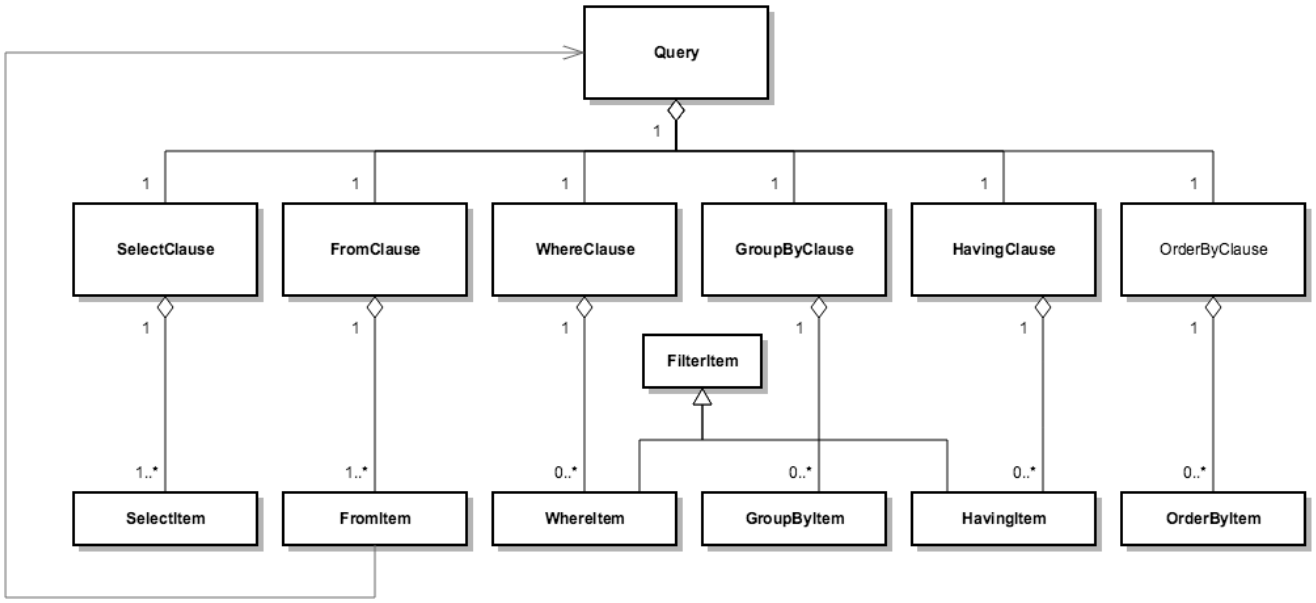
Query object, the QueryBuilder API and Query parsing

Apache MetaModel offers three different ways of defining a query for a DataContext to retrieve data. Each approach offers a set of features, making them appropriate for different situations.

1 - The `Query` object

At the core of all query execution in MetaModel is the `Query` object. `Query` is a regular Java class containing the structural elements that make up a query. The main elements of a query is a set of clauses (`SelectClause`, `FromClause` and so on) which in turn each contain a set of items (`SelectItem`, `FromItem` etc.).

The object structure is depicted in the below class diagram. Each item will typically refer to some structure in the `DataContext`'s schema model (a `Table`, a `Column` or such), or even an embedded sub-query. A query must have at least 1 `SelectItem` and 1 `FromItem` for it to be executable.



As such, you can instantiate `Query` and populate the object yourself:

```
DataContext dc = ...
Table table = dc.getDefaultSchema().getTableByName("person");
Column col1 = table.getColumnByName("id");
Column col2 = table.getColumnByName("name");
Column col3 = table.getColumnByName("age");

// Construct a query like "SELECT id, name FROM person WHERE age > 18"
Query q = new Query();
q.from(table);
q.select(col1, col2);
q.where(col3, OperatorType.GREATER_THAN, 18);

// Execute the Query
DataSet ds = dc.executeQuery(q);
```

Pros:

- Building your query this way gives you a lot of control. It allows you to effectively separate the static and dynamic parts of a query and grow awareness of any assumptions/hardcoded values etc.
- One of the main advantages of having the query as an object is that the building of the query lends itself to advanced construction logic, potentially spanning multiple components to aid in the building of a particular query. Therefore, using the Query object type is the most powerful of the three approaches to querying in MetaModel.
- Having the query available as an object also allows you to unittest / assert the structure of the queries you are creating.

Cons:

- Creating the query this way is a bit verbose, and as we shall see below, it can be made simpler in most cases.

2 - The Query-builder API

To make query definition easier and more developer-guided, Apache MetaModel also has the so-called Query-builder API. This term is used for the set of methods extending from a call to `DataContext.query()`, ending with either `.toQuery()` or `.execute()`. Using the same simple example as before, we can define and execute the query like this:

```
DataContext dc = ...
DataSet ds = dc.query().from("person").select("id", "name").where("age").gt(18).execute();
```

Pros:

- The API is easier to read than using the `Query` class directly.
- The API helps select which methods are meaningful in the given context, eliminating choices to what makes sense.
- The API does not require you to hold a `Query` object if you wish to immediately just execute it.

Cons:

- Some features are not available via this API, for instance sub-querying.

3 - Query parsing

Finally, Apache MetaModel allows you to simply parse a query from a string form. You can use this feature in a two-step *parse-then-execute* mode or in a single step to *parse-and-execute*.

- `DataContext.`**`executeQuery`**`(String): `**`DataSet`**
- `DataContext.`**`parseQuery`**`(String): `**`Query`**

The parseQuery method allows you to interrogate or even manipulate the parsed query, potentially leveraging the control and power of the `Query` object approach in combination with parsing.

Note that unlike the query builder API, the sequence of query clauses in a string query is like that of SQL (select before from):

> *SELECT ... FROM ... [WHERE ...] [GROUP BY ...] [HAVING ...] [ORDER BY ...]*

Pros:

- This approach is often the most readable way of defining a query.
- This approach allows for easy externalization of queries as strings, or of having the users type their own queries.

Cons:

- Using string literals for queries is un-safe and may introduce bugs in your code due to the non-typed and non-checked nature of the query.
- The fact that string SQL-like queries are supported may be misleading some users to think that the query is passed un-interpreted to the backing datastore. This is NOT the case.