DATA IN MOTION

SQL CASE STUDY
TINY SHOP

MEBU Manuella Kevine

A project proposed by KEDEISHA BRYAN

09/06/2023

1 TABLE

1 TINY SHOP SALES

1.1 Audit

We assume that, we did an audit at TINY SHOP; let us write a technical report on the field of study, and validate it with the client.

1.2 Technical report

A customer orders several items (or an item) from TINY SHOP, he/she receives a bill, the bill is on the customer's name and it is taken in charge by the customer. The bill is made up order date, customer's personal information, product name, order id, product id, price, total price, quantity bought and total quantity bought… etc. TINY SHOP sends the product to the customer after payment.

1.3 Organizational flow model (OFM)

We bring out the different actors in the field of study, both internal actors and external actors and what they share.

Internal actor: Shop

External actor: Customer

Exchange (flow): Bill (invoice), product

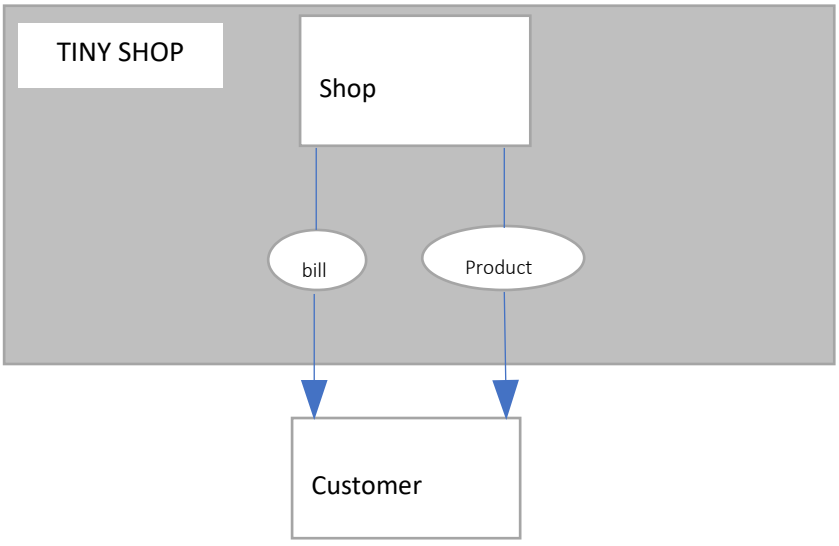


Figure 1: TINY SHOP OFM

1.4 Data dictionary

From the OFM and technical report, in each exchange, we fish each information that is shared between the actors.

N°	Wording	Example	P/C	
1	Order id	Order_0023	P	Primary
2	quantity	5 socks, 5 jeans	P	Primary
3	Total quantity	10 products	C	Computable
4	Price	1 000 F CFA	P	Primary
5	Total price	10 000 F CFA	C	Computable
6	Shop	TINY SHOP	P	Parameter
7	Order date	23/08/1996	P	Primary
8	Product id	socks_OO123	P	Primary
9	Product name	Socks GIVENCHY	P	Primary
10	Customer id	Cust_0003A	P	Primary
11	First name	Uriel	P	Primary
12	Last name	Gabriel	P	Primary
13	email	uriel.gabriel@aset.com	P	Primary

Tableau 1: TINY SHOP Data dictionary

Headings are like containers and what they take in is referred to as a content. Order_id is a container, and Order_OO23 is the content.

Computable headings can be derived from primary columns. No need to add it to our model.

Parameter headings will not change, and even if it does, the old name is substituted by the recent name. More to that, our field of study is only for TINY SHOP and not several shops.

We are left with a total of 10 headings.

1.5 Conceptual computerized data model (CCDM)

Create the different classes of entities from the above data dictionary. We dispatch the above headings into groups and attribute each group a name, this name is called “class of entities”.

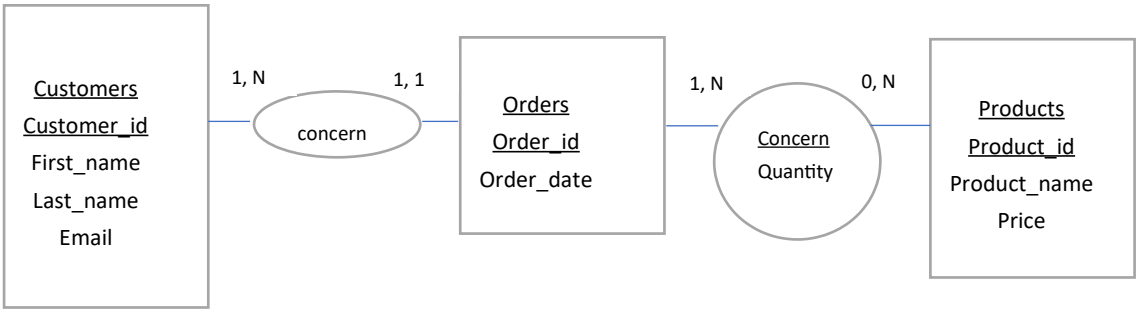


Figure 2: TINY SHOP CCDM

Entity Classes

- Customers
- Orders
- Products

Associations

- Concern

A customer is concerned at least by one order and at most by several orders.

An order is concerned at least by one customer and at most by one customer.

An order is concerned at least by a product and at most by several products.

A product is concerned at least by no order and at most several orders.

An association can carry a heading in it, if and only if the cardinalities on both sides of the association have a maximum relationship of N, example: the association **concern** between **Orders** and **Products**.

1.6 Data logic model (DLM)

It is also called a database relationship diagram (DRD). We transform entity classes to tables, multiple cardinalities on both sides of the association are transformed to tables, foreign keys are brought up in this part.

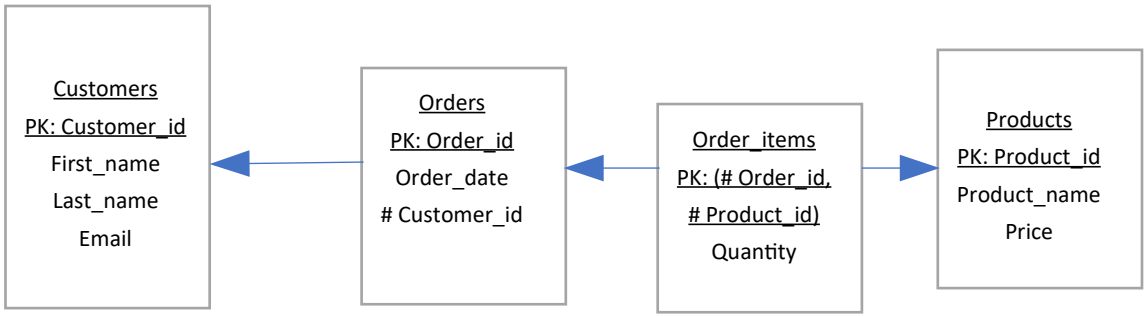


Figure 3: TINY SHOP DATA logic model (DLM)

PK stands for primary key; it is a unique identification for each entity or records in the table.

stands for foreign Key (Child table), it references a primary column in another table (Parent table).

The direction of the arrow (functional dependence) illustrates that, the table **Orders** has a foreign key (visual clue is a #) **Customer_id** which references the table **Customers.**

The association **concern** (the one which carries a heading in it) is replaced by a table and a name **Order_items**. It contains a composed primary key: Order_id and Product_id and they are both foreign keys in the **Order_items** table. These foreign Keys reference both the tables: **Orders** and **Products**.

Customers: (Customer_id, First_name, Last_name, Email)

Order: (Order_id, Order_date, # Customer_id)

Products: (Product_id, Product_name, Price)

Order_items: (# Order_id, # Product_id, Quantity)

1.7 Physical data model

This is quite similar to a data logic model, but it is specific to the relational database management system (RDBMS) used to generate the final model, thus the relational database.

Here we talk about data integrity constraint, the data type for each column and which RDBMS are we going to use to stock all the data for usage.

We use Microsoft SQL Server and the language structured query language (SQL), to generate the relational database.

When creating the different tables, we start by creating all the parent tables, lastly, the child tables.

1.7.1 DATABASE WITH MS SQL SERVER WITH THE LANGUAGE SQL

```
USE TINYSHOP

GO

DROP TABLE IF EXISTS customers
CREATE TABLE customers
(
customer_id int PRIMARY KEY,
first_name varchar(100),
last_name varchar(100),
email varchar(100)
)

GO

DROP TABLE IF EXISTS orders
CREATE TABLE orders
(
order_id int PRIMARY KEY,
order_date date,
customer_id int,
CONSTRAINT FK_customer_id FOREIGN KEY (customer_id) REFERENCES
customers (customer_id)
)

GO

DROP TABLE IF EXISTS products
CREATE TABLE products
(
product_id int PRIMARY KEY,
product_name varchar(100),
price decimal
)
```

```
GO

DROP TABLE IF EXISTS order_items
CREATE TABLE order_items
(
order_id int,
product_id int,
quantity int,
CONSTRAINT PK_order_items PRIMARY KEY(order_id, product_id),
CONSTRAINT FK_order_id FOREIGN KEY (order_id) REFERENCES  orders
(order_id) ,
CONSTRAINT FK_product_id FOREIGN KEY (product_id) REFERENCES products
(product_id)
)
```

```
GO
INSERT INTO customers (customer_id, first_name, last_name, email) VALUES
(1, 'John', 'Doe', 'johndoe@email.com'),
(2, 'Jane', 'Smith', 'janesmith@email.com'),
(3, 'Bob', 'Johnson', 'bobjohnson@email.com'),
(4, 'Alice', 'Brown', 'alicebrown@email.com'),
(5, 'Charlie', 'Davis', 'charliedavis@email.com'),
(6, 'Eva', 'Fisher', 'evafisher@email.com'),
(7, 'George', 'Harris', 'georgeharris@email.com'),
(8, 'Ivy', 'Jones', 'ivyjones@email.com'),
(9, 'Kevin', 'Miller', 'kevinmiller@email.com'),
(10, 'Lily', 'Nelson', 'lilynelson@email.com'),
(11, 'Oliver', 'Patterson', 'oliverpatterson@email.com'),
(12, 'Quinn', 'Roberts', 'quinnroberts@email.com'),
(13, 'Sophia', 'Thomas', 'sophiathomas@email.com');

GO
INSERT INTO orders (order_id, customer_id, order_date) VALUES
(1, 1, '2023-05-01'),
(2, 2, '2023-05-02'),
(3, 3, '2023-05-03'),
(4, 1, '2023-05-04'),
(5, 2, '2023-05-05'),
(6, 3, '2023-05-06'),
(7, 4, '2023-05-07'),
(8, 5, '2023-05-08'),
(9, 6, '2023-05-09'),
(10, 7, '2023-05-10'),
(11, 8, '2023-05-11'),
(12, 9, '2023-05-12'),
(13, 10, '2023-05-13'),
(14, 11, '2023-05-14'),
(15, 12, '2023-05-15'),
(16, 13, '2023-05-16');
```

```
GO
INSERT INTO products (product_id, product_name, price) VALUES
(1, 'Product A', 10.00),
(2, 'Product B', 15.00),
(3, 'Product C', 20.00),
(4, 'Product D', 25.00),
(5, 'Product E', 30.00),
(6, 'Product F', 35.00),
(7, 'Product G', 40.00),
(8, 'Product H', 45.00),
(9, 'Product I', 50.00),
(10, 'Product J', 55.00),
(11, 'Product K', 60.00),
(12, 'Product L', 65.00),
(13, 'Product M', 70.00);

GO
INSERT INTO order_items (order_id, product_id, quantity) VALUES
(1, 1, 2),
(1, 2, 1),
(2, 2, 1),
(2, 3, 3),
(3, 1, 1),
(3, 3, 2),
(4, 2, 4),
(4, 3, 1),
(5, 1, 1),
(5, 3, 2),
(6, 2, 3),
(6, 1, 1),
(7, 4, 1),
(7, 5, 2),
(8, 6, 3),
(8, 7, 1),
(9, 8, 2),
(9, 9, 1),
(10, 10, 3),
(10, 11, 2),
(11, 12, 1),
(11, 13, 3),
(12, 4, 2),
(12, 5, 1),
(13, 6, 3),
(13, 7, 2),
(14, 8, 1),
(14, 9, 2),
(15, 10, 3),
(15, 11, 1),
(16, 12, 2),
(16, 13, 3);
```

1.8 Data cleaning

The data is entered manually, there are very few records, we skip the data cleaning process and go directly to the part that consists of answering business questions.

1.9 Business Questions

1.9.1 WHICH PRODUCT HAS THE HIGHEST PRICE? ONLY RETURN A SINGLE ROW?

```
SELECT product_id, product_name, price FROM products
WHERE Price IN (SELECT MAX (price) FROM products)
```

product_id	product_name	price
13	Product M	70

1.9.2 WHICH CUSTOMER HAS MADE THE MOST ORDERS?

```
WITH cte_a AS
(
SELECT a.first_name, a.last_name, a.customer_id cust_id_a,
COUNT(b.order_id) nb_orders FROM customers a
JOIN orders b ON  a.customer_id= b.customer_id
GROUP BY a.customer_id,  b.customer_id, a.first_name, a.last_name
)

SELECT cust_id_a,first_name, last_name FROM cte_a
WHERE nb_orders IN (SELECT MAX(nb_orders) FROM cte_a)
```

Résultats | Messages

	cust_id_a	first_name	last_name
1	1	John	Doe
2	2	Jane	Smith
3	3	Bob	Johnson

1.9.3 WHAT'S THE TOTAL REVENUE PER PRODUCT?

```
WITH cte_a AS
(
SELECT a.product_name, a.price, c.order_id, c.product_id, c.quantity,
a.price*c.quantity revenue, SUM( a.price*c.quantity) OVER (PARTITION BY
c.product_id) total_revenue
FROM products a
JOIN order_items c ON a.product_id = c.product_id
)

SELECT DISTINCT product_id, product_name, total_revenue FROM cte_a
```

Résultats | Messages

	product_id	product_name	total_revenue
1	1	Product A	50
2	2	Product B	135
3	3	Product C	160
4	4	Product D	75
5	5	Product E	90
6	6	Product F	210
7	7	Product G	120
8	8	Product H	135
9	9	Product I	150
10	10	Product J	330
11	11	Product K	180
12	12	Product L	195
13	13	Product M	420

1.9.4 FIND THE DAY WITH THE HIGHEST REVENUE

```
WITH cte_a AS
(
SELECT a.product_name, a.price, c.order_id, c.product_id, c.quantity,
b.order_date, a.price*c.quantity revenue_per_order_id_per_product,
SUM( a.price*c.quantity) OVER (PARTITION BY b.order_date) total_revenue_day
FROM products a
JOIN order_items c ON a.product_id = c.product_id
JOIN orders b ON c.order_id = b.order_id
)

SELECT DISTINCT order_date, total_revenue_day FROM cte_a
WHERE total_revenue_day IN (SELECT MAX(total_revenue_day) FROM cte_a)
```

Résultats | Messages

	order_date	total_revenue_day
1	2023-05-16	340

1.9.5 FIND THE FIRST ORDER (BY DATE) FOR EACH CUSTOMER

```
SELECT DISTINCT b.customer_id,
MIN (b.order_date) OVER (PARTITION BY b.customer_id)
min_order_date_per_customer
FROM customers a
JOIN orders b ON a.customer_id = b.customer_id
ORDER BY b.customer_id
```

Résultats | Messages

	customer_id	min_order_date_per_customer
1	1	2023-05-01
2	2	2023-05-02
3	3	2023-05-03
4	4	2023-05-07
5	5	2023-05-08
6	6	2023-05-09
7	7	2023-05-10
8	8	2023-05-11
9	9	2023-05-12
10	10	2023-05-13
11	11	2023-05-14
12	12	2023-05-15
13	13	2023-05-16

1.9.6 FIND THE TOP 3 CUSTOMERS WHO HAVE ORDERED THE MOST DISTINCT PRODUCTS

I have two ways of viewing this question, if you read my solutions, tell me which one is the best way of understanding the question.

First way:

```
WITH cte_a AS
(
SELECT a.customer_id, b.order_id,c.product_id, d.quantity FROM customers a
JOIN orders b ON a.customer_id=b.customer_id
JOIN order_items d ON b.order_id=d.order_id
JOIN products c ON d.product_id=c.product_id
)


SELECT * INTO cte_c
FROM (
SELECT customer_id, COUNT (DISTINCT(product_id)) total_distinct_product,
SUM(quantity) total_quantity_ordered_per_customer
FROM cte_a
GROUP BY customer_id
 ) as cte_b

WITH cte_d AS
(
SELECT customer_id, total_distinct_product,
total_quantity_ordered_per_customer,
DENSE_RANK () OVER (ORDER BY total_distinct_product DESC
,total_quantity_ordered_per_customer desc) Rank_customer
FROM cte_c
)

SELECT a.first_name, a.last_name, a.customer_id,
cte_d.total_distinct_product,cte_d.total_quantity_ordered_per_customer,
Rank_customer FROM customers a
JOIN cte_d  ON cte_d.customer_id = a.customer_id
WHERE Rank_customer <= 3
```

Résultats | Messages

	first_name	last_name	customer_id	total_distinct_product	total_quantity_ordered_per_customer	Rank_customer
1	John	Doe	1	3	8	1
2	Jane	Smith	2	3	7	2
3	Bob	Johnson	3	3	7	2
4	George	Harris	7	2	5	3
5	Lily	Nelson	10	2	5	3
6	Sophia	Thomas	13	2	5	3

Second way:

```
WITH cte_a AS
(
SELECT a.customer_id, a.first_name, a.last_name,b.order_id,c.product_id,
d.quantity FROM customers a
JOIN orders b ON a.customer_id=b.customer_id
JOIN order_items d ON b.order_id=d.order_id
JOIN products c ON d.product_id=c.product_id
)


SELECT TOP 3 customer_id, first_name, last_name, COUNT (DISTINCT(product_id))
total_distinct_product
FROM cte_a
GROUP BY customer_id,first_name, last_name
ORDER BY COUNT (DISTINCT(product_id)) desc
```

Résultats | Messages

	customer_id	first_name	last_name	total_distinct_product
1	3	Bob	Johnson	3
2	2	Jane	Smith	3
3	1	John	Doe	3

1.9.7 WHICH PRODUCT HAS BEEN BOUGHT THE LEAST IN TERMS OF QUANTITY?

```
WITH cte_a AS
(
SELECT a.order_id, b.product_id, b.product_name, c.quantity,
SUM (c.quantity) OVER (PARTITION BY b.product_id) quantity_per_product
FROM orders a
JOIN order_items c ON c.order_id = a.order_id
JOIN products b ON b.product_id = c.product_id
)

SELECT DISTINCT product_id, product_name, quantity_per_product
FROM cte_a
WHERE quantity_per_product IN (SELECT MIN (quantity_per_product) FROM cte_a)
ORDER BY product_id
```

Résultats | Messages

	product_id	product_name	quantity_per_product
1	4	Product D	3
2	5	Product E	3
3	7	Product G	3
4	8	Product H	3
5	9	Product I	3
6	11	Product K	3
7	12	Product L	3

1.9.8 WHAT IS THE MEDIAN ORDER TOTAL?

```
WITH cte_a AS
(
SELECT DISTINCT a.order_id,
SUM(b.price * c.quantity) OVER (PARTITION BY a.order_id)
total_price_per_order
FROM orders a
JOIN order_items c ON c.order_id = a.order_id
JOIN products b ON b.product_id = c.product_id
)

SELECT DISTINCT PERCENTILE_CONT (0.5) WITHIN GROUP (ORDER BY
total_price_per_order) OVER () FROM  cte_a
```

Résultats | Messages

	(Aucun nom de colonne)
1	112,5

1.9.9 FOR EACH ORDER, DETERMINE IF IT WAS 'EXPENSIVE' (TOTAL OVER 300), 'AFFORDABLE' (TOTAL OVER 100), OR 'CHEAP'

```
SELECT DISTINCT a.order_id,
SUM(b.price * c.quantity) OVER (PARTITION BY a.order_id)
total_price_per_order,
CASE
WHEN SUM(b.price * c.quantity) OVER (PARTITION BY a.order_id) > 300 THEN
'Expensive'
WHEN SUM(b.price * c.quantity) OVER (PARTITION BY a.order_id) > 100 and
SUM(b.price * c.quantity) OVER (PARTITION BY a.order_id) <=300 THEN
'Affordable'
ELSE 'Cheap'
END as Type_of_order
FROM orders a
JOIN order_items c ON c.order_id = a.order_id
JOIN products b ON b.product_id = c.product_id
```

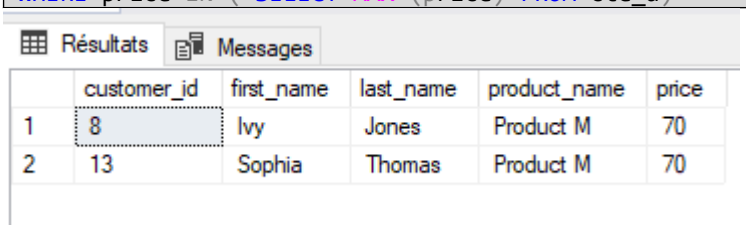
Résultats | Messages

	order_id	total_price_per_order	Type_of_order
1	1	35	Cheap
2	2	75	Cheap
3	3	50	Cheap
4	4	80	Cheap
5	5	50	Cheap
6	6	55	Cheap
7	7	85	Cheap
8	8	145	Affordable
9	9	140	Affordable
10	10	285	Affordable
11	11	275	Affordable
12	12	80	Cheap
13	13	185	Affordable
14	14	145	Affordable
15	15	225	Affordable
16	16	340	Expensive

1.9.10 FIND CUSTOMERS WHO HAVE ORDERED THE PRODUCT WITH THE HIGHEST PRICE

```
WITH cte_a AS
(
SELECT a.customer_id, a.first_name,
a.last_name,b.order_id,c.product_id,c.product_name, d.quantity, c.price
FROM customers a
JOIN orders b ON a.customer_id=b.customer_id
JOIN order_items d ON b.order_id=d.order_id
JOIN products c ON d.product_id=c.product_id
)

SELECT customer_id, first_name, last_name, product_name, price FROM cte_a
WHERE price IN ( SELECT MAX (price) FROM cte_a)
```



Résultats | Messages

	customer_id	first_name	last_name	product_name	price
1	8	Ivy	Jones	Product M	70
2	13	Sophia	Thomas	Product M	70