E Commerce Orders and Marketing

PROBLEM STATEMENT:

In this project, we are using a database that has a collection of past data from an **e-commerce website** that has details about **customers, products, transactions, sellers, and reviews** of various products which are listed on the website along with data of **marketing funnel** which can help us understand the performance of the company from a marketing perspective.

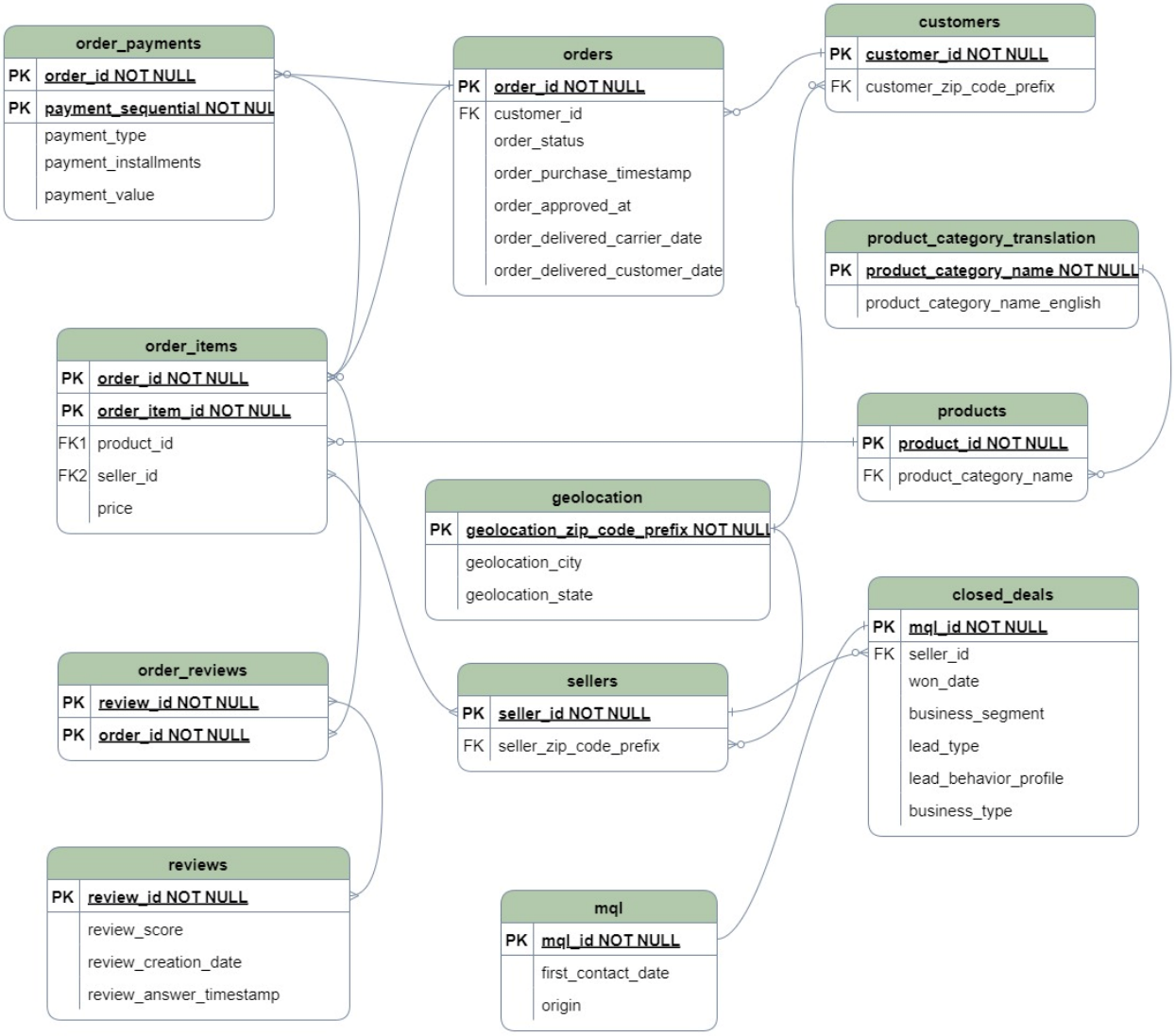
Data Source: Olist, a Brazilian e-commerce company

Why Database? Potential users?

- Database can be easily manipulated using DML queriers with high accuracy compared to an Excel file
- easy to make updates across the tables simple
- can help e-commerce websites understand product dynamics
- give insights to marketing teams to curate content for their target audience

DATABASE INTEGRITY:

Entity Relationship Diagram:



Relations and attributes:

Relation	Attributes	Data Type
Customers	Customer_id	VARCHAR
	Customer_zip_code	BIG INT
Geolocation	Geolocation_zip_code_ptrefix	BIGINT
	Geolocation_city	VARCHAR
	Geolocation_state	VARCHAR
Order_items	Order_id	VARCHAR
	Order_item_id	INT
	Product_id	VARCHAR
	Seller_id	VARCHAR
	Price	NUMERIC
Order_Payments	Order_id	VARCHAR
	Payment_sequential	INT
	Payment_type	VARCHAR
	Payment_installments	INT
	Payment_value	NUMERIC
orders	Order_id	VARCHAR
	Customer_id	VARCHAR
	Order_status	VARCHAR
	Order_purchase_timestamp	TIMESTAMP WITHOUT TIMEZONE
	Order_approved_at	TIMESTAMP WITHOUT TIMEZONE
	Order_delivered_carrier_date	TIMESTAMP WITHOUT TIMEZONE
	Order_delivered_customer_date	TIMESTAMP WITHOUT TIMEZONE
Products	Product_id	VARCHAR
	Product_category_namre	VARCHAR
Reviews	Review_id	VARCHAR
	Review_score	INT
	Review_creation_date	TIMESTAMP WITHOUT TIMEZONE
	Review_answer_timestamo	TIMESTAMP WITHOUT TIMEZONE
sellers	Seller_id	VARCHAR
	Seller_zip_code_prefix	INT
Product_category _translation	Product_category_name	VARCHAR
	Product_category_name_english	VARCHAR
mql	Mql_id	VARCHAR
	First_contact_date	DATE
	Origin	VARCHAR

Closed_deals	Mql_id	VARCHAR
	Seller_id	VARCHAR
	Won_date	TIMESTAMP WITHOUT TIMEZONE
	Business_segment	VARCHAR
	Lead_type	VARCHAR
	Lead_behavior_profile	VARCHAR
	Business_type	VARCHAR

BCNF proof:

Table Name	FDs	Primary key
orders	Order_id ➞ customer_id Order_id ➞ order_status Order_id ➞ order_purchase_timestamp Order_id ➞order_approved_at Order_id ➞order_delivered_carrier_date Order_id ➞order_delivered_customer_date	Order_id VARCHAR
Order_payments	Order_id, payment_sequential ➞payment_type Order_id, payment_sequential ➞payment_installments Order_id, payment_sequential ➞payment_value	(Order_id VARCHAR, payment_sequential NUMERIC)
Order_items	Order_id, order_item_id ➞product_id Order_id, order_item_id ➞ seller_id Order_id, order_item_id ➞price	(Order_id VARCHAR, order_item_id VARCHAR)
Order_reviews	None (Full keyed relation)	
Reviews	Review_id ➞ review_score Review_id ➞review_creation_date Review_id ➞ review_answer_timestamp	Review_id VARCHAR
Customers	Customer_id ➞customer_zip_code_prefix	Customer_id VARCHAR
Sellers	Seller_id ➞ seller_zip_code_prefix	Seller_id VARCHAR
Geolocation	Geolocation_zip_code_prefix ➞ geolocation_city Geolocation_zip_code_prefix ➞ geolocation_state	Geolocation_zip_code_prefix BIGINT
mql	Mql_id ➞ first_contact_date Mql_id ➞ origin	Mql_id VARCHAR
Closed_deals	Mql_id ➞ seller_id Mql_id ➞ won_date Mql_id ➞ business_segment Mql_id ➞ lead_type Mql_id ➞ lead_behavior_profile Mql_id ➞ business_type	Mql_id VARCHAR

Sample data:

```
--Viewing sample data
SELECT * FROM reviews
```

Output Explain Messages Notifications

review_id [PK] character varying	review_score integer	review_creation_date timestamp without time zone	review_answer_timestamp timestamp without time zone
7bc2406110b926393aa56f80a40eba40	4	2018-01-18 00:00:00	2018-01-18 21:46:00
80e641a11e56f04c1ad469d5645fdfde	5	2018-03-10 00:00:00	2018-03-11 03:05:00
228ce5500dc1d8e020d8d1322874b6f0	5	2018-02-17 00:00:00	2018-02-18 14:36:00
e64fb393e7b32834bb789ff8bb30750e	5	2017-04-21 00:00:00	2017-04-21 22:02:00
f7c4243c7fe1938f181bec41a392bdeb	5	2018-03-01 00:00:00	2018-03-02 10:26:00
15197aa66ff4d0650b5434f1b46cda19	1	2018-04-13 00:00:00	2018-04-16 00:39:00
07f9bee5d1b850860defd761afa7ff16	5	2017-07-16 00:00:00	2017-07-18 19:30:00

```
SELECT * FROM mql
```

Output Explain Messages Notifications

mql_id [PK] character varying	first_contact_date date	origin character varying
dac32acd4db4c29c230538b72f8dd87d	2018-02-01	social
8c18d1de7f67e60dbd64e3c07d7e9d5d	2017-10-20	paid_search
b4bc852d233dfefc5131f593b538befa	2018-03-22	organic_search
6be030b81c75970747525b843c1ef4f8	2018-01-22	email
5420aad7fec3549a85876ba1c529bd84	2018-02-21	organic_search
28bdfd5f057764b54c38770f95c69f2f	2018-01-14	organic_search
126a0d10becbaafcb2e72ce6848cf32c	2018-05-15	email

SQL Execution:

Handling timestamp column:

```
SELECT * FROM reviews
WHERE (EXTRACT(YEAR FROM review_creation_date) = 2018 AND
EXTRACT(MONTH FROM review_creation_date) = 2)
```

	review_id [PK] character varying	review_score integer	review_creation_date timestamp without time zone	review_answer_timestamp timestamp without time zone
1	228ce5500dc1d8e020d8d1322874b6f0	5	2018-02-17 00:00:00	2018-02-18 14:36:00
2	4b49719c8a200003f700d3d986ea1a19	4	2018-02-16 00:00:00	2018-02-20 10:52:00
3	eb26c2bfb5030f57dcef30d3f111eb1e	1	2018-02-08 00:00:00	2018-02-10 04:28:00
4	ca7402594d96f3231ee1bbae27da1e29	5	2018-02-20 00:00:00	2018-02-21 01:40:00
5	9924ac6169f2edc95ca3394e529f9580	5	2018-02-27 00:00:00	2018-03-02 08:33:00
6	5733e2aeeb52b99d044a2a3644e1305d	5	2018-02-23 00:00:00	2018-02-23 11:06:00
7	4ada105731802020d66105508076cb49	3	2018-02-01 00:00:00	2018-02-01 06:26:00

Using INSERT statement:

```
INSERT INTO customers(customer_id, customer_zip_code_prefix)
VALUES ('f78374342g34837gg47846274ghhh3hkk', 8775)
```

```
SELECT * FROM customers
WHERE customer_id = 'f78374342g34837gg47846274ghhh3hkk'
```

	customer_id [PK] character varying	customer_zip_code_prefix bigint
1	f78374342g34837gg47846274ghhh3hkk	8775

```
INSERT INTO reviews(review_id,
                    review_score,
                    review_creation_date,
                    review_answer_timestamp)
VALUES('F8786VY875656G5feef67746576vd',
       3,
       '2022-02-21 00:00:00',
       '2022-01-30 16:23:00')
```

```
SELECT * FROM reviews
WHERE EXTRACT(YEAR FROM review_creation_date) = 2022
```

	review_id [PK] character varying	review_score integer	review_creation_date timestamp without time zone	review_answer_timestamp timestamp without time zone
1	F8786VY875656G5feef67746576vd	3	2022-02-21 00:00:00	2022-01-30 16:23:00

Using DELETE Statement:

```
DELETE FROM closed_deals
WHERE lead_type IS NULL
```

```
SELECT * FROM closed_deals
WHERE lead_type IS NULL
```

mql_id [PK] character varying	seller_id character varying	won_date timestamp without time zone	business_segment character varying

Using UPDATE StatementL:

```
UPDATE order_items
SET price = 60
WHERE product_id = '4244733e06e7ecb4970a6e2683c13e61'

SELECT order_id, product_id, price FROM order_items
WHERE product_id = '4244733e06e7ecb4970a6e2683c13e61'
```

	order_id character varying	product_id character varying	price numeric
1	130898c0987d1801452a8ed92a670612	4244733e06e7ecb4970a6e2683c13e61	60
2	00010242fe8c5a6d1ba2dd792cb16214	4244733e06e7ecb4970a6e2683c13e61	60
3	532ed5e14e24ae1f0d735b91524b98b9	4244733e06e7ecb4970a6e2683c13e61	60
4	6f8c31653edb8c83e1a739408b5ff750	4244733e06e7ecb4970a6e2683c13e61	60
5	7d19f4ef4d04461989632411b7e588b9	4244733e06e7ecb4970a6e2683c13e61	60
6	a0f9acf0b6294ed8561e32cde1a966bc	4244733e06e7ecb4970a6e2683c13e61	60

Complex Queries:

- Finding the product categories with the most number of orders

```
SELECT t2.product_category, SUM(t1.num_orders) num_orders FROM
    (SELECT COUNT(order_id) AS num_orders, product_id
     FROM order_items
     GROUP BY product_id) AS t1
JOIN
    (SELECT product_id, product_category_name_english AS product_category
     FROM products pd
     JOIN product_category_translation pct
     ON pd.product_category_name = pct.product_category_name) AS t2
ON t1.product_id = t2.product_id
GROUP BY t2.product_category
ORDER BY num_orders DESC
```

	product_category character varying	num_orders numeric
1	bed_bath_table	11115
2	health_beauty	9670
3	sports_leisure	8641
4	furniture_decor	8334
5	computers_accessories	7827
6	housewares	6964

- Finding the proportion of each business type in closed deals

```
SELECT t2.business_type,
    t2.ct*100 /(SELECT SUM(ct) FROM
                (SELECT business_type, COUNT(business_type) ct
                 FROM closed_deals
                 GROUP BY business_type) t1
                ) pct_of_count
    FROM
    (SELECT business_type, COUNT(business_type) ct
    FROM closed_deals
    GROUP BY business_type) t2
```

	business_type character varying	pct_of_count numeric
1	reseller	75.9358288770053476
2	manufacturer	24.0641711229946524

#	Node	Exclusive	Inclusive	Rows X	Actual	Plan	Loops
1.	→ Subquery Scan (cost=57.46..57.51 rows=2 width=4...	0.012 ms	0.755 ms	↑ 1	2	2	1
2.	→ Aggregate (cost=28.75..28.76 rows=1 width=3...	0.011 ms	0.364 ms	↑ 1	1	1	1
3.	→ Aggregate (cost=28.7..28.72 rows=2 widt... Buckets: Batches: Memory Usage: 24 kB	0.278 ms	0.354 ms	↑ 1	2	2	1
4.	→ Seq Scan on closed_deals as closed_...	0.076 ms	0.076 ms	↑ 1.02	374	380	1
5.	→ Aggregate (cost=28.7..28.72 rows=2 width=17... Buckets: Batches: Memory Usage: 24 kB	0.267 ms	0.38 ms	↑ 1	2	2	1
6.	→ Seq Scan on closed_deals as closed_deal...	0.113 ms	0.113 ms	↑ 1.02	374	380	1

- Finding the state with the most number of customers

```
SELECT geolocation_state state,count(customer_id) num_cust
FROM customers c, geolocation g
WHERE c.customer_zip_code_prefix = g.geolocation_zip_code_prefix
GROUP BY g.geolocation_state
ORDER BY num_cust DESC limit 5
```

Graphical Analysis Statistics

1.	→ Limit (cost=3369.9..3369.92 rows=5 width...	0.004 ms	92.944 ms	↑ 1	5	5	1
2.	→ Sort (cost=3369.9..3369.97 rows=27 ...	0.026 ms	92.941 ms	↑ 5.4	5	27	1
3.	→ Aggregate (cost=3369.19..3369.... **Buckets**: **Batches**: **Memory Usage**: 24 kB	27.098 ms	92.915 ms	↑ 1	27	27	1
4.	→ Hash Inner Join (cost=686.... **Hash Cond**: (c.customer_zip_code_prefix = g.geolocation_zip_code_prefix)	50.542 ms	65.818 ms	↓ 1.01	99442	99441	1
5.	→ Seq Scan on customer...	9.282 ms	9.282 ms	↓ 1.01	99442	99441	1
6.	→ Hash (cost=446.77..44... **Buckets**: 32768 **Batches**: 1 **Memory Usage**: 1155 kB	3.224 ms	5.994 ms	↑ 1	19177	19177	1
7.	→ Seq Scan on geolo...	2.77 ms	2.77 ms	↑ 1	19177	19177	1

```
SELECT geolocation_state state,count(customer_id) num_cust FROM customers c
JOIN geolocation g
ON c.customer_zip_code_prefix = g.geolocation_zip_code_prefix
GROUP BY g.geolocation_state
ORDER BY num_cust DESC limit 5
```

Graphical Analysis Statistics

1.	→ Limit (cost=3369.9..3369.92 rows=5 width...	0.004 ms	88.163 ms	↑ 1	5	5	1
2.	→ Sort (cost=3369.9..3369.97 rows=27 ...	0.032 ms	88.16 ms	↑ 5.4	5	27	1
3.	→ Aggregate (cost=3369.19..3369.... **Buckets**: **Batches**: **Memory Usage**: 24 kB	26.558 ms	88.128 ms	↑ 1	27	27	1
4.	→ Hash Inner Join (cost=686.... **Hash Cond**: (c.customer_zip_code_prefix = g.geolocation_zip_code_prefix)	45.873 ms	61.571 ms	↓ 1.01	99442	99441	1
5.	→ Seq Scan on customer...	9.199 ms	9.199 ms	↓ 1.01	99442	99441	1
6.	→ Hash (cost=446.77..44... **Buckets**: 32768 **Batches**: 1 **Memory Usage**: 1155 kB	3.743 ms	6.499 ms	↑ 1	19177	19177	1
7.	→ Seq Scan on geolo...	2.756 ms	2.756 ms	↑ 1	19177	19177	1

	state character varying	num_cust bigint
1	SP	41747
2	RJ	12852
3	MG	11635
4	RS	5466
5	PR	5045

- Finding the state, city and number of sales of the highest seller

```
SELECT T2.seller_id,T1.geolocation_city, T1.geolocation_state, T2.sales
FROM
    (SELECT seller_id, count(order_id) sales
    FROM order_items
    GROUP BY seller_id
    ) T2
JOIN
    (SELECT s.seller_id, g.geolocation_city, g.geolocation_state
    FROM sellers s
    JOIN geolocation g
    ON s.seller_zip_code_prefix = g.geolocation_zip_code_prefix) T1
ON T1.seller_id =T2.seller_id
ORDER BY sales DESC LIMIT 10
```

	seller_id character varying	geolocation_city character varying	geolocation_state character varying	sales bigint
1	6560211a19b47992c3666cc44a7e94c0	sao paulo	SP	2033
2	4a3ca9315b744ce9f8e9374361493884	ibitinga	SP	1987
3	1f50f920176fa81dab994f9023523100	sao jose do rio preto	SP	1931
4	cc419e0650a3c5ba77189a1882b7556a	santo andre	SP	1775

- Stored procedure to update new product categories and their translation

```
CREATE PROCEDURE insert_data(product_category_name VARCHAR,
                             product_category_name_english VARCHAR)
LANGUAGE SQL
AS $$
INSERT INTO product_category_translation VALUES (product_category_name,
                                                 product_category_name_english);
$$
;

call insert_data('alimentos congelados', 'frozen foods')

select * from product_category_translation
where product_category_name_english = 'frozen foods'
```

	product_category_name [PK] character varying	product_category_name_english character varying
1	alimentos congelados	frozen foods

- Trigger to count the number of product categories

```
CREATE TABLE pr_cat_ct (product_category_count BIGINT);
```

```
CREATE FUNCTION count_pr_cat(
) RETURNS trigger AS $$
BEGIN
    PERFORM (SELECT COUNT(product_category_name) FROM product_category_translation);
END; $$
LANGUAGE plpgsql;
```

```
CREATE TRIGGER pr_cat_count
    AFTER INSERT
    ON product_category_translation
    FOR EACH ROW
    EXECUTE PROCEDURE public.count_pr_cat();
    END ;
```

```
SELECT * FROM pr_cat_ct
```

	product_category_count bigint
1	73

Summary and Findings:

- The product categories, ‘bed_bath_table’, had the highest number of orders followed by ‘health_beauty’, ‘sports_leisure’
- The business types ‘reseller’ and ‘manufacturers’ contributed to 75% and 25% of closed deals respectively.
- The state ‘SP’ had the highest number of customers and the highest number of sales
- Top sellers, their cities, states and their respective sales were extracted
- Created a stored procedure to insert new data into the product category table
- A trigger is implemented to count the number of product categories the website is handling at any given instance