

# qStudio/PRQL Quick Start #4 - More Transforms

PRQL has more *transforms* (statements) that modify a table as it passes through its pipeline. *Much of the discussion below comes from the [PRQL Reference Tutorial](#)*

We have already introduced several PRQL transforms. Let's review what each does and how each changes the "shape" of the table:

- **from** - begins a pipeline and passes the entire table to the next transform.
- **select** - Changes the number of columns by retaining only those named within the tuple (the list of column names inside the `{ ... }`) but never the number of rows.
- **filter** - Changes the number of rows by excluding the ones that don't match the criteria. Never changes the number of columns.
- **derive** - Changes the number of columns by adding a new column, calculated from other columns in the row. Never changes the number of rows.
- **sort** - Changes the *order* of the rows, but leaves the number of rows and columns unchanged.

This lesson talks about these new transforms.

## take

The `take` transform picks rows to pass through based on their position within the table. The number of columns is unchanged. The set of rows picked can be specified in two ways:

- a plain number `x`, which will pick the first `x` rows, or
- an inclusive range of rows `start..end`

For example:

```
1 | from invoices
2 | take 4           # takes the first four rows
3 |
4 | # - or -
5 |
6 | from invoices
7 | take 4..7       # takes rows 4, 5, 6, and 7
```

## aggregate

---

The `aggregate` transform takes a tuple (a list of column names) and “distills down” data from all the rows into a single row. This is frequently used for statistical analysis.

```
1 | from invoices
2 | aggregate { sum_of_orders = sum total }
```

In the query above, the "invoices" table has a column named `total` (perhaps it's the total of a single order). It sums all the values in the `total` column to produce a single row.

The number of columns is equal to the number of items within the tuple. In the example above, the result would be *one* column. In the example below, the resulting table has *two* columns - `sum_of_orders` and `avg_of_orders`

```
1 | from invoices
2 | aggregate {
3 |     sum_of_orders = sum total,
4 |     avg_of_orders = average total,
5 | }
```

## group

---

The `group` transform performs a set of operations on "groups" of rows based on some characteristic. You might use `group` to analyze data by city or some other value.

The `group` transform retains all the columns: the number of rows is equal to the number of different combinations within the group characteristic (say, the number of different cities).

[See the "grouping" section of the PRQL docs for more information.](#)

## join

---

The `join` transform combines the columns of two tables together "side-by-side" based on related columns of each table. It is often useful to "join" two separate tables that each have columns with interesting data using common parameters, and then use `select` to extract the interesting columns.

The `join` transform adds columns to the result (it contains *all* the columns from both tables). The number of

rows varies based on the data within the tables and the operators used to join the tables.

[See the "join" section of the PRQL docs for more information.](#)