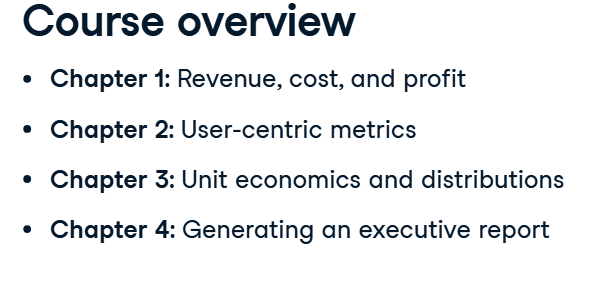
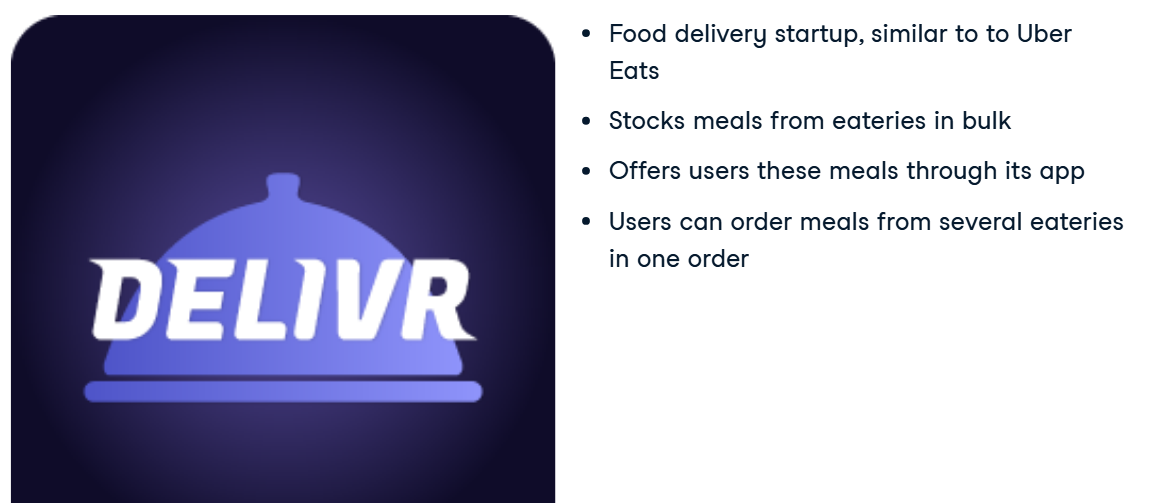
**ANALYZING BUSINESS DATA IN SQL**

# Course Overview

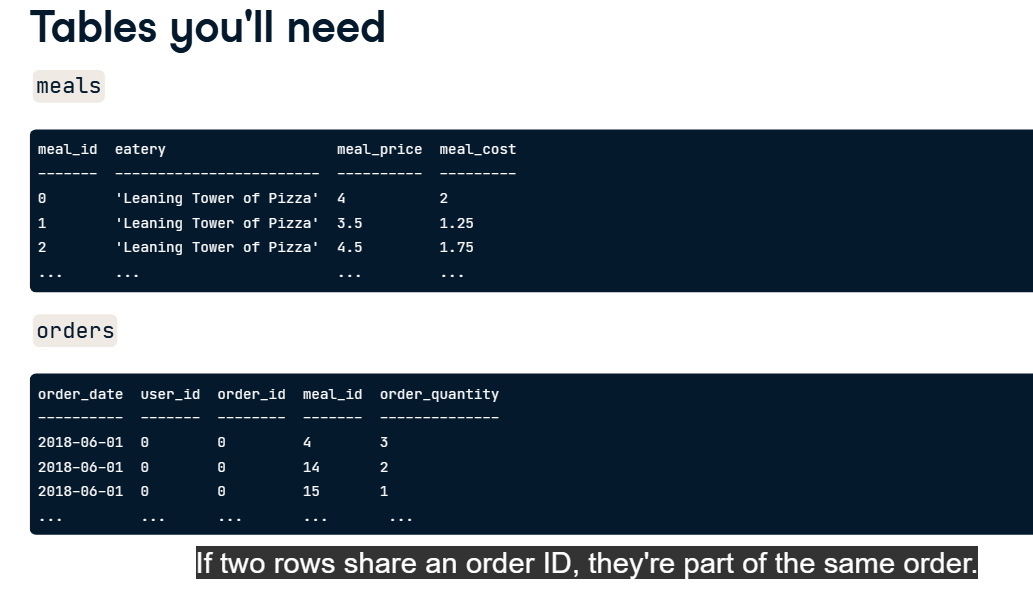




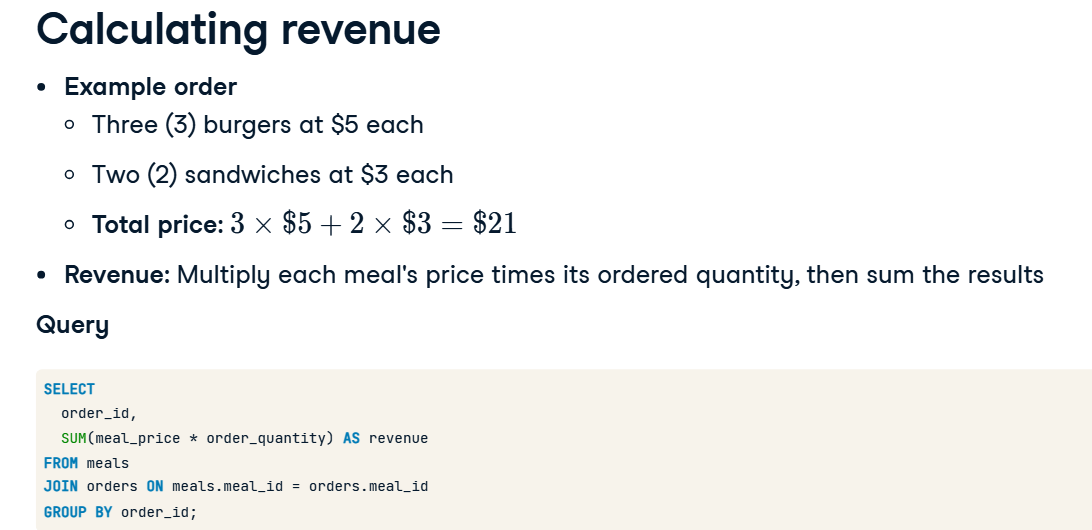
# Revenue, Cost and Profit



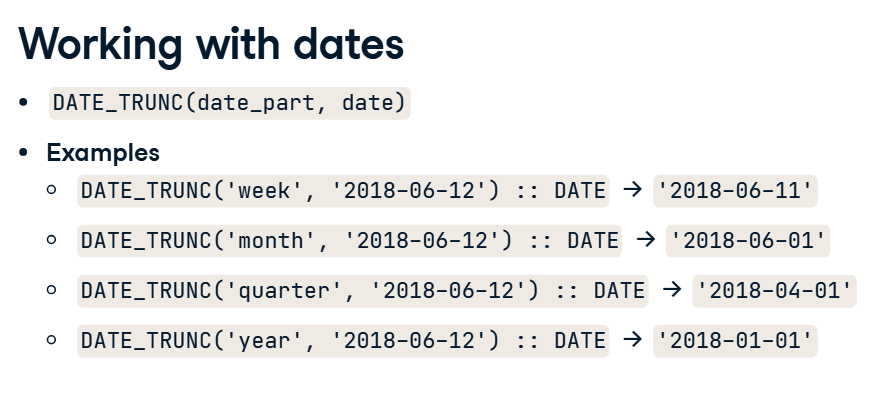
## Tables for Revenue



## Calculating Revenue



## Working with dates



## Calculate weekly revenue for June 2018

SELECT DATE\_TRUNC('week', order\_date) :: DATE AS delivr\_week,

       -- Calculate revenue

       SUM(meal\_price \* order\_quantity) AS revenue

  FROM meals

  JOIN orders ON meals.meal\_id = orders.meal\_id

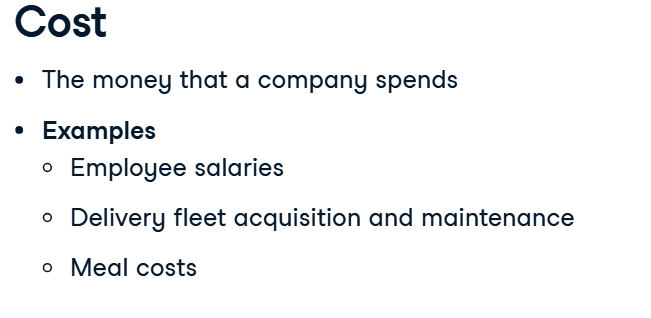
-- Keep only the records in June 2018

WHERE DATE\_TRUNC ('month', order\_date) = '2018-06-01'

GROUP BY delivr\_week

ORDER BY delivr\_week ASC;

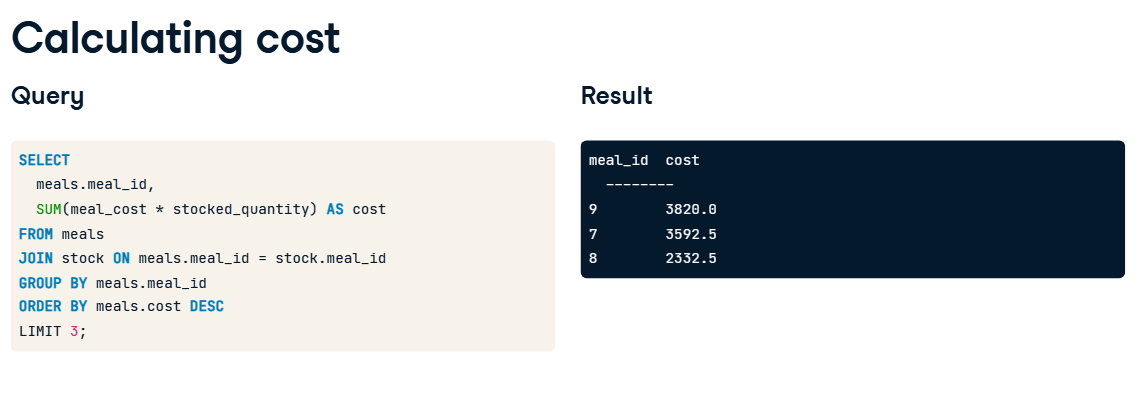
## Cost



## Tables for Cost



## Calculating Cost



## Combine Revenue and Cost - Common Table Expressions (CTEs)



## Query to calculate cost per month

SELECT

  -- Calculate cost

  DATE\_TRUNC('month', stocking\_date)::DATE AS delivr\_month,

  SUM (meal\_cost \* stocked\_quantity) AS cost

FROM meals

JOIN stock ON meals.meal\_id = stock.meal\_id

GROUP BY delivr\_month

ORDER BY delivr\_month ASC;

## How much Delivr spent per month on average during its early months (before September 2018)

-- Declare a CTE named monthly\_cost

WITH monthly\_cost AS (

  SELECT

    DATE\_TRUNC('month', stocking\_date)::DATE AS delivr\_month,

    SUM(meal\_cost \* stocked\_quantity) AS cost

  FROM meals

  JOIN stock ON meals.meal\_id = stock.meal\_id

  GROUP BY delivr\_month)

SELECT

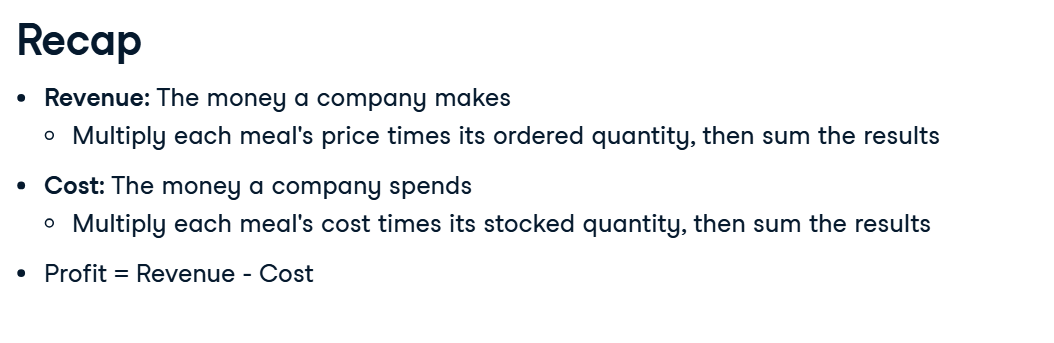
  -- Calculate the average monthly cost before September

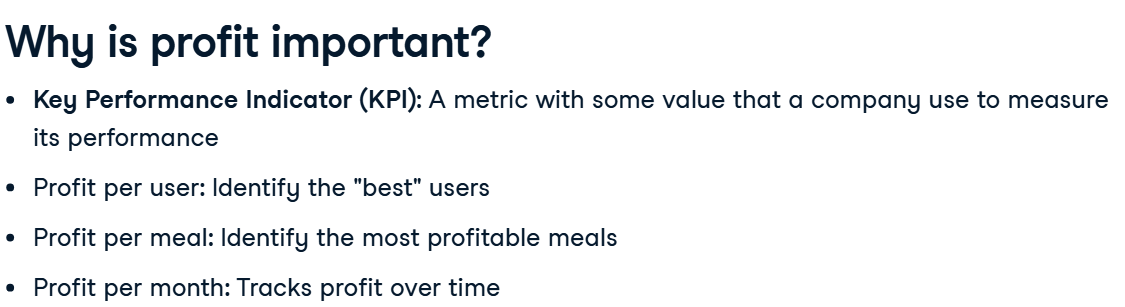
  AVG(cost)

FROM monthly\_cost

WHERE delivr\_month < '2018-09-01';

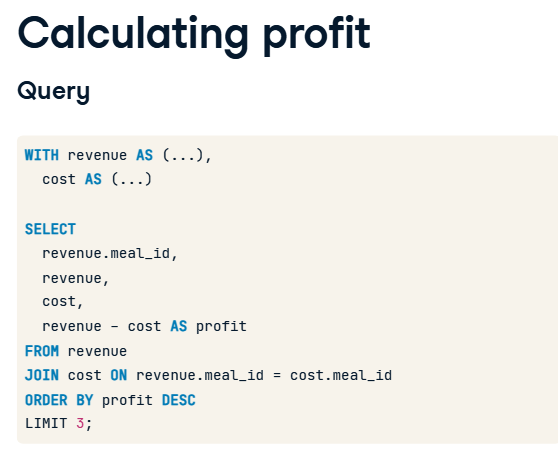
## Profit







**If you're calculating profit per some other metric, like eatery or month, make sure to group by that metric in both CTEs, since you'll join the two CTEs on that metric's column later on.**

****

## How much profit each eatery is generating

WITH revenue AS (

  -- Calculate revenue per eatery

  SELECT eatery,

         SUM(meal\_price \* order\_quantity) AS revenue

    FROM meals

    JOIN orders ON meals.meal\_id = orders.meal\_id

   GROUP BY eatery),

  cost AS (

  -- Calculate cost per eatery

  SELECT eatery,

         SUM(meal\_cost \* stocked\_quantity) AS cost

    FROM meals

    JOIN stock ON meals.meal\_id = stock.meal\_id

   GROUP BY eatery)

   -- Calculate profit per eatery

   SELECT revenue.eatery,

          revenue - cost AS profit

     FROM revenue

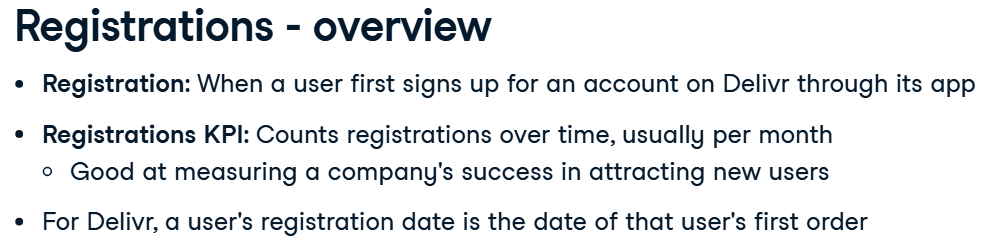
     JOIN cost ON revenue.eatery = cost.eatery

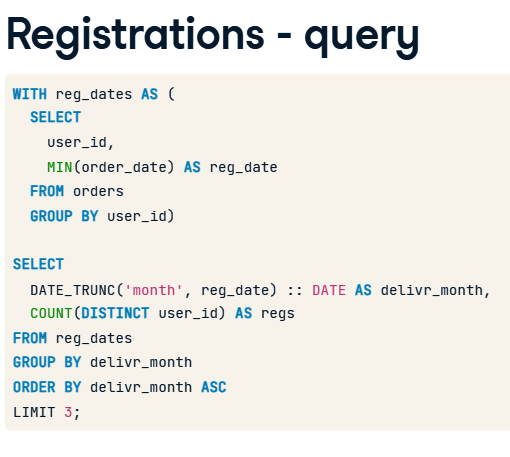
    ORDER BY profit DESC;

# User-centric metrics

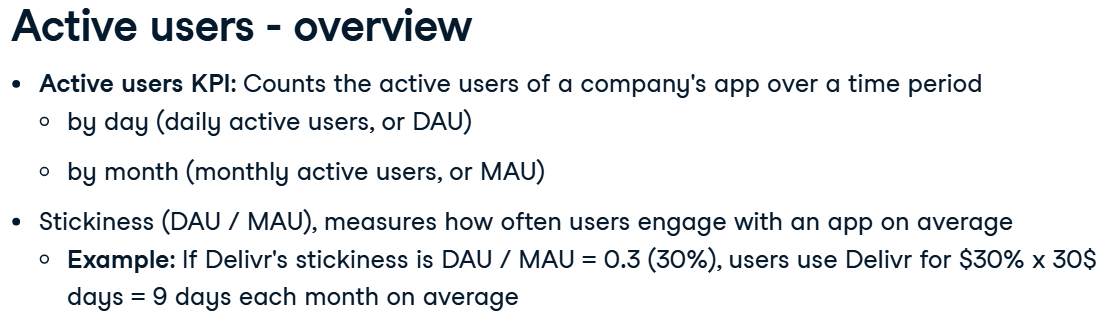


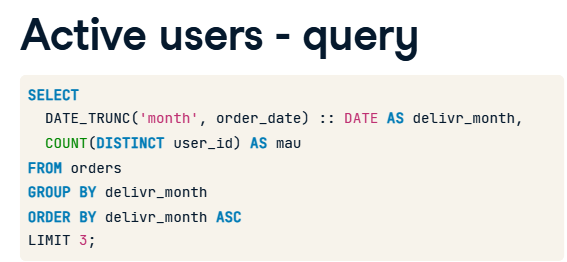
## Registrations



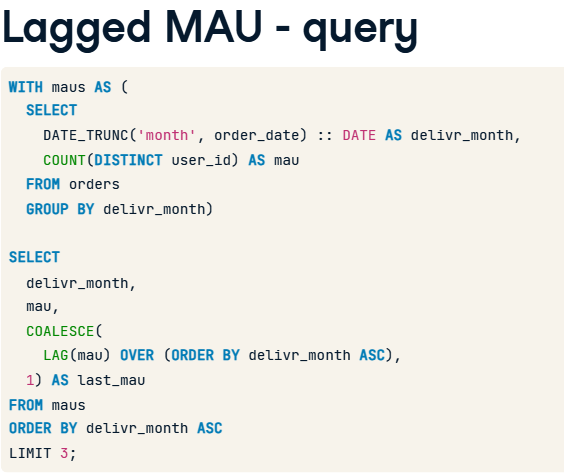


## Active users

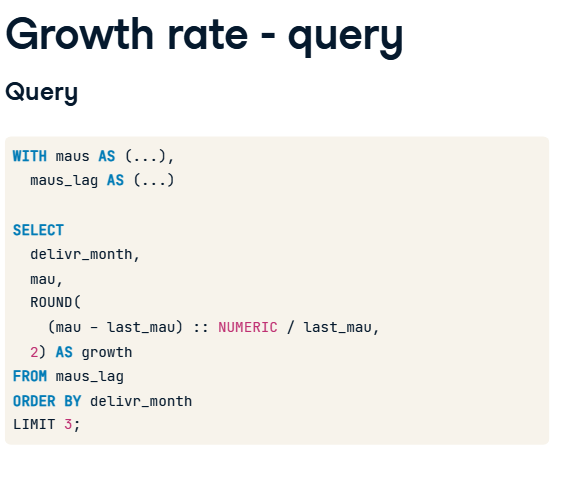
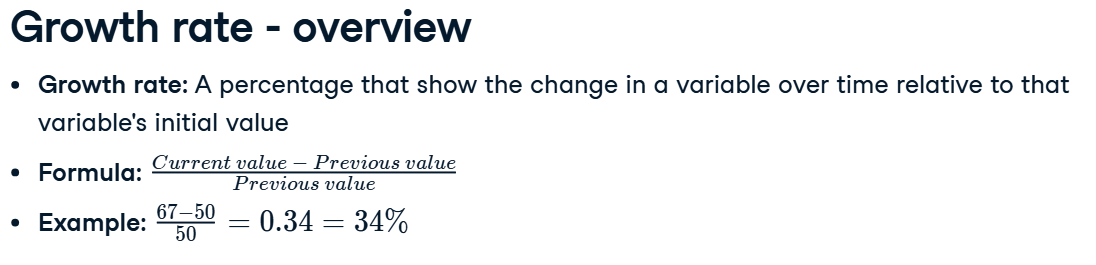
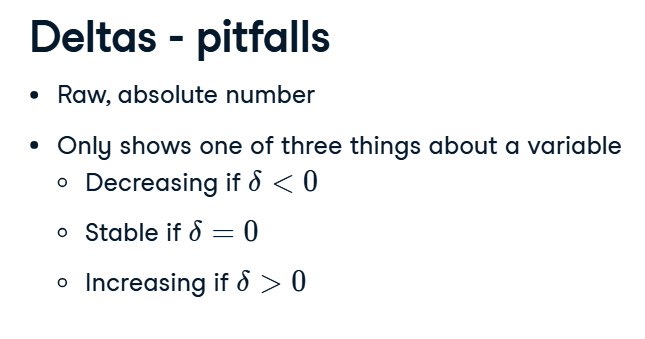




## Window Functions – Running Total and Lag



## Growth Rate and Delta



## Return a table of MoM order growth rates

WITH orders AS (

  SELECT

    DATE\_TRUNC('month', order\_date) :: DATE AS delivr\_month,

    --  Count the unique order IDs

    COUNT(DISTINCT order\_id) AS orders

  FROM orders

  GROUP BY delivr\_month),

  orders\_with\_lag AS (

  SELECT

    delivr\_month,

    -- Fetch each month's current and previous orders

    orders,

    COALESCE(

      LAG(orders) OVER (ORDER BY delivr\_month ASC),

    1) AS last\_orders

  FROM orders)

SELECT

  delivr\_month,

  -- Calculate the MoM order growth rate

  ROUND(

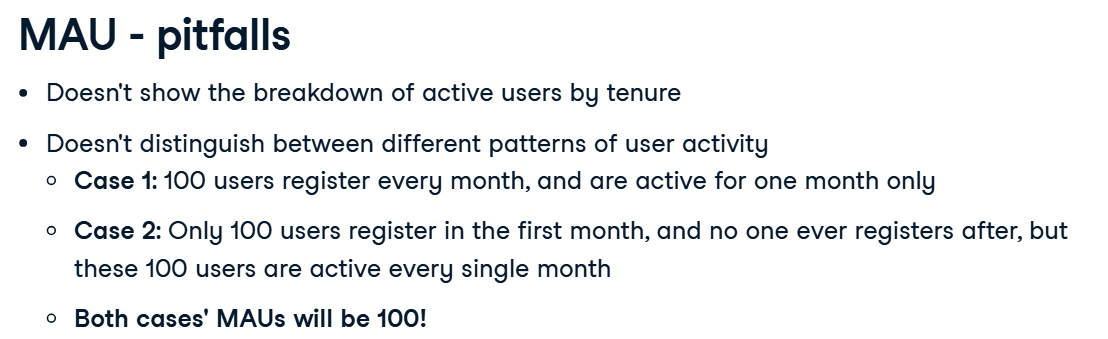
    (orders - last\_orders) :: NUMERIC / last\_orders,

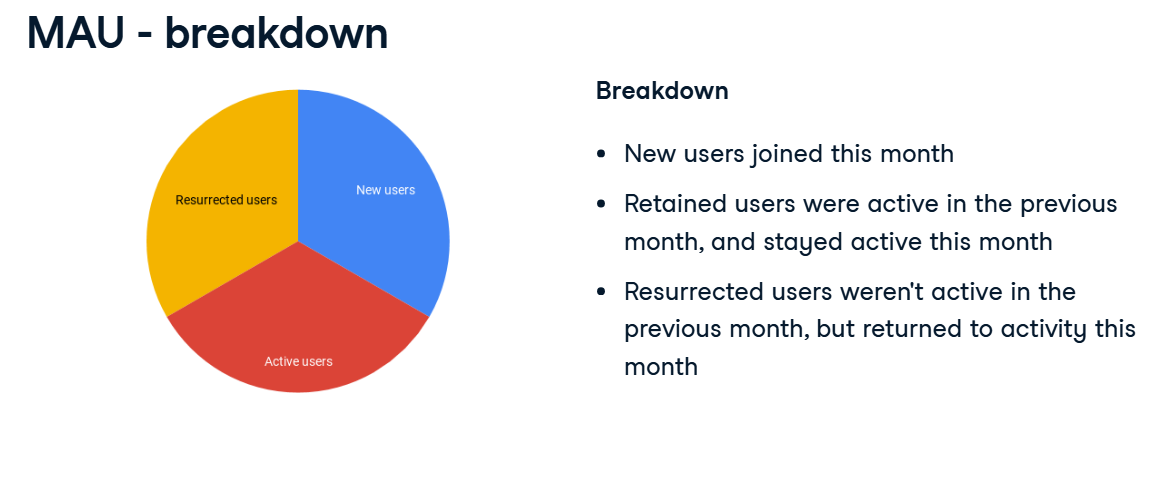
  2) AS growth

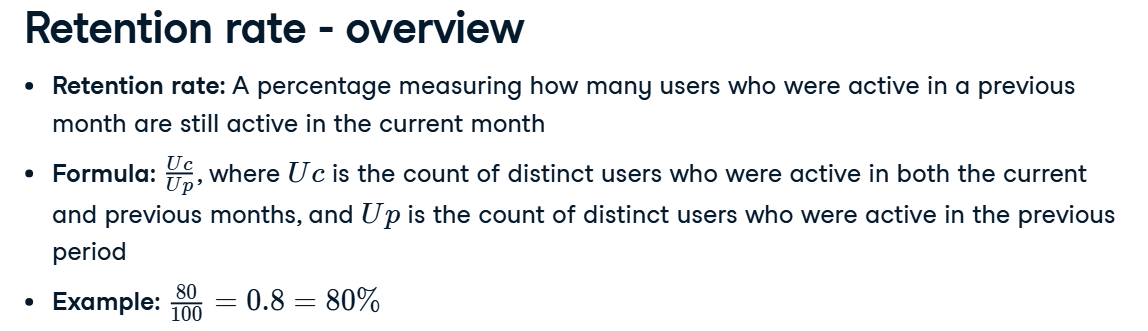
FROM orders\_with\_lag

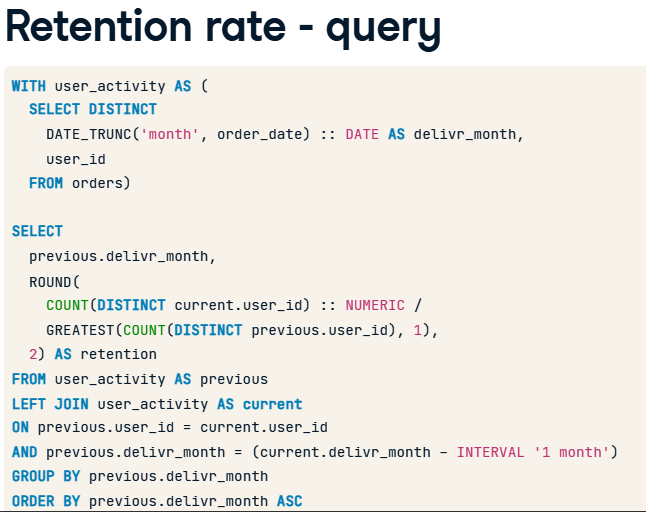
ORDER BY delivr\_month ASC;

## Retention









## Calculate MoM user retention rates

WITH user\_monthly\_activity AS (

  SELECT DISTINCT

    DATE\_TRUNC('month', order\_date) :: DATE AS delivr\_month,

    user\_id

  FROM orders)

SELECT

  -- Calculate the MoM retention rates

  previous.delivr\_month,

  ROUND(

    COUNT (DISTINCT current.user\_id) :: NUMERIC /

    GREATEST (COUNT (DISTINCT( previous.user\_id), 1),----returns 1 if divided by 0

  2) AS retention\_rate

FROM user\_monthly\_activity AS previous

LEFT JOIN user\_monthly\_activity AS current

-- Fill in the user and month join conditions

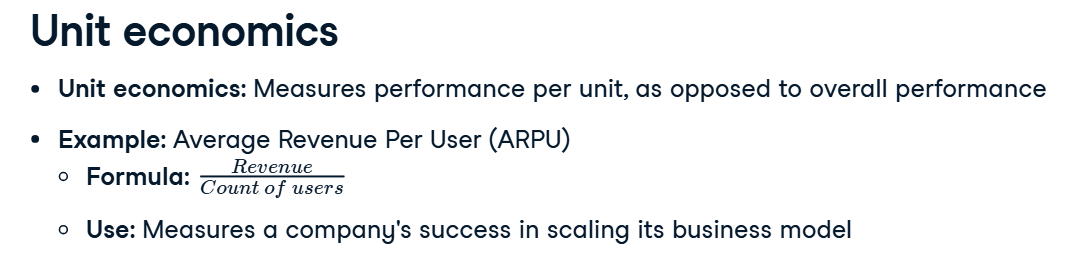
ON previous.user\_id = current.user\_id

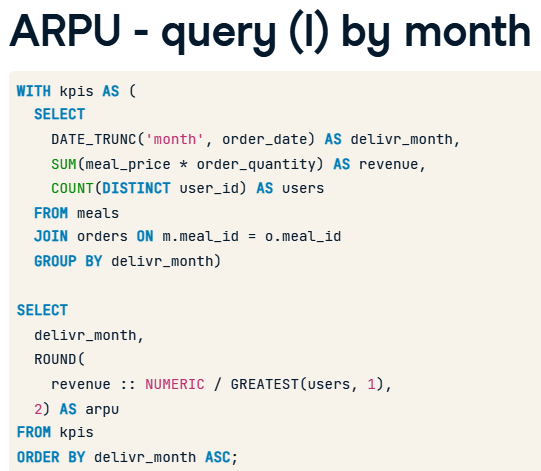
AND previous.delivr\_month = (current.delivr\_month - INTERVAL '1 month')

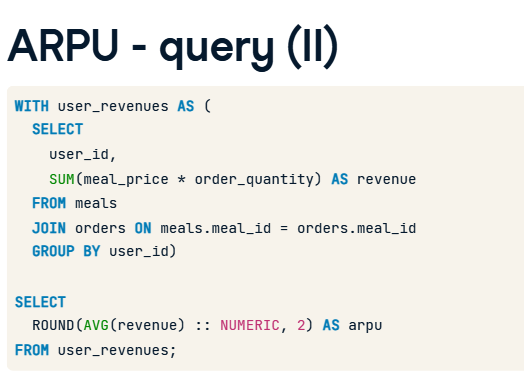
GROUP BY previous.delivr\_month

ORDER BY previous.delivr\_month ASC;

# Unit economics and distributions







## Calculate weekly ARPU

WITH kpi AS (

  SELECT

    -- Select the week, revenue, and count of users

    DATE\_TRUNC ('week', order\_date) :: DATE AS delivr\_week,

    SUM(meal\_price \* order\_quantity) AS revenue,

    COUNT (DISTINCT user\_id) AS users

  FROM meals AS m

  JOIN orders AS o ON m.meal\_id = o.meal\_id

  GROUP BY delivr\_week)

SELECT

  delivr\_week,

  -- Calculate ARPU

  ROUND(

    revenue :: NUMERIC / GREATEST (users, 1),

  2) AS arpu

FROM kpi

-- Order by week in ascending order

ORDER BY delivr\_week ASC;

## Calculate average orders per user

WITH kpi AS (

  SELECT

    -- Select the count of orders and users

    COUNT(DISTINCT order\_id) AS orders,

    COUNT(DISTINCT user\_id) AS users

  FROM orders)

SELECT

  -- Calculate the average orders per user

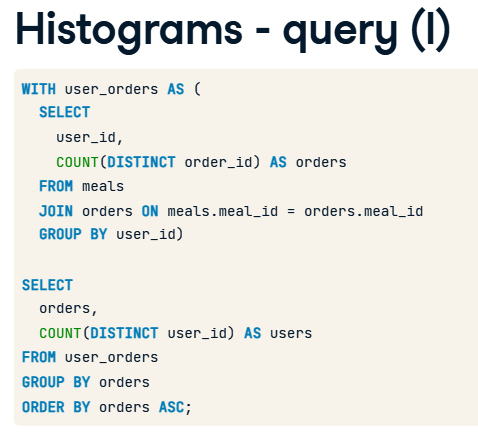
  ROUND(

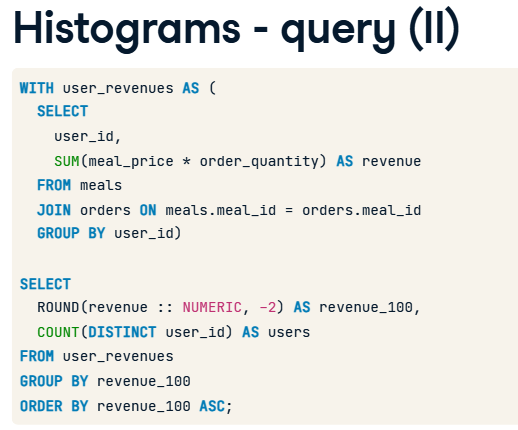
    orders :: NUMERIC / GREATEST(users, 1),

  2) AS arpu

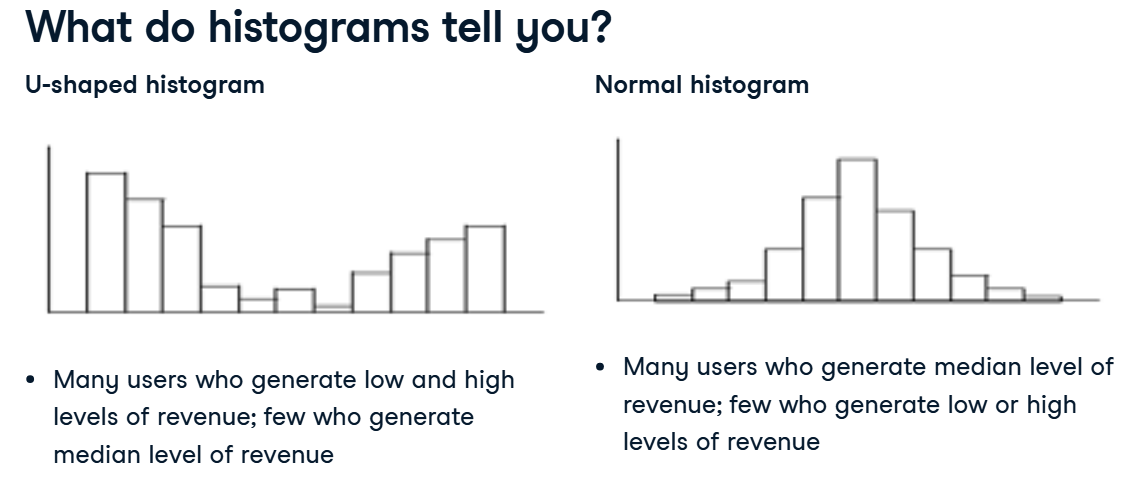
FROM kpi;

## Histograms

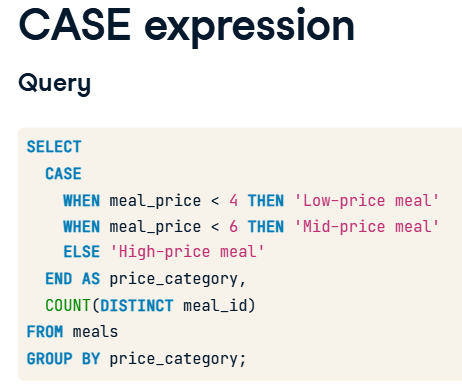


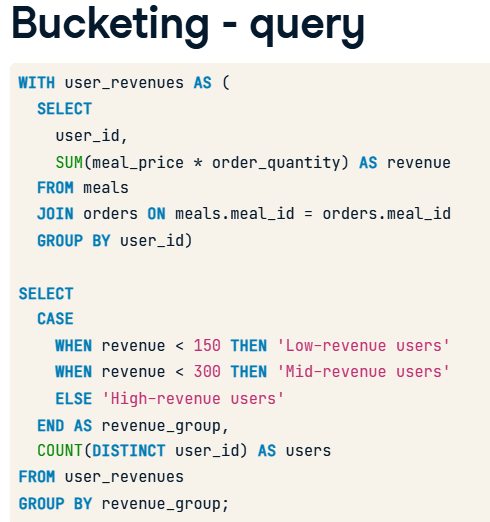


**Notice revenue is rounded to negative -2. When negative value is passed to ROUND as second argument, it rounds to the nearest 10 ^ power of nearest absolute value. E.g. passing -2 to ROUND rounds to the nearest hundred. Revenues are usually decimal values, so it’s very unlikely that two or more users generated the exact same revenue. This clutters the histogram. That's why revenue is rounded here.**

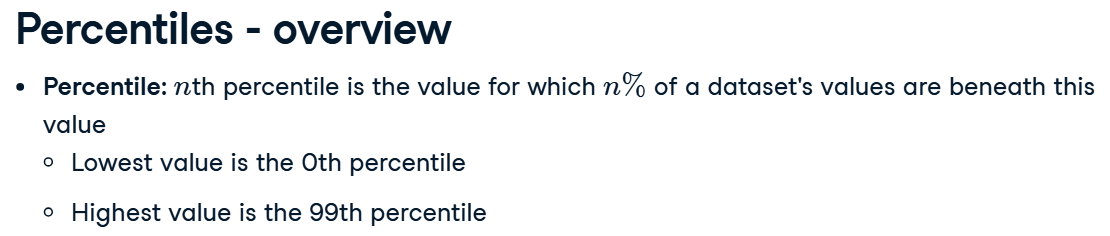
****

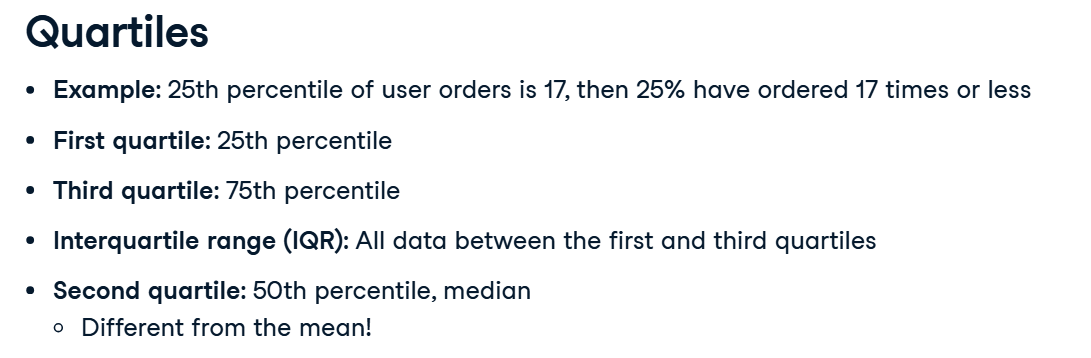
## Bucketing – CASE expression

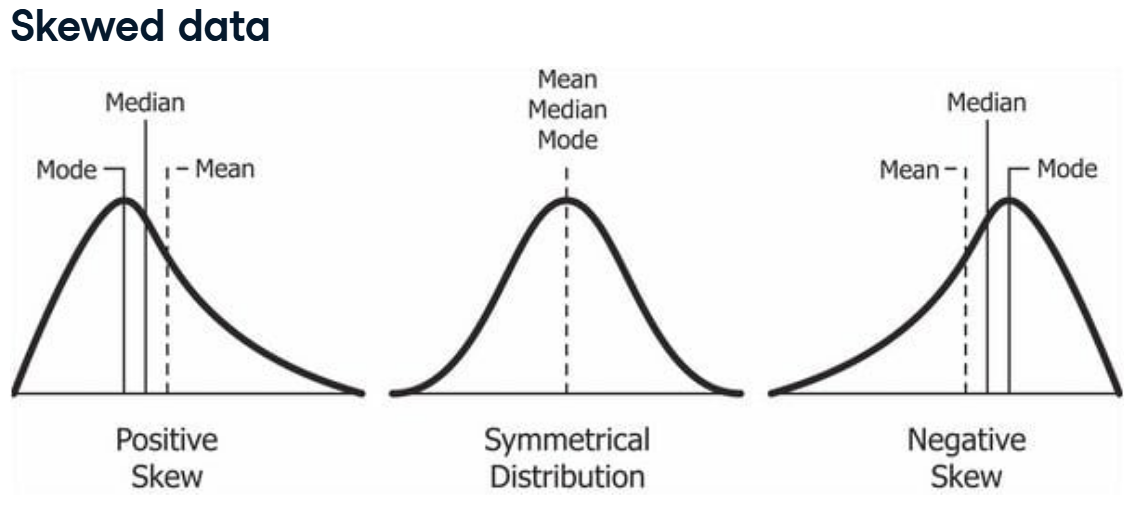


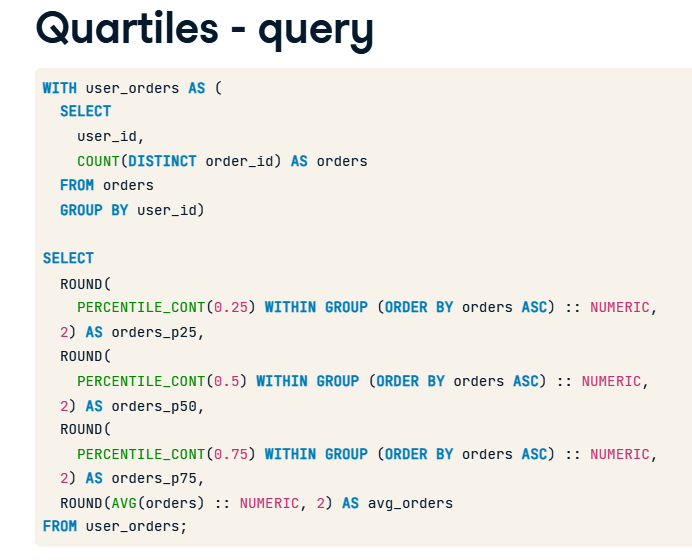


## Percentiles









## Count of users in the revenue interquartile range (IQR)

WITH user\_revenues AS (

  SELECT

    -- Select user\_id and calculate revenue by user

    user\_id,

    SUM(m.meal\_price \* o.order\_quantity) AS revenue

  FROM meals AS m

  JOIN orders AS o ON m.meal\_id = o.meal\_id

  GROUP BY user\_id),

  quartiles AS (

  SELECT

    -- Calculate the first and third revenue quartiles

    ROUND(

      PERCENTILE\_CONT(0.25) WITHIN GROUP

      (ORDER BY revenue ASC) :: NUMERIC,

    2) AS revenue\_p25,

    ROUND(

      PERCENTILE\_CONT(0.75) WITHIN GROUP

      (ORDER BY revenue ASC) :: NUMERIC,

    2) AS revenue\_p75

  FROM user\_revenues)

SELECT

  -- Count the number of users in the IQR

  COUNT(DISTINCT user\_id) AS users

FROM user\_revenues

CROSS JOIN quartiles

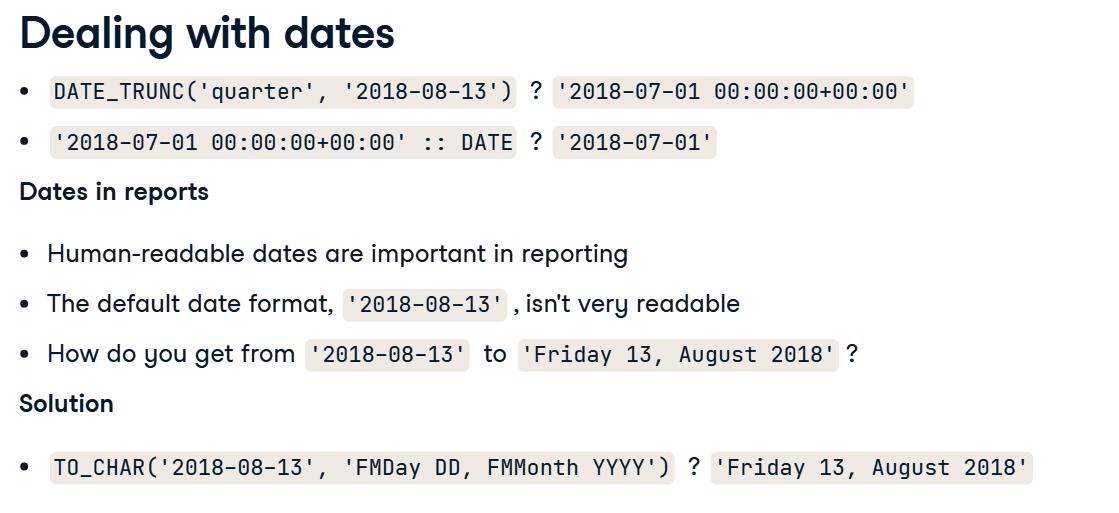
-- Only keep users with revenues in the IQR range

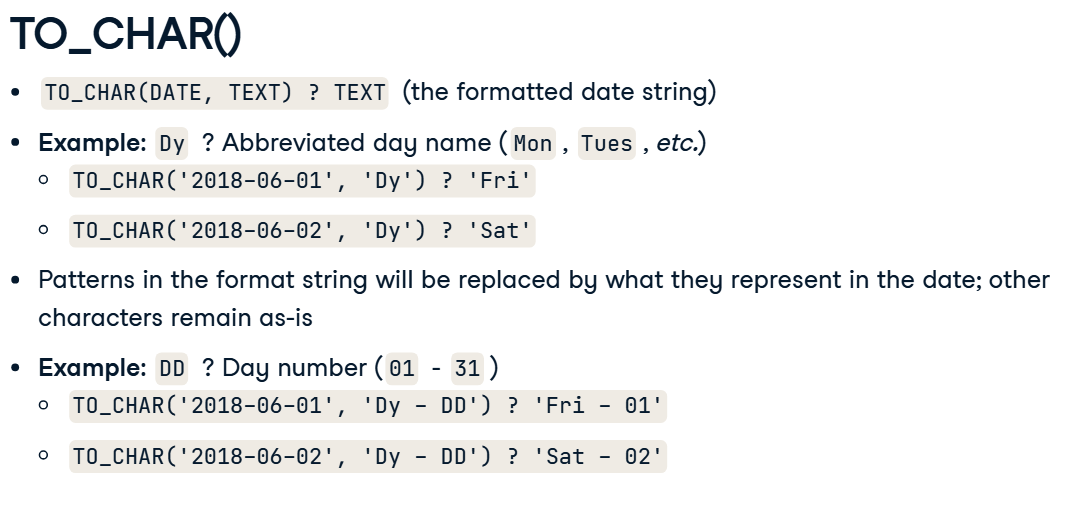
WHERE revenue :: NUMERIC >= revenue\_p25

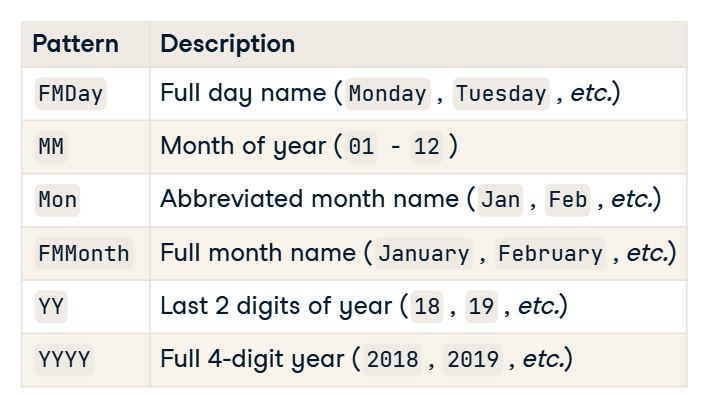
  AND revenue :: NUMERIC <= revenue\_p75;

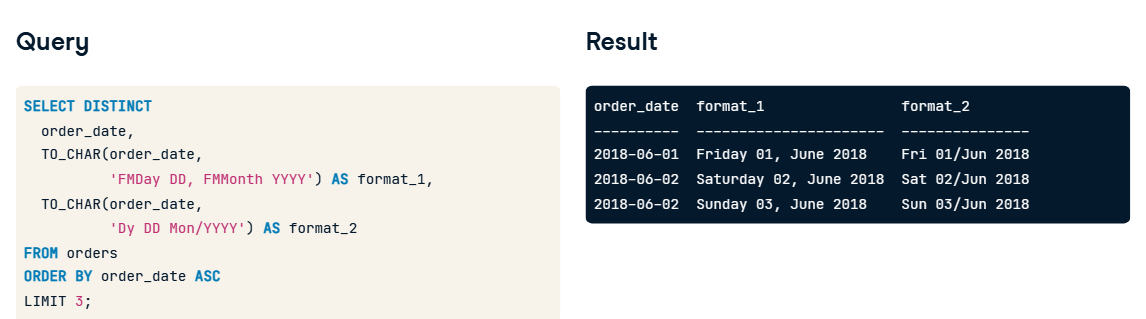
# Useful Functions

## Dates

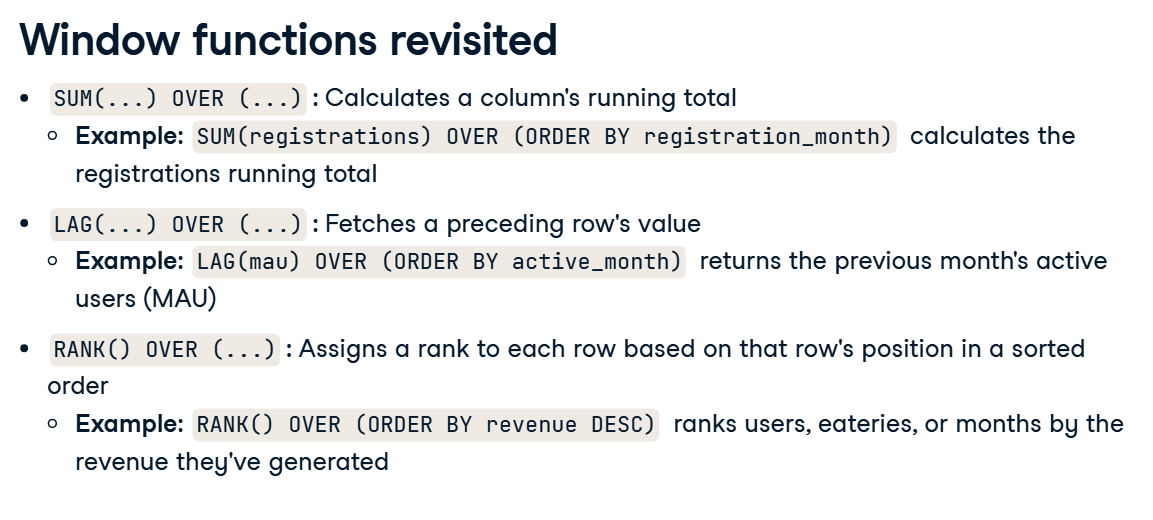








## Window Functions





## Find top 3 users by order count who ordered the most in Augst 2018

-- Set up the user\_count\_orders CTE

WITH user\_count\_orders AS (

  SELECT

    user\_id,

    COUNT(DISTINCT order\_id) AS count\_orders

  FROM orders

  -- Only keep orders in August 2018

  WHERE DATE\_TRUNC('month', order\_date) = '2018-08-01'

  GROUP BY user\_id)

SELECT

  -- Select user ID, and rank user ID by count\_orders

  user\_id,

  RANK() OVER (ORDER BY count\_orders DESC) AS count\_orders\_rank

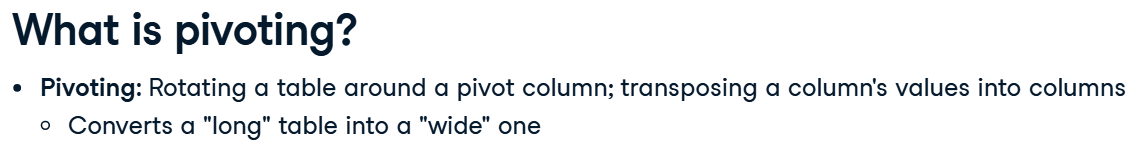
FROM user\_count\_orders

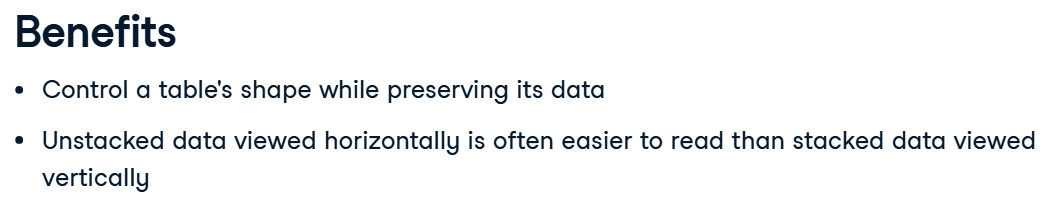
ORDER BY count\_orders\_rank ASC

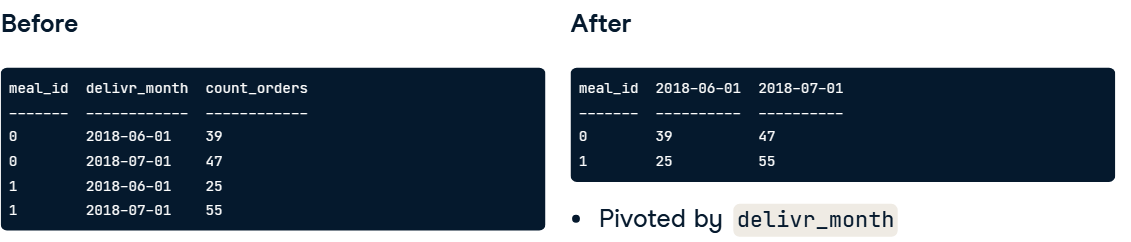
-- Limit the user IDs selected to 3

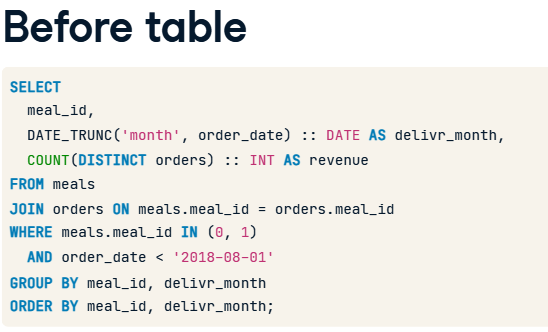
LIMIT 3;

## Pivoting

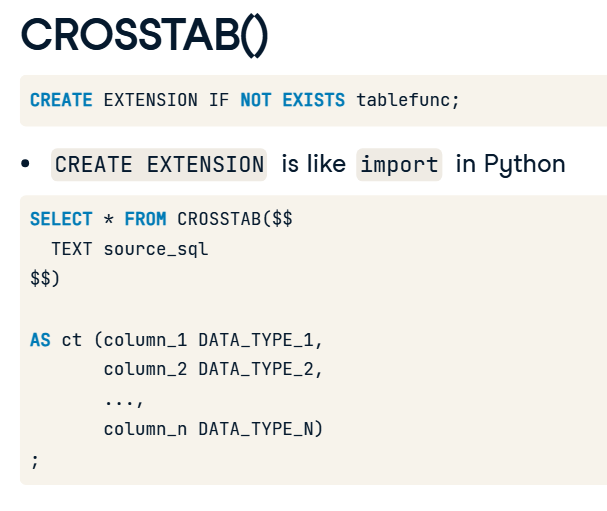


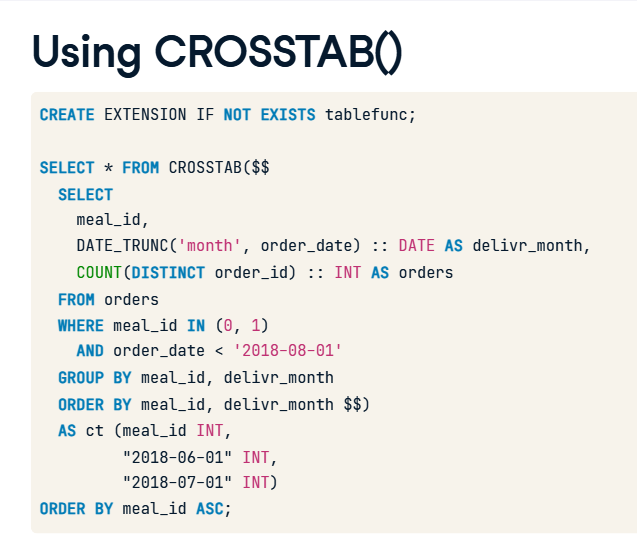


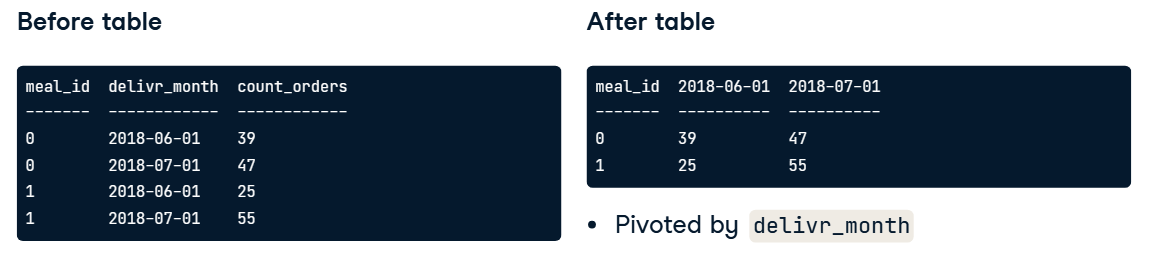




## CrossTab()







## First 5 user IDs' revenues from June to August 2018

-- Import tablefunc

CREATE EXTENSION IF NOT EXISTS tablefunc;

SELECT \* FROM CROSSTAB($$

  SELECT

    user\_id,

    DATE\_TRUNC('month', order\_date) :: DATE AS delivr\_month,

    SUM(meal\_price \* order\_quantity) :: FLOAT AS revenue

  FROM meals

  JOIN orders ON meals.meal\_id = orders.meal\_id

 WHERE user\_id IN (0, 1, 2, 3, 4)

   AND order\_date < '2018-09-01'

 GROUP BY user\_id, delivr\_month

 ORDER BY user\_id, delivr\_month;

$$)

-- Select user ID and the months from June to August 2018

AS ct (user\_id INT,

       "2018-06-01" FLOAT,

       "2018-07-01" FLOAT,

       "2018-08-01" FLOAT)

ORDER BY user\_id ASC;

## First 5 eateries’ cost in November to December 2018

-- Import tablefunc

CREATE EXTENSION IF NOT EXISTS tablefunc;

SELECT \* FROM CROSSTAB($$

  SELECT

    -- Select eatery and calculate total cost

    eatery,

    DATE\_TRUNC('month', stocking\_date) :: DATE AS delivr\_month,

    SUM(meal\_cost \* stocked\_quantity) :: FLOAT AS cost

  FROM meals

  JOIN stock ON meals.meal\_id = stock.meal\_id

  -- Keep only the records after October 2018

  WHERE DATE\_TRUNC('month', stocking\_date) > '2018-10-01'

  GROUP BY eatery, delivr\_month

  ORDER BY eatery, delivr\_month;

$$)

-- Select the eatery and November and December 2018 as columns

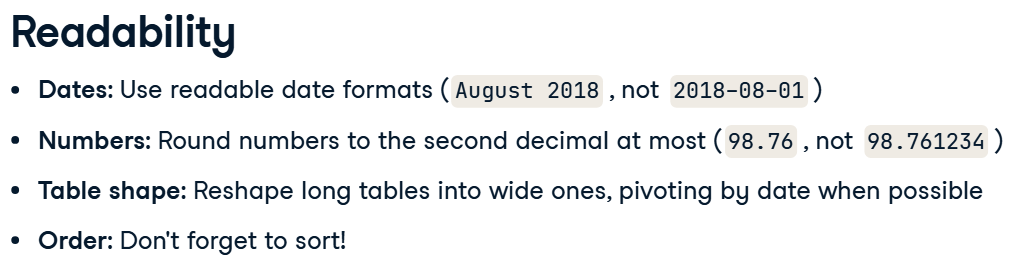
AS ct ("eatery" TEXT,

       "2018-11-01" FLOAT,

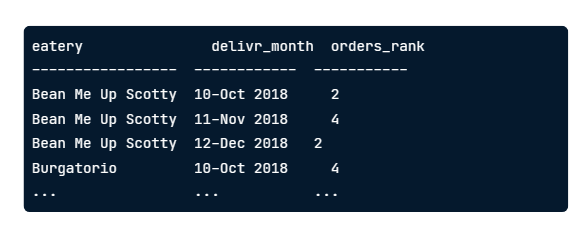
       "2018-12-01" FLOAT)

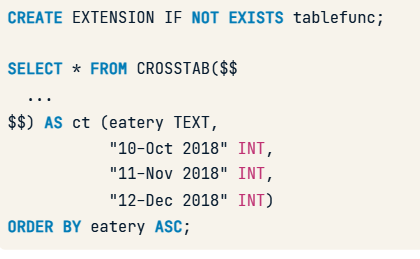
ORDER BY eatery ASC;

# Generating an Executive Report









WITH eatery\_users AS  (

  SELECT

    eatery,

    -- Format the order date so "2018-06-01" becomes "Q2 2018"

    TO\_CHAR(order\_date, '"Q"Q YYYY') AS delivr\_quarter,

    -- Count unique users

    COUNT(DISTINCT user\_id) AS users

  FROM meals

  JOIN orders ON meals.meal\_id = orders.meal\_id

  GROUP BY eatery, delivr\_quarter

  ORDER BY delivr\_quarter, users)

SELECT

  -- Select eatery and quarter

  eatery,

  delivr\_quarter,

  -- Rank rows, partition by quarter and order by users

  RANK() OVER

    (PARTITION BY delivr\_quarter

     ORDER BY users DESC) :: INT AS users\_rank

FROM eatery\_users

ORDER BY delivr\_quarter, users\_rank;