customer_order_selling_p...	customer_id	employee...
1	1	2019-09-10	2019-09-17	910000	OVDC1000001	OVDE1000
2	2	2019-08-10	2019-08-20	750000	OVDC1000004	OVDE1002
3	3	2019-10-03	2019-10-10	70000	OVDC1000000	OVDE1003
4	4	2019-09-20	2019-09-27	660000	OVDC1000003	OVDE1005
5	5	2019-09-11	2019-09-18	700000	OVDC1000002	OVDE1004
6	6	2019-09-15	2019-09-25	55000	OVDC1000005	OVDE1001
7	11	2019-09-16	2019-09-27	85000	OVDC1000005	OVDE1003
8	12	2019-10-01	2019-10-20	100000	OVDC1000003	OVDE1005

Figure 18 Trigger 1.3

2. AFTER TRIGGER –

For inserting a record in the incentive table for employees and after the trigger is implemented, which calculates the incentive amount for that employee using a function and then inserts it in the respective incentive table. This ensures that the incentive amount is correct every time the insertion is made in the Incentive table.



LAPTOP-164KM9F1\...iagram_Over_Drive | SQLQuery3.sql - L...64KM9F1\ASUS (55)) | Updates.sql - LAP...64KM9F1\ASUS (54)) | triggers.sql - LAP

```
-- 02 After
CREATE TRIGGER TriggerAddIncentive
ON Incentive
AFTER INSERT
AS
BEGIN
DECLARE @incentive_id int
DECLARE @incentive_date date
DECLARE @employee_id varchar(50)
DECLARE @salary int

SELECT @incentive_id = i.incentive_id FROM inserted i
SELECT @incentive_date = i.incentive_date FROM inserted i
SELECT @employee_id = i.employee_id FROM inserted i

SELECT @salary = (SELECT employee_salary FROM Employee WHERE employee_id = @employee_id)
PRINT(@salary)

UPDATE Incentive
SET incentive_amount = dbo.fn_Calc_Incentive(@salary)
WHERE incentive_id = @incentive_id
PRINT ('Incentive Added')
END
-----------------------------------------------------------------
INSERT INTO Incentive (incentive_id,incentive_date, employee_id)
VALUES (NEXT VALUE FOR SEQ_INCENTIVE_ID,'2019-10-30','OVDE1005')
GO
```

Figure 19 Trigger 2.1

Updates.sql - LAP...64KM9F1\ASUS (54)) | triggers.sql - LAP...64KM9F1\ASUS (53)) | views.sql - LAPTO...64KM9F1\ASUS (52))

```
ALTER TRIGGER TriggerAddIncentive
ON Incentive
AFTER INSERT
AS
BEGIN
DECLARE @incentive_id int
DECLARE @incentive_date date
DECLARE @employee_id varchar(50)
DECLARE @salary int


SELECT @incentive_id = i.incentive_id FROM inserted i
SELECT @incentive_date = i.incentive_date FROM inserted i
SELECT @employee_id = i.employee_id FROM inserted i

SELECT @salary = (SELECT employee_salary FROM Employee WHERE employee_id = @employee_id)
PRINT(@salary)

UPDATE Incentive
SET incentive_amount = dbo.fn_Calc_Incentive(@salary)
WHERE incentive_id = @incentive_id
PRINT ('Incentive Added')
```

100 %

Messages

```
18000

(1 row(s) affected)
Incentive Added

(1 row(s) affected)
```

Figure 20 Trigger 2.2

100 %

Results | Messages

	incentive...	incentive_amo...	incentive_d...	employee...
1	8	NULL	2019-11-29	OVDE1005
2	9	NULL	2019-10-29	OVDE1005
3	11	2000	2019-10-29	OVDE1003
4	12	1800	2019-10-30	OVDE1005

Figure 21 Trigger 2.3

Views

A View in a database is kind of a searchable object. It does not store data and often referred to as a virtual table. Querying a view can be the same as querying a table. A view can combine data from two or more tables, using joins, and also just contain a subset of information. Views are basically used to hide the complexity of the data. Views take very little place to store, and it does not store a copy of the data that it represents (*What is a Relational Database View?*, 2014).

1. VIEW 1 –

   In this, the profit gained on every order is displayed. The profit gained will be the difference between the selling price to the customer and the purchasing price from the manufacturer. Such a view can be used by the manager as well as auditors to track the profit earned on the cars.

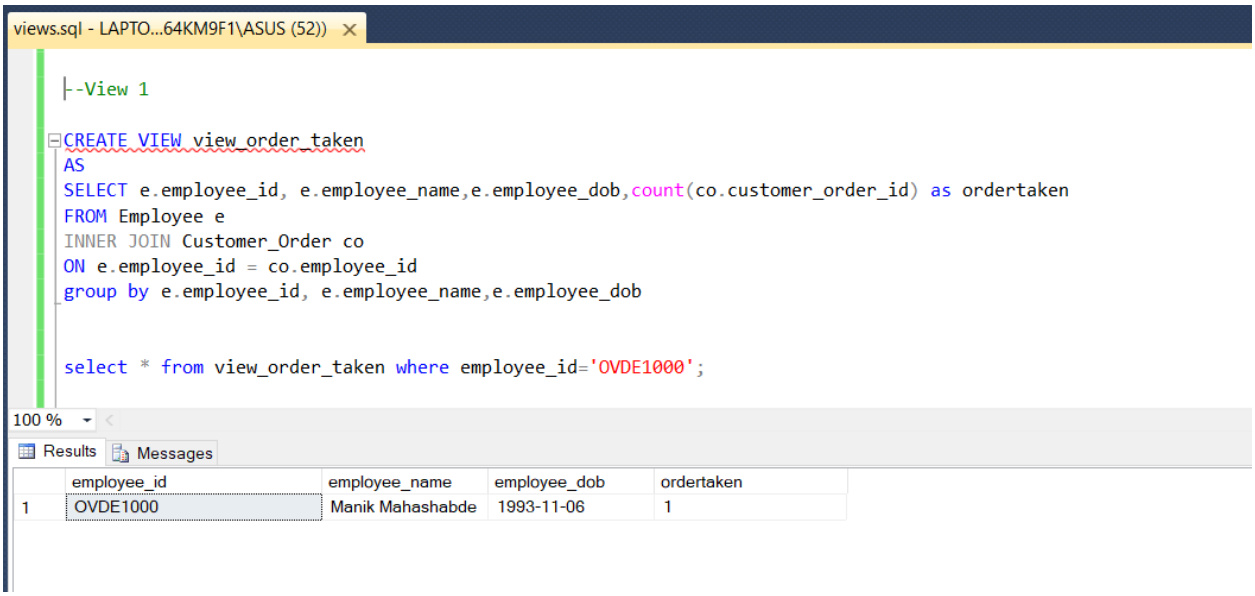


Figure 22 View 1

2. VIEW 2 –

   In this, the number of orders successfully achieved by every employee is displayed. This will help the manager to track the performing and under-performing employees at the OverDrive car dealers. That would influence his/her decision to give promotions and incentives.

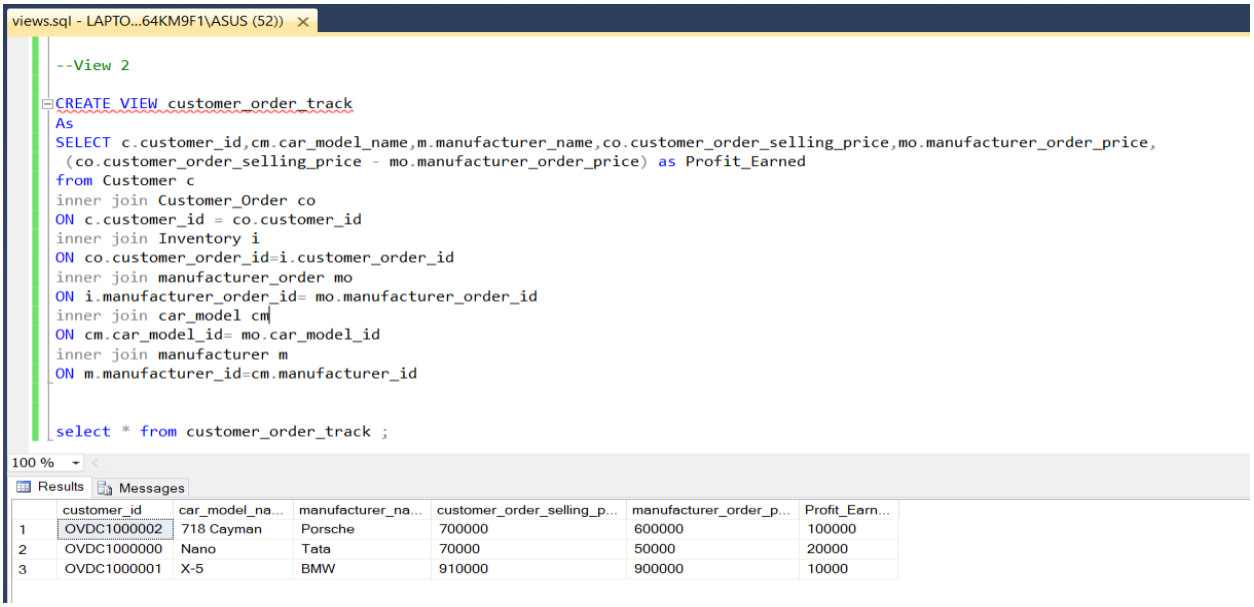


Figure 23 View 2

Innovation

In the modern age, business requirements are all about accessibility, speed, and scalability. These factors directly affect the growth of Overdrive car dealership. Trying to address these might require huge capital investment. The majority of the expenses during Overdrive's business expansion go into things like showroom, warehouse, vehicle procurement, personnel, and other such things. A lot of this cost can be reduced if certain common functional elements can be centralized like, for example, a common warehouse for stationing procured cars, single HR, and the Financial team to oversee operations across multiple branches. In order to enable this kind of collaboration among the multiple branches, a single database management system will be the best approach. This will help maintain centralized control over common data resources like employees, customers, manufacture orders, incentives, among others.

Cloud technologies can help achieve this for Overdrive. With Cloud technologies progressing leaps and bounds in the last decade, the ability to scale a business has been made accessible to anyone who desires at competitive prices. Taking advantage of these features, the Overdrive car dealership has come up with the innovative idea of moving their database management system to Azure SQL Database Management.

Azure SQL Databases provides the following advantages to Overdrive:

1. Database-as-a-service (DBaaS) – Azure provides SQL Database as a service platform with options ranging from general-purpose machines to Hyper scalable and business-critical ones. With the current business requirements, Overdrive chose to go ahead with a general-purpose serverless machine. These machines provide auto-scaling options depending on the minimum and maximum resource values set during initialization.
2. Hyper Scalable Database – With cloud databases, scaling the database is just a click of a button away. Within minutes Overdrive can fully scale up their database (up to 100TB) based on requirements. These reduce many infrastructure costs in building, maintaining, and upgrading a local database within the organization.
3. Azure SQL Database Intelligent Performance – Azure SQL databases provide intelligent performance insights like query performance insights, automatic tuning, continuous database monitoring to prevent disruptive events. The SQL Intelligent data protection also helps Overdrive to meet the data security and compliance requirements by proactively monitoring for potential threats and vulnerabilities.
4. Reduce cost and boost productivity – The ability to use all of the above features without having to purchase and set up any of the related hardware resources or personnel greatly reduces the business cost and increases productivity of Overdrive by letting the organization focus on the actual business rather than the chores of maintaining it. (*Three reasons Azure SQL Database is best for SQL Server migrations*, no date)

The figures below show the implementation of the Overdrive Database in Azure.

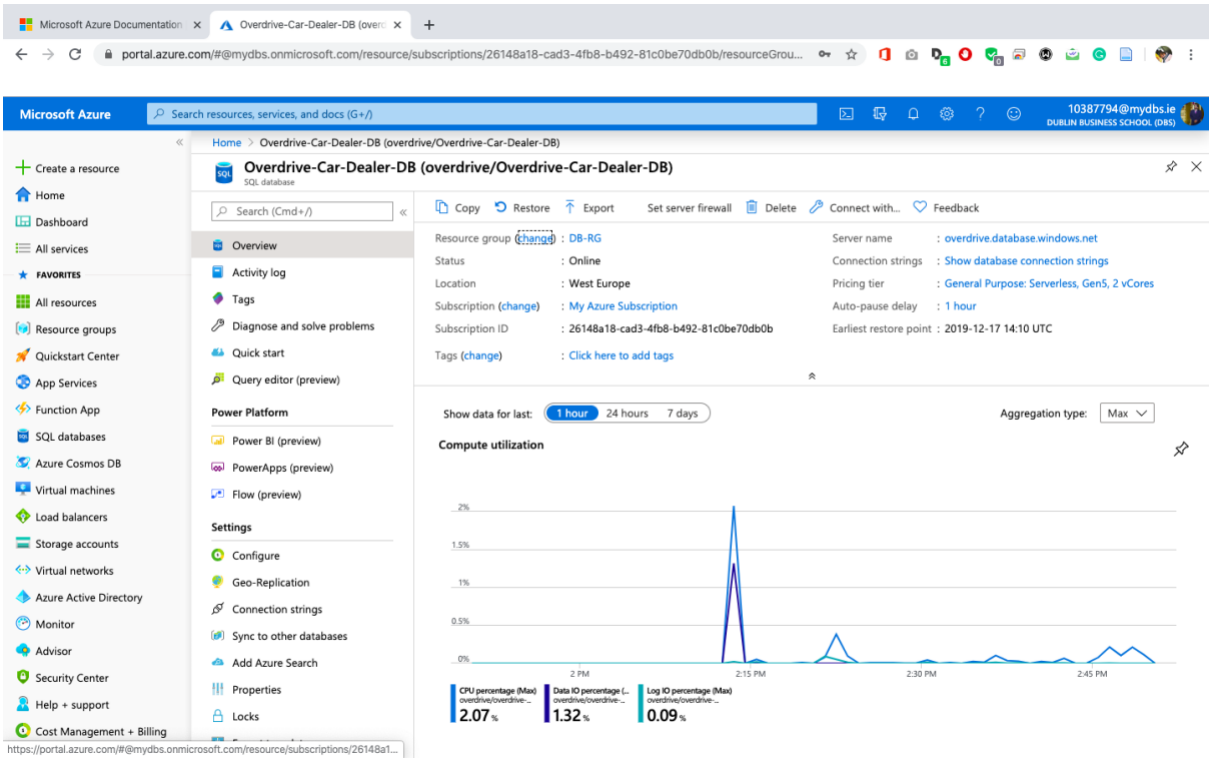


Figure 24 Azure DB Overview

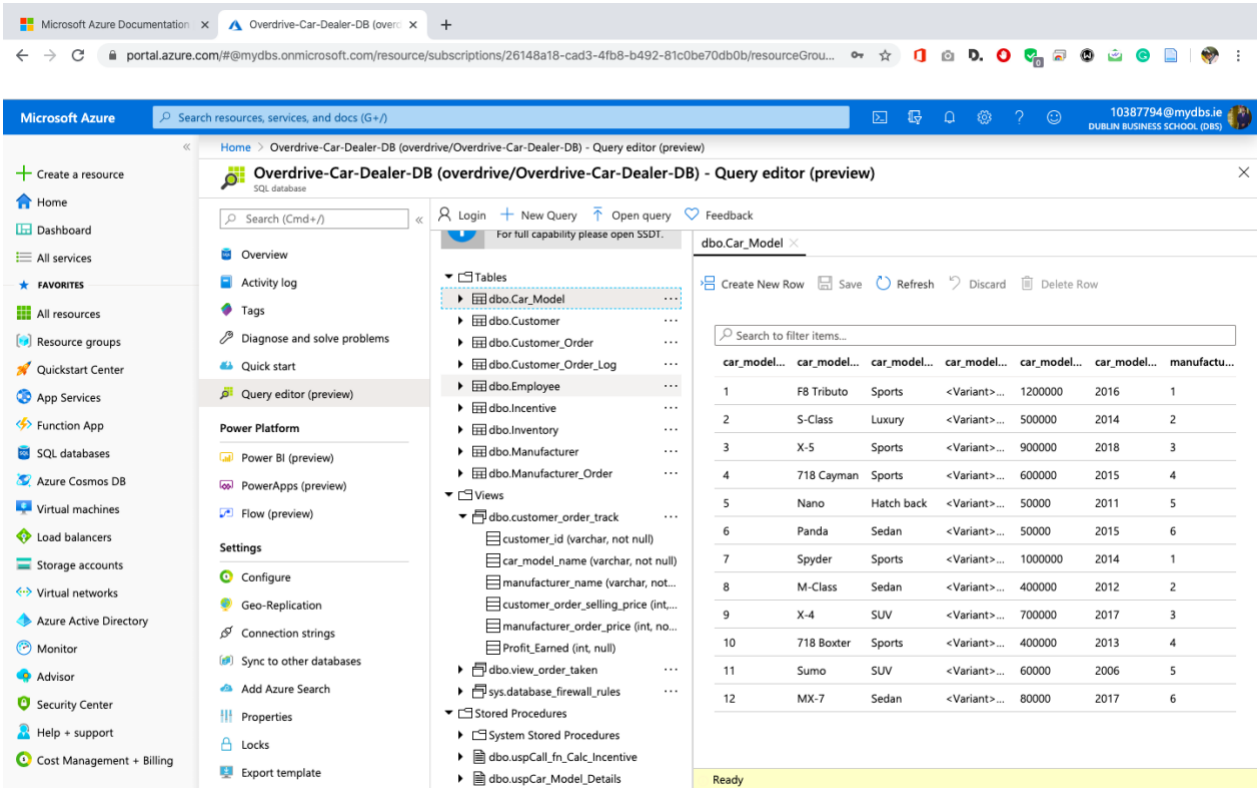


Figure 25 Azure DB Table View

Conclusion

From the screenshots, we can see that Overdrive car dealership has the Hybrid Database implemented successfully with all the business requirements fulfilled. For future expansion prospects, the same can also be hosted on the cloud, as depicted in the innovation section.

Individual Contribution

Vinit Sawant

Contributed to the selection of the business case to be implemented for the CA. Decided on the entities to be taken up for the scenario and the various business requirements which the business can have and which are essential for the functioning of the OverDrive car dealer's enterprise. Discussed with the team members about the various business rules that are essential and can be enforced to maintain a proper set of data in the database. Contributed in developing the ER diagrams and relational schema for the project. Discussed with the team on what fields need to be included in the various tables and what data columns should have the XML data type. Mainly those data fields were suggested by me, were semi-structured, and are subject to vary based on the entry like car specification as well as the address of the customer or the employee of OverDrive car dealers. I went through the various websites to understand the concept of stored procedures and the various benefits of using the stored procedures that our enterprise can get after the implementation of business requirements in the same. Decided upon the exact scenarios, where the triggers can be used and before that went through the lecture slides on triggers and different types of triggers. Understood what Views are? Moreover, how they can be used to present the data in a customized manner to the end-user or interested client without revealing the structure of the table or compromising on the storage of the database as views are just virtual tables. Contributed directly to coding the business requirements in the form of stored procedures, and views. Suggested the team member Kumuditha Athukorala on what needs to implement for the triggers section. Participated and lead the group meetings to discuss the various functionalities, analysis, planning, and implementation and, last but not least, content to be added in the report to make it a decent professional report. It was indeed a great learning experience to go through the various concepts of the Advanced Databases module for implementation of the Hybrid database for the OverDrive car dealers. A real-world market kind scenario was experienced by the team members during the implementation. Understanding each and every concept from the very scratch and implementing the same in the real world was a great learning outcome and would help me surely in the future. Besides learning the technical part of the implementation, the experience to lead the team and manage the various tasks, allocate them to the team members was also a milestone that was achieved within the decided timeline of the project. The minimum planned effort from each member every week was achieved by giving small tasks to the team members, and later a discussion on the same used to take place if anyone faced during the execution of the same. Also, due to the ever-guiding nature of our module guide Dr. Shazia Afzal we were able to achieve all the functionalities of our project.

Kumuditha Athukorala

After the discussion of the CA brief, three of us decided to implement a hybrid database system for a car dealer named OverDrive. From the beginning, I contributed my effort in order to implement the Overdrive database system. I have followed may web applications that are available on the internet to identify the business scenario and outline the scope of our database system. Then I initiated the business requirements collaboratively in order to proceed with the ER diagrams. After identifying the entities of the system, I draw the ER diagram. There I used the Advanced Databases theories and concepts which I have learned from this program. After the feedback we received from the lecturer, I have changed the ER diagram accordingly and develop the Database schema diagram based on that. According to the normalization theories, I have designed the schema diagram in 3NF to draw the diagrams; I used crow's foot notation. There we collaboratively define the data types with the length. Also, we collaboratively figure out XML fields based on the business scenario. Then I created the Overdrive database with all the tables. The overdrive database system consists of tables, triggers, views, and stored procedures. During the timetable creation for the Overdrive database, in order to apply the referential integrity and constraints, I have followed a few web sites and examples which I learned from this program. After the completion of the relational schema, I contributed my effort to insert sample data into the database. Then I generated the data diagram in SQL Server Management Studio. Then I have implemented most of the stored procedures in order to address the business requirements. In order to meet the business requirements, I have used to join quires with the tables. Also, I have modified stored procedures with arguments. To update the XML fields of the database tables, I designed the stored procedure with parameters as well, and there I had to learn about the sorted procedures, and I have followed many web sites for that. Remain sorted procedures we have implemented collaboratively based on our business requirements. In the Overdrive database, we have identified triggers to implement, myself and Vinit implemented the triggers in order to fulfill the business requirement.

The reflection of this project, for me, is really valuable since these concepts and theories are using in the software industry. Concerning my contribution to this CA, learnings such as ER diagrams, Database schema diagrams, normalization concepts were important to me. As well as to implement stored procedures based on business requirements, learnings and the class exercises were useful for me, and I learned how to implement stored procedures with arguments and how to manipulate XML values via stored procedure as well. Then I came to know how to use triggers on top of a database and how to alter these database objects if there is anything to be updated. At last but not least, I convey my gratitude to Dr. Shazia A Afzal regarding the given guidance.

Srikanth Shilesh Pasam

Reflection –

The Advanced Databases CA has been one of the most indulging learning experiences for me. I come from Electronics and Communication Engineering background with no prior knowledge of the subject. My professional experience has been majorly in Teach for India, a non-government organization within the education sector. So, I have no professional experience working with databases either. Because of this, I had to put in extra effort to learn everything from scratch. One of the most important things I am thankful for with regards to the CA is the amount of time that was given to submit it. I have been able to utilize this time to the fullest in catching up to the standard expected at the master's level. The course professor, Dr. Shazia Afzal, has been extremely supportive in this regard. Her empathy towards students who were similar to me and her patience in the class by walking through every single line of code has been of tremendous help. This was more evident when she broke down the CA into smaller deliverables making sure during every step that the project is flowing on the right track. If not for that, I would have been still struggling with many things.

I faced the most difficulty during the initial days of the project, where I was unable even to plot a plan of action on how to proceed. Most of this time, I engrossed myself with various books from the library and online courses on Udemy. The progress was slow, and whenever I heard my fellow students discuss their progress with their CA's, it used to fill me with many panics. However, I kept pushing myself, and through my perseverance, I slowly but steadily started to make progress with my CA.

Teamwork has been the quintessential part of our group. Every single aspect of the CA has been discussed and communicated with each other. A large part of the brainstorming sessions involved us questioning each other. Through techniques like 5, Why are we pushed each other's thinking in order to deepen our understanding. My teammates have been one of my strong pillars of support. They have been extremely patient with me and gave me the time and space to catch up with them and contribute to the CA. They have been able to address all my queries regarding the course whenever I got stuck anywhere.

Contribution –

The project has been divided into multiple parts, with each of us taking up ownership for our respective topics. Apart from the combined contributions put into every aspect of the CA, my contribution has been towards the Business Requirements and innovation. While my team has been working on implementing the database on MS SQL, I have been parallelly trying to get it done in Azure SQL Database. This approach helped us gain different perspectives about the various queries and table implementation logic. There were even instances where we went back to rethinking the entire business model because of the difference in perspectives. Finally, towards the end of the CA, we collaborated all our work into one CA.

Bibliography

Rouse, M. (no date) *What is a Database? - Definition from WhatIs.com*, *SearchSQLServer*. Available at: https://searchsqlserver.techtarget.com/definition/database (Accessed: 10 December 2019).

'SQL Server Stored Procedures Tutorial' (no date) *SQL Server Tutorial*. Available at: http://www.sqlservertutorial.net/sql-server-stored-procedures/ (Accessed: 17 December 2019).

'SQL Trigger | Student Database' (2018) *GeeksforGeeks*, 24 September. Available at: https://www.geeksforgeeks.org/sql-trigger-student-database/ (Accessed: 17 December 2019).

Three reasons Azure SQL Database is best for SQL Server migrations (no date). Available at: https://azure.microsoft.com/en-us/blog/three-reasons-azure-sql-database-is-best-for-sql-server-migration/ (Accessed: 17 December 2019).

What is a database? (no date). Available at: https://www.oracle.com/database/what-is-database.html (Accessed: 10 December 2019).

What is a relational database? (no date). Available at: https://www.oracle.com/ie/database/what-is-a-relational-database/ (Accessed: 10 December 2019).

What is a Relational Database View? (2014) *Essential SQL*. Available at: https://www.essentialsql.com/what-is-a-relational-database-view/ (Accessed: 17 December 2019).

What is Microsoft SQL Server, and What is it Used For? (2017) *Infotec*. Available at: https://www.infotectraining.com/blog/what-is-microsoft-sql-server-and-what-is-it-used-for (Accessed: 10 December 2019).

Appendix 1

```
CREATE TABLE Manufacturer(
manufacturer_id int not null,
manufacturer_name varchar(100) not null UNIQUE,
manufacturer_address varchar(150) not null,
manufacturer_email varchar(50) not null,
manufacturer_phone varchar(50) not null,
CONSTRAINT manufacturer_id_pk PRIMARY KEY (manufacturer_id)
)
GO

CREATE TABLE Car_Model(
car_model_id int not null,
car_model_name varchar(50) not null UNIQUE,
car_model_type varchar(50) not null,
car_model_variant xml,
car_model_price int not null,
car_model_year int not null,
manufacturer_id int not null,
CONSTRAINT car_model_id_pk PRIMARY KEY (car_model_id),
CONSTRAINT manufacturer_id_fk FOREIGN KEY (manufacturer_id)
REFERENCES Manufacturer(manufacturer_id)
)
GO

CREATE TABLE Manufacturer_Order(
```

```
manufacturer_order_id int not null,
manufacturer_order_date date not null,
manufacturer_order_price int not null,
car_model_id int not null,
CONSTRAINT manufacturer_order_id_pk PRIMARY KEY (manufacturer_order_id),
CONSTRAINT car_model_id_fk FOREIGN KEY (car_model_id)
REFERENCES Car_Model(car_model_id)
)
GO

CREATE TABLE Customer(
customer_id varchar(50) not null,
customer_name varchar(100) not null,
customer_address xml,
customer_phone varchar(50) not null,
CONSTRAINT customer_id_pk PRIMARY KEY (customer_id)
)
GO


CREATE TABLE Employee(
employee_id varchar(50) not null,
employee_name varchar(100) not null,
employee_designation varchar(50) not null,
employee_address xml,
employee_dob date not null,
employee_pps_number varchar(100) not null,
employee_salary int not null,
```

```
CONSTRAINT employee_id_pk PRIMARY KEY (employee_id)
)
GO

CREATE TABLE Customer_Order(
customer_order_id int not null,
customer_order_date date not null,
customer_order_delivery_date date not null,
customer_order_selling_price int not null,
customer_id varchar(50) not null,
employee_id varchar(50) not null,
CONSTRAINT customer_order_id_pk PRIMARY KEY (customer_order_id),
CONSTRAINT customer_id_fk FOREIGN KEY (customer_id)
REFERENCES Customer(customer_id),
CONSTRAINT employee_id_fk1 FOREIGN KEY (employee_id)
REFERENCES Employee(employee_id)
)
GO

CREATE TABLE Incentive(
incentive_id int not null,
incentive_amount int,
incentive_date date not null,
employee_id varchar(50) not null,
CONSTRAINT incentive_id_pk PRIMARY KEY (incentive_id),
CONSTRAINT employee_id_fk2 FOREIGN KEY (employee_id)
REFERENCES Employee(employee_id)
)
```

```
GO

CREATE TABLE Inventory(
inventory_id int not null,
inventory_date date not null,
inventory_status varchar(10) not null,
manufacturer_order_id int not null,
customer_order_id int,
CONSTRAINT inventory_id_pk PRIMARY KEY (inventory_id),
CONSTRAINT manufacturer_order_id_fk FOREIGN KEY (manufacturer_order_id)
REFERENCES Manufacturer_Order(manufacturer_order_id),
CONSTRAINT customer_order_id_fk FOREIGN KEY (customer_order_id)
REFERENCES Customer_Order(customer_order_id),
)
GO


CREATE TABLE Customer_Order_Log(
order_id int not null  PRIMARY KEY,
order_time date not null,
customer_id varchar(50) not null,
employee_id varchar(50) not null,
customer_order_selling_price int not null,
)
```

Appendix 2

```
-- Task 01


CREATE PROC uspCustomer_Order_Details_with_Id
@id varchar(50)
AS
SELECT *
FROM Customer c
INNER JOIN Customer_Order co
ON c.customer_id = co.customer_id
WHERE c.customer_id LIKE @id +'%'
GO

EXEC uspCustomer_Order_Details_with_Id 'OVDC1000000'
GO

-- Task 02


CREATE PROC uspEmployee_Performance
@price int
AS
SELECT e.employee_id, e.employee_name, SUM(co.customer_order_selling_price) AS
Total_Sales
FROM Employee e
INNER JOIN Customer_Order co
ON e.employee_id = co.employee_id
```

```
GROUP BY e.employee_id, e.employee_name
HAVING SUM(co.customer_order_selling_price) > @price
GO

EXEC uspEmployee_Performance 60000
GO

-- Task 03

CREATE function fn_Calc_Incentive(@salary as int)
returns float
AS
BEGIN
DECLARE @incentive as float

                SET @incentive = @salary * 0.1

 RETURN @incentive
END
GO

CREATE PROC uspCall_fn_Calc_Incentive
@sal int
AS
SELECT dbo.fn_Calc_Incentive(@sal) AS Incentive_Amount
GO

EXEC uspCall_fn_Calc_Incentive 20000
```

```
--Task 04 -

CREATE PROC uspXML_CUSTOMER_ORDERS
AS
SELECT
Customer.customer_id,car_model.car_model_name,manufacturer.manufacturer_name,c
ustomer_order.customer_order_selling_price,manufacturer_order.manufacturer_order_
price
from Customer
inner join Customer_Order
ON Customer.customer_id = Customer_Order.customer_id
inner join Inventory
ON Customer_Order.customer_order_id=Inventory.customer_order_id
inner join manufacturer_order
ON Inventory.manufacturer_order_id= manufacturer_order.manufacturer_order_id
inner join car_model
ON manufacturer_order.car_model_id= car_model.car_model_id
inner join manufacturer
ON manufacturer.manufacturer_id=car_model.manufacturer_id
FOR XML RAW, ELEMENTS, ROOT('ORDER');

exec uspXML_CUSTOMER_ORDERS;

--Task 05

CREATE PROC uspCar_Model_Details
@modelType varchar(50)
```

```
AS
SELECT c.car_model_name, c.car_model_price, c.car_model_type, c.car_model_variant,
m.manufacturer_name
FROM Car_Model c
INNER JOIN Manufacturer m
on c.manufacturer_id = m.manufacturer_id
WHERE c.car_model_type = @modelType
GO

EXEC uspCar_Model_Details sports

--Task 06

CREATE PROC uspUpdate_Emplyee_Address
@empId varchar(50),
@st varchar(50)
AS
UPDATE Employee
SET employee_address.modify('replace value of (/Address/Street/text())[1] with
sql:variable("@st")')
WHERE
  employee_id = @empId

EXEC uspUpdate_Emplyee_Address 'OVDE1000','Botanic'

--Task 07

CREATE PROC uspCarSeats
@seating varchar(50)
```

```
AS
SELECT *
FROM Car_Model
WHERE car_model_variant.value('(/Variant/SeatingCapacity)[1]', 'varchar(100)') LIKE
@seating

EXEC uspCarSeats 2
```

Appendix 3

```
-- Insert Manufacturer

GO

CREATE SEQUENCE SEQ_MANUFACTURER_ID START WITH 1 INCREMENT BY 1;


GO

SELECT * FROM Manufacturer

GO

INSERT INTO Manufacturer VALUES (NEXT VALUE FOR
SEQ_MANUFACTURER_ID,'Ferrari','Maranello,Italy','ferraicars@gmail.com','1800553946')

INSERT INTO Manufacturer VALUES (NEXT VALUE FOR
SEQ_MANUFACTURER_ID,'Mercedes','Stuttgrad,Germany','mercedesinfo@gmail.com','08
0097777777')

INSERT INTO Manufacturer VALUES (NEXT VALUE FOR
SEQ_MANUFACTURER_ID,'BMW','Munich,Germany','bmw@gmail.com','49893820')

INSERT INTO Manufacturer VALUES (NEXT VALUE FOR
SEQ_MANUFACTURER_ID,'Porsche','Stuttgrad,Germany','porsche@gmail.com','49875622
3')

INSERT INTO Manufacturer VALUES (NEXT VALUE FOR
SEQ_MANUFACTURER_ID,'Tata','Mumbai,India','tata@gmail.com','18002582553')

INSERT INTO Manufacturer VALUES (NEXT VALUE FOR
SEQ_MANUFACTURER_ID,'Micro','Colombo,Sri Lanka','micro@msn.com','94089554416')


-- Insert Car_Model


GO

CREATE SEQUENCE SEQ_CAR_MODEL_ID START WITH 1 INCREMENT BY 1;
```

```
INSERT INTO Car_Model VALUES (NEXT VALUE FOR SEQ_CAR_MODEL_ID,'F8 Tributo','Sports','','1200000','2016','1')

INSERT INTO Car_Model VALUES (NEXT VALUE FOR SEQ_CAR_MODEL_ID,'S-Class','Luxury','','500000','2014','2')

INSERT INTO Car_Model VALUES (NEXT VALUE FOR SEQ_CAR_MODEL_ID,'X-5','Sports','','900000','2018','3')

INSERT INTO Car_Model VALUES (NEXT VALUE FOR SEQ_CAR_MODEL_ID,'718 Cayman','Sports','','600000','2015','4')

INSERT INTO Car_Model VALUES (NEXT VALUE FOR SEQ_CAR_MODEL_ID,'Nano','Hatch back','','50000','2011','5')

INSERT INTO Car_Model VALUES (NEXT VALUE FOR SEQ_CAR_MODEL_ID,'Panda','Sedan','','50000','2015','6')


INSERT INTO Car_Model VALUES (NEXT VALUE FOR SEQ_CAR_MODEL_ID,'Spyder','Sports','','1000000','2014','1')

INSERT INTO Car_Model VALUES (NEXT VALUE FOR SEQ_CAR_MODEL_ID,'M-Class','Sedan','','400000','2012','2')

INSERT INTO Car_Model VALUES (NEXT VALUE FOR SEQ_CAR_MODEL_ID,'X-4','SUV','','700000','2017','3')

INSERT INTO Car_Model VALUES (NEXT VALUE FOR SEQ_CAR_MODEL_ID,'718 Boxter','Sports','','400000','2013','4')

INSERT INTO Car_Model VALUES (NEXT VALUE FOR SEQ_CAR_MODEL_ID,'Sumo','SUV','','60000','2006','5')

INSERT INTO Car_Model VALUES (NEXT VALUE FOR SEQ_CAR_MODEL_ID,'MX-7','Sedan','','80000','2017','6')



-- Insert Employee OVDE1000


GO

CREATE SEQUENCE SEQ_EMPLOYEE_ID START WITH 1000 INCREMENT BY 1;
```

```
INSERT INTO Employee VALUES (CONCAT('OVDE', NEXT VALUE FOR
SEQ_EMPLOYEE_ID),'Manik Mahashabde','Salesman','','1993-11-06','2341897AS','20000')

INSERT INTO Employee VALUES (CONCAT('OVDE',NEXT VALUE FOR
SEQ_EMPLOYEE_ID),'Kumuditha Athukorala','Salesman','','1990-03-
05','1233198ER','10000')

INSERT INTO Employee VALUES (CONCAT('OVDE',NEXT VALUE FOR
SEQ_EMPLOYEE_ID),'Vint Sawant','Salesman','','1994-09-11','3452789XC','15000')

INSERT INTO Employee VALUES (CONCAT('OVDE',NEXT VALUE FOR
SEQ_EMPLOYEE_ID),'Srikanth Pasam','Salesman','','1993-12-01','8967845BD','20000')

INSERT INTO Employee VALUES (CONCAT('OVDE',NEXT VALUE FOR
SEQ_EMPLOYEE_ID),'Deep Singh','Salesman','','1995-05-11','5674389AS','12000')

INSERT INTO Employee VALUES (CONCAT('OVDE',NEXT VALUE FOR
SEQ_EMPLOYEE_ID),'Salman Bhatt','Salesman','','1996-06-23','4563098AS','18000')

INSERT INTO Employee VALUES (CONCAT('OVDE',NEXT VALUE FOR
SEQ_EMPLOYEE_ID),'Salman Khan','Manager','','1986-07-23','4563098HS','80000')



-- Insert Incentive


GO

CREATE SEQUENCE SEQ_INCENTIVE_ID START WITH 1 INCREMENT BY 1;


INSERT INTO Incentive VALUES (NEXT VALUE FOR SEQ_INCENTIVE_ID,'1800','2019-11-
01','OVDE1005')

INSERT INTO Incentive VALUES (NEXT VALUE FOR SEQ_INCENTIVE_ID,'1200','2019-11-
03','OVDE1004')

INSERT INTO Incentive VALUES (NEXT VALUE FOR SEQ_INCENTIVE_ID,'2000','2019-11-
06','OVDE1003')

INSERT INTO Incentive VALUES (NEXT VALUE FOR SEQ_INCENTIVE_ID,'1500','2019-11-
10','OVDE1002')

INSERT INTO Incentive VALUES (NEXT VALUE FOR SEQ_INCENTIVE_ID,'1000','2019-10-
21','OVDE1001')
```

```
INSERT INTO Incentive VALUES (NEXT VALUE FOR SEQ_INCENTIVE_ID,'2000','2019-10-29','OVDE1000')



-- Insert Customer


GO
CREATE SEQUENCE SEQ_CUSTOMER_ID START WITH 1000000 INCREMENT BY 1;


INSERT INTO Customer VALUES (CONCAT('OVDC',NEXT VALUE FOR SEQ_CUSTOMER_ID),'Chaminda Vass','','09411290901')
INSERT INTO Customer VALUES (CONCAT('OVDC',NEXT VALUE FOR SEQ_CUSTOMER_ID),'Virat Kohli','','09188134541')
INSERT INTO Customer VALUES (CONCAT('OVDC',NEXT VALUE FOR SEQ_CUSTOMER_ID),'VVS Laxman','','09190987609')
INSERT INTO Customer VALUES (CONCAT('OVDC',NEXT VALUE FOR SEQ_CUSTOMER_ID),'Sachin Tendulkar','','09112122211')
INSERT INTO Customer VALUES (CONCAT('OVDC',NEXT VALUE FOR SEQ_CUSTOMER_ID),'Wasim Akram','','09013454321')
INSERT INTO Customer VALUES (CONCAT('OVDC',NEXT VALUE FOR SEQ_CUSTOMER_ID),'Mohomad Hafeez','','09023097895')



-- Insert Customer_Order


GO
CREATE SEQUENCE SEQ_CUSTOMER_ORDER_ID START WITH 1 INCREMENT BY 1;
```

```
INSERT INTO Customer_Order VALUES (NEXT VALUE FOR
SEQ_CUSTOMER_ORDER_ID,'2019-09-10','2019-09-
17','430000','OVDC1000001','OVDE1000')

INSERT INTO Customer_Order VALUES (NEXT VALUE FOR
SEQ_CUSTOMER_ORDER_ID,'2019-08-10','2019-08-
20','750000','OVDC1000004','OVDE1002')

INSERT INTO Customer_Order VALUES (NEXT VALUE FOR
SEQ_CUSTOMER_ORDER_ID,'2019-10-03','2019-10-
10','1100000','OVDC1000000','OVDE1003')

INSERT INTO Customer_Order VALUES (NEXT VALUE FOR
SEQ_CUSTOMER_ORDER_ID,'2019-09-20','2019-09-
27','660000','OVDC1000003','OVDE1005')

INSERT INTO Customer_Order VALUES (NEXT VALUE FOR
SEQ_CUSTOMER_ORDER_ID,'2019-09-11','2019-09-
18','84000','OVDC1000002','OVDE1004')

INSERT INTO Customer_Order VALUES (NEXT VALUE FOR
SEQ_CUSTOMER_ORDER_ID,'2019-09-15','2019-09-
25','55000','OVDC1000005','OVDE1001')



-- Insert Manufacturer_Order


GO

CREATE SEQUENCE SEQ_MANUFACTURER_ORDER_ID START WITH 1 INCREMENT BY 1;


INSERT INTO Manufacturer_Order VALUES (NEXT VALUE FOR
SEQ_MANUFACTURER_ORDER_ID,'2019-08-08','1200000','1')

INSERT INTO Manufacturer_Order VALUES (NEXT VALUE FOR
SEQ_MANUFACTURER_ORDER_ID,'2019-09-01','50000','6')

INSERT INTO Manufacturer_Order VALUES (NEXT VALUE FOR
SEQ_MANUFACTURER_ORDER_ID,'2019-08-01','600000','4')

INSERT INTO Manufacturer_Order VALUES (NEXT VALUE FOR
SEQ_MANUFACTURER_ORDER_ID,'2019-09-11','50000','5')
```

```
INSERT INTO Manufacturer_Order VALUES (NEXT VALUE FOR
SEQ_MANUFACTURER_ORDER_ID,'2019-08-25','500000','2')

INSERT INTO Manufacturer_Order VALUES (NEXT VALUE FOR
SEQ_MANUFACTURER_ORDER_ID,'2019-09-08','900000','3')


--Insert Inventory

GO

CREATE SEQUENCE SEQ_INVENTORY_ID START WITH 1 INCREMENT BY 1;


INSERT INTO Inventory VALUES (NEXT VALUE FOR SEQ_INVENTORY_ID,'2019-08-
10','AVAILABLE','1',null)

INSERT INTO Inventory VALUES (NEXT VALUE FOR SEQ_INVENTORY_ID,'2019-09-
02','AVAILABLE','2',null)

INSERT INTO Inventory VALUES (NEXT VALUE FOR SEQ_INVENTORY_ID,'2019-08-
05','SOLD','3','5')

INSERT INTO Inventory VALUES (NEXT VALUE FOR SEQ_INVENTORY_ID,'2019-09-
14','SOLD','4','3')

INSERT INTO Inventory VALUES (NEXT VALUE FOR SEQ_INVENTORY_ID,'2019-08-
29','AVAILABLE','5',null)

INSERT INTO Inventory VALUES (NEXT VALUE FOR SEQ_INVENTORY_ID,'2019-09-
15','SOLD','6','1')



--update employee


UPDATE Employee

SET employee_address = '<?xml version="1.0"?>

<Address>

        <Street>Grafton</Street>

        <Building>Paradise</Building>

        <RoomNo>256</RoomNo>
```

```
        <County>Dublin</County>
        <AreaCode>RR82</AreaCode>
</Address>'
WHERE employee_id = 'OVDE1000';

UPDATE Employee
SET employee_address = '<?xml version="1.0"?>
<Address>
        <Street>North Portland</Street>
        <Building>Portland Villa</Building>
        <RoomNo>10</RoomNo>
        <County>Dublin</County>
        <AreaCode>AR30</AreaCode>
</Address>'
WHERE employee_id = 'OVDE1001';

UPDATE Employee
SET employee_address = '<?xml version="1.0"?>
<Address>
        <Street>Downtown</Street>
        <Building>Downtown Court</Building>
        <RoomNo>119</RoomNo>
        <County>Carlow</County>
        <AreaCode>CW02</AreaCode>
</Address>'
WHERE employee_id = 'OVDE1002';

UPDATE Employee
```

```
SET employee_address = '<?xml version="1.0"?>
<Address>
        <Street>Rose Garden</Street>
        <Building>Old Castle</Building>
        <RoomNo>45</RoomNo>
        <County>Wicklow</County>
        <AreaCode>WK11</AreaCode>
</Address>'
WHERE employee_id = 'OVDE1003';

UPDATE Employee
SET employee_address = '<?xml version="1.0"?>
<Address>
        <Street>River Lower</Street>
        <Building>River Palace</Building>
        <RoomNo>15</RoomNo>
        <County>Dublin</County>
        <AreaCode>DB22</AreaCode>
</Address>'
WHERE employee_id = 'OVDE1004';

UPDATE Employee
SET employee_address = '<?xml version="1.0"?>
<Address>
        <Street>Old Garden</Street>
        <Building>Garden Tower</Building>
        <RoomNo>04</RoomNo>
        <County>Athlon</County>
```

```
        <AreaCode>AL77</AreaCode>
</Address>'
WHERE employee_id = 'OVDE1005';

UPDATE Employee
SET employee_address = '<?xml version="1.0"?>
<Address>
        <Street>Upper Town</Street>
        <Building>Uptown Court</Building>
        <RoomNo>02</RoomNo>
        <County>Carlow</County>
        <AreaCode>CW11</AreaCode>
</Address>'
WHERE employee_id = 'OVDE1006';

-- update customer

UPDATE Customer
SET customer_address = '<?xml version="1.0"?>
<Address>
        <Street>Balfe</Street>
        <Building>Paradise</Building>
        <RoomNo>56</RoomNo>
        <County>Dublin</County>
        <AreaCode>RR77</AreaCode>
</Address>'
WHERE customer_id = 'OVDC1000000';
```

```
UPDATE Customer
SET customer_address = '<?xml version="1.0"?>
<Address>
        <Street>Riverland</Street>
        <Building>River Villa</Building>
        <RoomNo>1</RoomNo>
        <County>Dublin</County>
        <AreaCode>AR31</AreaCode>
</Address>'
WHERE customer_id = 'OVDC1000001';

UPDATE Customer
SET customer_address = '<?xml version="1.0"?>
<Address>
        <Street>Townhall Street</Street>
        <Building>Street Paradise</Building>
        <RoomNo>11</RoomNo>
        <County>Carlow</County>
        <AreaCode>CW05</AreaCode>
</Address>'
WHERE customer_id = 'OVDC1000002';

UPDATE Customer
SET customer_address = '<?xml version="1.0"?>
<Address>
        <Street>Folwer Garden</Street>
        <Building>Castle Building</Building>
        <RoomNo>48</RoomNo>
```

```
        <County>Wicklow</County>
        <AreaCode>WK15</AreaCode>
</Address>'
WHERE customer_id = 'OVDC1000003';

UPDATE Customer
SET customer_address = '<?xml version="1.0"?>
<Address>
        <Street>River Liffy Rd.</Street>
        <Building>River Palace</Building>
        <RoomNo>15</RoomNo>
        <County>Dublin</County>
        <AreaCode>DB22</AreaCode>
</Address>'
WHERE customer_id = 'OVDC1000004';

UPDATE Customer
SET customer_address = '<?xml version="1.0"?>
<Address>
        <Street>Old Garden</Street>
        <Building>Garden Court</Building>
        <RoomNo>06</RoomNo>
        <County>Athlon</County>
        <AreaCode>AL72</AreaCode>
</Address>'
WHERE customer_id = 'OVDC1000005';

-- Car Variant
```

```
UPDATE Car_Model
SET car_model_variant = '<?xml version="1.0"?>
<Variant>
<Color>Red</Color>
<EngineNo>F488</EngineNo>
<Fuel>Diesel</Fuel>
<Power>301</Power>
<ZeroToSixty>4.3</ZeroToSixty>
<SeatingCapacity>4</SeatingCapacity>
<Airbags>Yes</Airbags>
</Variant>'
WHERE car_model_id = 1;

UPDATE Car_Model
SET car_model_variant = '<?xml version="1.0"?>
<Variant>
<Color>White</Color>
<EngineNo>M100</EngineNo>
<Fuel>Diesel</Fuel>
<Power>300</Power>
<ZeroToSixty>4.9</ZeroToSixty>
<SeatingCapacity>6</SeatingCapacity>
<Airbags>Yes</Airbags>
</Variant>'
WHERE car_model_id = 2;

UPDATE Car_Model
```

```
SET car_model_variant = '<?xml version="1.0"?>
<Variant>
<Color>Blue</Color>
<EngineNo>S190</EngineNo>
<Fuel>Petrol</Fuel>
<Power>290</Power>
<ZeroToSixty>5.3</ZeroToSixty>
<SeatingCapacity>2</SeatingCapacity>
<Airbags>Yes</Airbags>
</Variant>'
WHERE car_model_id = 3;

UPDATE Car_Model
SET car_model_variant = '<?xml version="1.0"?>
<Variant>
<Color>Ash</Color>
<EngineNo>FX100</EngineNo>
<Fuel>Diesel</Fuel>
<Power>295</Power>
<ZeroToSixty>5.6</ZeroToSixty>
<SeatingCapacity>4</SeatingCapacity>
<Airbags>Yes</Airbags>
</Variant>'
WHERE car_model_id = 4;

UPDATE Car_Model
SET car_model_variant = '<?xml version="1.0"?>
<Variant>
```

```
<Color>Gray</Color>
<EngineNo>F488</EngineNo>
<Fuel>Petrol</Fuel>
<Power>278</Power>
<ZeroToSixty>6.5</ZeroToSixty>
<SeatingCapacity>4</SeatingCapacity>
<Airbags>No</Airbags>
</Variant>'
WHERE car_model_id = 5;

UPDATE Car_Model
SET car_model_variant = '<?xml version="1.0"?>
<Variant>
<Color>White</Color>
<EngineNo>MX23</EngineNo>
<Fuel>Petrol</Fuel>
<Power>201</Power>
<ZeroToSixty>7.6</ZeroToSixty>
<SeatingCapacity>4</SeatingCapacity>
<Airbags>Yes</Airbags>
</Variant>'
WHERE car_model_id = 6;

------------------------------

UPDATE Car_Model
SET car_model_variant = '<?xml version="1.0"?>
<Variant>
```

```
<Color>Red</Color>
<EngineNo>RX8</EngineNo>
<Fuel>Diesel</Fuel>
<Power>301</Power>
<ZeroToSixty>4.3</ZeroToSixty>
<SeatingCapacity>2</SeatingCapacity>
<Airbags>Yes</Airbags>
</Variant>'
WHERE car_model_id = 7;

UPDATE Car_Model
SET car_model_variant = '<?xml version="1.0"?>
<Variant>
<Color>White</Color>
<EngineNo>M123</EngineNo>
<Fuel>Diesel</Fuel>
<Power>301</Power>
<ZeroToSixty>4.6</ZeroToSixty>
<SeatingCapacity>6</SeatingCapacity>
<Airbags>Yes</Airbags>
</Variant>'
WHERE car_model_id = 8;

UPDATE Car_Model
SET car_model_variant = '<?xml version="1.0"?>
<Variant>
<Color>White</Color>
<EngineNo>F488</EngineNo>
```

```
<Fuel>Petrol</Fuel>
<Power>290</Power>
<ZeroToSixty>4.9</ZeroToSixty>
<SeatingCapacity>4</SeatingCapacity>
<Airbags>Yes</Airbags>
</Variant>'
WHERE car_model_id = 9;

UPDATE Car_Model
SET car_model_variant = '<?xml version="1.0"?>
<Variant>
<Color>White</Color>
<EngineNo>M123</EngineNo>
<Fuel>Diesel</Fuel>
<Power>276</Power>
<ZeroToSixty>5.8</ZeroToSixty>
<SeatingCapacity>4</SeatingCapacity>
<Airbags>Yes</Airbags>
</Variant>'
WHERE car_model_id = 10;

UPDATE Car_Model
SET car_model_variant = '<?xml version="1.0"?>
<Variant>
<Color>Green</Color>
<EngineNo>R888</EngineNo>
<Fuel>Diesel</Fuel>
<Power>175</Power>
```

```
<ZeroToSixty>8.3</ZeroToSixty>
<SeatingCapacity>4</SeatingCapacity>
<Airbags>No</Airbags>
</Variant>'
WHERE car_model_id = 11;

UPDATE Car_Model
SET car_model_variant = '<?xml version="1.0"?>
<Variant>
<Color>Blue</Color>
<EngineNo>M192</EngineNo>
<Fuel>Petrol</Fuel>
<Power>205</Power>
<ZeroToSixty>5.8</ZeroToSixty>
<SeatingCapacity>4</SeatingCapacity>
<Airbags>Yes</Airbags>
</Variant>'
WHERE car_model_id = 12;
```

Appendix 3

```
--View 1


CREATE VIEW view_order_taken

AS

SELECT e.employee_id, e.employee_name,e.employee_dob,count(co.customer_order_id)
as ordertaken

FROM Employee e

INNER JOIN Customer_Order co

ON e.employee_id = co.employee_id

group by e.employee_id, e.employee_name,e.employee_dob



--View 2


CREATE VIEW customer_order_track

As

SELECT
c.customer_id,cm.car_model_name,m.manufacturer_name,co.customer_order_selling_p
rice,mo.manufacturer_order_price,

 (co.customer_order_selling_price - mo.manufacturer_order_price) as Profit_Earned

from Customer c

inner join Customer_Order co

ON c.customer_id = co.customer_id

inner join Inventory i

ON co.customer_order_id=i.customer_order_id

inner join manufacturer_order mo

ON i.manufacturer_order_id= mo.manufacturer_order_id
```

```
inner join car_model cm
ON cm.car_model_id= mo.car_model_id
inner join manufacturer m
ON m.manufacturer_id=cm.manufacturer_id
```

Appendix 4

```
-- 01- instead


CREATE TRIGGER triggerAddCustomerOrder
ON Customer_Order
INSTEAD OF INSERT
AS
BEGIN
DECLARE @order_time date
DECLARE @customer_order_delivery_date date
DECLARE @customer_order_selling_price int
DECLARE @customer_order_date date
DECLARE @customer_id varchar(50)
DECLARE @employee_id varchar(50)
DECLARE @cur_val int = NEXT VALUE FOR SEQ_CUSTOMER_ORDER_ID


SELECT @order_time = GETDATE()
SELECT @customer_order_date =i.customer_order_date FROM inserted i
SELECT @customer_order_delivery_date=i.customer_order_delivery_date FROM
inserted i
SELECT @customer_order_selling_price =i.customer_order_selling_price FROM inserted i
SELECT @customer_id = i.customer_id FROM inserted i
SELECT @employee_id = i.employee_id FROM inserted i
IF EXISTS (SELECT customer_order_id FROM Customer_Order WHERE customer_order_id
= @cur_val)
        BEGIN
```

```
            PRINT('Error, Order already exists')
        END
ELSE
        BEGIN
            INSERT INTO Customer_Order VALUES
(@cur_val,@customer_order_date,@customer_order_delivery_date,@customer_order_
selling_price,@customer_id,@employee_id)
            PRINT('Order Added')
            INSERT INTO dbo.Customer_Order_Log VALUES (@cur_val, @order_time,
@customer_id, @employee_id,@customer_order_selling_price)
            PRINT('Order Added to Log Table')
        END
END

INSERT INTO Customer_Order
(customer_order_date, customer_order_delivery_date, customer_order_selling_price,
customer_id, employee_id)
VALUES ('2019-09-16','2019-09-27','85000','OVDC1000005','OVDE1003')

-- 02 After


CREATE TRIGGER TriggerAddIncentive
ON Incentive
AFTER INSERT
AS
BEGIN
DECLARE @incentive_id int
DECLARE @incentive_date date
DECLARE @employee_id varchar(50)
```

```
DECLARE @salary int


SELECT @incentive_id = i.incentive_id FROM inserted i
SELECT @incentive_date = i.incentive_date FROM inserted i
SELECT @employee_id = i.employee_id FROM inserted i

SELECT @salary = (SELECT employee_salary FROM Employee WHERE employee_id =
@employee_id)
PRINT(@salary)

UPDATE Incentive
SET incentive_amount = dbo.fn_Calc_Incentive(@salary)
WHERE incentive_id = @incentive_id
PRINT ('Incentive Added')
END


INSERT INTO Incentive (incentive_id,incentive_date, employee_id)
VALUES (NEXT VALUE FOR SEQ_INCENTIVE_ID,'2019-10-29','OVDE1003')
GO
```