Item 51
git_comments:

1. * * Creates {@link net.hydromatic.optiq.ScalarFunction} for each method in a * given class.
2. * Use {@link org.eigenbase.sql.type.InferTypes#explicit(java.util.List)}.
3. Another method
4. * Tests user-defined function, with multiple methods per class.
5. java.lang.Math has abs(int) and abs(double).
6. Three overloads
7. * UDF class that has multiple methods, some overloaded.
8. **comment:** Non-static method cannot be used because constructor is private
   label: code-design
9. 3 overloads of "fun1", another method "fun2", but method "nonStatic" cannot be used as a function

git_commits:

1. **summary:** [OPTIQ-314] Allow simple UDFs based on methods
   message: [OPTIQ-314] Allow simple UDFs based on methods

github_issues:

github_issues_comments:

github_pulls:

github_pulls_comments:

github_pulls_reviews:

jira_issues:

1. **summary:** Allow simple UDFs based on methods
   description: We already allow UDFs based on classes, provided that that classes have just one method. This change would allow UDFs based on classes with several methods. E.g. should be able to use 'String String.replace(String s, String s2)' as a SQL UDF 'REPLACE(String, String, String) RETURNS String'.
2. **summary:** Allow simple UDFs based on methods
   description: We already allow UDFs based on classes, provided that that classes have just one method. This change would allow UDFs based on classes with several methods. E.g. should be able to use 'String String.replace(String s, String s2)' as a SQL UDF 'REPLACE(String, String, String) RETURNS String'.
3. **summary:** Allow simple UDFs based on methods
   description: We already allow UDFs based on classes, provided that that classes have just one method. This change would allow UDFs based on classes with several methods. E.g. should be able to use 'String String.replace(String s, String s2)' as a SQL UDF 'REPLACE(String, String, String) RETURNS String'.
4. **summary:** Allow simple UDFs based on methods
   description: We already allow UDFs based on classes, provided that that classes have just one method. This change would allow UDFs based on classes with several methods. E.g. should be able to use 'String String.replace(String s, String s2)' as a SQL UDF 'REPLACE(String, String, String) RETURNS String'.
   label: code-design
5. **summary:** Allow simple UDFs based on methods
   description: We already allow UDFs based on classes, provided that that classes have just one method. This change would allow UDFs based on classes with several methods. E.g. should be able to use 'String String.replace(String s, String s2)' as a SQL UDF 'REPLACE(String, String, String) RETURNS String'.
   label: code-design
6. **summary:** Allow simple UDFs based on methods
   description: We already allow UDFs based on classes, provided that that classes have just one method. This change would allow UDFs based on classes with several methods. E.g. should be able to use 'String String.replace(String s, String s2)' as a SQL UDF 'REPLACE(String, String, String) RETURNS String'.
7. **summary:** Allow simple UDFs based on methods
   description: We already allow UDFs based on classes, provided that that classes have just one method. This change would allow UDFs based on classes with several methods. E.g. should be able to use 'String String.replace(String s, String s2)' as a SQL UDF 'REPLACE(String, String, String) RETURNS String'.
8. **summary:** Allow simple UDFs based on methods
   description: We already allow UDFs based on classes, provided that that classes have just one method. This change would allow UDFs based on classes with several methods. E.g. should be able to use 'String String.replace(String s, String s2)' as a SQL UDF 'REPLACE(String, String, String) RETURNS String'.
9. **summary:** Allow simple UDFs based on methods
   description: We already allow UDFs based on classes, provided that that classes have just one method. This change would allow UDFs based on classes with several methods. E.g. should be able to use 'String String.replace(String s, String s2)' as a SQL UDF 'REPLACE(String, String, String) RETURNS String'.

jira_issues_comments:

1. Are you sure it is not available now? [public static ScalarFunction ScalarFunctionImpl.create(Method method)|https://github.com/julianhyde/optiq/blob/master/core/src/main/java/net/hydromatic/optiq/impl/ScalarFunctionImpl.java#L68] should do the trick.
2. Right now the method has to be called "eval". We'd need to add a "String methodName" member to JsonFunction to allow stuff like {code:javascript} functions: [{ className: "java.lang.String", methodName: "replace" }] {code} in the schema.
3. What with overloads then? Method name is not sufficient to identify a method. We either pick up all the methods with given name and use them as overloads, or provide a way to specify argument types.
4. **body:** I was thinking of automatic overloads. For example, if they specify {code:javascript} functions: [{ name: "MY_ABS", className: "java.lang.Math", methodName: "abs" }] {code} they will get SQL UDFs {{MY_ABS(INT)}}, {{MY_ABS(LONG)}}, {{MY_ABS(REAL)}}, {{MY_ABS(DOUBLE)}} because java.lang.Math has overloaded functions {{abs(int)}}, {{abs(long)}}, {{abs(float)}}, {{abs(double)}}. if you hoped that the {{public String replace(String, String)}} of {{java.lang.String}} could be used to implement {{REPLACE(VARCHAR, VARCHAR, VARCHAR) RETURNS VARCHAR}}, you are out of luck; it actually becomes {{REPLACE(VARCHAR, VARCHAR) RETURNS VARCHAR)}}. Optiq does not use the "this" as an extra parameter. Optiq will instantiate the class before the first row (the class must have a public default constructor), then uses the same instance (in this case, the String object) for each call. The instance can be used to hold state variables. (Not much use in this case, because String has no mutable fields.)
   label: code-design
5. **body:** Now if you specify {{methodName}}, Optiq finds all methods with that name in the class. (It excludes non-static ones if the class does not have a public zero-args constructor.) Overloads are resolved based on runtime arguments. If you specify {{methodName='*'}} it loads all methods, as above. You don't need to specify {{name}}; the name of the function is based on the name of the method. You cannot use {{methodName}} with aggregate functions. (They require several implementation methods, with fixed names.) As part of this change, I also improved how overloaded argument lists are resolved. In particular, an Object parameter becomes a SQL ANY type and can thus match anything. I also added {{OptiqAssert.AssertQuery.returnsValue}}, to make it easier to test statements that return 1 row with 1 column.
   label: code-design
6. Fixed in http://git-wip-us.apache.org/repos/asf/incubator-optiq/commit/679b31a2.
7. Close issues resolved in release 0.9.0-incubating (2014-08-25).