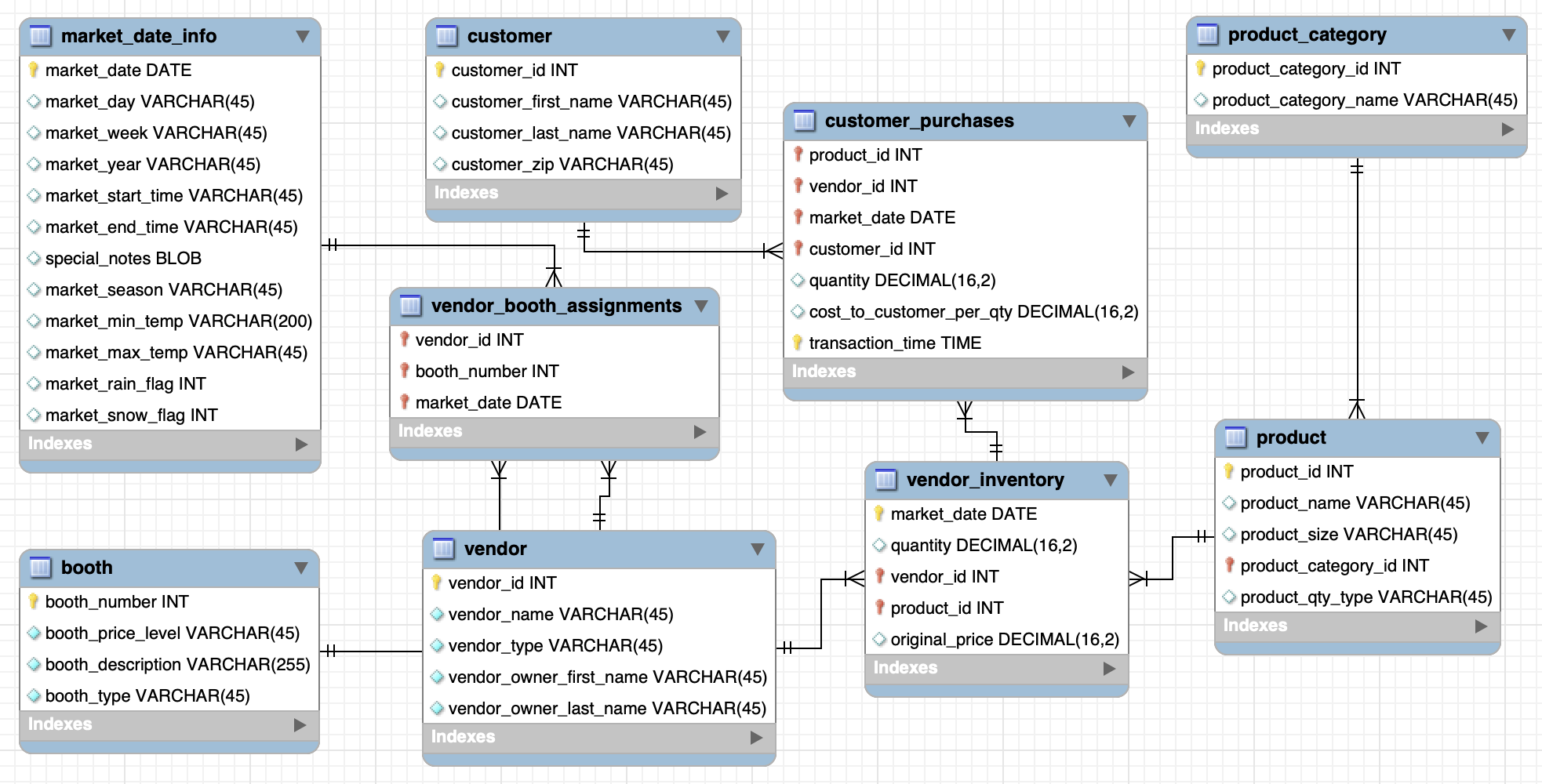
Date & Time Functions

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Problem Statement:

You are a Data Analyst at Amazon Fresh. You have been tasked to study the Farmer’s Market.

Dataset: Farmer’s Market database



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# So far…

* You now know how to aggregate data, group data, use JOINs to get data from multiple tables and how to do windowed calculations

But

* We haven’t dealt with date and time data type. One of the most important and common problems for data scientists is dealing with temporal data.
* Many **ML algorithms are “trained” to identify patterns in data** from the past and use those patterns to predict future outcomes. In order to build a dataset for that purpose, we have to be able to filter queries by time range.
* Often, datasets that are built for predictive models include summaries of activities within dynamic date ranges—for example, **how many times a user ordered an iPhone on Amazon during each of the past three months.**
* Or, in the case of time-series analysis, an input dataset might include one row per time period (hour, day, week, month) with a count of something associated with each time period; for example, the number of patients a doctor sees per week.

In our database, I have created one more table called [**datetime\_demo**](https://drive.google.com/file/d/1vmPuF0K1onScjDOaYfVyj8tAq2KqmGbI/view?usp=sharing)to showcase how you can work with datetime data types.

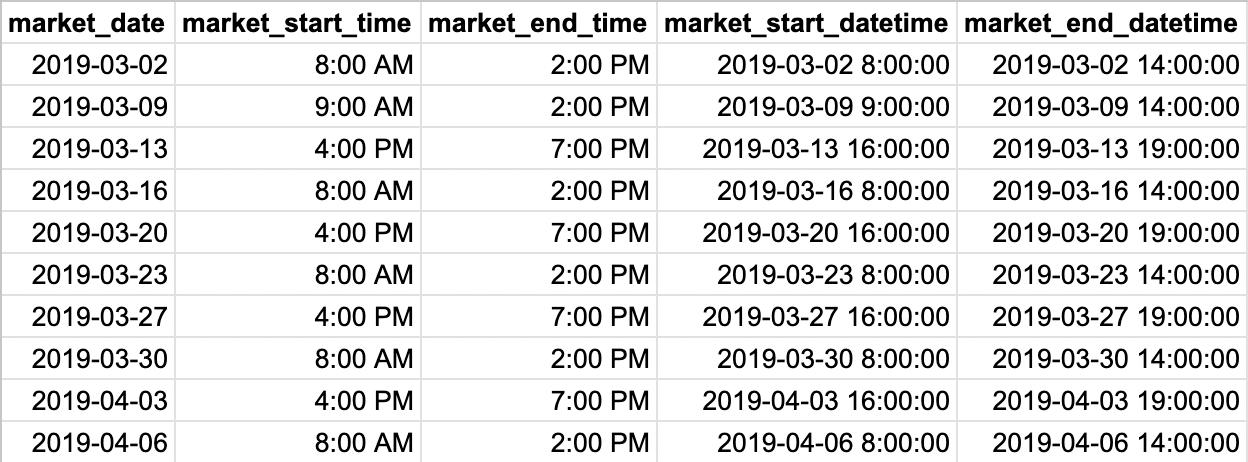
Inside this table,

* We have data from the “**market\_date\_info**” table i.e. market\_date, market\_start\_time, and market\_end\_time columns.

Along with two additional columns **market\_start\_datetime** and **market\_end\_datetime**.

Here’s what the “**datetime\_demo**” table looks like:

SELECT \* FROM farmers\_market.datetime\_demo;

****

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**DateTime format -**

Previously, we saw that

* the market date was in “YYYY-MM-DD”,
* the market start and end time were in “hr:mins AM/PM”, and
* the market start and end datetime in “YYYY-MM-DD HH:MM:SS” format.

But do we always get date and time in the same format?

No, it might differ when working on different data based on how the DateTime format is saved.

Let us have a look at some of the commonly used DateTime formats:

* %Y: Year with century (e.g., 2023)
* %m: Month (01-12)
* %d: Day of the month (01-31)
* %I: Hour (00-12)
* %H: Hour (00-23)
* %M: Minute (00-59)
* %S: Second (00-59)
* %P: AM/PM

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Oftentimes, you will encounter datetime data types, such as timestamps, in the databases you work with and might only need a portion of the stored date and time value.

Let’s answer a few questions to understand:

## **EXTRACT()**

Question: Suppose you wish to know from which year to which year data do we have in our database?

To extract the year, in MySQL, we can use the EXTRACT() function.

**For BigQuery:**

SELECT

MIN(EXTRACT(year

FROM

market\_start\_datetime)) AS min\_year,

max (EXTRACT(year

FROM

market\_start\_datetime)) AS max\_year

FROM

`farmers\_market.datetime\_demo`;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Question: What if you only want to see the hour at which the market started and ended on each date?

**For BigQuery:**

SELECT

EXTRACT(hour

FROM

market\_start\_datetime) AS start\_hr,

EXTRACT(hour

FROM

market\_end\_datetime) AS end\_hr

FROM

`farmers\_market.datetime\_demo`;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Similarly, we can extract the followings from each ‘market\_start\_datetime’ as well -

* Date
* Time
* Year
* Quarter
* Month
* Day
* Week
* Day Of Week
* Hour
* Minute
* Second

Here’s how the syntax goes:

**For BigQuery:**

SELECT market\_start\_datetime,

EXTRACT(date from market\_start\_datetime) as date,

EXTRACT(time from market\_start\_datetime) as time,

EXTRACT(year from market\_start\_datetime) as year\_no,

EXTRACT(quarter from market\_start\_datetime) as q\_no,

EXTRACT(month from market\_start\_datetime) as month\_no,

EXTRACT(day from market\_start\_datetime) as day\_no,

EXTRACT(week from market\_start\_datetime) as week\_no,

EXTRACT(DAYOFWEEK from market\_start\_datetime) as week\_day,

EXTRACT(hour from market\_start\_datetime) as hr,

EXTRACT(minute from market\_start\_datetime) as minute,

EXTRACT(second from market\_start\_datetime) as second,

FROM farmers\_market.datetime\_demo

**Note:** Depending on the database system you are using, the function that retrieves different portions of a datetime value may be called EXTRACT (MySQL), DATE\_ PART (Redshift), or DATEPART (Oracle and SQL Server).

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Question: Your manager asks you that for each market date, he wants to see what day & month it was. Display the name of the day & month.

**For BigQuery:**

SELECT

market\_start\_datetime,

**FORMAT\_DATETIME**("%B", market\_start\_datetime) as mktsrt\_month\_name,

**FORMAT\_DATETIME**("%A", market\_start\_datetime) as   
mktsrt\_day\_name

FROM

`farmers\_market.datetime\_demo`

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## **DATE() AND TIME()**

Question: Suppose you only have the ‘market\_start\_datetime’ column in the table and you want to view the **date** and **time** fields separately.

We also have shortcuts for extracting the entire **date** and entire **time** from the datetime field, so you don’t have to extract each part and re-concatenate it together.

**For BigQuery:**

SELECT

market\_start\_datetime,

**DATE**(market\_start\_datetime) AS mktsrt\_date,

**TIME**(market\_start\_datetime) AS mktsrt\_time

FROM

`farmers\_market.datetime\_demo`

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## **DATE\_ADD and DATE\_SUB**

The powerful thing about storing string dates as datetime values (or converting them using SQL) is that you can do date calculations, which is not possible when they are stored as numbers, punctuation, and letters in a string field.

Here, we’ll use the **market\_start\_datetime** and **market\_end\_datetime** fields to demonstrate.

**The DATE\_ADD() function adds a time/date interval to a date and then returns the date.**

We can use SQL to add 30 minutes to the start time by passing the datetime, the interval (minutes, in this case), and the number of minutes we want to add into the DATE\_ADD function:

SELECT market\_start\_datetime,

DATE\_ADD(market\_start\_datetime, INTERVAL 30 MINUTE) AS mktstrt\_date\_plus\_30min

FROM farmers\_market.datetime\_demo;

Note:

* If we instead wanted to do a calculation that required **looking 30 days past a date** (for example which would require calculating 30 days past a customer’s first purchase to determine if they made a second purchase within that time frame), we could change the interval parameter from MINUTE to DAY, and add 30 days instead:

SELECT market\_start\_datetime,

DATE\_ADD(market\_start\_datetime, INTERVAL 30 DAY) AS mktstrt\_date\_plus\_30days

FROM farmers\_market.datetime\_demo;

**There is also a related function called DATE\_SUB() that subtracts intervals from datetimes.**

However, instead of switching to DATE\_SUB(), you could also just add a **negative number** to the datetime if you prefer.

The following query demonstrates that using DATE\_ADD() to add **–30 days** to a date has the same effect as using DATE\_SUB() to subtract 30 days from a date,

SELECT market\_start\_datetime,

DATE\_ADD(market\_start\_datetime, INTERVAL -30 DAY) AS mktstrt\_date\_

plus\_neg30days,

DATE\_SUB(market\_start\_datetime, INTERVAL 30 DAY) AS mktstrt\_date\_

minus\_30days

FROM farmers\_market.datetime\_demo;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## **DATE\_DIFF()**

Question: Find the number of days between the first and last market dates.

DATE\_DIFF is a SQL function available in BigQuery that accepts two dates or DateTime values along with a **date\_part** and **returns the difference between them** specified date\_part intervals.

**For BigQuery:**

SELECT

x.first\_market,

x.last\_market,

DATE\_DIFF(x.last\_market, x.first\_market, DAY) days\_first\_to\_last

FROM

(

SELECT

min(market\_start\_datetime) first\_market,

max(market\_start\_datetime) last\_market

FROM farmers\_market.datetime\_demo

) x

Here, the inner query (by which I mean the query inside parentheses, aliased “x”) returns the first and last market dates from the **datetime\_demo** table, and the outer query (which is selecting from “x”) calculates the difference between those two dates in days using DATE\_DIFF.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Question: But what if we want the difference between the first and last market dates in hours instead of days?

The **DATE\_DIFF** function is used only for date values, but there is also a function in BigQuery called **TIMESTAMP\_DIFF** that returns the difference between two datetimes in any chosen interval.

Here, we calculate the **hours** and **minutes** between the **market start and end times** on each market date.

**For BigQuery:**

SELECT market\_start\_datetime, market\_end\_datetime,

TIMESTAMP\_DIFF(market\_end\_datetime, market\_start\_datetime ,HOUR)

AS market\_duration\_hours,

TIMESTAMP\_DIFF(market\_end\_datetime,market\_start\_datetime, MINUTE)

AS market\_duration\_mins

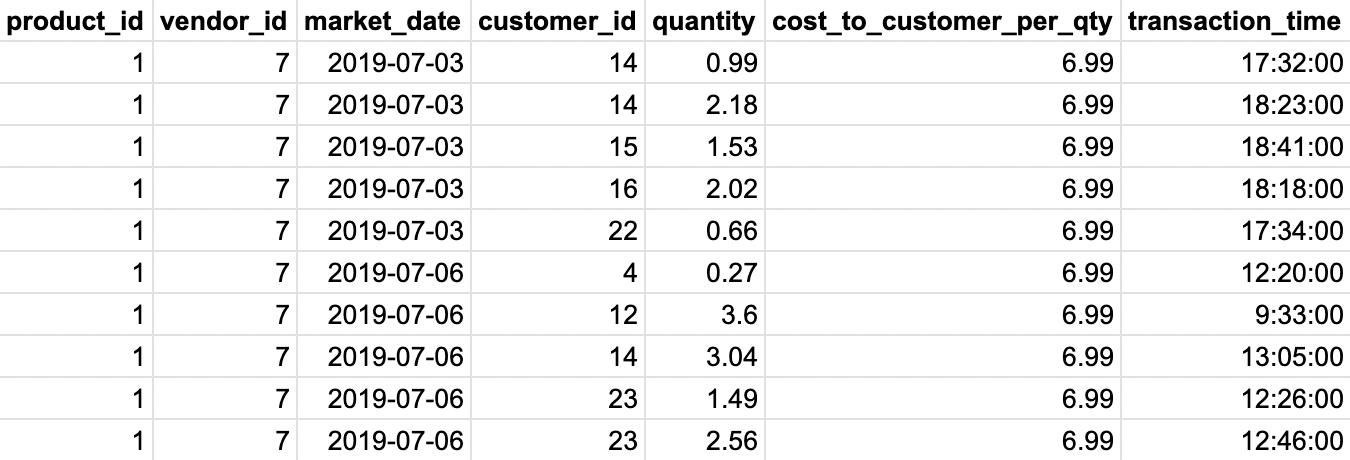
FROM farmers\_market.datetime\_demo;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## **PARSE\_DATETIME()**

Let’s have a look at the **transaction\_time** column in the “**customer\_purchases**” table.

SELECT transaction\_time FROM farmers\_market.customer\_purchases;



**Observation:** Notice that the transaction\_time is **24hr** format compared to **12hr** format in market\_start\_time and market\_end\_time.

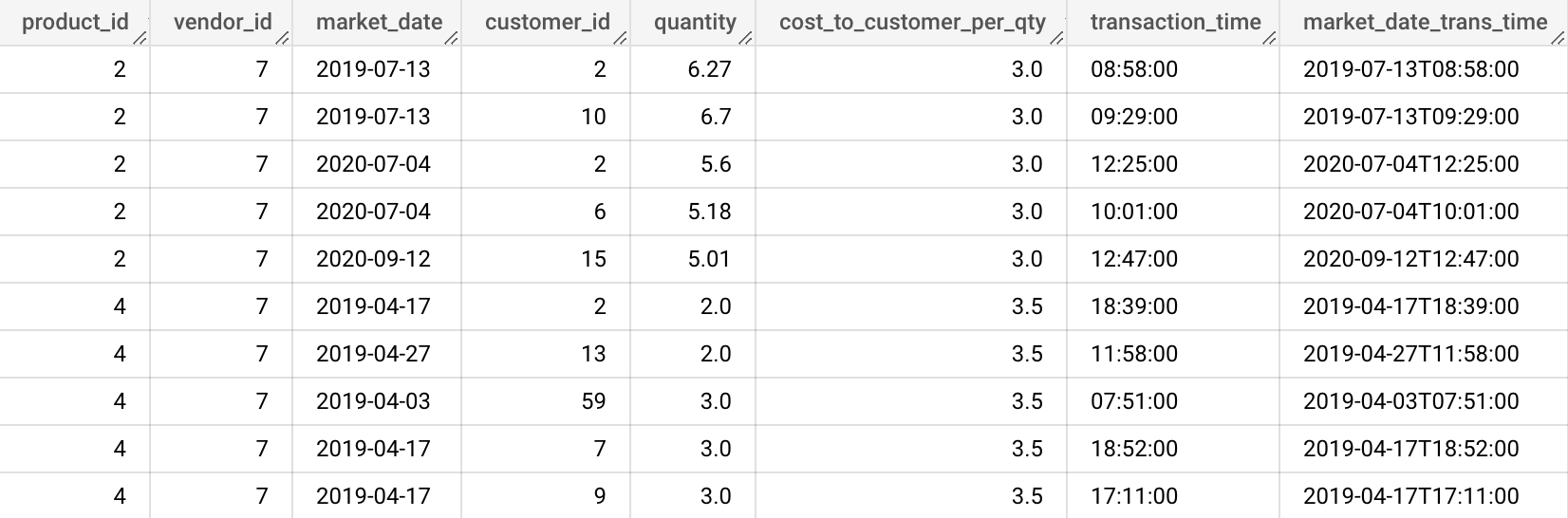
Question: Create a new column **market\_date\_trans\_time** that contains market date and transaction time in a proper **datetime** format.

**For BigQuery:**

SELECT \*,

**PARSE\_DATETIME**("%Y-%m-%d %H:%M:%S", CONCAT(market\_date," ", transaction\_time)) AS market\_date\_trans\_time

FROM farmers\_market.customer\_purchases;



* PARSE\_DATETIME("%Y-%m-%d %H:%M:%S", CONCAT(market\_date," ", transaction\_time)):
  + This function combines the values from the market\_date and transaction\_time columns, concatenating them into a single string with the format "YYYY-MM-DD HH:MM:SS".
  + It then parses this string into a datetime value using the specified format.
* %Y-%m-%d %H:%M:%S is the format used to describe how the datetime string should be interpreted.
  + It breaks down as follows:
    - %Y: Year with century (e.g., 2023)
    - %m: Month (01-12)
    - %d: Day of the month (01-31)
    - %H: Hour (00-23)
    - %M: Minute (00-59)
    - %S: Second (00-59)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Complex Question:** Let’s say you want to calculate how many sales occurred within the first 30 minutes after the farmer’s market opened, how would you dynamically determine what cutoff time to use? (automatically calculate it for every market date in your database)

Again, this is where **DATE\_ADD** and **PARSE\_DATETIME** functions come in.

For BigQuery:

SELECT

market\_date,

COUNT(\*) AS num\_sales

FROM (

SELECT

c.market\_date,

time(PARSE\_DATETIME('%I:%M %P', m.market\_start\_time)) AS market\_start\_time,

time(PARSE\_DATETIME('%I:%M %P', m.market\_end\_time)) AS market\_end\_time,

time(c.transaction\_time) AS transaction\_time, PARSE\_DATETIME('%Y-%m-%d %I:%M %P', CONCAT(c.market\_date, " ", m.market\_start\_time )) AS market\_start\_datetime,

PARSE\_DATETIME('%Y-%m-%d %I:%M %P', CONCAT(c.market\_date, " ", m.market\_end\_time )) AS market\_end\_datetime,

PARSE\_DATETIME('%Y-%m-%d %H:%M:%S', CONCAT(c.market\_date, " ", c.transaction\_time )) AS market\_date\_transaction\_time,

c.product\_id,

c.vendor\_id,

c.customer\_id,

c.quantity,

c.cost\_to\_customer\_per\_qty

FROM farmers\_market.customer\_purchases c

LEFT JOIN farmers\_market.market\_date\_info m

ON c.market\_date = m.market\_date)

WHERE market\_date\_transaction\_time <= DATE\_ADD(market\_start\_datetime, INTERVAL 30 MINUTE)

GROUP BY market\_date

ORDER BY market\_date;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## **Date Functions in Aggregate Summaries and Window Functions :-**

In this section, we’ll explore a few ways that you can use date functions when **summarizing** data.

## Question: Let’s say we wanted to get a profile of each farmer’s market customer’s habits over time.

* First purchase date
* Last purchase date
* Count of distinct purchases

Q1. If we wanted to determine for how long this person has been a customer of the farmer’s market?

We can get the **difference between the first and last purchase**.

**For BigQuery:**

SELECT customer\_id,

MIN(market\_date) AS first\_purchase,

MAX(market\_date) AS last\_purchase,

COUNT(DISTINCT market\_date) AS count\_of\_purchase\_dates,

DATE\_DIFF(MAX(market\_date), MIN(market\_date), day) AS days\_between\_first\_last\_purchase

FROM farmers\_market.customer\_purchases

GROUP BY customer\_id

Q2. If we wanted to also know how long it’s been since the customer last made a purchase?

We can use the **CURRENT\_DATE**() function.

**CURRENT\_DATE()** can be used to represent the current system date in any calculation that requires a date or datetime parameter.

**For BigQuery:**

SELECT customer\_id,

MIN(market\_date) AS first\_purchase,

MAX(market\_date) AS last\_purchase,

COUNT(DISTINCT market\_date) AS count\_of\_purchase\_dates,

DATE\_DIFF(MAX(market\_date), MIN(market\_date), day) AS days\_between\_first\_last\_purchase,

DATE\_DIFF(CURRENT\_DATE(), MAX(market\_date), day) AS days\_since\_last\_purchase

FROM farmers\_market.customer\_purchases

GROUP BY customer\_id

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**With Window Functions :-**

## Question: Write a query that gives us the days between each purchase a customer makes.

Step 1: How can we get the previous purchase? - LAG()

SELECT customer\_id,

market\_date,

LAG(market\_date,1) OVER (PARTITION BY customer\_id ORDER BY market\_

date) AS last\_purchase

FROM farmers\_market.customer\_purchases;

Step 2: How to get the no. of days between current and the last purchase date? - DATE\_DIFF()

SELECT

customer\_id,

market\_date,

LAG(market\_date, 1) OVER (PARTITION BY customer\_id ORDER BY market\_date) AS last\_purchase,

DATE\_DIFF(market\_date, (LAG(market\_date, 1) OVER (PARTITION BY customer\_id ORDER BY market\_date)), DAY) AS count\_bw\_prchs

FROM farmers\_market.customer\_purchases;

* Here, we didn’t quite accomplish the goal of finding the difference between each purchase date and the previous purchase date.
* Because there are multiple rows with the same date in cases where the customer purchased multiple items on the same date.

We can resolve this in a few ways.

* One approach is to remove the duplicates by using the DISTINCT keyword, and then use a WHERE clause filter to remove rows where the two dates (current and next purchase) are the same (because multiple purchases were made on the same date).
* Another is to remove duplicates in the initial dataset and use a subquery (a query inside a query) to get the date differences. Doing this and moving the window functions to the outer query will also fix the issue of the RANK counting each purchase, when we really want to count each purchase date.

**For BigQuery:**

SELECT

x.customer\_id,

x.market\_date,

LAG(x.market\_date, 1) OVER(PARTITION BY x.customer\_id ORDER BY x.market\_date) AS last\_prchs,

DATE\_DIFF(x.market\_date, LAG(x.market\_date, 1) OVER(PARTITION BY x.customer\_id ORDER BY x.market\_date), DAY) AS days\_bw\_prch

FROM

( SELECT

DISTINCT customer\_id,

market\_date

FROM farmers\_market.customer\_purchases

) AS x

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Question: Today’s date is May 31, 2019, and the marketing director of the farmer’s market wants to give infrequent customers (with only 1 purchase) an incentive to return to the market in April.

Pull up a list of everyone who only purchased once during the previous month, because they want to email all of those customers with a coupon to receive a discount on a purchase made in April.

* First, we must find everyone who made a purchase 31 days prior to May 31, 2019.
* Then, we need to filter that list to those who came to the market and made purchase(s) on a single market date.

This query would retrieve a list of one row per market date per customer within that date range:

**For BigQuery:**

SELECT DISTINCT customer\_id, market\_date

FROM farmers\_market.customer\_purchases

WHERE DATE\_DIFF('2019-05-31', market\_date, DAY) <= 31

Then, we could query the results of that query, **count the distinct *market\_date* values** per customer during that time, and **filter to those with exactly one market date**, using the HAVING clause.

**For BigQuery:**

SELECT x.customer\_id,

COUNT(x.market\_date) AS market\_count

FROM (

SELECT DISTINCT customer\_id, market\_date

FROM farmers\_market.customer\_purchases

WHERE DATE\_DIFF('2019-05-31', market\_date, DAY) BETWEEN 0 AND 31

) x

GROUP BY x.customer\_id

HAVING COUNT(DISTINCT market\_date) = 1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_