Business case Target SQL

Business Context

Target is one of the world's most recognized brands and one of America's leading retailers. Target makes itself a preferred shopping destination by offering outstanding value, inspiration, innovation and an exceptional guest experience that no other retailer can deliver.
This business case has information of 100k orders from 2016 to 2018 made at Target in Brazil. Its features allow viewing an order from multiple dimensions: from order status, price, payment and freight performance to customer location, product attributes and finally reviews written by customers.
Dataset

Data is available in 8 csv files. Each feature or columns of different CSV files are described below:

1. customers.csv

Features	Description
customer_id	Id of the consumer who made the purchase.
customer_unique_id	Unique Id of the consumer.
customer_zip_code_prefix	Zip Code of the location of the consumer.
customer_city	Name of the City from where order is made.
customer_state	State Code from where order is made(Ex- sao paulo-SP).

2. geolocation.csv

Features	Description
geolocation_zip_code_prefix	first 5 digits of zip code
geolocation_lat	latitude
geolocation_lng	longitude
geolocation_city	city name
geolocation_state	state

3. order_items.csv

Features	Description
order_id	A unique id of order made by the consumers.
order_item_id	A Unique id given to each item ordered in the order.
product_id	A unique id given to each product available on the site.
seller_id	Unique Id of the seller registered in Target.
shipping_limit_date	The date before which shipping of the ordered product must be completed.
price	Actual price of the products ordered .
freight_value	Price rate at which a product is delivered from one point to another.

4. payments.csv

Features	Description
order_id	A unique id of order made by the consumers.
payment_sequential	sequences of the payments made in case of EMI.
payment_type	mode of payment used.(Ex-Credit Card)
payment_installments	number of installments in case of EMI purchase.
payment_value	Total amount paid for the purchase order.

5. reviews.csv

Features	Description
review_id	Id of the review given on the product ordered by the order id.
order_id	A unique id of order made by the consumers.
review_score	review score given by the customer for each order on the scale of 1–5.
review_comment_title	Title of the review
review_comment_message	Review comments posted by the consumer for each order.
review_creation_date	Timestamp of the review when it is created.
review_answer_timestamp	Timestamp of the review answered.

6. orders.csv

Features	Description
order_id	A unique id of order made by the consumers.
customer_id	Id of the consumer who made the purchase.
order_status	status of the order made i.e delivered, shipped etc.
order_purchase_timestamp	Timestamp of the purchase.
order_delivered_carrier_date	delivery date at which carrier made the delivery.
order_delivered_customer_date	date at which customer got the product.
order_estimated_delivery_date	estimated delivery date of the products.

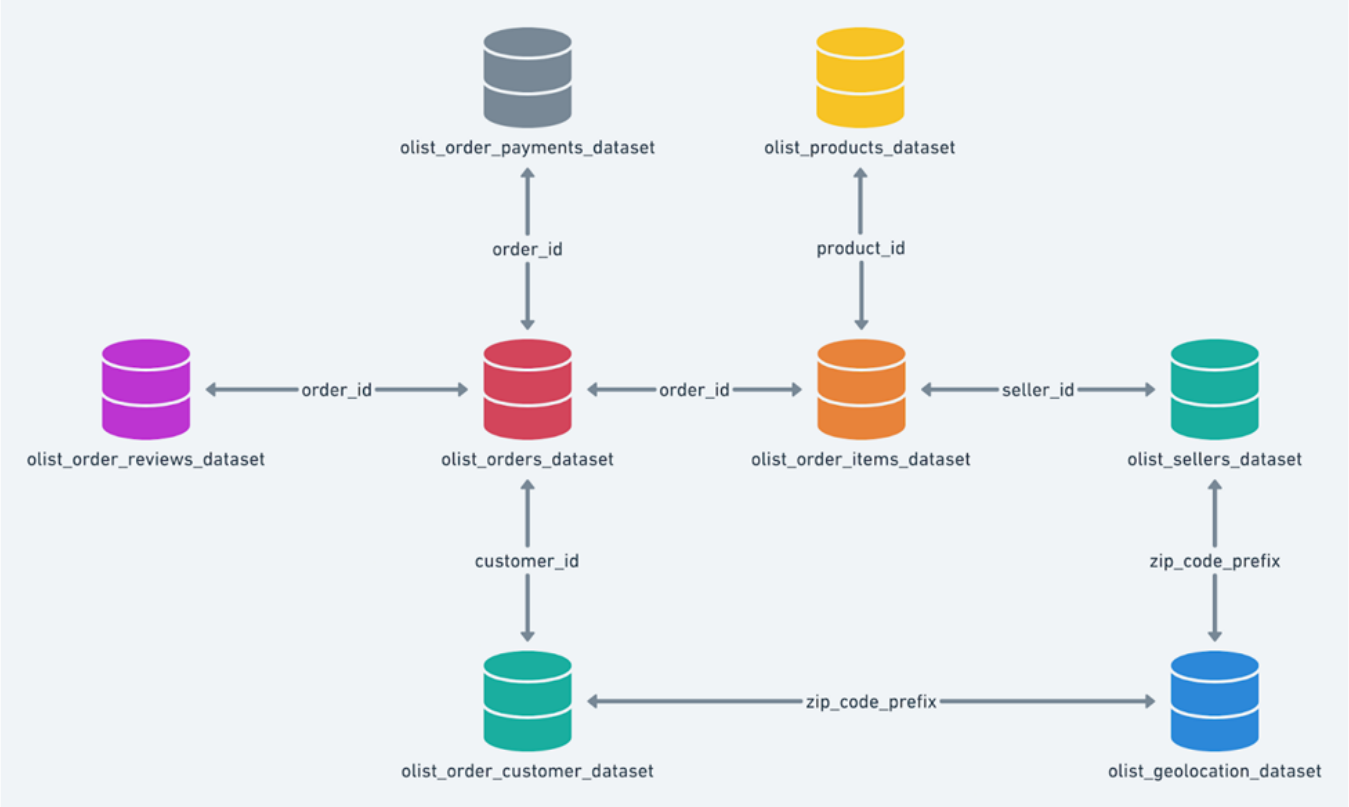
7. products.csv

Features	Description
product_id	A unique identifier for the proposed project.
product_category_name	Name of the product category
product_name_lenght	length of the string which specifies the name given to the products ordered.
product_description_lenght	length of the description written for each product ordered on the site.
product_photos_qty	Number of photos of each product ordered available on the shopping portal.
product_weight_g	Weight of the products ordered in grams.
product_length_cm	Length of the products ordered in centimeters.
product_height_cm	Height of the products ordered in centimeters.
product_width_cm	width of the product ordered in centimeters.

8. sellers.csv

Features	Description
seller_id	Unique Id of the seller registered
seller_zip_code_prefix	Zip Code of the location of the seller.
seller_city	Name of the City of the seller.
seller_state	State Code (Ex- sao paulo-SP)

High level overview of relationship between datasets:



Additional views:

We will use MySQL for this case study. We will first create schema, tables, constraints and indexes as per the given details and use MySQL Load Data command to import csv files data into tables. In the data exploration section, we will attempt to analyze month over month orders data across states, payment type, and payment installation. We will rely on lag window function after partitioning data over suitable column. However, in the given data-set, we do not have enough rows to fill all the combinations of given attribute and time series data. Therefore, we will not apply window functions directly on given tables. We will first compute time series reference table containing all combinations of yyyy-mm values in the given range (2016-09 to 2018-10). We will then take left outer join between time series reference table and other relevant tables before applying window functions. This additional time series table and the corresponding creating stored procedure is described in the additional constructs section.

Solution

Schema Creation

```
-- MySQL Workbench Forward Engineering

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
```

```
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR
_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

-- -----------------------------------------------------
-- Schema target
-- -----------------------------------------------------

-- -----------------------------------------------------
-- Schema target
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `target` DEFAULT CHARACTER SET utf8mb3;
USE `target` ;

-- -----------------------------------------------------
-- Table `target`.`geolocation`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `target`.`geolocation` ;

CREATE TABLE IF NOT EXISTS `target`.`geolocation` (
  `zip_code_prefix` INT NOT NULL,
  `lat` DECIMAL(24,20) NULL,
  `lng` DECIMAL(24,20) NULL,
  `city` VARCHAR(45) NULL,
  `state` VARCHAR(5) NULL)
ENGINE = InnoDB;


CREATE TABLE IF NOT EXISTS `target`.`customers` (
  `customer_id` CHAR(32) NOT NULL,
  `customer_unique_id` CHAR(32) NOT NULL,
  `zip_code_prefix` INT NULL DEFAULT NULL,
  `city` VARCHAR(45) NULL DEFAULT NULL,
  `state` VARCHAR(5) NULL DEFAULT NULL,
  PRIMARY KEY (`customer_id`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb3;

CREATE INDEX state_index ON `target`.`customers` (`state` ASC);
-- -----------------------------------------------------
-- Table `target`.`sellers`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `target`.`sellers` ;

CREATE TABLE IF NOT EXISTS `target`.`sellers` (
```

```
  `seller_id` CHAR(32) NOT NULL,
  `zip_code_prefix` INT NULL,
  `state` VARCHAR(5) NULL,
  `city` VARCHAR(45) NULL,
  PRIMARY KEY (`seller_id`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `target`.`products`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `target`.`products` ;

CREATE TABLE IF NOT EXISTS `target`.`products` (
  `product_id` CHAR(32) NOT NULL,
  `category` VARCHAR(100) NULL,
  `name_length` INT NULL,
  `description_length` INT NULL,
  `photos_qty` SMALLINT(255) NULL,
  `weight_g` INT NULL,
  `length_cm` INT NULL,
  `height_cm` INT NULL,
  `width_cm` INT NULL,
  PRIMARY KEY (`product_id`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `target`.`orders`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `target`.`orders` ;

CREATE TABLE IF NOT EXISTS `target`.`orders` (
  `order_id` CHAR(32) NOT NULL,
  `customer_id` CHAR(32) NULL,
  `order_status` VARCHAR(45) NULL,
  `order_purchase_ts` DATETIME NULL,
  `order_approved_at` DATETIME NULL,
  `order_delivered_carrier_date` DATETIME NULL,
  `order_delivered_customer_date` DATETIME NULL,
  `order_estimated_delivery_date` DATETIME NULL,
  PRIMARY KEY (`order_id`),
  INDEX `cust_id_fk_idx` (`customer_id` ASC) VISIBLE,
  CONSTRAINT `orders_cust_id_fk`
    FOREIGN KEY (`customer_id`)
    REFERENCES `target`.`customers` (`customer_id`)
```

```
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `target`.`order_items`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `target`.`order_items` ;

CREATE TABLE IF NOT EXISTS `target`.`order_items` (
  `order_id` CHAR(32) NULL,
  `order_item_id` INT NULL,
  `product_id` CHAR(32) NULL,
  `seller_id` CHAR(32) NULL,
  `shipping_limit_date` DATETIME NULL,
  `price` DECIMAL(10,2) NULL,
  `freight_value` DECIMAL(10,2) NULL,
  INDEX `order_id_fk_idx` (`order_id` ASC) VISIBLE,
  INDEX `prod_id_fk_idx` (`product_id` ASC) VISIBLE,
  INDEX `seller_id_fk_idx` (`seller_id` ASC) VISIBLE,
  CONSTRAINT `order_items_order_id_fk`
    FOREIGN KEY (`order_id`)
    REFERENCES `target`.`orders` (`order_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `order_items_prod_id_fk`
    FOREIGN KEY (`product_id`)
    REFERENCES `target`.`products` (`product_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `order_items_seller_id_fk`
    FOREIGN KEY (`seller_id`)
    REFERENCES `target`.`sellers` (`seller_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `target`.`order_payment`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `target`.`order_payment` ;

CREATE TABLE IF NOT EXISTS `target`.`order_payment` (
  `order_id` CHAR(32) NULL,
```

```
  `payment_sequential` INT NULL,
  `payment_type` VARCHAR(45) NULL,
  `payment_installment` INT NULL,
  `payment_value` DECIMAL(10,2) NULL,
  `order_paymentcol` VARCHAR(45) NULL,
  INDEX `order_id_fk_idx` (`order_id` ASC) VISIBLE,
  CONSTRAINT `payments_order_id_fk`
    FOREIGN KEY (`order_id`)
    REFERENCES `target`.`orders` (`order_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `target`.`order_reviews`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `target`.`order_reviews` ;

CREATE TABLE IF NOT EXISTS `target`.`order_reviews` (
  `review_id` CHAR(32) NOT NULL,
  `order_id` CHAR(32) NOT NULL,
  `review_score` TINYINT NULL,
  `review_comment_title` VARCHAR(255) NULL,
  `review_creation_date` DATETIME NULL,
  `review_answer_timestamp` DATETIME NULL,
  INDEX `order_id_fk_idx` (`order_id` ASC) INVISIBLE,
  UNIQUE INDEX `reviews_unique_key` (`review_id` ASC, `order_id` ASC) VISIBLE,
  CONSTRAINT `reviews_order_id_fk`
    FOREIGN KEY (`order_id`)
    REFERENCES `target`.`orders` (`order_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

Data Import

```
use target;
```

```sql
-- load geolocation data
LOAD DATA local INFILE
"C:\\Users\\chins\\OneDrive\\source\\scaler_business_casestudies\\10_target_case\\T
arget- SQL Business Case\\geolocation.csv"
INTO TABLE geolocation
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
;

select count(*) from geolocation;

-- load customers data
LOAD DATA local INFILE
"C:\\Users\\chins\\OneDrive\\source\\scaler_business_casestudies\\10_target_case\\T
arget- SQL Business Case\\customers.csv"
INTO TABLE customers
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
;

select count(*) from customers;

-- load sellers data
LOAD DATA local INFILE
"C:\\Users\\chins\\OneDrive\\source\\scaler_business_casestudies\\10_target_case\\T
arget- SQL Business Case\\sellers.csv"
INTO TABLE sellers
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
;

select count(*) from sellers;

-- load products data
LOAD DATA local INFILE
"C:\\Users\\chins\\OneDrive\\source\\scaler_business_casestudies\\10_target_case\\T
arget- SQL Business Case\\products.csv"
INTO TABLE products
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
```

```
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
;

select count(*) from products;

-- load orders data
LOAD DATA local INFILE
"C:\\Users\\chins\\OneDrive\\source\\scaler_business_casestudies\\10_target_case\\T
arget- SQL Business Case\\orders.csv"
INTO TABLE orders
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
;

select count(*) from orders;

-- load order_items data
LOAD DATA local INFILE
"C:\\Users\\chins\\OneDrive\\source\\scaler_business_casestudies\\10_target_case\\T
arget- SQL Business Case\\order_items.csv"
INTO TABLE order_items
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
;

select count(*) from order_items;

-- load order_payment data
LOAD DATA local INFILE
"C:\\Users\\chins\\OneDrive\\source\\scaler_business_casestudies\\10_target_case\\T
arget- SQL Business Case\\payments.csv"
INTO TABLE order_payment
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
;

select count(*) from order_payment;

-- load order_reviews data
```

```
LOAD DATA local INFILE
"C:\\Users\\chins\\OneDrive\\source\\scaler_business_casestudies\\10_target_case\\Target- SQL Business Case\\order_reviews.csv"
INTO TABLE order_reviews
CHARACTER SET latin1
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
;

select count(*) from order_reviews;
```

Create additional utility constructs

1. time series reference table.

```
/* create table to hold time series data (year-months) */
drop table if exists ts_year_month_ref;
create table ts_year_month_ref(
        year int,
    month int,
    order_year_month char(7)
);

/* create procedure to populate time series data.
 * Takes start and end dates as inputs and populates ts_year_month_ref table
 * with all yyyy-mm values between the given start and end dates.
*/
drop procedure if exists populate_ref_ts;
DELIMITER //
create procedure populate_ref_ts(IN dt1 datetime, IN dt2 datetime)
sp:begin
        declare year_curr int default extract(year from dt1);
    declare year_end int default extract(year from dt2);
    declare mon_curr int default extract(month from dt1);
    declare mon_end int default extract(month from dt2);

        while year_curr <= year_end do
                while mon_curr <= 12 do
                        insert into ts_year_month_ref values(
                                year_curr,
                                mon_curr,
                                concat(year_curr, '-', lpad(mon_curr, 2, 0))
```

```
                        );

            if(year_curr = year_end and mon_curr >= mon_end) then
                                leave sp;
                        end if;

                        set mon_curr = mon_curr + 1;
        end while;
        set mon_curr = 1;
        set year_curr = year_curr + 1;
    end while;
end //
DELIMITER ;

/* populate time series reference table */
set @min_date = (select min(order_purchase_ts) from orders);
set @max_date = (select max(order_purchase_ts) from orders);
call populate_ref_ts(@min_date, @max_date);

select * from ts_year_month_ref;
```

year	month	order_year_month
2016	9	2016-09
2016	10	2016-10
2016	11	2016-11
2016	12	2016-12
2017	1	2017-01
2017	2	2017-02
2017	3	2017-03
2017	4	2017-04
2017	5	2017-05
2017	6	2017-06
2017	7	2017-07
2017	8	2017-08
2017	9	2017-09
2017	10	2017-10
2017	11	2017-11
2017	12	2017-12
2018	1	2018-01
2018	2	2018-02
2018	3	2018-03

_year_month_ref 96 ×

2. Order details view

```
/* create view for orders with derived fields */
drop view if exists orders_details;
create view orders_details as (select
        order_id,
    customer_id,
```

```
    order_status,
    order_purchase_ts,
    extract(month from order_purchase_ts) as order_month,
    extract(year from order_purchase_ts) as order_year,
    concat(extract(year from order_purchase_ts), '-', lpad(extract(month from
order_purchase_ts), 2, 0)) as order_year_month
from orders o
where order_status not in ('canceled', 'unavailable')
order by order_year_month);

select * from orders_details;
```

order_id	customer_id	order_status	order_purchase_ts	order_month	order_year	order_year_month
bfbd0f9bdef84302105ad712db648a6c	86dc2ffce2dfff336de2f386a786e574	delivered	2016-09-15 12:16:38	9	2016	2016-09
2e7a8482f6fb09756ca50c10d7bfc047	08c5351a6aca1c1589a38f244edeee9d	shipped	2016-09-04 21:15:19	9	2016	2016-09
2d9e3c3c7f0f3ba8a8fa3db2f1211ceb	6b79d5d9914b88e0e05a19e76dfe926b	delivered	2016-10-05 17:09:07	10	2016	2016-10
2e32dea8a4d1ae5499a67674b387bc6a	7fd973e7865ee210512c6222d387e7aa	delivered	2016-10-06 02:53:39	10	2016	2016-10
2ce9683175cdab7d1c95bcbb3e36f478	b2d7ae0415dbbca535b5f7b38056dd1f	invoiced	2016-10-05 21:03:33	10	2016	2016-10
2e8e21db96a8ab922e51cd297678c6b2	6a58b195be76f8ad6544e032b6c7316b	delivered	2016-10-09 12:13:58	10	2016	2016-10
45973912e490866800c0aea8f63099c8	912f108a7026f25f99240a5c4c60e2c3	shipped	2016-10-07 22:45:28	10	2016	2016-10
42c5091238dc63339c79f5f35f8a5a6b	da231380834f900e3fc3a958a1533a33	invoiced	2016-10-10 09:57:05	10	2016	2016-10
45bff70fb7b99a5a33bc8ccb3786e170	21cf8b03074c935aa1de2d527371916f	delivered	2016-10-09 22:38:37	10	2016	2016-10
4421f094ff828b78aba6d18db005821d	497444aa14246e71ed88c2c37540ddfb	delivered	2016-10-10 10:22:55	10	2016	2016-10
323dad8c483c7f8b818a825d257f4aa0	021833e9737a1145b5a6e3053f0f4329	delivered	2016-10-06 20:05:04	10	2016	2016-10
2f9d4ee8ca77fe0951ebfe9064cad953	2d2037b93b52f5a7ac9e286e8bd69d15	delivered	2016-10-09 23:38:19	10	2016	2016-10
31b0dd6152d2e471443debf037ae171d	031f08a1ebdcfb5e706831bf64de3860	delivered	2016-10-05 12:32:55	10	2016	2016-10

ders_details 97 ×

Data exploration

1. Import the dataset and do usual exploratory analysis steps like checking the structure & characteristics of the dataset

a. Data type of columns in a table

```
desc geolocation;
```

Field	Type	Null	Key	Default
zip_code_prefix	int	NO		NULL
lat	decimal(24,20)	YES		NULL
lng	decimal(24,20)	YES		NULL
city	varchar(45)	YES		NULL
state	varchar(5)	YES		NULL

```
desc customers;
```

Field	Type	Null	Key	Default
customer_id	char(32)	NO	PRI	NULL
customer_unique_id	char(32)	NO		NULL
zip_code_prefix	int	YES		NULL
city	varchar(45)	YES		NULL
state	varchar(5)	YES	MUL	NULL

```
desc sellers;
```

Field	Type	Null	Key	Default
seller_id	char(32)	NO	PRI	NULL
zip_code_prefix	int	YES		NULL
state	varchar(5)	YES		NULL
city	varchar(45)	YES		NULL

```
desc products;
```

Field	Type	Null	Key	Default
product_id	char(32)	NO	PRI	NULL
category	varchar(100)	YES		NULL
name_length	int	YES		NULL
description_length	int	YES		NULL
photos_qty	smallint	YES		NULL
weight_g	int	YES		NULL
length_cm	int	YES		NULL
height_cm	int	YES		NULL
width_cm	int	YES		NULL

```
desc orders;
```

Field	Type	Null	Key	Default
order_id	char(32)	NO	PRI	NULL
customer_id	char(32)	YES	MUL	NULL
order_status	varchar(45)	YES		NULL
order_purchase_ts	datetime	YES		NULL
order_approved_at	datetime	YES		NULL
order_delivered_carrier_date	datetime	YES		NULL
order_delivered_customer_date	datetime	YES		NULL
order_estimated_delivery_date	datetime	YES		NULL

```
desc order_items;
```

Field	Type	Null	Key	Default
order_id	char(32)	YES	MUL	NULL
order_item_id	int	YES		NULL
product_id	char(32)	YES	MUL	NULL
seller_id	char(32)	YES	MUL	NULL
shipping_limit_date	datetime	YES		NULL
price	decimal(10,2)	YES		NULL
freight_value	decimal(10,2)	YES		NULL

```
desc order_payment;
```

Field	Type	Null	Key	Default
order_id	char(32)	YES	MUL	NULL
payment_sequential	int	YES		NULL
payment_type	varchar(45)	YES		NULL
payment_installment	int	YES		NULL
payment_value	decimal(10,2)	YES		NULL
order_paymentcol	varchar(45)	YES		NULL

```
desc order_reviews;
```

Field	Type	Null	Key	Default
review_id	char(32)	NO	PRI	NULL
order_id	char(32)	NO	PRI	NULL
review_score	tinyint	YES		NULL
review_comment_title	varchar(255)	YES		NULL
review_creation_date	datetime	YES		NULL
review_answer_timestamp	datetime	YES		NULL

b. Time period for which the data is given

```
select min(order_purchase_ts), max(order_purchase_ts) from orders;
```

min(order_purchase_ts)	max(order_purchase_ts)
2016-09-04 21:15:19	2018-10-17 17:30:18

Observations: The given data-set has orders data collected between 4th Sept 2016 and 17th Oct 2018.

c. Cities and States of customers ordered during the given period

```
/* unique states */
select distinct state from customers;
```

state
AC
AL
AM
AP
BA
CE
DF
ES
GO
MA
MG
MS
MT
PA

ustomers 68 ×

```
/* unique cities (a city name may be repeated across states, so we take unique
combination of city and state) */

select distinct city, state from customers;
```

city	state
osasco	SP
itapecerica	MG
nova venecia	ES
mendonca	MG
sao paulo	SP
valinhos	SP
niteroi	RJ
rio de janeiro	RJ
ijui	RS
oliveira	MG
novo hambu…	RS
vitoria da co…	BA
campinas	SP
jacarei	SP

Observations:

i. Customers belong to 27 unique states.

ii. Customers belong to 4310 different cities (If a city name repeats across states, it's considered different)

2. In-depth Exploration:

a. Is there a growing trend on e-commerce in Brazil? How can we describe a complete scenario? Can we see some seasonality with peaks at specific months?

To answer the first question, we first check year on year monthly growth in terms of **number of orders** as well as **total sales** (by aggregating order price). This ensures we do not see seasonal impact in our results.

```
/* calculate yoy monthly growth in total sale price and number of orders */
with order_price as
(select order_id, sum(price) as price
from order_items
group by order_id),

orders_by_year_month as
(select od.order_year_month, od.order_year, od.order_month, sum(op.price) as
total_order_sale, count(1) as total_order_count
from orders_details od inner join order_price op on od.order_id = op.order_id
group by od.order_year_month, od.order_year, od.order_month
)

select t.*,
        case
                when t.prev_yoy_total_order_sale = 0 then 0.0
        else round(((t.total_order_sale - t.prev_yoy_total_order_sale) /
t.prev_yoy_total_order_sale) * 100, 1)
    end as yoy_order_sale_growth,
    case
                when t.prev_yoy_total_order_count = 0 then 0.0
        else round(((t.total_order_count - t.prev_yoy_total_order_count) /
t.prev_yoy_total_order_count) * 100, 1)
    end as yoy_order_count_growth
from
(       select
                order_month,
                order_year,
                total_order_sale,
                lag(total_order_sale, 1, 0) over(win) as prev_yoy_total_order_sale,
                total_order_count,
                lag(total_order_count, 1, 0) over(win) as
prev_yoy_total_order_count
        from orders_by_year_month
        window win as (partition by order_month order by order_year asc)
) as t;
```

order_month	order_year	total_order_sale	prev_yoy_total_order_sale	total_order_count	prev_yoy_total_order_count	yoy_order_sale_growth	yoy_order_count_growth
1	2017	120098.27	0.00	787	0	0.0	0.0
1	2018	945456.29	120098.27	7187	787	687.2	813.2
2	2017	244959.35	0.00	1718	0	0.0	0.0
2	2018	837895.43	244959.35	6624	1718	242.1	285.6
3	2017	368341.32	0.00	2617	0	0.0	0.0
3	2018	981051.06	368341.32	7168	2617	166.3	173.9
4	2017	353842.98	0.00	2377	0	0.0	0.0
4	2018	993592.98	353842.98	6919	2377	180.8	191.1
5	2017	503159.19	0.00	3640	0	0.0	0.0
5	2018	992871.75	503159.19	6833	3640	97.3	87.7
6	2017	429916.61	0.00	3205	0	0.0	0.0
6	2018	863265.53	429916.61	6145	3205	100.8	91.7
7	2017	492287.30	0.00	3946	0	0.0	0.0
7	2018	878044.27	492287.30	6233	3946	78.4	58.0
8	2017	568245.79	0.00	4272	0	0.0	0.0
8	2018	848860.10	568245.79	6421	4272	49.4	50.3
9	2016	207.86	0.00	2	0	0.0	0.0
9	2017	621415.91	207.86	4227	2	298858.9	211250.0

sult 131 ×

Observations: year on year growth in monthly sales ranges from 49.4% (for 2018-Aug) to 1383% (for 2017-Oct). Similarly year on year growth in monthly order count ranges from 50% (2018-Aug) to 1467% (for 2017-Oct). In general, we see high year on year growth percentage across all months. Thus, based on the given sample, we can conclude that Target in Brazil is experiencing upward trend.

Next, we check for seasonality at months level. We will compute total sales and total orders at month level (across 2016-2018). We also compute average monthly sales and average total orders by diving total sales/count by number of unique years data available.

```
with order_price as
(select order_id, sum(price) as price
from order_items
group by order_id),

monthly_sales_data as
(select
        od.order_month as month,
        count(distinct order_year) as unique_years,
    sum(op.price) as total_monthly_sale_all_years,
    count(*) as total_orders_all_years,
    (sum(op.price) / count(distinct order_year)) as avg_monthly_sale,
    (count(1) / count(distinct order_year)) as avg_monthly_orders

from orders_details od inner join order_price op on od.order_id = op.order_id
group by od.order_month
)

select * from monthly_sales_data order by avg_monthly_sale desc;
```

month	unique_years	total_monthly_sale_all_years	total_orders_all_years	avg_monthly_sale	avg_monthly_orders
11	1	1003862.14	7421	1003862.140000	7421.0000
5	2	1496030.94	10473	748015.470000	5236.5000
8	2	1417105.89	10693	708552.945000	5346.5000
7	2	1370331.57	10179	685165.785000	5089.5000
3	2	1349392.38	9785	674696.190000	4892.5000
4	2	1347435.96	9296	673717.980000	4648.0000
6	2	1293182.14	9350	646591.070000	4675.0000
2	2	1082854.78	8342	541427.390000	4171.0000
1	2	1065554.56	7974	532777.280000	3987.0000
12	2	742194.69	5619	371097.345000	2809.5000
10	2	704686.92	4837	352343.460000	2418.5000
9	3	621768.77	4230	207256.256667	1410.0000

```
select * from monthly_sales_data order by avg_monthly_orders desc;
```

month	unique_years	total_monthly_sale_all_years	total_orders_all_years	avg_monthly_sale	avg_monthly_orders
11	1	1003862.14	7421	1003862.140000	7421.0000
8	2	1417105.89	10693	708552.945000	5346.5000
5	2	1496030.94	10473	748015.470000	5236.5000
7	2	1370331.57	10179	685165.785000	5089.5000
3	2	1349392.38	9785	674696.190000	4892.5000
6	2	1293182.14	9350	646591.070000	4675.0000
4	2	1347435.96	9296	673717.980000	4648.0000
2	2	1082854.78	8342	541427.390000	4171.0000
1	2	1065554.56	7974	532777.280000	3987.0000
12	2	742194.69	5619	371097.345000	2809.5000
10	2	704686.92	4837	352343.460000	2418.5000
9	3	621768.77	4230	207256.256667	1410.0000

Observations: We observe that top 3 months in terms of both average total sale and average orders count are November, August, and May. Similarly, bottom 3 months in terms of average total sale and average orders count are Sept, Oct, and December. Thus we do see seasonality in the given data at monthly level.

b. What time do Brazilian customers tend to buy (Dawn, Morning, Afternoon or Night)?

```
/* find total orders */
set @total_orders = (select count(1) from orders);

/*
   Orders by time of the day.
   dawn - 5-6
   morning - 7 to 11
   afternoon - 12 to 16
   evening - 17 to 21
   night - 22 - 4
*/
select
```

```
        case
                when hour(order_purchase_ts) between 5 and 6 then 'dawn'
                        when hour(order_purchase_ts) between 7 and 11 then
'morning'
                        when hour(order_purchase_ts) between 12 and 16 then
'afternoon'
                        when hour(order_purchase_ts) between 17 and 21 then
'evening'
                        when hour(order_purchase_ts) between 22 and 23 then 'night'
                        when hour(order_purchase_ts) between 0 and 4 then 'night'
                else 'invalid time'
        end time_of_day,
    count(*) as order_count,
    round((count(1) / @total_orders) * 100, 2) as order_percent
from orders
group by time_of_day
order by order_count desc;
```

time_of_day	order_count	order_percent
afternoon	32211	32.39
evening	30311	30.48
morning	21738	21.86
night	14491	14.57
dawn	690	0.69

```
/* By each hour */
select
        hour(order_purchase_ts) as purchase_hour,
    count(1) as order_count,
    round((count(1) / @total_orders) * 100, 2) as order_percent
from orders
group by purchase_hour
order by order_count desc;
```

purchase_hour	order_count	order_percent
16	6675	6.71
11	6578	6.61
14	6569	6.61
13	6518	6.55
15	6454	6.49
21	6217	6.25
20	6193	6.23
10	6177	6.21
17	6150	6.18
12	5995	6.03
19	5982	6.02
22	5816	5.85
18	5769	5.80
9	4785	4.81
23	4123	4.15
8	2967	2.98
0	2394	2.41
7	1231	1.24
1	1170	1.18

Observations:

• Maximum orders (around 32.4%) are placed during afternoon hours (12.00 to 16.59).

• Second highest orders (around 30.48%) are placed during evening hours (17.00 to 21.59).

• Around 21.86% orders are placed during morning hours (7.00 to 11.59) and around 14.57% orders are placed during night hours (22.00 to 4.59).

• Less than one percent (0.69%) orders are placed around dawn (5.00 to 6.59).

• In terms of hours, top 10 hours receiving highest orders are : 16, 11,14, 13, 15, 21, 20, 10, 17, and 12.

3. Evolution of E-commerce orders in the Brazil region:

a. Get month on month orders by states

```
with
/* cross join of state and timeseries reference table to produce all combinations
*/
state_ts as (
        select *
    from (select distinct state from customers) as states, ts_year_month_ref
),

/*
        To address gaps in timeseries, we take left outer join of state_ts with
orders_details and customers table.
    We then group by (state and order_year_month) and count number of orders per
group.
*/
orders_by_state_and_year_month as (
```

```
        select
                s_ts.state,
                s_ts.order_year_month,
        count(t.order_purchase_ts) as order_count,
        lag(count(t.order_purchase_ts), 1, 0) over(partition by s_ts.state order by
s_ts.order_year_month asc) as prev_month_order_count
        from state_ts s_ts
                left outer join(
                                select c.state, o.order_purchase_ts,
o.order_year_month
                                from orders_details o join customers c
                                        on o.customer_id = c.customer_id
                ) as t on t.state = s_ts.state and t.order_year_month =
s_ts.order_year_month
        group by s_ts.state, s_ts.order_year_month
        order by s_ts.state, s_ts.order_year_month
)

/* show mom orders by state */
select
        state,
        order_year_month,
    order_count,
        prev_month_order_count,
    case
                when prev_month_order_count = 0 then 0.0
        else round(((order_count - prev_month_order_count) /
prev_month_order_count) * 100, 1)
    end as mom_order_growth_percent
from orders_by_state_and_year_month;
```

state	order_year_month	order_count	prev_month_order_count	mom_order_growth_percent
AC	2016-09	0	0	0.0
AC	2016-10	0	0	0.0
AC	2016-11	0	0	0.0
AC	2016-12	0	0	0.0
AC	2017-01	2	0	0.0
AC	2017-02	3	2	50.0
AC	2017-03	2	3	-33.3
AC	2017-04	5	2	150.0
AC	2017-05	8	5	60.0
AC	2017-06	4	8	-50.0
AC	2017-07	5	4	25.0
AC	2017-08	4	5	-20.0
AC	2017-09	5	4	25.0
AC	2017-10	6	5	20.0
AC	2017-11	5	6	-16.7
AC	2017-12	5	5	0.0
AC	2018-01	6	5	20.0
AC	2018-02	3	6	-50.0
AC	2018-03	2	3	-33.3

b. Distribution of customers across the states in Brazil

```
set @total_cust = (select count(*) from customers);
select
    state,
    count(*) as customer_count,
    round((count(*) / @total_cust) * 100, 2) as customer_percent
from customers
group by state
order by count(*) desc
```

state	customer_count	customer_percent
SP	41746	41.98
RJ	12852	12.92
MG	11635	11.70
RS	5466	5.50
PR	5045	5.07
SC	3637	3.66
BA	3380	3.40
DF	2140	2.15
ES	2033	2.04
GO	2020	2.03
PE	1652	1.66
CE	1336	1.34
PA	975	0.98
MT	907	0.91
MA	747	0.75
MS	715	0.72
PB	536	0.54
PI	495	0.50
RN	485	0.49

Observations: State **SP** constitute highest number of customers (42%), states **RJ** and **MG** follow next constituting 13% and 11.7% respectively. State **RS** and **PR** constitute around 5.5% and 5% respectively. The contribution of remaining states is low (ranging from 3.66% to 0.05%).

c. Distribution of sales and orders by States

```
select c.state, sum(order_price) as total_sale, count(*) as total_orders
from orders o
        inner join customers c on o.customer_id = c.customer_id
    inner join (select order_id, sum(price) as order_price from order_items group
by order_id) as op on o.order_id = op.order_id
group by c.state
order by total_sale desc
```

state	total_sale
SP	5202955.05
RJ	1824092.67
MG	1585308.03
RS	750304.02
PR	683083.76
SC	520553.34
BA	511349.99
DF	302603.94
GO	294591.95
ES	275037.31
PE	262788.03

sult 138

Observations: SP, RJ and MG are top 3 states with highest total sales and total orders. AC, AP, RR are bottom 3 states with lowest total sales and total orders.

4. Impact on Economy: Analyze the money movement by e-commerce by looking at order prices, freight and others.

a. Get % increase in cost of orders from 2017 to 2018 (include months between Jan to Aug only) - You can use "payment_value" column in payments table

```
/* compute total cost by year */
with cost_by_year as (
        select o.order_year, sum(op.payment_value) as total_order_cost
        from (
                select
                order_id,
                extract(month from order_purchase_ts) as order_month,
                extract(year from order_purchase_ts) as order_year
                from orders o where order_status not in ('canceled', 'unavailable')
        ) as o
        join order_payment as op on o.order_id = op.order_id
        where o.order_month between 1 and 8 and o.order_year in (2017, 2018)
        group by o.order_year
        order by o.order_year asc
)

select order_year,
        case
                when prev_year_order_cost = 0 then 0.0
        else round((total_order_cost - prev_year_order_cost) * 100 /
prev_year_order_cost , 1)
    end as cost_growth
from (
        select
        order_year,
    total_order_cost,
    lag(total_order_cost, 1, 0) over(order by order_year asc) as
prev_year_order_cost
        from cost_by_year
) as t;
```

order_year	cost_growth
2017	0.0
2018	140.3

Observation: Overall we see increase of 140.3% in total orders cost from 2017 to 2018.

b. Mean & Sum of price and freight value by customer state

```
with price_freight_by_State as
(select
         c.state,
         sum(order_price) as total_price,
         avg(order_price) as avg_price,
         sum(order_freight_value) as total_freight_value,
         avg(order_freight_value) as avg_freight_value
from
        /* aggregate price and freight values at order level*/
    (select
            o.order_id,
            o.customer_id,
            sum(oi.price) as order_price,
            sum(oi.freight_value) as order_freight_value
        from orders o join order_items oi on o.order_id = oi.order_id
        group by o.order_id, o.customer_id) as t
    inner join customers c on t.customer_id = c.customer_id
group by c.state)

select * from price_freight_by_State order by total_price desc;
select * from price_freight_by_State order by avg_price desc;
select * from price_freight_by_State order by total_freight_value desc;
select * from price_freight_by_State order by avg_freight_value desc;
```

state	total_price	avg_price	total_freight_value	avg_freight_value
SP	5202955.05	125.751179	718723.07	17.370950
RJ	1824092.67	142.931568	305589.31	23.945252
MG	1585308.03	137.327445	270853.46	23.462704
RS	750304.02	138.126661	135522.74	24.948958
PR	683083.76	136.671421	117851.68	23.579768
SC	520553.34	144.117757	89660.26	24.822885
BA	511349.99	152.278139	100156.68	29.826289
DF	302603.94	142.401854	50625.50	23.823765
GO	294591.95	146.782237	53114.98	26.464863
ES	275037.31	135.820894	49764.60	24.575111
PE	262788.03	159.458756	59449.66	36.073823
CE	227254.71	171.254491	48351.59	36.436767
PA	178947.81	184.482278	38699.30	39.896186
MT	156453.53	173.259723	29715.43	32.907453
MA	119648.22	161.686784	31523.77	42.599689
MS	116812.64	164.756897	19144.03	27.001453
PB	115268.08	216.669323	25719.73	48.345357
PI	86914.08	176.296308	21218.20	43.038945
RN	83034.98	172.271743	18860.10	39.128838

Observations:

1. states by total price:
    1. top 3 - SP, RG, MG
    2. bottom 3 - AC, AP, RR
2. states by average price:
    1. top 3 - PB, AP, AC
    2. bottom 3 - PR, ES, SP
3. states by total freight value:
    1. top 3 - SP, RJ, MG
    2. bottom 3 - AC, AP, RR
4. states by average freight value:
    1. top 3: RR, PB, RO
    2. bottom 3: PR, MG, SP

5. Analysis on sales, freight and delivery time

a. Calculate days between purchasing, delivering and estimated delivery.
b. Find time_to_delivery & diff_estimated_delivery. Formula for the same given below:
i. time_to_delivery = order_purchase_timestamp-order_delivered_customer_date
ii. diff_estimated_delivery = order_estimated_delivery_date-order_delivered_customer_date

```
with delivery_details as(
select
        order_id,
        order_purchase_ts,
        order_estimated_delivery_date,
        datediff(order_estimated_delivery_date, order_purchase_ts) estimated_days,
    datediff(order_delivered_customer_date, order_purchase_ts) actual_days,
    datediff(order_estimated_delivery_date, order_delivered_customer_date)
estimated_actual_diff
from orders
where order_status = 'delivered')

select * from delivery_details;
```

order_id	order_purchase_ts	order_estimated_delivery_date	estimated_time_to_delivery	time_to_delivery	diff_estimated_delivery
00010242fe8c5a6d1ba2dd792cb16214	2017-09-13 08:59:02	2017-09-29 00:00:00	16	7	9
00018f77f2f0320c557190d7a144bdd3	2017-04-26 10:53:06	2017-05-15 00:00:00	19	16	3
000229ec398224ef6ca0657da4fc703e	2018-01-14 14:33:31	2018-02-05 00:00:00	22	8	14
00024acbcdf0a6daa1e931b038114c75	2018-08-08 10:00:35	2018-08-20 00:00:00	12	6	6
00042b26cf59d7ce69dfabb4e55b4fd9	2017-02-04 13:57:51	2017-03-17 00:00:00	41	25	16
00048cc3ae777c65dbb7d2a0634bc1ea	2017-05-15 21:42:34	2017-06-06 00:00:00	22	7	15
00054e8431b9d7675808bcb819fb4a32	2017-12-10 11:53:48	2018-01-04 00:00:00	25	8	17
000576fe39319847cbb9d288c5617fa6	2018-07-04 12:08:27	2018-07-25 00:00:00	21	5	16
0005a1a1728c9d785b8e2b08b904576c	2018-03-19 18:40:33	2018-03-29 00:00:00	10	10	0
0005f50442cb953dcd1d21e1fb923495	2018-07-02 13:59:39	2018-07-23 00:00:00	21	2	19
00061f2a7bc09da83e415a52dc8a4af1	2018-03-24 22:16:10	2018-04-09 00:00:00	16	5	11
00063b381e2406b52ad429470734ebd5	2018-07-27 17:21:27	2018-08-07 00:00:00	11	11	0
0006ec9db01a64e59a68b2c340bf65a7	2018-07-24 17:04:17	2018-08-22 00:00:00	29	7	22
0008288aa423d2a3f00fcb17cd7d8719	2018-02-13 22:10:21	2018-03-06 00:00:00	21	13	8
0009792311464db532ff765bf7b182ae	2018-08-14 20:43:09	2018-08-28 00:00:00	14	8	6
0009c9a17f916a706d71784483a5d643	2018-04-25 09:10:41	2018-05-09 00:00:00	14	5	9
000aed2e25dbad2f9ddb70584c5a2ded	2018-05-11 20:33:38	2018-05-22 00:00:00	11	7	4
000c3e6612759851cc3cbb4b83257986	2017-08-12 10:08:57	2017-09-01 00:00:00	20	7	13
000e562887b1f2006d75e0be9558292e	2018-02-22 11:54:42	2018-03-19 00:00:00	25	18	7

sult 102 ×

c. Group data by state, take mean of freight_value, time_to_delivery, diff_estimated_delivery

```
/* calculate time_to_delivery and diff_estimated_delivery for delivered orders */
drop view if exists delivery_details;
create view delivery_details as
(select order_id, order_purchase_ts, order_estimated_delivery_date,
    datediff(order_delivered_customer_date, order_purchase_ts) time_to_delivery,
    datediff(order_estimated_delivery_date, order_delivered_customer_date)
diff_estimated_delivery
from orders
where order_status = 'delivered');

/* aggregate price and freight at order level */
drop view if exists order_price_details;
create view order_price_details as
(select o.order_id, o.customer_id, c.state, sum(oi.price) as order_price,
sum(oi.freight_value) as order_freight_value
        from orders o
                inner join order_items oi on o.order_id = oi.order_id
        inner join customers c on o.customer_id = c.customer_id
        group by o.order_id, o.customer_id, c.state);

/* aggregate price, freight, time_to_delivery, and diff_estimated_delivery at state
level */
drop view if exists state_delivery_metrics;
create view state_delivery_metrics as
(select od.state,
        avg(order_freight_value) avg_freight_value,
    avg(time_to_delivery) avg_time_to_delivery,
    avg(diff_estimated_delivery) avg_diff_estimated_delivery
from order_price_details od inner join delivery_details dd on od.order_id =
```

```
dd.order_id
group by od.state);
```

d. Sort the data to get the following:

i. Top 5 states with highest/lowest average freight value - sort in desc/asc limit 5

```
/* top 5 states with lowest avg_freight_value */
select * from state_delivery_metrics
order by avg_freight_value asc limit 5;
```

state	avg_freight_value	avg_time_to_delivery	avg_diff_estimated_delivery
SP	17.334633	8.7006	11.0755
MG	23.463963	11.9450	13.2430
PR	23.490817	11.9380	13.3142
DF	23.858144	12.8990	12.0481
RJ	23.947404	15.2370	11.7612

```
/* top 5 states with highest avg freight value */
select * from state_delivery_metrics
order by avg_freight_value desc limit 5;
```

state	avg_freight_value	avg_time_to_delivery	avg_diff_estimated_delivery
PB	48.842805	20.3888	13.2611
RR	48.342683	29.3415	17.2927
RO	46.433086	19.2840	20.1029
AC	45.554500	21.0000	20.7250
PI	42.977290	19.3950	11.3067

ii. Top 5 states with highest/lowest average time to delivery

```
/* top 5 states with lowest avg_time_to_delivery */
select * from state_delivery_metrics
order by avg_time_to_delivery asc limit 5;
```

state	avg_freight_value	avg_time_to_delivery	avg_diff_estimated_delivery
SP	17.334633	8.7006	11.0755
PR	23.490817	11.9380	13.3142
MG	23.463963	11.9450	13.2430
DF	23.858144	12.8990	12.0481
SC	24.849309	14.9030	11.5034

```
/* top 5 states with highest avg_time_to_delivery */
select * from state_delivery_metrics
```

```
order by avg_time_to_delivery desc limit 5;
```

state	avg_freight_value	avg_time_to_delivery	avg_diff_estimated_delivery
RR	48.342683	29.3415	17.2927
AP	41.298507	27.1791	19.6866
AM	37.445724	26.3586	19.5655
AL	38.581285	24.5013	8.7078
PA	39.696596	23.7252	14.0666

iii. Top 5 states where delivery is really fast/ not so fast compared to estimated date

```
/*top 5 states with lowest (not so fast) avg_diff_estimated_delivery */
select * from state_delivery_metrics
order by avg_diff_estimated_delivery asc limit 5;
```

state	avg_freight_value	avg_time_to_delivery	avg_diff_estimated_delivery
AL	38.581285	24.5013	8.7078
MA	42.948633	21.5119	9.5718
SE	40.940119	21.4627	10.0209
ES	24.568662	15.7238	10.4962
BA	29.961201	19.2786	10.7945

```
/*top 5 states with highest (faster) avg_diff_estimated_delivery */
select * from state_delivery_metrics
order by avg_diff_estimated_delivery desc limit 5;
```

state	avg_freight_value	avg_time_to_delivery	avg_diff_estimated_delivery
AC	45.554500	21.0000	20.7250
RO	46.433086	19.2840	20.1029
AP	41.298507	27.1791	19.6866
AM	37.445724	26.3586	19.5655
RR	48.342683	29.3415	17.2927

Observations:

1. States with lowest average freight value:
    1. top 5: SP, MG, PR, DF, RJ
    2. bottom 5: PB, RR, RO, AC, PI
2. States with shortest average actual delivery time
    1. top 5: SP, PR, MG, DF, SC
    2. bottom 5: RR, AP, AM, AL, PA
3. States with highest (faster) average difference between estimated and actual delivery time
    1. top 5: AC, RO, AP, AM, RR
    2. bottom 5: AL, MA, SE, ES, BA

6. Payment type analysis:

a. Month over Month count of orders for different payment types

```
with
/*cross join payment_type with time series */
payment_type_ts as (
        select *
    from (select distinct payment_type from order_payment) as payment_types,
ts_year_month_ref
),

/* distinct order and payment_type */
order_payment_type as (select distinct order_id, payment_type from order_payment),

/* aggregated orders by payment type and time series */
orders_by_payment_type_and_year_month as (
        select
                pt_ts.payment_type,
                pt_ts.order_year_month,
        count(t.order_purchase_ts) as order_count,
        lag(count(t.order_purchase_ts), 1, 0) over(partition by pt_ts.payment_type
order by pt_ts.order_year_month asc) as prev_month_order_count
        from payment_type_ts pt_ts
                left outer join (
                        select
                                op.payment_type,
                                o.order_purchase_ts,
                                o.order_year_month
                        from orders_details o
                                join order_payment_type op on o.order_id =
op.order_id
                ) as t
                        on t.payment_type = pt_ts.payment_type and
                        t.order_year_month = pt_ts.order_year_month
        group by pt_ts.payment_type, pt_ts.order_year_month
        order by pt_ts.payment_type, pt_ts.order_year_month
)

/* show mom orders by payment_type */
select payment_type,
        order_year_month,
    order_count,
        prev_month_order_count,
    case
                when prev_month_order_count = 0 then 0.0
```

```
        else round(((order_count - prev_month_order_count) /
prev_month_order_count) * 100, 1)
    end as mom_order_growth_percent
from orders_by_payment_type_and_year_month;
```

payment_type	order_year_month	order_count	prev_month_order_count	mom_order_growth_percent
credit_card	2017-12	4319	5782	-25.3
credit_card	2018-01	5449	4319	26.2
credit_card	2018-02	5149	5449	-5.5
credit_card	2018-03	5641	5149	9.6
credit_card	2018-04	5424	5641	-3.8
credit_card	2018-05	5443	5424	0.4
credit_card	2018-06	4778	5443	-12.2
credit_card	2018-07	4698	4778	-1.7
credit_card	2018-08	4933	4698	5.0
credit_card	2018-09	0	4933	-100.0
credit_card	2018-10	0	0	0.0
debit_card	2016-09	0	0	0.0
debit_card	2016-10	2	0	0.0
debit_card	2016-11	0	2	-100.0
debit_card	2016-12	0	0	0.0
debit_card	2017-01	9	0	0.0
debit_card	2017-02	13	9	44.4
debit_card	2017-03	30	13	130.8

sult 110 ×

b. Count of orders based on the no. of payment installments

```
set @order_count = (select count(distinct order_id) from order_payment);

select t.payment_installment, count(*) as total_orders, round((count(*)/
@order_count)*100, 2) as orders_percent
from
(select order_id, payment_installment, sum(payment_value) as payment_value
from order_payment op
group by order_id, payment_installment) as t
group by t.payment_installment
order by total_orders desc
```

payment_installment	total_orders	orders_percent
1	49060	49.34
2	12389	12.46
3	10443	10.50
4	7088	7.13
10	5315	5.34
5	5234	5.26
8	4253	4.28
6	3916	3.94
7	1623	1.63
9	644	0.65
12	133	0.13
15	74	0.07
18	27	0.03
11	23	0.02
24	18	0.02
20	17	0.02

c. orders by payment_type

```
/* orders by payment type */
select op.payment_type, sum(payment_value) as total_payment, count(*) as
total_orders
from orders o
        inner join customers c on o.customer_id = c.customer_id
    inner join (select order_id, payment_type, sum(payment_value) as payment_value
from order_payment group by order_id, payment_type) as op on o.order_id =
op.order_id
group by op.payment_type
order by total_payment desc
```

payment_type	total_payment	total_orders
credit_card	12542084.19	76505
UPI	2869361.27	19784
voucher	379436.87	3866
debit_card	217989.79	1528
not_defined	0.00	3

Observations:

1. Around 49% orders are paid in single installment. Next highest percent orders are paid with 2 installments (12.46%), 3 installments (10.50%), 4 installments (7.13%) and 10 installments (5.34%)
2. Around 75% orders are paid through credit cards, 19% paid through UPI. Around 78% of payment received is through credit cards, 18% through UPI.

Insights and Recommendations

- **Customers:**

    1. Customers belong to 27 unique states and 4310 different cities.
    2. State **SP** constitute highest number of customers (42%), states **RJ** and **MG** follow next constituting 13% and 11.7% respectively. State **RS** and **PR** constitute around 5.5% and 5% respectively. The contribution of remaining states is low (ranging from 3.66% to 0.05%).

    - **Recommendations:**
        - A large proportion of customers (43%) belong to SP state. Business therefore should keep SP state central to its plans and strategies.
        - There is a further potential to grow customer base in states RJ, MG, RS, and PR.

- **Ecommerce trend**

    1. year on year growth in monthly sales ranges from 49.4% (for 2018-Aug) to 1383% (for 2017-Oct). Similarly year on year growth in monthly order count ranges from 50% (2018-Aug) to 1467% (for 2017-Oct). In general, we see high year on year growth percentage across all months. Thus, based on the given sample, we can conclude that Target in Brazil is experiencing upward trend.
    2. We observe that top 3 months in terms of both average total sale and average orders count are November, August, and May. Similarly, bottom 3 months in terms of average total sale and average orders count are Sept, Oct, and December. Thus we do see seasonality in the given data at monthly level.

    - **Recommendations:**
        - There is a clear upward trend in ecommerce in Brazil across states from 2017 to 2018. The business should continue to plan for growth and expansion.
        - November, August, and May months bring the maximum orders and sales. Sept, Oct, and December, on the other hand, see relatively lower orders and sales. Business can plan promotional activities, discounts, schemes to increase sales, attract new customers, and increase sales further.

- **Purchase time**

    1. Maximum orders (around 32.4%) are placed during afternoon hours (12.00 to 16.59).
    2. Second highest orders (around 30.48%) are placed during evening hours (17.00 to 21.59).
    3. Around 21.86% orders are placed during morning hours (7.00 to 11.59) and around 14.57% orders are placed during night hours (22.00 to 4.59).

4. less than one percent (0.69%) orders are placed around dawn (5.00 to 6.59).
5. In terms of hours, top 10 hours receiving highest orders are : 16, 11,14, 13, 15, 21, 20, 10, 17, and 12.

- **Recommendations**
  - Users are most active during afternoon, evening, and morning hours. The business can plan launch of new products, schemes during these times to catch users attention more.

- **orders and sales distribution**

1. Overall we see increase of 140.3% in total orders cost from 2017 to 2018.
2. SP, RJ and MG are top 3 states with highest total sales and total orders. AC, AP, RR are bottom 3 states with lowest total sales and total orders.

- **Recommendations:**
  - The business should continue to prioritize SP, RJ and MG states in its plans and strategies as they are top 3 states in terms of sales and orders.

- **Freight, price, and delivery time analysis**

1. States with lowest average freight value:
   - top 5: SP, MG, PR, DF, RJ
   - bottom 5: PB, RR, RO, AC, PI
2. States with shortest average actual delivery time
   - top 5: SP, PR, MG, DF, SC
   - bottom 5: RR, AP, AM, AL, PA
3. States with highest (faster) average difference between estimated and actual delivery time
   - top 5: AC, RO, AP, AM, RR
   - bottom 5: AL, MA, SE, ES, BA
4. states by total price:
   - top 3 - SP, RG, MG
   - bottom 3 - AC, AP, RR
5. states by average price:
   - top 3 - PB, AP, AC
   - bottom 3 - PR, ES, SP

- **Recommendations:**
  - PB, RR, RO, AC, PI have highest average freight value. RR, AP, AM, AL, PA states, on the other hand have longest average delivery time. The business can continue to invest in logistics arrangements/partners to bring down delivery costs and speed

up delivery time. This will help get more business and in turn increase business revenue in these states.
    - PB, AP, AC states have highest average order price. This could potentially indicate presence of premium customers with more affordability. Business can explore targeting these consumers for more premium offerings and products.

- **payments**

    1. Around 49% orders are paid in single installment. Next highest percent orders are paid with 2 installments (12.46%), 3 installments (10.50%), 4 installments (7.13%) and 10 installments (5.34%)
    2. Around 75% orders are paid through credit cards, 19% paid through UPI. Around 78% of payment received is through credit cards, 18% through UPI.
    - **Recommendations**
        - Credit card is the most common instrument for payment constituting around 75% orders and 78% of total payment value. This indicates that customers are open and willing to purchase products on credit. This represents a good opportunity for the business to bring new credit card offerings (for instance in collaboration with banks or NBFCs) offering exclusive benefits on purchasing product on Target site and thereby tapping into untapped customer segment.
        - Half of the orders are paid in single installment. There may be scope of increasing sales by offering loan schemes (such as zero EMI loans) to attract customers with low affordability to opt for EMI plans to buy products.