Introduction

Tek bir tablodan istediğiniz formatta veri almak için gerekli araçlara sahipsiniz. Peki ya istediğiniz veriler birden fazla tabloya dağılmışsa?

İşte tam da bu noktada JOIN devreye giriyor! JOIN, pratik SQL iş akışlarında inanılmaz derecede önemlidir. Öyleyse başlayalım.

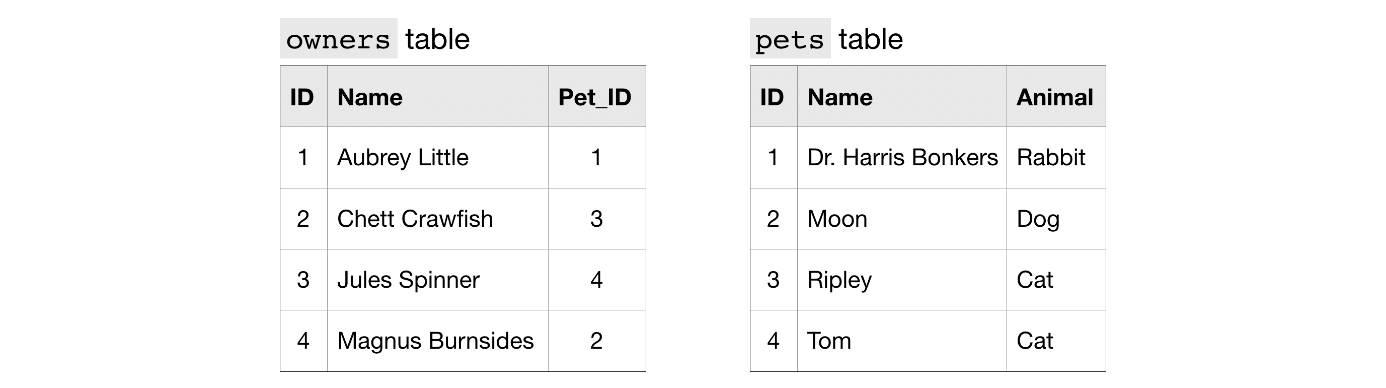
# Example

Hayali **pets** (evcil hayvanlar) tablomuza, üç sütunu olan bir tabloya bakacağız:

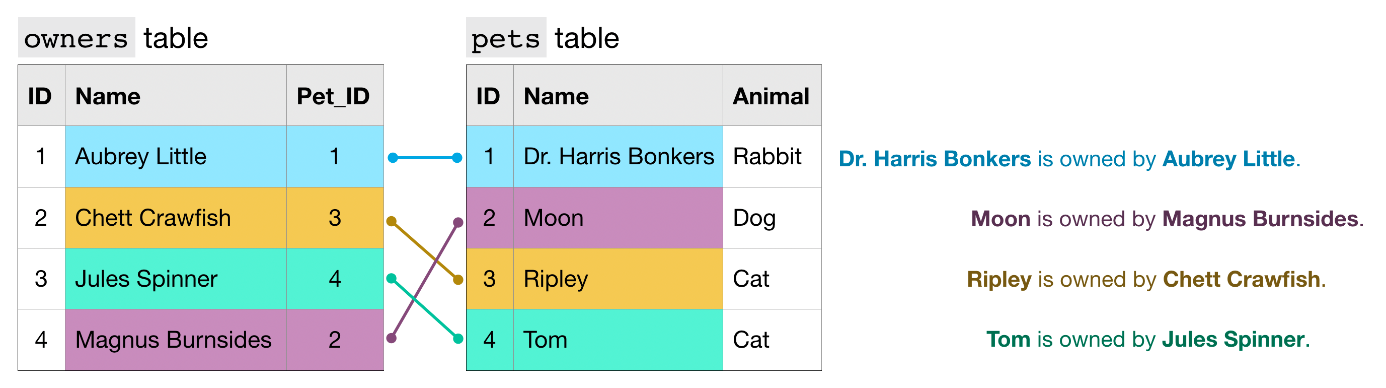
* **ID** - Evcil hayvan için ID numarası
* **Name** - Evcil hayvanın adı
* **Animal** - Hayvanın türü

Ayrıca **owners** (sahipler) adında başka bir tablo daha ekleyeceğiz. Bu tablonun da üç sütunu var:

* **ID** - Sahibi için ID numarası (evcil hayvanın ID numarasından farklı)
* **Name** - Sahibinin adı
* **Pet\_ID** - Sahibe ait evcil hayvanın ID numarası (**pets** tablosundaki evcil hayvanın ID numarasıyla eşleşir)



Belirli bir evcil hayvana ait bilgileri elde etmek için, pets tablosundaki ID sütununu owners tablosundaki Pet\_ID sütunuyla eşleştiriyoruz.



Örneğin,

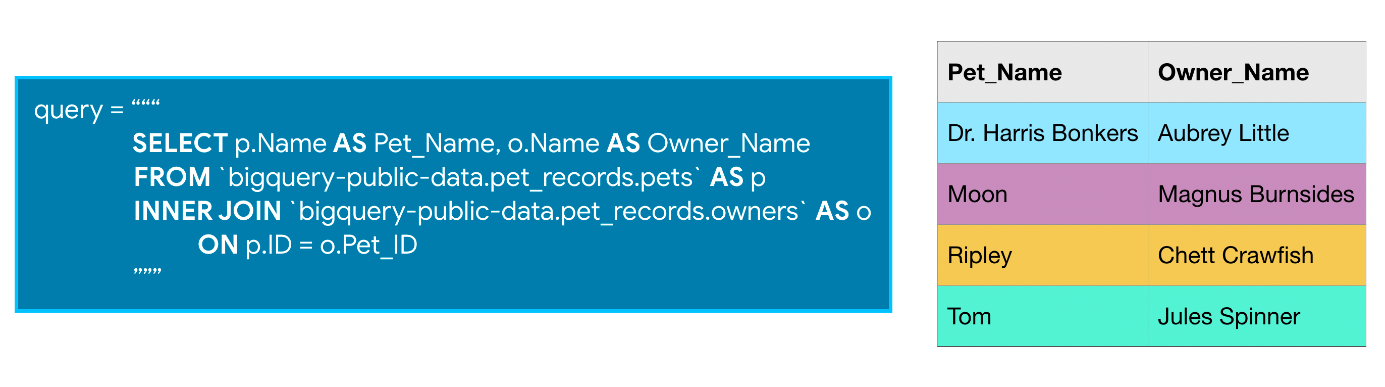
* **pets** tablosu, ID numarası 1 olan evcil hayvanın Dr. Harris Bonkers olduğunu gösteriyor.
* **owners** tablosu, ID numarası 1 olan evcil hayvanın sahibinin Aubrey Little olduğunu gösteriyor.

Bu iki bilgiyi bir araya getirdiğimizde, Dr. Harris Bonkers'ın sahibinin Aubrey Little olduğunu anlıyoruz.

Neyse ki, hangi sahibin hangi evcil hayvana ait olduğunu bulmak için bunu el yordamıyla yapmak zorunda değiliz. Bir sonraki bölümde, **pets** ve **owners** tablolarındaki bilgileri birleştiren yeni bir tablo oluşturmak için **JOIN**'i nasıl kullanacağınızı öğreneceksiniz.

# JOIN

JOIN kullanarak, sadece iki sütundan oluşan bir tablo oluşturmak için bir sorgu yazabiliriz: evcil hayvanın adı ve sahibinin adı.



Her iki tablodaki bilgileri, **pets** tablosundaki **ID** sütununun **owners** tablosundaki **Pet\_ID** sütunuyla eşleştiği satırları birleştirerek bir araya getiriyoruz.

Sorguda, **ON** ifadesi, tabloları birleştirmek için her bir tabloda hangi sütunun kullanılacağını belirler. **ID** sütunu her iki tabloda da bulunduğundan, hangisinin kullanılacağını netleştirmemiz gerekir. **p.ID**'yi **pets** tablosundaki **ID** sütununa, **o.Pet\_ID**'yi ise **owners** tablosundaki **Pet\_ID** sütununa atıfta bulunmak için kullanırız.

Genel olarak, tabloları birleştirirken her sütunun hangi tablodan geldiğini belirtmek iyi bir alışkanlıktır. Bu şekilde, sorguyu okumak için geri döndüğünüzde şemayı her seferinde kontrol etmek zorunda kalmazsınız.

Bugün kullandığımız **JOIN** türü **INNER JOIN** (iç birleştirme) olarak adlandırılır. Bu, birleştirme için kullandığınız sütunlardaki değer her iki tabloda da mevcutsa, bir satırın nihai çıktı tablosuna yerleştirileceği anlamına gelir. Örneğin, Tom'un 4 olan ID numarası **pets** tablosunda olmasaydı, bu sorgudan yalnızca 3 satır elde ederdik. Başka **JOIN** türleri de vardır, ancak **INNER JOIN** çok yaygın olarak kullanılır, bu nedenle başlamak için iyi bir türdür.

# Example: How many files are covered by each type of software license?[¶](https://www.kaggle.com/code/dansbecker/joining-data#Example:-How-many-files-are-covered-by-each-type-of-software-license?)

GitHub, yazılım projelerinde iş birliği yapmak için en popüler yerdir. Bir GitHub **reposu (repository)**, belirli bir projeyle ilişkili dosyaların bir koleksiyonudur.

GitHub'daki çoğu depo, nasıl kullanılacaklarına dair yasal kısıtlamaları belirleyen belirli bir yasal lisans altında paylaşılır. Örneğimizde, her bir lisans altında kaç farklı dosyanın yayınlandığına bakacağız.

Veritabanındaki iki tabloyla çalışacağız. İlk tablo, her GitHub deposunun adını (**repo\_name** sütununda) ve buna karşılık gelen lisansını sağlayan **licenses** tablosudur. İşte ilk beş satırın görünümü.

from google.cloud import bigquery

*# Create a "Client" object*

client = bigquery.Client()

*# Construct a reference to the "github\_repos" dataset*

dataset\_ref = client.dataset("github\_repos", project="bigquery-public-data")

*# API request - fetch the dataset*

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

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

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

*# API request - fetch the table*

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

*# Preview the first five lines of the "licenses" table*

client.list\_rows(licenses\_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.

| repo\_name | license |
| --- | --- |
| 0 | autarch/Dist-Zilla-Plugin-Test-TidyAll | artistic-2.0 |
| 1 | thundergnat/Prime-Factor | artistic-2.0 |
| 2 | kusha-b-k/Turabian\_Engin\_Fan | artistic-2.0 |
| 3 | onlinepremiumoutlet/onlinepremiumoutlet.github.io | artistic-2.0 |
| 4 | huangyuanlove/LiaoBa\_Service | artistic-2.0 |

İkinci tablo, diğer bilgilerin yanı sıra her dosyanın ait olduğu GitHub deposunu (repo\_name sütununda) sağlayan sample\_files tablosudur. Bu tablonun ilk birkaç satırı aşağıda basılmıştır.

*# Construct a reference to the "sample\_files" table*

files\_ref = dataset\_ref.table("sample\_files")

*# API request - fetch the table*

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

*# Preview the first five lines of the "sample\_files" table*

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

/opt/conda/lib/python3.7/site-packages/ipykernel\_launcher.py:8: UserWarning: Cannot use bqstorage\_client if max\_results is set, reverting to fetching data with the tabledata.list endpoint.

| repo\_name | ref | path | mode | id | symlink\_target |
| --- | --- | --- | --- | --- | --- |
| 0 | EOL/eol | refs/heads/master | generate/vendor/railties | 40960 | 0338c33fb3fda57db9e812ac7de969317cad4959 | /usr/share/rails-ruby1.8/railties |
| 1 | np/ling | refs/heads/master | tests/success/merger\_seq\_inferred.t/merger\_seq... | 40960 | dd4bb3d5ecabe5044d3fa5a36e0a9bf7ca878209 | ../../../fixtures/all/merger\_seq\_inferred.ll |
| 2 | np/ling | refs/heads/master | fixtures/sequence/lettype.ll | 40960 | 8fdf536def2633116d65b92b3b9257bcf06e3e45 | ../all/lettype.ll |
| 3 | np/ling | refs/heads/master | fixtures/failure/wrong\_order\_seq3.ll | 40960 | c2509ae1196c4bb79d7e60a3d679488ca4a753e9 | ../all/wrong\_order\_seq3.ll |
| 4 | np/ling | refs/heads/master | issues/sequence/keep.t | 40960 | 5721de3488fb32745dfc11ec482e5dd0331fecaf | ../keep.t |

Daha sonra, her iki tablodaki bilgileri kullanarak her lisansta kaç dosyanın yayınlandığını belirleyen bir sorgu yazıyoruz.

*# Query to determine the number of files per license, sorted by number of files*

query = """

SELECT L.license, COUNT(1) AS number\_of\_files

FROM `bigquery-public-data.github\_repos.sample\_files` AS sf

INNER JOIN `bigquery-public-data.github\_repos.licenses` AS L

ON sf.repo\_name = L.repo\_name

GROUP BY L.license

ORDER BY number\_of\_files DESC

"""

*# 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, job\_config=safe\_config)

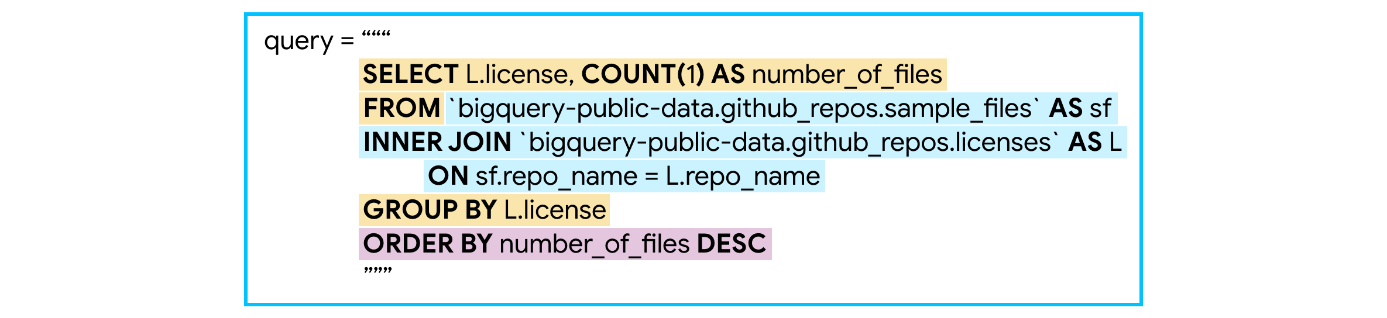
*# API request - run the query, and convert the results to a pandas DataFrame*

file\_count\_by\_license = query\_job.to\_dataframe()

/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 "

Bu büyük bir soru, bu yüzden her parçayı ayrı ayrı inceleyeceğiz.



Yukarıda maviyle vurgulanan **JOIN** ile başlayacağız. Bu, veri kaynaklarını ve bunların nasıl birleştirileceğini belirtir. Tabloları, **repo\_name** sütunlarındaki değerleri eşleştirerek birleştirdiğimizi belirtmek için **ON** ifadesini kullanıyoruz.

Sırada, sarıyla vurgulanan **SELECT** ve **GROUP BY** var. **GROUP BY**, her lisans için veriyi farklı bir gruba ayırır. Ardından, her bir lisansa karşılık gelen **sample\_files** tablosundaki satır sayısını **COUNT** ile sayarız. (Satır sayısını **COUNT(1)** ile sayabileceğinizi unutmayın.)

Son olarak, morla vurgulanan **ORDER BY**, sonuçları daha fazla dosyası olan lisansların önce görüneceği şekilde sıralar.

Sorgu uzundu, ancak bize her bir lisans altında kaç dosyanın işlendiğini özetleyen güzel bir tablo verdi:

*# Print the DataFrame*

file\_count\_by\_license

| license | number\_of\_files |
| --- | --- |
| 0 | mit | 20560894 |
| 1 | gpl-2.0 | 16608922 |
| 2 | apache-2.0 | 7201141 |
| 3 | gpl-3.0 | 5107676 |
| 4 | bsd-3-clause | 3465437 |
| 5 | agpl-3.0 | 1372100 |
| 6 | lgpl-2.1 | 799664 |
| 7 | bsd-2-clause | 692357 |
| 8 | lgpl-3.0 | 582277 |
| 9 | mpl-2.0 | 457000 |
| 10 | cc0-1.0 | 449149 |
| 11 | epl-1.0 | 322255 |
| 12 | unlicense | 208602 |
| 13 | artistic-2.0 | 147391 |
| 14 | isc | 118332 |

JOIN ifadelerini çok kullanacaksınız ve biraz pratik yaptıkça bunları kullanmakta çok daha verimli olacaksınız.