Join Support

Contact:	Justin Deoliveira
Tracker:	http://jira.codehaus.org/browse/GEOT-3657
Tagline:	Adding joins to query api

Children:

Description

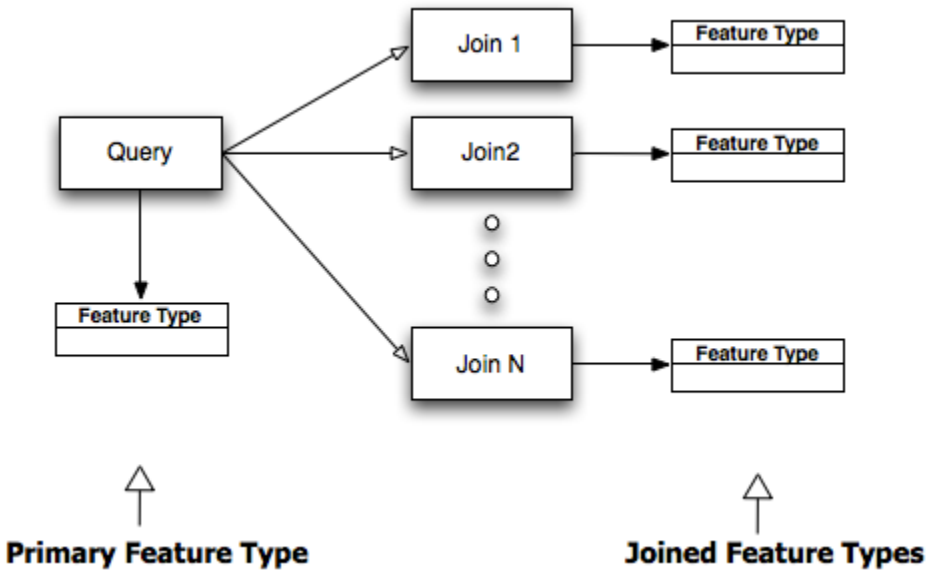
Joins are a powerful construct for analyzing relationships among datasets. In the geo domain it gets even more interesting because you can start to join on spatial relationships like contains and intersects. The semantics of join in this context is that of an SQL join.

The current Query interface only supports queries against a single dataset or feature type. The latest version of WFS, 2.0, adds support for joins to the WFS protocol. The GeoTools query api historically has been driven in part by requirements of WFS. Joins are the latest addition.

This proposal is to enhance the existing Query api to support joins.

Join Model

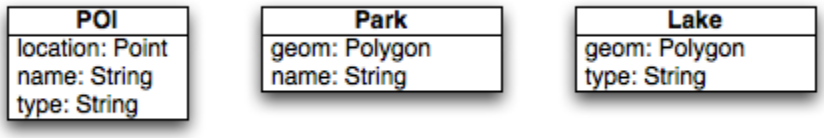
A join query consists of a single query object, like in a non join query, and multiple join objects. A join object exists for each additional feature type in the join, referred to here as a "joined feature type". The query refers to the "primary feature type".



Each join specifies the condition on which to join the primary feature type to each joined feature type. This condition is specified by a "join filter". This filter can be a standard binary comparison operator like `PropertyIsEqualTo`, or a spatial operator like `Intersects` or `Contains`. Or a combination specified with a binary logic operator like `And` or `Or`.

Most often a comparison filter is composed of a `PropertyName` specifying a feature type attribute, and a `Literal` or `Function` specifying a value to compare the attribute to. However for a join filter the comparison must consist of two `PropertyName` instances. One `PropertyName` ref erences the primary feature type, and the other references the joined feature type.

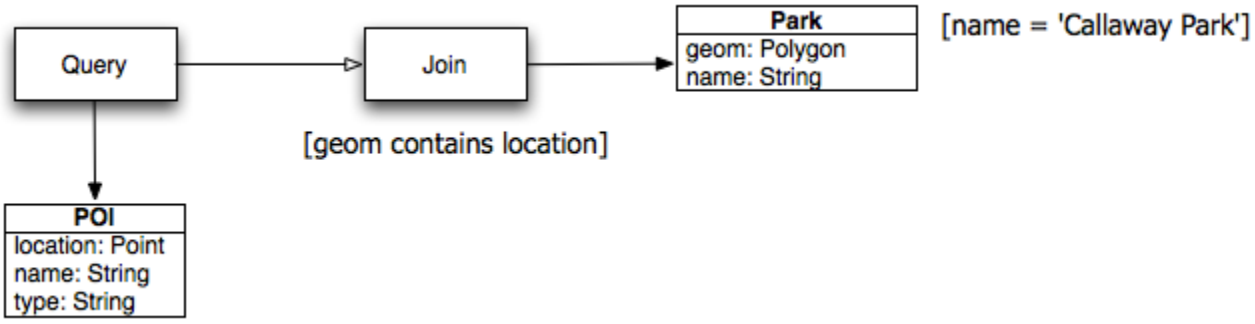
To illustrate further with an example consider the following feature types:



We want to perform a query that tells us what points of interest (poi's) are located within our favourite park. In SQL this would be accomplished with a SQL statement like:

```
SELECT a.name, a.type FROM POI a, Park b
 WHERE st_contains(b.geom, a.location)
   AND b.name = 'Callaway Park'
```

In our query/join model the above looks like:



In code the above would look like:

```
FilterFactory ff = ...;

Query query = new Query("POI");

Join join = new Join("Park", ff.contains(ff.property("geom"),
ff.property("location")));
join.setFilter(ff.equal(ff.property("name"), ff.literal("Callaway Park")));
query.getJoins().add(join);
```

The above code snippet creates two filters. The first is the join filter that specifies the join condition, in this case containment specifying that we want poi's that are contained in a park. The second filter is a regular filter set on the join itself and specifies that we only want to consider a particular park. This second filter is identical in nature to `Query.getFilter` except for that it works against the joined feature type, and not the primary feature type.

Let's consider a more complex query. Let's say we want to find a park that has a lake in it, but does not contain any tourist attractions. The following code implements this query with two joins.

```
Query query = new Query("Park");

//lake join
Join join1 = new Join("Lake", ff.contains(ff.property("geom"), ff.property("l.geom"));
join1.setAlias("l");
query.getJoins().add(join1);

//poi join
Join join2 = new Join("POI", ff.disjoint(ff.property("geom"),
ff.property("location")));
join2.setFilter(ff.notEquals(ff.property("type"), ff.literal("tourism"));
query.getJoins().add(join2);
```

The above join illustrates the use of an "alias". Both the Park and Lake feature types contain an attribute named "geom". That makes any comparison between the two ambiguous. Therefore we must use an alias for the joined feature type and use it to qualify the property name in the join filter.

Returned Feature Schema

The response to a join query are features with the following structure. The feature contains all the selected attributes from the primary feature types, and additional attributes for each joined feature type. The name of a joined feature type attribute is the joined feature type name, or if an alias is used the attribute name is the same as the alias.

For example, in the first query above returned features would have the following schema:

```
POI(location:Point, name:String, type: String, Park: SimpleFeature)
```

The second query returns:

```
Park(geom:Polygon, name:String, l:SimpleFeature, POI:SimpleFeature)
```

Status

This proposal is completed.

Voting has not started yet:

- Andrea Aime: +1
- Ben Caradoc-Davies
- Christian Mueller: +1
- Ian Turton
- Justin Deoliveira: +1
- Jody Garnett: +1
- Simone Giannecchini

Tasks

This section is used to make sure your proposal is complete (did you remember documentation?) and has enough paid or volunteer time lined up to be a success

	no progress		done		impeded		lack mandate/funds/time		volunteer needed

1. API changed based on BEFORE / AFTER
2. Update default implementation
3. Update wiki (both module matrix and upgrade to to 2.5 pages) |
4. Remove deprecated code from GeoTools project
5. Update the user guide
6. Update or provided sample code in demo
7. review user documentation

API Changes

Join

A new Join class is added.

Query

The `Query` class is modified with a list of `Join`.

```
public class Query {

  ...

  List<Join> getJoins();

}
```

QueryCapabilities

The `QueryCapabilities` class is modified with a new method `isJoiningSupports()`.

```
public class QueryCapabilities {

  ...

  boolean isJoiningSupported();

}
```

Documentation Changes

list the pages effected by this proposal

- gt-api query examples updated to reflect api change