Advanced SQL minicourse from Kaggle

(<https://www.kaggle.com/learn/advanced-sql>)

1. Inner join and Union
2. Inner join

|  |  |
| --- | --- |
| ID | Symbol |
| A | 1 |
| B | 2 |

|  |  |
| --- | --- |
| ID | Name |
| A | Hello |
| B | I |
| C | am |
| D | Vinh |

|  |  |  |
| --- | --- | --- |
| ID | Symbol | Name |
| A | 1 | Hello |
| B | 2 | I |

------->

1. Union

|  |  |
| --- | --- |
| ID | Symbol |
| A | 1 |
| B | 2 |
| C | 3 |

|  |  |
| --- | --- |
| ID | Symbol |
| A | 1 |
| B | 2 |
| C | 3 |
| D | 4 |
| E | 3 |

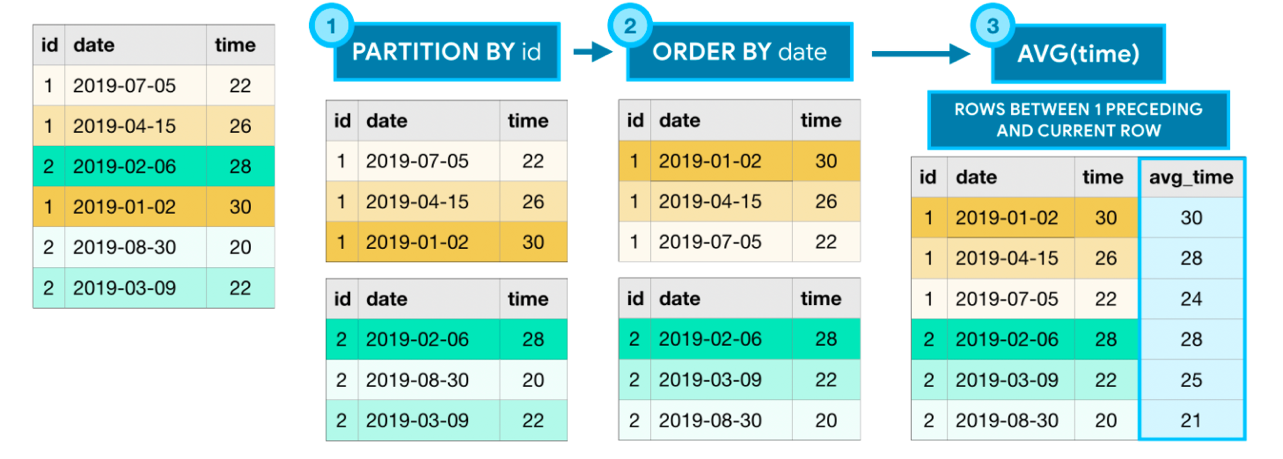
|  |  |
| --- | --- |
| ID | Symbol |
| D | 4 |
| E | 3 |

-------->

1. Analytic function

Syntax:





# Window frame clauses

There are many ways to write window frame clauses:

* 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.

Of course, this is not an exhaustive list, and you can imagine that there are many more options!

# Three types of analytic functions

The example above uses only one of many analytic functions. BigQuery supports a wide variety of analytic functions, and we'll explore a few here. For a complete listing, you can take a look at the [documentation](https://cloud.google.com/bigquery/docs/reference/standard-sql/analytic-function-concepts).

### 1) Analytic aggregate functions

As you might recall, **AVG()** (from the example above) is an aggregate function. The **OVER** clause is what ensures that it's treated as an analytic (aggregate) function. **Aggregate functions** take all of the values within the window as input and return a single value.

* **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

**Navigation functions** assign a value based on the value in a (usually) different row than the current row.

* **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

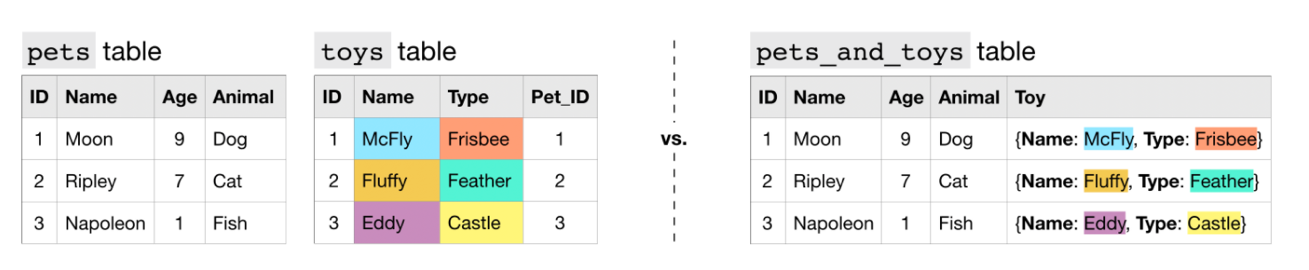
### 3) Analytic numbering functions

**Numbering functions** assign integer values to each row based on the ordering.

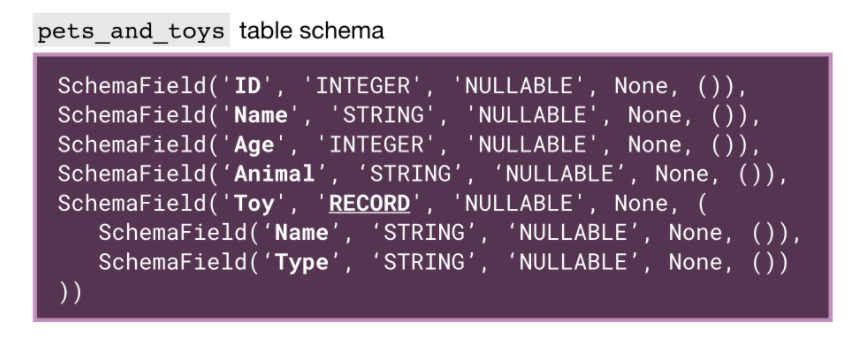
* **ROW\_NUMBER()** - Returns the order in which rows appear in the input (starting with 1)
* **RANK()** - All rows with the same value in the ordering column receive the same rank value, where the next row receives a rank value which increments by the number of rows with the previous rank value.

1. Nested and repeated data
   * 1. Nested:

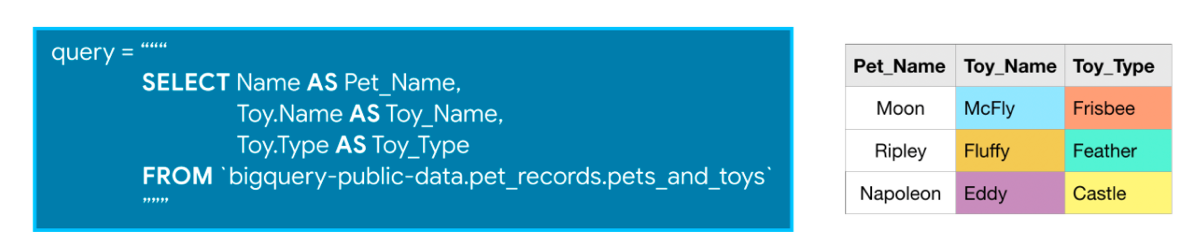
* Nested columns have type **STRUCT** (or type **RECORD**). The column Toy is nested data.



The schema:

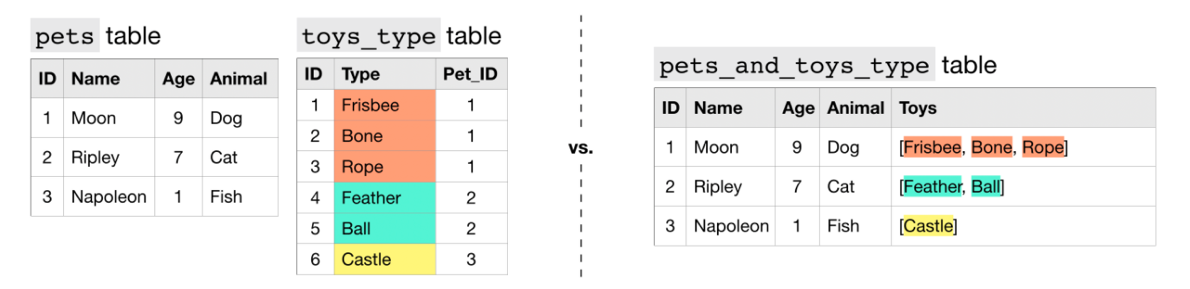


Query: access with “.” operator.

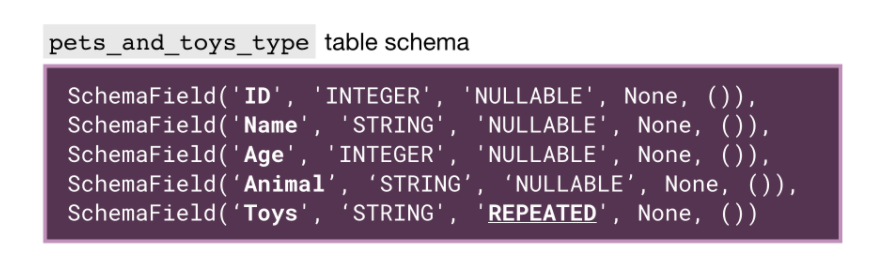


1. Repeated:

Toys have the same ID is group.



The schema:

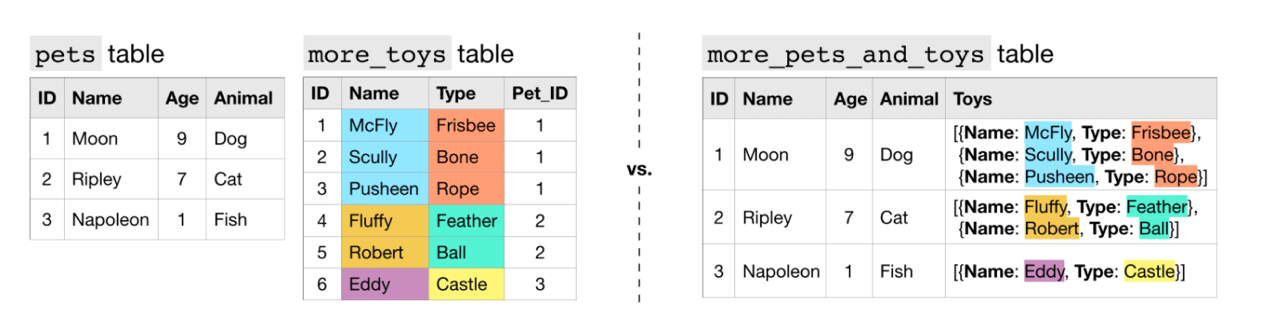


Use UNNEST function to access:

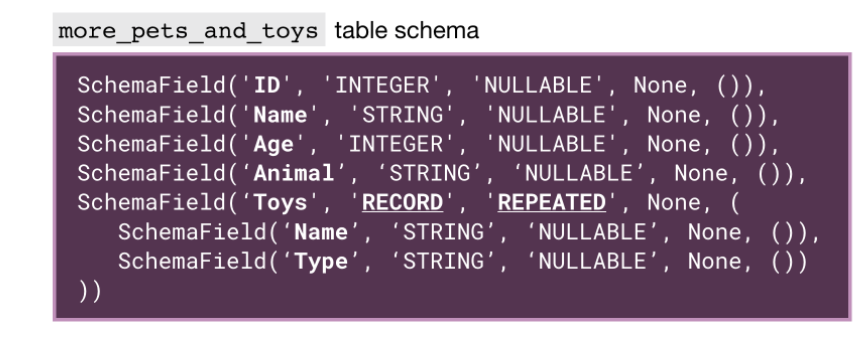


1. Nested and repeated data

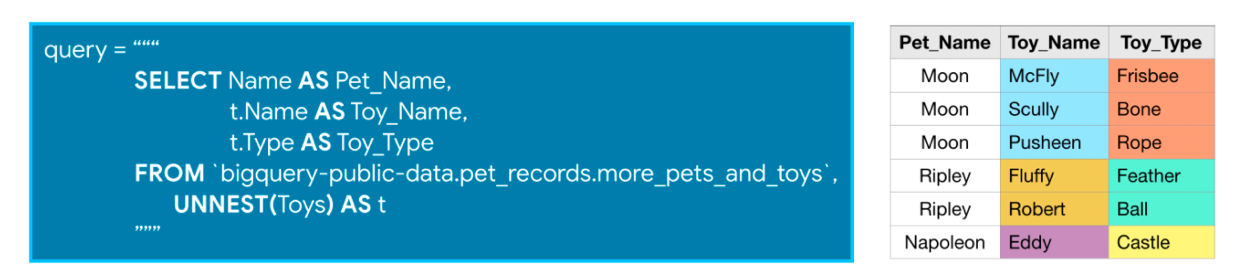
The column Toys of more\_pets\_and\_toys have nested and repeated data:



The schema:



Query:

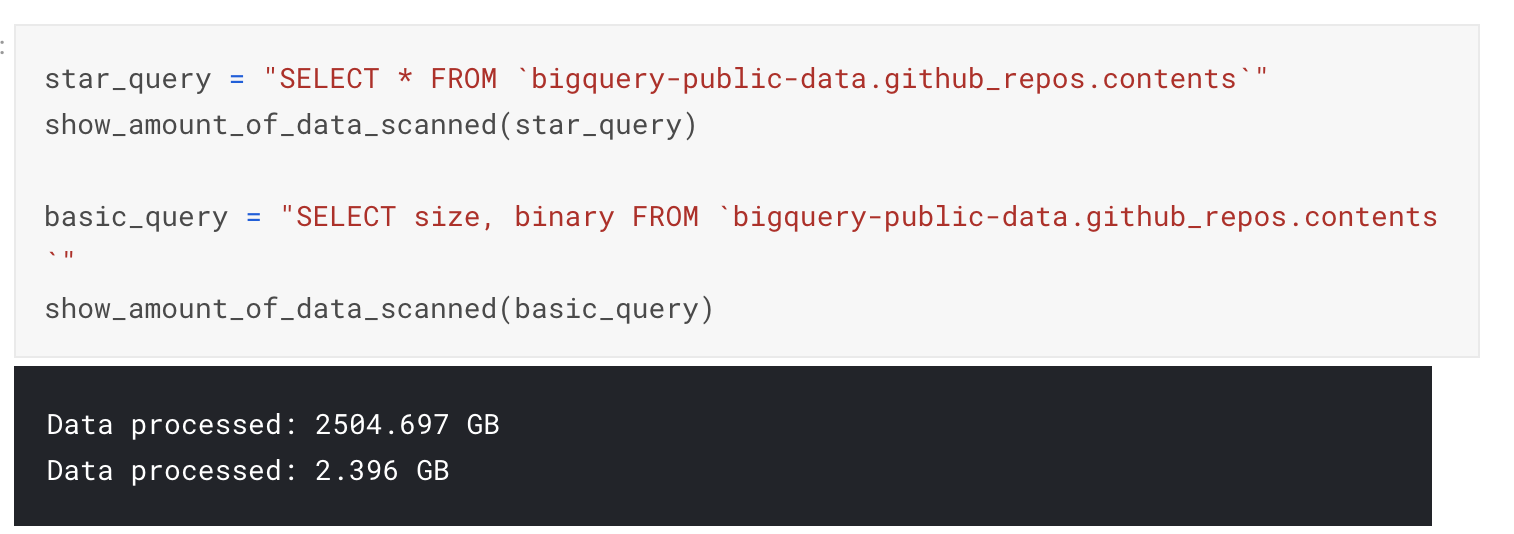


1. Writing efficient queries

Some useful functions

We will use two functions to compare the efficiency of different queries:

* 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.



Strategies:

1. Only select the column you want.
2. Read less data. Query from small table to big table.
3. Avoid N:N join