Queries

Queries are the output of a QL program. They evaluate to sets of results.

There are two kinds of queries. For a given query module, the queries in that module are:

- The select clause, if any, defined in that module.
- Any query predicates in that module's predicate namespace. That is, they can be defined in the module itself, or imported from a different module.

We often also refer to the whole QL program as a query.

Select clauses

When writing a query module, you can include a **select clause** (usually at the end of the file) of the following form:

```
from /* ... variable declarations ... */
where /* ... logical formula ... */
select /* ... expressions ... */
```

The `from` and `where` parts are optional.

Apart from the expressions described in "Expressions," you can also include:

- The `as` keyword, followed by a name. This gives a "label" to a column of results, and allows you to use them in subsequent select expressions.
- The `order by` keywords, followed by the name of a result column, and optionally the keyword `asc` or `desc`. This determines the order in which to display the results.

For example:

```
from int x, int y
where x = 3 and y in [0 .. 2]
select x, y, x * y as product, "product: " + product
```

This select clause returns the following results:

x	y	product	
3	0	0	product: 0
3	1	3	product: 3
3	2	6	product: 6

You could also add `order by y desc` at the end of the select clause. Now the results are ordered according to the values in the `y` column, in descending order:

x	y	product	
3	2	6	product: 6
3	1	3	product: 3
3	0	0	product: 0

Query predicates

A query predicate is a non-member predicate with a `query` annotation. It returns all the tuples that the predicate evaluates to.

For example:

```
query int getProduct(int x, int y) {
  x = 3 and
  y in [0 .. 2] and
  result = x * y
}
```

This predicate returns the following results:

x	y	result
3	0	0
3	1	3
3	2	6

A benefit of writing a query predicate instead of a select clause is that you can call the predicate in other parts of the code too. For example, you can call `getProduct` inside the body of a class:

```
class MultipleOfThree extends int {
  MultipleOfThree() { this = getProduct(_, _) }
}
```

In contrast, the select clause is like an anonymous predicate, so you can't call it later.

It can also be helpful to add a `query` annotation to a predicate while you debug code. That way you can explicitly see the set of tuples that the predicate evaluates to.