About the QL language ¶

QL is the powerful query language that underlies CodeQL, which is used to analyze code.

About query languages and databases

QL is a declarative, object-oriented query language that is optimized to enable efficient analysis of hierarchical data structures, in particular, databases representing software artifacts.

A database is an organized collection of data. The most commonly used database model is a relational model which stores data in tables and SQL (Structured Query Language) is the most commonly used query language for relational databases.

The purpose of a query language is to provide a programming platform where you can ask questions about information stored in a database. A database management system manages the storage and administration of data and provides the querying mechanism. A query typically refers to the relevant database entities and specifies various conditions (called predicates) that must be satisfied by the results. Query evaluation involves checking these predicates and generating the results. Some of the desirable properties of a good query language and its implementation include:

- Declarative specifications - a declarative specification describes properties that the result must satisfy, rather than providing the procedure to compute the result. In the context of database query languages, declarative specifications abstract away the details of the underlying database management system and query processing techniques. This greatly simplifies query writing.
- Expressiveness - a powerful query language allows you to write complex queries. This makes the language widely applicable.
- Efficient execution - queries can be complex and databases can be very large, so it is crucial for a query language implementation to process and execute queries efficiently.

Properties of QL

The syntax of QL is similar to SQL, but the semantics of QL are based on Datalog, a declarative logic programming language often used as a query language. This makes QL primarily a logic language, and all operations in QL are logical operations. Furthermore, QL inherits recursive predicates from Datalog, and adds support for aggregates, making even complex queries concise and simple. For example, consider a database containing parent-child relationships for people. If we want to find the number of descendants of a person, typically we would:

1. Find a descendant of the given person, that is, a child or a descendant of a child.
2. Count the number of descendants found using the previous step.

When you write this process in QL, it closely resembles the above structure. Notice that we used recursion to find all descendants of the given person, and an aggregate to count the number of descendants. Translating these steps into the final query without adding any procedural details is possible due to the declarative nature of the language. The QL code would look something like this:

```
Person getADescendant(Person p) {
  result = p.getAChild() or
  result = getADescendant(p.getAChild())
}

int getNumberOfDescendants(Person p) {
  result = count(getADescendant(p))
}
```

For more information about the important concepts and syntactic constructs of QL, see the individual reference topics such as "Expressions" and "Recursion." The explanations and examples help you understand how the language works, and how to write more advanced QL code.

For a formal specification of the QL language, see the "QL language specification."

QL and object orientation

Object orientation is an important feature of QL. The benefits of object orientation are well known – it increases modularity, enables information hiding, and allows code reuse. QL offers all these benefits without compromising on its logical foundation. This is achieved by defining a simple object model where classes are modeled as predicates and inheritance as implication. The libraries made available for all supported languages make extensive use of classes and inheritance.

QL and general purpose programming languages

Here are a few prominent conceptual and functional differences between general purpose programming languages and QL:

- QL does not have any imperative features such as assignments to variables or file system operations.
- QL operates on sets of tuples and a query can be viewed as a complex sequence of set operations that defines the result of the query.
- QL's set-based semantics makes it very natural to process collections of values without having to worry about efficiently storing, indexing and traversing them.
- In object oriented programming languages, instantiating a class involves creating an object by allocating physical memory to hold the state of that instance of the class. In QL, classes are just logical properties describing sets of already existing values.

Further reading

Academic references also provide an overview of QL and its semantics. Other useful references on database query languages and Datalog:

- Database theory: Query languages

- Logic Programming and Databases book
- Foundations of Databases
- Datalog