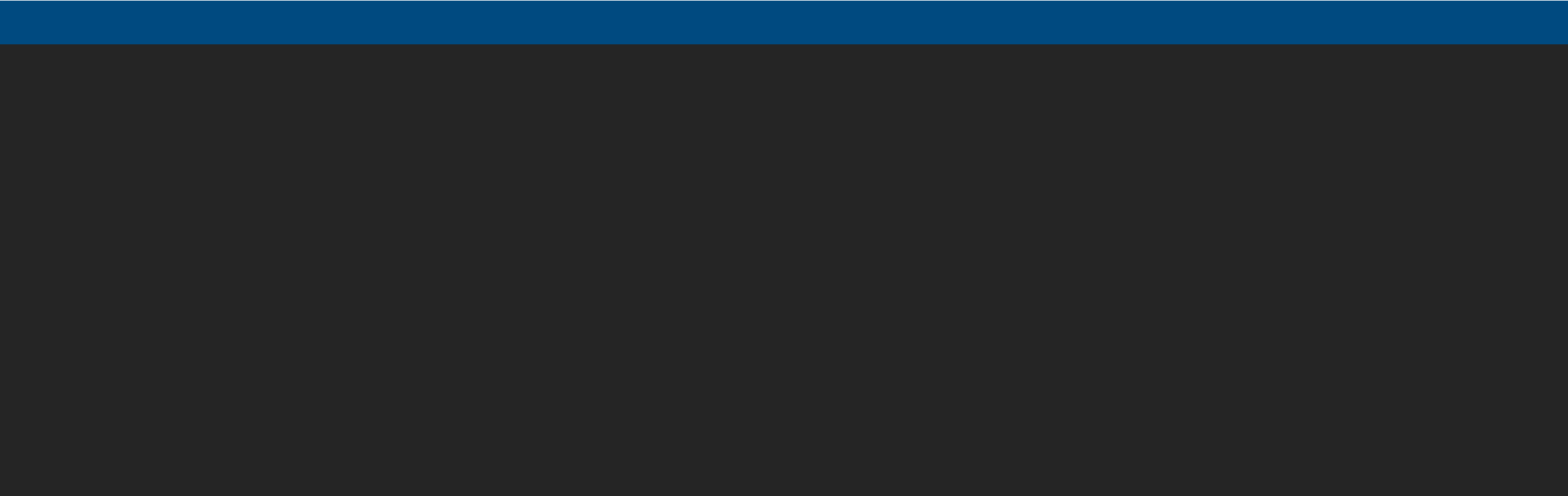
***JTI POLINEMA***



**Jurusan Teknologi Informasi**

**POLITEKNIK NEGERI MALANG**



**Week**

**6**

**SQL SERVER**

–

**TABLE EXPRESSION**



**JOBSHEET**

**PRAKTIKUM BASIS DATA LANJUT**

**Topics**

Information

Technology

Department,

Malang

State

Polytechnic

**Jobsheet 6: Table Expression**

**Supervisor:**

Advanced Database Teaching Team

*September 202*

*4*



1. Table Expressions

**SAFRIZAL RAHMAN\_19\_SIB\_2G**

# Objective

1. Students understand how to use VIEWS
2. Students understand how to use derived tables
3. Students understand how to use common table-expression (CTE)
4. Students understand how to use inline table-valued functions (TVF)

# General Instructions

1. Follow the steps in the practical sections in the order given.
2. You can use SQL Server 2012 Standard Edition to try the practicum on this jobsheet. Adjust it to your computer's condition.
3. Answer all questions marked [Question-X] that are found in certain steps in each part of the practicum.
4. In each step of the practicum, there is an explanation that will help you answer the questions in instruction number 3, so read and do all the practicum parts in this jobsheet.
5. Write the answers to the questions in the instructions number 3 in a report that is done using a word processing application (Word, OpenOffice, or other similar). Export as a **PDF file** with the following name format:
   * **BDL\_Task 6 \_Class\_2X\_AbsenteeNumberDigit\_YourFullName** .pdf - Example:

o **BDL\_Assignment 6 \_SIB2Q\_99\_DonaldDuck** .pdf- Pay close attention to the naming format.

* + Collect the PDF files as a practical report to the supervising lecturer.
  + In addition to the file name, also include your identity on the first page of the report.

# Lab – Part 1: View - Write a SELECT query to get all products in a particular category

|  |  |
| --- | --- |
| **Step** | **Information** |
| **1** | Make sure your database is connected to 'TSQL' |
| **2** | [Question- 1 ] Write a SELECT query to display the *productid* , *productname* , *supplierid* , *unitprice* and *discontinued columns* from the **Productions.Product table** .  Then filter the results to only display products in the Beverages category ( categoryid = 1 ) |

|  |  |
| --- | --- |
| **3** | Execute the query in step 2 above and compare it with the results shown in the following display: |
| **4** | [Question- 2 ] Modify the T-SQL code from no. 2 above by adding the following T-SQL code (place it before the SELECT query)    CREATE VIEW Production . ProductsBeverages AS |
| **5** | Execute the query in step 4 above, which will produce a *VIEW object* named **ProductsBeverages** in the **Production schema.** |

# Practical – Part 2: View - Writing a SELECT query against the VIEW that has been created

|  |  |
| --- | --- |
| **Step** | **Information** |
| **1** | [Question- 3 ] Create a SELECT query consisting of the *productid* and *productname columns* from *VIEW* **Production.ProductsBeverages** . Then filter the results to only display products with supplierid = 1 . |

|  |  |
| --- | --- |
| **2** | Execute the query in step 1 above and compare it with the results shown in the following display: |

# Lab – Part 3: View - Adding an ORDER BY clause to *a VIEW*

|  |  |
| --- | --- |
| **Step** | **Information** |
| **1** | Consider the following T-SQL script:    ALTER VIEW Production . ProductsBeverages AS  SELECT  productid , product name , supplierid , unit price , discontinued  FROM Production . Products  WHERE Category ID = 1  ORDER BY product name ; |
| **2** | [Question- 4 ] After executing the T-SQL above, what happens? Write down the error message and explain the cause of the error!     **Invalid Usage Without TOP**:   * When defining a view, an ORDER BY clause is not allowed unless it is combined with TOP, OFFSET, or FOR XML. This is why your initial attempt resulted in an error.    **Workaround with TOP (100) PERCENT**:   * Adding TOP (100) PERCENT is a common workaround to include an ORDER BY in a view. While it allows the view definition to succeed, it doesn’t ensure that the data will be sorted when queried.    **Unordered Result Sets**:   * Even if you include ORDER BY in the view definition, SQL Server treats views as unordered sets of data. Thus, the rows returned by the view may not reflect the defined order unless explicitly specified in the query that selects from the view. |
| **3** | Modify the T-SQL in step 1 above by adding TOP(100) PERCENT so that now the query becomes:    ALTER VIEW Production . ProductsBeverages AS  SELECT TOP ( 100 ) PERCENT  productid , product name , supplierid , unit price , discontinued  FROM Production . Products  WHERE Category ID = 1  ORDER BY product name ; |
| **4** | Execute the T-SQL in step 3 above and notice that the query has successfully changed the VIEW **Production.ProductsBeverages** even though there is still an ORDER BY clause in the query. |
| **5** | [Question- 5 ] If a query is run against a modified VIEW **Production.ProductsBeverages , will the rows generated from the VIEW always be sorted by** *productname* ? Explain!   **View Definition vs. Query Execution**:   * When you define a view in SQL Server, the ORDER BY clause is primarily used for sorting the result set returned by the view's definition. However, this ordering is not guaranteed when the view is queried. SQL Server treats views as sets of data, which are inherently unordered.    **ORDER BY in Views**:   * The ORDER BY clause in the view definition (even with TOP (100) PERCENT) is more of a syntactic allowance than a directive that enforces sorting. When the view is queried without an explicit ORDER BY, SQL Server may return the results in any order it deems efficient. The database engine does not promise to maintain the order specified in the view. |

# Lab – Part 4: View - Adding columns to *a VIEW*

|  |  |
| --- | --- |
| **Step** | **Information** |
| **1** | Consider the following T-SQL statement that adds an additional column to the VIEW **Production.ProductsBeverages** that was created in the Practical - Part 1 with the ALTER VIEW command.    ALTER VIEW Production . ProductsBeverages AS  SELECT  productid , product name , supplierid , unit price , discontinued ,  CASE WHEN unit price > 100. THEN N'high' ELSE N'normal' END  FROM Production . Products  WHERE Category ID = 1 ; |
| **2** | [Question- 6 ] After executing the T-SQL above, what happens? Write down the error message and explain the cause of the error!   **Invalid Column Names**: The error arises because the column names in the SELECT statement contain spaces. SQL Server does not recognize product name, unit price, and Category ID as valid column identifiers due to the presence of spaces. Column names must either not contain spaces or must be enclosed in square brackets.   **Syntax Issues**: When column names contain special characters or spaces, they need to be explicitly specified to avoid ambiguity. |
| **3** | [Question- 7] Fix the T-SQL script above so that it runs correctly. |

# Lab – Part 5: View - Deleting a VIEW

|  |  |
| --- | --- |
| **Step** | **Information** |
| **1** | To delete the VIEW **Production.ProductsBeverages** , execute the following T-SQL command:   **OBJECT\_ID Function**:   * The OBJECT\_ID function is used to retrieve the object ID for a specified object. In this case, it checks for the view named Production.ProductsBeverages. * The first parameter is the name of the object (the view in this case), and the second parameter specifies the type of the object. Here, N'V' indicates that the object is a view.    **IF Condition**:   * The IF statement checks if the result of OBJECT\_ID is not NULL. If the view exists, OBJECT\_ID returns the object ID; if it does not exist, it returns NULL.    **DROP VIEW Command**:   * If the view exists, the DROP VIEW command is executed to delete the view from the database.     IF OBJECT\_ID ( N'Production.ProductsBeverages' , N'V' ) IS NOT NULL DROP VIEW Production . ProductsBeverages ; |

# Practical – Part 6: Derived Table - Creating a SELECT query in a derived table

|  |  |
| --- | --- |
| **Step** | **Information** |
| **1** | [ Question-8 ] Using the TSQL database, create a SELECT query against *the derived table* containing the *productid* and *productname columns* , with a filter to only display data whose *'pricetype'* is 'high'.  Use the SELECT query in the Practical - Part 4 - Step 1 as *the derived table* . Give the alias name *p* to the *derived table* .  SELECT  p.productid,  p.[productname]  FROM  (SELECT  productid,  [productname],  [unitprice],  discontinued,  CASE WHEN [unitprice] > 100 THEN N'high' ELSE N'normal' END AS price\_category  FROM  Production.Products  WHERE  [CategoryID] = 1) AS p  WHERE  p.price\_category = 'high'; |
|  |  |

|  |  |
| --- | --- |
| **2** | Execute the query in step 1 above and compare it with the results shown in the following display: |

# Practical – Part 7: Derived Table - Create a SELECT query to find out the total and average number of orders (nominal)

|  |  |
| --- | --- |
| **Step** | **Information** |
| **1** | [ Question- 9 ] Create a SELECT query to get the *custid column* and 2 (two) calculation columns, namely *totalsalesamount* (total nominal amount of orders per customer) and *avgsalesamount* (average nominal amount of orders per customer).    To find out the average nominal order per customer, you must first find the total nominal amount per order. The way to do this is by creating a *derived table* that contains a JOIN query between the **Sales.Orders** and **Sales.OrderDetails tables. After that, you can use the** *custid* and *orderid* columns from the **Sales.Orders table** , as well as the *qty* and *unitprice columns* from the **Sales.OrderDetails table** .   1. **SELECT Statement**:    * o.custid: Selecting the customer ID.    * SUM(od.qty \* od.unitprice) AS totalsalesamount: Calculating the total sales amount for each customer by summing the product of quantity (qty) and unit price (unitprice) from the Sales.OrderDetails.    * AVG(od.qty \* od.unitprice) AS avgsalesamount: Calculating the average sales amount per customer. 2. **FROM Clause**:    * Specifies the Sales.Orders table as the primary table (o is an alias for easier reference). 3. **JOIN Clause**:    * Joins Sales.OrderDetails (od) on the common orderid to combine the orders with their respective details. 4. **GROUP BY Clause**:    * Groups the results by custid to aggregate the total and average sales amounts for each customer. 5. **ORDER BY Clause**:    * Sorts the result set by custid for better readability.   **Execution and Comparison**  You would run the above query on your SQL Server database, and it should provide you with the total and average sales amounts for each customer. You can then compare the results with the values you've provided in your display output. |
| **2** | Execute the query in step 1 above and compare it with the results shown in the following display: |
|  |  |

# Practical – Part 8: Derived Table - Create a SELECT query to get the sales growth percentage

|  |  |
| --- | --- |
| **Step** | **Information** |
| **1** | [ Question- 10 ] Write a SELECT query that contains the following columns:   * *orderyear* : year from order date * *curtotalsales* : total amount of sales in the year * *prevtotalsales* : total sales amount in the previous year * *percentgrowth* : percentage of sales growth from the current year compared to the previous year |
| **2** | You need to create a T-SQL query using 2 (two) *derived tables* . To get the year and total sales for each SELECT query, you can use the existing VIEW named **Sales.OrderValues** . In that view, the *val column* represents the sales amount. |
| **3** | It should be noted that in the TSQL database, 2006 is the earliest order year (there are no previous years), but the query can still be executed.   **Common Table Expressions (CTEs)**:   * **YearlySales**: This CTE calculates the total sales amount for each order year by grouping the sales values from the Sales.OrderValues view. * **SalesGrowth**: This CTE calculates the current year’s total sales (curtotalsales) and the previous year’s total sales (prevtotalsales). The growth percentage is calculated only when there is a previous year’s sales data.    **LEFT JOIN**: The LEFT JOIN between the current and previous year’s totals allows you to include years with no previous data (like 2006).   **Final SELECT**: The final SELECT statement retrieves the required columns and orders the results by orderyear. |
| **4** | Execute the query in step 1 above and compare it with the results shown in the following display: |
|  |  |

# Practical – Part 9 : CTE - Creating a SELECT query using CTE

|  |  |
| --- | --- |
| **Step** | **Information** |
| **1** | [ Question-11] While still using the TSQL database, create a SELECT query like in the Practical - Part 6 , but using Common Table Expressions (CTE). Name the CTE query alias as **ProductBeverages** .   **CTE Definition**: The ProductBeverages CTE retrieves product IDs and names for products categorized as "Beverages." Adjust the filtering condition based on your requirements.   **Final SELECT**: The final SELECT retrieves product IDs and names from the ProductBeverages CTE. |
| **2** | Execute the query in step 1 and compare it with the results shown in the following display: |

**Practical – Part 10 : CTE - Create a SELECT query to get the total sales amount (nominal) for each customer.**

|  |  |
| --- | --- |
| **Step** | **Information** |
| **1** | [ Question-12] Create a SELECT query against the **Sales.OrderValues view** to get the customer ID and total sales amount in 2008. Name this CTE as **c2008** , which consists of the *custid* and *salesamt2008 columns.* |
|  | Then, perform a JOIN operation between the **Sales.Customers table** and the CTE c2008, resulting in the *custid* and *contactname columns* from the **Sales.Customer** table and *the salesamt2008 column* from the CTE **c2008** . |
| **2** | Execute the query in step 1 above and compare it with the results shown in the following display: |

**Practical – Part 11 : CTE - Create a SELECT query to compare the total sales amount for each customer with the previous year.**

|  |  |
| --- | --- |
| **Step** | **Information** |
| **1** | [ Question- 13 ] Create a SELECT query containing the *custid* and *contactname columns* against the **Sales.Customers table** . Also, get the values for the following columns:   * *salesamt2008* : total sales amount in 2008 * *salesamt2007* : total sales amount in 2007 * *percentgrowth* : percentage growth in sales between 2007 and 2008 If *percentgrowth* returns NULL, display it as 0.     You can use the CTE from Lab Part 10 and create another CTE for the year 2007. Then, perform a JOIN operation between the two CTEs with the **Sales.Customers table** . Sort the results by the *percentgrowth column.*    WITH c2008 AS (  SELECT  ov.custid,  SUM(CASE WHEN ov.orderdate >= '2008-01-01' AND ov.orderdate < '2009-01-01' THEN ov.val ELSE 0 END) AS salesamt2008  FROM  Sales.OrderValues ov  GROUP BY  ov.custid  ),  c2007 AS (  SELECT  ov.custid,  SUM(CASE WHEN ov.orderdate >= '2007-01-01' AND ov.orderdate < '2008-01-01' THEN ov.val ELSE 0 END) AS salesamt2007  FROM  Sales.OrderValues ov  GROUP BY  ov.custid  )  SELECT  c.custid,  c.contactname,  c2008.salesamt2008,  c2007.salesamt2007,  CASE  WHEN c2007.salesamt2007 = 0 THEN 0  ELSE (c2008.salesamt2008 - c2007.salesamt2007) / c2007.salesamt2007 \* 100  END AS percentgrowth  FROM  Sales.Customers c  INNER JOIN  c2008 ON c.custid = c2008.custid  INNER JOIN  c2007 ON c.custid = c2007.custid  ORDER BY  percentgrowth DESC; |
| **2** | Execute the query in step 1 above and compare it with the results shown in the following display: |
|  |  |

**Practical – Part 12: Inline TVF - Create a SELECT query to get the total sales amount (nominal) for each customer.**

|  |  |
| --- | --- |
| **Step** | **Information** |
| **1** | [ Question- 14 ] Using a TSQL database, create a SELECT query against the **Sales.OrderValues view** that contains the *custid column* and the *totalsalesamount column* (the total of the *val column* ). Filter the results to only display orders in 2007. |
| **2** | Execute the query in step 1 above and compare it with the results shown in the following display: |
| **3** | [ Question- 15 ] Create an inline TVF/Table-Valued Function by adding the following line and placing it before the SELECT query in Step 1 above.    CREATE FUNCTION dbo . fnGetSalesByCustomer  ( @orderyear US INT ) RETURNS TABLE AS |
|  | RETURN |
| **4** | [ Question- 16 ] Modify the query by replacing the constant value of 2007 in the WHERE clause, with the **@orderyear parameter** . |
| **5** | Run the script in step 4 above so that an inline TVF named **dbo.fnGetSalesByCustomer will be created.** |

# Practical – Part 12 : Inline ITF - Creating a SELECT query that operates on an inline table-valued function

|  |  |
| --- | --- |
| **Step** | **Information** |
| **1** | [ Question- 17 ] Create a SELECT query containing the *custid* and *totalsalesamount columns* against the inline TVF **dbo.fnGetSalesByCustomer** . Enter the value 2007 as the parameter. |
| **2** | Execute the query in step 1 above and compare it with the results shown in the following  display: |

# Practical – Part 13 : Inline ITF - Creating a SELECT query to get the 3 best-selling products for a particular customer

|  |  |
| --- | --- |
| **Step** | **Information** |
| **1** | [ Question-1 8 ] Create a SELECT query that displays the top 3 best-selling products for a customer with ID = 1. Get the *productid* and *productname columns* from the  **Production.Products table** . Use the *qty* and *unitprice columns from the Sales.OrderDetails* table to calculate the nominal value for each order row, which is then added up for each product to produce the *totalsalesamount column* .  Filter the results to only display data with a custid value = 1. |

|  |  |
| --- | --- |
| **2** | Execute the query in step 1 above and compare it with the results shown in the following display: |
| **3** | [ Question-1 9 ] Using the SELECT query in step 1 above, create an inline TVF by adding a few lines of function before the SELECT query and set the value of *the custid constant* in the query with the **@custid parameter** , as follows:    CREATE FUNCTION dbo . fnGetTop3ProductsForCustomer  ( @custid US INT ) RETURNS TABLE  AS  RETURN |
| **4** | Run the script so that an inline TVF named **dbo.fnGetTop3ProductsForCustomer will be created** which has a customer ID parameter. |
|  |  |
| **5** | [ Question-20] Test it by creating a SELECT query on the inline TVF and insert the value 1 as the customer ID parameter. Display the *productid* , *productname* , *totalsalesamount columns* , and give the alias name *p* for the inline TVF. |
| **6** | Execute the query in step 1 above and compare it with the results shown in the following display: |

# Lab – Part 14 : Inline TVF - Deleting inline Table-valued function

|  |  |
| --- | --- |
| **Step** | **Information** |
| **1** | Delete the inline TVF that has been created by running the following script:    IF OBJECT\_ID ( 'dbo.fnGetSalesByCustomer' ) IS NOT NULL DROP FUNCTION dbo . fnGetSalesByCustomer ;  IF OBJECT\_ID ( 'dbo.fnGetTop3ProductsForCustomer' ) IS NOT NULL DROP FUNCTION dbo . fnGetTop3ProductsForCustomer ; |

***--- Have a great time doing it ----***