B9DA102: Data Storage Solutions for Data Analytics

CA1 – Vignesh Muthumani (10385771)

Part 1 – Business Driver:

1.1 Subject area for analysis:

Managing the inventory efficiently based on the sales history by identifying the products that are in surplus amounts because of poor sales, and getting it sold to customers through strategic marketing.

1.2 Reasons for opting this subject area:

- One of the main reasons for opting this subject area is to minimize the losses incurred by certain products in the inventory that doesn't get sold and end up in surplus amounts.
- From the analysis of the inventory data, we can see few products that have terrible sales record with too many items in stock.

1.3 Identifying the key stakeholders:

- **Suppliers** – Managing the supplies in the inventory directly impacts the suppliers.

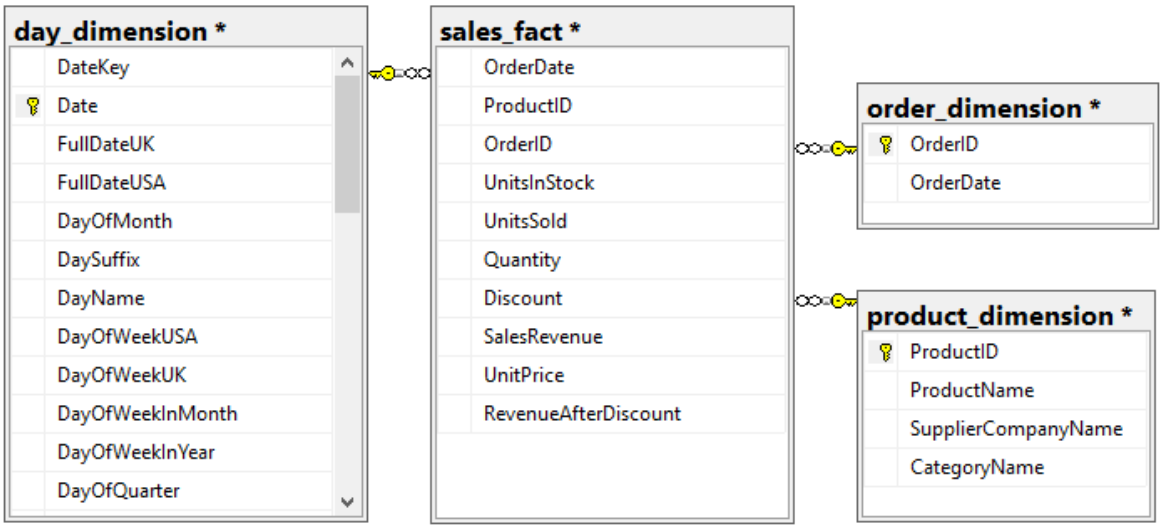
- **Products** – Products that are in surplus amounts are affected, as we tend to provide discounts and focus on their sales
- **Customers** – Ultimately customers are benefited with the discounts that we provide for these products.
- **Financial** – Stabilizing the revenue by minimizing the losses incurred.

1.4 Formalising the vision and goal of this data warehouse:

- Our goal is to get these products sold to customers through strategic marketing such as BOGOF and cross-selling the products of same categories (i.e., pairing the products that are doing terrible at sales with a highly sold product and selling it together at a discounted price).
- This, in turn, will help in balancing the stock in the inventory efficiently, thereby stabilizing the revenue of the company without incurring any significant losses.

Part 2 – Data Modelling:

2.1 Devising a star schema model for the data warehouse:



2.2 Reasons for the design:

- There are many ways to design a relational database. Star schema and snowflake schema models are the ones that are widely used in the industry. From which, we have opted for the star schema model for this subject.
- One of the main reasons for choosing the star schema over snowflake schema is,
    - Because of its simplest style and its simpler approach in effectively handling queries.
    - Unlike the Snowflake Schema, it possesses denormalized dimension tables, thereby enabling us to access the data warehouse using simple queries.

    - Because of its simple design, it excels in speed and performance in accessing the data warehouse
    - Easier to navigate and understand the model because of its compact design and clear join path.
- From the operational database, Northwind, the tables we are looking at to build our data warehouse are
    - Order Details
    - Orders
    - Products
- These tables help us in tracking the inventory and the sales record of the products. The other fields from the operational database are deemed unnecessary for our data warehouse as they aren't essential to address the issues of the subject area we have opted.
- With the help of the order details and product details fields, we can calculate the count of orders placed for each product. By summing that with the units in stock, we get the total no. of each product the inventory has ever had. Now by comparing this figure with the no. of orders placed, we can see the sales trend of each product to find out if their sales record is poor or good.

$$\boldsymbol{Total\ units = No.\ of\ units\ sold + No.\ of\ units\ in\ stock}$$

$$\% \; of \; Sales = \frac{No. \; of \; units \; sold}{Total \; units} * 100$$

- Now, as per our objective, we focus on selling the products that are in surplus with poor sales record through strategic marketing to efficiently manage the inventory and minimize the losses that these products incur.

Part 3 – Implementing Tables and ETL Procedures:

3.1 Using Microsoft SQL Server, we are implementing the tables for our data warehouse using ETL procedures from the operational database, Northwind.

As per the star schema model we had designed in the part 2, we create the fact table and dimensional tables in our data warehouse, 'Northwind_DW' with the help of the Microsoft SQL Server. The process is as follows,

3.1.1 Creation of the Product Dimension table:

Creating our product dimension table by declaring the fields that are to be present in our table.

```
-- creating the product dimension table
CREATE TABLE Northwind_DW.dbo.product_dimension (
    ProductID INT NOT NULL,
    ProductName NVARCHAR(40) NOT NULL,
    SupplierCompanyName NVARCHAR(40) NOT NULL,
    CategoryName NVARCHAR(15) NOT NULL,
```

Setting the primary key for our table

```
-- setting the primary key
PRIMARY KEY CLUSTERED
(
    [ProductID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
```

Selecting the Northwind database for populating our data warehouse

```
-- selecting the northwind database for populating
USE Northwind
GO
```

Populating our created table with the data from Northwind database

```
-- populating the product_dimension table
INSERT INTO NorthWind_DW.[dbo].product_dimension
        (ProductID,
        ProductName,
        SupplierCompanyName,
        CategoryName
        )
SELECT DISTINCT
    p.ProductID,
    p.ProductName,
    s.CompanyName AS SupplierCompanyName,
    c.CategoryName
FROM [dbo].[Products] AS p
LEFT JOIN [dbo].[Suppliers] AS s
    ON p.SupplierID = s.SupplierID
LEFT JOIN [dbo].[Categories] AS c
    ON p.CategoryID = c.CategoryID;
```

Now, our product dimensional table has been created and populated with data in our data warehouse, ‘Northwind_DW’.

To check if it has been populated correctly,

```
-- checking if the table is populated correctly
SELECT * FROM NorthWind_DW.dbo.product_dimension;
```

We can see that our product dimension table has been created successfully. We can see a glimpse of it below,

	ProductID	ProductName	SupplierCompanyName	CategoryName
1	1	Chai	Exotic Liquids	Beverages
2	2	Chang	Exotic Liquids	Beverages
3	3	Aniseed Syrup	Exotic Liquids	Condiments
4	4	Chef Anton's Cajun Seasoning	New Orleans Cajun Delights	Condiments
5	5	Chef Anton's Gumbo Mix	New Orleans Cajun Delights	Condiments
6	6	Grandma's Boysenberry Spread	Grandma Kelly's Homestead	Condiments
7	7	Uncle Bob's Organic Dried Pears	Grandma Kelly's Homestead	Produce
8	8	Northwoods Cranberry Sauce	Grandma Kelly's Homestead	Condiments
9	9	Mishi Kobe Niku	Tokyo Traders	Meat/Poultry
10	10	Ikura	Tokyo Traders	Seafood
11	11	Queso Cabrales	Cooperativa de Quesos 'Las Cabras'	Dairy Products

3.1.2 Creation of the Order Dimension table:

Creating our order dimension table by declaring the fields that are to be present in our table.

```
-- creating the order dimension table
CREATE TABLE Northwind_DW.dbo.order_dimension (
    OrderID INT NOT NULL,
    OrderDate datetime NOT NULL,
```

Setting the primary key for our table

```
-- setting the primary key
PRIMARY KEY CLUSTERED
(
    OrderID ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
```

Selecting the Northwind database for populating our data warehouse

```
-- selecting the northwind database for populating
USE Northwind
GO
```

Populating our created table with the data from Northwind database

```
-- populating the order dimension table
INSERT INTO NorthWind_DW.[dbo].order_dimension
        (OrderID,
        OrderDate)

SELECT DISTINCT
    o.OrderID,
    o.OrderDate

FROM [dbo].Orders AS o
```

Now, our order dimensional table has been created and populated with data in our data warehouse, 'Northwind_DW'.

To check if it has been populated correctly,

```
-- checking if the table is populated correctly
SELECT * FROM NorthWind_DW.dbo.order_dimension;
```

We can see that our product dimension table has been created successfully. We can see a glimpse of it below,

	OrderID	OrderDate
1	10248	1996-07-04 00:00:00.000
2	10249	1996-07-05 00:00:00.000
3	10250	1996-07-08 00:00:00.000
4	10251	1996-07-08 00:00:00.000
5	10252	1996-07-09 00:00:00.000
6	10253	1996-07-10 00:00:00.000
7	10254	1996-07-11 00:00:00.000
8	10255	1996-07-12 00:00:00.000

3.1.3 Creation of the Day Dimension table:

```
USE NorthWind_DW;

CREATE TABLE    [dbo].[day_dimension]
    (   [DateKey] INT,
        [Date] DATETIME primary key,
        [FullDateUK] CHAR(10), -- Date in dd-MM-yyyy format
        [FullDateUSA] CHAR(10),-- Date in MM-dd-yyyy format
        [DayOfMonth] INT, -- Field will hold day number of Month
        [DaySuffix] VARCHAR(4), -- Apply suffix as 1st, 2nd ,3rd etc
        [DayName] VARCHAR(9), -- Contains name of the day, Sunday, Monday
        [DayOfWeekUSA] INT,-- First Day Sunday=1 and Saturday=7
        [DayOfWeekUK] INT,-- First Day Monday=1 and Sunday=7
        [DayOfWeekInMonth] INT, --1st Monday or 2nd Monday in Month
        [DayOfWeekInYear] INT,
        [DayOfQuarter] INT,
        [DayOfYear] INT,
        [WeekOfMonth] INT,-- Week Number of Month
        [WeekOfQuarter] INT, --Week Number of the Quarter
        [WeekOfYear] INT,--Week Number of the Year
        [Month] INT, --Number of the Month 1 to 12
        [MonthName] VARCHAR(9),--January, February etc
        [MonthOfQuarter] INT,-- Month Number belongs to Quarter
        [Quarter] INT,
        [QuarterName] VARCHAR(9),--First,Second..
        [Year] INT,-- Year value of Date stored in Row
        [YearName] CHAR(7), --CY 2012,CY 2013
        [MonthYear] CHAR(10), --Jan-2013,Feb-2013
        [MMYYYY] INT,

        [FirstDayOfMonth] DATE,
        [LastDayOfMonth] DATE,
        [FirstDayOfQuarter] DATE,
        [LastDayOfQuarter] DATE,
        [FirstDayOfYear] DATE,
        [LastDayOfYear] DATE,
        [IsHolidayUSA] BIT,-- Flag 1=National Holiday, 0-No National Holiday
        [IsWeekday] BIT,-- 0=Week End ,1=Week Day
        [HolidayUSA] VARCHAR(50),--Name of Holiday in US
        [IsHolidayUK] BIT Null,-- Flag 1=National Holiday, 0-No National Holiday
        [HolidayUK] VARCHAR(50) Null --Name of Holiday in UK
    )
GO

/*****************************************************************************************************
--Specify Start Date and End date here
--Value of Start Date Must be Less than Your End Date

DECLARE @StartDate DATETIME = '01/01/1990' --Starting value of Date Range
DECLARE @EndDate DATETIME = '01/01/2015' --End Value of Date Range
```

```
--Temporary Variables To Hold the Values During Processing of Each Date of Year
DECLARE
    @DayOfWeekInMonth INT,
    @DayOfWeekInYear INT,
    @DayOfQuarter INT,
    @WeekOfMonth INT,
    @CurrentYear INT,
    @CurrentMonth INT,
    @CurrentQuarter INT

/*Table Data type to store the day of week count for the month and year*/
DECLARE @DayOfWeek TABLE (DOW INT, MonthCount INT, QuarterCount INT, YearCount INT)

INSERT INTO @DayOfWeek VALUES (1, 0, 0, 0)
INSERT INTO @DayOfWeek VALUES (2, 0, 0, 0)
INSERT INTO @DayOfWeek VALUES (3, 0, 0, 0)
INSERT INTO @DayOfWeek VALUES (4, 0, 0, 0)
INSERT INTO @DayOfWeek VALUES (5, 0, 0, 0)
INSERT INTO @DayOfWeek VALUES (6, 0, 0, 0)
INSERT INTO @DayOfWeek VALUES (7, 0, 0, 0)

--Extract and assign various parts of Values from Current Date to Variable

DECLARE @CurrentDate AS DATETIME = @StartDate
SET @CurrentMonth = DATEPART(MM, @CurrentDate)
SET @CurrentYear = DATEPART(YY, @CurrentDate)
SET @CurrentQuarter = DATEPART(QQ, @CurrentDate)


/********************************************************************************************/
--Proceed only if Start Date(Current date ) is less than End date you specified above

WHILE @CurrentDate < @EndDate
BEGIN

/*Begin day of week logic*/

         /*Check for Change in Month of the Current date if Month changed then
          Change variable value*/
    IF @CurrentMonth != DATEPART(MM, @CurrentDate)
    BEGIN
        UPDATE @DayOfWeek
        SET MonthCount = 0
        SET @CurrentMonth = DATEPART(MM, @CurrentDate)
    END

        /* Check for Change in Quarter of the Current date if Quarter changed then change
         Variable value*/

    IF @CurrentQuarter != DATEPART(QQ, @CurrentDate)
    BEGIN
        UPDATE @DayOfWeek
        SET QuarterCount = 0
        SET @CurrentQuarter = DATEPART(QQ, @CurrentDate)
    END
```

```
    /* Check for Change in Year of the Current date if Year changed then change
     Variable value*/


IF @CurrentYear != DATEPART(YY, @CurrentDate)
BEGIN
    UPDATE @DayOfWeek
    SET YearCount = 0
    SET @CurrentYear = DATEPART(YY, @CurrentDate)
END

    -- Set values in table data type created above from variables

UPDATE @DayOfWeek
SET
    MonthCount = MonthCount + 1,
    QuarterCount = QuarterCount + 1,
    YearCount = YearCount + 1
WHERE DOW = DATEPART(DW, @CurrentDate)

SELECT
    @DayOfWeekInMonth = MonthCount,
    @DayOfQuarter = QuarterCount,
    @DayOfWeekInYear = YearCount
FROM @DayOfWeek
WHERE DOW = DATEPART(DW, @CurrentDate)


/* Populate Your Dimension Table with values*/

    INSERT INTO [dbo].[day_dimension]
    SELECT

        CONVERT (char(8),@CurrentDate,112) as DateKey,
        @CurrentDate AS Date,
        CONVERT (char(10),@CurrentDate,103) as FullDateUK,
        CONVERT (char(10),@CurrentDate,101) as FullDateUSA,
        DATEPART(DD, @CurrentDate) AS DayOfMonth,
        --Apply Suffix values like 1st, 2nd 3rd etc..
        CASE
            WHEN DATEPART(DD,@CurrentDate) IN (11,12,13)
            THEN CAST(DATEPART(DD,@CurrentDate) AS VARCHAR) + 'th'
            WHEN RIGHT(DATEPART(DD,@CurrentDate),1) = 1
            THEN CAST(DATEPART(DD,@CurrentDate) AS VARCHAR) + 'st'
            WHEN RIGHT(DATEPART(DD,@CurrentDate),1) = 2
            THEN CAST(DATEPART(DD,@CurrentDate) AS VARCHAR) + 'nd'
            WHEN RIGHT(DATEPART(DD,@CurrentDate),1) = 3
            THEN CAST(DATEPART(DD,@CurrentDate) AS VARCHAR) + 'rd'
            ELSE CAST(DATEPART(DD,@CurrentDate) AS VARCHAR) + 'th'
            END AS DaySuffix,
```

```
DATENAME(DW, @CurrentDate) AS DayName,
DATEPART(DW, @CurrentDate) AS DayOfWeekUSA,

-- check for day of week as Per US and change it as per UK format
CASE DATEPART(DW, @CurrentDate)
    WHEN 1 THEN 7
    WHEN 2 THEN 1
    WHEN 3 THEN 2
    WHEN 4 THEN 3
    WHEN 5 THEN 4
    WHEN 6 THEN 5
    WHEN 7 THEN 6
    END
    AS DayOfWeekUK,

@DayOfWeekInMonth AS DayOfWeekInMonth,
@DayOfWeekInYear AS DayOfWeekInYear,
@DayOfQuarter AS DayOfQuarter,
DATEPART(DY, @CurrentDate) AS DayOfYear,
DATEPART(WW, @CurrentDate) + 1 - DATEPART(WW, CONVERT(VARCHAR,
DATEPART(MM, @CurrentDate)) + '/1/' + CONVERT(VARCHAR,
DATEPART(YY, @CurrentDate))) AS WeekOfMonth,
(DATEDIFF(DD, DATEADD(QQ, DATEDIFF(QQ, 0, @CurrentDate), 0),
@CurrentDate) / 7) + 1 AS WeekOfQuarter,
DATEPART(WW, @CurrentDate) AS WeekOfYear,
DATEPART(MM, @CurrentDate) AS Month,
DATENAME(MM, @CurrentDate) AS MonthName,
CASE


    WHEN DATEPART(MM, @CurrentDate) IN (1, 4, 7, 10) THEN 1
    WHEN DATEPART(MM, @CurrentDate) IN (2, 5, 8, 11) THEN 2
    WHEN DATEPART(MM, @CurrentDate) IN (3, 6, 9, 12) THEN 3
    END AS MonthOfQuarter,
DATEPART(QQ, @CurrentDate) AS Quarter,
CASE DATEPART(QQ, @CurrentDate)
    WHEN 1 THEN 'First'
    WHEN 2 THEN 'Second'
    WHEN 3 THEN 'Third'
    WHEN 4 THEN 'Fourth'
    END AS QuarterName,
DATEPART(YEAR, @CurrentDate) AS Year,
'CY ' + CONVERT(VARCHAR, DATEPART(YEAR, @CurrentDate)) AS YearName,
LEFT(DATENAME(MM, @CurrentDate), 3) + '-' + CONVERT(VARCHAR,
DATEPART(YY, @CurrentDate)) AS MonthYear,
RIGHT('0' + CONVERT(VARCHAR, DATEPART(MM, @CurrentDate)),2) +
CONVERT(VARCHAR, DATEPART(YY, @CurrentDate)) AS MMYYYY,
CONVERT(DATETIME, CONVERT(DATE, DATEADD(DD, - (DATEPART(DD,
@CurrentDate) - 1), @CurrentDate))) AS FirstDayOfMonth,
CONVERT(DATETIME, CONVERT(DATE, DATEADD(DD, - (DATEPART(DD,
```

```
        (DATEADD(MM, 1, @CurrentDate)))), DATEADD(MM, 1,
        @CurrentDate)))) AS LastDayOfMonth,
        DATEADD(QQ, DATEDIFF(QQ, 0, @CurrentDate), 0) AS FirstDayOfQuarter,
        DATEADD(QQ, DATEDIFF(QQ, -1, @CurrentDate), -1) AS LastDayOfQuarter,
        CONVERT(DATETIME, '01/01/' + CONVERT(VARCHAR, DATEPART(YY,
        @CurrentDate))) AS FirstDayOfYear,
        CONVERT(DATETIME, '12/31/' + CONVERT(VARCHAR, DATEPART(YY,
        @CurrentDate))) AS LastDayOfYear,
        NULL AS IsHolidayUSA,
        CASE DATEPART(DW, @CurrentDate)
            WHEN 1 THEN 0
            WHEN 2 THEN 1
            WHEN 3 THEN 1
            WHEN 4 THEN 1
            WHEN 5 THEN 1
            WHEN 6 THEN 1
            WHEN 7 THEN 0
            END AS IsWeekday,
        NULL AS HolidayUSA, Null, Null

    SET @CurrentDate = DATEADD(DD, 1, @CurrentDate)
END
```

Our day dimension table has been successfully created.

3.1.4 Creating the Sales Fact table:

Creating our sales fact table by declaring the fields that are to be present in our table.

```
-- selecting my datawarehouse
USE NorthWind_DW
GO

-- creating the sales fact table
CREATE TABLE [dbo].[sales_fact](
    [OrderDate] [datetime] NOT NULL,
    [ProductID] [int] NOT NULL,
    OrderID INT NOT NULL,
    UnitsInStock INT NULL,
    UnitsSold INT NULL,
    [Quantity] [smallint] NULL,
    [Discount] [real] NULL,
    [SalesRevenue] [money] NULL,
    [UnitPrice] [money] NULL,
    [RevenueAfterDiscount] [money] NULL,
```

Setting the primary keys for our fact table

```
-- setting the primary keys
 CONSTRAINT [IX_sales_fact] UNIQUE NONCLUSTERED
(
    [OrderDate] ASC,
    [ProductID] ASC,
    OrderID ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO
```

Populating the fact table with data from Northwind database

```
-- populating the table
INSERT INTO NorthWind_DW.[dbo].sales_fact
        (OrderDate,
        ProductID,
        OrderID,
        UnitsInStock,
        UnitsSold,
        Quantity,
        SalesRevenue,
        UnitPrice,
        RevenueAfterDiscount
        )

-- selecting the northwind database for populating
USE Northwind
GO


SELECT
    o.OrderDate,
    od.ProductID,
    od.OrderID,
    p.UnitsInStock,
    COUNT(od.OrderID) AS UnitsSold,
    SUM(od.Quantity) AS Quantity,
    SUM(od.UnitPrice * od.Quantity) AS SalesRevenue,
    SUM(od.UnitPrice * od.Quantity) / SUM(od.Quantity) AS UnitPrice,
    SUM(CONVERT(money,
        (od.UnitPrice*od.Quantity*(1-od.Discount)/100))*100)
        AS RevenueAfterDiscount
```

```
FROM [dbo].[Order Details] AS od
JOIN [dbo].[Orders] AS o
    ON od.OrderID = o.OrderID
JOIN [Northwind].[dbo].[Products] AS p
    ON od.ProductID = p.ProductID
GROUP BY o.OrderDate, od.ProductID, od.OrderID, p.UnitsInStock;
```

Now, our sales fact table has been created and populated with data in our data warehouse, 'Northwind_DW'.

To check if it has been populated correctly,

```
-- checking if the table is populated correctly
SELECT * FROM [NorthWind_DW].[dbo].sales_fact;
```

We can see that our product dimension table has been created successfully. We can see a glimpse of it below,

	OrderDate	ProductID	OrderID	UnitsInStock	UnitsSold	Quantity	Discount	SalesRevenue	UnitPrice	RevenueAfterDiscount
1	1996-07-04 00:00:00.000	11	10248	22	1	12	NULL	168.00	14.00	168.00
2	1996-07-04 00:00:00.000	42	10248	26	1	10	NULL	98.00	9.80	98.00
3	1996-07-04 00:00:00.000	72	10248	14	1	5	NULL	174.00	34.80	174.00
4	1996-07-05 00:00:00.000	14	10249	35	1	9	NULL	167.40	18.60	167.40
5	1996-07-05 00:00:00.000	51	10249	20	1	40	NULL	1696.00	42.40	1696.00
6	1996-07-08 00:00:00.000	41	10250	85	1	10	NULL	77.00	7.70	77.00
7	1996-07-08 00:00:00.000	51	10250	20	1	35	NULL	1484.00	42.40	1261.40
8	1996-07-08 00:00:00.000	65	10250	76	1	15	NULL	252.00	16.80	214.20
9	1996-07-08 00:00:00.000	22	10251	104	1	6	NULL	100.80	16.80	95.76
10	1996-07-08 00:00:00.000	57	10251	36	1	15	NULL	234.00	15.60	222.30

3.2 Reasons for using the Microsoft SQL server:

Microsoft products are known for their simplicity, flexibility and familiarity. Their SQL Server is no exception. They provide numerous benefits

- **Ease of installation** - Microsoft products are usually easy to install with simple installation procedures and simple

graphical user interface with clear set of instructions for the layman. MS SQL Server possess all these characteristics and it is user friendly unlike other database servers that require command configurations which is a complex process.

- **Performance** - It has good compression and encryption capabilities which makes it excel in data storage and retrieval functions.
- **Security** - It is one of the most secure database servers with complex encryption algorithms making it virtually impossible to crack the security layers enforced by the user. It is not an open source database server which reduces the risk of attacks on the database server.
- **Editions and Pricing** - It is available in various editions to cater the needs of corporate sectors to a domestic user. Each of these editions are priced in such a way that users can pick the ones they can afford.
- **Data Recovery and Restoration** - It has excellent features that allows the users to recover and restore their data which was lost or damaged. Though it is not possible to recover individual records, it is possible to restore an entire database using advanced recovery tools that it possesses.
- **Trustworthy and familiar** - Building on a platform that users can work with in-built security features, familiar tools, and financially backed SLAs.

3.3 Comparison of MS solution with other data warehousing solutions:

Some of the other top data warehouse solutions are Teradata, Oracle, Amazon Web Services, Cloudera, Snowflake, Vertica, Aster Data, Greenplum, IBM Netezza, HANA, Hbase, etc.

- **Teradata** – It is a market leader in the data warehousing space. Its EDW platform provides businesses with robust, scalable hybrid-storage capabilities and analytics from mounds of unstructured and structured data leading to real-time business intelligence insights, trends, and opportunities. But its hardware and licenses are the most expensive of all options. Staff costs can be expensive, and it takes a great deal of effort to configure and administer.
- **Oracle** - It possess the industry standard for high performance scalable, optimized data warehousing. Its specialized platform for the data warehousing is Oracle Exadata. It provides advanced features such as Flash Storage for low I/O overhead and Hybrid Columnar Compression (HCC), which enables high level compression of data for reduced I/O especially for analytics.
- **Amazon Web Services** - The whole shift in data storage and warehousing to the cloud over the last several years has been momentous and Amazon has been a market leader in

that whole paradigm. It offers a whole ecosystem of data storage tools and resources that complement its cloud services platform.

- **Cloudera** – It is a major enterprise provider of Hadoop-based data storage and processing solutions. It offers an Enterprise Data Hub (EDH) for its variety of operational data warehouse. The EDH is its proprietary framework for the "information-driven enterprise" and focuses on "batch processing, interactive SQL, enterprise search, and advanced analytics. Its data warehouse is based on CDH, which is Cloudera's version of Apache Hadoop.
- **Hadoop Hbase** – Commodity hardware and no license cost, resulting in lowest up-front cost. Likely to buy more hardware for redundancy and load. But requires highly technical staff and implementation is less productive than more mature options.
- **IBM dashDB** – It is a fully managed cloud data warehouse, ideal for business intelligence reporting and analytics. It is used for storing relational data, including special types such as geospatial data.
- **Snowflake** – Snowflake is the eponymous data warehouse with an emphasis on analysis.

Part 4 – Reporting and Analysis:

The data visualization tools that we have used for the reporting and analysis of our data warehouse are Tableau and Power BI

Products Info

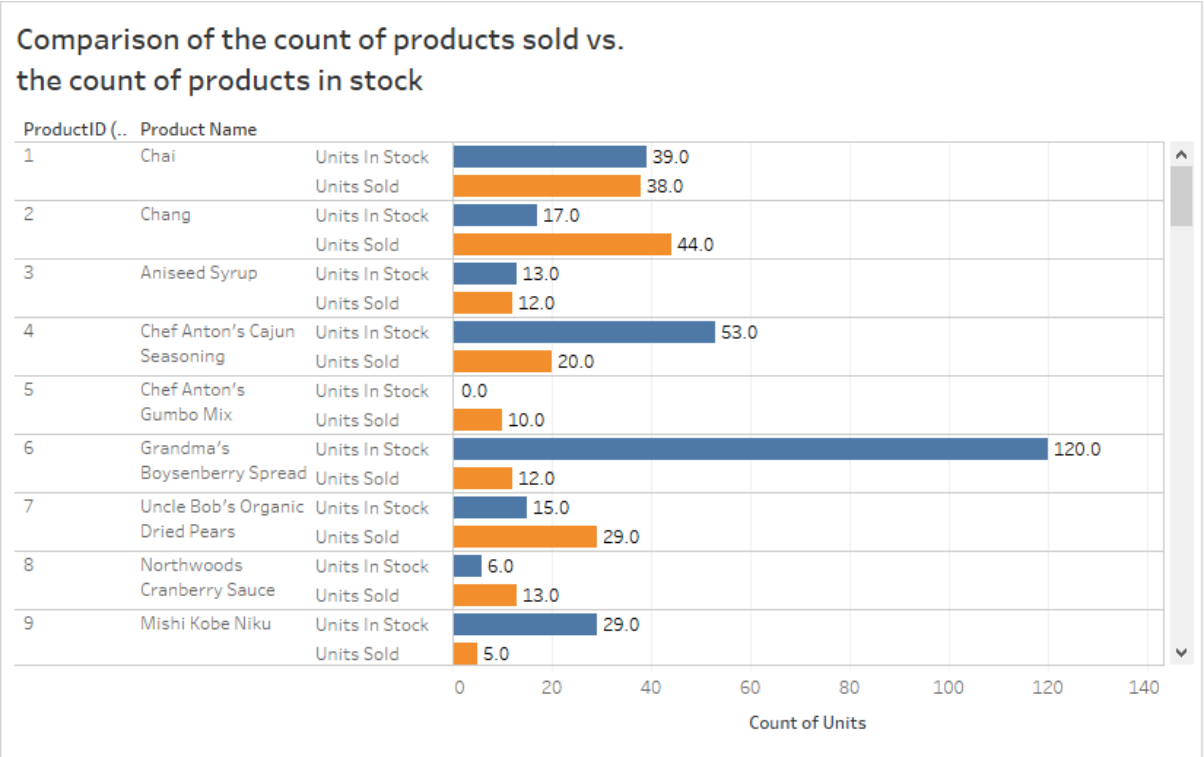
ProductID (..	Product Name	Units In Stock	Units Sold	Total Units	% of sales
1	Chai	39.0	38.0	77.0	49.4
2	Chang	17.0	44.0	61.0	72.1
3	Aniseed Syrup	13.0	12.0	25.0	48.0
4	Chef Anton's Cajun Seaso..	53.0	20.0	73.0	27.4
5	Chef Anton's Gumbo Mix	0.0	10.0	10.0	100.0
6	Grandma's Boysenberry S..	120.0	12.0	132.0	9.1
7	Uncle Bob's Organic Dried..	15.0	29.0	44.0	65.9
8	Northwoods Cranberry Sa..	6.0	13.0	19.0	68.4
9	Mishi Kobe Niku	29.0	5.0	34.0	14.7
10	Ikura	31.0	33.0	64.0	51.6
11	Queso Cabrales	22.0	38.0	60.0	63.3
12	Queso Manchego La Pasto..	86.0	14.0	100.0	14.0
13	Konbu	24.0	40.0	64.0	62.5
14	Tofu	35.0	22.0	57.0	38.6
15	Genen Shouyu	39.0	6.0	45.0	13.3
16	Pavlova	29.0	43.0	72.0	59.7
17	Alice Mutton	0.0	37.0	37.0	100.0
18	Carnarvon Tigers	42.0	27.0	69.0	39.1
19	Teatime Chocolate Biscuits	25.0	37.0	62.0	59.7
20	Sir Rodney's Marmalade	40.0	16.0	56.0	28.6
21	Sir Rodney's Scones	3.0	39.0	42.0	92.9
22	Gustaf's Knäckebröd	104.0	14.0	118.0	11.9

23	Tunnbröd	61.0	20.0	81.0	24.7
24	Guaraná Fantástica	20.0	51.0	71.0	71.8
25	NuNuCa Nuß-Nougat-Cre..	76.0	18.0	94.0	19.1
26	Gumbär Gummibärchen	15.0	32.0	47.0	68.1
27	Schoggi Schokolade	49.0	9.0	58.0	15.5
28	Rössle Sauerkraut	26.0	33.0	59.0	55.9
29	Thüringer Rostbratwurst	0.0	32.0	32.0	100.0
30	Nord-Ost Matjeshering	10.0	32.0	42.0	76.2
31	Gorgonzola Telino	0.0	51.0	51.0	100.0
32	Mascarpone Fabioli	9.0	15.0	24.0	62.5
33	Geitost	112.0	32.0	144.0	22.2
34	Sasquatch Ale	111.0	19.0	130.0	14.6
35	Steeleye Stout	20.0	36.0	56.0	64.3
36	Inlagd Sill	112.0	31.0	143.0	21.7
37	Gravad lax	11.0	6.0	17.0	35.3
38	Côte de Blaye	17.0	24.0	41.0	58.5
39	Chartreuse verte	69.0	30.0	99.0	30.3
40	Boston Crab Meat	123.0	41.0	164.0	25.0
41	Jack's New England Clam ..	85.0	47.0	132.0	35.6
42	Singaporean Hokkien Frie..	26.0	30.0	56.0	53.6
43	Ipoh Coffee	17.0	28.0	45.0	62.2
44	Gula Malacca	27.0	24.0	51.0	47.1
45	Rogede sild	5.0	14.0	19.0	73.7
46	Spegesild	95.0	27.0	122.0	22.1
47	Zaanse koeken	36.0	21.0	57.0	36.8
48	Chocolade	15.0	6.0	21.0	28.6
49	Maxilaku	10.0	21.0	31.0	67.7
50	Valkoinen suklaa	65.0	10.0	75.0	13.3
51	Manjimup Dried Apples	20.0	39.0	59.0	66.1
52	Filo Mix	38.0	29.0	67.0	43.3
53	Perth Pasties	0.0	30.0	30.0	100.0
54	Tourtière	21.0	36.0	57.0	63.2
55	Pâté chinois	115.0	33.0	148.0	22.3
56	Gnocchi di nonna Alice	21.0	50.0	71.0	70.4
57	Ravioli Angelo	36.0	23.0	59.0	39.0
58	Escargots de Bourgogne	62.0	18.0	80.0	22.5
59	Raclette Courdavault	79.0	54.0	133.0	40.6
60	Camembert Pierrot	19.0	51.0	70.0	72.9
61	Sirop d'érable	113.0	24.0	137.0	17.5
62	Tarte au sucre	17.0	48.0	65.0	73.8
63	Vegie-spread	24.0	17.0	41.0	41.5
64	Wimmers gute Semmelkn..	22.0	30.0	52.0	57.7
65	Louisiana Fiery Hot Peppe..	76.0	32.0	108.0	29.6
66	Louisiana Hot Spiced Okra	4.0	8.0	12.0	66.7

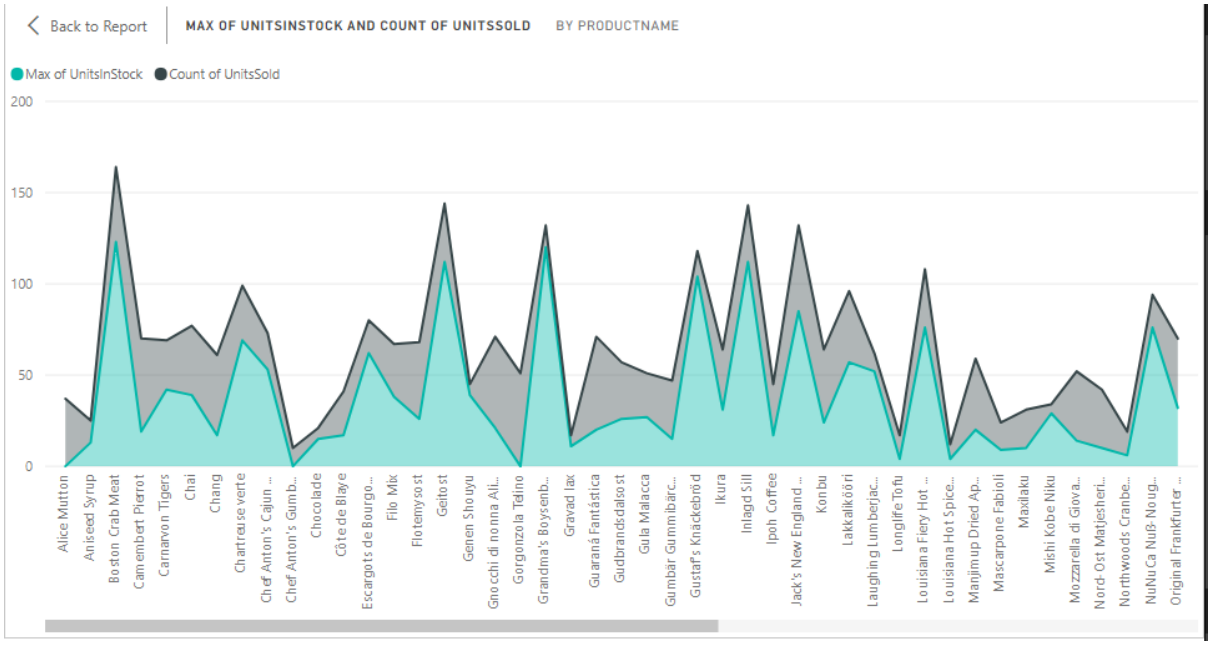
67	Laughing Lumberjack Lager	52.0	10.0	62.0	16.1
68	Scottish Longbreads	6.0	34.0	40.0	85.0
69	Gudbrandsdalsost	26.0	31.0	57.0	54.4
70	Outback Lager	15.0	39.0	54.0	72.2
71	Flotemysost	26.0	42.0	68.0	61.8
72	Mozzarella di Giovanni	14.0	38.0	52.0	73.1
73	Röd Kaviar	101.0	14.0	115.0	12.2
74	Longlife Tofu	4.0	13.0	17.0	76.5
75	Rhönbräu Klosterbier	125.0	46.0	171.0	26.9
76	Lakkalikööri	57.0	39.0	96.0	40.6
77	Original Frankfurter grün..	32.0	38.0	70.0	54.3

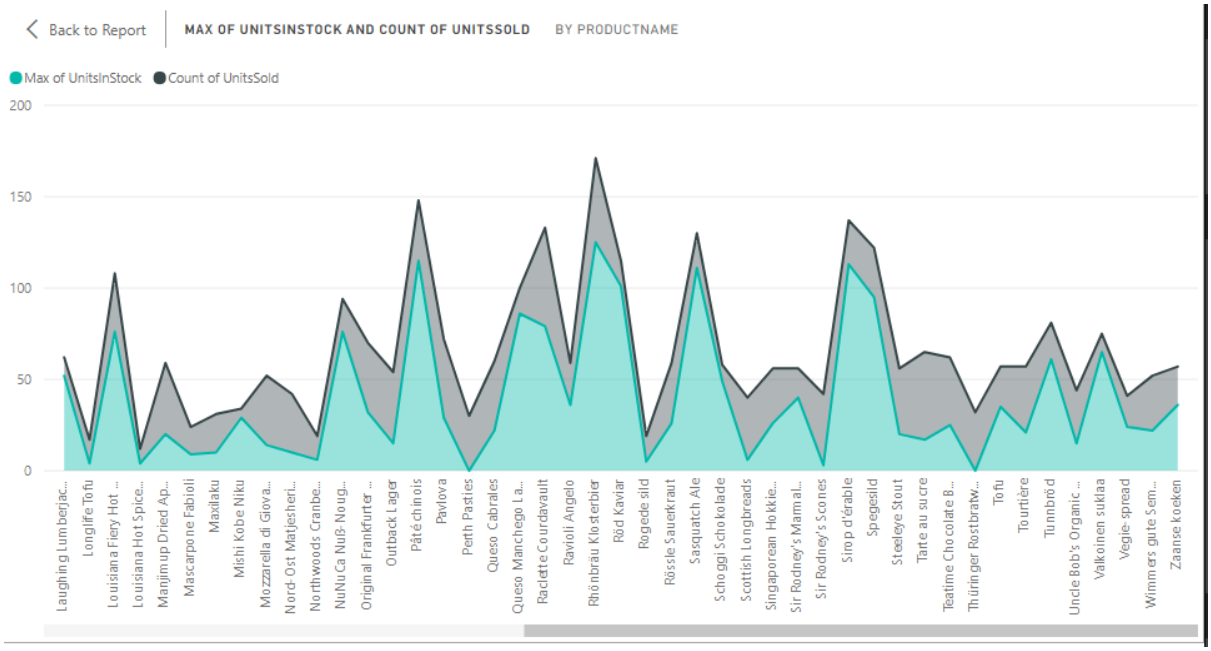
In the above table, we can see an overview of info of all the 77 products. We can see each of their count of units in stock and the count of units sold, which is followed by their total to calculate the percentage of sales for each product.

4.1 Comparison of products sold vs. products in stock:



In the above chart, we compare these two fields to find out the products that are in surplus amounts in the inventory. For (e.g., we can see that the **Product ID – 6 (Grandma's Spread) is in abundant with a poor sales record.**

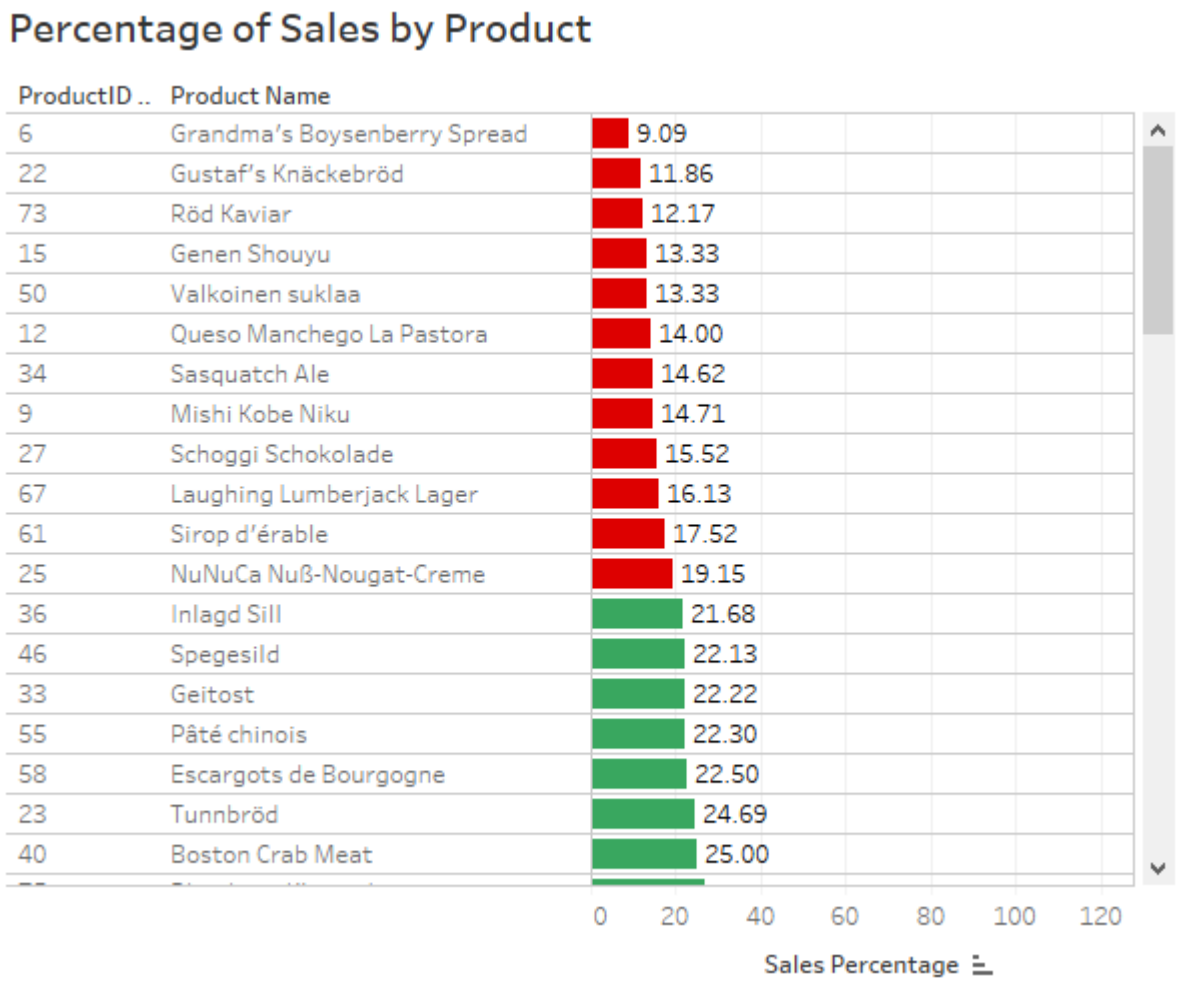




In the above 2 charts, we see an alternate representation of the comparison of same fields using the power BI. We see a significant difference for products such as Aniseed Syrup, Tarte au Sauce, etc.

4.2 Percentage of Sales by Product:

Now, we calculate the sales ratio of all the 77 products.



Here, we have sorted the chart based on the poor sales ratio of the products. We can see that the first 12 products that are in red colour have less than 20% sales record compared to what we have in stock.

It’s certain that these 12 products are in surplus amounts in our inventory with a poor sales history. Our goal is to provide discounts and focus more on the sales of these products to manage the inventory efficiently.

The products with less than 20% of sales record that we need to focus on selling are:

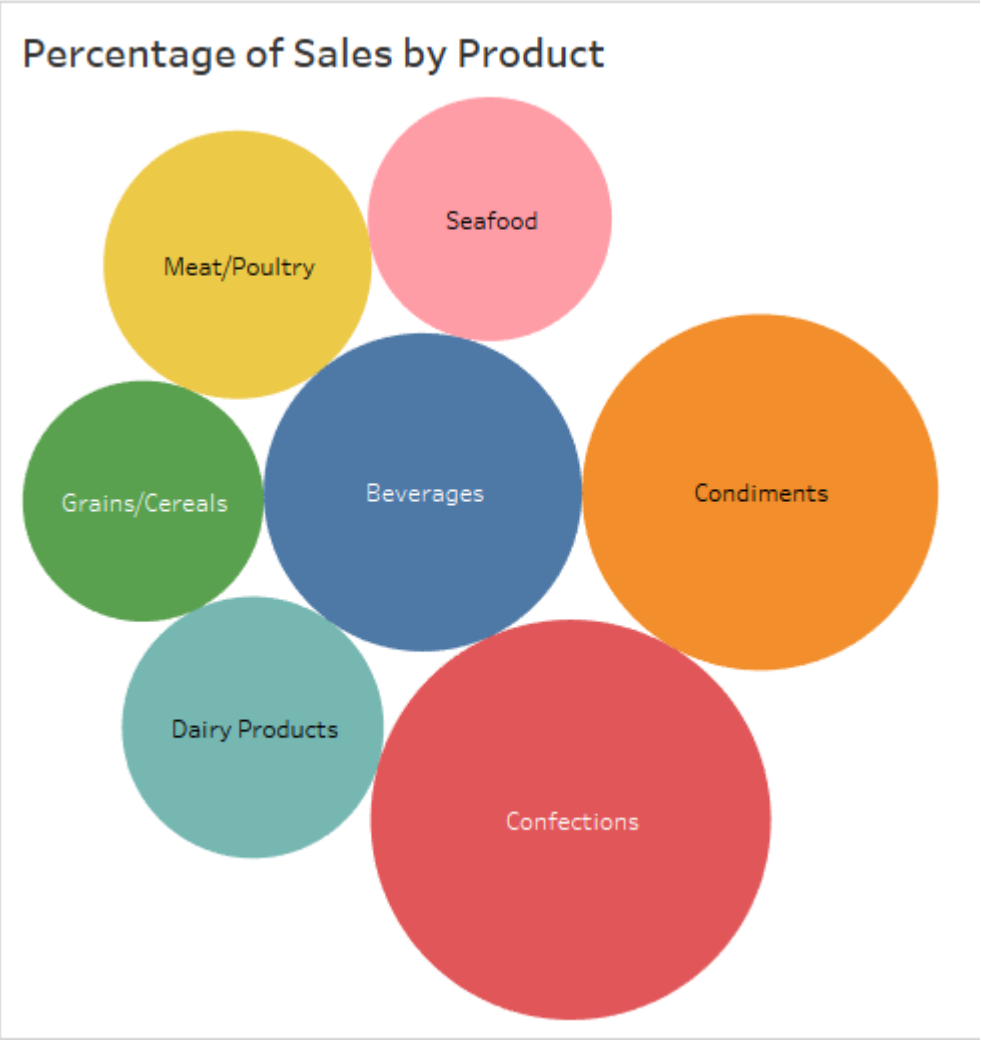
1. Grandma's Boysenberry Spread
2. Gustaf's Knäckebröd
3. Röd Kaviar
4. Genen Shouyu
5. Valkoinen suklaa
6. Queso Manchego La Pastora
7. Sasquatch Ale
8. Mishi Kobe Niku
9. Schoggi Schokolade
10. Laughing Lumberjack Lager
11. Sirop d'érable
12. NuNuCa Nuß-Nougat-Crème

Now as per our objective, we focus our sales on these products through strategic marketing such as BOGOF or by cross-selling the products of same categories (i.e., pairing the products that are doing terrible at sales with a highly sold product and selling it together at a discounted price).

4.3 Percentage of Sales of the poorly sold products by Product Category:

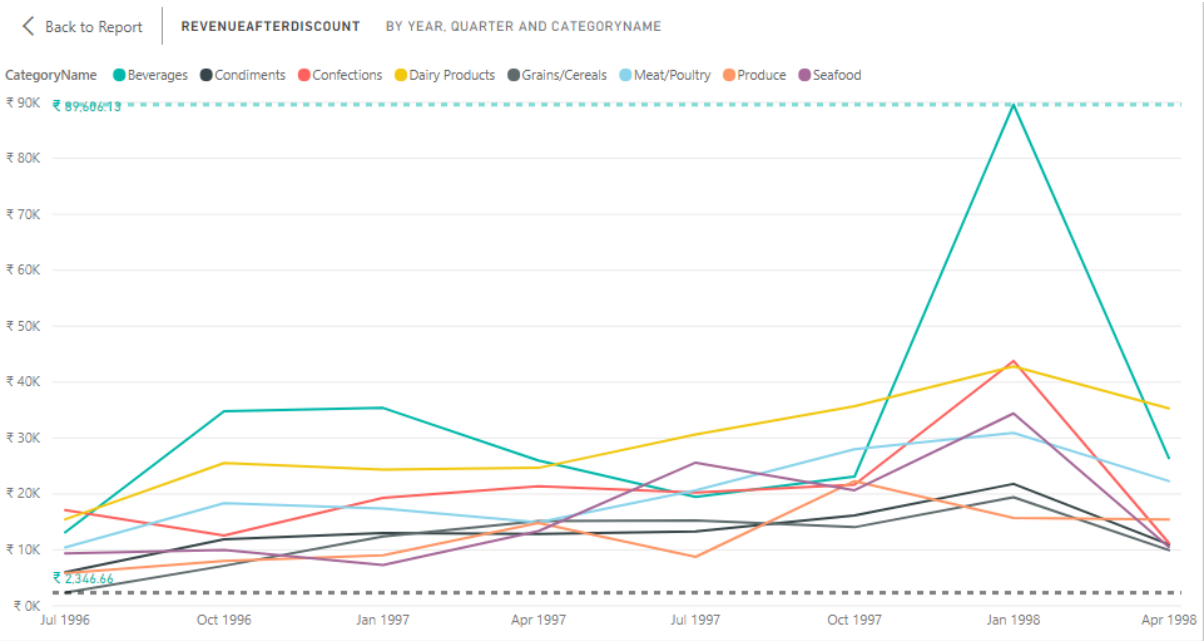
We group the segregated 12 products with poor sales record, category wise. We can see that Seafood ranks the least, while confections rank the highest among those 12 products.

1. Seafood – 12.17%
2. Confections – 32.74%



We can also see that the category Meat/Poultry ranks the second least, while the Condiments rank the second highest.

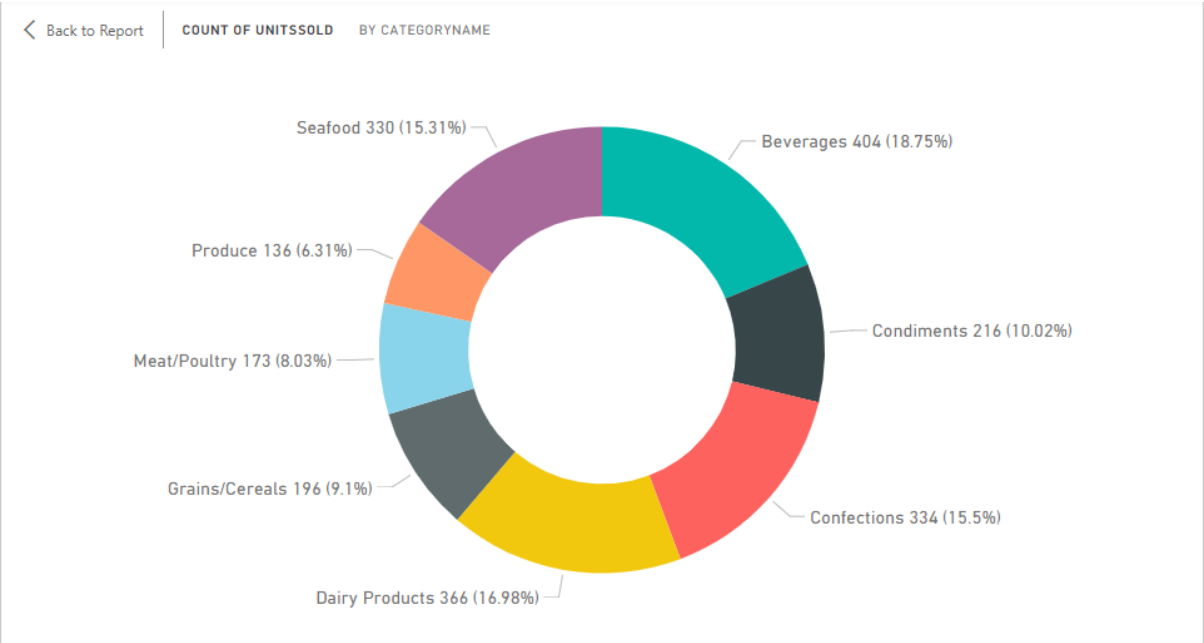
4.4 Sales Revenue by product category over a period of time:



From the above chart we can note the following two observations,

1. Beverages were sold the most among all categories during the period of Jan 1998 - $89,606
2. Grains/Cereals were sold the least of all categories during the period of Jul 1996 - $2,344

4.5 Count of Units Sold by Product Category



Products that were sold the most are

1. Beverages – 404 (18.75%)
2. Dairy Products – 366 (16.98%)

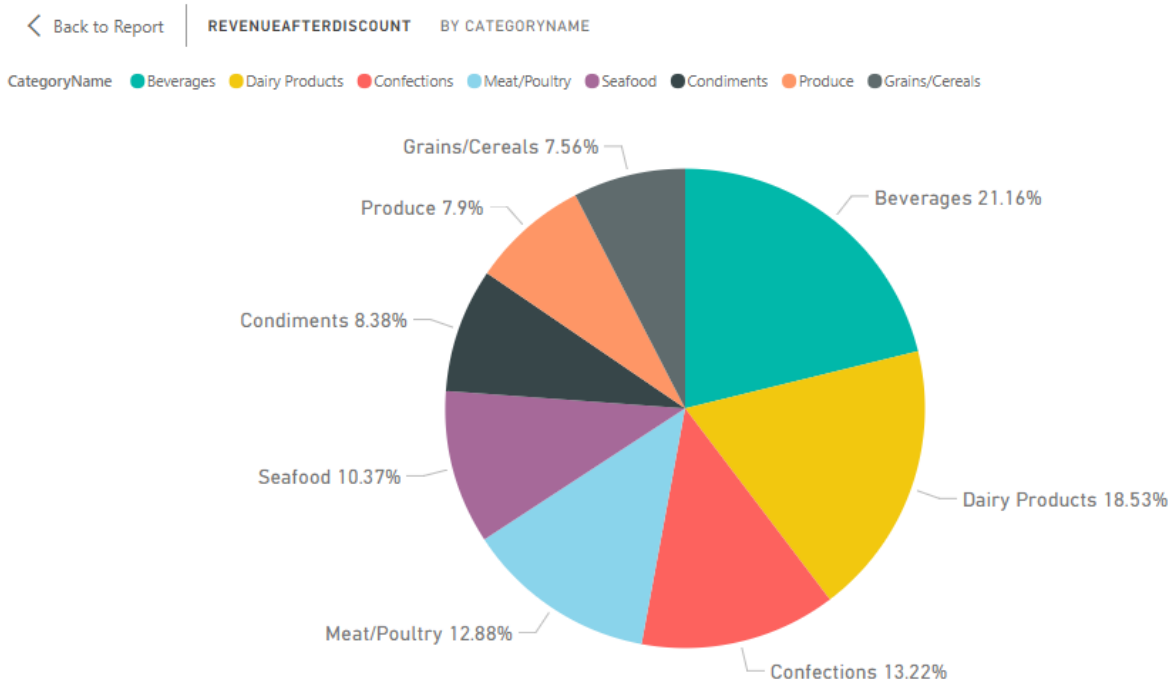
4.6 Revenue after Discount by Product Category

In the below chart, we can see that the product categories that generated the most revenue are

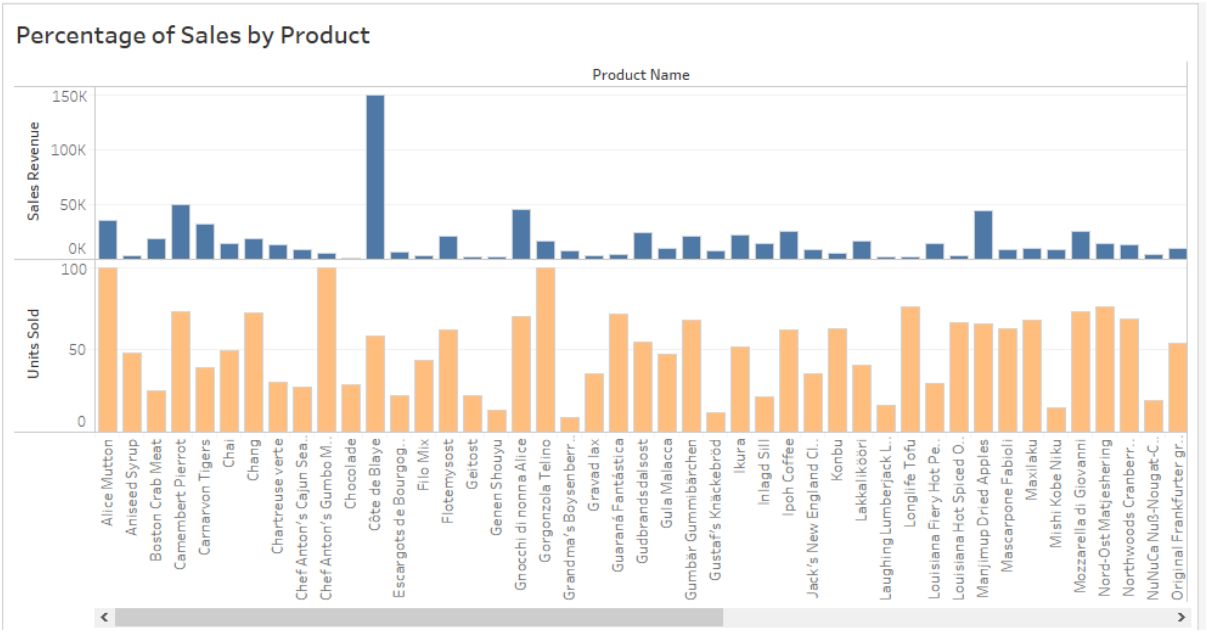
1. Beverages – 21.6%
2. Dairy Products – 18.53%

Similarly, the categories that generated the least revenue are

1. Grains/Cereals – 7.56%
2. Produce – 7.9%



4.7 Comparison of Products by Units Sold and Sales Revenue

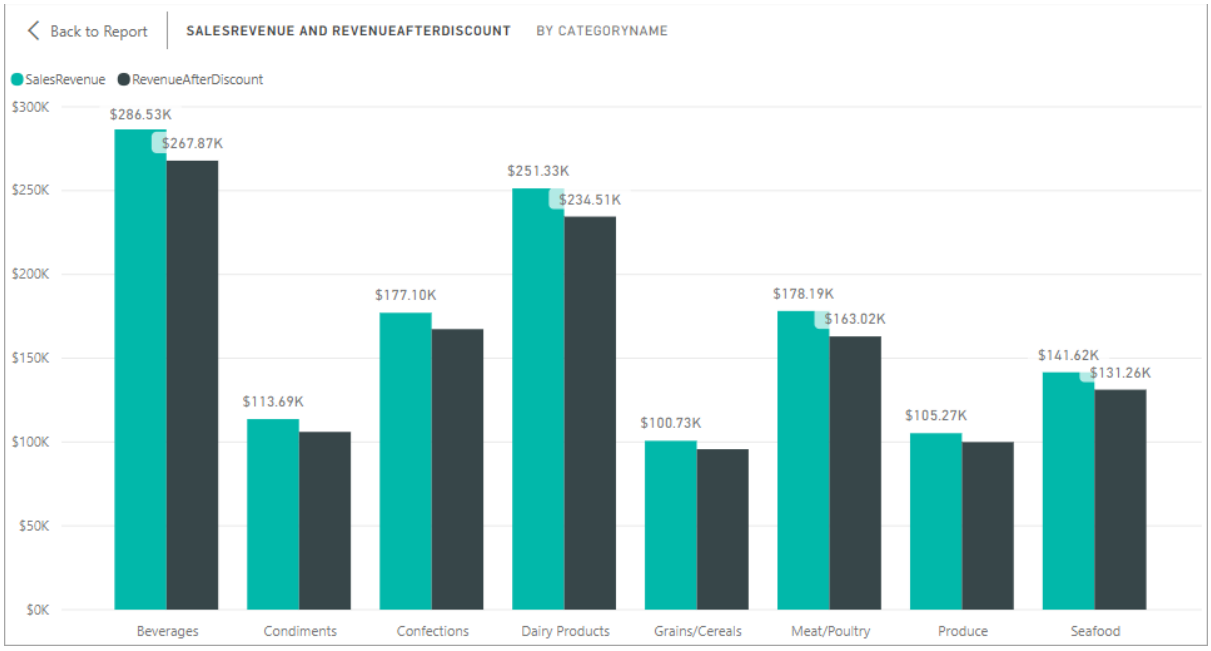


Here, we compare the count of units sold with the sales revenue for each of the products. Because there are certain products that can generate more revenue even though they are sold in less quantities,

while there are some that won't generate a significant revenue even though they are sold in huge quantities. For e.g.,

1. **Côte de Blaye** – 24 units sold; generated $149,984 revenue
2. **Guaraná Fantástica** – 51 units sold; generated $4,783 revenue

4.8 Comparison of Sales Revenue and Revenue after Discount



Since we are keen on providing discounts for products with poor sales record, it's essential to do an analysis on the losses incurred by discounts, though it's insignificant and benefits the organization by preventing from a significant loss.

The categories that incurred the highest and lowest loss are,

1. Beverages – (286,526.95 - 267,868.19) = $18,658.76 loss
2. Grains/Cereals – (100,726.80 – 95,744.58) = $4,982.22 loss

4.9 Reasons for selecting Tableau and PowerBI

- **Tableau:**
    - Data visualization is a key aspect in presenting data. A good UI is needed for visual representation of data. Tableau is one of the most widely used software tools for data visualization.
    - Ease of data import and export. It supports all major data sources. Numerous data sources like MySQL, SAP, JSON could be visualized in Tableau.
    - It provides support for advanced analytics and for languages like python and R.
    - Tableau provides one year of free license for students. It also has a nominal pricing of $35 per user per month and provides free trial for new users.
    - Supporting wide range of data sources and its flexibility in accessing data anywhere, anytime is one of the reasons why it is widely preferred.
    - Understanding data, creating reports, consolidating data in one place and complex software deployment are some of the major issues. Tableau handles all these with ease.
- **PowerBI:**
    - It can ingest data virtually from any source and is highly compressed.

- It possesses a brand-new visualization engine, which also has open source visualizations
- Easy to use than Tableau
- Can be accessed via cloud, mobile apps, embedded in custom apps, unlike Tableau.
- Comparatively cheaper than Tableau in pricing.
- It has a self-service BI, with dashboards for consolidation
- It has a natural language query engine and integrates seamlessly with PPT, R, Excel, etc. without having the need to export

Part 5 - Modelling and Graph Data Models

5.1 Limitations of ER model:

One of the drawbacks of using the relational entity relationship models is with its restrictions and specification checks. Also, due to these restrictions, some of the information or in-built data dimensional models would be not found or might be hidden in somewhere in the ER model. This limited access of data affects internally, sometimes it makes the platform not more user friendly. It also carries some of these limitations when representing entity relationship model on comparison with other data models (E.g., Relational Model etc.). Considering the complication in the data

manipulation in the ER model, mostly it is known for designing high level model.

5.2 Some of the other limitations are discussed below:

- The content of the ER model is mainly based on assumption, whereas it could be directly represented in a relation database.
- With semi structured data in the system, entity relationship and dimensional models are not evident enough to be readily represented in the relational form.
- The adequate changes to the data in many of the systems are basically non-trivial and “Important enough to warrant explicit specification”
- Extracting data in support to diagramming and design on RDMS (Relational database Management System) which clearly resembles the ER model in the existing database have been an ease with most of the abundant tools, so ER modelling has been rarely considered as a separate activity.

5.3 Limitations of star schema:

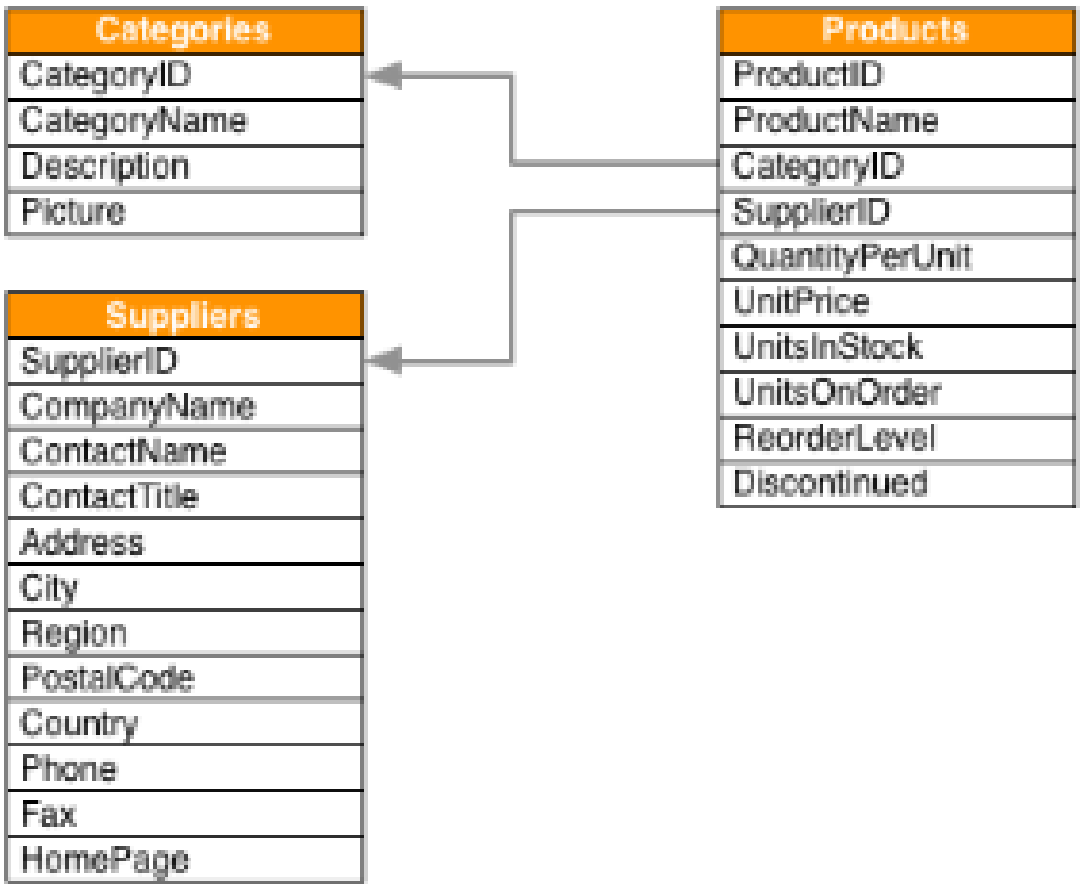
- **No reusability of Master Data**: Since the master data is a dimension table which is inside the cube, it cannot be reused.
- **Performance degradation:** The numeric characters gets processed much faster than Alpha Numeric characters, as all

the tables inside the cube contains Alpha Numeric values it degrades the performance of queries.

- **Limited Analysis:** Fact table can comprise a maximum of only 16 variables, as each column in the fact table connects to one dimensional table, so maximum no. of dimension tables that it can limit to is 16.

5.4 Implementing the Northwind ER model as a graph using Neo4j:

5.4.1 Product catalog: Northwind sells food products in a few categories, provided by suppliers. Let's start by loading the product catalog tables.



Loading records:

```
LOAD CSV WITH HEADERS FROM "http://data.neo4j.com/northwind/products.csv" AS row
CREATE (n:Product)
SET n = row,
  n.unitPrice = toFloat(row.unitPrice),
  n.unitsInStock = toInteger(row.unitsInStock), n.unitsOnOrder = toInteger(row.unitsOnOrder),
  n.reorderLevel = toInteger(row.reorderLevel), n.discontinued = (row.discontinued <> "0")
```

```
LOAD CSV WITH HEADERS FROM "http://data.neo4j.com/northwind/categories.csv" AS row
CREATE (n:Category)
SET n = row
```

```
LOAD CSV WITH HEADERS FROM "http://data.neo4j.com/northwind/suppliers.csv" AS row
CREATE (n:Supplier)
SET n = row
```

Creating indexes:

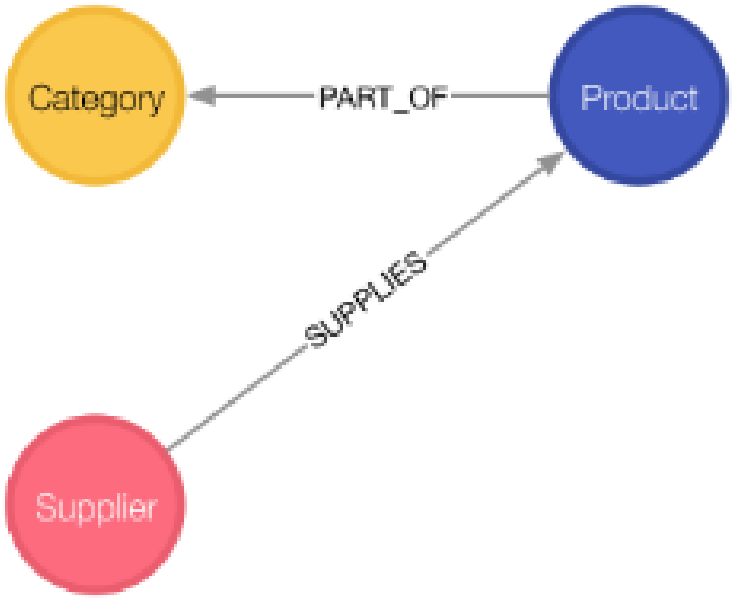
```
CREATE INDEX ON :Product(productID)
```

```
CREATE INDEX ON :Category(categoryID)
```

```
CREATE INDEX ON :Supplier(supplierID)
```

Product Catalog Graph:

The suppliers, products and categories are interlinked through foreign key references. Let's relate those to data relationships to realize the graph.

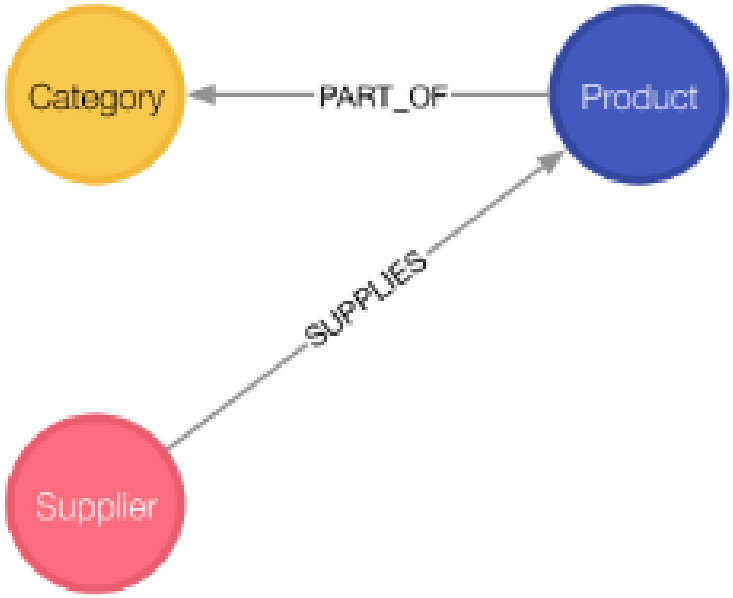


Creating data relationships:

```
MATCH (p:Product),(c:Category)
WHERE p.categoryID = c.categoryID
CREATE (p)-[:PART_OF]->(c)
```

```
MATCH (p:Product),(s:Supplier)
WHERE p.supplierID = s.supplierID
CREATE (s)-[:SUPPLIES]->(p)
```

Querying Product Catalog Graph:



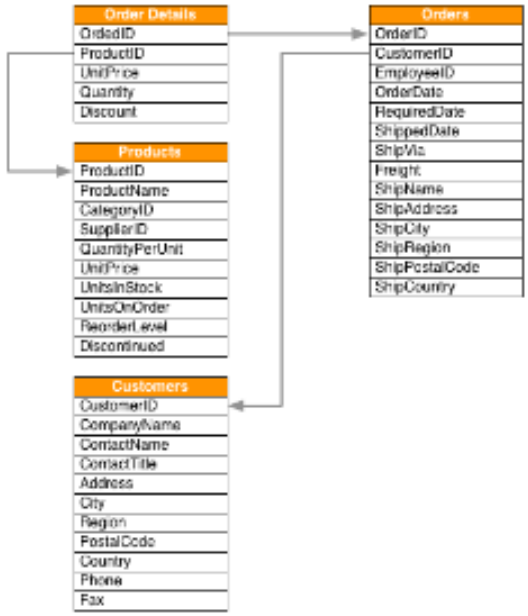
Query using patterns:

```
MATCH (s:Supplier)-->(:Product)-->(c:Category)
RETURN s.companyName as Company, collect(distinct c.categoryName) as Categories
```

```
MATCH (c:Category {categoryName:"Produce"})<--(:Product)<--(s:Supplier)
RETURN DISTINCT s.companyName as ProduceSuppliers
```

Customer Orders:

Northwind customers place orders which may detail multiple products.



Load and index records:

```
LOAD CSV WITH HEADERS FROM "http://data.neo4j.com/northwind/customers.csv" AS row
CREATE (n:Customer)
SET n = row
```

```
LOAD CSV WITH HEADERS FROM "http://data.neo4j.com/northwind/orders.csv" AS row
CREATE (n:Order)
SET n = row
```

```
CREATE INDEX ON :Customer(customerID)
```

```
CREATE INDEX ON :Order(orderID)
```

Create data relationships:

```
MATCH (c:Customer),(o:Order)
WHERE c.customerID = o.customerID
CREATE (c)-[:PURCHASED]->(o)
```

Customer Order Graph:

The Order Details are always part of an Order that is related to a product. We'll directly relate each Order Detail record into a relationship in the graph

Load and index records:

```
LOAD CSV WITH HEADERS FROM "http://data.neo4j.com/northwind/order-details.csv" AS row
MATCH (p:Product), (o:Order)
WHERE p.productID = row.productID AND o.orderID = row.orderID
CREATE (o)-[details:ORDERS]->(p)
SET details = row,
  details.quantity = toInteger(row.quantity)
```

Query using patterns:

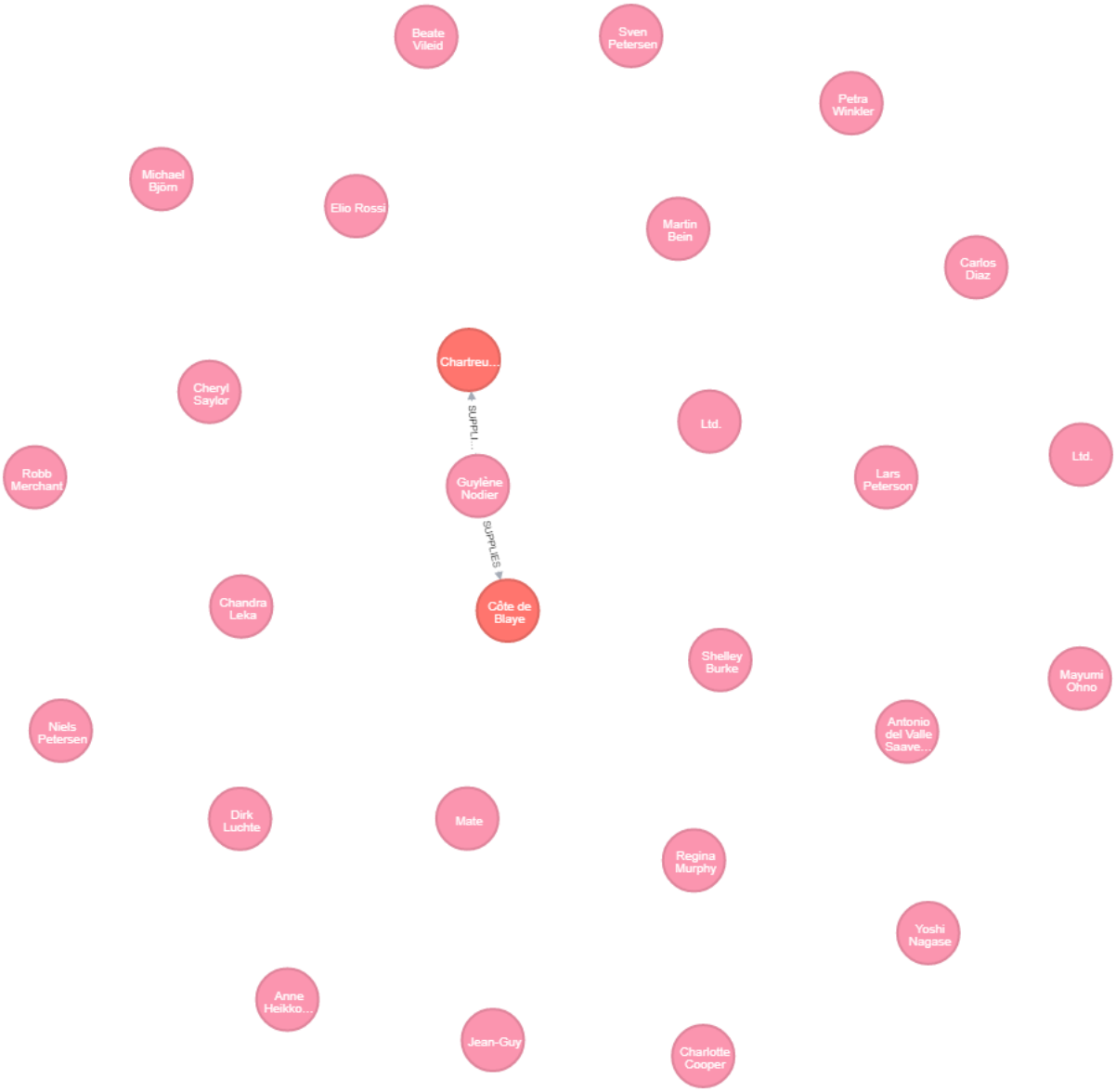
```
MATCH (cust:Customer)-[:PURCHASED]->(:Order)-[o:ORDERS]->(p:Product),
      (p)-[:PART_OF]->(c:Category {categoryName:"Produce"})
RETURN DISTINCT cust.contactName as CustomerName, SUM(o.quantity) AS TotalProductsPurchased
```

Developing a Graph Model:

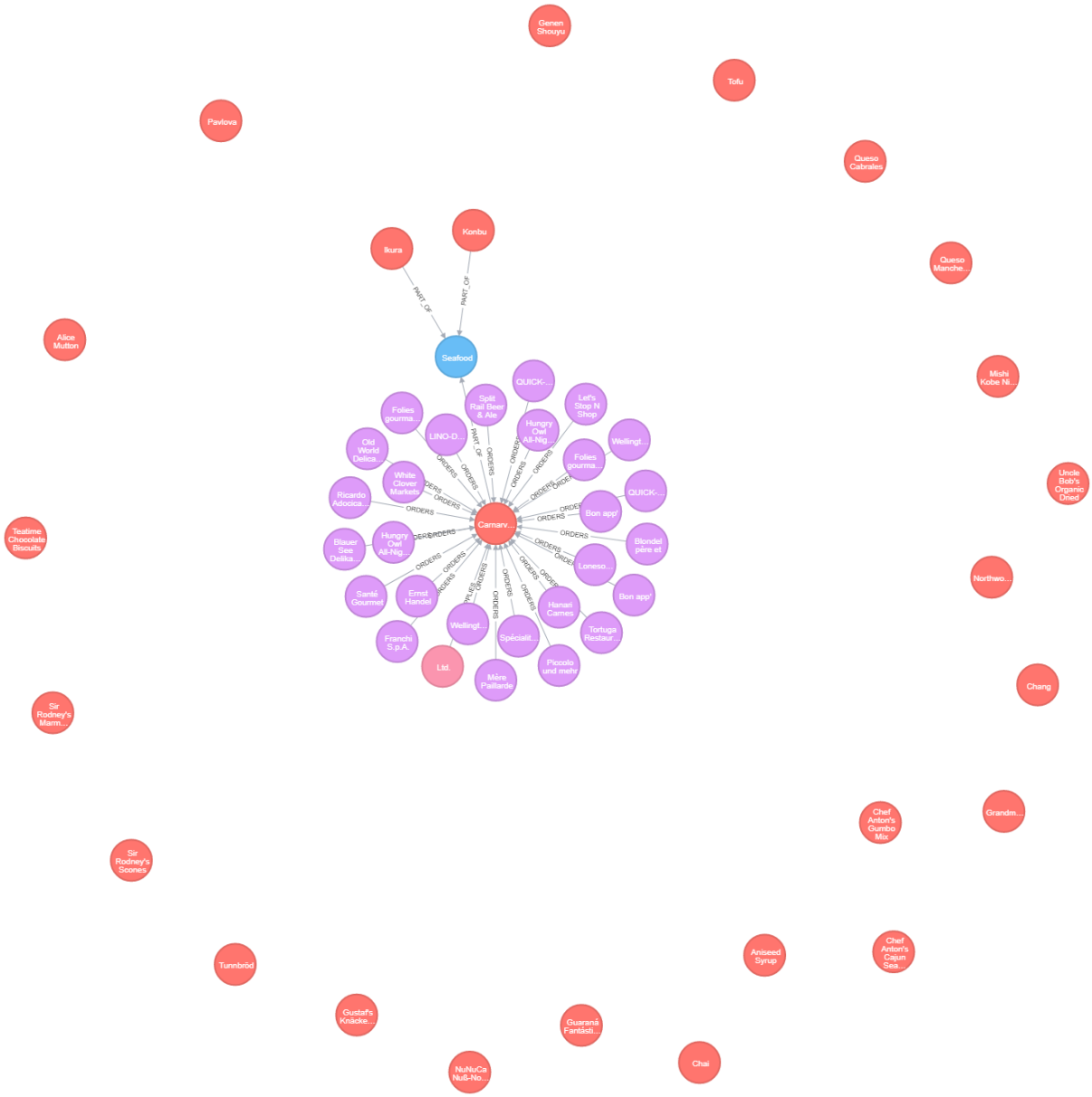
We are deriving a graph model from a relational model, we need to take care of the following guidelines in mind:

- Node is a row
- Label name is a table name

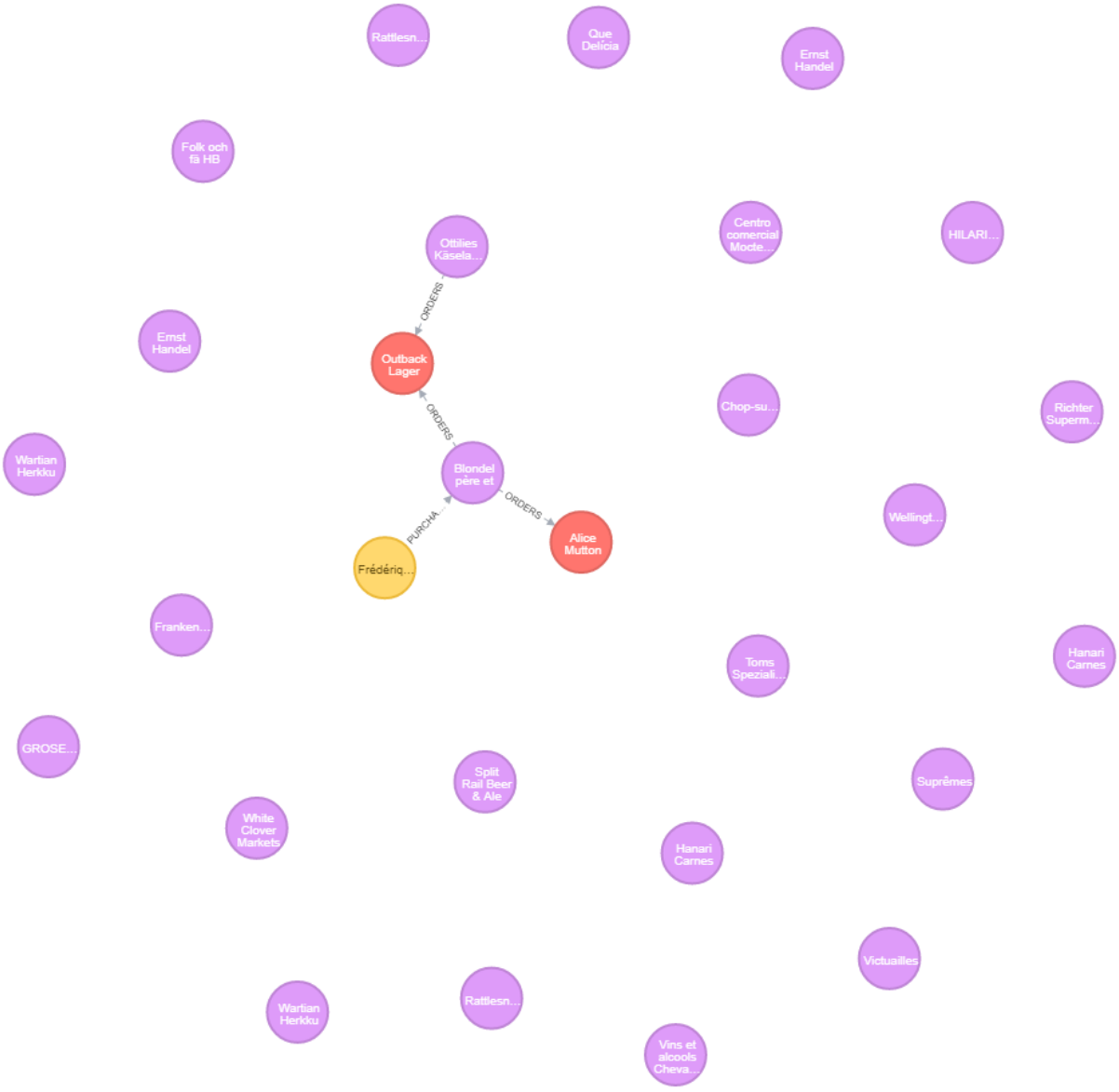
Supplier nodes:



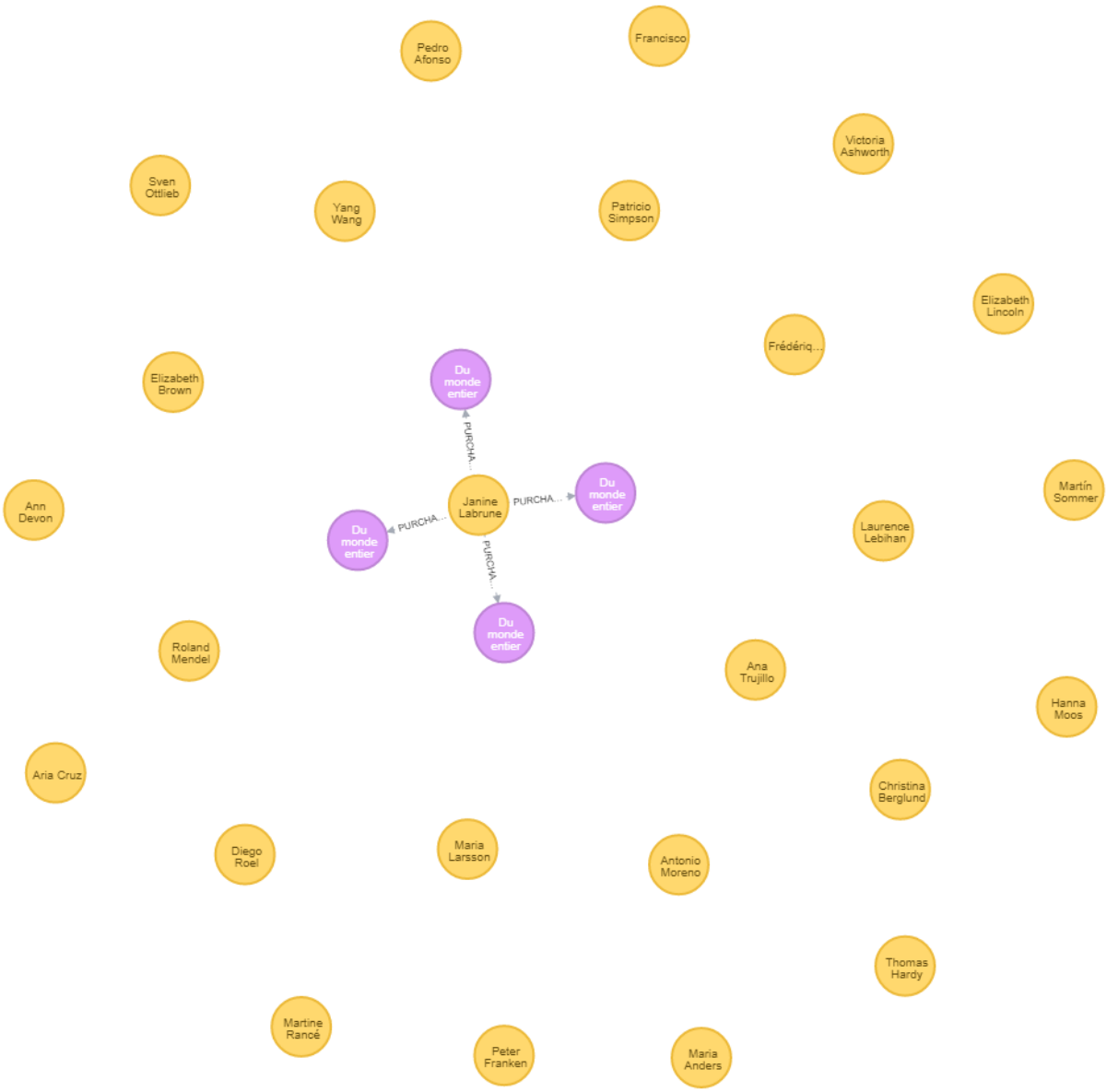
Product nodes



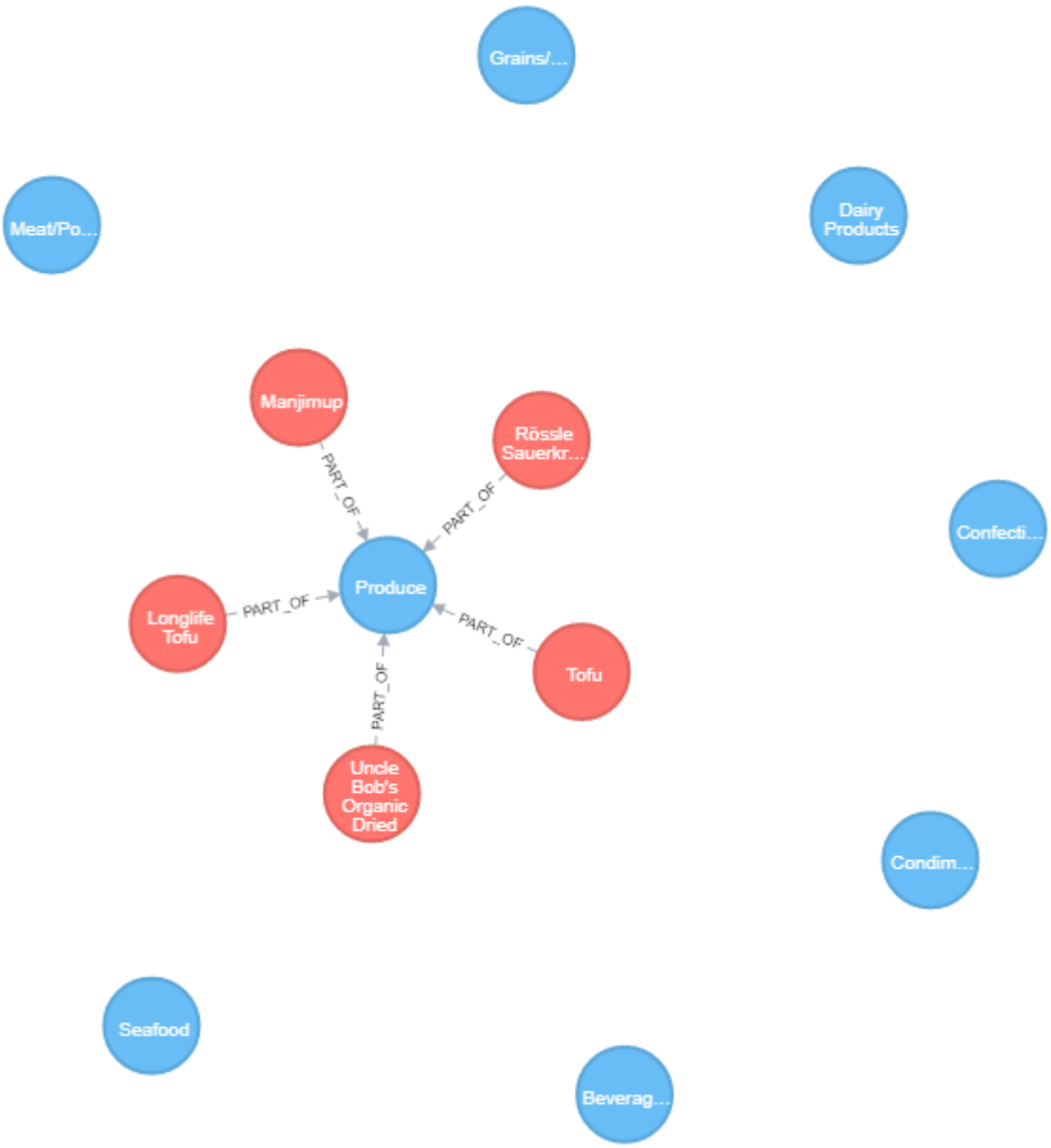
Order nodes


Rattlesn...
Que Delicia
Ernst Handel
Folk och fä HB
Ottilies Käsela...
Centro comercial Mocte...
HILARI...
ORDERS
Ernst Handel
Outback Lager
ORDERS
Chop-su...
Richter Superm...
Wartian Herkku
Blondel père et
ORDERS
PURCHA...
Wellingt...
Alice Mutton
Frédériq...
Franken...
Toms Speziali...
Hanari Carnes
GROSE...
Split Rail Beer & Ale
Suprêmes
White Clover Markets
Hanari Carnes
Victuailles
Wartian Herkku
Rattlesn...
Vins et alcools Cheva...

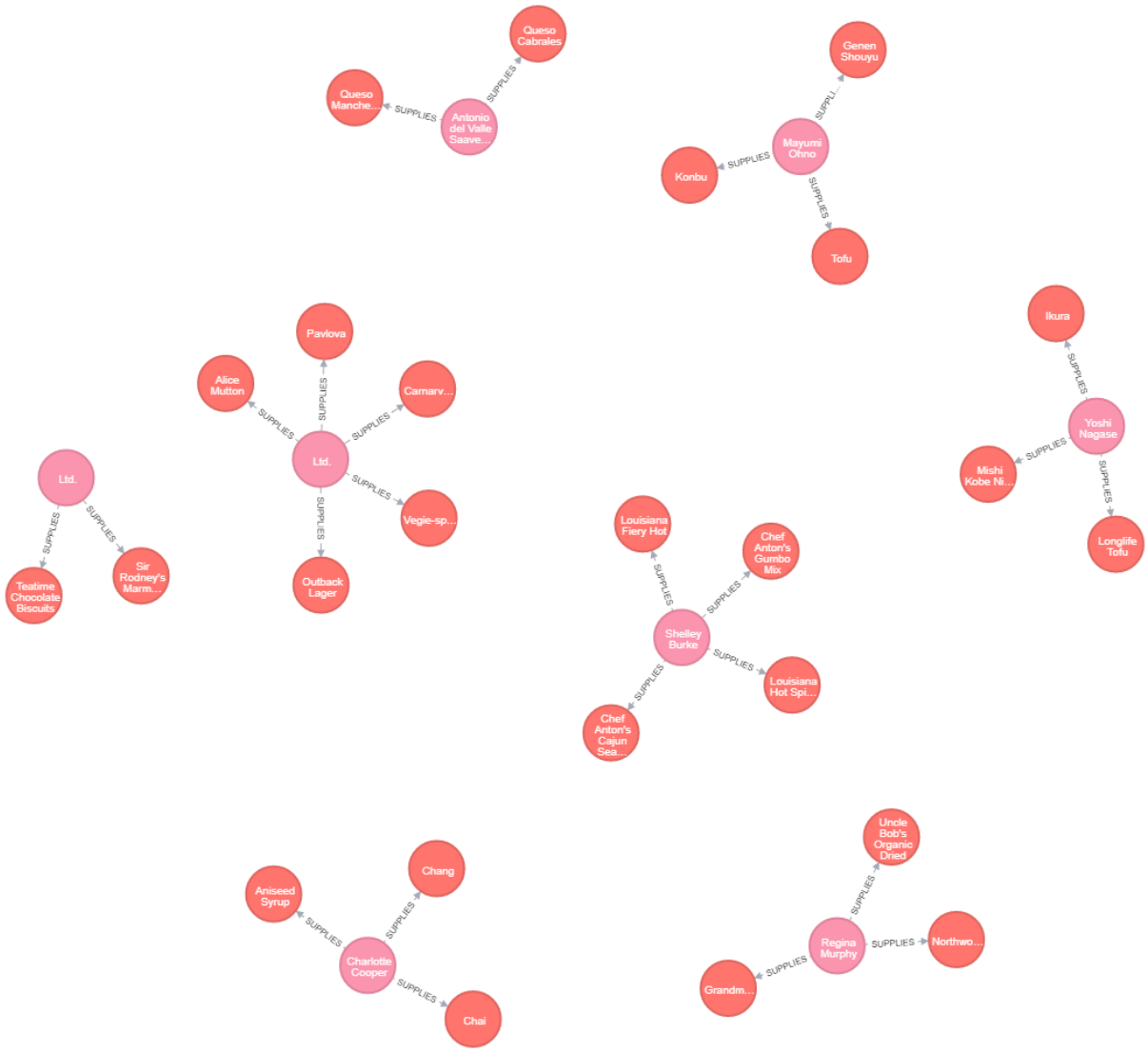
Customer nodes:



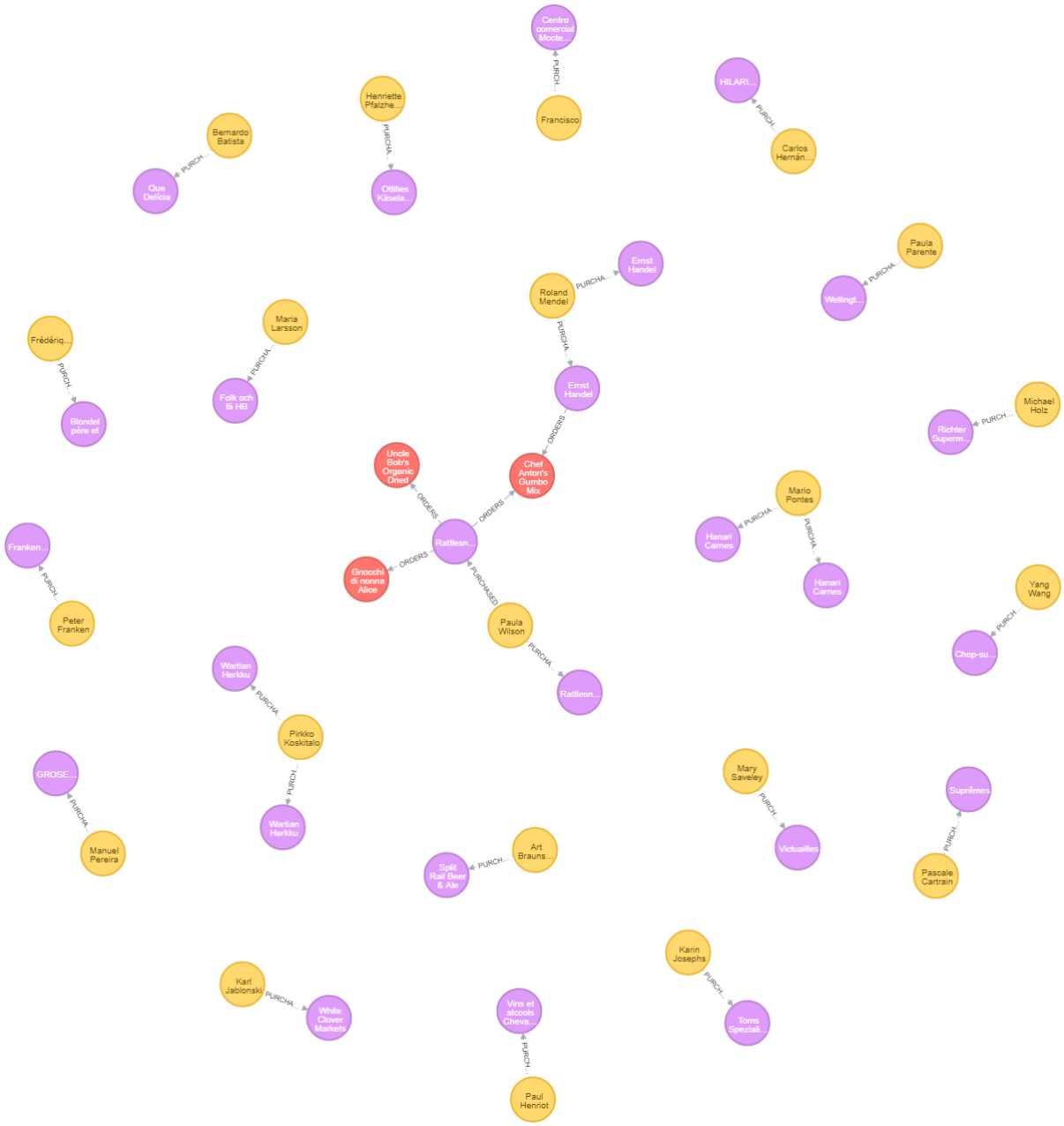
Category nodes:



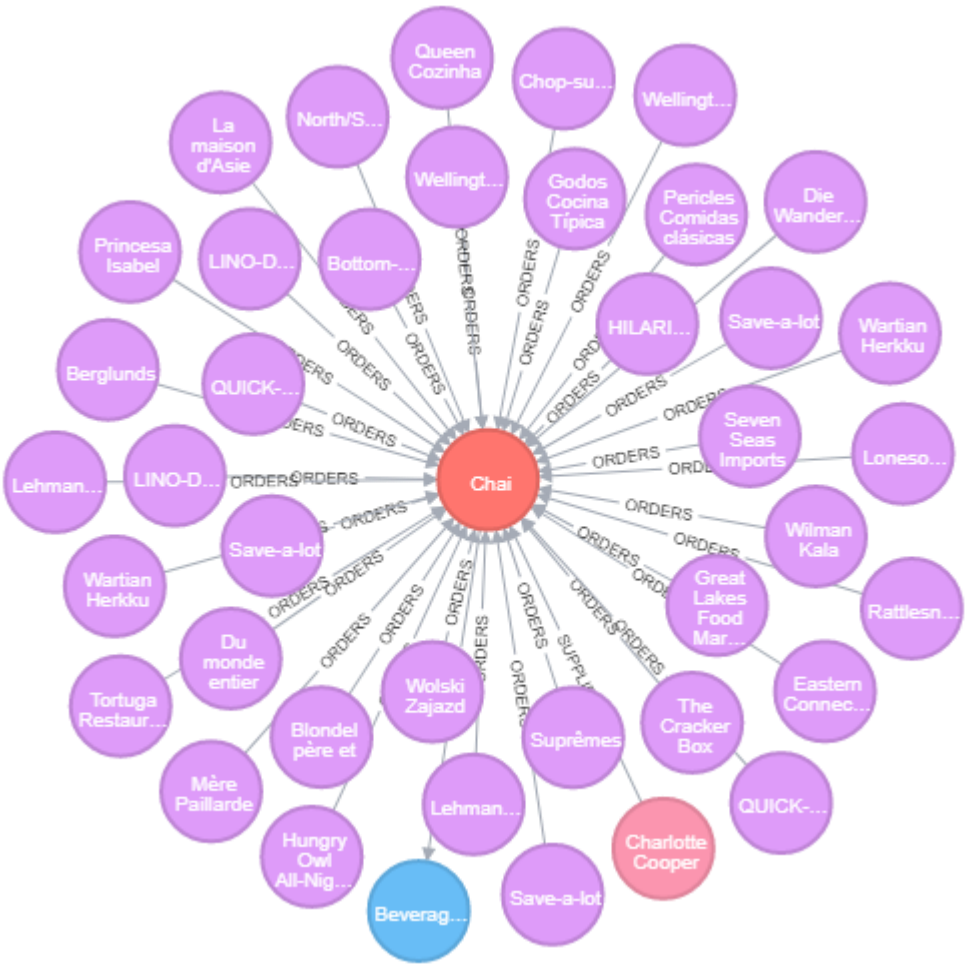
Suppliers relationship:



Purchased relationship:



Order relationship:



Part 6 - Graph information retrieval and analysis:

The pivotal moment in the evolution of graph data space is with creation of open Cypher. The graph processing and analysis accelerates its usage worldwide as the data storage, analytics or tooling platform becomes easier which offers access to graph capabilities using query language which is universally accepted, and this is same as SQL did for relational databases.

The data stored in RDBMS can be revealed through connections with the concept called Neo4j ETL and Neo4j helps to have a good initial and an ongoing experience as it is moved into it via Neo4j ETL.

With Neo4j ETL, the below-stated points are possible:

- Before execution, it is possible to map or modify the labels and properties.
- The data can be fed through CSV files like how the data is exported out of it.
- Altering values can be accomplished using a GUI.
- Finding JOIN and JOINs as graph relationships and adjusting structures as labelled nodes using Neo4j ETL.

As data is imported via relational joins and materialized with graph connections, there is a persistence of permanent data after the import.

Neo4j Data Lake Integrator:

It is also quite challenging when it comes to wrangling and adding its contents, analytical delivery and operational use cases where as it is easy to fill in to the data lake and moreover finding and utilizing the connections in the data is next to impossible. The values which are surfaced by Neo4j Data Lake Integrator and merged into data lakes makes it possible to read, prepare and interpret data in memory and the Neo4j graph platform enables the persisted graph to be used in real time with transactional applications and graph analytics exercises. The enterprises which are been helped through data lake integrator are:

- **Build Metadata Catalogs**: To weave the highly connected view of information in the data lakes is done through discovering data object definitions and their relationships with each other. The, the resulted meta data graph would help in exploring the data, compliance initiatives and impact analysis.
- **Wrangle Data**: The data from the data lake and from the other sources with Neo4j has been wrangled in with cypher which is the SQL for the graphs. Therefore, using Apache Spark, the data which is memory graph format composed through Cypher queries could be returned. In tabular or CSV formats.
- **Perform Graph Analytics**: Algorithms for graph such as PageRank, path finding and community detection to infer new

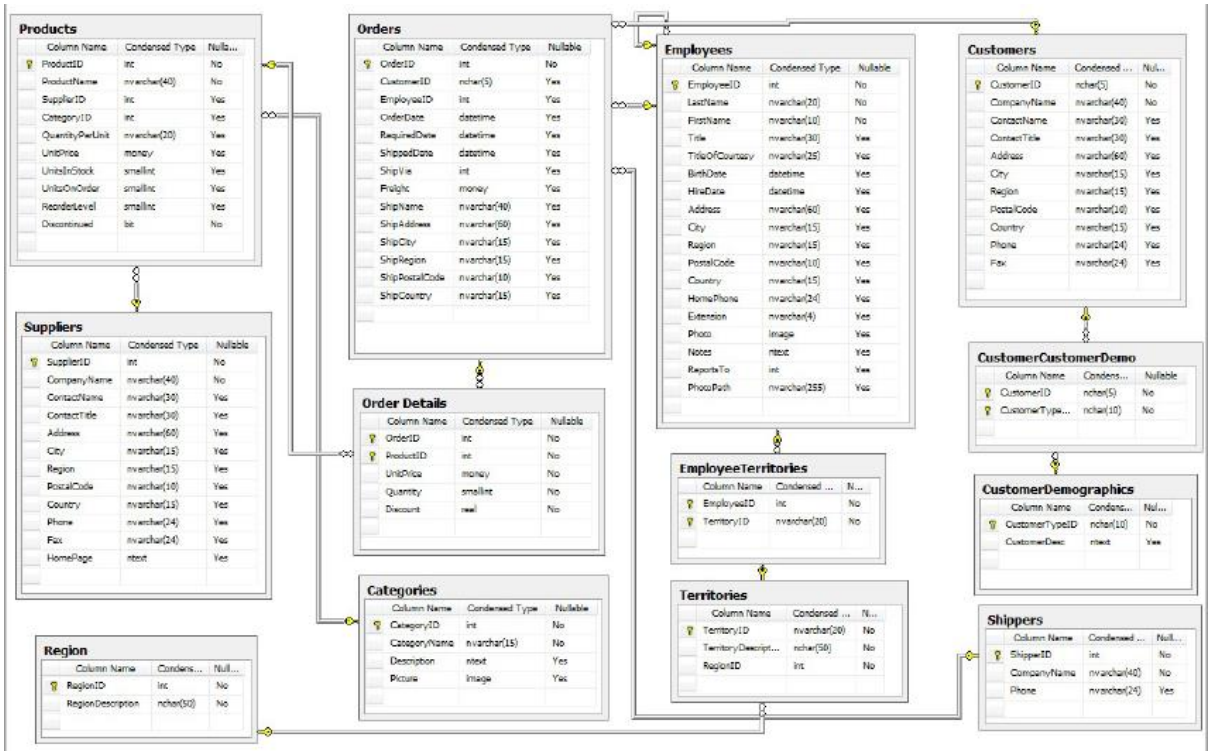
insights could be done by directly importing the data from the data lake into Neo4j graph platform to achieve faster and intuitive graph analytics.

- **Operationalize Data**: For real time transactional and traversal applications the data can be imported directly from the data lake into the Neo4j graph platform.
- **Snapshot Graphs**: The Neo4j graph data is properly reconstituted in structured CSV as the output obtained from the data lake integrator. These files can be saved as snapshots, versioned, diffed, reused and backed up in HDFS

Northwind Introduction:

Northwind dataset which has been used by Microsoft to feature some of its products. NorthWind dataset has been used here to demonstrate SQL and relational databases and the graphical representation makes it more interesting.

Entity Relationship diagram for NorthWind dataset is,



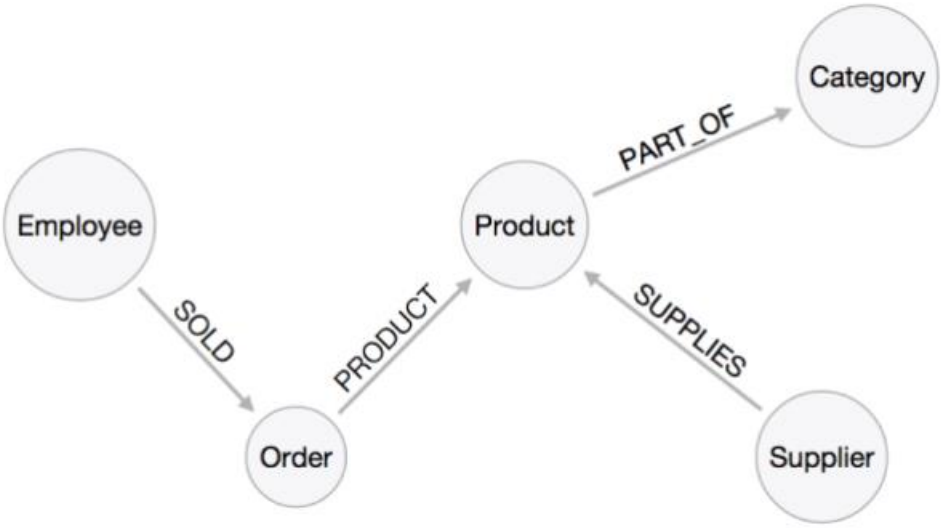
How does the Graph Model Differ from the Relational model:

- Basically, there are no nulls.
- In the relational version, tracking the employee hierarchy is done by having a null entry in the ‘ReportsTo’ column if they don’t report to anybody. And in the graph version, we just don’t define a relationship.
- Non-existing value entries (properties) are just not present.
- Relationships are described in more detail. For example, we know that an employee SOLD an order rather than having a foreign key relationship between the Orders and Employees tables. We could also choose to add more metadata about that relationship should we wish.
- It will often be more normalized when compared to each other.

Developing a Graph Model:

The guidelines to be noticed while deriving a graph model from relational are

- Node is a row
- Label name is a table name



Northwind Graph:

Using a dataset, converting a relational database management system into Graph

The demonstration in changing a RDBMS to Neo4j is done through Northwind Graph. By emphasizing the shift from relational tables to nodes and relationships of a graph is through iterations and transformations.

- **Load**: Data is loaded externally from CSV files
- **Index**: Label is created based on index nodes

- **Relate**: Data relationships are altered using foreign key reference.
- **Promote**: Altering join records in relationships

Product catalog:

Northwind basically sells food products in a few categories, provided by suppliers. Here we start loading the product catalog tables.

The load statements to the right require public internet access. LOAD CSV will retrieve a CSV file from a valid URL, applying a Cypher statement to each row using a named map (here we're using the name row).

Load records:

```
LOAD CSV WITH HEADERS FROM "http://data.neo4j.com/northwind/products.csv" AS row
CREATE (n:Product)
SET n = row,
  n.unitPrice = toFloat(row.unitPrice),
  n.unitsInStock = toInt(row.unitsInStock), n.unitsOnOrder = toInt(row.unitsOnOrder),
  n.reorderLevel = toInt(row.reorderLevel), n.discontinued = (row.discontinued <> "0")
LOAD CSV WITH HEADERS FROM "http://data.neo4j.com/northwind/categories.csv" AS row
CREATE (n:Category)
SET n = row
LOAD CSV WITH HEADERS FROM "http://data.neo4j.com/northwind/suppliers.csv" AS row
CREATE (n:Supplier)
SET n = row
```

Create indexes:

```
CREATE INDEX ON :Product(productID)
CREATE INDEX ON :Category(categoryID)
CREATE INDEX ON :Supplier(supplierID)
```

Product catalog graph:

The products, categories and suppliers are related through foreign key references.

Create data relationships:

```
MATCH (p:Product),(c:Category)
WHERE p.categoryID = c.categoryID
CREATE (p)-[:PART_OF]->(c)
```

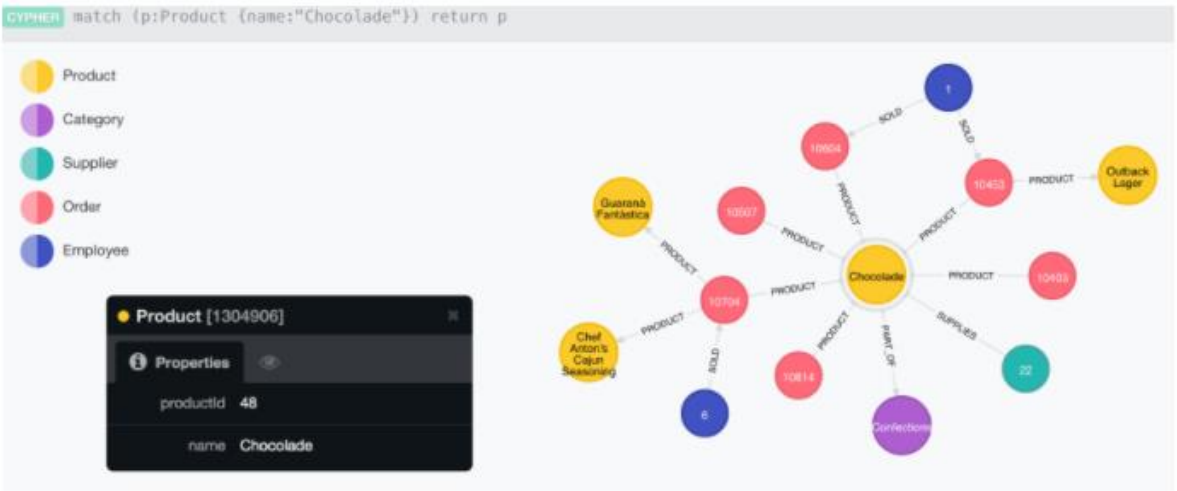
We need to compare property values like this when first creating relationships

Calculate join, materialize relationship.

```
MATCH (p:Product),(s:Supplier)
WHERE p.supplierID = s.supplierID
CREATE (s)-[:SUPPLIES]->(p)
```

Query product catalog graph:

Let’s try some queries using patterns.



Query using patterns:

List the product categories provided by each supplier

```
MATCH (s:Supplier)-->(:Product)-->(c:Category)
RETURN s.companyName as Company, collect(distinct c.categoryName) as Categories
MATCH (c:Category {categoryName:"Produce"})<--(:Product)<--(s:Supplier)
RETURN DISTINCT s.companyName as ProduceSuppliers
```

Find the produce suppliers.

employee.employeeId	other.productName	count
1	Pavlova	56
1	Camembert Pierrot	56
1	Ikura	55
1	Chang	47
1	Pâté chinois	45

Looking at this, we can say that the employee no.1 was very busy.

Who Reports to Whom? How are Employees Organized?

```
MATCH path = (e:Employee)<-[:REPORTS_TO]-(sub)
RETURN e.employeeID AS manager, sub.employeeID AS employee;
```

manager	employee
2	1
2	3
2	4
2	5
2	8
5	6
5	7
5	9

Notice that employee No. 5 has people reporting to them but also reports to employee No. 2.

Let's go with that a bit more:

Which Employees Report to Each Other Indirectly?

```
MATCH path = (e:Employee)<-[:REPORTS_TO*]-(sub)
WITH e, sub, [person in NODES(path) | person.employeeID][1..-1] AS
path
RETURN e.employeeID AS manager, sub.employeeID AS employee, CASE WHEN
LENGTH(path) = 0 THEN "Direct Report" ELSE path END AS via
ORDER BY LENGTH(path);
```

e.EmployeeID	sub.EmployeeID	via
2	1	Direct Report
2	3	Direct Report
2	4	Direct Report
2	5	Direct Report
2	8	Direct Report
5	6	Direct Report
5	7	Direct Report
5	9	Direct Report
2	6	[5]
2	7	[5]
2	9	[5]

How Many Orders were Made by Each Part of the Hierarchy?

```
MATCH (e:Employee)
OPTIONAL MATCH (e)<-[:REPORTS_TO*0..]-(sub)-[:SOLD]->(order)
RETURN e.employeeID, [x IN COLLECT(DISTINCT sub.employeeID) WHERE x <>
e.employeeID] AS reports, COUNT(distinct order) AS totalOrders
ORDER BY totalOrders DESC;
```

e.EmployeeID	reports	totalOrders
2	[1,3,4,5,6,7,9,8]	2155
5	[6,7,9]	568
4	[]	420
1	[]	345
3	[]	321
8	[]	260
7	[]	176
6	[]	168
9	[]	107

Updating the Graph:

Now if we need to update our graph data, we need to first find the relevant information and then update or extend the graph structures.

Janet is now reporting to Steven

We need to find Steven first, and Janet and her `REPORTS_TO` relationship. Then we remove the existing relationship and create a new one to Steven.

```
MATCH (mgr:Employee {EmployeeID:5})
MATCH (emp:Employee {EmployeeID:3})-[rel:REPORTS_TO]->()
DELETE rel
CREATE (emp)-[:REPORTS_TO]->(mgr)
RETURN *;
```

It is to be noted that you only need to compare property values like this when first creating relationships

Role of graph databases in the Hadoop-Spark Ecosystem:

Apache Spark is a clustered, in-memory data processing solution that scales processing of large datasets easily across many machines. It also comes with GraphX and GraphFrames two frameworks for running graph compute operations on your data. You can integrate with Spark in a variety of ways. Either to pre-process (aggregate, filter, convert) your raw data to be imported into Neo4j.

Spark can also serve as external Graph Compute solution, where you

- Export data of selected subgraphs from Neo4j to Spark
- Compute the analytic aspects
- Write the results back to Neo4j
- To be used in your Neo4j operations and Cypher queries.

Neo4j-Spark-Connector:

The Neo4j Spark Connector uses the binary Bolt protocol to transfer data from and to a Neo4j server. It offers Spark-2.0 APIs for RDD, DataFrame, GraphX and GraphFrames, so you're free to choose how you want to use and process your Neo4j graph data in Apache Spark. Spark Config options. The general usage is:

1. create org.neo4j.spark.Neo4j(sc)
2. set cypher(query,[params]),nodes(query,[params]),rels(query,[params]) as direct query, or
   pattern("Label1",Seq("REL"),"Label2") or pattern(
   ("Label1","prop1"),("REL","prop"),("Label2","prop2"))
3. optionally define partitions(n), batch(size), rows(count) for parallelism
4. choose which data type to return
   - loadRowRdd, loadNodeRdds, loadRelRdd, loadRdd[T]
   - loadDataFrame,loadDataFrame(schema)
   - loadGraph[VD,ED]
   - loadGraphFrame[VD,ED]

Neo4j-Mazerunner:

An interest in analytical graph processing led Kenny Bastani to work on an integration solution. It allows to export dedicated datasets, e.g. node or relationship-lists to Spark. It supports these algorithms:

- PageRank
- Closeness Centrality
- Betweenness Centrality
- Triangle Counting
- Connected Components
- Strongly Connected Components

After running graph processing algorithms, the results are written back concurrently and transactionally to Neo4j.

One focus of this approach is on data safety, that's why it uses a persistent queue (RabbitMQ) to communicate data between Neo4j and Spark.

The infrastructure is set up using Docker containers, there are dedicated containers for Spark, RabbitMQ, HDFS and Neo4j with the Mazerunner Extension.

Spark for data processing:

One example of pre-processing raw data (Chicago Crime dataset) into a format that's well suited for import into Neo4j, was demonstrated by Mark Needham. He combined several functions into a Spark-job that takes the existing data, cleans and aggregates it and outputs fragments which are recombined later to larger files.

Cypher for Apache Spark:

"Cypher for Apache Spark enables the execution of cypher queries on property graphs stored in an Apache Spark cluster in the same way that SparkSQL allows for the querying of tabular data". Cypher queries as a more programmatic API are enabled by the system with an ability to run the queries and working with graphs which inspired by the Apache Spark API.

The support for working with multiple graphs and query composition is done with first implementation method of Cypher called as CAPS. The queries can dynamically construct new graphs, access multiple graphs and return the query result. In addition, the results would also act as an input to a follow up query. Complex data processing

pipelines across multiple heterogeneous data sources which are constructed incrementally is also enabled with it.

An extensible API for integrating additional data sources for loading and storing graphs is provided by CAPS. Loading the graphs from HDFS (Hadoop distributed file system), session local storage and the bolt protocol is also supported by CAPS. Scope for integrating further technologies at both the data source level as well as the API level in the future is also done. CAPS has been the first open source implementation of cypher on a relationship big data processing system. Set of scan tables are used for representation of property graphs which corresponds to all the nodes with a label or with a type of relationship. To do this, a new schema model for describing the labels, relationship types and keys that occur in a graph is efficiently achieved. Bringing up cypher's capabilities to work with heterogeneous data to the apache spark eco system plays a vital role with this schema.

References:

[1]. R. Kimball, “Kimball DW/BI Lifecycle Methdology” [Online]. Available: https://www.kimballgroup.com/data-warehouse-business-intelligence-resources/kimball-techniques/dw-bi-lifecycle-method [Accessed: May. 04, 2018].

[2] Kimball, R (1996). Data Warehouse Toolkit – Practical Techniques for Building Dimensional Data Warehouse, New York: Wiley & Sons.

[3] Neo4j [Online]. Available: https://neo4j.com/ [Accessed: May. 04, 2018]

[4] GitHub Repository [Online]. Available: https://github.com/neo4j-contrib/neo4j-graph-algorithms [Accessed: May. 04, 2018]

[5] SQL Server – Data Warehousing [Online]. Available:

https://www.microsoft.com/en-us/sql-server/data-warehousing [Accessed: May. 04, 2018]