```sql
select * from customers;
select * from restaurants;
select * from orders;
select * from riders;
select * from deliveries;

--- Checking for Null Values

select * from Customers
where
   customer_id is null or
   customer_name is null or
   reg_date is null;

select * from Restaurants
where
   restaurant_name is null or
   opening_hours is null or
   city is null;

select * from Orders
where
   order_id is null or
   order_item is null or
   order_date is null or
   Order_time is null or
   order_status is null or
   total_amount is null;

select * from Riders
where
   rider_id is null or
   rider_name is null or
   sign_up is null;

select * from Deliveries
where
   delivery_id is null or
   order_id is null or
   rider_id is null or
   delivery_status is null;

--- There are no null values in the dataset
```

```sql
-- Checking for duplicates

select 'customers' as table_name, count(*) - count(distinct customer_id) as
duplicate_count from customers
union all
select 'restaurants', count(*) - count(distinct restaurant_id) from restaurants
union all
select 'orders', count(*) - count(distinct order_id) from orders
union all
select 'deliveries', count(*) - count(distinct delivery_id) from deliveries
union all
select 'riders', count(*) - count(distinct rider_id) from riders;

-- There are no duplicates in this dataset

-- Q1 Write a query to find the top 5 most frequently ordered dishes by customer called
"Arjun Mehta" in the last 2 years
select customer_name, dishes, rank_of_dishes
from
(
select
   c.customer_id,
   c.customer_name,
   o.order_item as dishes,
   count(o.order_item) as nr_of_time_item_order,
   dense_rank() over (order by count(o.order_item) desc) as rank_of_dishes
from orders o
join customers c
on o.customer_id = c.customer_id
where
   c.customer_name = 'Arjun Mehta'
   and order_date > current_date - interval '2 years'
group by 1,2,3
)sub
where rank_of_dishes <= 5;

-- Q2 Popular Time slot: Identify the time Slots during which the most orders are placed.
based on 2 hours interval
with popular_time_slot as
(
select
   *,
   case
      when extract(hour from order_time) between 0 and 1 then '00:00 - 02:00'
```

```
        when extract(hour from order_time) between 2 and 3 then '02:00 - 04:00'
        when extract(hour from order_time) between 4 and 5 then '04:00 - 06:00'
        when extract(hour from order_time) between 6 and 7 then '06:00 - 08:00'
        when extract(hour from order_time) between 8 and 9 then '08:00 - 10:00'
        when extract(hour from order_time) between 10 and 11 then '10:00 - 12:00'
        when extract(hour from order_time) between 12 and 13 then '12:00 - 14:00'
        when extract(hour from order_time) between 14 and 15 then '14:00 - 16:00'
        when extract(hour from order_time) between 16 and 17 then '16:00 - 18:00'
        when extract(hour from order_time) between 18 and 19 then '18:00 - 20:00'
        when extract(hour from order_time) between 20 and 21 then '20:00 - 22:00'
        when extract(hour from order_time) between 22 and 23 then '22:00 - 00:00'
    end as time_slot
from orders
)
select time_slot, count(*) as total_orders
from popular_time_slot
group by 1
order by 2 desc;

-- Q3 Order Value Analysis: Find the Average Order value per customer who has placed
more than 750 orders.
-- Return Customer_name, and AOV(Average Order Value)
select
    c.customer_id,
    c.customer_name,
    cast(avg(o.total_amount) as decimal(10, 2)) as avg_order_value
from customers c
join orders o
on c.customer_id = o.customer_id
group by c.customer_id, c.customer_name
having count(o.order_id) > 750
order by 3 desc;

-- Q4  High Value Customers: List the Customers who have spent more than 100K in total
on food orders.
-- Return customer_name, and customer_id
select
    c.customer_id,
    c.customer_name,
    sum(o.total_amount) as total_spent
from customers c
join orders o
on c.customer_id = o.customer_id
group by c.customer_id, c.customer_name
```

```sql
having sum(o.total_amount) > 100000
order by 3 desc;

-- Q5 Orders without Delivery: Write query to find orders that were placed but not delivered.
Return each restaurant name,
-- city and number of not delivered orders.Here we have to include both cases where orders
was not fulfilled and Delivery
-- status is "Not delivered".
select
   r.restaurant_name,
   r.city,
   count(o.order_id) as total_not_delivered_orders
from orders o
left join deliveries d
on o.order_id = d.order_id
left join restaurants r
on r.restaurant_id = o.restaurant_id
where d.delivery_status = 'Not Delivered' or d.delivery_status is null
group by 1,2
order by 3 desc;

-- Q6 Restaurant Revenue Ranking: Rank restaurants by their total revenue from the last
year including their name, Total
-- Revenue, and rank within their city. Display the top 3 Restaurants in their City based on
Their Highest Revenue
with ranking_table as
(
select
   r.city as city,
   r.restaurant_name as restaurant,
   sum(o.total_amount) as revenue,
   dense_rank() over(partition by r.city order by sum(o.total_amount) desc) as
rank_of_restaurant
from orders o
left join restaurants r
on r.restaurant_id = o.restaurant_id
where extract(year from o.order_date) < 2024
group by r.city, r.restaurant_name
)
select city, restaurant
from ranking_table
where rank_of_restaurant <= 3;
```

```
-- Q7 Most popular dish by City: Identify the Most Popular dish in each city based on the
number of order
with most_popular_dish as
(
select
   r.city,
   o.order_item as dishes,
   count(o.order_id) as nr_of_orders,
   dense_rank() over(partition by r.city order by count(o.order_id) desc) as rank_of_dish
from orders o
left join restaurants r
on r.restaurant_id = o.restaurant_id
group by r.city, o.order_item
)
select *
from most_popular_dish
where rank_of_dish = 1
order by 3 desc;

-- Q8 Customer Churn: Find Customers who haven't placed an order in 2024 but did in
2023
select distinct c.*
from orders o
left join customers c
   on o.customer_id = c.customer_id
where
   extract (year from o.order_date) = 2023
   and c.customer_id not in (
      select distinct customer_id
      from orders
      where extract(year from order_date) = 2024
      )
order by c.customer_id;

-- Q9 Cancelled Rate Comparison:Calculate and Compare the order Cancellation rate for
each restaurant between the currrent
-- year and previous year
with cancelled_rate_2023 as
(
select
   o.restaurant_id,
   count(o.order_id) as total_orders,
   count(case when d.delivery_id is null then 1 end) as nr_of_cancelled_orders
from orders o
left join deliveries d
on d.order_id = o.order_id
where extract(year from o.order_date) = 2023
group by 1
),
cancelled_rate_2024 as
(
select
   o.restaurant_id,
   count(o.order_id) as total_orders,
   count(case when d.delivery_id is null then 1 end) as nr_of_cancelled_orders
from orders o
left join deliveries d
on d.order_id = o.order_id
where extract(year from o.order_date) = 2024
group by 1
),
last_year_data as
(
select
   restaurant_id,
   total_orders,
   nr_of_cancelled_orders,
   cast(nr_of_cancelled_orders as decimal(10,2)) / cast(total_orders as decimal(10,2)) *
100 as cancel_percent
from cancelled_rate_2023
),
current_year_data as
(
select
   restaurant_id,
   total_orders,
   nr_of_cancelled_orders,
   cast(nr_of_cancelled_orders as decimal(10,2)) / cast(total_orders as decimal(10,2)) *
100 as cancel_percent
from cancelled_rate_2024
)
select
   cy.restaurant_id,
   round(ly.cancel_percent,2) || '%' as cancellation_percent_of_2023,
   round(cy.cancel_percent,2) || '%' as cancellation_percent_of_2024
from current_year_data cy
join last_year_data ly
on cy.restaurant_id = ly.restaurant_id
```

```
order by 1;

-- Q10 Rider Average Delivery Time: Determine each rider's average delivery time
select
   r.rider_name,
   round(avg(extract(epoch from d.delivery_time) / 60), 2) as avg_delivery_time_minutes
from deliveries d
join riders r
   on d.rider_id = r.rider_id
where d.delivery_status = 'Delivered'
group by 1;

-- Q11 Monthly Restaurant Growth Ratio:
-- Calculate each restaurant's growth ratio based on the total number of delivered orders
since its joining
with growth_rate_of_delivered_orders as
(
select
   o.restaurant_id,
   extract(year from o.order_date) as order_year,
   extract(month from o.order_date) as order_month,
   to_char(o.order_date, 'Mon yyyy') as month_year,
   cast(count(d.delivery_id) as decimal(10,2)) as current_month_orders_delivered,
   cast(lag(count(d.delivery_id)) over(partition by o.restaurant_id order by extract(year from
o.order_date), extract(month from o.order_date)) as decimal(10,2)) as
prev_month_orders_delivered
from orders o
left join deliveries d
on o.order_id = d.order_id
where d.delivery_status = 'Delivered'
group by 1,2,3,4
)
select
   restaurant_id,
   month_year,
   current_month_orders_delivered,
   prev_month_orders_delivered,
   round((current_month_orders_delivered - prev_month_orders_delivered) /
prev_month_orders_delivered * 100, 2) as growth_rate_in_orders_delivered
from growth_rate_of_delivered_orders
order by
   restaurant_id,
   order_year,
   order_month;
```

```
--Q12 Customer Segmentations:
-- (1) Segment Customers into "Gold" or "Silver" groups based on their total spending
-- (2) Compare to the Average Order Value
-- If Customer's total spending  exceeds AOV Label them with gold other wise label them as
silver
-- Write a Query to Determine each segment's total number of orders and total revenue
select
   customer_category,
   sum(total_spend) as total_revenue,
   sum(nr_of_orders) as total_orders
from
(
select
   c.customer_name,
   count(o.order_id) as nr_of_orders,
   sum(o.total_amount) as total_spend,
   case when sum(o.total_amount) > (select avg(total_amount) from orders) then 'Gold' else
'Silver' end as customer_category
from orders o
join customers c
on c.customer_id = o.customer_id
group by c.customer_id,c.customer_name
) sub
group by 1;

-- Q13 Rider Monthly Earning:
-- Calculate each rider's total monthly earnings, assuming they earn 8% of the Delivered
Order Amount
with riders_monthly_earning as
(
select
   rd.rider_id,
   rd.rider_name,
   extract(year from o.order_date) as order_year,
   extract(month from o.order_date) as order_month,
   to_char(o.order_date, 'Mon yyyy') as month_year,
   cast(sum(total_amount) * 0.08 as decimal(10,2)) as total_earning_of_rider
from orders o
left join deliveries d on d.order_id = o.order_id
left join riders rd
on rd.rider_id = d.rider_id
where d.delivery_status = 'Delivered'
group by 1,2,3,4,5
)
select rider_id, rider_name, month_year, total_earning_of_rider
from riders_monthly_earning;

--Q 14 Rider Rating Analysis:
-- Find the number of 5 Star. 4 star, and 3 star rating Each riders has.
-- Riders recieve this rating based on delivery time
-- IF orders are delivered less than 15 Minutes of order recieved time the rider get 5 star
rating.
-- IF they delivery is 15 to 20 Minute then they get a 4 star rating
-- IF they deliver after 20 Minute they get 3 star rating.
with cte as
(
select
   d.rider_id,
   o.order_time,
   d.delivery_time,
case
   when d.delivery_time < o.order_time
      then (1440 - abs(extract(epoch from (d.delivery_time - o.order_time)) / 60.0))
   else abs(extract(epoch from (d.delivery_time - o.order_time)) / 60.0)
end
 as time_taken_to_deliver
   from orders o
   join deliveries d
      on o.order_id = d.order_id
   where d.delivery_status = 'Delivered'
),
final as
(
select
   rider_id,
   case
      when time_taken_to_deliver <= 15 then '5 star'
      when time_taken_to_deliver > 15 and time_taken_to_deliver <= 20 then '4 star'
      else '3 star'
      end as stars,
   time_taken_to_deliver
from cte
)
select
   rider_id,
   stars,
   count(stars) as total_stars
```

```
from final
group by 1,2
order by 1,3 desc;

-- Q 15 Order Frequency by Day: Analyze order fequency per day of the week and identify
the peak day for each restaurant
with peak_day_for_restaurant as
(
select
   o.restaurant_id,
   r.restaurant_name as restaurant,
   extract(dow from o.order_date) as week_number,
   to_char(o.order_date, 'Day') as weekday_name,
   count(o.order_id) as nr_of_orders,
   dense_rank() over(partition by o.restaurant_id order by count(o.order_id) desc) as
rank_of_week_day
from orders o
left join restaurants r
on r.restaurant_id = o.restaurant_id
group by 1,2,3,4
)
select restaurant, weekday_name, nr_of_orders
from peak_day_for_restaurant
where rank_of_week_day = 1
order by 3 desc;

-- Q16 Customer Lifetime value(CLV): Calculate the Total Revenue Generated by each
customer over all  their orders
select
   c.customer_id,
   c.customer_name,
   sum(o.total_amount) as customer_lifetime_value
from orders o
join customers c
   on c.customer_id = o.customer_id
group by 1,2
order by 3 desc;

-- Q 17 Monthly Sales Trends: Identify Sales Trends by Comparing each month's total Sales
to the previous months
with monthly_sales_trends as
(
select
   extract(year from order_date) as year,
   extract(month from order_date) as month_number,
   to_char(order_date, 'Month') as month_name,
   sum(total_amount) as current_month_sales,
   lag(sum(total_amount)) over (order by extract(year from order_date), extract(month from
order_date)) as prev_month_sales
from orders
group by 1,2,3
)
select
   year,
   month_name,
   prev_month_sales,
   current_month_sales,
   round((current_month_sales - prev_month_sales) / prev_month_sales * 100, 2) || '%' as
percent_growth_in_sales
from monthly_sales_trends
order by 1;

-- Q 18 Rider Effeciency
-- Evaluate rider Effeciency by determining Average Delivery times and Identifying those
with lowest
-- and highest Average Delivery time
select
   min(avg_time_taken_to_deliver) as min_avg_time_taken_to_deliver,
   max(avg_time_taken_to_deliver) as max_avg_time_taken_to_deliver
from (
   select rider_id, rider_name,
      round(avg(time_taken_to_deliver), 2) as avg_time_taken_to_deliver
   from (
      select
         r.rider_id,
         r.rider_name,
         abs(extract(epoch from (d.delivery_time - o.order_time)) / 60.0) as
time_taken_to_deliver
      from orders o
      left join deliveries d
         on o.order_id = d.order_id
      left join riders r
         on r.rider_id = d.rider_id
      where d.delivery_status = 'Delivered'
      ) sub1
   group by 1,2
) sub2;
```

```
-- Q19 Order Item Popularity :Track the Popularity of specific order items over time and
identify seasonal demand spike
select order_item, season, count(*) as nr_of_orders
from
(
select
   order_item,
   case
      when extract(month from order_date) between 3 and 5 then 'spring'
      when extract(month from order_date) between 6 and 8 then 'summer'
      when extract(month from order_date) between 9 and 11 then 'autumn'
      else 'winter'
   end as season
from orders
) sub
group by order_item, season
order by 1, 3 desc;

-- Q 20 Rank each City based on the Total revenue for last year 2023
select
   r.city,
   sum(o.total_amount) as total_revenue,
   rank() over(order by sum(o.total_amount) desc) as rank_of_city_by_revenue
from orders o
left join restaurants r
on r.restaurant_id = o.restaurant_id
where extract(year from order_date) = '2023'
group by 1;
```