FeatureBase
DOCUMENTATION

START FREE TRIAL

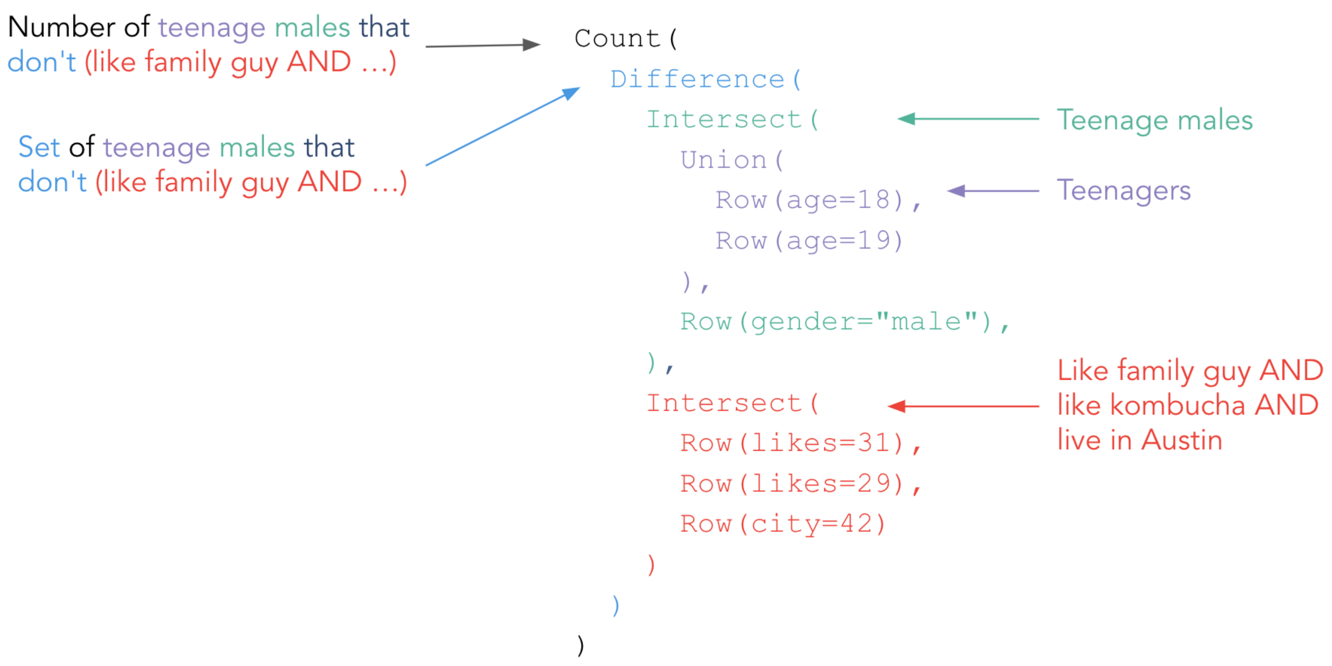
Pilosa Query Language (PQL) is FeatureBase's native query language.

Like SQL, PQL is a declarative, set-based language. Unlike SQL, PQL expressions are built by composing function calls, using the familiar parentheses call syntax. With this function-compositional syntax, PQL functions operate on bitmaps as the atomic elements of the data store, expressing boolean operations of arbitrary complexity. PQL also provides a number of functions outside of this paradigm, which implement common data tasks like aggregation.

Many PQL calls return or accept objects representing rows (row calls) or columns (rows calls), and understanding these types is key to composing the various types of PQL call to build a query.

For example, the row call is the basic query type for reading bitmaps and performing boolean operations on them. The input type signatures of row calls vary, but they always return an object representing a set contained in a single row (i.e. a *computed row*, not necessarily a *stored row*), which can be passed as an argument to other queries.

For a simple example, consider this diagram:



```
Count(
  Difference(
    Intersect(
      Union(
        Row(age=18),
        Row(age=19)
      ),
      Row(gender="male"),
    ),
    Intersect(
      Row(likes=31),
      Row(likes=29),
      Row(city=42)
    )
  )
)
```

Number of teenage males that don't (like family guy AND ...)
Set of teenage males that don't (like family guy AND ...)
Teenage males
Teenagers
Like family guy AND like kombucha AND live in Austin

This is a PQL query on a table of users which includes fields for gender, age, location, and interests. It asks a question involving a moderately complex boolean combination of these fields. Note that each annotated component within the `Count` is a separate row call, representing a set of users; each of these could be a standalone query, or passed to a `Count` call, for example. `Count` accepts a single row call, regardless of its structure, and returns a scalar value.

While other calls may accept different types, this composition of function calls is an important concept for PQL usage.

A Note About Terminology

For historical reasons, FeatureBase's concepts of "row" and "column" are not conventional, and may be confusing.

In PQL:

- An *index* is a single “universe” of data, with a shared column space, and a group of related fields. The term *table* may be used interchangeably, as well as the deprecated term *VDS* (Virtual Data Source).
- A *column* represents an instance of the entity of concern for a given table, such as a user or an event. Columns are common to all fields and rows within an index. Because this concept might be referred to as a row in other data stores, to reduce ambiguity, a column may be referred to as a *record*.
- A *row* represents a single, binary characteristic that may be associated with any record. A row maintains a set of bits — a bitmap — indicating which records have a bit Set for its attribute. To reduce ambiguity, a row may be referred to as a field value. Note that this is conceptually different from the use of “column” in other data stores, in that each row stores membership information per value. While this is mainly an implementation detail in normal FeatureBase usage, it can be helpful in understanding the underlying data model.
- A *field* is a grouping of rows, for example, to combine several single rows into a non-boolean categorical value, or an integer value. From a user’s perspective, this term aligns well with usage by other data stores.

The names of the *Row Calls* and *Rows Calls* types, as well as calls like `Row` and `Rows` are all based on these definitions.

Because of the nested structure of the language, subexpressions in a query are often meaningful queries on their own, and terms like “query”, “call”, “keyword”, and “function” may be used somewhat interchangeably. Here, *query* is used to refer to a conceptually complete, runnable query, and *call* refers to the individual named PQL functions from which queries are composed.

In some contexts, particularly FeatureBase's `/index/{index}/query` HTTP endpoint, a PQL *query request* may consist of any number of disjoint *queries* (also known as top-level calls), optionally separated by whitespace. These queries are executed serially, and their results are returned in order.

Query Types

PQL queries can be broadly grouped into:

- Read operations, which comprise the majority of PQL keywords, and often the majority of PQL usage.
- Write operations, some of which are often avoided in favor of using ingesters.
- Other operations

Read operations can be further grouped into several types. All of these types are summarized below.

Row Calls

Row calls return an object representing a set of column keys, contained in a single row. Row calls may be used as input arguments to other queries.

Row calls include:

- Row selection
  - All() returns the *universal set* for the index – i.e. it returns all the record IDs or record keys in the index.
  - ConstRow() provides a "literal" bitmap value – i.e. it returns the record IDs or record keys specified by the user.

    - Row() selects from a single set row, by row key – i.e. it returns the record IDs or record keys that have a specified value in a specified field.
    - Limit() is also a row call by return type. It wraps other row calls to select a subset of results, in a pagination sense.
- Boolean operations
    - Difference() computes the set difference between its first argument and all subsequent arguments (all row calls).
    - Intersect() computes the set intersection across its two or more row call arguments.
    - Not() computes the complement of the single row call argument, relative to the *universal set* for the index – i.e. it returns the difference between `All()` and some other row call.
    - Union() computes the set union across its one or more row call arguments.
    - UnionRows() computes the set union across many rows. Rather than accepting several row call arguments, `UnionRows()` accepts any number of `Rows` arguments.
    - Xor() computes the exclusive set difference between its first argument and all subsequent arguments (all row calls).

Distinct is a special row-like call, in that it can be used in the same context that other row calls can be used, despite a slight variation in its output type.

Rows Calls

Rows calls return an object representing a set of row keys, contained in a single column. Rows calls may be used as input arguments to other queries.

Rows calls include:

- Rows returns a list of row IDs in the given field which have at least one bit set – i.e. it returns a list of value in a field for all values associated with a record.

Membership Calls

Membership calls return a boolean value indicating set membership.

Membership calls include:

- IncludesColumn indicates whether a given record ID or record key is present in a given row call.

Count Calls

Count calls return counts of the number of elements in *groups of sets*. Count calls are often the top-level call in a query.

Count calls include:

- Count computes the scalar count of elements contained in its single row call argument.
- GroupBy computes counts of the intersection of every combination of its rows call arguments. GroupBy provides much of the basic grouping functionality that is available in a relational database.
- TopK computes the count of the top `K` rows in a field, with fewer caveats.
- TopN computes the count of the top `N` rows in a field, with some caveats.

Aggregation Calls

Aggregation calls perform other aggregation operations on *individual sets*.

Aggregation calls include:

- Max computes the maximum integer value or timestamp value in a field, from a single row call argument.
- Min computes the minimum integer value or timestamp value in a field, from a single row call argument.
- Percentile computes the percentile of integer values in a field, from a single row call argument.
- Sum computes the sum of integer values in a field, from a single row call argument.

Exploratory Calls

Exploratory calls are used to drill down into a data set.

- Extract is analogous to a general `select` query in a relational database, returning a subset of both rows and columns.

DataFrame Calls

- Apply executes Ivy code against data stored using the `float64` and / or `int64` data type.
- Arrow is analogous to `Extract()` and a `SELECT <columns> FROM <table> WHERE <condition>` query in SQL. Executes on data stored using the `float64` and / or `int64` data type.

Write Operations

- Clear sets a single specified bit to zero – said another way, it removes a value from a field for a specified record.

- ClearRow sets all bits to zero in the specified row – said another way, it removes a value from a field for all records in an index.
- Delete iterates over all fields and views in a set of provided columns, removing the columns – i.e. removes an entire record or set of records from an index.
- Set sets a single specified bit to one – said another way, it gives a specified record a value in a field.
- Store writes the results of the row call argument to the specified row.

Other operations

- Options modifies the execution context and return types of the queries passed to it.

Something missing or incorrect?

Help improve this article or join us on Discord!