DSO 552: SQL Databases for Business Analysts

Homework 4

Northwind Traders Company

Northwind Traders is a company that imports and exports food globally. The database captures all the sales transactions that occurs between the company i.e. Northwind traders and its customers as well as the purchase transactions between Northwind and its suppliers.

The following explains each table (used in this assignment) in the Northwind database:

Table	**Description**
Customers	who buy from Northwind
Orders	stores transaction sale orders from customers
OrderDetails	stores line items of sale orders
Products	the products that Northwind trades in
Suppliers	who supply to the company
Shippers	details of the shippers who ship the products from the traders to the end-customers
Employees	who work for Northwind

Check Figure 1 (on the last page) for more details for each of the above tables.

Make sure that you are logged in as your own personal user and that you save all objects into your own personal schema (the same as your username).

1. ***(1 point)* Create a table called `sms_events` that will store SMS text messages received by the shipping company. The first column should be titled `id` and be an `INTEGER`. The second column should be `from` and should be a `TEXT` type, and the third field should be `message_body`, and should contain `TEXT`. The last column should be an `updated_at` field of type `TIMESTAMP`.**

```sql
CREATE TABLE sms_events
(
    id           INT,
    "from"       TEXT,
    message_body TEXT,
    updated_at   TIMESTAMP
);
```

How we will grade: we will attempt to insert 3 rows into this table with the specified data types.

2. ***(1 point)* Create a table called `high_freight_orders` that contains all the columns from the `public.orders` table that have a `freight` value higher than 30.**

```sql
CREATE TABLE high_freight_orders AS
SELECT *
FROM public.orders
WHERE freight > 30
```

How we will grade: We will run the below query and expect to see the following result (substitute `ychen220` with your own schema):

```
SELECT COUNT(*) total_orders, COUNT(DISTINCT customerid) AS total_customers
FROM ychen220.high_freight_orders
```

Expected:

total_orders	total_customers
483	84

3. ***(1 point)*** **Create a table called `customers_adapted` in your schema that is a copy of all the columns and rows from the `customers` table in the `public` schema. Then, insert two new customers (up to you to determine their name, company name, etc.). The only requirement is that these two customers are from the city of London.**

```
CREATE TABLE IF NOT EXISTS customers_adapted
SELECT * FROM public.customers
```

```
INSERT INTO ychen220.customers_adapted (customerid, companyname,
contactname, contacttitle, address, city, region,
                                        postalcode,
                                        country, phone, fax)
VALUES
('DADKD', 'Bing Bong', 'Yu Chen',
'Intern', NULL, 'London', NULL,
NULL, NULL, NULL, NULL),
('AADKD', 'Ping Pong', 'You Chen',
'Assistant to the Vice President', NULL, 'London', NULL, NULL, NULL, NULL, NULL);
```

How we will grade: We will run the below query and expect to see the following result (substitute `ychen220` with your own schema):

```
SELECT city, COUNT(DISTINCT customerid)
FROM ychen220.customers_adapted
GROUP BY 1
HAVING city = 'London'
```

Expected:

city	count
London	9

4. ***(1 points)*** **Using a common table expression to compute the average number of line items per order**

```
WITH line_items_per_order AS (SELECT o.orderid,
                                     COUNT(od.*)
                                         FROM orders o JOIN order_details od
                              ON od.orderid = o.orderid
                              GROUP BY 1)
SELECT AVG(count) AS average_line_items_per_order
FROM line_items_per_order
```

Expected:

average_line_items_per_order
2.596385

5. *(1 points)* **Management is concerned we are giving too many discounts to customers. They classify a high discount order as any order that contains a line item that has a discount value higher than the average discount across all line items. Using a common table expression to list the top five employees in terms the number of high discount orders they have processed.**

```
WITH average_discount AS (SELECT AVG(discount) FROM order_details)
SELECT e.employeeid, COUNT(DISTINCT o.orderid) orders_with_high_discounts
FROM customers c
         JOIN orders o ON c.customerid = o.customerid
         JOIN order_details od ON od.orderid = o.orderid
         JOIN employees e ON e.employeeid = o.employeeid
WHERE discount > (SELECT * FROM average_discount)
GROUP BY 1
ORDER BY 2 DESC
LIMIT 5;
```

Expected:

employeeid	orders_with_high_discounts
4	56
3	42
8	42
1	39
7	29

6. *(1 points)* **Create a view called `most_recent_shipped_orders` that contains the 10 most recent orders shipped.**

```
CREATE OR REPLACE VIEW most_recent_shipped_orders AS
SELECT *
FROM orders
WHERE orders.shippeddate IS NOT NULL
ORDER BY orders.shippeddate DESC
LIMIT 10;
```

How we will grade: We will run the below query and expect to see the following result:

```
SELECT orderid, customerid, employeeid, orderdate, shipvia, shipname, shippeddate
FROM ychen220.most_recent_shipped_orders;
```

Expected:

orderid	customerid	employeeid	orderdate	shipvia	shipname	shippeddate
11067	DRACD	1	1998-05-04	2	Drachenblut Delikatessen	1998-05-06
11069	TORTU	1	1998-05-04	2	Tortuga Restaurante	1998-05-06
11063	HUNGO	3	1998-04-30	2	Hungry Owl All-Night Grocers	1998-05-06
11050	FOLKO	8	1998-04-27	2	Folk och fä HB	1998-05-05
11055	HILAA	7	1998-04-28	2	HILARION-Abastos	1998-05-05
11066	WHITC	7	1998-05-01	2	White Clover Markets	1998-05-04
11060	FRANS	2	1998-04-30	2	Franchi S.p.A.	1998-05-04
11022	HANAR	9	1998-04-14	2	Hanari Carnes	1998-05-04
11049	GOURL	3	1998-04-24	1	Gourmet Lanchonetes	1998-05-04
11064	SAVEA	1	1998-05-01	1	Save-a-lot Markets	1998-05-04

7. ***(1 points)*** **Create a table called `new_regions` that is a copy of the `region` table. Then, insert into this table, `Northeast` region (with an ID of 5) and the `Southwest` region with an ID of 6.**

```
CREATE TABLE IF NOT EXISTS new_regions
(
    regionid          SMALLINT,
    regiondescription BPCHAR
);
```

```
INSERT INTO ychen220.new_regions (regionid, regiondescription)
VALUES
(5, 'Northeast'),
(6, 'Southwest');
```

How we will grade: We will run the below query and expect to see the following result:

```
SELECT *
FROM ychen220.new_regions
```

Expected:

regionid	regiondescription
1	Eastern
2	Western
3	Northern
4	Southern
5	Northeast
6	Southwest

8. **(1 points) Create a materialized view called `most_popular_orders` the product names that have been ordered the most times (in terms of quantity) and have above average `unitprice`.**

```
CREATE MATERIALIZED VIEW IF NOT EXISTS most_popular_orders AS
SELECT p.productname,
       sum(od.quantity) AS total_times_ordered
FROM products p
         JOIN public.order_details od ON od.productid = p.productid
GROUP BY p.productname
HAVING avg(od.unitprice) > ((SELECT avg(products.unitprice) AS avg
                             FROM products))
ORDER BY (sum(od.quantity)) DESC;
```

How we will grade: We will run the below query and expect to see the following result:

```
SELECT * FROM
ychen220.most_popular_orders LIMIT 5;
```

Expected:

productname	total_times_ordered
Camembert Pierrot	1577
Raclette Courdavault	1496
Gnocchi di nonna Alice	1263
Tarte au sucre	1083
Alice Mutton	978

9. **(1 points) Create a copy of the `order_details` and `orders` tables in your own schema (named the same as in `public`). Then delete all records in your schema's order details table that belong to `Camembert Pierrot`.**

How we will grade: We will run the below query and expect to see the following result:

```
SELECT COUNT(*)
FROM ychen220.order_details od
JOIN products p ON p.productid = od.productid
WHERE p.productname = 'Alice Mutton'
```

Expected:

count
0

10. **(1 points) Create a view called `late_orders_by_month` that returns the average number of monthly late shipments.**

```
CREATE OR REPLACE VIEW late_orders_by_month(month, late_orders) AS
SELECT date_trunc('month'::TEXT, orders.orderdate::TIMESTAMP WITH TIME ZONE)::DATE AS month,
       count(*)                                                                      AS late_orders
FROM orders
WHERE orders.shippeddate > orders.requireddate
GROUP BY (date_trunc('month'::TEXT, orders.orderdate::TIMESTAMP WITH TIME ZONE)::DATE)
ORDER BY (date_trunc('month'::TEXT, orders.orderdate::TIMESTAMP WITH TIME ZONE)::DATE);
```

How we will grade: We will run the below query and expect to see the following result:

```
SELECT AVG(late_orders) FROM ychen220.late_orders_by_month;
```

Expected:

avg
2.055556

Note that to help you visualize what `late_orders_by_month` should be this is the following result of the view:

month	late_orders
1996-07-01	1
1996-08-01	2
1996-09-01	2
1996-10-01	1
1996-12-01	1
1997-01-01	2
1997-02-01	2
1997-03-01	1
1997-04-01	1
1997-05-01	2
1997-06-01	1
1997-07-01	2
1997-09-01	3
1997-10-01	2
1997-11-01	3
1997-12-01	3
1998-01-01	4
1998-03-01	4

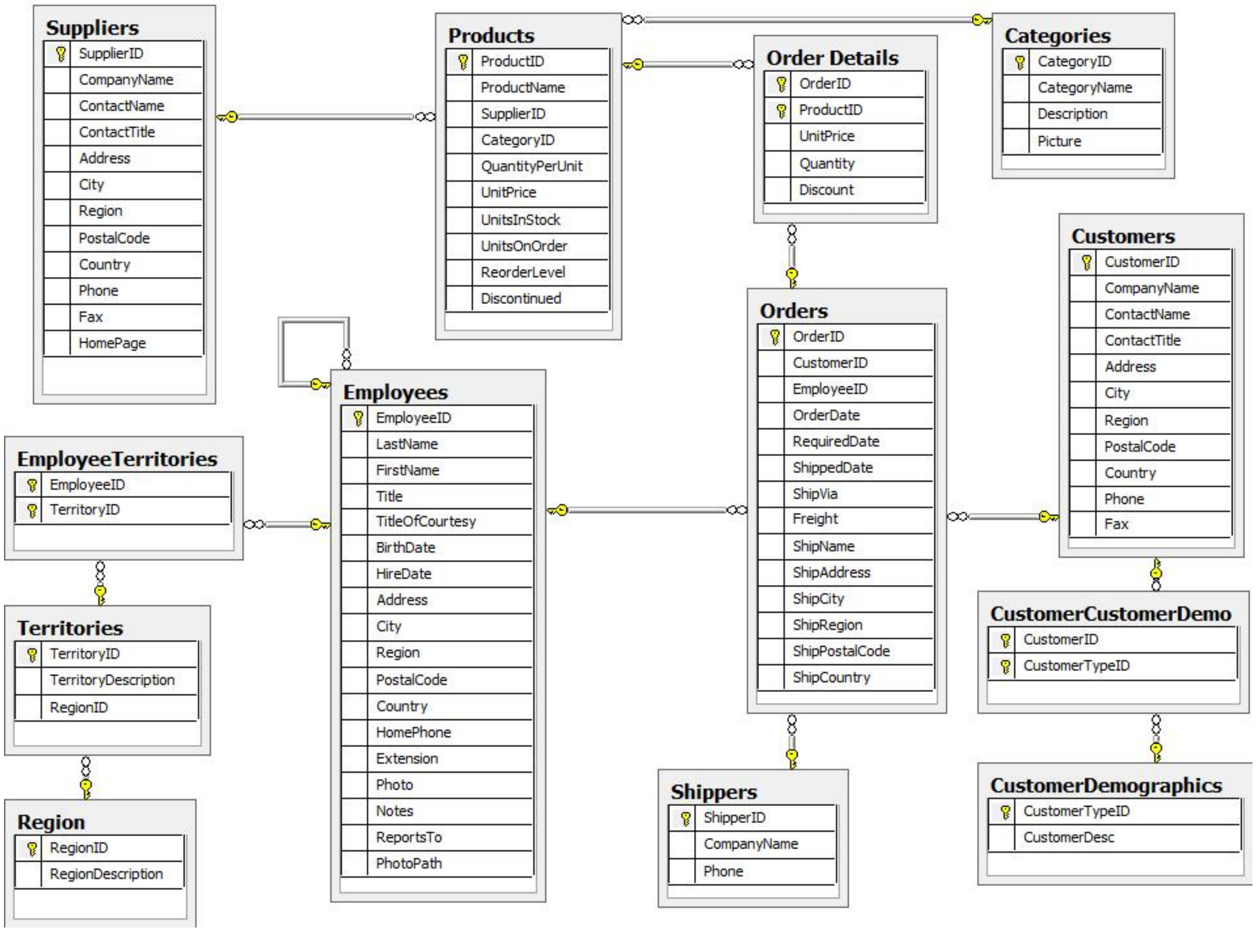

Figure 1: Northwind ERD