TASK 1 – SQL QUERIES

1. Provide information about the global sales amount (money), number of orders and volume (items sold) of the AdventureWorks business.
    a. An example of the query result is shown in Table 2.1:

Table 1.1. Result structure for task 1.1

Sales Amount	Volume	Number of orders
…	…	…

2. Provide information about the sales amount, volume, and number of orders in individual years of operation of the business.
    a. An example of the query result is shown in Table 2.2:

Table 1.2. Result excerpt for task 1.2

Year	Sales Amount	Volume	Number of orders
…	…	…	…

3. Prepare a SQL query that provides top 5 customers with the highest number of orders, try using the customer's name (it might be tricky).
    a. An example of the query result is shown in Table 2.3.:

Table 1.3. Result excerpt for task 1.3

CustomerID	Last name, name	Number of orders
…	…	…

4. Prepare a SQL query that provides the names of all individual customers with the total sum of purchases (use SalesOrderHeader.SubTotal) greater than 1500USD – sorted (descending) by the total sales amount.
    a. An example of the query result is shown in Table 2.4.:

Table 1.4. Result excerpt for task 1.4

CustomerID	Last name, name	SalesAmount
…	…	…

5. Prepare a query that provides information about average price, total sales amount, and total volume in individual product categories of the AdventureWorks business.
    a. An example of the query result is shown in Table 2.5.:

Table 1.5. Result excerpt for task 1.5

CategoryID	Category name	Average price	Total Sales Amount	Total Volume
…	…	…	…	…

6. Display all subcategories which average price is higher than the average price of all categories.
    a. An example of the query result is shown in Table 2.6.:

Table 1.6. Result excerpt for task 1.6

SubcategoryID	Subcategory Name	Average price	Average price (over all categories)
…	…	…	…

7. Select sales territory (name) with sales in May 2013 higher than the average monthly sales per sales territory.
    a. An example of the query result is shown in Table 2.7.:

Table 1.7. Result excerpt for task 1.7

SalesTerritoryID	Sales Territory Name	Sales (May 2013)	Average monthly sales (per territory)
…	…	…	…

8. Create a list of sales territories (ids are enough) with an average number of orders (both real value and the largest integer less than the value) made by customers who have more than 10 orders in general (use CTE)
    a. An example of the query result is shown in Table 2.8.:

Table 1.8. Result excerpt for task 1.8

TerritoryID	Average number of orders	Average number of orders (INT)
…	…	…

9. (*) Show monthly sales amount by each sales territory in year 2013 and calculate the difference with the previous month (use 0 for 12/2012) to identify trends.
    a. An example of the query result is shown in Table 2.4.:

Table 1.9. Result excerpt for task 1.9

TerritoryID	Sales Territory Name	Mnt Sales Amt	Diff to prev
…	…	…	…

10. (**) Propose a query (formulate the question and present SQL here) for the entire group to solve at the beginning of the next class. Try not to use more than 4 joins. If no other group manages to solve it during the class, you will receive a bonus point.

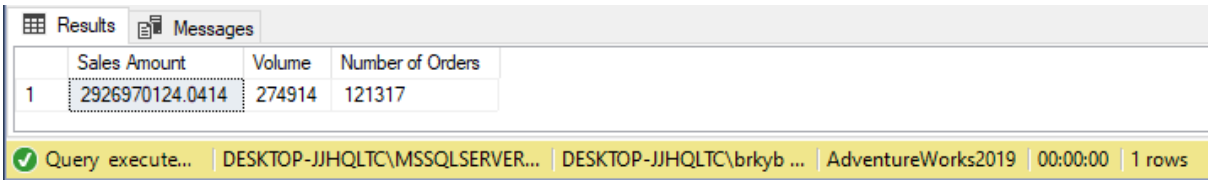
All results should be additionally stored in a single text .sql file. After completing the task, double check your approach and please upload the file to ePortal. Note that the task allows for submission of multiple files.

TASK 1 - SOLUTIONS:

Use this section to provide your solutions, please remember to present an expert from the obtained results (several records, along with headers) and basic metadata (like number of rows), try to explain your approach:

1

```
SELECT
    SUM(Sales.SalesOrderHeader.TotalDue) AS [Sales Amount],
    SUM(Sales.SalesOrderDetail.OrderQty) AS [Volume],
    COUNT(Sales.SalesOrderHeader.SalesOrderID) AS [Number of Orders]
FROM
    Sales.SalesOrderHeader
JOIN
    Sales.SalesOrderDetail ON Sales.SalesOrderHeader.SalesOrderID =
Sales.SalesOrderDetail.SalesOrderID;
```



Below 2 aggregate functions(SUM(), COUNT()) are used in the query to perform calculations on set of values:

- *SUM (SalesOrderHeader.TotalDue) -> Calculates the total sales amount.*
- *SUM (SalesOrderDetail.OrderQty) -> Calculates the total quantity of items sold.*
- *COUNT(SalesOrderHeader.SalesOrderID) -> Counts the number of unique sales orders.*

JOIN Clause: is used in the query to combine rows from two tables based on the related attributes. Inner join is used because that the objective is to retrieve info about Sales orders along with their associated details. For instance, using left join would include all rows from the SalesOrderHeader even if there are no matching rows with SalesOrderDetail. In same Logic, Right join would include all the rows from SalesOrderDetail although there is no matching rows in SOH.

In this query I am intent to join the SalesOrderHeader table with the SalesOrderDetail table based on the common attributes of both tables which are: SalesOrderID. So be sure that each row in SOH & SOD matched with the corresponding row in other table where the values are same(SalesOrderID).

Overall, The primary goal of this task is to gather key Sales for data. this task demonstrates extracting valuable information from raw transactional data. The query effectively, combines data from multiple tables and uses aggregate functions(SUM, COUNT) to provide a summary of a Sales-related data overview ensuring the comprehensive information about the sales performance.

2

...

```
SELECT
    YEAR(Sales.SalesOrderHeader.OrderDate) AS "Year",
    SUM(SalesOrderDetail.UnitPrice * SalesOrderDetail.OrderQty) AS "Sales Amount",
    SUM(SalesOrderDetail.OrderQty) AS "Volume",
    COUNT(DISTINCT Sales.SalesOrderHeader.SalesOrderID) AS "Number of Orders"
FROM
    Sales.SalesOrderHeader
JOIN
    Sales.SalesOrderDetail  ON Sales.SalesOrderHeader.SalesOrderID =
SalesOrderDetail.SalesOrderID
GROUP BY
    YEAR(Sales.SalesOrderHeader.OrderDate)
ORDER BY
    YEAR ASC;
```

Results | Messages

	Year	Sales Amount	Volume	Number of Orders
1	2011	12646112.1607	12888	1607
2	2012	33710896.9379	68579	3915
3	2013	43922050.7113	131788	14182
4	2014	20094829.5035	61659	11761

Below 2 aggregate functions(SUM(), COUNT()) are used in the query to perform calculations on set of values:

- *SUM(SalesOrderDetail.UnitPrice * SalesOrderDetail.OrderQty) -> Calculates the total sales amount by multiplying the unit price of each item with its quantity and sums up the values. It is named as "Sales Amount".*
- *SUM(SalesOrderDetail.OrderQty -> Calculates the total quantity of items sold and assigns it an alias "Volume".*
- *COUNT(DISTINCT Sales.SalesOrderHeader.SalesOrderID) -> Counts the number of unique sales orders and assigns it an alias "Number of Orders".*

JOIN Clause: Combines rows from the header and detail tables based on the common attribute SalesOrderID. Sales.SalesOrderHeader.SalesOrderID = SalesOrderDetail.SalesOrderID: so that we are ensuring that each row in the header table matches with the corresponding row in the detail table where the sales order IDs are the same. For instance, LEFT join: all rows from the SalesOrderHeader table would be included in the result set, regardless of whether there is a matching row in the SalesOrderDetail table or not. If there is no matching row in the SalesOrderDetail table for a row in the SalesOrderHeader table, the columns from the SalesOrderDetail table would contain NULL values. RIGHT JOIN includes all rows from SalesOrderDetail table, even without matches in SalesOrderHeader. Unmatched SalesOrderHeader columns would have NULL values.

Overall ,The task focuses on summarizing sales data annually, extracting insights from raw transactional data. It combines tables, employing aggregate functions like SUM and COUNT to analyse sales performance over different years. Grouping data by year enables trend analysis for informed decision-making.

3

…

```
SELECT TOP 5 SOH.CustomerID,
    CONCAT(p.LastName, ', ', p.FirstName) AS [Last name, name],
      COUNT(SOH.SalesOrderID) AS [Number of orders]
FROM Sales.SalesOrderHeader AS SOH
JOIN Person.Person AS p ON SOH.CustomerID = p.BusinessEntityID
GROUP BY SOH.CustomerID, p.LastName, p.FirstName
ORDER BY [Number of orders] DESC;
```

Results | Messages

	CustomerID	Last name, name	Number of orders
1	11176	Miller, Morgan	28
2	11091	Taylor, Jennifer	28
3	11287	Carlson, Ruben	27
4	11185	Jackson, Morgan	27
5	11711	Jones, Brianna	27

SELECT TOP 5: This part limits the results to only the top 5 customers based on their order count.

Alias(): SOH and SOD improves the readability of the query and prevents retyping of table names. Desc: decreasing number (from the highest to lowest)

Below 2 aggregate functions(SUM(), COUNT()) are used in the query to perform calculations on set of values:

- *CONCAT(p.LastName, ', ', p.FirstName) AS [Last name, name] -> Combines the customer's last name, first name, and a comma with a space to create a nicely formatted "Last name, First name" display.*
- *COUNT(SOH.SalesOrderID) AS [Number of orders] -> Calculates the total number of orders associated with each customer.*

JOIN Clause: The query uses an INNER JOIN. This means it will only return rows where there's a match between the CustomerID in the SalesOrderHeader table and the BusinessEntityID in the Person table. In this query, the Sales.SalesOrderHeader and Person.Person tables are joined. Because;

Sales.SalesOrderHeader: This table contains details about orders, including the CustomerID. However, it likely doesn't store the customer's full name in a readily usable format.

Person.Person: This table stores customer information, including their first and last names. However, it doesn't directly contain information about their order history.

The JOIN bridges the gap, allowing you to link order data in Sales.SalesOrderHeader to the matching customer name information in Person.Person.

GROUP BY SOH.CustomerID, p.LastName, p.FirstName: Groups the results by CustomerID and the concatenated last name, first name to aggregate the data for each customer.

ORDER BY [Number of orders] DESC: Orders the results in descending order based on the number of orders, ensuring that the customers with the highest number of orders appear first.

Tricky part to define the Last Name and First name has done through associated Person.person table.

Overall, The query finds the top 5 customers based on their order count. It includes their ID, full name, and order count. By linking sales and customer tables, it summarizes sales behavior. Using COUNT and CONCAT functions, it analyzes orders per customer. Grouping by customer ID and sorting by order count helps identify active customers and behaviour.

4

...

```
SELECT c.CustomerID,
    CONCAT(p.LastName, ', ', p.FirstName) AS [Last name, name],
    SUM(soh.SubTotal) AS SalesAmount
FROM Sales.SalesOrderHeader AS soh
JOIN Sales.Customer AS c ON soh.CustomerID = c.CustomerID
JOIN Person.Person AS p ON soh.CustomerID = p.BusinessEntityID
WHERE p.PersonType = 'IN'
GROUP BY c.CustomerID, p.LastName, p.FirstName
HAVING SUM(soh.SubTotal) > 1500
ORDER BY SalesAmount DESC;
```

Results | Messages

	CustomerID	Last name, name	SalesAmount
1	12301	Clark, Hannah	13295.38
2	12132	Jones, Taylor	13294.27
3	12308	Lee, Hannah	13269.27
4	12131	Chander, Kelli	13265.99
5	12300	She, Colleen	13242.70
6	12321	Miller, Madison	13215.65
7	12124	Smith, Taylor	13195.64

Query executed successf... | DESKTOP-JJHQLTC\MSSQLSERVER... | DESKTOP-JJHQLTC\brkyb ... | AdventureWorks2019 | 00:00:00 | 4,306 rows

Query retrieves the names of individual customers along with the total sum of their purchases, where the total sum exceeds $1500.

JOIN Clause:

JOIN between ProductSubcategory and Product Tables:This joins pairs each product in Product table with the corresponsing category in ProductSubCategory table. The pairing is done based on the ProductSubcategoryID attribute in both Product and ProductSubCategory. Through this way each product is associated with its SubCategory and provides access to info of product.

JOIN between ProductCategory and ProductSubcategory Tables: This joins pairs eachSubCategory in the ProductSubCategory table with its corresponding category in the ProductCategor table. The pairing is dne based on the ProductCategoryID attributes in both the ProductSubCategor and ProductCategory table.

This returns only the matching records, ensuring that rows without sales details or category/subcategory information are not included. Therefore, by using INNER JOINs, each product is linked to its sales details and category/subcategory information.

Overall, This query continues our exploration of customer behaviour by identifying individual customers with high purchase amounts. By filtering and summarizing sales data, we pinpoint customers making significant purchases, which is helpful for targeted marketing. It links sales order details with customer information, calculates the total sales amount per customer, and filters out customers whose total purchases exceed $1500.

5

...

```
SELECT
    PC.ProductCategoryID AS CategoryID,
    PC.Name AS [Category name],
    AVG(P.ListPrice) AS [Average price],
    SUM(SOD.LineTotal) AS [Total Sales Amount],
    SUM(SOD.OrderQty) AS [Total Volume]
FROM
    Production.Product AS P
JOIN
    Sales.SalesOrderDetail AS SOD ON P.ProductID = SOD.ProductID
JOIN
    Production.ProductSubcategory AS PS ON P.ProductSubcategoryID =
    PS.ProductSubcategoryID
JOIN
    Production.ProductCategory AS PC ON PS.ProductCategoryID = PC.ProductCategoryID
GROUP BY
    PC.ProductCategoryID, PC.Name
ORDER BY
    PC.ProductCategoryID;
```

Results | Messages

	CategoryID	Category name	Average price	Total Sales Amount	Total Volume
1	1	Bikes	1672.3917	94651172.704731	90268
2	2	Components	433.8962	11802593.286430	49044
3	3	Clothing	43.2621	2120542.524801	73670
4	4	Accessories	21.5881	1272072.883926	61932

The query fetches data related to product categories, including category ID, name, average price, total sales amount, and total volume.

JOIN Clause: The query uses INNER JOINs to combine data from multiple tables. It connects the Product table with SalesOrderDetail using ProductID to access sales info. Then, it links ProductSubcategory to Product via ProductSubcategoryID for associating products with their subcategories. Lastly, it connects ProductCategory to ProductSubcategory using ProductCategoryID to associate subcategories with their main categories, allowing for organized sales analysis.

GROUP BY: Data is grouped by *ProductCategoryID and Name to aggregate information at the category level.*

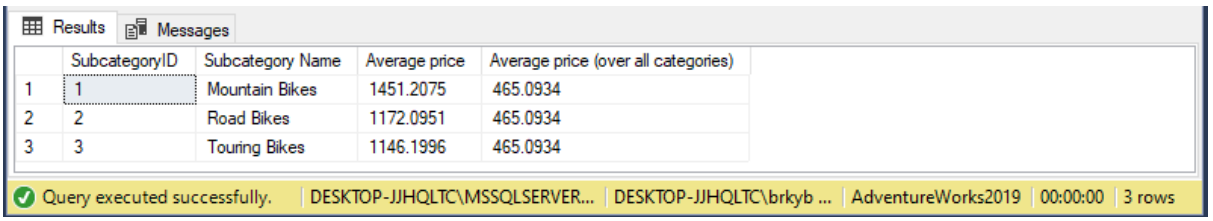
ORDER BY: Results are sorted by ProductCategoryID.

The query examines product categories by analyzing sales data. It connects product details with sales records and groups products into categories. By calculating average price, total sales amount, and volume for each category, we understand category performance better. This focus on product categories helps us see which ones are doing well based on sales data.

6

...

```
SELECT
    psc.ProductSubcategoryID AS SubcategoryID,
    psc.Name AS [Subcategory Name],
    AVG(sod.UnitPrice) AS [Average price],
    (SELECT AVG(sod.UnitPrice) FROM Sales.SalesOrderDetail AS sod) AS [Average price
(over all categories)]
FROM
    Sales.SalesOrderDetail AS sod
JOIN
    Production.Product AS p ON sod.ProductID = p.ProductID
JOIN
    Production.ProductSubcategory AS psc ON p.ProductSubcategoryID =
psc.ProductSubcategoryID
GROUP BY
    psc.ProductSubcategoryID, psc.Name
HAVING
    AVG(sod.UnitPrice) > (SELECT AVG(sod.UnitPrice) FROM Sales.SalesOrderDetail AS
sod);
```



Results | Messages

	SubcategoryID	Subcategory Name	Average price	Average price (over all categories)
1	1	Mountain Bikes	1451.2075	465.0934
2	2	Road Bikes	1172.0951	465.0934
3	3	Touring Bikes	1146.1996	465.0934

Query executed successfully. | DESKTOP-JJHQLTC\MSSQLSERVER... | DESKTOP-JJHQLTC\brkyb ... | AdventureWorks2019 | 00:00:00 | 3 rows

These joins link: sales data with product details and then extend the connection to include product subcategory information. By matching IDs between tables, it combines sales, product, and subcategory data for comprehensive analysis.
GROUP BY organizes data by product subcategory for detailed analysis, separating it into segments for comparison. This helps understand sales performance across different product types.

HAVING filters subcategories with prices higher than the overall average. It selects only those meeting the condition, aiding in identifying higher-priced categories. It's applied after grouping to analyze aggregated data. This helps pinpoint subcategories with above-average prices.

The query explores product subcategories, comparing their average prices to the overall average. It filters subcategories with prices above the overall average. INNER JOINs link sales data to product and subcategory info, enabling focused analysis. This shift highlights subcategory performance relative to the market, informing pricing and marketing strategies.

7

...

```
WITH AverageMonthlySales AS (
    SELECT
        st.TerritoryID,
        AVG(sod.LineTotal) AS AvgMonthlySales
    FROM
        Sales.SalesOrderHeader AS soh
    JOIN
        Sales.SalesOrderDetail AS sod ON soh.SalesOrderID = sod.SalesOrderID
    JOIN
        Sales.SalesPerson AS sp ON soh.SalesPersonID = sp.BusinessEntityID
    JOIN
        Sales.SalesTerritory AS st ON sp.TerritoryID = st.TerritoryID
    WHERE
        YEAR(soh.OrderDate) = 2013
    GROUP BY
        st.TerritoryID
)
SELECT
    st.TerritoryID AS SalesTerritoryID,
    st.Name AS [Sales Territory Name],
    sum(CASE
            WHEN MONTH(soh.OrderDate) = 5 AND YEAR(soh.OrderDate) = 2013 THEN
sod.LineTotal
            ELSE 0
        END) AS [Sales (May 2013)],
    ams.AvgMonthlySales AS [Average monthly sales (per territory)]
FROM
    Sales.SalesOrderHeader AS soh
JOIN
    Sales.SalesOrderDetail AS sod ON soh.SalesOrderID = sod.SalesOrderID
JOIN
    Sales.SalesPerson AS sp ON soh.SalesPersonID = sp.BusinessEntityID
JOIN
    Sales.SalesTerritory AS st ON sp.TerritoryID = st.TerritoryID
JOIN
    AverageMonthlySales AS ams ON st.TerritoryID = ams.TerritoryID
WHERE
    MONTH(soh.OrderDate) = 5 AND YEAR(soh.OrderDate) = 2013
GROUP BY
    st.TerritoryID,
    st.Name,
    ams.AvgMonthlySales
HAVING
    sum(CASE
```

```
            WHEN MONTH(soh.OrderDate) = 5 AND YEAR(soh.OrderDate) = 2013 THEN
sod.LineTotal
            ELSE 0
        END) > ams.AvgMonthlySales;
```

Results | Messages

	SalesTerritoryID	Sales Territory Name	Sales (May 2013)	Average monthly sales (per territory)
1	9	Australia	49824.713775	965.421620
2	10	United Kingdom	484386.316437	1228.257258
3	4	Southwest	662522.894446	1288.117139
4	8	Germany	178101.545566	1018.363679
5	5	Southeast	170875.456476	1006.477820
6	6	Canada	169144.418343	1072.252874
7	3	Central	262105.729335	1058.845469
8	7	France	55465.407616	1333.708875
9	1	Northwest	382992.737350	1322.786666
10	2	Northeast	248292.291335	1300.285448

Query executed successfully. | DESKTOP-JJHQLTC\MSSQLSERVER... | DESKTOP-JJHQLTC\brkyb ... | AdventureWorks2019 | 00:00:00 | 10 rows

The query calculates and compares the sales performance of different sales territories in May 2013 against their average monthly sales.

The query retrieves sales data for May 2013 from the SalesOrderHeader and SalesOrderDetail tables. It joins these tables with SalesPerson and SalesTerritory tables to get the sales territory information. Additionally, it joins with the CTE AverageMonthlySales to access the average monthly sales data for each territory.

Common Table Expression (CTE) - AverageMonthlySales: figures out how much money each sales territory made each month in the year 2013.

- *Sales Orders: Where the details of each sale are stored.*
- *Salespeople: Who made the sale.*
- *Sales Territories: The region each salesperson covers.*

WHERE: It filters the data to include only records for May 2013.

GROUP BY: It groups the results by TerritoryID, Territory Name, and average monthly sales.

HAVING: It further filters the grouped data to include only those territories where the sales for May 2013 exceed the average monthly sales for that territory.

Overall, We looked at how sales territories performed in May 2013. By comparing May 2013 sales to the average monthly sales per territory, we found areas with outstanding sales performance. This helps in making strategic decisions and allocating resources effectively.

8

```
    WITH CustomersWithMoreThan10Orders AS (
SELECT
    soh.CustomerID,
    COUNT(soh.SalesOrderID) AS NumOrders
FROM
    Sales.SalesOrderHeader AS soh
GROUP BY
    soh.CustomerID
HAVING
```

```
        COUNT(soh.SalesOrderID) > 10
)

SELECT
    st.TerritoryID,
    AVG(CAST(cw.NumOrders AS FLOAT)) AS [Average number of orders],
    AVG(FLOOR(cw.NumOrders)) AS [Average number of orders (INT)]
FROM
    Sales.SalesTerritory AS st
INNER JOIN
    Sales.SalesPerson AS sp ON st.TerritoryID = sp.TerritoryID
INNER JOIN
    Sales.SalesOrderHeader AS soh ON sp.BusinessEntityID = soh.SalesPersonID
INNER JOIN
    CustomersWithMoreThan10Orders AS cw ON soh.CustomerID = cw.CustomerID
GROUP BY
    st.TerritoryID;
```

Results | Messages

	TerritoryID	Average number of orders	Average number of orders (INT)
1	1	12	12
2	2	11.921875	11
3	3	11.9160305343511	11
4	4	11.8	11
5	5	11.7441860465116	11
6	6	11.9619047619048	11
7	10	11.9255319148936	11

Query executed successfully.

This query first identifies customers who have made more than 10 orders in total. It uses a Common Table Expression (CTE) named "CustomersWithMoreThan10Orders" to achieve this. Then, it calculates the average number of orders for each sales territory, considering only those customers who meet the criteria specified in the CTE.

The use of a CTE simplifies the query structure and improves readability by separating the logic for identifying customers with more than 10 orders

By filtering customers based on their total number of orders first, we ensure that we're focusing only on active and frequent customers for the analysis.
This query helps us grasp how often top customers order in each sales area, revealing where customer interest is strongest. It's valuable for spotting areas for focused marketing or sales efforts.

9*

…

10**

Create a list of orders placed in the first year of order registration (ID, Year, Order Amount).

```
SELECT DISTINCT [SalesOrderID]  AS "Identifier" ,YEAR([OrderDate]) AS "Year",
ROUND([TotalDue]*1,2)  AS "Amount"
FROM [Sales].[SalesOrderHeader]
WHERE  [SalesOrderID] BETWEEN 45266 AND 45269
ORDER BY Year;
```

Results | Messages

	Identifier	Year	Amount
1	45266	2012	27605.63
2	45267	2012	3899.68
3	45268	2012	944.62
4	45269	2012	2280.14

…

COMMENTS:

The queries that we focused on looks at sales data to give useful insights. They cover things like total sales and analysis aiming to customer behaviour and performance of the product, customer habits, product categories, and sales territory success. We used functions, joins, and groupings to summarize sales info. Overall, they help businesses make better decisions about sales."

- *There is sort of attributes and tables that I was struggling to understand and which table or attribute to use to define the tables.*

…

TASK 2 – DATA MODELLING

Please analyse the conceptual data model of "Orders" (Fig. 1.), which is incomplete, but the classes and the relationships between them may represent a part of the reality under consideration.

Consider a straightforward situation (intuitive understanding) of handling orders made by customers, for set of products, and handled by different shops. Please assume, that a customer can repeatedly shop in the same store, and any customer can shop in any store. Each purchase is made by the customer in the store on a specific day and time. A store must offer at least one product and each store can individually propose the price and quantity of the offered product. Finally, the same product (type) can be offered by multiple stores.

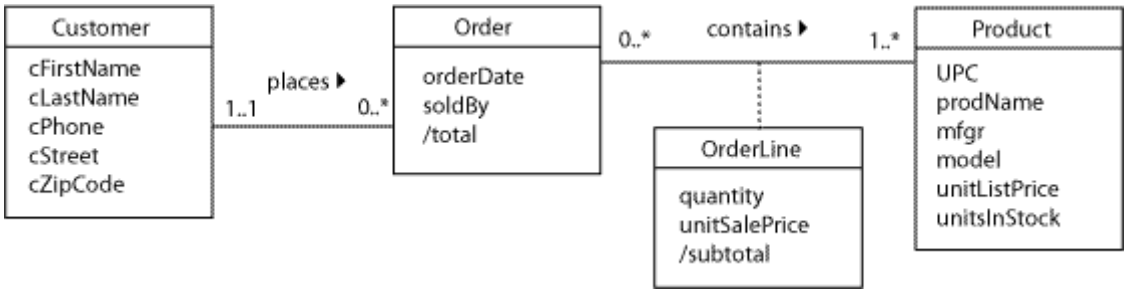


Figure 1 Conceptual data Model, UML