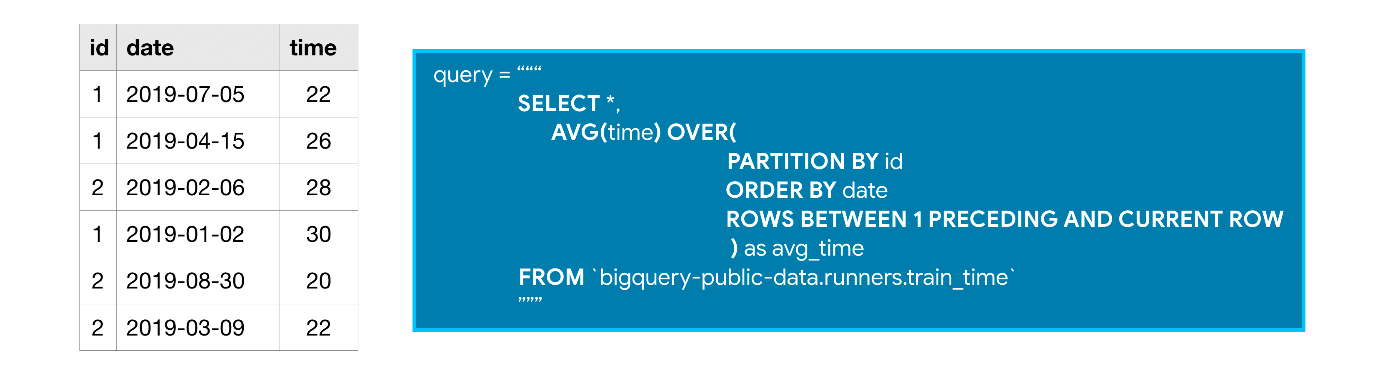
Introduction

SQL'e Giriş mikro-kursunda, satır kümelerine göre hesaplamalar yapan **toplama fonksiyonlarını (aggregate functions)** nasıl kullanacağınızı öğrenmiştiniz. Bu eğitimde ise, yine bir satır kümesi üzerinde çalışan **analitik fonksiyonları (analytic functions)** nasıl tanımlayacağınızı öğreneceksiniz. Ancak, toplama fonksiyonlarından farklı olarak, analitik fonksiyonlar orijinal tablodaki her bir satır için (potansiyel olarak farklı) bir değer döndürür.

Analitik fonksiyonlar, nispeten basit bir sözdizimiyle karmaşık hesaplamalar yapmamızı sağlar. Örneğin, diğer niceliklerin yanı sıra hareketli ortalamaları ve kümülatif toplamları hızlı bir şekilde hesaplayabiliriz.

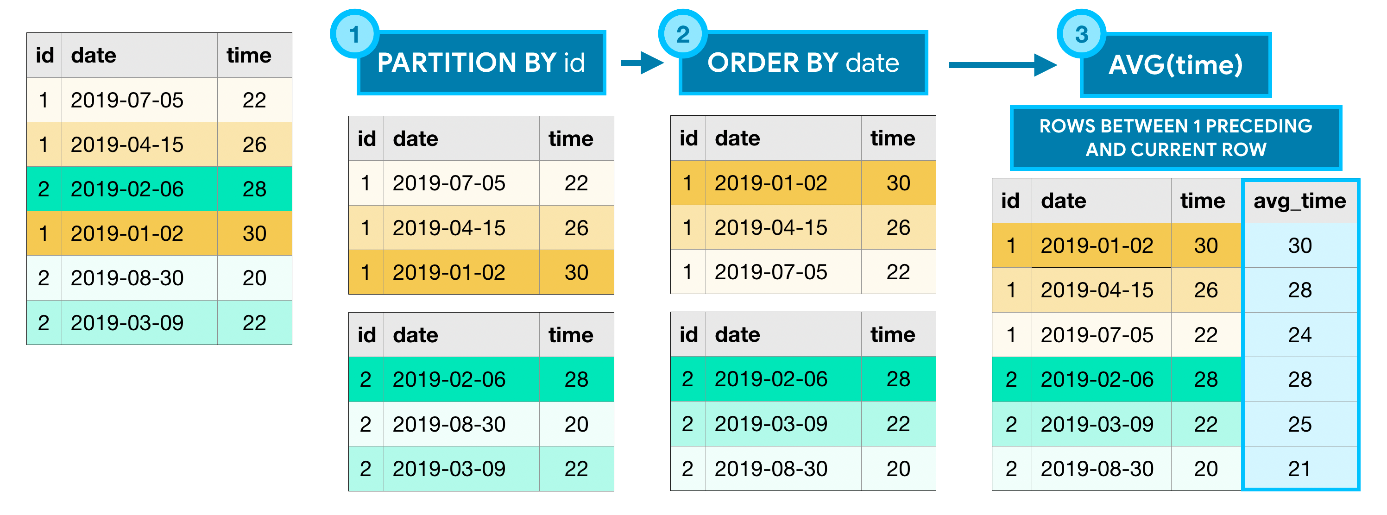
# Syntax

Analitik fonksiyonların nasıl yazıldığını anlamak için, bir yarışa antrenman yapan iki farklı kişiye ait verileri içeren küçük bir tabloyla çalışacağız. **id** sütunu her bir koşucuyu tanımlar, **date** sütunu antrenman seansının gününü tutar ve **time** sütunu ise koşucunun antrenmana ayırdığı süreyi (dakika cinsinden) gösterir. Diyelim ki, her koşucu için, mevcut ve önceki antrenman seanslarının ortalamasını alarak bir hareketli ortalama hesaplamak istiyoruz. Bunu aşağıdaki sorgu ile yapabiliriz:



Tüm analitik fonksiyonların, her bir hesaplamada kullanılan satır kümelerini tanımlayan bir **OVER** cümlesi vardır. **OVER** cümlesinin üç (isteğe bağlı) bölümü bulunur:

* **PARTITION BY** cümlesi, tablodaki satırları farklı gruplara ayırır. Yukarıdaki sorguda, hesaplamaların koşucuya göre ayrılması için **id**'ye göre bölme işlemi yapıyoruz.
* **ORDER BY** cümlesi, her bir bölüm içinde bir sıralama tanımlar. Örnek sorguda, **date** sütununa göre sıralama yapmak, daha önceki antrenman seanslarının önce gelmesini sağlar.
* Son cümle (**ROWS BETWEEN 1 PRECEDING AND CURRENT ROW**) bir **pencere çerçevesi (window frame)** cümlesi olarak bilinir. Bu, her bir hesaplamada kullanılan satır kümesini tanımlar. Bu satır grubuna bir **pencere (window)** diyebiliriz. (Aslında, analitik fonksiyonlara bazen **analitik pencere fonksiyonları** veya kısaca **pencere fonksiyonları** da denir!)



# (More on) window frame clauses[¶](https://www.kaggle.com/code/alexisbcook/analytic-functions" \l "(More-on)-window-frame-clauses" \t "_self)

Pencere çerçevesi cümlelerini yazmanın birçok yolu vardır:

* ROWS BETWEEN 1 PRECEDING AND CURRENT ROW - the previous row and the current row.
* ROWS BETWEEN 3 PRECEDING AND 1 FOLLOWING - the 3 previous rows, the current row, and the following row.
* ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING - all rows in the partition.

Elbette, bu kapsamlı bir liste değil ve tahmin edebileceğiniz gibi daha birçok seçenek var! Aşağıdaki kodda, bu ifadelerden bazılarının nasıl kullanıldığını göreceksiniz.

# Three types of analytic functions[¶](https://www.kaggle.com/code/alexisbcook/analytic-functions" \l "Three-types-of-analytic-functions" \t "_self)

Yukarıdaki örnekte birçok analitik fonksiyondan yalnızca biri kullanılmaktadır. BigQuery çok çeşitli analitik fonksiyonları destekler ve burada bunlardan birkaçını inceleyeceğiz. Tam liste için dokümanlara göz atabilirsiniz.

### **1) Analytic aggregate functions[¶](https://www.kaggle.com/code/alexisbcook/analytic-functions" \l "1)-Analytic-aggregate-functions" \t "_self)**

Hatırlayacağınız gibi, AVG() (yukarıdaki örnekten de anlaşılacağı gibi) bir toplama fonksiyonudur. OVER ifadesi, bir analitik (toplama) fonksiyonu olarak ele alınmasını sağlar. Toplama fonksiyonları, pencere içindeki tüm değerleri girdi olarak alır ve tek bir değer döndürür.

* **MIN()** (or **MAX()**) - Returns the minimum (or maximum) of input values
* **AVG()** (or **SUM()**) - Returns the average (or sum) of input values
* **COUNT()** - Returns the number of rows in the input

### **2) Analytic navigation functions**

Gezinme işlevleri, geçerli satırdan (genellikle) farklı bir satırdaki değere göre bir değer atar.

* **FIRST\_VALUE()** (or **LAST\_VALUE()**) - Returns the first (or last) value in the input
* **LEAD()** (and **LAG()**) - Returns the value on a subsequent (or preceding) row

### **3) Analytic numbering functions[¶](https://www.kaggle.com/code/alexisbcook/analytic-functions" \l "3)-Analytic-numbering-functions" \t "_self)**

Numaralandırma fonksiyonları, sıralamaya göre her satıra tam sayı değerleri atar.

* **ROW\_NUMBER()** - Returns the order in which rows appear in the input (starting with 1)
* **RANK()** - All rows with the same value in the ordering column receive the same rank value, where the next row receives a rank value which increments by the number of rows with the previous rank value.

# Example[¶](https://www.kaggle.com/code/alexisbcook/analytic-functions" \l "Example" \t "_self)

San Francisco Açık Veri veri kümesiyle çalışacağız. Öncelikle bikeshare\_trips tablosunun ilk birkaç satırını inceleyeceğiz. (İlgili kod gizlidir, ancak aşağıdaki "Kod" butonuna tıklayarak onu tekrar açabilirsiniz.)

from google.cloud import bigquery

*# Create a "Client" object*

client = bigquery.Client()

*# Construct a reference to the "san\_francisco" dataset*

dataset\_ref = client.dataset("san\_francisco", project="bigquery-public-data")

*# API request - fetch the dataset*

dataset = client.get\_dataset(dataset\_ref)

*# Construct a reference to the "bikeshare\_trips" table*

table\_ref = dataset\_ref.table("bikeshare\_trips")

*# API request - fetch the table*

table = client.get\_table(table\_ref)

*# Preview the first five lines of the table*

client.list\_rows(table, max\_results=5).to\_dataframe()

Using Kaggle's public dataset BigQuery integration.

/opt/conda/lib/python3.7/site-packages/ipykernel\_launcher.py:19: UserWarning: Cannot use bqstorage\_client if max\_results is set, reverting to fetching data with the tabledata.list endpoint.

| trip\_id | duration\_sec | start\_date | start\_station\_name | start\_station\_id | end\_date | end\_station\_name | end\_station\_id | bike\_number | zip\_code | subscriber\_type |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1235850 | 1540 | 2016-06-11 08:19:00+00:00 | San Jose Diridon Caltrain Station | 2 | 2016-06-11 08:45:00+00:00 | San Jose Diridon Caltrain Station | 2 | 124 | 15206 | Customer |
| 1 | 1219337 | 6324 | 2016-05-29 12:49:00+00:00 | San Jose Diridon Caltrain Station | 2 | 2016-05-29 14:34:00+00:00 | San Jose Diridon Caltrain Station | 2 | 174 | 55416 | Customer |
| 2 | 793762 | 115572 | 2015-06-04 09:22:00+00:00 | San Jose Diridon Caltrain Station | 2 | 2015-06-05 17:28:00+00:00 | San Jose Diridon Caltrain Station | 2 | 190 | 95391 | Customer |
| 3 | 453845 | 54120 | 2014-09-15 16:53:00+00:00 | San Jose Diridon Caltrain Station | 2 | 2014-09-16 07:55:00+00:00 | San Jose Diridon Caltrain Station | 2 | 127 | 81 | Customer |
| 4 | 1245113 | 5018 | 2016-06-17 20:08:00+00:00 | San Jose Diridon Caltrain Station | 2 | 2016-06-17 21:32:00+00:00 | San Jose Diridon Caltrain Station | 2 | 153 | 95070 | Customer |

Tablonun her satırı farklı bir bisiklet yolculuğuna karşılık geliyor ve 2015 yılında her tarih için toplam yolculuk sayısını hesaplamak için analitik bir fonksiyon kullanabiliriz.

*# Query to count the (cumulative) number of trips per day*

num\_trips\_query = """

WITH trips\_by\_day AS

(

SELECT DATE(start\_date) AS trip\_date,

COUNT(\*) as num\_trips

FROM `bigquery-public-data.san\_francisco.bikeshare\_trips`

WHERE EXTRACT(YEAR FROM start\_date) = 2015

GROUP BY trip\_date

)

SELECT \*,

SUM(num\_trips)

OVER (

ORDER BY trip\_date

ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW

) AS cumulative\_trips

FROM trips\_by\_day

"""

*# Run the query, and return a pandas DataFrame*

num\_trips\_result = client.query(num\_trips\_query).result().to\_dataframe()

num\_trips\_result.head()

/opt/conda/lib/python3.7/site-packages/google/cloud/bigquery/client.py:440: UserWarning: Cannot create BigQuery Storage client, the dependency google-cloud-bigquery-storage is not installed.

"Cannot create BigQuery Storage client, the dependency "

| trip\_date | num\_trips | cumulative\_trips |
| --- | --- | --- |
| 0 | 2015-01-20 | 1213 | 16742 |
| 1 | 2015-01-22 | 1224 | 19280 |
| 2 | 2015-04-22 | 1376 | 108778 |
| 3 | 2015-03-23 | 1247 | 76875 |
| 4 | 2015-09-04 | 1129 | 248182 |

Bu sorgu, ilk olarak günlük sefer sayısını hesaplamak için bir **ortak tablo ifadesi (CTE)** kullanır. Ardından, bir toplama fonksiyonu olarak **SUM()** kullanırız.

* **PARTITION BY** cümlesi olmadığı için, tüm tablo tek bir bölüm olarak ele alınır.
* **ORDER BY** cümlesi, satırları tarihe göre sıralar ve böylece daha eski tarihler önce gelir.
* **Pencere çerçevesi (window frame)** cümlesini **ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW** olarak ayarlayarak, mevcut tarih de dahil olmak üzere o tarihe kadar olan tüm satırların (kümülatif) toplamı hesaplamak için kullanılmasını sağlarız. (Not: Eğer **dokümantasyonu** okursanız, bunun varsayılan davranış olduğunu ve bu **pencere çerçevesi** cümlesini çıkardığımızda da sorgunun aynı sonucu döndüreceğini göreceksiniz.)

Bir sonraki sorgu, her bisikletin 25 Ekim 2015 tarihinde güne başladığı (**start\_station\_id**) ve bitirdiği (**end\_station\_id**) istasyonları takip eder.

*# Query to track beginning and ending stations on October 25, 2015, for each bike*

start\_end\_query = """

SELECT bike\_number,

TIME(start\_date) AS trip\_time,

FIRST\_VALUE(start\_station\_id)

OVER (

PARTITION BY bike\_number

ORDER BY start\_date

ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING

) AS first\_station\_id,

LAST\_VALUE(end\_station\_id)

OVER (

PARTITION BY bike\_number

ORDER BY start\_date

ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING

) AS last\_station\_id,

start\_station\_id,

end\_station\_id

FROM `bigquery-public-data.san\_francisco.bikeshare\_trips`

WHERE DATE(start\_date) = '2015-10-25'

"""

*# Run the query, and return a pandas DataFrame*

start\_end\_result = client.query(start\_end\_query).result().to\_dataframe()

start\_end\_result.head()

/opt/conda/lib/python3.7/site-packages/google/cloud/bigquery/client.py:440: UserWarning: Cannot create BigQuery Storage client, the dependency google-cloud-bigquery-storage is not installed.

"Cannot create BigQuery Storage client, the dependency "

| bike\_number | trip\_time | first\_station\_id | last\_station\_id | start\_station\_id | end\_station\_id |
| --- | --- | --- | --- | --- | --- |
| 0 | 294 | 14:29:00 | 64 | 77 | 64 | 77 |
| 1 | 395 | 00:24:00 | 70 | 72 | 70 | 72 |
| 2 | 480 | 15:56:00 | 39 | 67 | 39 | 67 |
| 3 | 581 | 12:41:00 | 77 | 73 | 77 | 60 |
| 4 | 581 | 12:56:00 | 77 | 73 | 60 | 73 |

Bu sorgu, hem **FIRST\_VALUE()** hem de **LAST\_VALUE()** fonksiyonlarını analitik fonksiyonlar olarak kullanır.

* **PARTITION BY** cümlesi, verileri **bike\_number** sütununa göre bölümlere ayırır. Bu sütun bisikletler için benzersiz kimlikler içerdiğinden, bu işlem hesaplamaların her bisiklet için ayrı ayrı yapılmasını sağlar.
* **ORDER BY** cümlesi, her bir bölümdeki satırları kronolojik sıraya göre dizer.
* **Pencere çerçevesi (window frame)** cümlesi **ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING** olduğundan, her bir satır için hesaplamayı yapmak üzere tüm bölümü kullanılır. (Bu, aynı bölümdeki satırlar için hesaplanan değerlerin aynı olmasını sağlar.)