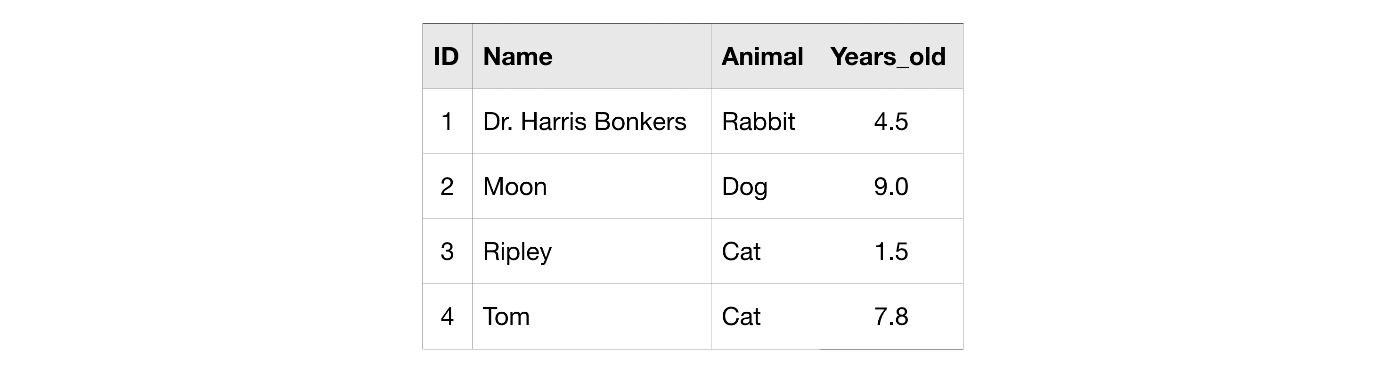
Introduction

Öğrendiğiniz tüm bilgilerle, SQL sorgularınız oldukça uzamaya başladı. Bu da onları anlamayı (ve hatalarını ayıklamayı) zorlaştırabilir.

Şimdi, sorgularınızı düzenlemek ve daha kolay okunabilir hale getirmek için **AS** ve **WITH** anahtar kelimelerini nasıl kullanacağınızı öğreneceksiniz.

Bu süreçte, aşina olduğumuz **pets** tablosunu kullanacağız, ancak şimdi hayvanların yaşlarını da içeriyor.



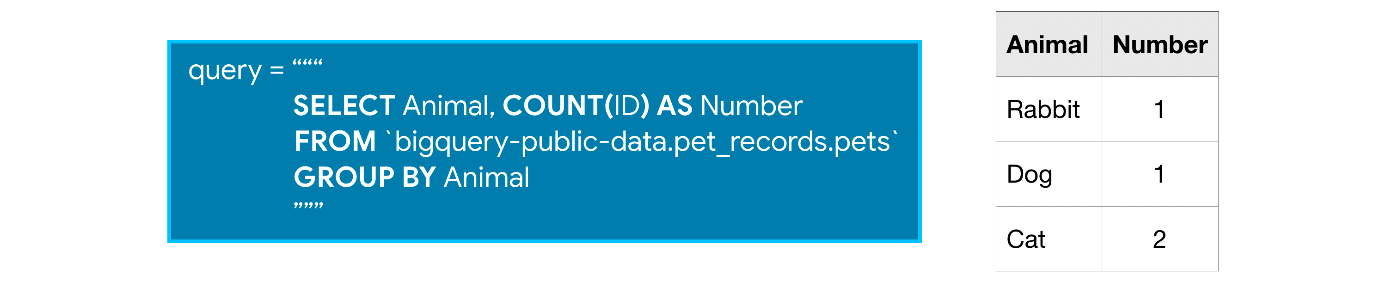
# AS

Daha önceki bir eğitimde, sorgularınız tarafından oluşturulan sütunları yeniden adlandırmak için **AS**'i nasıl kullanacağınızı öğrenmiştiniz. Buna **takma ad verme (aliasing)** adı verilir. Bu, Python'ın import pandas as pd veya import seaborn as sns gibi içe aktarma işlemlerinde as kullanmasına benzer.

SQL'de **AS** kullanmak için, onu seçtiğiniz sütunun hemen arkasına ekleyin. İşte **AS** ifadesi **olmadan** bir sorgu örneği:



Ve işte aynı sorgunun AS ile yapılmış bir örneği.



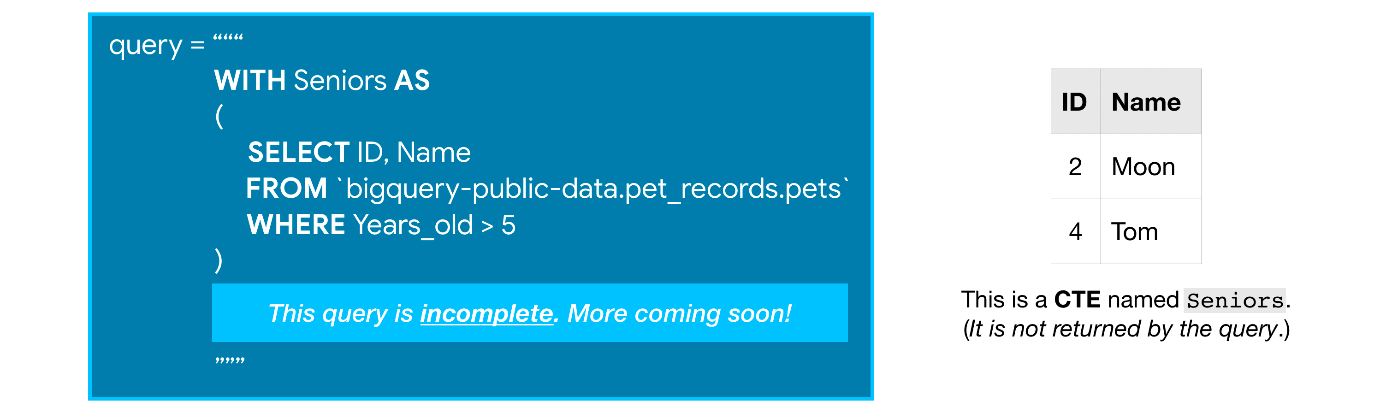
Bu sorgular aynı bilgiyi döndürür, ancak ikinci sorguda COUNT() fonksiyonu tarafından döndürülen sütun varsayılan f0\_\_ adı yerine Number olarak adlandırılacaktır.

# WITH ... AS

Yalnız başına **AS**, sorgunuz tarafından döndürülen verileri düzenlemek için uygun bir yoldur. Ancak, "ortak tablo ifadesi" (common table expression) adı verilen bir yapıda **WITH** ile birleştirildiğinde çok daha güçlü hale gelir.

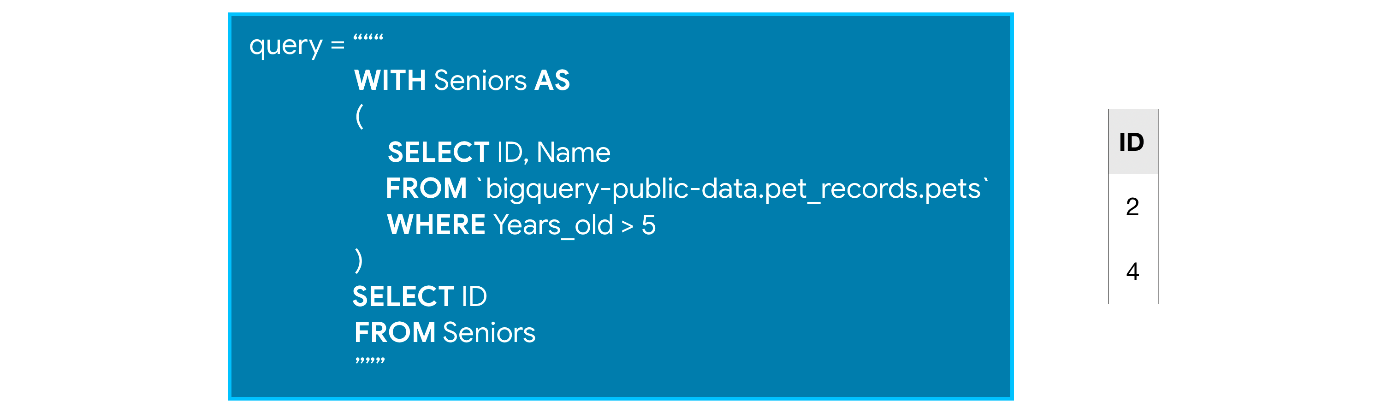
**Ortak tablo ifadesi (CTE)**, sorgunuz içinde döndürdüğünüz geçici bir tablodur. CTE'ler, sorgularınızı okunabilir parçalara ayırmak için faydalıdır ve onlara karşı sorgular yazabilirsiniz.

Örneğin, özellikle yaşlı hayvanlar hakkında sorular sormak için **pets** tablosunu kullanmak isteyebilirsiniz. Bunun için, beş yaşından büyük hayvanlar hakkındaki bilgileri içeren bir CTE oluşturarak başlayabilirsiniz:



Yukarıdaki bu eksik sorgu herhangi bir şey döndürmeyecektir, ancak daha sonra **Seniors** olarak adlandırarak geri kalan sorguyu yazarken referans verebileceğimiz bir CTE (ortak tablo ifadesi) oluşturur.

CTE'den istediğimiz bilgileri çekerek sorguyu tamamlayabiliriz. Aşağıdaki tam sorgu, önce CTE'yi oluşturur, ardından ondan tüm ID'leri döndürür.



Bunu bir CTE (ortak tablo ifadesi) olmadan da yapabilirsiniz, ancak bu, çok uzun bir sorgunun ilk bölümü olsaydı, CTE'yi kaldırmak sorgunun takip edilmesini çok daha zor hale getirirdi.

Ayrıca, CTE'lerin yalnızca onları oluşturduğunuz sorgu içinde var olduğunu ve daha sonraki sorgularda onlara referans veremeyeceğinizi unutmamak önemlidir. Bu nedenle, bir CTE kullanan herhangi bir sorgu her zaman iki bölüme ayrılır: (1) önce CTE'yi oluştururuz ve sonra (2) CTE'yi kullanan bir sorgu yazarız.

# Example: How many Bitcoin transactions are made per month?[¶](https://www.kaggle.com/code/dansbecker/as-with#Example:-How-many-Bitcoin-transactions-are-made-per-month?)

Bir Bitcoin işlem veri kümesinin tüm zaman aralığı boyunca her gün kaç Bitcoin işlemi yapıldığını bulmak için bir CTE kullanacağız.

İşlemler tablosunu inceleyeceğiz. İşte ilk birkaç satırın görünümü. (İlgili kod gizlidir, ancak aşağıdaki "Kod" butonuna tıklayarak gizlemeyi kaldırabilirsiniz.)

from google.cloud import bigquery

*# Create a "Client" object*

client = bigquery.Client()

*# Construct a reference to the "crypto\_bitcoin" dataset*

dataset\_ref = client.dataset("crypto\_bitcoin", project="bigquery-public-data")

*# API request - fetch the dataset*

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

*# Construct a reference to the "transactions" table*

table\_ref = dataset\_ref.table("transactions")

*# API request - fetch the table*

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

*# Preview the first five lines of the "transactions" 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.

| hash | size | virtual\_size | version | lock\_time | block\_hash | block\_number | block\_timestamp | block\_timestamp\_month | input\_count | output\_count | input\_value | output\_value | is\_coinbase | fee | inputs | outputs |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | a16f3ce4dd5deb92d98ef5cf8afeaf0775ebca408f708b... | 275 | 275 | 1 | 0 | 00000000dc55860c8a29c58d45209318fa9e9dc2c1833a... | 181 | 2009-01-12 06:02:13+00:00 | 2009-01-01 | 1 | 2 | 4000000000.000000000 | 4000000000.000000000 | False | 0E-9 | [{'index': 0, 'spent\_transaction\_hash': 'f4184... | [{'index': 0, 'script\_asm': '04b5abd412d4341b4... |
| 1 | 591e91f809d716912ca1d4a9295e70c3e78bab077683f7... | 275 | 275 | 1 | 0 | 0000000054487811fc4ff7a95be738aa5ad9320c394c48... | 182 | 2009-01-12 06:12:16+00:00 | 2009-01-01 | 1 | 2 | 3000000000.000000000 | 3000000000.000000000 | False | 0E-9 | [{'index': 0, 'spent\_transaction\_hash': 'a16f3... | [{'index': 0, 'script\_asm': '0401518fa1d1e1e3e... |
| 2 | 12b5633bad1f9c167d523ad1aa1947b2732a865bf5414e... | 276 | 276 | 1 | 0 | 00000000f46e513f038baf6f2d9a95b2a28d8a6c985bcf... | 183 | 2009-01-12 06:34:22+00:00 | 2009-01-01 | 1 | 2 | 2900000000.000000000 | 2900000000.000000000 | False | 0E-9 | [{'index': 0, 'spent\_transaction\_hash': '591e9... | [{'index': 0, 'script\_asm': '04baa9d3665315562... |
| 3 | 828ef3b079f9c23829c56fe86e85b4a69d9e06e5b54ea5... | 276 | 276 | 1 | 0 | 00000000fb5b44edc7a1aa105075564a179d65506e2bd2... | 248 | 2009-01-12 20:04:20+00:00 | 2009-01-01 | 1 | 2 | 2800000000.000000000 | 2800000000.000000000 | False | 0E-9 | [{'index': 0, 'spent\_transaction\_hash': '12b56... | [{'index': 0, 'script\_asm': '04bed827d37474bef... |
| 4 | 35288d269cee1941eaebb2ea85e32b42cdb2b04284a56d... | 277 | 277 | 1 | 0 | 00000000689051c09ff2cd091cc4c22c10b965eb8db3ad... | 545 | 2009-01-15 05:48:32+00:00 | 2009-01-01 | 1 | 2 | 2500000000.000000000 | 2500000000.000000000 | False | 0E-9 | [{'index': 0, 'spent\_transaction\_hash': 'd71fd... | [{'index': 0, 'script\_asm': '044a656f065871a35... |

**block\_timestamp** sütunu, her işlemin tarihini **DATETIME** formatında içerdiğinden, bu tarihleri **DATE()** komutunu kullanarak **DATE** formatına dönüştüreceğiz.

Bunu bir CTE (ortak tablo ifadesi) kullanarak yapacağız. Ardından, sorgunun bir sonraki bölümü her tarih için işlem sayısını sayacak ve tabloyu daha eski tarihler ilk sırada görünecek şekilde sıralayacaktır.

*# Query to select the number of transactions per date, sorted by date*

query\_with\_CTE = """

WITH time AS

(

SELECT DATE(block\_timestamp) AS trans\_date

FROM `bigquery-public-data.crypto\_bitcoin.transactions`

)

SELECT COUNT(1) AS transactions,

trans\_date

FROM time

GROUP BY trans\_date

ORDER BY trans\_date

"""

*# Set up the query (cancel the query if it would use too much of*

*# your quota, with the limit set to 10 GB)*

safe\_config = bigquery.QueryJobConfig(maximum\_bytes\_billed=10\*\*10)

query\_job = client.query(query\_with\_CTE, job\_config=safe\_config)

*# API request - run the query, and convert the results to a pandas DataFrame*

transactions\_by\_date = query\_job.to\_dataframe()

*# Print the first five rows*

transactions\_by\_date.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 "

| transactions | trans\_date |
| --- | --- |
| 0 | 1 | 2009-01-03 |
| 1 | 14 | 2009-01-09 |
| 2 | 61 | 2009-01-10 |
| 3 | 93 | 2009-01-11 |
| 4 | 101 | 2009-01-12 |

Sıralı olarak döndürüldükleri için, ham sonuçları kolayca çizerek bu veri kümesinin tüm zaman aralığı boyunca günlük Bitcoin işlem sayısını gösterebiliriz.

transactions\_by\_date.set\_index('trans\_date').plot()

<AxesSubplot:xlabel='trans\_date'>

Gördüğünüz gibi, ortak tablo ifadeleri (CTE'ler), veri temizliğinizin çoğunu SQL'e taşımanıza olanak tanır. Bu, özellikle BigQuery söz konusu olduğunda iyi bir özelliktir, çünkü Pandas'ta yapmaktan çok daha hızlıdır.