Diagrams for Loyd Flores

Easy Prompt 1: Determine the names of departments along with the average tenure of employees in those departments. The tenure is determined by the difference in years between the start and end dates from the `EmployeeDepartmentHistory` table. If the end date is null, consider the current date for calculations.

DataBase : AdventureWorks2017

Tables Involved : HumanResources.Department & HumanResources.EmployeeDepartmentHistory

```sql
USE AdventureWorks2017;
SELECT
    d.Name AS DepartmentName,
    AVG(CASE
        WHEN edh.EndDate IS NULL THEN DATEDIFF(YEAR, edh.StartDate, GETDATE())
        ELSE DATEDIFF(YEAR, edh.StartDate, edh.EndDate)
    END) AS AverageTenure
FROM
    HumanResources.Department d
JOIN
    HumanResources.EmployeeDepartmentHistory edh
ON
    d.DepartmentID = edh.DepartmentId
GROUP BY
    d.Name;

-- We take the average
--      If null : StartDate - CurrentDate in years for current employees
--         Else : StartDate - End Date
```

Tables Joined :
HumanResources.Department ->
HumanResources.EmployeeDeparmentHistory
Relationship : DepartmentID

SQL

BEFORE JOIN

HumanResources.EmployeeDepartmentHistory

	BusinessEntityID	DepartmentID	ShiftID	StartDate	EndDate
1	2	1	1	2008-01-31	NULL
2	3	1	1	2007-11-11	NULL
3	4	1	1	2007-12-05	2010-05-30
4	5	1	1	2008-01-06	NULL
5	6	1	1	2008-01-24	NULL
6	14	1	1	2010-12-30	NULL
7	15	1	1	2011-01-18	NULL
8	11	2	1	2010-12-05	NULL
9	12	2	1	2007-12-11	NULL
10	13	2	1	2010-12-23	NULL

HumanResources.EmployeeDepartment

	DepartmentID	Name	GroupName
1	1	Engineering	Research and Development
2	2	Tool Design	Research and Development
3	3	Sales	Sales and Marketing
4	4	Marketing	Sales and Marketing
5	5	Purchasing	Inventory Management
6	6	Research and Development	Research and Development
7	7	Production	Manufacturing
8	8	Production Control	Manufacturing
9	9	Human Resources	Executive General and Administration
10	10	Finance	Executive General and Administration

AFTER JOIN
DepartmentName -> HumanResources.Department
AverageTenure -> HumanResourcesDepartmentHistory + Subquery to compute Average Tenure

	DepartmentName	AverageTenure
1	Engineering	12
2	Tool Design	13
3	Sales	11
4	Marketing	12
5	Purchasing	12
6	Research and Development	14
7	Production	14
8	Production Control	14
9	Human Resources	14
10	Finance	13

Easy Prompt 2:

Retrieve the list of products that have been purchased by vendors along with their average lead time, standard price, and the last receipt cost. Limit the results to products that have an average lead time greater than 10 days.

Database : AdventureWorks2017

Tables Involved :
Production.Product & Production.ProductVendor

[4]

```sql
Use AdventureWorks2017;
SELECT
    p.Name AS ProductName,
    pv.AverageLeadTime,
    pv.StandardPrice,
    pv.LastReceiptCost
FROM
    Production.Product p
JOIN
    Purchasing.ProductVendor pv
ON
    p.ProductID = pv.ProductID
WHERE
    pv.AverageLeadTime > 10
ORDER BY pv.AverageLeadTime;

-- We are first selecting the tables we need from Production.Product
-- Then We are Inner Joining, taking only those with mathcing Product ID
-- In the ProductVendor table

-- Final filter is AverageTime > 10 because that is the constraint of our problem

```

Tables Joined :

Production.Product
Purchasing.ProductVendor

Relationship : ProductID

SQL

BEFORE JOIN

Production.Product

	ProductID	Name	ProductNumber	MakeFlag	FinishedG
1	1	Adjustable Race	AR-5381	0	0
2	2	Bearing Ball	BA-8327	0	0
3	3	BB Ball Bearing	BE-2349	1	0
4	4	Headset Ball Bearings	BE-2908	0	0
5	316	Blade	BL-2036	1	0
6	317	LL Crankarm	CA-5965	0	0
7	318	ML Crankarm	CA-6738	0	0

Purchasing.ProductVendor

	ProductID	BusinessEntityID	AverageLeadTime	StandardPrice	Last
1	1	1580	17	47.87	50
2	2	1688	19	39.92	41
3	4	1650	17	54.31	57
4	317	1578	19	28.17	29
5	317	1678	17	25.77	27
6	318	1578	19	34.38	36
7	318	1678	17	31.98	33

AFTER JOIN :

ProductName -> Production.Product

AverageLeadTime, AverageLeadTime, StandardPrice,LastReceiptCost -> Purchasing.ProductVendor

	ProductName	AverageLeadTime	StandardPrice	LastReceiptCost
1	Mountain Bike Socks, M	12	3.10	3.40
2	Mountain Bike Socks, L	12	3.12	3.40
3	LL Shell	15	2.21	2.3205
4	HL Shell	15	3.47	3.6435
5	Tension Pulley	15	3.32	3.486
6	Rear Derailleur Cage	15	5.50	5.775
7	Rear Derailleur Cage	15	5.90	6.195
8	Reflector	15	8.76	9.198
9	LL Mountain Rim	15	22.28	23.394
10	LL Mountain Rim	15	21.11	22.1655

Easy Prompt 3 :

Management wants to analyze the sales performance of their employees. The objective of this analysis is to identify the top 5 employees with the highest sales amounts for year 2010

Database : AdventureWorksDW2017

Tables Involved : dbo.FactSalesQuota and dbo.DimEmployee

[17]

```sql
USE AdventureWorksDW2017
SELECT TOP(5)
    e.FirstName,
    e.LastName,
    sq.CalendarYear,
    SUM(sq.SalesAmountQuota) as TotalSalesAmount
FROM
    dbo.FactSalesQuota sq
JOIN
    dbo.DimEmployee e ON sq.EmployeeKey = e.EmployeeKey
WHERE
    sq.CalendarYear = 2010
GROUP BY
    e.FirstName,
    e.LastName,
    sq.CalendarYear
ORDER BY
    TotalSalesAmount DESC;

-- We Want to Select only the TOP(5)
-- SUM(Sq.SalesAmountQuota) as we add all the entries of Sales
-- We then Join dbo.DimEmployee on Employee key
-- To access the calendar year and set it to only 2010
-- We Then Group our results by the non-aggregate entries
-- We finish Order by to arrange the top 5

```

Tables Joined :
dbo.DimEmployee,
sq.EmployeeKey

Relationship : EmployeeKey

SQL

Before Join :

dbo.FactSalesQuota

	SalesQuotaKey	EmployeeKey	DateKey	CalendarYear	CalendarQuarter
1	1	272	20101229	2010	4
2	2	281	20101229	2010	4
3	3	282	20101229	2010	4
4	4	283	20101229	2010	4
5	5	284	20101229	2010	4
6	6	285	20101229	2010	4
7	7	286	20101229	2010	4

dbo.DimEmployee

	EmployeeKey	ParentEmployeeKey	EmployeeNationalIDAlternateKey	ParentEm
5	5	3	112457891	NULL
6	6	267	480168528	NULL
7	7	112	24756624	NULL
8	8	112	24756624	NULL
9	9	23	309738752	NULL
10	10	189	690627818	NULL
11	11	3	695256908	NULL

AFTER JOIN

FirstName, LastName -> dbo.DimEmployee
CalendarYear -> dbo.FactSalesQuota
TotalSalesAmount -> dbo.FactSalesQuota + Built in SUM() function

	FirstName	LastName	CalendarYear	TotalSalesAmount
1	Tsvi	Reiter	2010	669000.00
2	Linda	Mitchell	2010	637000.00
3	Jillian	Carson	2010	565000.00
4	José	Saraiva	2010	525000.00
5	Shu	Ito	2010	460000.00

Complex Prompt 1 :

The HR department wants to review the compensation structure for the employees over the past year. They aim to understand how monthly salaries have been determined and to gain insights into the average monthly salaries across different departments and shifts. Additionally, they wish to view the average monthly vacation and sick leave usage per department and shift.

Database : AdventureWorks2017

Tables Involved : `HumanResources.Shift`,

HumanResources.Employee, HumanResources.EmployeeDepartmentHistory,

HumanResources.Department, HumanResources.EmployeePayHistory

CTE : MonthlyRateChangeCTE

FUNCTION : MonthlySalary : Computes Monthly salary based on provided rate and pay frequency

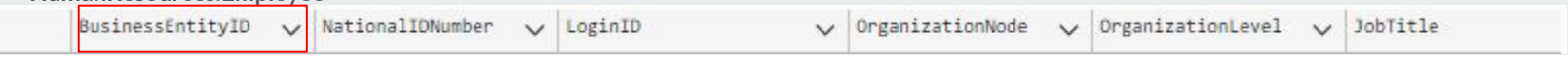
```sql
-- Check if the function `dbo.fn_MonthlySalary` exists in the database.
-- If it does, then we will proceed to drop it.
IF EXISTS (SELECT 1 FROM sys.objects WHERE object_id = OBJECT_ID(N'dbo.fn_MonthlySalary') AND type = 'FN')
DROP FUNCTION dbo.fn_MonthlySalary;
GO

-- Define and create a new function `dbo.fn_MonthlySalary`
-- This function computes the monthly salary based on provided rate and pay frequency.
    -- Using a CASE statement, determine the monthly salary.
    -- If PayFrequency is 1, it's assumed to be Monthly, hence salary is same as rate.
    -- If PayFrequency is 2, it's assumed to be Bi-Weekly, hence salary is twice the rate.
    -- For any other value, the monthly salary is assumed to be the rate itself.
CREATE FUNCTION fn_MonthlySalary (@Rate decimal(10,2), @PayFrequency int)
RETURNS decimal(10,2)
AS
BEGIN
    DECLARE @MonthlySalary decimal(10,2)

    SET @MonthlySalary = CASE
                            WHEN @PayFrequency = 1 THEN @Rate
                            WHEN @PayFrequency = 2 THEN @Rate * 2
                            ELSE @Rate
                        END
    RETURN @MonthlySalary
END;
GO
```

SELF DEFINED FUNCTION

```sql
-- Switch to the `AdventureWorks2017` database for subsequent operations.
USE AdventureWorks2017;

-- Using a CTE named `MonthlyRateChangeCTE`, compute the average rate change for employees over the past year.
-- This will be used in the final query to associate with other employee-related data.
WITH MonthlyRateChangeCTE AS
(
    SELECT
        eph.BusinessEntityID,
        AVG(eph.Rate) AS AverageMonthlyRateChange
    FROM HumanResources.EmployeePayHistory eph
    WHERE eph.RateChangeDate > DATEADD(YEAR, -1, GETDATE())
    GROUP BY eph.BusinessEntityID
)

-- The main query fetches data about:
-- - Department and Shift Names
-- - Average Monthly Salary
-- - Average Monthly Vacation and Sick Hours
-- This is achieved by joining various tables and using the function and CTE defined above.
SELECT
    d.Name AS DepartmentName,
    s.Name AS ShiftName,
    AVG(dbo.fn_MonthlySalary(eph.Rate, eph.PayFrequency)) AS AverageMonthlySalary,
    AVG(e.VacationHours/12.0) AS AverageMonthlyVacationHours,
    AVG(e.SickLeaveHours/12.0) AS AverageMonthlySickHours
FROM HumanResources.Employee e
JOIN HumanResources.EmployeeDepartmentHistory edh ON e.BusinessEntityID = edh.BusinessEntityID
JOIN HumanResources.Department d ON edh.DepartmentID = d.DepartmentID
JOIN HumanResources.EmployeePayHistory eph ON e.BusinessEntityID = eph.BusinessEntityID
JOIN HumanResources.Shift s ON edh.ShiftID = s.ShiftID
LEFT JOIN MonthlyRateChangeCTE mrc ON e.BusinessEntityID = mrc.BusinessEntityID
GROUP BY d.Name, s.Name, mrc.AverageMonthlyRateChange;}
```

Tables Used
CTE MonthlyRateChange,
HumanResources.Employee,
HumanResources.EmployeeDepartmentHistory,
HumanResources.Department,
HumanResources.EmployeePayHistory,
HumanResources.Shift,
MonthlyRateChangeCTE

HumanResources.Employee

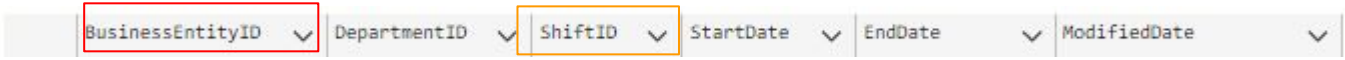


HumanResources.EmployeeDepartmentHistory

HumanResources.Department

HumanResources.PayHistory

HumanResources.EmployeeDepartmentHistory



HumanResources.Shift

MonthlyRateChangeCTE

AFTER JOIN

DepartmentName -> HumanResources.Department
ShiftName -> HumanResources.Shift
AverageMonthlySalary - > AVG() , fn_MonthlySalary From the HumanResources.Employee
AverageMonthlyVacationHours -> AVG() from the HumanResources.Employee
AverageMonthlySickHours-> AVG() from the HumanResources.Employee

	DepartmentName	ShiftName	AverageMonthlySalary	AverageMonthlyVacationHours	AverageMonthlySickHours
1	Document Control	Day	25.526666	6.499999	4.888888
2	Engineering	Day	67.344444	1.527777	3.416666
3	Executive	Day	136.610000	2.062500	2.687499
4	Facilities and Maintenance	Day	36.146666	7.361110	5.333333
5	Finance	Day	59.549230	3.814102	3.557691
6	Human Resources	Day	36.050000	4.291666	3.791666
7	Information Services	Day	68.691111	5.787036	4.537036
8	Marketing	Day	37.870000	3.708333	3.499999
9	Production	Day	16.056632	3.787414	3.539115
10	Production Control	Day	34.066666	3.722222	3.499999

Complex Prompt 2 :

You are an analyst at PrestigeCars, a renowned car dealership with a significant sales volume. PrestigeCars has detailed records of its sales across several years. As the business grew, the sales records were maintained in separate tables for each year in the DataTransfer schema.

The executive team at PrestigeCars is interested in a consolidated view of sales across these years. They wish to understand:

The maximum sale price for each car model sold to a customer from a specific country.

The average profit earned from each model, given the various costs associated with getting the car ready for sale.

For strategic reasons, the team is particularly interested in models where the average profit is greater than $5000

Database : PrestigeCars

Tables Involved :
DataTransfer.Sales2015, DataTransfer.Sales2016,

DataTransfer.Sales2017, DataTransfer.Sales2018,

ConsolidatedSales , Data.Customer

CTE : `ConsolidatedSales`

FUNCTION : `fnCalculateProfit`: Computes Monthly salary based on provided rate and pay frequency

Predefined Function

```sql
-- Check if the custom function named 'dbo.fnYearDiff' already exists in the database.
-- If it does, we drop it to ensure our new definition is the one being used.
IF EXISTS (SELECT 1 FROM sys.objects WHERE object_id = OBJECT_ID(N'dbo.fnYearDiff') AND type = 'FN')
DROP FUNCTION dbo.fnYearDiff;
GO

-- Create a user-defined function 'dbo.fnYearDiff' to compute the difference between two dates in years.
CREATE FUNCTION dbo.fnYearDiff(@StartDate DATE, @EndDate DATE)
RETURNS INT
AS
BEGIN
    -- Calculate the difference in years, considering the month and day to be precise.
    RETURN (YEAR(@EndDate) - YEAR(@StartDate) +
            CASE
                WHEN MONTH(@EndDate) > MONTH(@StartDate) OR (MONTH(@EndDate) = MONTH(@StartDate) AND DAY(@EndDate) >= DAY(@StartDate)) THEN 1
                ELSE 0
            END)
END;
GO
```

```sql
-- Use a Common Table Expression (CTE) named 'TopSuppliers' to determine the top 10 suppliers
-- based on the total volume of items ordered.
WITH TopSuppliers AS
(
    SELECT TOP (10)
        s.SupplierId,
        s.SupplierName,
        -- Aggregate the total ordered items for each supplier.
        SUM(pol.OrderedOuters) AS TotalOrderedItems
    FROM Purchasing.Suppliers s
    -- Join with PurchaseOrders and PurchaseOrderLines tables to access order details.
    JOIN Purchasing.PurchaseOrders po ON s.SupplierID = po.SupplierID
    JOIN Purchasing.PurchaseOrderLines pol ON po.PurchaseOrderID = pol.PurchaseOrderID
    GROUP BY s.SupplierId, s.SupplierName
    ORDER BY TotalOrderedItems DESC
)

-- The main query extracts detailed metrics for each of these top suppliers.
SELECT
    ts.SupplierName,
    sg.StockGroupName,
    -- Compute the average transaction amount for each supplier-stock group combination.
    AVG(st.TransactionAmount) AS AverageTransactionAmount,
    -- Count distinct transactions to get the frequency.
    COUNT(DISTINCT st.SupplierTransactionID) AS RecentTransactionCount
FROM
    TopSuppliers ts
-- Join with multiple tables to access transaction details, stock items, stock groups, etc.
JOIN Purchasing.SupplierTransactions st ON ts.SupplierId = st.SupplierId
JOIN Purchasing.PurchaseOrders po ON st.PurchaseOrderID = po.PurchaseOrderID
JOIN Purchasing.PurchaseOrderLines pol ON po.PurchaseOrderID = pol.PurchaseOrderID
JOIN Warehouse.StockItems si ON pol.StockItemID = si.StockItemID
JOIN Warehouse.StockItemStockGroups sisg ON si.StockItemID = sisg.StockItemID
JOIN Warehouse.StockGroups sg ON sisg.StockGroupID = sg.StockGroupID
-- Use the created function to filter transactions that happened more than 10 years ago.
WHERE
    dbo.fnYearDiff(st.TransactionDate, GETDATE()) > 10
GROUP BY
    ts.SupplierName,
    sg.StockGroupName
-- Order results by average transaction amount in descending order.
ORDER BY
    AverageTransactionAmount DESC;
```

TopSuppliers

SupplierId	SupplierName	TotalOrderedItems

```
JOIN Purchasing.SupplierTransactions st ON ts.SupplierId = st.SupplierId
```

SupplierTransactionID	SupplierID	TransactionTypeID	PurchaseOrderID	PaymentMethodID

```
JOIN Purchasing.PurchaseOrders po ON st.PurchaseOrderID = po.PurchaseOrderID
```

PurchaseOrderID	SupplierID	OrderDate	DeliveryMethodID	ContactPersonID	ExpectedDel

```
JOIN Purchasing.PurchaseOrderLines pol ON po.PurchaseOrderID = pol.PurchaseOrderID
```

PurchaseOrderLineID	PurchaseOrderID	StockItemID	OrderedOuters	Description

```
JOIN Warehouse.StockItems si ON pol.StockItemID = si.StockItemID
```

StockItemID	StockItemName	SupplierID	ColorID	Ur

```
JOIN Warehouse.StockItemStockGroups sisg ON si.StockItemID = sisg.StockItemID
```

StockItemStockGroupID	StockItemID	StockGroupID	LastEditedBy	LastEditedWhen

```
JOIN Warehouse.StockGroups sg ON sisg.StockGroupID = sg.StockGroupID
```

StockGroupID	StockGroupName	LastEditedBy	ValidFrom	ValidTo

AFTER JOINS

SupplierName -> TopSupplier CTE
StockGroupName -> Warehouse.StockGroups
AverageTransactionAmount -> AVG() , Purchasing.SupplierTransactions
RecentTransactionAmount -> COUNT() , Purchasing.SupplierTransactions

	SupplierName	StockGroupName	AverageTransactionAmount	RecentTransactionCount
1	Fabrikam, Inc.	T-Shirts	170438.985124	248
2	Fabrikam, Inc.	Computing Novelties	170438.985124	248
3	Fabrikam, Inc.	Clothing	164988.106014	248
4	Litware, Inc.	Packaging Materials	25214.132187	212
5	Northwind Elec…	Toys	21569.790555	10

Complex Prompt 5 :
In the NorthWinds2022TSQLV7 database, we have information regarding employees, products, suppliers, customers, orders, and more. We want to analyze the sales performance of each employee for the year 2022. Specifically, find the total number of products sold, total sales amount (considering discounts), and the sales commission for each employee based on their total sales. The sales commission is calculated as 5% of the total sales amount. Additionally, list the region where the employee is located. Order the result based on the total sales amount in descending order.

Database : NorthWinds2022TSQLV7

Tables Involved :

Sales.[Order], Sales.OrderDetail, HumanResources.Employee, o.OrderID

CTE : SalesDetails

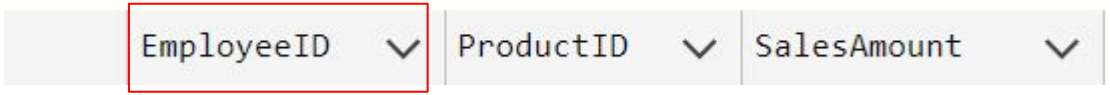
FUNCTION : `dbo.fnCalculateSalesCommission : compute the sales commission based on total sales amount.`

Self-Defined Function

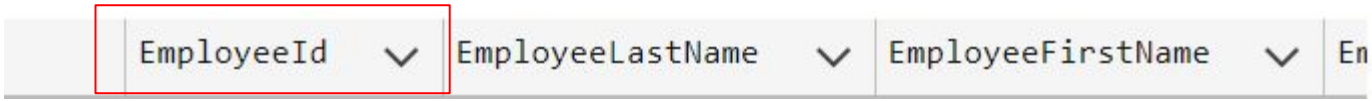
```
-- Set the active database to NorthWinds2022TSQLV7
USE NorthWinds2022TSQLV7;
GO

-- Check if the function `dbo.fnCalculateSalesCommission` exists in the database.
-- If it exists, drop it.
IF EXISTS (SELECT 1 FROM sys.objects WHERE object_id = OBJECT_ID(N'dbo.fnCalculateSalesCommission') AND type = 'FN')
DROP FUNCTION dbo.fnCalculateSalesCommission;
GO

-- Create a user-defined function to compute the sales commission based on total sales amount.
CREATE FUNCTION dbo.fnCalculateSalesCommission (@TotalSales DECIMAL(18,2))
RETURNS DECIMAL(18,2)
AS
BEGIN
    RETURN @TotalSales * 0.05
END;
GO
```

```sql
-- Using a Common Table Expression (CTE) to compute sales details for each order and product.
WITH SalesDetails AS
(
    SELECT
        o.EmployeeID,
        od.ProductID,
        od.UnitPrice * od.Quantity * (1 - od.DiscountPercentage / 100) AS SalesAmount
    FROM Sales.[Order] o
    JOIN Sales.OrderDetail od ON o.OrderID = od.OrderID
    WHERE YEAR(o.OrderDate) = 2014
)

-- Main query to fetch the sales performance of each employee.
SELECT
    e.EmployeeFirstName + ' ' + e.EmployeeLastName AS EmployeeName,
    COUNT(DISTINCT sd.ProductID) AS NumberOfProductsSold,
    SUM(sd.SalesAmount) AS TotalSalesAmount,
    -- Use the user-defined function to calculate sales commission.
    dbo.fnCalculateSalesCommission(SUM(sd.SalesAmount)) AS SalesCommission
FROM
    SalesDetails sd
JOIN HumanResources.Employee e ON sd.EmployeeID = e.EmployeeID
GROUP BY
    e.EmployeeFirstName,
    e.EmployeeLastName,
    e.EmployeeRegion
ORDER BY
    TotalSalesAmount DESC;
```

SalesDetails sd CTE




```
JOIN HumanResources.Employee e ON sd.EmployeeID = e.EmployeeID
```



EmployeeName -> Concatenation of two elements from Human.Resources.Employee
NumberProductsSold -> COUNT(DISTINCT(salesAmount) from SalesDetails CTE
TotalSalesAmount -> SUM(SalesAmount) from SalesDetails CTE
SalesComission -> fnCalculateSalesCommission(SUM(sd.SalesAmount) from SalesDetailsCTE

	EmployeeName	NumberOfProductsSold	TotalSalesAmount	SalesCommission
1	Yael Peled	53	53083.10315000000	2654.16
2	Sara Davis	40	38758.75515000000	1937.94
3	Maria Cameron	34	23152.18720000000	1157.61
4	Don Funk	31	22823.92360000000	1141.20
5	Sven Mortensen	19	21929.38720000000	1096.47
6	Judy Lew	37	19221.72160000000	961.09
7	Russell King	23	18076.07360000000	903.80
8	Paul Suurs	27	17720.21505000000	886.01
9	Patricia Doyle	12	11350.98815000000	567.55