Introduction

Bazı durumlarda sorgunuzun verimli olup olmaması önemli değildir. Örneğin, sadece bir kez çalıştırmayı düşündüğünüz ve küçük bir veri seti üzerinde çalışan bir sorgu yazabilirsiniz. Bu durumda, ihtiyacınız olan cevabı veren her şey işe yarayacaktır.

Peki ya bir web sitesine veri sağlayan bir sorgu gibi, birçok kez çalıştırılacak sorgulara ne demeli? Bu sorguların verimli olması gerekir ki kullanıcıları web sitenizin yüklenmesini beklemek zorunda bırakmayın.

Ya da devasa veri setleri üzerindeki sorgulara ne demeli? Bunlar yavaş olabilir ve kötü yazıldıklarında bir işletmeye çok paraya mal olabilir.

Çoğu veritabanı sisteminin, sorgunuzu mümkün olan en etkili şekilde yorumlamaya/çalıştırmaya çalışan bir **sorgu eniyileyicisi (query optimizer)** vardır. Ancak yine de, birçok durumda büyük kazançlar sağlayabilecek birkaç strateji mevcuttur.

# Some useful functions[¶](https://www.kaggle.com/code/alexisbcook/writing-efficient-queries" \l "Some-useful-functions" \t "_self)

Farklı sorguların verimliliğini karşılaştırmak için iki fonksiyon kullanacağız:

* show\_amount\_of\_data\_scanned() shows the amount of data the query uses.
* show\_time\_to\_run() prints how long it takes for the query to execute.

from google.cloud import bigquery

from time import time

client = bigquery.Client()

def show\_amount\_of\_data\_scanned(query):

*# dry\_run lets us see how much data the query uses without running it*

dry\_run\_config = bigquery.QueryJobConfig(dry\_run=True)

query\_job = client.query(query, job\_config=dry\_run\_config)

print('Data processed: **{}** GB'.format(round(query\_job.total\_bytes\_processed / 10\*\*9, 3)))

def show\_time\_to\_run(query):

time\_config = bigquery.QueryJobConfig(use\_query\_cache=False)

start = time()

query\_result = client.query(query, job\_config=time\_config).result()

end = time()

print('Time to run: **{}** seconds'.format(round(end-start, 3)))

Using Kaggle's public dataset BigQuery integration.

# Strategies[¶](https://www.kaggle.com/code/alexisbcook/writing-efficient-queries" \l "Strategies" \t "_self)

### **1) Only select the columns you want.**[**¶**](https://www.kaggle.com/code/alexisbcook/writing-efficient-queries#1)-Only-select-the-columns-you-want.)

Sorgulara SELECT \* FROM ... ile başlamak cazip gelebilir. Hangi sütunlara ihtiyacınız olduğunu düşünmenize gerek kalmadığı için kullanışlıdır. Ancak çok verimsiz olabilir.

Bu, özellikle ihtiyaç duymadığınız metin alanları varsa önemlidir, çünkü metin alanları genellikle diğer alanlardan daha büyüktür.

star\_query = "SELECT \* FROM `bigquery-public-data.github\_repos.contents`"

show\_amount\_of\_data\_scanned(star\_query)

basic\_query = "SELECT size, binary FROM `bigquery-public-data.github\_repos.contents`"

show\_amount\_of\_data\_scanned(basic\_query)

Data processed: 2682.118 GB

Data processed: 2.531 GB

Bu durumda, sorguyu tamamlamak için taranan verilerde 1000 kat azalma görüyoruz çünkü ham veriler, ihtiyaç duyabileceğimiz alanlardan 1000 kat daha büyük bir metin alanı içeriyordu.

### **2) Read less data.**[**¶**](https://www.kaggle.com/code/alexisbcook/writing-efficient-queries#2)-Read-less-data.)

Aşağıdaki her iki sorgu da San Francisco şehrinde tek yönlü bisiklet yolculuklarının ortalama süresini (saniye cinsinden) hesaplar.

more\_data\_query = """

SELECT MIN(start\_station\_name) AS start\_station\_name,

MIN(end\_station\_name) AS end\_station\_name,

AVG(duration\_sec) AS avg\_duration\_sec

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

WHERE start\_station\_id != end\_station\_id

GROUP BY start\_station\_id, end\_station\_id

LIMIT 10

"""

show\_amount\_of\_data\_scanned(more\_data\_query)

less\_data\_query = """

SELECT start\_station\_name,

end\_station\_name,

AVG(duration\_sec) AS avg\_duration\_sec

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

WHERE start\_station\_name != end\_station\_name

GROUP BY start\_station\_name, end\_station\_name

LIMIT 10

"""

show\_amount\_of\_data\_scanned(less\_data\_query)

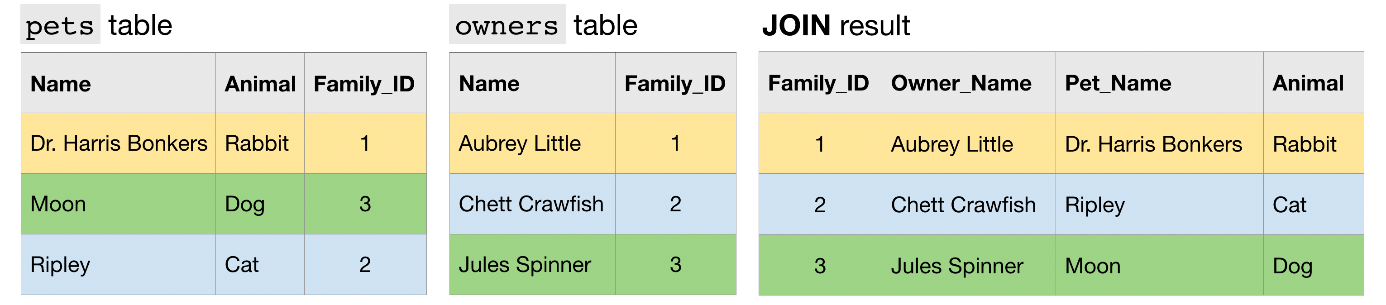
Data processed: 0.076 GB

Data processed: 0.06 GB

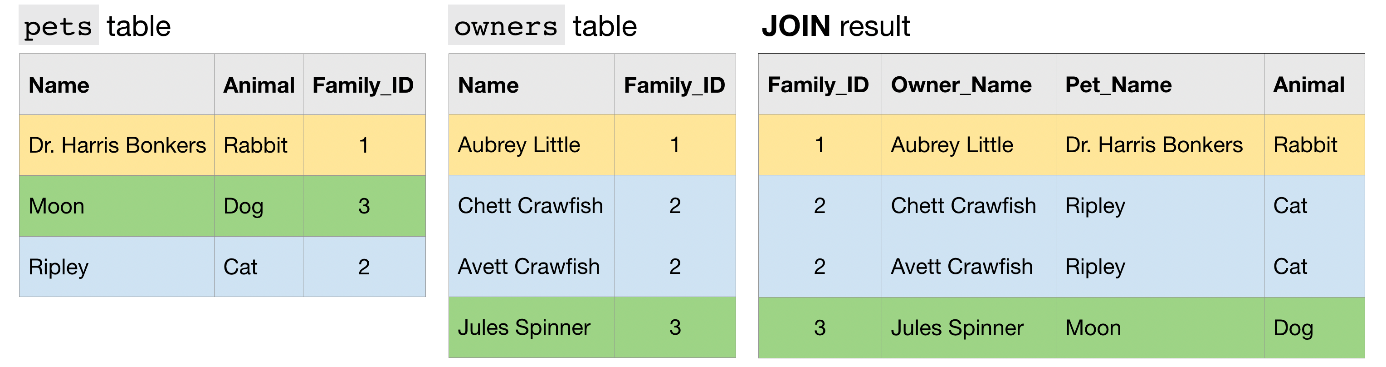
İstasyon kimliği ile istasyon adı arasında 1:1 ilişki olduğundan, sorguda start\_station\_id ve end\_station\_id sütunlarını kullanmamıza gerek yok. Yalnızca istasyon kimliklerini içeren sütunları kullandığımız için daha az veri tarıyoruz.

### **3) Avoid N:N JOINs.**[**¶**](https://www.kaggle.com/code/alexisbcook/writing-efficient-queries#3)-Avoid-N:N-JOINs.)

Bu kursta gerçekleştirdiğiniz JOIN'lerin çoğu 1:1 JOIN'lerdi. Bu durumda, her tablodaki her satırın diğer tabloda en fazla bir eşleşmesi vardır.



Bir diğer JOIN türü de N:1 JOIN'dir. Burada, bir tablodaki her satır, diğer tablodaki potansiyel olarak birçok satırla eşleşir.



Son olarak, N:N JOIN, bir tablodaki bir satır grubunun diğer tablodaki bir satır grubuyla eşleşebildiği bir JOIN türüdür. Genel olarak, diğer tüm koşullar eşit olduğunda, bu JOIN türünün, JOIN'e katılan iki (orijinal) tablodan çok daha fazla satıra sahip bir tablo ürettiğini unutmayın.



Şimdi gerçek bir veri kümesinden bir örnek üzerinde çalışacağız. Aşağıdaki her iki örnek de farklı commit yapanların sayısını ve çeşitli GitHub depolarındaki dosya sayısını sayar.

big\_join\_query = """

SELECT repo,

COUNT(DISTINCT c.committer.name) as num\_committers,

COUNT(DISTINCT f.id) AS num\_files

FROM `bigquery-public-data.github\_repos.commits` AS c,

UNNEST(c.repo\_name) AS repo

INNER JOIN `bigquery-public-data.github\_repos.files` AS f

ON f.repo\_name = repo

WHERE f.repo\_name IN ( 'tensorflow/tensorflow', 'facebook/react', 'twbs/bootstrap', 'apple/swift', 'Microsoft/vscode', 'torvalds/linux')

GROUP BY repo

ORDER BY repo

"""

show\_time\_to\_run(big\_join\_query)

small\_join\_query = """

WITH commits AS

(

SELECT COUNT(DISTINCT committer.name) AS num\_committers, repo

FROM `bigquery-public-data.github\_repos.commits`,

UNNEST(repo\_name) as repo

WHERE repo IN ( 'tensorflow/tensorflow', 'facebook/react', 'twbs/bootstrap', 'apple/swift', 'Microsoft/vscode', 'torvalds/linux')

GROUP BY repo

),

files AS

(

SELECT COUNT(DISTINCT id) AS num\_files, repo\_name as repo

FROM `bigquery-public-data.github\_repos.files`

WHERE repo\_name IN ( 'tensorflow/tensorflow', 'facebook/react', 'twbs/bootstrap', 'apple/swift', 'Microsoft/vscode', 'torvalds/linux')

GROUP BY repo

)

SELECT commits.repo, commits.num\_committers, files.num\_files

FROM commits

INNER JOIN files

ON commits.repo = files.repo

ORDER BY repo

"""

show\_time\_to\_run(small\_join\_query)

Time to run: 13.028 seconds

Time to run: 4.413 seconds

İlk sorgu büyük bir N:N JOIN'e sahip. Sorguyu JOIN boyutunu küçültecek şekilde yeniden yazdığımızda, çok daha hızlı çalıştığını görüyoruz.