

qSQL

for Microsoft SQL Sever 2008

v1.0



1. qSQL

1.1 Installation

1.2 Basis

1.2.1 Allgemein

1.2.2 Encoder

1.2.3 Matching

1.2.4 Tokenizer

1.2.5 Erweiterungen

1.3 Verstehen

1.3.1 Allgemein

1.3.2 Daten- und Schemaeigenschaften

1.3.3 Primär- und Fremdschlüssel

1.3.4 Regelinduktion

1.3.5 Erweiterungen

A SQL-Schnittstelle

A.1 Allgemein

A.2 Date

A.3 Distance

A.4 Number

A.5 Text

A.6 Understand

A.7 Improve

A.8 Tools

B Klassenbibliothek

B.1 Allgemein

B.2 Basis

B.2.1 Encoder

B.2.2 Matching

B.2.3 Tokenizer

B.3 Verstehen

B.3.1 Daten- und Schemaeigenschaften

B.3.2 Primär- und Fremdschlüssel

B.3.3 Regelinduktion

B.4 Verbessern

B.4.1 Daten standardisieren, korrigieren, anreichern

B.4.2 Fusion

Lizenzbedingungen für qSQL

Die nachfolgenden Lizenzbedingungen regeln die kostenlose Überlassung der qSQL Software. Diese wird im Folgenden als Software bezeichnet.

1. Umfang der Nutzungsrechte

crQ gewährt Ihnen eine kostenlose Lizenz für die Software. Die Lizenz gibt Ihnen die Berechtigung, die Software auf einem oder mehreren PCs, die Sie privat oder beruflich in branchenüblicher Weise nutzen, zu installieren.

Die Nutzung der Software ist zeitlich unbegrenzt. Die Nutzungsrechte sind weltweit gültig, nicht ausschließlich und nicht übertragbar.

Die Software wird mit den Funktionen und Eigenschaften überlassen, die sie zum Zeitpunkt der Installation enthält.

2. Urheberrecht

crQ hat die alleinigen, ausschließlichen Rechte an der Software. Die Überlassung einer Lizenz gewährt Ihnen allein das Recht zur Nutzung der Software in dem Umfang, der durch diese Lizenzbedingungen eingeräumt wird; alle Urheberrechte verbleiben vollständig bei crQ.

3. Weitere Nutzungsbeschränkungen

Sie dürfen die Software weder verkaufen, vermieten noch verleasen oder verleihen. Allerdings dürfen Sie die überlassenen Lizenzen vollständig und auf Dauer an einen Dritten übertragen, vorausgesetzt, dass der Empfänger sich mit diesen Lizenzbedingungen einverstanden erklärt.

Es ist nicht gestattet, die Software oder ihre Datenstrukturen zurückzuentwickeln.

4. Gewährleistung/Haftung

Dem Lizenznehmer wird die Software im Rahmen dieser Lizenz so wie sie ist, ohne Gewährleistung, zur Verfügung gestellt. Keinesfalls übernimmt der Lizenzgeber und/oder deren Lieferanten Haftungen für indirekte oder Folgeschäden oder sonstige Schäden, die auf Nutzungsausfall, Verlust von Daten oder entgangenem Gewinn gestützt werden und die durch die oder im Zusammenhang mit der Verwendung von auf diesem Server verfügbaren Informationen entstanden sind. Der Lizenzgeber und deren Lieferanten behalten sich die Möglichkeit vor, jederzeit Verbesserungen und/oder Änderungen an den hier beschriebenen Produkten und/oder Programmen vorzunehmen. Insbesondere ist die Haftung für Veränderungen von dritter Seite ausgeschlossen. crQ beschränkt die Haftung auf Schäden, die durch Vorsatz verursacht wurden.

crQ übernimmt keine Gewähr dafür, dass die Software für die von Ihnen bestimmten Zwecke, für die Sie die Software einsetzen wollen, tauglich ist oder mit anderer, von Ihnen gewählter Software kompatibel ist. Sie tragen die alleinige Verantwortung für Auswahl, Installation und Nutzung sowie für die damit beabsichtigten Ergebnisse. crQ haftet nicht für irgendeinen Schaden, der durch die Verwendung oder die Unmöglichkeit der Verwendung der Software verursacht worden ist. Dies gilt ohne Ausnahme auch für entgangenen Geschäftsgewinn, Betriebsunterbrechungen, entgangene Geschäftsinformation oder anderen wirtschaftlichen Verlust, auch wenn crQ vorher auf die Möglichkeit eines solchen Schadens hingewiesen hat.

<http://www.crQuality.de>

1. qSQL

1.1 Installation

qSQL bietet die Möglichkeit, über SQL die Datenqualität von Daten zu analysieren und zu überprüfen. Dazu wird auf dem SQL Server eine Datenbank „q“ angelegt, die zahlreiche Routinen mit Datenqualitätsverfahren enthält.

Zur Installation von qSQL unter dem Microsoft SQL Server 2008, legen Sie bitte einen Ordner auf dem Computer an, auf dem der SQL Server läuft (z.B. „c:\qSQL“). Starten Sie dann das „Microsoft SQL Server Management Studio“ (SSMS) und öffnen Sie das SQL-Skript „Install.sql“. Passen Sie gegebenenfalls den Pfad zu den Datenbankdateien an (als Standard wird hier der Pfad „c:\qSQL“ verwendet).

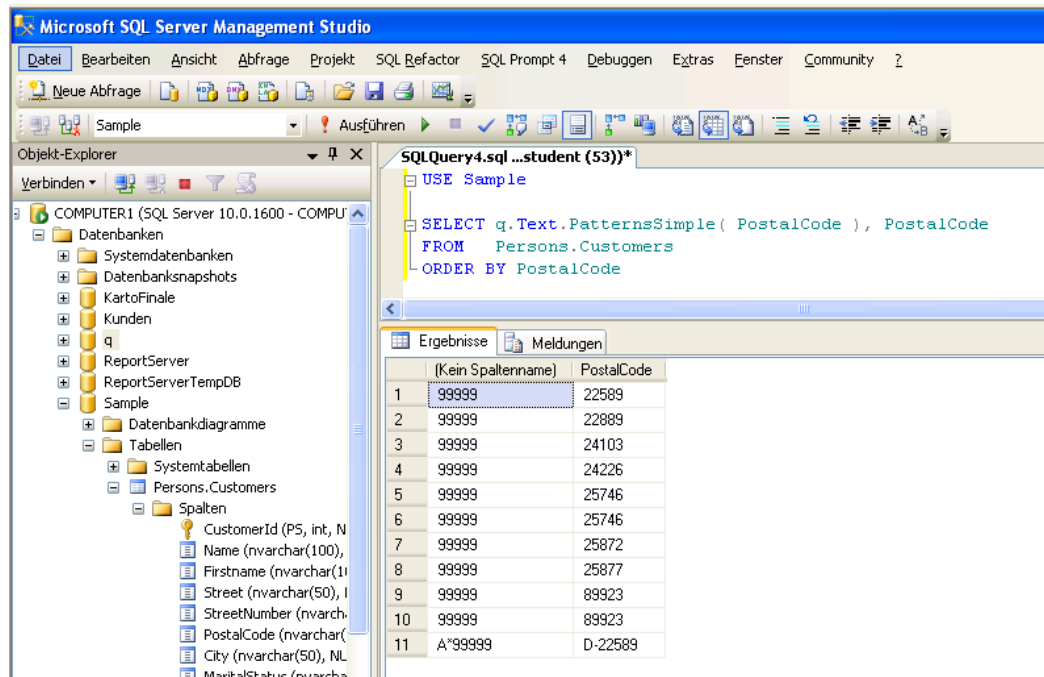
Führen Sie das Skript aus!

Sofern das Skript ohne Fehlermeldungen ausgeführt wurde, ist qSQL nun auf Ihrem Computer installiert und kann verwendet werden. Zur Überprüfung sehen Sie im SSMS nach, ob die beiden Datenbanken „q“ und „sample“ existieren. Für einen ersten Test, geben Sie im SMSS folgende SQL-Anweisung ein und führen Sie aus.

```
USE Sample
GO
```

```
SELECT q.Text.PatternsSimple( PostalCode ), PostalCode
FROM   Persons.Customers
GROUP BY PostalCode
```

Abb. 1 Beispiel



Quelle: eigene Darstellung

qSQL verwendet vor allem Sprachkonstrukte aus den bekannten vier RDBMS (IBM DB2, Oracle, Informix, Microsoft SQL Server), die sich weitgehend an den Standard anlehnen. Zu diesen Sprachkonstrukten zählen SQL-Funktionen¹, SQL-Prozeduren und Trigger. Zusätzlich wurden bestimmte Verfahren mit benutzerdefinierten Aggregatfunktionen gelöst, die zwar im SQL-Standard noch nicht enthalten, aber von den meisten bekannten RDBMS wie z.B. dem Informix Dynamics Server, Microsoft SQL Server, Oracle, MySQL und PostgreSQL unterstützt werden. Für die effiziente Nutzung der Datenqualitätskomponente sind diese jedoch nicht zwingend notwendig, erleichtern aber die Verwendung.

Beim Microsoft SQL Server 2008 wurden die meisten Verfahren mit der .NET CLR-Integration implementiert, da in .NET CLR verschiedene Programmiersprachen parallel in einem System verwendet werden können.

Alle Verfahren werden in einer eigenen Datenbank gespeichert, die aus mehreren Schemata besteht. Hierzu gehören:

¹ Hierzu gehören auch die seit SQL:2003 standardisierten Tabellenrückgabefunktionen.

- Text
- Number
- Date
- Distance
- Understand
- Improve
- Tools

Die Basisverfahren sind in den Schemata „Text“, „Number“, „Date“ und „Distance“ gespeichert. Verfahren zum Verstehen der Daten befinden sich im Schema „Understand“. Das Schema „Tools“ enthält schließlich allgemeine Routinen wie z.B. Registrieren eines Plug-in.

Außerdem existieren weitere Schemata zum persistenten Speichern von Metadaten und Nachschlagetabellen:

- Repository
- Lookup

In den folgenden Abschnitten werden alle Verfahren sowie die exemplarische Anwendung einzelner Methoden über SQL kurz erklärt. Eine ausführliche Beschreibung der SQL-Routinen mit einer Auflistung benannter Parameter ist in Anhang A und B zu finden.

1.2 Basis

1.2.1 Allgemein

Die Basisverfahren bestehen aus Verfahren zum Erzeugen von Codes aus einem Wert (Encoder), zum Überprüfen der Ähnlichkeit zweier Werte (Matching) und zum Aufteilen eines Wertes in Teilwerte wie z.B. linguistische Einheiten oder N-Grams (Tokenizer). Da diese

drei Verfahrenstypen vom Datentyp (Text, Number, Date, Distance) abhängig sind, werden sie den jeweiligen Schemata für diesen Datentyp untergeordnet. Für die Überprüfung, ob zwei Zahlen oder zwei Zeichenketten ähnlich sind, werden z.B. weitgehend unterschiedliche Algorithmen.

Für Matching, Encoder und Tokenizer existiert jeweils eine generische Funktion, die ein beliebiges Matching-, Encoder- oder Tokenizer-Verfahren aufrufen kann. Dazu werden an diese generische Funktion neben den spezifischen Parametern der vollständige CLR-Klassenname des zu verwendenden Verfahrens, die Bibliothek (Assembly) und benannte Parameter übergeben. Diese Vorgehensweise findet sich nicht nur in der Basis, sondern durchgängig im gesamten Framework. Dadurch ist es möglich, eigene Bibliotheken, die als Plug-in bei der Datenqualitätskomponente registriert wurden, aufzurufen. Diese Vorgehensweise wird in den folgenden Abschnitten zu Erweiterungen jeweils erläutert.

1.2.2 Encoder

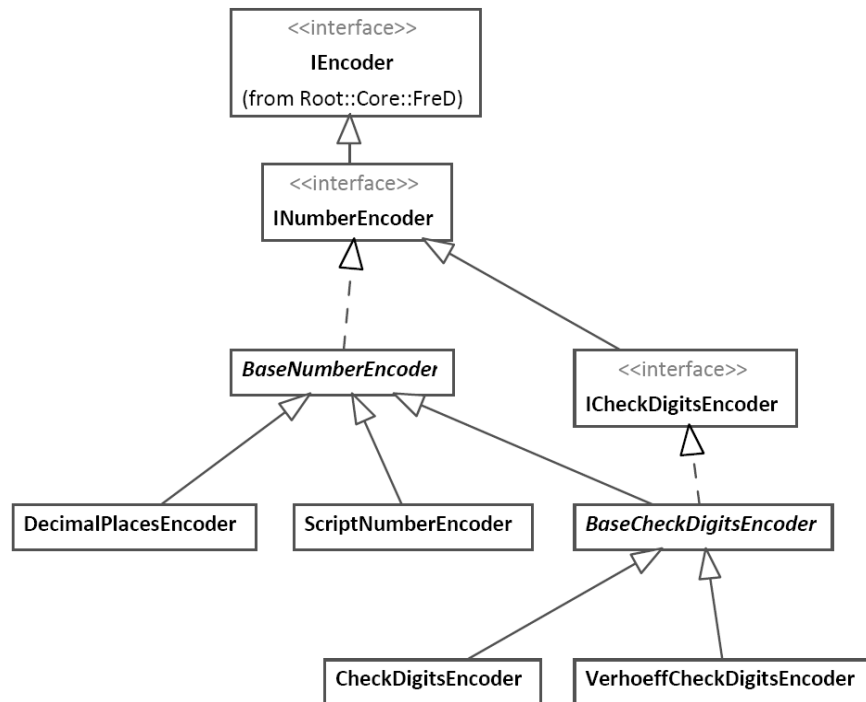
Bei einem Encoder handelt es sich um einen Kodierer, der aus einem Wert einen Code erzeugt.

Der Namensraum *FreD.Number.Encoder* enthält alle in der Klassenbibliothek implementierten Kodierer für Zahlen. Ein Kodierer für Zahlen muss die Schnittstelle *INumberEncoder* implementieren, die wiederum von der allgemeinen Schnittstelle *IEncoder* abgeleitet ist. Diese Schnittstelle definiert die Methode *Encode*, der eine Zahl zum Kodieren übergeben wird.

Über die Klasse *DecimalPlacesEncoder* wird für eine Zahl die Anzahl der Nachkommastellen ermittelt. *CheckDigitsEncoder* und *VerhoeffCheckDigitsEncoder* erzeugen eine Prüfziffer und implementieren zusätzlich über ihre Basisklasse *BaseCheckDigitsEncoder* die Methode *Decode* zur Überprüfung einer Prüfziffer.

Der Klasse *ScriptNumberEncoder* kann Quellcode in einer .NET Programmiersprache übergeben werden, die einen Kodierer für Zahlen implementiert.

Abb. 1 FreD.Number.Encoder



Quelle: eigene Darstellung

Die folgende SQL-Anweisung zeigt die Verwendung im RDBMS über die generische Encoder-Funktion (für die folgenden SQL-Beispiele wurde die Datenbank „AdventureWorks“ verwendet):

```

SELECT CardType, CardNumber,
       q.Number.Encode( CardNumber,
                        'FreD.Number.Encoder.VerhoeffCheckDigitsEncoder',
                        'Core', null )
FROM   Sales.CreditCard
    
```

Entsprechend sieht der spezifische Encoder-Aufruf folgendermaßen aus:

```

SELECT CardType, CardNumber,
       q.Number.CheckDigitsVerhoeffEncode(CardNumber)
    
```

```
FROM Sales.CreditCard
```

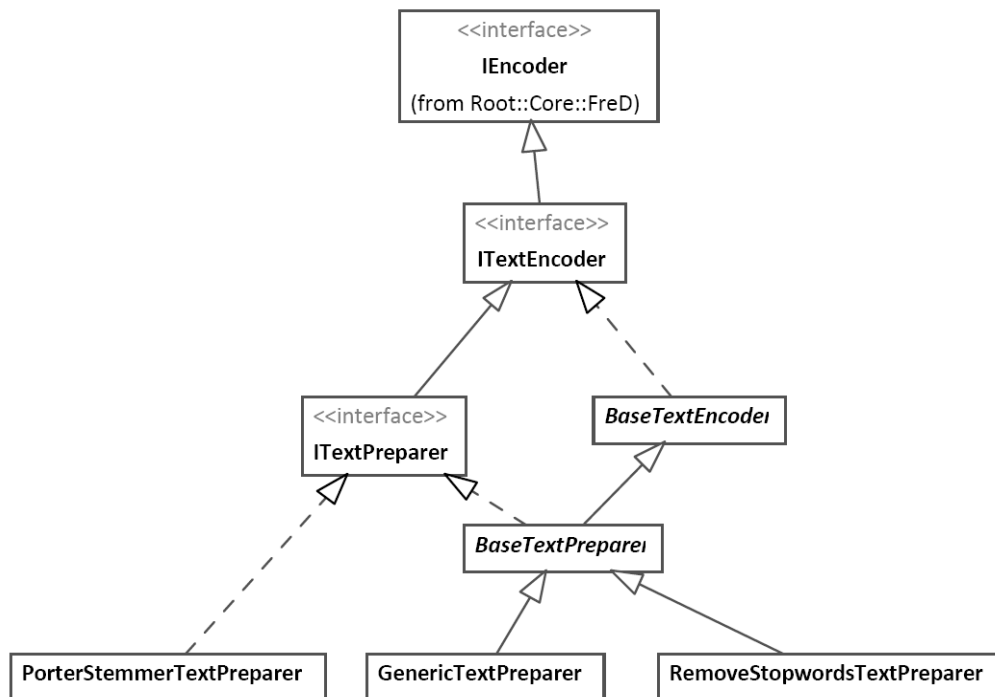
Im Gegensatz zum generischen können beim spezifischen Aufruf zugunsten der einfacheren Anwendung keine zusätzlichen benannten Parameter übergeben werden.

Der Namensraum *FreD.Text.Encoder* enthält alle in der Klassenbibliothek implementierten Kodierer für Text bzw. alphanumerische Daten. Hierbei kann unterschieden werden zwischen Kodierv Verfahren zur Bildung eines Hashwertes (Hashing), zur Musterbildung (Pattern), zur Erzeugung phonetischer Codes und zur Vorbereitung von Text (Preparer). Ein Kodierer für Text implementiert die Schnittstelle *ITextEncoder*, die genau wie bei numerischen Encodern von der allgemeinen Schnittstelle *IEncoder* abgeleitet ist.

Preparer implementieren die Schnittstelle *ITextPreparer*. In der Klassenbibliothek sind zwei Preparer implementiert: Der *GenericTextPreparer* (zur allgemeinen Umwandlung von Text, Groß-/Klein, Zeichen ignorieren u.ä.) und der *RemoveStopwordsTextPreparer*, der aus einer ihm übergebenen Zeichenkette Stoppwörter aus einer im Repository gespeicherten Liste ausfiltert. Daneben existieren als Plug-ins verschiedene Stemmingalgorithmen (z.B. der *PorterStemmerTextPreparer*²), die direkt von *ITextPreparer* abgeleitet sind.

² Als Plug-In's sind außerdem die Stemming-Algorithmen von Lovins, Paice/Husk, Krovetz, Caumann eingebunden und der UEA-Stemming-Algorithmus (diese werden nicht mit qSQL ausgeliefert, sondern sind nur auf Anfrage erhältlich)

Abb. 2 FreD.Text.Encoder (Preparer)



Quelle: eigene Darstellung

Die folgende SQL-Anweisung zeigt zunächst einen generischen Aufruf, der das Plug-in zum Porter-Stemming aufruft.³

```

SELECT q.Text.Encode( DocumentSummary,
                      'FreD.Text.Encoder.PorterStemmerTextPreparer',
                      'PorterStemmerTextPreparer', null )
FROM   Production.Document
  
```

Ein spezifischer Aufruf zum Entfernen von Stopwörtern sieht wie folgt aus.

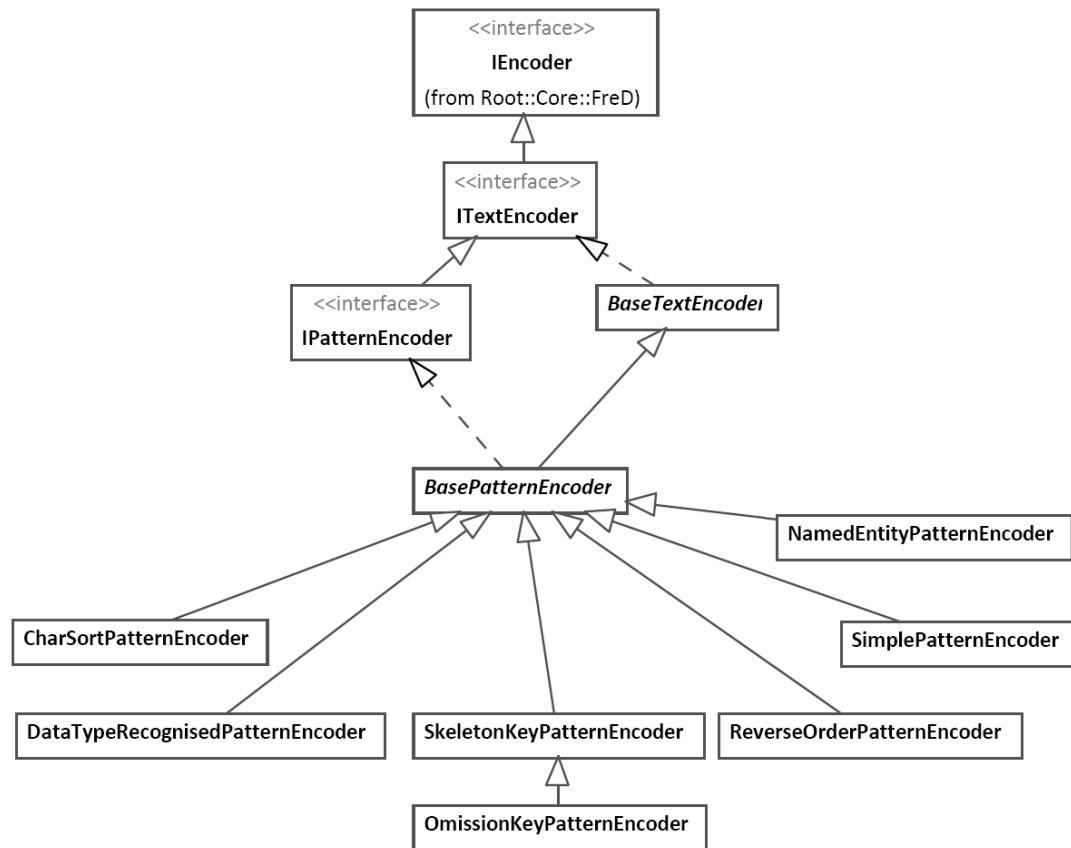
```

SELECT q.Text.PreparerRemoveStopwords( DocumentSummary )
FROM   Production.Document
  
```

Verfahren zur Erzeugung von Pattern (siehe Abb. 3) implementieren die *IPatternEncoder*-Schnittstelle.

³ Für die folgende Anweisung muss das Plugin „PorterStemmerTextPreparer“ auf dem Microsoft SQL Server 2008 installiert sein

Abb. 3 FreD.Text.Encoder (Pattern)



Quelle: eigene Darstellung

Neben der in der Klassenbibliothek enthaltenen *SimplePatternEncoder*-Klasse, für die ein gemeinsames Zeichen für Zahlen, Großbuchstaben, Kleinbuchstaben und andere Zeichen festgelegt wird, existieren in der Klassenbibliothek noch sechs weitere Pattern-Kodierer. Der *CharSortPatternEncoder* sortiert eine Zeichenkette alphabetisch. Der *DataTypeRecognisedPatternEncoder* kann verwendet werden, um den eigentlichen Datentyp eines Attributes zu ermitteln, indem die Klasse den Datentyp für eine Zeichenkette erkennt und diesen als Datentypbeschreibung zurückliefert. Über *ReverseOrderPatternEncoder* werden die übergebenen Zeichen in umgekehrter Reihenfolge zurückgeliefert. Dies kann z.B. bei der Duplikaterkennung sinnvoll sein, wenn eine Zeichenkette aus mehreren unterschiedlichen semantischen Werten besteht. *NamedEntityPatternEncoder* liefert für einen übergebenen Wert den semantischen Typ zurück und kann z.B. zum Parsing von Spalten mit

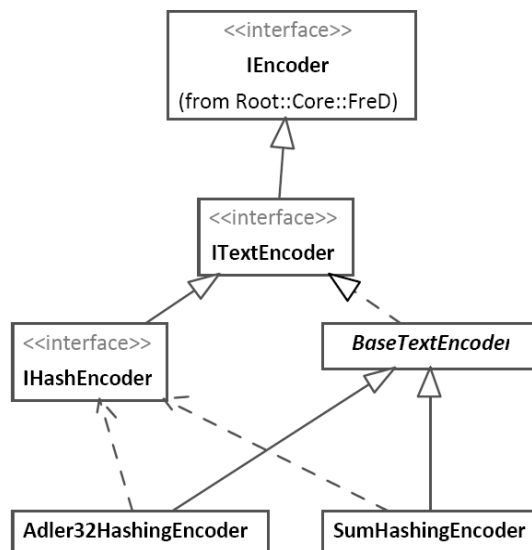
mehreren Werten eingesetzt werden. *SkeletonKeyPatternEncoder* und *OmissionKeyEncoder* schließlich werden vorrangig zur Erkennung von Eingabefehlern verwendet.

Die folgende SQL-Anweisung zeigt die spezifischen Aufrufe zu den beschriebenen Pattern-klassen:

```
SELECT Lastname, Phone,
       q.Text.PatternsSimple( Phone ),
       q.Text.PatternsCharSort( Phone ),
       q.Text.PatternsReverseOrder( Phone ),
       q.Text.PatternsOmissionKey( Lastname ),
       q.Text.PatternsSkeletonKey( Lastname ),
       q.Text.PatternsDataTypeRecognised( Phone ),
       q.Text.PatternsNamedEntity( Lastname )
FROM   Person.Contact
```

Als Hashverfahren (siehe Abb. 4) sind in der Bibliothek der Adler32-Algorithmus und die einfache Quersummenbildung implementiert. Hashverfahren implementieren direkt die Schnittstelle *IHashEncoder*.

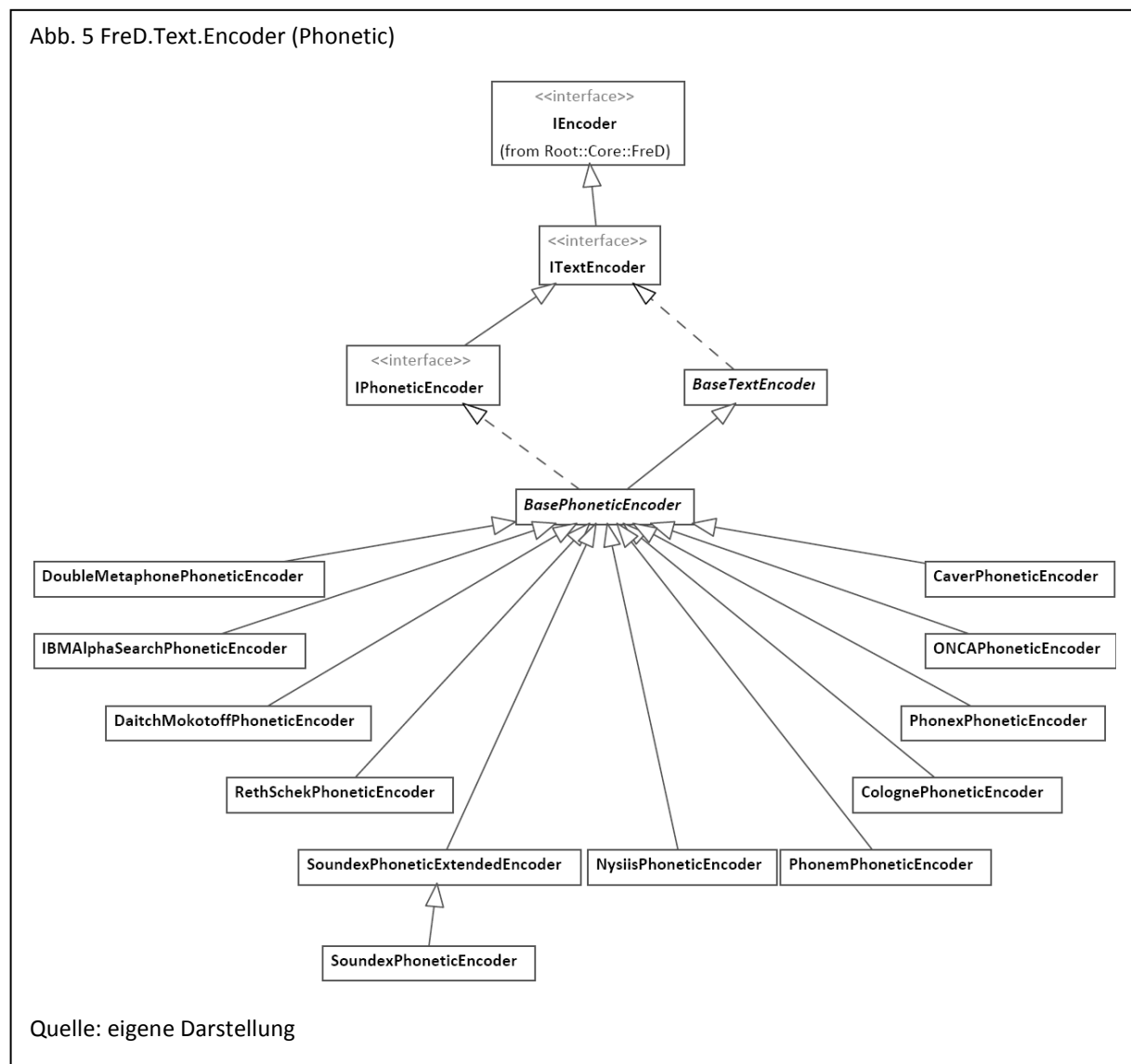
Abb. 4 FreD.Text.Encoder (Hashing)



Quelle: eigene Darstellung

Algorithmen zur Erzeugung phonetischer Codes (siehe Abb. 5) müssen die Schnittstelle *IPhoneticEncoder* implementieren. Neben dem bekannten phonetischen Soundex-Code, der über die Klasse *SoundexPhoneticEncoder* implementiert ist, sind die meisten bekannten Algorithmen implementiert. Der von J. Michael entwickelte phonet-Code ist nicht in der Klassenbibliothek qSQL, sondern als Plug-in enthalten, da der originale C-Code aus dem Jahr 1992 verwendet wurde.⁴ Die Klasse *SoundexPhoneticEncoder* ist von der *SoundexPhoneticExtendedEncoder* abgeleitet, die ein eigenes Mapping für die einzelnen alphabetischen Zeichen zu einem gemeinsamen Codezeichen erlaubt.

Abb. 5 FreD.Text.Encoder (Phonetic)



⁴ Siehe zum Quellcode <ftp://ftp.heise.de/pub/ct/listings/phonet.zip>

Die folgende SQL-Anweisung beinhaltet einen generischen Aufruf, um ein eigenes Mapping für den Soundex-Code zu verwenden, der allen Vokalen das Zeichen ‚X‘ zuweist:

```
SELECT LastName,
       q.Text.Encode( LastName,
                     'FreD.Text.Encoder.SoundexPhoneticEncoder', 'Core',
                     'Mapping=X123X12-X22455X12623X1-2-2' )
FROM   Person.Contact
```

Den Aufruf spezifischer phonetischer Encoder zeigt die folgende SQL-Anweisung:

```
SELECT LastName,
       q.Text.PhoneticCaver( LastName ),
       q.Text.PhoneticCologne( LastName ),
       q.Text.PhoneticDaitchMokotoff( LastName ),
       q.Text.PhoneticDoubleMetaphone( LastName ),
       q.Text.PhoneticIBMAAlphaSearch( LastName ),
       q.Text.PhoneticNysiis( LastName ),
       q.Text.PhoneticONCA( LastName ),
       q.Text.PhoneticPhonem( LastName ),
       q.Text.PhoneticPhonex( LastName ),
       q.Text.PhoneticRethSchek( LastName ),
       q.Text.PhoneticSoundex( LastName )
FROM   Person.Contact
```

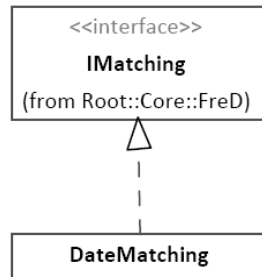
1.2.3 Matching

Beim Matching wird die Ähnlichkeit zwischen zwei Werten über ein Ähnlichkeitsmaß ausgedrückt. Genau wie bei der Erzeugung von Codes gibt es je nach Datentyp unterschiedliche Algorithmen. Im Folgenden werden die Algorithmen zur Überprüfung der Ähnlichkeit bei den Datentypen Date, Distance, Number und Text beschrieben.

Für den Datentyp Date existiert in der Klassenbibliothek eine einfache Klasse zur Überprüfung der Ähnlichkeit zweier Datumswerte, indem ein maximaler Differenzwert festgelegt

wird. Die Klasse *DateMatching* ist direkt von *IMatching* abgeleitet und implementiert die Schnittstellenmethode *Match*.

Abb. 6 FreD.Date.Matching



Quelle: eigene Darstellung

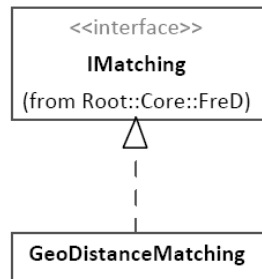
Die folgende SQL-Anweisung vergleicht über einen generischen Aufruf zwei Datumswerte und errechnet deren Ähnlichkeit auf Basis der Eigenschaft *MaxTimeSpan*.

```

SELECT OrderDate,
       ShipDate,
       q.Date.Match( OrderDate, ShipDate,
                     'FreD.Date.Matching.DateMatching', 'core',
                     'MaxTimeSpan=14.00:00:00' )
FROM   Sales.SalesOrderHeader
    
```

Über den Datentyp *Distance* werden Entfernungen zwischen Orten auf Grundlage von Geodaten (Längen-, Breitengrad) errechnet. An die *Match*-Methode der *GeoDistanceMatching*-Klasse werden zwei Postleitzahlen übergeben. Dann wird die Entfernung in Kilometern berechnet und auf Basis einer vom Benutzer angegebenen maximalen Entfernung ein Ähnlichkeitsmaß zurückgeliefert.

Abb. 7 FreD.Distance.Matching



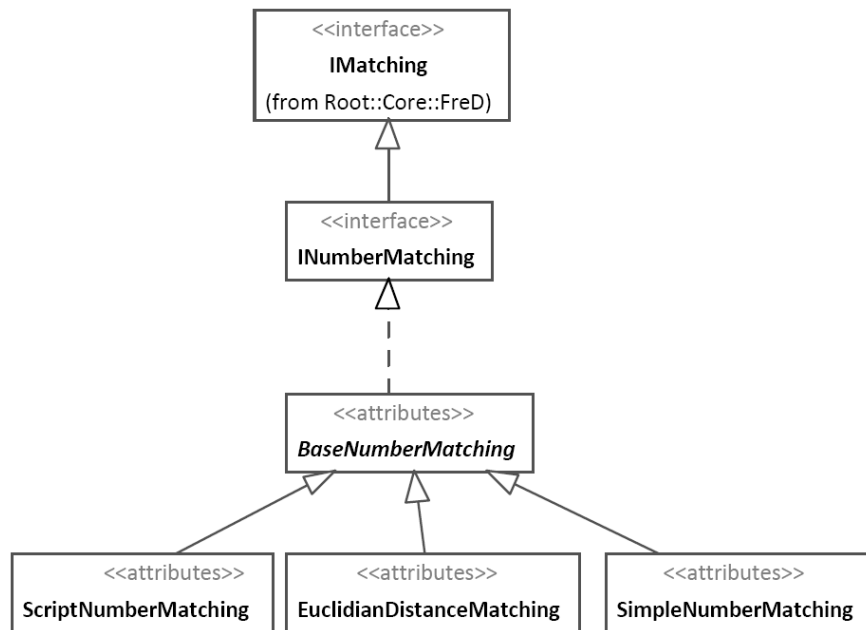
Quelle: eigene Darstellung

Die folgende SQL-Anweisung vergleicht über einen generischen Aufruf zwei Postleitzahlen und errechnet deren Ähnlichkeit auf Basis der Eigenschaft *MaxDistance*.

```
SELECT q.Distance.Match(
    '22880', '20095',
    'FreD.Distance.Matching.GeoDistanceMatching', 'core',
    'Country=DE, MaxDistance=100' )
```

Algorithmen zur Errechnung eines Ähnlichkeitsmaßes von zwei Zahlen sind im Namensraum *FreD.Number.Matching* implementiert.

Abb. 8 FreD.Number.Matching



Quelle: eigene Darstellung

Neben der Klasse *ScriptNumberMatchingEncoder*, mit der eigene Algorithmen als Skript an die *Match*-Methode übergeben werden, existieren Implementierungen zur euklidischen Distanz in *EuclidianDistanceMatching* und eine einfache Berechnung in *SimpleNumberMatching*, dem ein maximaler Distanzwert übergeben wird.

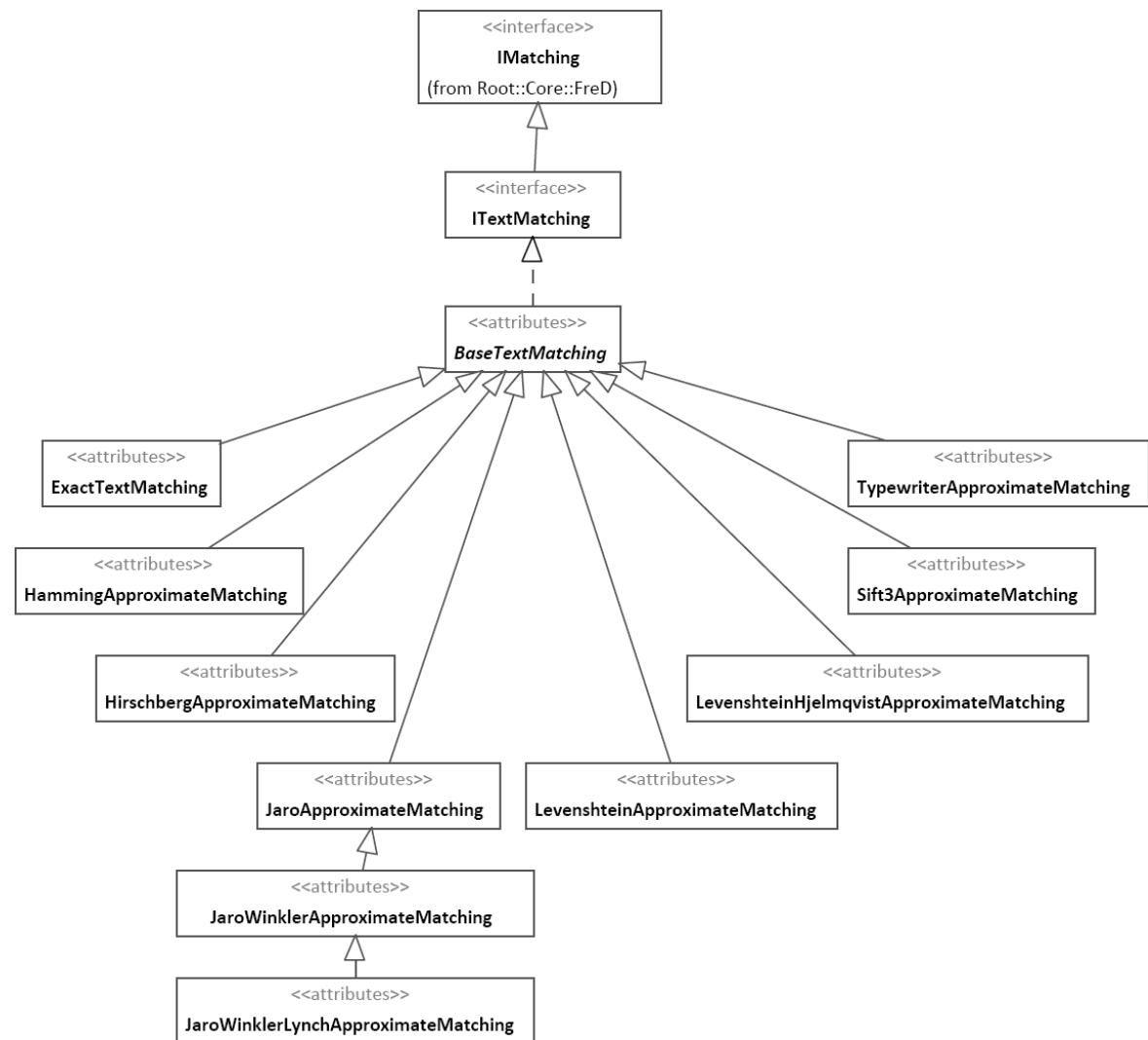
Folgendes Beispiel einer SQL-Anweisung berechnet die euklidische Distanz als Ähnlichkeitsmaß umgerechnet:

```

SELECT StandardCost, ListPrice,
       q.Number.MatchingEuclidian( StandardCost, ListPrice )
FROM   Production.Product
  
```

Der Namensraum *FreD.Text.Matching* enthält die bekanntesten Verfahren zum Vergleich von Zeichenketten. Bei den Algorithmen kann unterschieden werden zwischen Zeichen- und Token-basierten Verfahren. Abb. 9 zeigt zunächst die Klassenhierarchie für die Zeichen-basierten Algorithmen.

Abb. 9 FreD.Text.Matching (Zeichen-basiert)



Quelle: eigene Darstellung

Die Klasse *ExactTextMatching* überprüft die exakte Übereinstimmung zweier Zeichenketten. Dementsprechend wird als Proximitätsmaß entweder 1 bei Übereinstimmung oder 0 bei Nicht-Übereinstimmung zurückgegeben.

Die folgende SQL-Anweisung verwendet Matching-Routinen zur Ermittlung von Duplikaten in einer Tabelle:

```

WITH Employees( EmployeeId, LastName, AddressLine1, City, PostalCode ) AS
(
    SELECT e.EmployeeId,
           pc.LastName,
           pa.AddressLine1,
           pa.City,

```

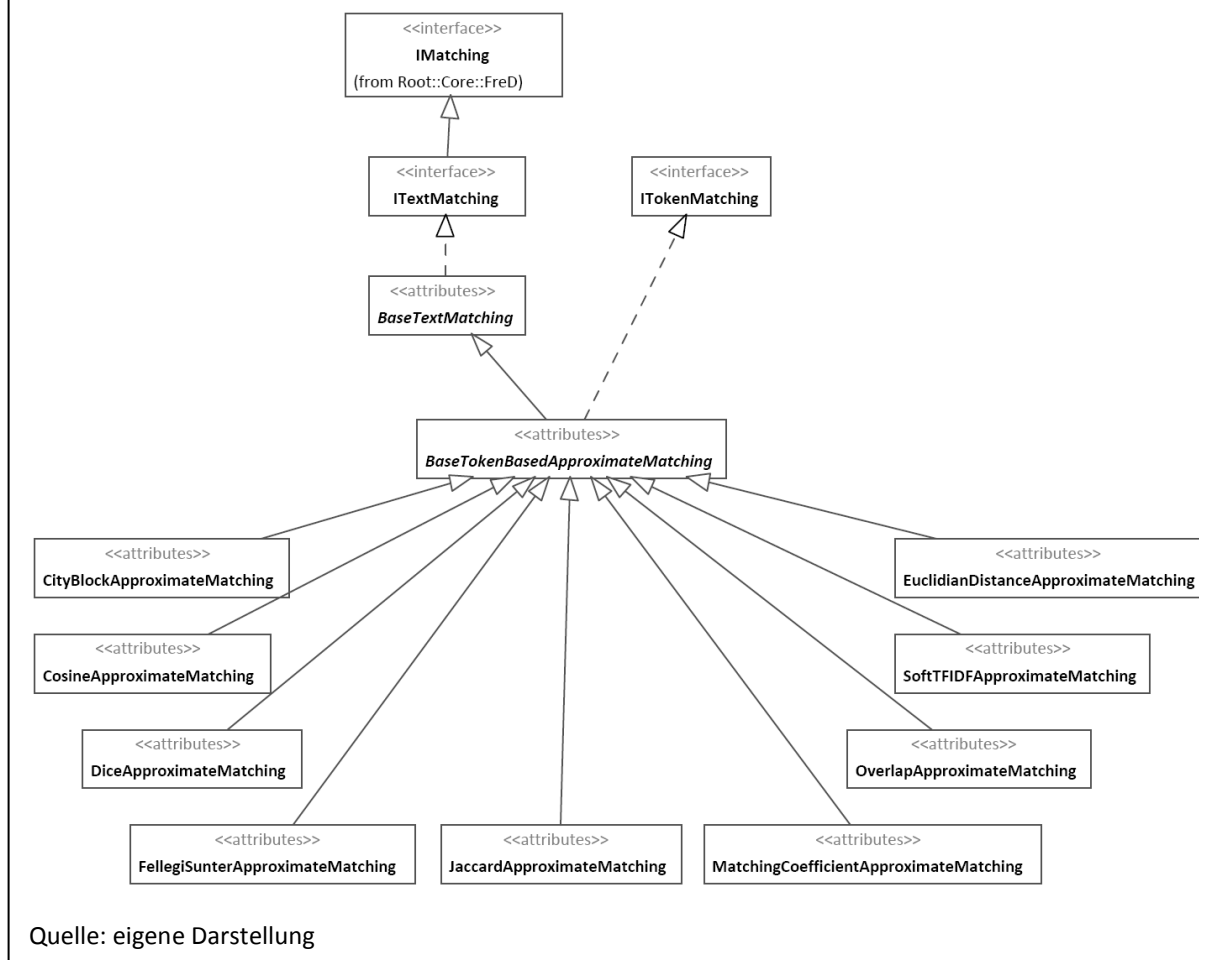
```

        pa.PostalCode
FROM    Person.Address pa INNER JOIN HumanResources.EmployeeAddress ea
        ON pa.AddressID = ea.AddressID INNER JOIN
        HumanResources.Employee e ON ea.EmployeeID = e.EmployeeID
        INNER JOIN Person.Contact pc ON e.ContactID = pc.ContactID
)
SELECT a.LastName, b.LastName,
       a.AddressLine1, b.AddressLine1,
       a.City, b.City,
       a.PostalCode, b.PostalCode
FROM    Employees a, Employees b
WHERE   a.EmployeeId > b.EmployeeId AND
       q.Text.PhoneticSoundex( a.LastName ) =
       q.Text.PhoneticSoundex( b.LastName ) AND
       q.Text.MatchingJaro( a.LastName, b.LastName ) > 0.7 AND
       q.Text.MatchingHirschberg(a.AddressLine1, b.AddressLine1 ) > 0.7 AND
       q.Distance.MatchingSimple(a.PostalCode, b.PostalCode, 'US', 50) > 0.9

```

In Abb. 10 ist die Klassenhierarchie der Token-basierten Algorithmen zum Vergleich von Zeichenketten dargestellt. Token-basierte Algorithmen implementieren sowohl die Schnittstelle *ITextMatching*, als auch die Schnittstelle *ITokenMatching*. Zu den probabilistischen Matching-Verfahren gehören der Algorithmus zur Duplikaterkennung nach Fellegi und Sunter und der TF-IDF-Algorithmus, die beide in der Klassenbibliothek implementiert sind. Alle anderen implementierten Algorithmen sind deterministische Verfahren. Alle Klassen benötigen zwingend einen Tokenizer, um die einzelnen Zeichenketten, die verglichen werden sollen, in einzelne Token aufzuteilen. Als Standard wird hierbei die Klasse *GenericTokenizer* (siehe nächsten Abschnitt) verwendet, sofern kein Tokenizer definiert wird. Über den benannten Parameter *TokenizerTokenBased* kann jedoch jeder Klasse der Token-basierten Verfahren auch ein beliebiger anderer Tokenizer zugewiesen werden.

Abb. 10 FreD.Text.Matching (Token-basiert)



Die folgende erste SQL-Anweisung berechnet für zwei Zeichenketten das Jaccard-Ähnlichkeitsmaß (zur Trennung der Zeichenkette in Token wird als Standard der *GenericTokenizer* verwendet). Die zweite SQL-Anweisung verwendet die generische Funktion *Match*, um einen N-Gram-Tokenizer der Größe 3 zur Berechnung des Jaccard-Maßes zu verwenden:

```

SELECT q.Text.MatchingTokenJaccard( 'Hans Muster Hamburg',
                                     'Hamburg, Hans Muster' )

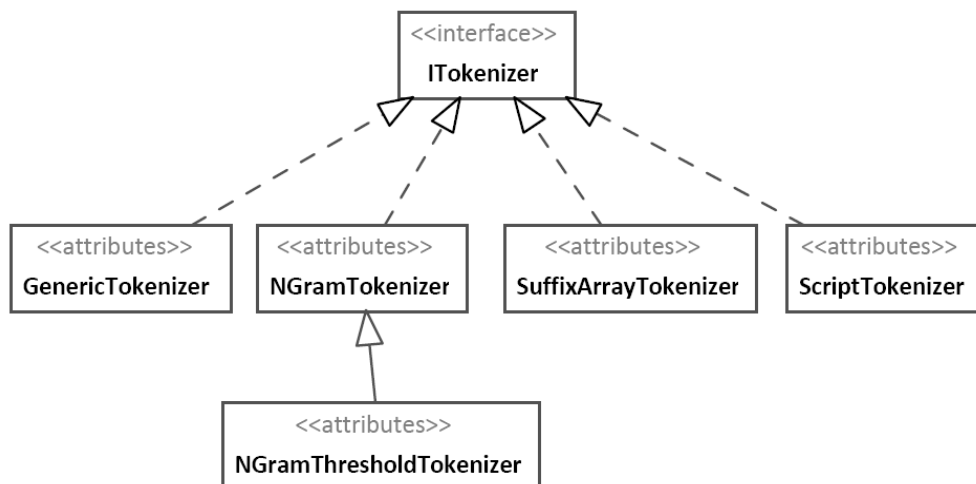
SELECT q.Text.Match( 'Hans Muster Hamburg', 'Hamburg, Hans Muster',
                    'FreD.Text.Matching.JaccardApproximateMatching',
                    'Core',
                    '{Tokenizer=FreD.Text.Tokenizer.NGramTokenizer, NGramSize=3}' )
  
```

1.2.4 Tokenizer

Algorithmen, die Zeichenketten in bestimmte Segmente (z.B. in Wörter) aufteilen, bezeichnet man als Tokenizer. Neben der Klasse *GenericTokenizer*, über die eine Zeichenkette nach bestimmten Trennzeichen wie z.B. Komma, Semikolon, Punkt usw. geteilt wird, sind in der Klassenbibliothek vier weitere Tokenizer implementiert.

Die Klasse *NGramTokenizer* teilt die Zeichenkette in gleich große Einheiten auf. Die Klassen *SuffixArrayTokenizer* und *NGramThresholdTokenizer* werden vor allem zur Reduktion des Suchraums bei der Duplikaterkennung verwendet. Über die Klasse *ScriptTokenizer* können schließlich eigene Tokenizer über ein Skript zur Verfügung gestellt werden.

Abb. 11 FreD.Text.Tokenizer



Quelle: eigene Darstellung

Im Gegensatz zu den bisherigen Funktionen werden Tokenizer hauptsächlich von anderen Klassen, wie den Algorithmen zur Berechnung Token-basierter Distanz- oder Ähnlichkeitsmaße, als Parameter verwendet. Als Schnittstelle zum RDBMS existieren die beiden generischen Funktionen *Tokenize* und *TokenizeFrequency*. Der Funktion *Tokenize* wird neben der Klassenbezeichnung, der Bibliothek und benannten Parametern eine Zeichenkette übergeben, die entsprechend dem Algorithmus in Token aufgeteilt wird. Das folgende SQL-Beispiel verwendet die Klasse *SuffixArrayTokenizer* zum Trennen:

```
SELECT q.Text.Tokenize( LastName, ' | ',
                        'FreD.Text.Tokenizer.SuffixArrayTokenizer',
                        'Core', 'MinSuffixLength=5' )
FROM   AdventureWorks.Person.Contact
```

Die Funktion *TokenizeFrequency* liefert eine Häufigkeitstabelle aller Token für eine Spalte in einer Tabelle zurück. Die Häufigkeitstabelle enthält drei Spalten: Die Spalte „Token“ enthält den aufgetrennten Wert, „Frequency“ gibt die absolute und „FrequencyRelative“ die relative Häufigkeit an. Das folgende Beispiel liefert eine Häufigkeitstabelle von N-Grams der Größe 2 für die Spalte „LastName“ der Tabelle „Person.Contact“.

```
SELECT *
FROM q.Text.TokenizeFrequency(
    'AdventureWorks.Person.Contact', 'LastName', null,
    'FreD.Text.Tokenizer.NGramTokenizer', 'Core', 'NGramSize=2', 0 )
ORDER BY Frequency DESC
```

1.2.5 Erweiterungen

Um eigene Encoder-, Matching-, oder Tokenizer-Verfahren in das Datenqualitäts-Framework zu integrieren, müssen die entsprechenden Schnittstellen oder die virtuellen Methoden der dazugehörigen Basisklasse implementiert werden. So muss eine neue Encoder-Klasse die *IEncoder* oder eine davon abgeleitete Schnittstelle implementieren. Das folgende Beispiel zeigt einen einfachen Encoder in Java, der eine Zeichenkette in Großbuchstaben umwandelt.

```
package DataQuality.Extension;

import java.lang.String.*;
import FreD.Text.Encoder.*;

public class UpperCase implements ITextPreparer
{
    public String Encode( String text )
    {
        if (text == null )
            return null;

        return text.toUpperCase();
    }
}
```

Zum Registrieren dieser Erweiterung beim Framework existiert die SQL-Prozedur *AddPlugin*. Dieser Funktion wird der Pfad auf die Quellcodedatei oder der Quellcode selbst übergeben sowie der Name der Bibliothek, unter der dieser im RDBMS registriert werden soll, und eventuell zu verwendende weitere Klassenbibliotheken, auf die im Quellcode referenziert wird. Die folgende SQL-Anweisung registriert den obigen Encoder unter dem Bibliotheks-namen „SampleJava“. Die Klassenbibliothek des Datenqualitäts-Frameworks befindet sich in der Datei „core.dll“ und muss referenziert werden, da in ihr die Schnittstellen beschrieben sind.

```
EXEC q.Tools.AddPlugin 'c:\FreD\samples\Encoder.java',
                      'SampleJava',
                      'c:\FreD\core.dll'
```

Zum Aufruf des Encoders wird die generische Encoder-Funktion wie folgt verwendet:

```
SELECT q.Text.Encode( LastName,
                    'DataQuality.Extension.UpperCase', 'SampleJava', null )
FROM Person.Contact
```

Benannte Parameter, die an die generische Funktion übergeben werden können, werden als Properties implementiert. Um einen zusätzlichen Property *IgnoreVowels* zum Herausfiltern von Vokalen in einer Zeichenkette zu implementieren, wird die Java-Klasse wie folgt geändert:

```
public class UpperCase implements ITextPreparer
{
    public String Encode(String text)
    {
        if (text == null)
            return null;

        text = text.toUpperCase();

        if (_ignoreVowels)
        {
            text = text.Replace("A", "");
            text = text.Replace("E", "");
            text = text.Replace("I", "");
            text = text.Replace("O", "");
            text = text.Replace("U", "");
        }
        return text;
    }
}

/** @property */
```



```
public void set_IgnoreVowels(boolean value)
{
    _ignoreVowels = value;
}

private boolean _ignoreVowels = false;
}
```

Das Setzen des Properties in einer SQL-Anweisung erfolgt im generischen Aufruf über den Parameter *namedParameters* wie folgt:

```
SELECT q.Text.Encode( LastName,
                    'DataQuality.Extension.UpperCase', 'SampleJava',
                    'IgnoreVowels=true' )
FROM AdventureWorks.Person.Contact
```

Müssen Encoder die *IEncoder*-Schnittstelle implementieren, so ist dies für die Matching-Algorithmen die Schnittstelle *IMatching* oder eine davon abgeleitete Schnittstelle. Der folgende in C# implementierte Matching-Algorithmus überprüft die Übereinstimmung der ersten n Zeichen zweier Zeichenketten. Stimmen diese überein, so wird das Ähnlichkeitsmaß als Quotient übereinstimmender Zeichen und der überprüften Anzahl an Zeichen „n“ errechnet.

```
namespace DataQuality.Extension
{
    public class SimpleMatching : FreD.Text.Matching.ITextMatching
    {
        public int StartLength
        {
            set
            {
                _startLength = value;
            }
        }

        public double Match( string text1, string text2 )
        {
            int i = 0;

            if( string.IsNullOrEmpty( text1 ) ||
                string.IsNullOrEmpty( text2 ) )
                return 0.0;

            text1 = text1.ToUpper();
            text2 = text2.ToUpper();

            for( ; i<text1.Length &&
                i<text2.Length &&
                i<_startLength; i++ )
            {
```

```

        char c1 = text1[ i ];
        char c2 = text2[ i ];

        if( c1 != c2 )
            break;
    }

    return ( double )i / _startLength;
}

private int _startLength = 5;
}
}

```

Nachdem das Plug-in über die SQL-Prozedur *AddPlugin* unter dem Bibliotheksnamen „SampleC#“ registriert wurde, kann es beispielhaft wie folgt über die generische Match-Funktion verwendet werden:

```

SELECT q.Text.Match( 'Muster', 'Musder',
                    'DataQuality.Extension.SimpleMatching', 'SampleC#', null )

```

Tokenizer müssen die *ITokenizer* oder eine davon abgeleitete Schnittstelle implementieren. Das folgende Beispiel in Visual Basic implementiert einen Tokenizer, der zum Auftrennen der einzelnen Token das Semikolon als Trennzeichen verwendet:

```

Namespace DataQuality.Extension

    Public Class SimpleTokenizer
        Implements FreD.Text.Tokenizer.ITokenizer

        Public Function Tokenize(ByVal text As String) As String() _
            Implements FreD.Text.Tokenizer.ITokenizer.Tokenize

            Dim chars() As Char = {";"}
            Return text.Split(chars)

        End Function

    End Class

End Namespace

```

Nachdem die Klasse über *AddPlugin* als Plug-in registriert ist, kann sie über die generische *Tokenize*-Funktion benutzt werden.

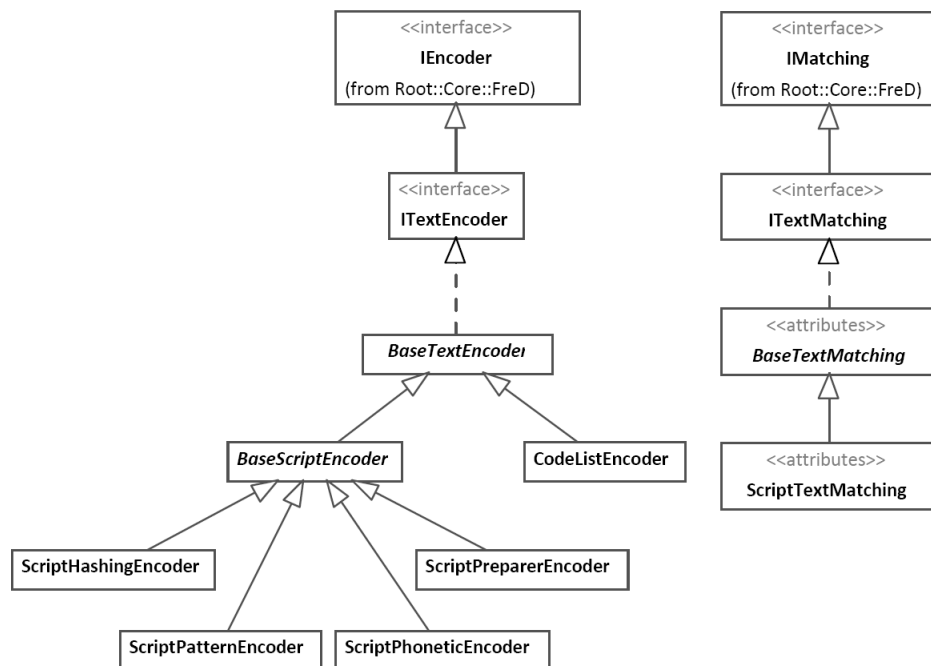
Neben dem bisher beschriebenen Verfahren zum Erweitern des Frameworks, gibt es spezielle Klassen, die die Implementierung eines speziellen Plug-ins erleichtern. Alle mit dem Namen *Script* beginnenden Klassen sind über eine spezielle *AddPlugin*-Prozedur zur Verfügung gestellt, die bei der Implementierung das Deklarieren der Schnittstelle usw. abnimmt. So kann damit die Implementierung eines Encoders etwas kürzer ausfallen. Ein Encoder, der die Eingabe in Großbuchstaben umwandelt, wird mit der speziellen Plug-in-Prozedur wie folgt registriert und aufgerufen:

```
EXEC q.Text.AddPluginPreparerScript
      'return text.ToUpper();', 'Plugin.Preparer'

SELECT LastName,
       q.Text.Encode( LastName,
                     'Quality.Text.Encoder.Encoder', 'Plugin.Preparer', null )
FROM AdventureWorks.Person.Contact
```

Erweiterungen über ein eigenes Plug-in sind zwar flexibel, aber auch aufwändig. Alle im Framework implementierten Verfahren und Algorithmen nutzen deshalb ausgiebig benannte Parameter, um das Verhalten der jeweiligen Klasse möglichst flexibel zu steuern. Z.B. kann den Token-basierten Verfahren zum Vergleichen zweier Zeichenketten ein anderer als der *GenericTokenizer* zugewiesen werden. So können beispielsweise über den *NGramTokenizer* auch typographische Fehler berücksichtigt werden. Ein Zeichen-basierter Algorithmus wiederum kann auch über einen benannten Parameter einen Tokenizer verwenden, um dadurch einen hybriden Algorithmus zu verwenden. Ein spezieller Meta-Encoder wird über die Klasse *CodeListEncoder* implementiert, die es erlaubt, beliebige Encoder zu einem gemeinsamen Encoder zu kombinieren (siehe Abb. 12).

Abb. 12 Erweiterungen der Basis



Quelle: eigene Darstellung

1.3 Verstehen

1.3.1 Allgemein

Verfahren zum Verstehen und Analysieren der Daten finden sich im Schema „Understand“. Neben den Eigenschaften einer Spalte, gehören hierzu die Analyse von Fremd-/Primärschlüsselbeziehungen und das Suchen nach Geschäftsregeln in den Daten über Regelinduktionsverfahren.

Bei den Eigenschaften einer Spalte kann man unterscheiden zwischen Daten- und Schemaeigenschaften. Dateneigenschaften ergeben sich aus den Daten selbst, wie z.B. statistische Kennzahlen, Anzahl NULL-Werte usw. Schemaeigenschaften dagegen betreffen die Struktur einer Spalte, wie z.B. Datentyp usw.

Die Analyse von Primärschlüsseln und Fremdschlüsseln betrifft sowohl die Suche nach einem geeigneten Primärschlüssel in einer Tabelle als auch die Suche der Beziehung eines Fremdschlüssels in einer Tabelle zu einem Primärschlüssel in einer anderen Tabelle.

Im Folgenden werden zunächst die SQL-Routinen zum Ermitteln von Daten- und Schemaeigenschaften beschrieben, es folgt die Analyse von Primär- und Fremdschlüsseln und Verfahren zur Regelinduktion. Im letzten Abschnitt wird erläutert, wie das Framework um eigene Verfahren erweitert werden kann.

1.3.2 Daten- und Schemaeigenschaften

Dateneigenschaften werden aus den Daten einer Spalte selbst ermittelt. Dabei gibt es sowohl generelle als auch spezielle Dateneigenschaften, die vom Datentyp abhängen. Alle Klassen, die eine Dateneigenschaft implementieren, werden von der Schnittstelle *IDataProperty* abgeleitet.

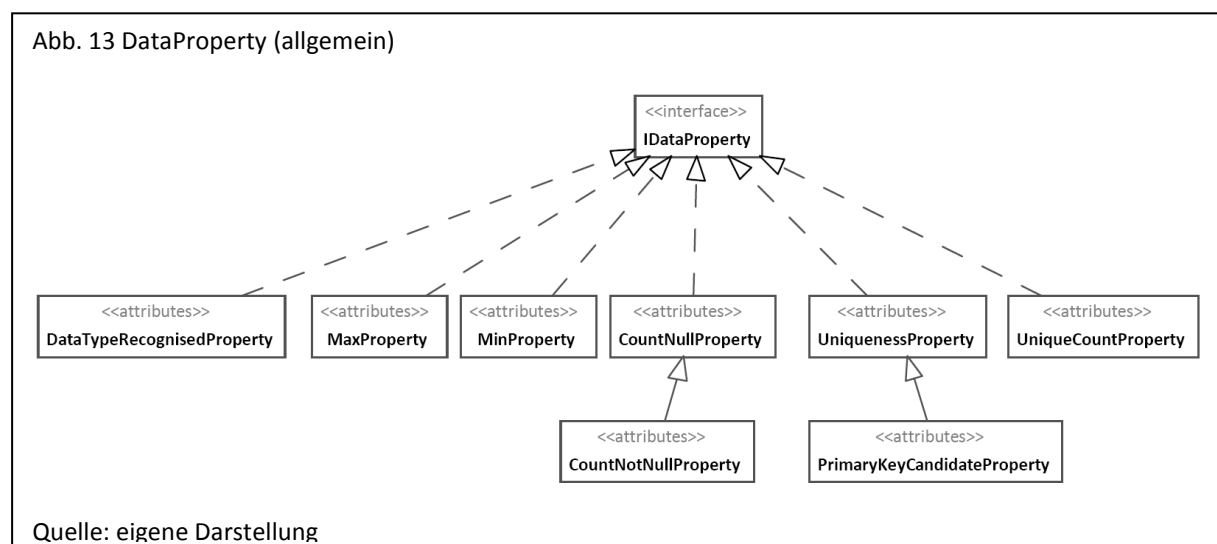


Abb. 13 zeigt alle in der Klassenbibliothek implementierten Methoden zur Ermittlung allgemeiner vom Datentyp unabhängigen Dateneigenschaften. Die Klasse *UniqueCountProperty* ermittelt die Anzahl eindeutiger Werte. *UniquenessProperty* zeigt den Anteil eindeutiger Werte im Verhältnis zur Gesamtanzahl und *PrimaryKeyCandidate* gibt

abhängig von einem Schwellwert an, ob es sich um einen Primärschlüssel handelt. Über *CountNullProperty* und *CountNotNullProperty* wird die Anzahl von NULL bzw. vorhandener Werte ermittelt. *MinProperty* und *MaxProperty* suchen die minimalen und maximalen Werte für eine Spalte, wobei deren Anzahl über einen benannten Parameter gesetzt werden kann. *DataTypeRecognisedProperty* schließlich gibt für eine Spalte den erkannten Datentyp zurück. Hierüber können Unterschiede in dem deklarierten Datentyp der Spalte und den wirklichen gespeicherten Daten überprüft werden.

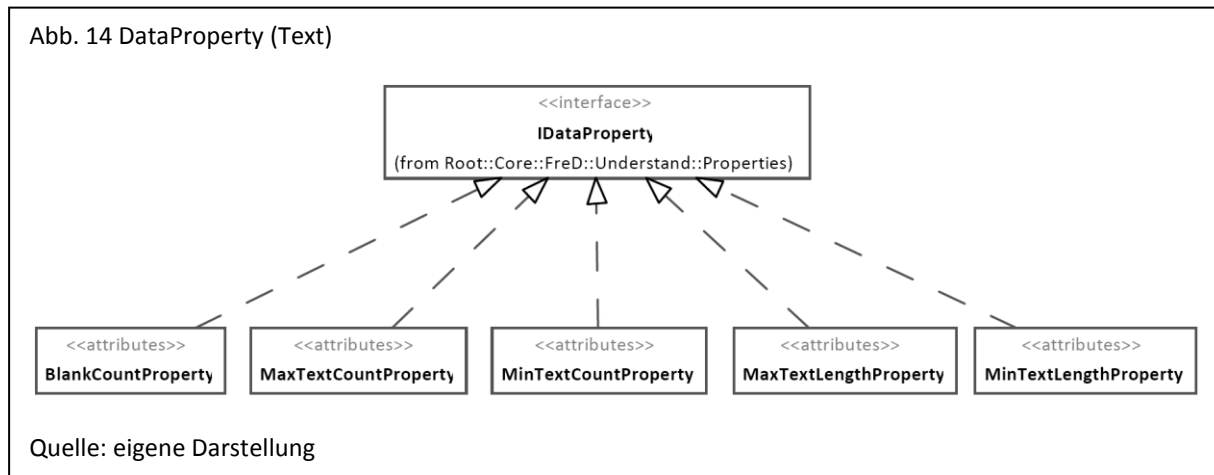
Dateneigenschaften werden im RDBMS als benutzerdefinierte Aggregatfunktionen implementiert. Neben den spezifischen existiert eine generische Aggregatfunktion zum Aufruf einer beliebigen Dateneigenschaftsklasse bzw. eines Plug-ins. Das folgende Beispiel zeigt zunächst eine Anweisung, bei der die Dateneigenschaften über spezifische Aggregatfunktionen aufgerufen werden, sowie eine weitere SQL-Anweisung mit einem generischen Aufruf zur Ermittlung der NULL-Werte:

```
SELECT q.Understand.PropertiesDataTypeRecognised( AddressLine2 ),
       q.Understand.PropertiesCountNull( AddressLine2 ),
       q.Understand.PropertiesCountNotNull( AddressLine2 ),
       q.Understand.PropertiesMax( AddressLine2 ),
       q.Understand.PropertiesMin( AddressLine2 )
FROM   Person.Address

SELECT q.Understand.PropertiesDataProperty(
        '{FreD.Understand.Properties.MinProperty|Core|MinCount=10}'+
        COALESCE( AddressLine2, '{NULL}' ) )
FROM   Person.Address
```

Spezielle Klassen, die Dateneigenschaften für Spalten mit dem Datentyp Text implementieren, sind in der folgenden Abbildung dargestellt. Über die Klasse *BlankCountProperty* wird die Anzahl an Werten ermittelt, die nur aus Leerzeichen bestehen. *MaxTextCountProperty* und *MinTextCountProperty* geben die Anzahl der Werte mit dem kürzesten bzw. längsten Text zurück und *MinTextLengthProperty* und *MaxTextLengthProperty* die jeweils kürzeste und längste Länge.

Abb. 14 DataProperty (Text)



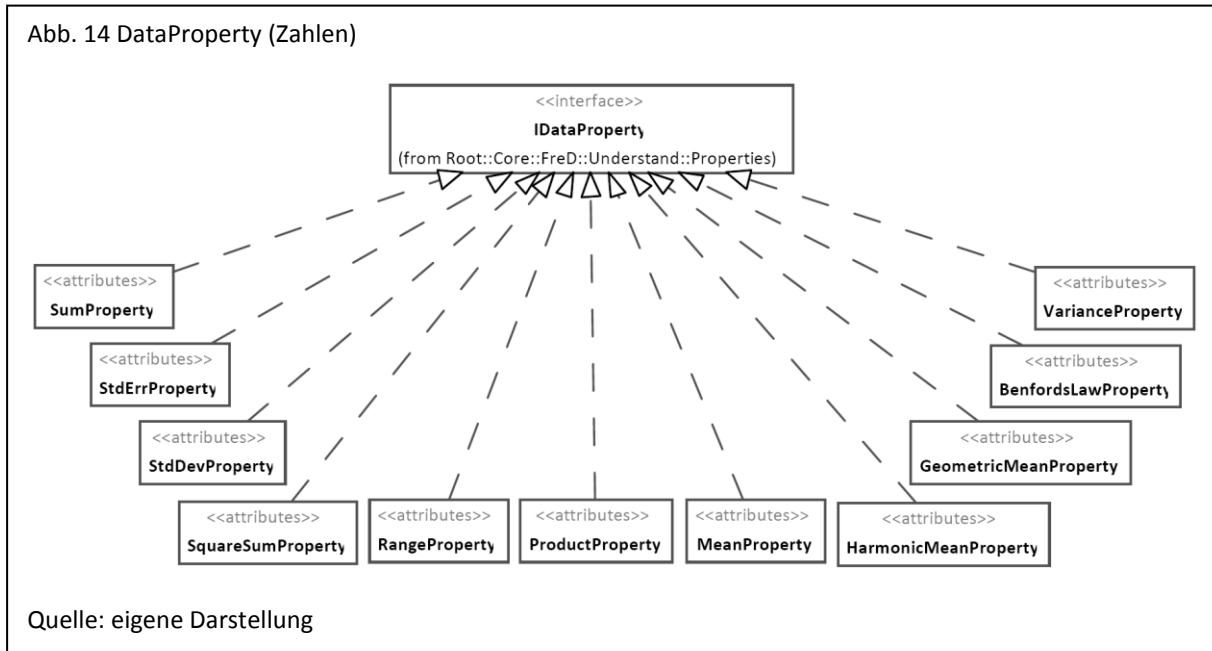
Folgende SQL-Anweisung verwendet die spezifischen Funktionen zum Ermitteln der Dateigenschaften für Text:

```

SELECT q.Understand.PropertiesMinTextLength( PostalCode ) MinTextLength,
       q.Understand.PropertiesMinTextCount( PostalCode ) MinTextCount,
       q.Understand.PropertiesMaxTextLength( PostalCode ) MaxTextLength,
       q.Understand.PropertiesMaxTextCount( PostalCode ) MaxTextCount,
       q.Understand.PropertiesBlankCount( PostalCode ) BlankCount
FROM   Person.Address
  
```

Spezielle Klassen für Zahlen berechnen vor allem statistische Kennzahlen wie Mittelwert, Standardabweichung usw. Bei Zahlen wird zusätzlich unterschieden zwischen Dateneigenschaften für Zahlen allgemein und für Fließkommazahlen. Abb. 49 zeigt alle Dateneigenschaften für Zahlen. Neben den Klassen für bekannte statistische Kennzahlen (*VarianceProperty*, *StdErrProperty*, *StdDevProperty*, *SquareSumProperty*, *MeanProperty* usw.) existiert die Klasse *BenfordsLawProperty*, die eine Überprüfung der Verteilung der einzelnen Ziffern nach Benford vornimmt.

Abb. 14 DataProperty (Zahlen)



Die folgende erste SQL-Anweisung liefert für eine Spalte alle im Framework implementierten Dateneigenschaften über die jeweiligen spezifischen SQL-Aggregatfunktionen zurück. Die zweite SQL-Anweisung errechnet den Mittelwert über die generische Aggregatfunktion, wobei die maximalen 10 und minimalen 10 Werte nicht berücksichtigt werden (benannter Parameter *TrimmedCount*):

```

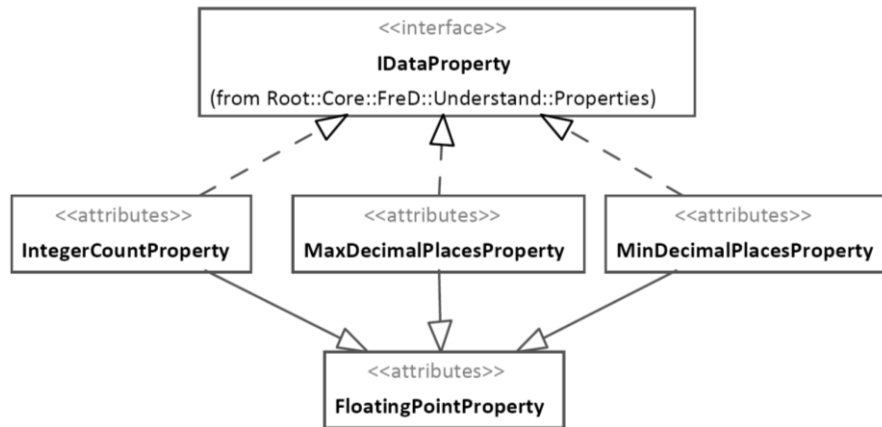
SELECT q.Understand.PropertiesBenfordsLaw( ListPrice ),
       q.Understand.PropertiesGeometricMean( ListPrice ),
       q.Understand.PropertiesHarmonicMean( ListPrice ),
       q.Understand.PropertiesMean( ListPrice ),
       q.Understand.PropertiesProduct( ListPrice ),
       q.Understand.PropertiesRange( ListPrice ),
       q.Understand.PropertiesSquareSum( ListPrice ),
       q.Understand.PropertiesStdDev( ListPrice ),
       q.Understand.PropertiesStdErr( ListPrice ),
       q.Understand.PropertiesSum( ListPrice ),
       q.Understand.PropertiesVariance( ListPrice )
FROM   Production.Product

SELECT q.Understand.PropertiesMean( ListPrice ),
       q.Understand.PropertiesDataProperty(
           '{FreD.Understand.Properties.Number.MeanProperty|Core|TrimmedCount=10}' +
           COALESCE( CAST( ListPrice AS NVARCHAR(50) ), '{NULL}' ) )
FROM   Production.Product
    
```

Neben diesen für alle Zahlen zu ermittelnden Dateneigenschaften, existieren drei Implementierungen ausschließlich für Fließkommazahlen. Über die Klasse *IntegerCountProperty* wird die Anzahl an Ganzzahlen ermittelt. *MaxDecimalPlacesProperty* und

MinDecimalPlacesProperty liefern die maximale bzw. minimale Anzahl an Nachkommastellen.

Abb. 15 DataProperty (Fließkommazahlen)



Quelle: eigene Darstellung

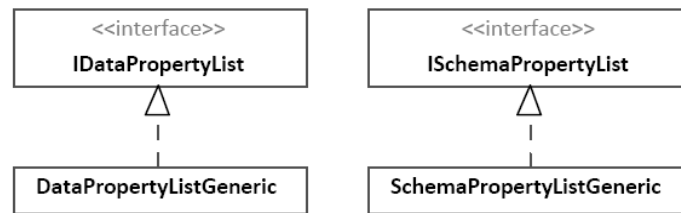
Folgende SQL-Anweisung liefert die Dateneigenschaften für Fließkommazahlen über deren spezifische Funktionen zurück:

```

SELECT q.Understand.PropertiesIntegerCount( ListPrice ),
       q.Understand.PropertiesMaxDecimalPlaces( ListPrice ),
       q.Understand.PropertiesMinDecimalPlaces( ListPrice )
FROM   Production.Product
    
```

Neben den Dateneigenschaften, die über Aggregationsfunktionen in SQL abgebildet werden, existiert eine weitere SQL-Funktion, die als Rückgabewert eine Tabelle zurückliefert. Diese Tabelle enthält alle in einer Bibliothek vorhandenen Dateneigenschaften und deren Werte für eine oder mehrere Spalten einer Tabelle. So wie es diese Funktion für Dateneigenschaften gibt, existiert eine weitere SQL-Funktion für Schemaeigenschaften.

Abb. 51 Daten- und Schemaeigenschaften



Quelle: eigene Darstellung

Klassen, die eine Tabelle mit Daten- oder Schemaeigenschaften zurückliefern, müssen die Schnittstellen *IDataPropertyList* bzw. *ISchemaPropertyList* implementieren. Die Klassenbibliothek enthält für Dateneigenschaften eine Implementierung in der Klasse *DataPropertyListGeneric* und für Schemaeigenschaften in *SchemaPropertyListGeneric*. *DataPropertyListGeneric* ermittelt in einer übergebenen Bibliothek alle Klassen, die die Schnittstelle *IDataProperty* implementieren und ermittelt für eine oder mehrere Spalten die entsprechenden Dateneigenschaften. *SchemaPropertyListGeneric* gibt alle Schemaeigenschaften für ein oder mehrere Spalten einer Tabelle über ADO.NET zurück.

Die folgende SQL-Anweisung ermittelt für die übergebenen Spalten „Color“ und „ListPrice“ alle Dateneigenschaften, die in der Bibliothek „core“ über die Schnittstelle *IDataProperty* implementiert sind.

```

SELECT *
FROM q.Understand.PropertyListDataGeneric(
    'AdventureWorks.Production.Product',
    'Color, ListPrice',
    null, 'core' )
  
```

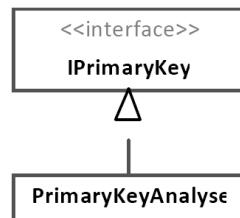
Daneben existiert noch die generische SQL-Funktion *PropertyListData*, über die eigene Implementierungen als Plug-in aufgerufen werden können. Entsprechend gibt es für Schemaeigenschaften eine SQL-Funktion *PropertyListSchema* zum generischen Aufrufen eigener Implementierungen. Das folgende SQL-Beispiel zeigt noch einmal die Verwendung der Klasse *SchemaPropertyListGeneric* über die SQL-Funktion *PropertyListSchemaGeneric*, die in diesem Beispiel für alle Spalten einer Tabelle alle ermittelten Schemaeigenschaften zurückliefert.

```
SELECT *
FROM q.Understand.PropertyListSchemaGeneric(
    'AdventureWorks.Person.Address', '*' )
```

1.3.3 Primär- und Fremdschlüssel

Klassen, die Algorithmen zur Erkennung von Primärschlüsseln in einer Tabelle einbinden, müssen die Schnittstelle *IPrimaryKey* mit der Methode *Analyse* implementieren. In der Klassenbibliothek des Frameworks ist zurzeit eine einfache Methode in der Klasse *PrimaryKeyAnalyse* implementiert, die aus einer Liste von übergebenen Spalten zunächst die ausfiltert, die aufgrund des Datentyps als Primärschlüssel nicht in Frage kommen (z.B. Fließkommazahlen). Für jede der übriggebliebenen Spalten wird die Anzahl eindeutiger Werte ermittelt. Liegt diese Anzahl über einem vorgegebenen Schwellwert, so wird diese Spalte als Primärschlüsselkandidat in einer Liste zurückgegeben. Dieses Verfahren wird für jeweils eine einzelne Spalte und für Kombinationen aus zwei Spalten durchgeführt.

Abb. 16 PrimaryKey



Quelle: eigene Darstellung

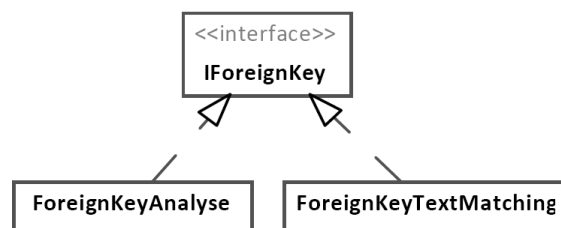
Zur Nutzung im RDBMS existiert die spezifische SQL-Funktion *PrimaryKeyAnalyse* und die generische Funktion *PrimaryKey*. Folgendes SQL-Beispiel zeigt zunächst den spezifischen Aufruf am Beispiel der Tabelle „Person.Contact“ und danach den generischen Aufruf.

```
SELECT * FROM q.Understand.PrimaryKeyAnalyse(
    'AdventureWorks.Person.Contact',
    'ContactId, Title, LastName, FirstName', null );

SELECT * FROM q.Understand.PrimaryKey(
    'AdventureWorks.Person.Contact',
    'ContactId, Title, LastName, FirstName', null,
    'FreD.Understand.Dependencies.PrimaryKeyAnalyse',
    'Core', 'Threshold=0.5' );
```

Zur Erkennung von Primär-/Fremdschlüsselbeziehungen zwischen zwei Tabellen muss eine Klasse die Schnittstelle *IForeignKey* umsetzen. In der Klassenbibliothek existieren hierzu zwei Implementierungen. Die Klasse *ForeignKeyAnalyse* überprüft die Übereinstimmung zwischen zwei Spalten instanzbasiert anhand der Dateninhalte. Die Klasse *ForeignKeyTextMatching* ist schemabasiert und verwendet einen approximativen Textvergleichsalgorithmus, um ähnliche Spaltenbezeichnungen zu erkennen.

Abb. 17 ForeignKey



Quelle: eigene Darstellung

Die folgenden SQL-Anweisungen ermitteln Primär-/Fremdschlüsselbeziehungen über die spezifischen SQL-Funktionen zu diesen beiden Klassen.

```

SELECT * FROM q.Understand.ForeignKeyAnalyse(
    'AdventureWorks.Sales.SalesOrderHeader',
    'CustomerId, SalesPersonId, TerritoryId, BillToAddressId',
    'CustomerId < 100',
    'AdventureWorks.Sales.Customer',
    'CustomerId, TerritoryId', 'CustomerId < 100' )

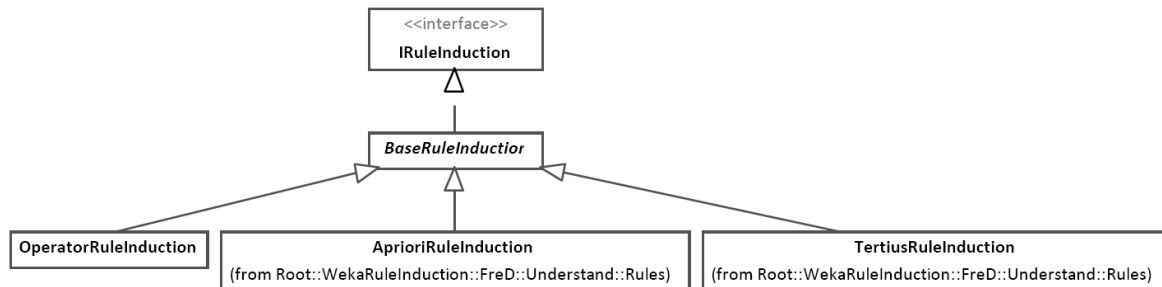
SELECT * FROM q.Understand.ForeignKeyTextMatchingAnalyse(
    'AdventureWorks.Sales.SalesOrderHeader', '*', null,
    'AdventureWorks.Sales.Customer', '*', null )
  
```

1.3.4 Regelinduktion

Zur Implementierung von Regelinduktions-Klassen muss die Schnittstelle *IRuleInduction* realisiert werden. Die in der Klassenbibliothek implementierte Klasse *OperatorRuleInduction* erkennt zwischen zwei Spalten, welcher Anteil an zwei Werten eines Datensatzes größer, kleiner oder gleich ist.

Über das Plug-in „WekaRuleInduction“ wurden die an der Universität von Waikato entwickelten Regelinduktionsverfahren „Tertius“ und „Apriori“ eingebunden.⁵

Abb. 18 RuleInduction



Quelle: eigene Darstellung

Zum Suchen von Regeln in einem Datenbestand existieren die generische SQL-Funktion *RuleInduction* und die spezifische *RuleInductionOperator*.

Die erste der folgenden drei SQL-Anweisungen erzeugt Regeln über die Funktion *RuleInductionOperator*, um für eine Tabelle Bestell-, Auslieferungs- und Fälligkeitsdaten zu überprüfen. Die zwei darauf folgenden Anweisungen verwenden die Klassen *AprioriRuleInduction* und *TertiusRuleInduction* über einen generischen SQL-Aufruf, um Regeln in einer Produkttabelle zu finden.⁶

```

SELECT "Rule", Support, Frequency
FROM q.Understand.RuleInductionOperator(
    'AdventureWorks.Sales.SalesOrderHeader',
    'OrderDate, DueDate, ShipDate', null )
    
```

```

SELECT *
FROM q.Understand.RuleInduction(
    'AdventureWorks.Production.Product',
    'Color,ProductLine, Class', null,
    'FreD.Understand.Rules.AprioriRuleInduction', 'WekaRuleInduction',
    'MinItemFrequency=3, Delta=0.05')
    
```

```

SELECT "Rule", Indicator
    
```

⁵ Siehe hierzu das in Java entwickelte Open Source Projekt „Weka“ („Waikato Environment for Knowledge Analysis“): <http://www.cs.waikato.ac.nz/ml/weka/>

⁶ Das zweite und dritte SQL-Beispiel funktioniert nur mit dem Plugin „WekaRuleInduction“

```
FROM q.Understand.RuleInduction(
    'AdventureWorks.Production.Product',
    'Color,ProductLine, Class', null,
    'FreD.Understand.Rules.TertiusRuleInduction', 'WekaRuleInduction',
    'AutoPreProcess=false')
```

1.3.5 Erweiterungen

Zur Erweiterung der Funktionalitäten müssen Plug-ins Klassen implementieren, die die entsprechenden Schnittstellen für Daten-/Schemaeigenschaften, Primär-/Fremdschlüssel-erkennung und Regelinduktion umsetzen. Um z.B. das Framework um eine eigene Dateneigenschaft zu erweitern, muss eine Klasse die Schnittstelle *IDataProperty* implementieren. Diese Schnittstelle besteht aus den drei Methoden *Process*, *Terminate*, *GetValue* und den Eigenschaften *Name* und *Description*. Da eine Dateneigenschaft ein aggregierter Wert ist, erhält die Klasse über die Methode *Process* die einzelnen Werte. *Terminate* beendet die Übergabe der Daten und über *GetValue* kann der aggregierte Wert ermittelt werden. Das folgende in C# geschriebene Plug-in ermittelt für eine Spalte mit Textwerten die Anzahl derer, die ausschließlich in Großbuchstaben gespeichert sind.

```
namespace myDataProperties
{
    [System.Serializable]
    public class UpperTextCount:FreD.Understand.Properties.IDataProperty
    {
        public string Name
        {
            get { return "Text.UpperTextCount"; }
        }

        public string Description
        {
            get { return "Count of text only in upper case."; }
        }

        public object Process( object value )
        {
            if( value != null )
            {
                string text = value.ToString();

                if( text == text.ToUpper() )
                    _count++;
            }

            return _count;
        }
    }
}
```

```

        public object Terminate()
        {
            return _count;
        }

        public object GetValue()
        {
            return _count;
        }

        private int _count = 0;
    }
}

```

Die Quellcodedatei wird über die SQL-Prozedur *AddPlugin* beim Framework registriert. Über die generische SQL-Aggregationsfunktion *PropertiesDataProperty* oder über die SQL-Funktion *PropertyListDataGeneric* kann die neue Dateneigenschaft für eine Tabellenspalte ermittelt werden. Das folgende Beispiel registriert zunächst das Plug-in und verwendet dann die beiden SQL-Funktionen, um die neue Eigenschaft für ausgewählte Spalten zu berechnen.

```

EXEC q.Tools.AddPlugin 'c:\FreD\samples\DataProperty.cs',
                      'PluginDataProperty','c:\FreD\bin\core.dll'

SELECT q.Understand.PropertiesDataProperty(
        '{myDataProperties.UpperTextCount|PluginDataProperty}' +
        COALESCE(Class, '{NULL}') )
FROM   Production.Product

SELECT *
FROM q.Understand.PropertyListDataGeneric(
        'AdventureWorks.Production.Product',
        'Color, Weight', null, 'PluginDataProperty' )

```

Um Dateneigenschaften aus mehreren Bibliotheken für ausgewählte Spalten zu ermitteln, können der Funktion *PropertyListDataGeneric* mehrere registrierte Bibliotheksnamen übergeben werden. Das folgende Beispiel gibt neben der Dateneigenschaft der Bibliothek „PluginDataProperty“ auch alle Dateneigenschaften der Bibliothek „core“ aus.

```

SELECT *
FROM   q.Understand.PropertyListDataGeneric(
        'AdventureWorks.Production.Product', '*', null,
        'PluginDataProperty|Core' )
ORDER BY TypeName

```

A SQL-Schnittstelle

A.1 Allgemein

Alle Verfahren des Datenqualitäts-Frameworks befinden sich in der Datenbank „q“. Über verschiedene Schemata und Präfixe wird eine hierarchische Abbildung der Verfahren ähnlich der von Namensräumen realisiert.

Die wesentliche Logik des Datenqualitäts-Frameworks ist in der Klassenbibliothek implementiert, deren Verfahren über generische SQL-Routinen unter Angabe des vollständigen Klassennamens oder aber über spezifische SQL-Routinen, die direkt eine gewünschte Klasse instanziiieren, umgesetzt ist.

Im Folgenden werden die gespeicherten Prozeduren und Funktionen zur Anwendung des Datenqualitäts-Frameworks für das jeweilige Schema alphabetisch geordnet beschrieben.

A.2 Date

Routine Date.Match
Typ Skalarwertfunktion

Beschreibung Errechnet die Ähnlichkeit zwischen zwei Datumswerten, indem über einen generischen Aufruf eine Instanz der übergebenen Klasse instanziiert wird, die die Schnittstelle *IMatching* implementiert hat.

Parameter

@date1 Erstes zu vergleichendes Datum.
 @date2 Zweites zu vergleichendes Datum.
 @type Vollständiger Klassenname.
 @assemblyString Vollständige Beschreibung der Identität einer Assembly. Wird nur der Name der Assembly angegeben, so werden Standardwerte für Version (0.0.0.0), Culture (neutral) und PublicKeyToken (NULL) verwendet.
 @namedParameters Benannte Parameter, die zum Setzen von Eigenschaften der Klasseninstanz verwendet werden.

Beispiel:

```
SELECT OrderDate, ShipDate,
       q.Date.Match( OrderDate, ShipDate,
                    'FreD.Date.Matching.DateMatching',
                    'Core',
                    'MaxTimeSpan=14.00:00:00' )
FROM Sales.Orders
```

qSQL

| | |
|---------------------|--|
| Routine | Date.MatchingSimple |
| Typ | Skalarwertfunktion |
| Beschreibung | Errechnet die Ähnlichkeit zwischen zwei Datumswerten, indem eine maximale Zeitspanne als Differenz übergeben wird. |
| Parameter | |
| @date1 | Erstes zu vergleichendes Datum. |
| @date2 | Zweites zu vergleichendes Datum. |
| @type | Vollständiger Klassenname. |
| @maxTimeSpan | Zeitspanne, die maximal zwischen den Datumswert liegen soll. |

Beispiel:

```
SELECT OrderDate, ShipDate,  
       q.Date.MatchingSimple( OrderDate, ShipDate, '14.00:00:00' )  
FROM Sales.Orders
```

A.3 Distance

Routine Distance.Match
Typ Skalarwertfunktion

Beschreibung Errechnet die Ähnlichkeit zwischen zwei Postleitzahlen, indem über einen generischen Aufruf eine Instanz der übergebenen Klasse instanziiert wird, die die Schnittstelle *IMatching* implementiert hat.

Parameter

@zipCode1 Erste Postleitzahl.
 @zipCode 2 Zweite Postleitzahl.
 @type Vollständiger Klassenname.
 @assemblyString Vollständige Beschreibung der Identität einer Assembly. Wird nur der Name der Assembly angegeben, so werden Standardwerte für Version (0.0.0.0), Culture (neutral) und PublicKeyToken (NULL) verwendet.
 @namedParameters Benannte Parameter, die zum Setzen von Eigenschaften der Klasseninstanz verwendet werden.

Beispiel:

```
SELECT a.PostalCode, b.PostalCode,
       q.Distance.Match( a.PostalCode, b.PostalCode,
                        'FreD.Distance.Matching.GeoDistanceMatching',
                        'Core',
                        'MaxDistance=100, Country=DE' )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Distance.Match( a.PostalCode, b.PostalCode,
                        'FreD.Distance.Matching.GeoDistanceMatching',
                        'Core',
                        'MaxDistance=100, Country=DE' ) > 0.7 AND
       a.CustomerId > b.CustomerId AND
       a.PostalCode <> b.PostalCode
GROUP BY a.PostalCode, b.PostalCode
```

Routine Distance.MatchingSimple

Typ Skalarwertfunktion

Beschreibung Errechnet die Ähnlichkeit zwischen zwei Postleitzahlen, indem über einen spezifischen Aufruf die Klasse *GeoDistanceMatching* instanziiert wird.

Parameter

| | |
|--------------|--|
| @zipCode1 | Erste Postleitzahl. |
| @zipCode 2 | Zweite Postleitzahl. |
| @country | Land, in dem die Postleitzahlen gelten. |
| @maxDistance | Maximale Distanz in Kilometern, die zwischen den Postleitzahlen liegen darf. |

Beispiel:

```
SELECT a.PostalCode, b.PostalCode,
       q.Distance.MatchingSimple(
           a.PostalCode, b.PostalCode, 'DE', 100 )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Distance.MatchingSimple(
           a.PostalCode, b.PostalCode, 'DE', 100 ) > 0.7 AND
       a.CustomerId > b.CustomerId AND a.PostalCode <> b.PostalCode
GROUP BY a.PostalCode, b.PostalCode
```

A.4 Number

Routine Number.AddPluginEncoderScript

Typ Gespeicherte Prozedur

Beschreibung Registriert ein Plug-in als Encoder für Zahlen beim Datenqualitäts-Framework, das über die generischen Funktionen verwendet werden kann.

Parameter

@script Quellcode in Java, C# oder VB. Falls NULL, wird ein einfaches Standardskript verwendet.

@assemblyName Name, unter dem das Plug-in als Assembly im RDBMS gespeichert werden soll.

Beispiel:

```
EXEC q.Number.AddPluginEncoderScript
    ,
    string code = "";
    foreach( char c in number )
    {
        if( char.IsDigit( c ) )
            code += '9';
        else
            code += c;
    }
    return code;
    ,
    'Plugin_NumberEncoder'
```

```
SELECT q.Number.Encode(
    ListPrice,
    'Quality.Number.Encoder.Encoder',
    'Plugin_NumberEncoder',
    NULL ),
    COUNT(*)
FROM Sales.Articles
GROUP BY q.Number.Encode(
    ListPrice,
    'Quality.Number.Encoder.Encoder',
    'Plugin_NumberEncoder',
    NULL )
```

Routine Number.AddPluginMatchingScript

Typ Gespeicherte Prozedur

Beschreibung Registriert ein Plug-in als Vergleichsfunktion für Zahlen beim Datenqualitäts-Framework, das über die generischen Funktionen verwendet werden kann.

Parameter

@script Quellcode in Java, C# oder VB. Falls NULL, wird ein einfaches Standardskript verwendet.

@assemblyName Name, unter dem das Plug-in als Assembly im RDBMS gespeichert werden soll.

Beispiel:

```
EXEC q.Number.AddPluginMatchingScript
    ,
    double num1 = Convert.ToDouble( number1 );
    double num2 = Convert.ToDouble( number2 );

    if( Math.Max( num1, num2 ) == 0.0 ) return 0.0;

    return 1.0 - ( Math.Abs( num1 - num2 ) /
                  Math.Max( num1, num2 ) );
    ,
    'Plugin_NumberMatching'

SELECT Stock, StockReorderPoint,
       q.Number.Match(
           Stock, StockReorderPoint,
           'Quality.Number.Matching.Matching',
           'Plugin_NumberMatching',
           NULL )
FROM   Sales.Articles
```

| | |
|---------------------|--|
| Routine | Number.CheckDigitsDecode |
| Typ | Skalarwertfunktion |
| Beschreibung | Überprüft die letzte Ziffer einer übergebenen Ziffernfolge auf ihre Gültigkeit als Prüfziffer (Default: Quersumme mit Modulo 10). Ist die Prüfziffer ungültig, wird eine Ausnahme erzeugt, ansonsten die Ziffernfolge ohne die Prüfziffer zurückgeliefert. |
| Parameter | |
| @text | Ziffernfolge mit Prüfziffer. |

Beispiel:

```
SELECT q.Number.CheckDigitsDecode( '12340' )  
SELECT q.Number.CheckDigitsDecode( '12341' )
```

qSQL

| | |
|---------------------|---|
| Routine | Number.CheckDigitsEncode |
| Typ | Skalarwertfunktion |
| Beschreibung | Errechnet eine Prüfziffer für eine übergebene Ziffernfolge (Default: Quersumme mit Modulo 10) und gibt die vollständige Ziffernfolge mit Prüfziffer zurück. |
| Parameter | |
| @text | Ziffernfolge ohne Prüfziffer. |

Beispiel:

```
SELECT DISTINCT ProductNumber,  
               q.Number.CheckDigitsEncode( ProductNumber )  
FROM   Sales.Articles
```


qSQL

| | |
|---------------------|---|
| Routine | Number.CheckDigitsVerhoeffDecode |
| Typ | Skalarwertfunktion |
| Beschreibung | Überprüft die letzte Ziffer einer übergebenen Ziffernfolge auf ihre Gültigkeit als Prüfziffer nach dem Verhoeff-Algorithmus. Ist die Prüfziffer ungültig, wird eine Ausnahme erzeugt, ansonsten die Ziffernfolge ohne die Prüfziffer zurückgeliefert. |
| Parameter | |
| @text | Ziffernfolge mit Prüfziffer. |

Beispiel:

```
SELECT q.Number.CheckDigitsVerhoeffDecode( '12342' )
SELECT q.Number.CheckDigitsVerhoeffDecode( '12341' )
```

| | |
|---------------------|--|
| Routine | Number.CheckDigitsVerhoeffEncode |
| Typ | Skalarwertfunktion |
| Beschreibung | Errechnet eine Prüfziffer für eine übergebene Ziffernfolge nach dem Verhoeff-Algorithmus und gibt die vollständige Ziffernfolge mit Prüfziffer zurück. |
| Parameter | |
| @text | Ziffernfolge ohne Prüfziffer. |

Beispiel:

```
SELECT DISTINCT ProductNumber,  
       q.Number.CheckDigitsVerhoeffEncode( ProductNumber )  
FROM   Sales.Articles
```

qSQL

| | |
|---------------------|---|
| Routine | Number.DecimalPlacesEncode |
| Typ | Skalarwertfunktion |
| Beschreibung | Ermittelt die Anzahl der Nachkommastellen von Fließkommazahlen. |
| Parameter | |
| @number | Zahl. |

Beispiel:

```
SELECT DISTINCT ListPrice, q.Number.DecimalPlacesEncode( ListPrice )
FROM Sales.Articles
```

| | |
|---------------------|---|
| Routine | Number.Decode |
| Typ | Skalarwertfunktion |
| Beschreibung | Generische Funktion zum Verwenden von Klassen, die die Kodierung einer Zahl überprüfen. |
| Parameter | |
| @text | Zahl. |
| @type | Vollständiger Klassenname. |
| @assemblyString | Vollständige Beschreibung der Identität einer Assembly. Wird nur der Name der Assembly angegeben, so werden Standardwerte für Version (0.0.0.0), Culture (neutral) und PublicKeyToken (NULL) verwendet. |
| @namedParameters | Benannte Parameter, die zum Setzen von Eigenschaften der Klasseninstanz verwendet werden. |

Beispiel:

```
SELECT q.Number.Decode( -- Prüfziffernverfahren ISBN-10
    '123456789X',
    'FreD.Number.Encoder.CheckDigitsEncoder',
    'Core',
    'Weights=10|9|8|7|6|5|4|3|2, Modulo=11, Mappings=10:X' )
```

| | |
|---------------------|---|
| Routine | Number.Encode |
| Typ | Skalarwertfunktion |
| Beschreibung | Generische Funktion zum Verwenden von Klassen, die Zahlen in einen Code umwandeln. |
| Parameter | |
| @text | Zahl |
| @type | Vollständiger Klassenname. |
| @assemblyString | Vollständige Beschreibung der Identität einer Assembly. Wird nur der Name der Assembly angegeben, so werden Standardwerte für Version (0.0.0.0), Culture (neutral) und PublicKeyToken (NULL) verwendet. |
| @namedParameters | Benannte Parameter, die zum Setzen von Eigenschaften der Klasseninstanz verwendet werden. |

Beispiel:

```
SELECT DISTINCT ProductNumber,
               q.Number.Encode( -- Prüfziffernverfahren ISBN-10
                               ProductNumber,
                               'FreD.Number.Encoder.CheckDigitsEncoder',
                               'Core',
                               'Weights=10|9|8|7|6|5|4|3|2, Modulo=11, Mappings=10:X' )
FROM   Sales.Articles
```

| | |
|---------------------|---|
| Routine | Number.Match |
| Typ | Skalarwertfunktion |
| Beschreibung | Generische Funktion zum Verwenden von Klassen, die zwei Zahlen auf Ähnlichkeit überprüfen und einen Indikator hierfür zurückliefern. |
| Parameter | |
| @number1 | Erste Zahl. |
| @number2 | Zweite Zahl. |
| @type | Vollständiger Klassenname. |
| @assemblyString | Vollständige Beschreibung der Identität einer Assembly. Wird nur der Name der Assembly angegeben, so werden Standardwerte für Version (0.0.0.0), Culture (neutral) und PublicKeyToken (NULL) verwendet. |
| @namedParameters | Benannte Parameter, die zum Setzen von Eigenschaften der Klasseninstanz verwendet werden. |

Beispiel:

```
SELECT DISTINCT Stock, StockReorderPoint,
    q.Number.Match(
        Stock, StockReorderPoint,
        'FreD.Number.Matching.SimpleNumberMatching',
        'Core',
        'MaxDistance=100' )
FROM Sales.Articles
```

Routine Number.MatchingEuclidian

Typ Skalarwertfunktion

Beschreibung Berechnet auf Grundlage der Euklidischen Distanz zwischen zwei Zahlen einen Ähnlichkeitswert.

Parameter

| | |
|----------|--------------|
| @number1 | Erste Zahl. |
| @number2 | Zweite Zahl. |

Beispiel:

```
SELECT DISTINCT Stock, StockReorderPoint,
               q.Number.MatchingEuclidian( Stock, StockReorderPoint )
FROM Sales.Articles
```

qSQL

Routine Number.MatchingSimple

Typ Skalarwertfunktion

Beschreibung Berechnet auf Grundlage eines maximalen Distanzwertes (Default: 100) zwischen zwei Zahlen einen Ähnlichkeitswert.

Parameter

| | |
|----------|--------------|
| @number1 | Erste Zahl. |
| @number2 | Zweite Zahl. |

Beispiel:

```
SELECT DISTINCT Stock, StockReorderPoint,
               q.Number.MatchingSimple( Stock, StockReorderPoint )
FROM Sales.Articles
```


A.5 Text

Routine Text.AddPluginHashingScript

Typ Gespeicherte Prozedur

Beschreibung Registriert ein Plug-in als Hashing Encoder für Text beim Datenqualitäts-Framework, das über die generischen Funktionen verwendet werden kann.

Parameter

@script Quellcode in Java, C# oder VB. Falls NULL, wird ein einfaches Standardskript verwendet.

@assemblyName Name, unter dem das Plug-in als Assembly im RDBMS gespeichert werden soll.

Beispiel:

```
EXEC q.Text.AddPluginHashingScript
    '
        string code = "";
        int sum = 0;

        text = text.ToUpper();

        // Quersumme der ersten 3 Alphazeichen berechnen
        int maxLen = 3;
        for( int i = 0;
            i < System.Math.Min( text.Length, maxLen ); i++ )
        {
            if( char.IsLetter( text[ i ] ) )
                sum += ( int )text[ i ] - ( int )'A' + 1;
            else
                maxLen++;
        }
        // Modulo 99 zur Quersumme berechnen
        return ( ( int )( sum % 99 ) ).ToString();
    ',
    'Plugin_TextHashingEncoder'
```

```
SELECT Name,
       q.Text.Encode(
           Name,
           'Quality.Text.Encoder.Encoder',
           'Plugin_TextHashingEncoder',
           NULL )
FROM   Sales.Articles
ORDER BY 2
```

Routine Text.AddPluginMatchingScript

Typ Gespeicherte Prozedur

Beschreibung Registriert ein Plug-in als Vergleichsfunktion für Text beim Datenqualitäts-Framework, das über die generischen Funktionen verwendet werden kann.

Parameter

@script Quellcode in Java, C# oder VB. Falls NULL, wird ein einfaches Standardskript verwendet.

@assemblyName Name, unter dem das Plug-in als Assembly im RDBMS gespeichert werden soll.

Beispiel:

```
EXEC q.Text.AddPluginMatchingScript
    'return text1.IndexOf( text2 ) >= 0 ? 1.0 : 0.0;',
    'Plugin_TextMatching'
```

```
SELECT a.CustomerId, a.Name, b.CustomerId, b.Name
FROM   Persons.Customers a, Persons.Customers b
WHERE  a.CustomerId > b.CustomerId AND
       a.Name <> b.Name AND
       q.Text.Match(
           a.Name, b.Name,
           'Quality.Text.Matching.Matching',
           'Plugin_TextMatching',
           NULL ) = 1.0
```

qSQL

| | |
|---------------------|--|
| Routine | Text.AddPluginPatternScript |
| Typ | Gespeicherte Prozedur |
| Beschreibung | Registriert ein Plug-in als Pattern Encoder für Text beim Datenqualitäts-Framework, das über die generischen Funktionen verwendet werden kann. |
| Parameter | |
| @script | Quellcode in Java, C# oder VB. Falls NULL, wird ein einfaches Standardskript verwendet. |
| @assemblyName | Name, unter dem das Plug-in als Assembly im RDBMS gespeichert werden soll. |

Beispiel:

```
EXEC q.Text.AddPluginPatternScript
    ,
    string code = "";

    foreach( char c in text )
    {
        if( char.IsLetter( c ) )
            code += 'X';
        else if( char.IsDigit( c ) )
            code += '9';
        else code += char.ToUpper( c );
    }

    return code;
    ,
    'Plugin_TextPatternEncoder'
```

```
SELECT q.Text.Encode(
    PostalCode,
    'Quality.Text.Encoder.Encoder',
    'Plugin_TextPatternEncoder',
    NULL ),
COUNT( * )
FROM Persons.Customers
GROUP BY q.Text.Encode(
    PostalCode,
    'Quality.Text.Encoder.Encoder',
    'Plugin_TextPatternEncoder',
    NULL )
```

Routine Text.AddPluginPhoneticScript

Typ Gespeicherte Prozedur

Beschreibung Registriert ein Plug-in als Phonetic Encoder für Text beim Datenqualitäts-Framework, das über die generischen Funktionen verwendet werden kann.

Parameter

@script Quellcode in Java, C# oder VB. Falls NULL, wird ein einfaches Standardskript verwendet.

@assemblyName Name, unter dem das Plug-in als Assembly im RDBMS gespeichert werden soll.

Beispiel:

```
EXEC q.Text.AddPluginPhoneticScript
    ,
    text = text.ToUpper();
    text = text.Replace( "A", "" );
    text = text.Replace( "E", "" );
    text = text.Replace( "I", "" );
    text = text.Replace( "O", "" );
    text = text.Replace( "U", "" );
    text = text.Replace( " ", "" );
    return text.Substring( 0,
        System.Math.Min( 5, text.Length ) );
    ,
    'Plugin_TextPhoneticEncoder'
```

```
SELECT Name, q.Text.Encode(
    "Name",
    'Quality.Text.Encoder.Encoder',
    'Plugin_TextPhoneticEncoder',
    NULL )
FROM Persons.Customers
ORDER BY 1
```

qSQL

| | |
|---------------------|---|
| Routine | Text.AddPluginPreparerScript |
| Typ | Gespeicherte Prozedur |
| Beschreibung | Registriert ein Plug-in als Preparer Encoder für Text beim Datenqualitäts-Framework, das über die generischen Funktionen verwendet werden kann. |
| Parameter | |
| @script | Quellcode in Java, C# oder VB. Falls NULL, wird ein einfaches Standardskript verwendet. |
| @assemblyName | Name, unter dem das Plug-in als Assembly im RDBMS gespeichert werden soll. |

Beispiel:

```
EXEC q.Text.AddPluginPreparerScript
    'return text.ToUpper();',
    'Plugin_TextPreparerEncoder'

SELECT q.Text.Encode(
    City,
    'Quality.Text.Encoder.Encoder',
    'Plugin_TextPreparerEncoder',
    NULL )
FROM   Persons.Customers
```

Routine Text.AddPluginTokenizerScript

Typ Gespeicherte Prozedur

Beschreibung Registriert ein Plug-in als Tokenizer für Text beim Datenqualitäts-Framework, das über die generischen Funktionen verwendet werden kann.

Parameter

@script Quellcode in Java, C# oder VB. Falls NULL, wird ein einfaches Standardskript verwendet.

@assemblyName Name, unter dem das Plug-in als Assembly im RDBMS gespeichert werden soll.

Beispiel:

```
EXEC q.Text.AddPluginTokenizerScript
    'return text.Split( new char[] { ' ' ', ';' },
        System.StringSplitOptions.RemoveEmptyEntries );',
    'Plugin_TextTokenizer'
```

```
SELECT q.Text.Tokenize(
    Name,
    ' | ',
    'Quality.Text.Tokenizer.Tokenizer',
    'Plugin_TextTokenizer',
    NULL )
FROM Persons.Customers
```

qSQL

| | |
|---------------------|---|
| Routine | Text.CodeListEncode |
| Typ | Skalarwertfunktion |
| Beschreibung | Sequentielle Verknüpfung mehrere Encoder, die einen gemeinsamen Code erzeugen. |
| Parameter | |
| @text | Zu kodierender Text. |
| @assemblyString | Vollständige Beschreibung der Identität einer Assembly. Wird nur der Name der Assembly angegeben, so werden Standardwerte für Version (0.0.0.0), Culture (neutral) und PublicKeyToken (NULL) verwendet. |
| @typesString | List von vollständigen Klassennamen, die Encoder implementieren. |

Beispiel:

```

SELECT Name,
       q.Text.CodeListEncode (
           Name,
           'Core',
           'FreD.Text.Encoder.CharSortPatternEncoder|
           FreD.Text.Encoder.DoubleMetaphonePhoneticEncoder|
           FreD.Text.Encoder.SoundexPhoneticEncoder' )
FROM   Persons.Customers

```

| | |
|---------------------|---|
| Routine | Text.Encode |
| Typ | Skalarwertfunktion |
| Beschreibung | Generische Funktion zum Verwenden von Klassen, die Text in einen Code umwandeln. |
| Parameter | |
| @text | Text. |
| @type | Vollständiger Klassenname. |
| @assemblyString | Vollständige Beschreibung der Identität einer Assembly. Wird nur der Name der Assembly angegeben, so werden Standardwerte für Version (0.0.0.0), Culture (neutral) und PublicKeyToken (NULL) verwendet. |
| @namedParameters | Benannte Parameter, die zum Setzen von Eigenschaften der Klasseninstanz verwendet werden. |

Beispiel:

```
SELECT Name, q.Text.Encode (
    Name,
    'FreD.Text.Encoder.CharSortPatternEncoder',
    'Core', 'IgnoreMultipleChar=true' )
FROM Persons.Customers
ORDER BY 2
```


qSQL

| | |
|---------------------|---|
| Routine | Text.HashingAdler32 |
| Typ | Skalarwertfunktion |
| Beschreibung | Berechnet für den übergebenen Wert den Hashwert nach dem Adler32-Algorithmus. |

Parameter

| | |
|-------|-------|
| @text | Text. |
|-------|-------|

Beispiel:

```
SELECT Name, q.Text.HashingAdler32( Name )  
FROM   Persons.Customers  
ORDER BY 2
```

qSQL

Routine Text.HashingSum

Typ Skalarwertfunktion

Beschreibung Berechnet für den übergebenen Wert den Hashwert, indem die Quersumme gebildet wird.

Parameter

@text Text.

Beispiel:

```
SELECT Name, q.Text.HashingSum( Name )
FROM   Persons.Customers
ORDER BY 2
```

Routine Text.Match
Typ Skalarwertfunktion

Beschreibung Generische Funktion zum Verwenden von Klassen, die zwei Texte auf Ähnlichkeit überprüfen und einen Indikator hierfür zurückliefern.

Parameter

@text1 Erster Text.
@text2 Zweiter Text.
@type Vollständiger Klassenname.
@assemblyString Vollständige Beschreibung der Identität einer Assembly. Wird nur der Name der Assembly angegeben, so werden Standardwerte für Version (0.0.0.0), Culture (neutral) und PublicKeyToken (NULL) verwendet.
@namedParameters Benannte Parameter, die zum Setzen von Eigenschaften der Klasseninstanz verwendet werden.

Beispiel:⁷

```
SELECT a.Name, b.Name,
       q.Text.Match(
           a.Name, b.Name,
           'FreD.Text.Matching.CompressApproximateMatching',
           'CompressApproximateMatching',
           'DeflateAlgorithm=false' )
FROM   Persons.Customers a, Persons.Customers b
WHERE  a.CustomerId > b.CustomerId
ORDER BY 3 DESC
```

⁷ Das SQL-Beispiel funktioniert nur mit dem Plugin „CompressApproximateMatching“

qSQL

| | |
|---------------------|---|
| Routine | Text.MatchingExact |
| Typ | Skalarwertfunktion |
| Beschreibung | Überprüft zwei Texte auf Gleichheit und liefert bei Übereinstimmung einen Wert von 1, sonst 0 zurück. |
| Parameter | |
| @text1 | Erster Text. |
| @text2 | Zweiter Text. |

Beispiel:

```
SELECT a.Name, b.Name
FROM   Persons.Customers a, Persons.Customers b
WHERE  a.CustomerId > b.CustomerId AND
       q.Text.MatchingExact( a.Name, b.Name ) > 0
```

| | |
|---------------------|---|
| Routine | Text.MatchingHamming |
| Typ | Skalarwertfunktion |
| Beschreibung | Berechnet über die Hamming-Distanz zeichenbasiert die Ähnlichkeit zwischen zwei Zeichenketten und liefert einen Wert zwischen 0 und 1 zurück. |
| Parameter | |
| @text1 | Erster Text. |
| @text2 | Zweiter Text. |

Beispiel:

```
SELECT a.Name, b.Name, q.Text.MatchingHamming( a.Name, b.Name )
FROM   Persons.Customers a, Persons.Customers b
WHERE  a.CustomerId > b.CustomerId AND
       q.Text.MatchingHamming( a.Name, b.Name ) > 0.8
ORDER BY 3
```

qSQL

Routine Text.MatchingHirschberg

Typ Skalarwertfunktion

Beschreibung Berechnet über den Hirschberg-Algorithmus zeichenbasiert die Ähnlichkeit zwischen zwei Zeichenketten und liefert einen Wert zwischen 0 und 1 zurück.

Parameter

@text1 Erster Text.

@text2 Zweiter Text.

Beispiel:

```
SELECT a.Name, b.Name, q.Text.MatchingHirschberg( a.Name, b.Name )
FROM   Persons.Customers a, Persons.Customers b
WHERE  a.CustomerId > b.CustomerId AND
       q.Text.MatchingHirschberg( a.Name, b.Name ) > 0.8
ORDER BY 3
```

qSQL

| | |
|---------------------|--|
| Routine | Text.MatchingJaro |
| Typ | Skalarwertfunktion |
| Beschreibung | Berechnet über den Jaro-Algorithmus zeichenbasiert die Ähnlichkeit zwischen zwei Zeichenketten und liefert einen Wert zwischen 0 und 1 zurück. |
| Parameter | |
| @text1 | Erster Text. |
| @text2 | Zweiter Text. |

Beispiel:

```
SELECT a.Name, b.Name, q.Text.MatchingJaro( a.Name, b.Name )
FROM   Persons.Customers a, Persons.Customers b
WHERE  a.CustomerId > b.CustomerId AND
       q.Text.MatchingJaro( a.Name, b.Name ) > 0.8
ORDER BY 3
```

qSQL

| | |
|---------------------|--|
| Routine | Text.MatchingJaroWinkler |
| Typ | Skalarwertfunktion |
| Beschreibung | Berechnet über den Jaro-Winkler-Algorithmus zeichenbasiert die Ähnlichkeit zwischen zwei Zeichenketten und liefert einen Wert zwischen 0 und 1 zurück. |
| Parameter | |
| @text1 | Erster Text. |
| @text2 | Zweiter Text. |

Beispiel:

```
SELECT a.Name, b.Name, q.Text.MatchingJaroWinkler( a.Name, b.Name )
FROM   Persons.Customers a, Persons.Customers b
WHERE  a.CustomerId > b.CustomerId AND
       q.Text.MatchingJaroWinkler( a.Name, b.Name ) > 0.8
ORDER BY 3
```


| | |
|---------------------|--|
| Routine | Text.MatchingJaroWinklerLynch |
| Typ | Skalarwertfunktion |
| Beschreibung | Berechnet über den Jaro-Winkler-Lynch-Algorithmus zeichenbasiert die Ähnlichkeit zwischen zwei Zeichenketten und liefert einen Wert zwischen 0 und 1 zurück. |
| Parameter | |
| @text1 | Erster Text. |
| @text2 | Zweiter Text. |

Beispiel:

```
SELECT a.Name, b.Name, q.Text.MatchingJaroWinklerLynch( a.Name, b.Name )
FROM   Persons.Customers a, Persons.Customers b
WHERE  a.CustomerId > b.CustomerId AND
       q.Text.MatchingJaroWinklerLynch( a.Name, b.Name ) > 0.8
ORDER BY 3
```

| | |
|---------------------|---|
| Routine | Text.MatchingLevenshtein |
| Typ | Skalarwertfunktion |
| Beschreibung | Berechnet über die Damerau-Levenshtein-Distanz zeichenbasiert die Ähnlichkeit zwischen zwei Zeichenketten und liefert einen Wert zwischen 0 und 1 zurück. |
| Parameter | |
| @text1 | Erster Text. |
| @text2 | Zweiter Text. |

Beispiel:

```
SELECT a.Name, b.Name, q.Text.MatchingLevenshtein( a.Name, b.Name )
FROM   Persons.Customers a, Persons.Customers b
WHERE  a.CustomerId > b.CustomerId AND
       q.Text.MatchingLevenshtein( a.Name, b.Name ) > 0.8
ORDER BY 3
```

| | |
|---------------------|--|
| Routine | Text.MatchingLevenshteinHjelmqvist |
| Typ | Skalarwertfunktion |
| Beschreibung | Berechnet über die Damerau-Levenshtein-Distanz zeichenbasiert die Ähnlichkeit zwischen zwei Zeichenketten über den Algorithmus von S. Hjelmqvist und liefert einen Wert zwischen 0 und 1 zurück. |
| Parameter | |
| @text1 | Erster Text. |
| @text2 | Zweiter Text. |

Beispiel:

```
SELECT a.Name, b.Name,
       q.Text.MatchingLevenshteinHjelmqvist( a.Name, b.Name )
FROM   Persons.Customers a, Persons.Customers b
WHERE  a.CustomerId > b.CustomerId AND
       q.Text.MatchingLevenshteinHjelmqvist ( a.Name, b.Name ) > 0.8
ORDER BY 3
```

Routine Text.MatchingRatcliffObershelp

Typ Skalarwertfunktion

Beschreibung Berechnet über den Ratcliff-Obershelp-Algorithmus zeichenbasiert die Ähnlichkeit zwischen zwei Zeichenketten und liefert einen Wert zwischen 0 und 1 zurück.

Parameter

@text1 Erster Text.

@text2 Zweiter Text.

Beispiel:⁸

```
SELECT a.Name, b.Name,
       q.Text.MatchingRatcliffObershelp( a.Name, b.Name )
FROM   Persons.Customers a, Persons.Customers b
WHERE  a.CustomerId > b.CustomerId AND
       q.Text.MatchingRatcliffObershelp ( a.Name, b.Name ) > 0.8
ORDER BY 3
```

⁸ Das SQL-Beispiel funktioniert nur mit der DLL „RatcliffObershelpApproximateMatching_SQLExtendedProc.dll“

| | |
|---------------------|--|
| Routine | Text.MatchingSift3 |
| Typ | Skalarwertfunktion |
| Beschreibung | Berechnet zeichenbasiert über den „Longest Common Substring“ die Ähnlichkeit zwischen zwei Zeichenketten und liefert einen Wert zwischen 0 und 1 zurück. |
| Parameter | |
| @text1 | Erster Text. |
| @text2 | Zweiter Text. |

Beispiel:

```
SELECT a.Name, b.Name,  
       q.Text.MatchingSift3( a.Name, b.Name )  
FROM   Persons.Customers a, Persons.Customers b  
WHERE  a.CustomerId > b.CustomerId AND  
       q.Text.MatchingSift3( a.Name, b.Name ) > 0.8  
ORDER BY 3
```

Routine Text.MatchingTokenCityBlock

Typ Skalarwertfunktion

Beschreibung Berechnet über die City-Block-Distanz tokenbasiert die Ähnlichkeit zwischen zwei Zeichenketten und liefert einen Wert zwischen 0 und 1 zurück.

Parameter

@text1 Erster Text.

@text2 Zweiter Text.

Beispiel:

```
SELECT a.Name + ' ' + a.City, b.Name + ' ' + b.City,
       q.Text.MatchingTokenCityBlock( a.Name + ' ' + a.City,
                                      b.Name + ' ' + b.City )
FROM   Persons.Customers a, Persons.Customers b
WHERE  a.CustomerId > b.CustomerId AND
       q.Text.MatchingTokenCityBlock( a.Name + ' ' + a.City,
                                      b.Name + ' ' + b.City ) > 0.5
ORDER BY 3
```

Routine Text.MatchingTokenCosine

Typ Skalarwertfunktion

Beschreibung Berechnet über das Kosinus-Maß tokenbasiert die Ähnlichkeit zwischen zwei Zeichenketten und liefert einen Wert zwischen 0 und 1 zurück.

Parameter

@text1 Erster Text.

@text2 Zweiter Text.

Beispiel:

```
SELECT a.Name + ' ' + a.City, b.Name + ' ' + b.City,
       q.Text.MatchingTokenCosine( a.Name + ' ' + a.City,
                                   b.Name + ' ' + b.City )
FROM   Persons.Customers a, Persons.Customers b
WHERE  a.CustomerId > b.CustomerId AND
       q.Text.MatchingTokenCosine( a.Name + ' ' + a.City,
                                   b.Name + ' ' + b.City ) > 0.5
ORDER BY 3
```

qSQL

Routine Text.MatchingTokenDice

Typ Skalarwertfunktion

Beschreibung Berechnet über das Dice-Maß tokenbasiert die Ähnlichkeit zwischen zwei Zeichenketten und liefert einen Wert zwischen 0 und 1 zurück.

Parameter

@text1 Erster Text.

@text2 Zweiter Text.

Beispiel:

```
SELECT a.Name + ' ' + a.City, b.Name + ' ' + b.City,
       q.Text.MatchingTokenDice( a.Name + ' ' + a.City,
                                b.Name + ' ' + b.City )
FROM   Persons.Customers a, Persons.Customers b
WHERE  a.CustomerId > b.CustomerId AND
       q.Text.MatchingTokenDice( a.Name + ' ' + a.City,
                                b.Name + ' ' + b.City ) > 0.5
ORDER BY 3
```


Routine Text.MatchingTokenEuclidian

Typ Skalarwertfunktion

Beschreibung Berechnet über die Euklidische Distanz tokenbasiert die Ähnlichkeit zwischen zwei Zeichenketten und liefert einen Wert zwischen 0 und 1 zurück.

Parameter

@text1 Erster Text.

@text2 Zweiter Text.

Beispiel:

```
SELECT a.Name + ' ' + a.City, b.Name + ' ' + b.City,  
       q.Text.MatchingTokenEuclidian( a.Name + ' ' + a.City,  
                                      b.Name + ' ' + b.City )  
FROM   Persons.Customers a, Persons.Customers b  
WHERE  a.CustomerId > b.CustomerId AND  
       q.Text.MatchingTokenEuclidian( a.Name + ' ' + a.City,  
                                      b.Name + ' ' + b.City ) > 0.3  
ORDER BY 3
```

Routine Text.MatchingTokenFellegiSunter

Typ Skalarwertfunktion

Beschreibung Berechnet über den Fellegi & Sunter-Algorithmus tokenbasiert die Ähnlichkeit zwischen zwei Zeichenketten und liefert einen Wert zwischen 0 und 1 zurück.

Parameter

@text1 Erster Text.

@text2 Zweiter Text.

Beispiel:

```
SELECT a.Name + ' ' + a.City, b.Name + ' ' + b.City,  
       q.Text.MatchingTokenFellegiSunter( a.Name + ' ' + a.City,  
                                           b.Name + ' ' + b.City )  
FROM   Persons.Customers a, Persons.Customers b  
WHERE  a.CustomerId > b.CustomerId AND  
       q.Text.MatchingTokenFellegiSunter( a.Name + ' ' + a.City,  
                                           b.Name + ' ' + b.City ) > 8  
ORDER BY 3
```

qSQL

Routine Text.MatchingTokenJaccard

Typ Skalarwertfunktion

Beschreibung Berechnet über das Jaccard-Maß tokenbasiert die Ähnlichkeit zwischen zwei Zeichenketten und liefert einen Wert zwischen 0 und 1 zurück.

Parameter

@text1 Erster Text.

@text2 Zweiter Text.

Beispiel:

```
SELECT a.Name + ' ' + a.City, b.Name + ' ' + b.City,  
       q.Text.MatchingTokenJaccard( a.Name + ' ' + a.City,  
                                   b.Name + ' ' + b.City )  
FROM   Persons.Customers a, Persons.Customers b  
WHERE  a.CustomerId > b.CustomerId AND  
       q.Text.MatchingTokenJaccard( a.Name + ' ' + a.City,  
                                   b.Name + ' ' + b.City ) > 0.5  
ORDER BY 3
```

Routine Text.MatchingTokenMatchingCoefficient

Typ Skalarwertfunktion

Beschreibung Berechnet über das Matching Coefficient Maß tokenbasiert die Ähnlichkeit zwischen zwei Zeichenketten und liefert einen Wert zwischen 0 und 1 zurück.

Parameter

@text1 Erster Text.

@text2 Zweiter Text.

Beispiel:

```
SELECT a.Name + ' ' + a.City, b.Name + ' ' + b.City,
       q.Text.MatchingTokenMatchingCoefficient( a.Name + ' ' + a.City,
                                                b.Name + ' ' + b.City )
FROM   Persons.Customers a, Persons.Customers b
WHERE  a.CustomerId > b.CustomerId AND
       q.Text.MatchingTokenMatchingCoefficient( a.Name + ' ' + a.City,
                                                b.Name + ' ' + b.City )
       > 0.5
ORDER BY 3
```

qSQL

Routine Text.MatchingTokenOverlap

Typ Skalarwertfunktion

Beschreibung Berechnet über das Overlap-Maß tokenbasiert die Ähnlichkeit zwischen zwei Zeichenketten und liefert einen Wert zwischen 0 und 1 zurück.

Parameter

@text1 Erster Text.

@text2 Zweiter Text.

Beispiel:

```
SELECT a.Name + ' ' + a.City, b.Name + ' ' + b.City,  
       q.Text.MatchingTokenOverlap( a.Name + ' ' + a.City,  
                                   b.Name + ' ' + b.City )  
FROM   Persons.Customers a, Persons.Customers b  
WHERE  a.CustomerId > b.CustomerId AND  
       q.Text.MatchingTokenOverlap( a.Name + ' ' + a.City,  
                                   b.Name + ' ' + b.City ) > 0.5  
ORDER BY 3
```

qSQL

Routine Text.MatchingTokenTFIDF

Typ Skalarwertfunktion

Beschreibung Berechnet über den TF-IDF-Algorithmus tokenbasiert die Ähnlichkeit zwischen zwei Zeichenketten und liefert einen Wert zwischen 0 und 1 zurück.

Parameter

@text1 Erster Text.

@text2 Zweiter Text.

Beispiel:

```
SELECT a.Name + ' ' + a.City, b.Name + ' ' + b.City,
       q.Text.MatchingTokenTFIDF( a.Name + ' ' + a.City,
                                   b.Name + ' ' + b.City )
FROM   Persons.Customers a, Persons.Customers b
WHERE  a.CustomerId > b.CustomerId AND
       q.Text.MatchingTokenTFIDF( a.Name + ' ' + a.City,
                                   b.Name + ' ' + b.City ) > 0.5
ORDER BY 3
```

qSQL

Routine Text.MatchingTypewriter

Typ Skalarwertfunktion

Beschreibung Berechnet über die Typewriter-Distanz zeichenbasiert die Ähnlichkeit zwischen zwei Zeichenketten und liefert einen Wert zwischen 0 und 1 zurück.

Parameter

@text1 Erster Text.

@text2 Zweiter Text.

Beispiel:

```
SELECT a.Name, b.Name,  
       q.Text.MatchingTypewriter( a.Name, b.Name )  
FROM   Persons.Customers a, Persons.Customers b  
WHERE  a.CustomerId > b.CustomerId AND  
       q.Text.MatchingTypewriter( a.Name, b.Name ) > 0.8  
ORDER BY 3
```

Routine Text.PatternsCharSort

Typ Skalarwertfunktion

Beschreibung Sortiert die Zeichen der übergebenen Zeichenkette alphabetisch und liefert diese als Zeichenkette zurück.

Parameter

@text Text.

Beispiel:

```
SELECT Name, q.Text.PatternsCharSort( Name )
FROM   Persons.Customers
ORDER BY 2
```


qSQL

Routine Text.PatternsDataTypeRecognised

Typ Skalarwertfunktion

Beschreibung Ermittelt den Datentyp eines Wertes.

Parameter

@text Text.

Beispiel:

```
SELECT DISTINCT q.Text.PatternsDataTypeRecognised( PostalCode )
FROM Persons.Customers
```

qSQL

Routine Text.PatternsNamedEntity

Typ Skalarwertfunktion

Beschreibung Ermittelt den oder die semantischen Typen eines Wertes.

Parameter

@text Text.

Beispiel:

```
SELECT Name, q.Text.PatternsNamedEntity( Name )
FROM   Persons.Customers
WHERE  q.Text.PatternsNamedEntity( Name ) <> '{FirstName} {LastName}'
```

qSQL

Routine Text.PatternsOmissionKey

Typ Skalarwertfunktion

Beschreibung Liefert einen Code nach dem Omission Key-Algorithmus von J. Pollock und A. Zamora zurück.

Parameter

@text Text.

Beispiel:

```
SELECT Name, q.Text.PatternsOmissionKey( Name )  
FROM   Persons.Customers  
ORDER BY 2
```

qSQL

Routine Text.PatternsReverseOrder

Typ Skalarwertfunktion

Beschreibung Gibt die übergebene Zeichenfolge in umgekehrter Reihenfolge zurück.

Parameter

@text Text.

Beispiel:

```
SELECT Name, q.Text.PatternsReverseOrder( Name )
FROM   Persons.Customers
ORDER BY 2
```

qSQL

| | |
|---------------------|--|
| Routine | Text.PatternsSimple |
| Typ | Skalarwertfunktion |
| Beschreibung | Ersetzt Zahlen durch die ,9', Kleinbuchstaben durch ,a' und Großbuchstaben durch ,A'. Alle anderen Zeichen werden durch ein ,*' ersetzt. |
| Parameter | |
| @text | Text. |

Beispiel:

```
SELECT DISTINCT q.Text.PatternsSimple( PostalCode )
FROM   Persons.Customers
```

qSQL

Routine Text.PatternsSkeletonKey

Typ Skalarwertfunktion

Beschreibung Liefert einen Code nach dem Skeleton Key-Algorithmus von J. Pollock und A. Zamora zurück.

Parameter

@text Text.

Beispiel:

```
SELECT Name, q.Text.PatternsSkeletonKey( Name )
FROM   Persons.Customers
ORDER BY 2
```

qSQL

| | |
|---------------------|--|
| Routine | Text.PhoneticCaver |
| Typ | Skalarwertfunktion |
| Beschreibung | Liefert den phonetischen Caverphone-Code zurück. |

Parameter

| | |
|-------|-------|
| @text | Text. |
|-------|-------|

Beispiel:

```
SELECT Name, q.Text.PhoneticCaver( Name )  
FROM   Persons.Customers  
ORDER BY 2
```

qSQL

| | |
|---------------------|---|
| Routine | Text.PhoneticCologne |
| Typ | Skalarwertfunktion |
| Beschreibung | Liefert den phonetischen Kölner Phonetik-Code zurück. |
| Parameter | |
| @text | Text. |

Beispiel:

```
SELECT Name, q.Text.PhoneticCologne( Name )  
FROM   Persons.Customers  
ORDER BY 2
```


qSQL

Routine Text.PhoneticDaitchMokotoff

Typ Skalarwertfunktion

Beschreibung Liefert den phonetischen Code nach Daitch/Mokotoff zurück.

Parameter

@text Text.

Beispiel:

```
SELECT Name, q.Text.PhoneticDaitchMokotoff( Name )  
FROM   Persons.Customers  
ORDER BY 2
```

qSQL

| | |
|---------------------|---|
| Routine | Text.PhoneticDoubleMetaphone |
| Typ | Skalarwertfunktion |
| Beschreibung | Liefert den phonetischen Metaphone-Code zurück. |
| Parameter | |
| @text | Text. |

Beispiel:

```
SELECT Name, q.Text.PhoneticDoubleMetaphone( Name )
FROM   Persons.Customers
ORDER BY 2
```

qSQL

Routine Text.PhoneticGerman

Typ Skalarwertfunktion

Beschreibung Liefert den phonetischen Code für deutschsprachige Namen nach dem Algorithmus von J. Michael zurück.

Parameter

@text Text..

Beispiel:⁹

```
SELECT Name, q.Text.PhoneticGerman( Name )
FROM   Persons.Customers
ORDER BY 2
```

⁹ Das SQL-Beispiel funktioniert nur mit der DLL „GermanPhoneticEncoder_SQLExtendedProc.dll“

qSQL

Routine Text.PhoneticIBMAAlphaSearch

Typ Skalarwertfunktion

Beschreibung Liefert den phonetischen Code nach dem „IBM Alpha Inquiry System Personal Name Encoding“-Algorithmus zurück.

Parameter

@text Text.

Beispiel:

```
SELECT Name, q.Text.PhoneticIBMAAlphaSearch( Name )  
FROM   Persons.Customers  
ORDER BY 2
```

qSQL

Routine Text.PhoneticNysiis

Typ Skalarwertfunktion

Beschreibung Liefert den phonetischen Code nach dem NYSIIS-Algorithmus („New York State Identification and Intelligence System“) zurück.

Parameter

@text Text.

Beispiel:

```
SELECT Name, q.Text.PhoneticNysiis( Name )  
FROM   Persons.Customers  
ORDER BY 2
```

qSQL

| | |
|---------------------|---|
| Routine | Text.PhoneticONCA |
| Typ | Skalarwertfunktion |
| Beschreibung | Liefert den phonetischen Code nach dem ONCA („Oxford Name Compression Algorithm“) zurück. |
| Parameter | |
| @text | Text. |

Beispiel:

```
SELECT Name, q.Text.PhoneticONCA( Name )
FROM   Persons.Customers
ORDER BY 2
```

Routine Text.PhoneticPhonem

Typ Skalarwertfunktion

Beschreibung Liefert den phonetischen Code nach dem PHONEM-Algorithmus von G. Wilde und C. Meyer zurück.

Parameter

@text Text.

Beispiel:

```
SELECT Name, q.Text.PhoneticPhonem( Name )
FROM   Persons.Customers
ORDER BY 2
```

qSQL

Routine Text.PhoneticPhonex

Typ Skalarwertfunktion

Beschreibung Liefert den phonetischen Code nach dem Phonex-Algorithmus von A.J. Lait und B. Randell zurück.

Parameter

@text Text.

Beispiel:

```
SELECT Name, q.Text.PhoneticPhonex( Name )  
FROM   Persons.Customers  
ORDER BY 2
```


Routine Text.PhoneticRethSchek

Typ Skalarwertfunktion

Beschreibung Liefert den phonetischen Code für deutsche Namen nach einem Algorithmus von H.-P. von Reth und H.-J. Schek zurück.

Parameter

@text Text.

Beispiel:

```
SELECT Name, q.Text.PhoneticRethSchek( Name )  
FROM   Persons.Customers  
ORDER BY 2
```

Routine Text.PhoneticSoundex

Typ Skalarwertfunktion

Beschreibung Liefert den phonetischen Code nach dem Soundex-Algorithmus zurück.

Parameter

@text Text.

Beispiel:

```
SELECT Name, q.Text.PhoneticSoundex( Name )
FROM   Persons.Customers
ORDER BY 2
```

qSQL

Routine Text.PreparerGenericText

Typ Skalarwertfunktion

Beschreibung Bereitet den Text für andere Algorithmen vor, indem nicht-druckbare Zeichen entfernt werden.

Parameter

@text Text.

Beispiel:

```
SELECT DISTINCT Description,  
       q.Text.PreparerGenericText( Description )  
FROM   Sales.Articles
```

qSQL

| | |
|---------------------|--|
| Routine | Text.PreparerRemoveStopwords |
| Typ | Skalarwertfunktion |
| Beschreibung | Bereitet den Text für andere Algorithmen vor, indem Stoppwörter entfernt werden. |

Parameter

| | |
|-------|-------|
| @text | Text. |
|-------|-------|

Beispiel:

```
SELECT DISTINCT Description,
       q.Text.PreparerRemoveStopwords( Description )
FROM   Sales.Articles
```

Routine Text.Tokenize

Typ Skalarwertfunktion

Beschreibung Generische Funktion zum Auftrennen von Zeichenketten in einzelne Token durch beliebige Tokenizer-Klassen.

Parameter

| | |
|------------------|---|
| @text | Text. |
| @delimiter | Zeichenkette zur Anzeige der einzelnen getrennten Token |
| @type | Vollständiger Klassenname. |
| @assemblyString | Vollständige Beschreibung der Identität einer Assembly. Wird nur der Name der Assembly angegeben, so werden Standardwerte für Version (0.0.0.0), Culture (neutral) und PublicKeyToken (NULL) verwendet. |
| @namedParameters | Benannte Parameter, die zum Setzen von Eigenschaften der Klasseninstanz verwendet werden. |

Beispiel:

```
SELECT Name,
       q.Text.Tokenize( Name, ' | ',
                        'FreD.Text.Tokenizer.NGramTokenizer',
                        'Core',
                        'NGramSize=4, NormalizeHeadTail=false' )
FROM   Persons.Customers
```

Routine Text.TokenizeFrequency

Typ Tabellenwertfunktion

Beschreibung Generische Funktion zum Auftrennen von Zeichenketten in einzelne Token durch eine beliebige Tokenizer-Klasse und Berechnen der Häufigkeiten dieser.

Parameter

| | |
|------------------|---|
| @table | Tabellenname. |
| @columns | Spaltennamen. |
| @where | WHERE-Klausel zur Einschränkung der Werte. |
| @type | Vollständiger Klassenname. |
| @assemblyString | Vollständige Beschreibung der Identität einer Assembly. Wird nur der Name der Assembly angegeben, so werden Standardwerte für Version (0.0.0.0), Culture (neutral) und PublicKeyToken (NULL) verwendet. |
| @namedParameters | Benannte Parameter, die zum Setzen von Eigenschaften der Klasseninstanz verwendet werden. |
| @distinct | Wenn true, wird ein in einem Datensatz mehrfach vorkommender Wert nur einmal gezählt. |

Beispiel:

```
SELECT *
FROM q.Text.TokenizeFrequency(
    'Sample.Persons.Customers',
    'PostalCode',
    '',
    'FreD.Text.Tokenizer.NGramTokenizer',
    'Core',
    'NGramSize=3, NormalizeHeadTail=false',
    1 )
ORDER BY Frequency DESC
```

| | |
|---------------------|--|
| Routine | Text.TokenizerGeneric |
| Typ | Skalarwertfunktion |
| Beschreibung | Funktion zum Auftrennen von Zeichenketten in einzelne Token durch die <i>GenericTokenizer</i> -Klasse. |

Parameter

| | |
|------------|--|
| @text | Text. |
| @delimiter | Zeichenkette zur Anzeige der einzelnen getrennten Token. |

Beispiel:

```
SELECT Name,
       q.Text.TokenizerGeneric( Name, ' | ' )
FROM   Persons.Customers
```

| | |
|---------------------|--|
| Routine | Text.TokenizerGenericFrequency |
| Typ | Tabellenwertfunktion |
| Beschreibung | Funktion zum Auftrennen von Zeichenketten in einzelne Token durch die <i>GenericTokenizer</i> -Klasse und Berechnen der Häufigkeiten dieser. |
| Parameter | |
| @table | Tabellenname. |
| @columns | Spaltennamen. |
| @where | WHERE-Klausel zur Einschränkung der Werte. |
| @distinct | Wenn true, wird ein in einem Datensatz mehrfach vorkommender Wert nur einmal gezählt. |
| @useTokenizer | Wenn true, Verwendung der <i>GenericTokenizer</i> -Klasse, ansonsten kein Auftrennen. |

Beispiel:

```
SELECT *
FROM   q.Text.TokenizerGenericFrequency(
        'Sample.Persons.Customers', 'Name', NULL, 1, 1 )
ORDER BY Frequency DESC
```


| | |
|---------------------|--|
| Routine | Text.TokenizerNGram |
| Typ | Skalarwertfunktion |
| Beschreibung | Funktion zum Auftrennen von Zeichenketten in einzelne Token durch die <i>NGramTokenizer</i> -Klasse. |
| Parameter | |
| @text | Text. |
| @delimiter | Zeichenkette zur Anzeige der einzelnen getrennten Token. |

Beispiel:

```
SELECT Name,  
       q.Text.TokenizerNGram( Name, ' | ' )  
FROM   Persons.Customers
```

qSQL

| | |
|---------------------|--|
| Routine | Text.TokenizerNGramFrequency |
| Typ | Tabellenwertfunktion |
| Beschreibung | Funktion zum Auftrennen von Zeichenketten in einzelne Token durch die <i>NGramTokenizer</i> -Klasse und Berechnen der Häufigkeiten dieser. |
| Parameter | |
| @table | Tabellenname. |
| @columns | Spaltennamen. |
| @where | WHERE-Klausel zur Einschränkung der Werte. |
| @distinct | Wenn true, wird ein in einem Datensatz mehrfach vorkommender Wert nur einmal gezählt. |
| @useTokenizer | Wenn true, Verwendung der <i>GenericTokenizer</i> -Klasse, ansonsten kein Auftrennen. |

Beispiel:

```
SELECT *
FROM   q.Text.TokenizerNGramFrequency(
        'Sample.Persons.Customers', 'PostalCode', NULL, 1 )
ORDER BY Frequency DESC
```

A.6 Understand

Routine Understand.ForeignKey

Typ Tabellenwertfunktion

Beschreibung Generische Funktion zum Analysieren von Primär-/Fremdschlüsselbeziehungen über Klassen, die die Schnittstelle *IForeignKey* implementieren.

Parameter

| | |
|------------------|---|
| @table1 | Erste Tabelle. |
| @columns1 | Zu untersuchende Spalten der ersten Tabelle. |
| @where1 | WHERE-Klausel für erste Tabelle zum Einschränken der zu untersuchenden Daten. |
| @table2 | Zweite Tabelle. |
| @columns2 | Zu untersuchende Spalten der zweiten Tabelle. |
| @where2 | WHERE-Klausel für zweite Tabelle zum Einschränken der zu untersuchenden Daten. |
| @type | Vollständiger Klassenname. |
| @assemblyString | Vollständige Beschreibung der Identität einer Assembly. Wird nur der Name der Assembly angegeben, so werden Standardwerte für Version (0.0.0.0), Culture (neutral) und PublicKeyToken (NULL) verwendet. |
| @namedParameters | Benannte Parameter, die zum Setzen von Eigenschaften der Klasseninstanz verwendet werden. |

Beispiel:

```
SELECT *
FROM q.Understand.ForeignKey(
    'Sample.Persons.Customers', '*', NULL,
    'Sample.Sales.Orders', '*', NULL,
    'FreD.Understand.Dependencies.ForeignKeyAnalyse',
    'Core',
    'Threshold=0.0' )
```

Routine Understand.ForeignKeyAnalyse

Typ Tabellenwertfunktion

Beschreibung Funktion zum Analysieren von Primär-/Fremdschlüsselbeziehungen, indem instanzbasiert die Schnittmengen an übereinstimmenden Daten zwischen den einzelnen Spalten jeder Tabelle ermittelt werden.

Parameter

@table1 Erste Tabelle.

@columns1 Zu untersuchende Spalten der ersten Tabelle.

@where1 WHERE-Klausel für erste Tabelle zum Einschränken der zu untersuchenden Daten.

@table2 Zweite Tabelle.

@columns2 Zu untersuchende Spalten der zweiten Tabelle.

@where2 WHERE-Klausel für zweite Tabelle zum Einschränken der zu untersuchenden Daten.

Beispiel:

```
SELECT *
FROM   q.Understand.ForeignKeyAnalyse(
        'Sample.Persons.Customers', '*', NULL,
        'Sample.Sales.Orders',    '*', NULL )
WHERE  RightValues = 0 OR LeftValues = 0
```

qSQL

| | |
|---------------------|---|
| Routine | Understand.ForeignKeyTextMatchingAnalyse |
| Typ | Tabellenwertfunktion |
| Beschreibung | <p>Funktion zum Analysieren von</p> <p>Primär-/Fremdschlüsselbeziehungen, indem schemabasiert die Spaltennamen über einen Textvergleichsalgorithmus auf Übereinstimmung überprüft werden.</p> |
| Parameter | |
| @table1 | Erste Tabelle. |
| @columns1 | Zu untersuchende Spalten der ersten Tabelle. |
| @where1 | WHERE-Klausel für erste Tabelle zum Einschränken der zu untersuchenden Daten. |
| @table2 | Zweite Tabelle. |
| @columns2 | Zu untersuchende Spalten der zweiten Tabelle. |
| @where2 | WHERE-Klausel für zweite Tabelle zum Einschränken der zu untersuchenden Daten. |

Beispiel:

```
SELECT *
FROM   q.Understand.ForeignKeyTextMatchingAnalyse(
        'Sample.Persons.Customers', '*', NULL,
        'Sample.Sales.Orders',     '*', NULL )
```

Routine Understand.PrimaryKey

Typ Tabellenwertfunktion

Beschreibung Generische Funktion zum Erkennen von möglichen Primärschlüsseln über Klassen, die die Schnittstelle *IPrimaryKey* implementieren.

Parameter

| | |
|------------------|---|
| @table | Tabellenname. |
| @columns | Zu untersuchende Spalten der Tabelle. |
| @where | WHERE-Klausel zum Einschränken der zu untersuchenden Daten. |
| @type | Vollständiger Klassenname. |
| @assemblyString | Vollständige Beschreibung der Identität einer Assembly. Wird nur der Name der Assembly angegeben, so werden Standardwerte für Version (0.0.0.0), Culture (neutral) und PublicKeyToken (NULL) verwendet. |
| @namedParameters | Benannte Parameter, die zum Setzen von Eigenschaften der Klasseninstanz verwendet werden. |

Beispiel:

```
SELECT *
FROM q.Understand.PrimaryKey(
    'Sample.Sales.Articles', '*', NULL,
    'FreD.Understand.Dependencies.PrimaryKeyAnalyse',
    'Core', 'Threshold=0.4' )
```

qSQL

| | |
|---------------------|---|
| Routine | Understand.PrimaryKeyAnalyse |
| Typ | Tabellenwertfunktion |
| Beschreibung | Funktion zum Erkennen von möglichen Primärschlüsseln über die <i>PrimaryKeyAnalyse</i> -Klasse. |
| Parameter | |
| @table | Tabellenname. |
| @columns | Zu untersuchende Spalten der Tabelle. |
| @where | WHERE-Klausel zum Einschränken der zu untersuchenden Daten. |

Beispiel:

```
SELECT *
FROM q.Understand.PrimaryKeyAnalyse(
    'Sample.Sales.Articles',
    '*',
    NULL )
```

qSQL

| | |
|---------------------|---|
| Routine | Understand.PropertiesBenfordsLaw |
| Typ | Aggregatfunktion |
| Beschreibung | Funktion zum Ermitteln der Häufigkeiten der einzelnen Ziffern einer Zahl zur Überprüfung von Benford's Law. |
| Parameter | |
| @value | Zu analysierende Spalte. |

Beispiel:

```
SELECT q.Understand.PropertiesBenfordsLaw( ListPrice )
FROM   Sales.Articles
```


qSQL

Routine Understand.PropertiesBlankCount

Typ Aggregatfunktion

Beschreibung Funktion zum Ermitteln der Anzahl leerer Einträge (nicht NULL).

Parameter

@value Zu analysierende Spalte.

Beispiel:

```
SELECT q.Understand.PropertiesBlankCount( FirstName )
FROM   Persons.Customers
```

qSQL

| | |
|---------------------|---|
| Routine | Understand.PropertiesCountNotNull |
| Typ | Aggregatfunktion |
| Beschreibung | Funktion zum Ermitteln der Anzahl Einträge, die nicht NULL enthalten. |
| Parameter | |
| @value | Zu analysierende Spalte. |

Beispiel:

```
SELECT q.Understand.PropertiesCountNotNull( FirstName )
FROM   Persons.Customers
```

qSQL

Routine Understand.PropertiesCountNull

Typ Aggregatfunktion

Beschreibung Funktion zum Ermitteln der Anzahl Einträge, die NULL enthalten.

Parameter

@value Zu analysierende Spalte.

Beispiel:

```
SELECT q.Understand.PropertiesCountNull( FirstName )  
FROM   Persons.Customers
```

| | |
|---------------------|---|
| Routine | Understand.PropertiesDataProperty |
| Typ | Aggregatfunktion |
| Beschreibung | Generische Aggregatfunktion zum Aufrufen von Klassen, die die Schnittstelle <i>IDataProperty</i> implementieren. |
| Parameter | |
| @value | Konkatenierte Zeichenkette aus Name der Klasse, die <i>IDataProperty</i> implementiert, Assemblyname, benannte Parameter und zu analysierende Spalte. |

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
    '{FreD.Understand.Properties.Number.MeanProperty|
    Core|
    TrimmedCount=10}' +
    COALESCE( CAST( ListPrice AS NVARCHAR(50) ), '{NULL}' ) )
FROM Sales.Articles
```

| | |
|---------------------|---|
| Routine | Understand.PropertiesDataTypeRecognised |
| Typ | Aggregatfunktion |
| Beschreibung | Funktion zum Ermitteln des Datentyps der Spalte anhand des Dateninhaltes. |
| Parameter | |
| @value | Zu analysierende Spalte. |

Beispiel:

```
SELECT q.Understand.PropertiesDataTypeRecognised( Weight )
FROM   Sales.Articles
```

qSQL

| | |
|---------------------|---|
| Routine | Understand.PropertiesGeometricMean |
| Typ | Aggregatfunktion |
| Beschreibung | Errechnet den geometrischen Mittelwert. |
| Parameter | |
| @value | Zu analysierende Spalte. |

Beispiel:

```
SELECT q.Understand.PropertiesGeometricMean( ListPrice )
FROM   Sales.Articles
WHERE  ListPrice < 10
```

qSQL

| | |
|---------------------|--|
| Routine | Understand.PropertiesHarmonicMean |
| Typ | Aggregatfunktion |
| Beschreibung | Errechnet den harmonischen Mittelwert. |
| Parameter | |
| @value | Zu analysierende Spalte. |

Beispiel:

```
SELECT q.Understand.PropertiesHarmonicMean( ListPrice )
FROM   Sales.Articles
```

qSQL

Routine Understand.PropertiesIntegerCount

Typ Aggregatfunktion

Beschreibung Ermittelt die Anzahl an Ganzzahlen.

Parameter

@value Zu analysierende Spalte.

Beispiel:

```
SELECT q.Understand.PropertiesIntegerCount( ListPrice )
FROM   Sales.Articles
```


qSQL

| | |
|---------------------|--|
| Routine | Understand.PropertiesMax |
| Typ | Aggregatfunktion |
| Beschreibung | Ermittelt die fünf maximalen Werte einer Spalte. |
| Parameter | |
| @value | Zu analysierende Spalte. |

Beispiel:

```
SELECT q.Understand.PropertiesMax( ListPrice )  
FROM   Sales.Articles
```

qSQL

| | |
|---------------------|--|
| Routine | Understand.PropertiesMaxDecimalPlaces |
| Typ | Aggregatfunktion |
| Beschreibung | Ermittelt die maximale Anzahl an Nachkommastellen. |
| Parameter | |
| @value | Zu analysierende Spalte. |

Beispiel:

```
SELECT q.Understand.PropertiesMaxDecimalPlaces( ListPrice )
FROM Sales.Articles
```

qSQL

Routine Understand.PropertiesMaxTextCount

Typ Aggregatfunktion

Beschreibung Ermittelt die Anzahl an Zeichenketten, die am längsten sind.

Parameter

@value Zu analysierende Spalte.

Beispiel:

```
SELECT q.Understand.PropertiesMaxTextCount( Name )  
FROM   Sales.Articles
```

qSQL

| | |
|---------------------|--|
| Routine | Understand.PropertiesMaxTextLength |
| Typ | Aggregatfunktion |
| Beschreibung | Ermittelt die Länge der längsten Zeichenkette. |
| Parameter | |
| @value | Zu analysierende Spalte. |

Beispiel:

```
SELECT q.Understand.PropertiesMaxTextLength( Name )  
FROM   Sales.Articles
```

qSQL

| | |
|---------------------|--|
| Routine | Understand.PropertiesMean |
| Typ | Aggregatfunktion |
| Beschreibung | Berechnet den arithmetischen Mittelwert. |
| Parameter | |
| @value | Zu analysierende Spalte. |

Beispiel:

```
SELECT q.Understand.PropertiesMean( ListPrice )
FROM   Sales.Articles
```

qSQL

| | |
|---------------------|-------------------------------------|
| Routine | Understand.PropertiesMin |
| Typ | Aggregatfunktion |
| Beschreibung | Ermittelt die fünf minimalen Werte. |
| Parameter | |
| @value | Zu analysierende Spalte. |

Beispiel:

```
SELECT q.Understand.PropertiesMin( ListPrice )
FROM Sales.Articles
```

qSQL

| | |
|---------------------|--|
| Routine | Understand.PropertiesMinDecimalPlaces |
| Typ | Aggregatfunktion |
| Beschreibung | Ermittelt die minimale Anzahl an Nachkommastellen. |
| Parameter | |
| @value | Zu analysierende Spalte. |

Beispiel:

```
SELECT q.Understand.PropertiesMaxDecimalPlaces( ListPrice )
FROM Sales.Articles
```

Routine Understand.PropertiesMinTextCount

Typ Aggregatfunktion

Beschreibung Ermittelt die Anzahl an Zeichenketten, die am kürzesten sind.

Parameter

@value Zu analysierende Spalte.

Beispiel:

```
SELECT q.Understand.PropertiesMinTextCount( Name )  
FROM   Sales.Articles
```


qSQL

| | |
|---------------------|---|
| Routine | Understand.PropertiesMinTextLength |
| Typ | Aggregatfunktion |
| Beschreibung | Ermittelt die Länge der kürzesten Zeichenkette. |
| Parameter | |
| @value | Zu analysierende Spalte. |

Beispiel:

```
SELECT q.Understand.PropertiesMinTextLength( Name )  
FROM   Sales.Articles
```

qSQL

| | |
|---------------------|--|
| Routine | Understand.PropertiesPrimaryKeyCandidate |
| Typ | Aggregatfunktion |
| Beschreibung | Ermittelt anhand der Werte für eine Spalte, ob diese als Primärschlüssel geeignet ist. |
| Parameter | |
| @value | Zu analysierende Spalte. |

Beispiel:

```
SELECT q.Understand.PropertiesSquareSum( ListPrice )
FROM Sales.Articles
```

qSQL

| | |
|---------------------|---|
| Routine | Understand.PropertiesProduct |
| Typ | Aggregatfunktion |
| Beschreibung | Errechnet das Produkt der Werte der numerischen Spalte. |
| Parameter | |
| @value | Zu analysierende Spalte. |

Beispiel:

```
SELECT q.Understand.PropertiesProduct( ListPrice )
FROM   Sales.Articles
WHERE  ListPrice < 10
```

qSQL

| | |
|---------------------|---|
| Routine | Understand.PropertiesRange |
| Typ | Aggregatfunktion |
| Beschreibung | Errechnet die Spannbreite für eine numerische Spalte. |
| Parameter | |
| @value | Zu analysierende Spalte. |

Beispiel:

```
SELECT q.Understand.PropertiesRange( ListPrice )
FROM   Sales.Articles
```

qSQL

| | |
|---------------------|--|
| Routine | Understand.PropertiesSquareSum |
| Typ | Aggregatfunktion |
| Beschreibung | Errechnet die Quadratsumme für eine numerische Spalte. |
| Parameter | |
| @value | Zu analysierende Spalte. |

Beispiel:

```
SELECT q.Understand.PropertiesSquareSum( ListPrice )  
FROM   Sales.Articles
```

qSQL

Routine Understand.PropertiesStdDev

Typ Aggregatfunktion

Beschreibung Errechnet die Standardabweichung für eine numerische Spalte.

Parameter

@value Zu analysierende Spalte.

Beispiel:

```
SELECT q.Understand.PropertiesStdDev( ListPrice )
FROM   Sales.Articles
```

qSQL

Routine Understand.PropertiesStdErr

Typ Aggregatfunktion

Beschreibung Errechnet den Standardfehler für eine numerische Spalte.

Parameter

@value Zu analysierende Spalte.

Beispiel:

```
SELECT q.Understand.PropertiesStdErr( ListPrice )  
FROM   Sales.Articles
```

qSQL

| | |
|---------------------|---|
| Routine | Understand.PropertiesSum |
| Typ | Aggregatfunktion |
| Beschreibung | Errechnet die Summe für eine numerische Spalte. |
| Parameter | |
| @value | Zu analysierende Spalte. |

Beispiel:

```
SELECT q.Understand.PropertiesSum( ListPrice )  
FROM   Sales.Articles
```


qSQL

| | |
|---------------------|---|
| Routine | Understand.PropertiesVariance |
| Typ | Aggregatfunktion |
| Beschreibung | Errechnet die Varianz für eine numerische Spalte. |
| Parameter | |
| @value | Zu analysierende Spalte. |

Beispiel:

```
SELECT q.Understand.PropertiesVariance( ListPrice )
FROM   Sales.Articles
```

Routine Understand.PropertyListData

Typ Tabellenwertfunktion

Beschreibung Generische Funktion zum Ermitteln aller Daten-Eigenschaften für Spalten einer Tabelle aus einer Klasse einer Assembly, die *IDataPropertyList* implementiert.

Parameter

| | |
|------------------|---|
| @table | Tabellenname. |
| @columns | Zu untersuchende Spalten der Tabelle. |
| @where | WHERE-Klausel zum Einschränken der zu untersuchenden Daten. |
| @type | Vollständiger Klassenname. |
| @assemblyString | Vollständige Beschreibung der Identität einer Assembly. Wird nur der Name der Assembly angegeben, so werden Standardwerte für Version (0.0.0.0), Culture (neutral) und PublicKeyToken (NULL) verwendet. |
| @namedParameters | Benannte Parameter, die zum Setzen von Eigenschaften der Klasseninstanz verwendet werden. |

Beispiel:

```
SELECT "Column", Name, Value, Description
FROM q.Understand.PropertyListData(
    'Sample.Sales.Articles',
    '*',
    NULL,
    'FreD.Understand.Properties.DataPropertyListGeneric',
    'Core',
    NULL )
ORDER BY "Column", Name
```

| | |
|---------------------|---|
| Routine | Understand.PropertyListDataGeneric |
| Typ | Tabellenwertfunktion |
| Beschreibung | Ermittelt für Spalten einer Tabelle alle Daten-Eigenschaften aus Klassen einer Assembly, die <i>IDataProperty</i> implementieren. |
| Parameter | |
| @table | Tabellenname. |
| @columns | Zu untersuchende Spalten der Tabelle. |
| @where | WHERE-Klausel zum Einschränken der zu untersuchenden Daten. |
| @type | Vollständiger Klassenname. |
| @assemblyString | Vollständige Beschreibung der Identität einer Assembly. Wird nur der Name der Assembly angegeben, so werden Standardwerte für Version (0.0.0.0), Culture (neutral) und PublicKeyToken (NULL) verwendet. |
| @namedParameters | Benannte Parameter, die zum Setzen von Eigenschaften der Klasseninstanz verwendet werden. |

Beispiel:

```
SELECT "Column", Name, Value, Description
FROM q.Understand.PropertyListDataGeneric(
    'Sample.Sales.Articles',
    '*',
    null, 'core' )
ORDER BY "Column", Name
```

Routine Understand.PropertyListSchema

Typ Tabellenwertfunktion

Beschreibung Generische Funktion zum Ermitteln aller Schema-Eigenschaften für Spalten einer Tabelle aus einer Klasse einer Assembly, die *ISchemaPropertyList* implementiert.

Parameter

| | |
|------------------|---|
| @table | Tabellenname. |
| @columns | Zu untersuchende Spalten der Tabelle. |
| @type | Vollständiger Klassenname. |
| @assemblyString | Vollständige Beschreibung der Identität einer Assembly. Wird nur der Name der Assembly angegeben, so werden Standardwerte für Version (0.0.0.0), Culture (neutral) und PublicKeyToken (NULL) verwendet. |
| @namedParameters | Benannte Parameter, die zum Setzen von Eigenschaften der Klasseninstanz verwendet werden. |

Beispiel:

```
SELECT "Column", Name, Value, Description
FROM q.Understand.PropertyListSchema(
    'Sample.Sales.Articles',
    '*',
    'FreD.Understand.Properties.SchemaPropertyListGeneric',
    'Core',
    NULL )
ORDER BY "Column", Name
```

qSQL

| | |
|---------------------|---|
| Routine | Understand.PropertyListSchemaGeneric |
| Typ | Tabellenwertfunktion |
| Beschreibung | Ermittelt für Spalten einer Tabelle alle Schema-Eigenschaften über ADO.NET. |
| Parameter | |
| @table | Tabellenname. |
| @columns | Zu untersuchende Spalten der Tabelle. |

Beispiel:

```
SELECT "Column", Name, Value, Description
FROM q.Understand.PropertyListSchemaGeneric(
    'Sample.Sales.Articles',
    '*' )
ORDER BY "Column", Name
```

| | |
|---------------------|---|
| Routine | Understand.RuleInduction |
| Typ | Tabellenwertfunktion |
| Beschreibung | Generische Funktion zum Suchen nach Regeln im Datenbestand. |
| Parameter | |
| @table | Tabellenname. |
| @columns | Zu untersuchende Spalten der Tabelle. |
| @where | WHERE-Klausel zum Einschränken der zu untersuchenden Daten. |
| @type | Vollständiger Klassenname. |
| @assemblyString | Vollständige Beschreibung der Identität einer Assembly. Wird nur der Name der Assembly angegeben, so werden Standardwerte für Version (0.0.0.0), Culture (neutral) und PublicKeyToken (NULL) verwendet. |
| @namedParameters | Benannte Parameter, die zum Setzen von Eigenschaften der Klasseninstanz verwendet werden. |

Beispiel:¹⁰

```
SELECT "Rule", Confidence, Support, Frequency
FROM q.Understand.RuleInduction(
    'Sample.Persons.Customers',
    'Title, MaritalStatus, Sex',
    null,
    'FreD.Understand.Rules.AprioriRuleInduction',
    'WekaRuleInduction',
    'MinItemFrequency=3, Delta=0.05' )
```

¹⁰ Das SQL-Beispiel funktioniert nur mit dem Plugin „WekaRuleInduction“

Routine Understand.RuleInductionOperator

Typ Tabellenwertfunktion

Beschreibung Funktion zum Suchen nach Größer-, Kleiner-, Gleich-Regeln für numerische Spalten im Datenbestand.

Parameter

@table Tabellenname.

@columns Zu untersuchende Spalten der Tabelle.

@where WHERE-Klausel zum Einschränken der zu untersuchenden Daten.

Beispiel:

```
SELECT "Rule", Support, Frequency
FROM q.Understand.RuleInductionOperator(
    'Sample.Sales.Articles',
    '*',
    NULL )
```

A.7 Improve

Routine Improve.DeduplicationConcat

Typ Aggregatfunktion

Beschreibung Konkateniert alle Werte einer Gruppe zu einer gemeinsamen Zeichenkette mit dem ,|'-Zeichen als Trenner.

Parameter

@value Spaltenname.

Beispiel:

```
SELECT "q.Group",  
       q.Improve.DeduplicationConcat( Name )  
FROM Persons.Customers  
WHERE "q.Group" IS NOT NULL  
GROUP BY "q.Group"
```


qSQL

Routine Improve.DeduplicationDiscard

Typ Aggregatfunktion

Beschreibung Liefert nur dann kein NULL zurück, wenn alle Werte der Gruppe identisch sind.

Parameter

@value Spaltenname.

Beispiel:

```
SELECT "q.Group",  
       q.Improve.DeduplicationDiscard( Street ),  
       q.Improve.DeduplicationConcat( Name )  
FROM   Persons.Customers  
WHERE  "q.Group" IS NOT NULL  
GROUP BY "q.Group"
```

qSQL

Routine Improve.DeduplicationFirst

Typ Aggregatfunktion

Beschreibung Liefert den ersten Wert innerhalb der Gruppe zurück.

Parameter

@value Spaltenname.

Beispiel:

```
SELECT "q.Group",  
       q.Improve.DeduplicationFirst( Name )  
FROM   Persons.Customers  
WHERE  "q.Group" IS NOT NULL  
GROUP BY "q.Group"
```

qSQL

| | |
|---------------------|---|
| Routine | Improve.DeduplicationFuse |
| Typ | Aggregatfunktion |
| Beschreibung | Generische Aggregatfunktion zum Verwenden einer Klasse, die die Schnittstelle <i>IFuse</i> implementiert. |
| Parameter | |
| @value | Spaltenname. |

Beispiel:

```
SELECT "q.Group",
       q.Improve.DeduplicationFuse(
           '{FreD.Improve.Deduplication.Fuse.Concat|Core}'+
           COALESCE( Name, '{NULL}') )
FROM   Persons.Customers
WHERE  "q.Group" IS NOT NULL
GROUP BY "q.Group"
```

Routine Improve.DeduplicationLast

Typ Aggregatfunktion

Beschreibung Liefert den letzten Wert innerhalb der Gruppe zurück.

Parameter

@value Spaltenname.

Beispiel:

```
SELECT "q.Group",  
       q.Improve.DeduplicationLast( Name )  
FROM   Persons.Customers  
WHERE  "q.Group" IS NOT NULL  
GROUP BY "q.Group"
```

qSQL

Routine Improve.DeduplicationLongestText

Typ Aggregatfunktion

Beschreibung Liefert die längste Zeichenkette innerhalb der Gruppe zurück.

Parameter

@value Spaltenname.

Beispiel:

```
SELECT "q.Group",
       q.Improve.DeduplicationLongestText( Name )
FROM   Persons.Customers
WHERE  "q.Group" IS NOT NULL
GROUP BY "q.Group"
```

qSQL

Routine Improve.DeduplicationMax

Typ Aggregatfunktion

Beschreibung Liefert den größten Wert innerhalb der Gruppe zurück.

Parameter

@value Spaltenname.

Beispiel:

```
SELECT "q.Group",
       q.Improve.DeduplicationMax( PostalCode )
FROM   Persons.Customers
WHERE  "q.Group" IS NOT NULL
GROUP BY "q.Group"
```

qSQL

| | |
|---------------------|--|
| Routine | Improve.DeduplicationMaxFrequency |
| Typ | Aggregatfunktion |
| Beschreibung | Liefert den am häufigsten vorkommenden Wert innerhalb der Gruppe zurück. |
| Parameter | |
| @value | Spaltenname. |

Beispiel:

```
SELECT "q.Group",
       q.Improve.DeduplicationMaxFrequency( Street )
FROM   Persons.Customers
WHERE  "q.Group" IS NOT NULL
GROUP BY "q.Group"
```

qSQL

Routine Improve.DeduplicationMean

Typ Aggregatfunktion

Beschreibung Liefert den arithmetischen Mittelwert einer Gruppe von Zahlen zurück.

Parameter

@value Spaltenname.

Beispiel:

```
SELECT "q.Group",  
       q.Improve.DeduplicationMean( ListPrice )  
FROM   Sales.Articles  
WHERE  "q.Group" IS NOT NULL  
GROUP BY "q.Group"
```


Routine Improve.DeduplicationMin

Typ Aggregatfunktion

Beschreibung Liefert den kleinsten Wert innerhalb der Gruppe zurück.

Parameter

@value Spaltenname.

Beispiel:

```
SELECT "q.Group",  
       q.Improve.DeduplicationMin( PostalCode )  
FROM   Persons.Customers  
WHERE  "q.Group" IS NOT NULL  
GROUP BY "q.Group"
```

Routine Improve.DeduplicationShortestText

Typ Aggregatfunktion

Beschreibung Liefert die kürzeste Zeichenkette innerhalb der Gruppe zurück.

Parameter

@value Spaltenname.

Beispiel:

```
SELECT "q.Group",
       q.Improve.DeduplicationShortestText( Name )
FROM   Persons.Customers
WHERE  "q.Group" IS NOT NULL
GROUP BY "q.Group"
```

Routine Improve.DeduplicationStdDev

Typ Aggregatfunktion

Beschreibung Liefert die Standardabweichung einer Gruppe von Zahlen zurück.

Parameter

@value Spaltenname.

Beispiel:

```
SELECT "q.Group",
       q.Improve.DeduplicationStdDev( ListPrice )
FROM   Sales.Articles
WHERE  "q.Group" IS NOT NULL
GROUP BY "q.Group"
```

qSQL

| | |
|---------------------|---|
| Routine | Improve.DeduplicationSum |
| Typ | Aggregatfunktion |
| Beschreibung | Liefert die Summe einer Gruppe von Zahlen zurück. |
| Parameter | |
| @value | Spaltenname. |

Beispiel:

```
SELECT "q.Group",
       q.Improve.DeduplicationSum( ListPrice )
FROM   Sales.Articles
WHERE  "q.Group" IS NOT NULL
GROUP BY "q.Group"
```

Routine Improve.DeduplicationVariance

Typ Aggregatfunktion

Beschreibung Liefert die Varianz einer Gruppe von Zahlen zurück.

Parameter

@value Spaltenname.

Beispiel:

```
SELECT "q.Group",
       q.Improve.DeduplicationVariance( ListPrice )
FROM   Sales.Articles
WHERE  "q.Group" IS NOT NULL
GROUP BY "q.Group"
```

Routine Improve.PredictorBuild

Typ Gespeicherte Prozedur

Beschreibung Erzeugt einen Predictor zur Vorhersage eines Spaltenwertes über Eingabespalten.

Parameter

| | |
|------------------|---|
| @table | Tabellenname. |
| @fields | Eingabespalten. |
| @fieldClass | Vorhersagespalte. |
| @where | WHERE-Klausel zum Einschränken der zu untersuchenden Daten. |
| @predictionName | Name des Predictors, unter dem dieser im Repository gespeichert werden soll. |
| @type | Vollständiger Klassenname des Partitioners. |
| @assemblyString | Vollständige Beschreibung der Identität einer Assembly. Wird nur der Name der Assembly angegeben, so werden Standardwerte für Version (0.0.0.0), Culture (neutral) und PublicKeyToken (NULL) verwendet. |
| @namedParameters | Benannte Parameter, die zum Setzen von Eigenschaften der Klasseninstanz verwendet werden. |

Beispiel:¹¹

```
EXEC q.Improve.PredictorBuild
    'Sample.Persons.Customers',
    'Age, MaritalStatus, Sex, State',
    'Rating',
    null,
    'PredictorCustomerRating',
    'FreD.Improve.Prediction.Id3Predictor',
    'WekaPrediction',
    null
```

¹¹ Das SQL-Beispiel funktioniert nur mit dem Plugin „WekaPrediction“

qSQL

| | |
|---------------------|--|
| Routine | Improve.PredictorBuildLookup |
| Typ | Gespeicherte Prozedur |
| Beschreibung | Erzeugt einen LookupPredictor zur Vorhersage eines Spaltenwertes über Eingabespalten, indem aus einer Nachschlagetabelle der Zielwert ausgelesen wird. |
| Parameter | |
| @table | Tabellenname. |
| @fields | Eingabespalten. |
| @fieldClass | Vorhersagespalte. |
| @where | WHERE-Klausel zum Einschränken der zu untersuchenden Daten. |
| @predictionName | Name des Predictors, unter dem dieser im Repository gespeichert werden soll. |

Beispiel:

```
EXEC q.Improve.PredictorBuildLookup
      'q.Lookup."City.US"',
      'PostalCode',
      'City',
      null,
      'PredictorCity'
```

qSQL

| | |
|---------------------|--|
| Routine | Improve.PredictorBuildZeroR |
| Typ | Gespeicherte Prozedur |
| Beschreibung | Erzeugt einen ZeroRPredictor zur Vorhersage eines Spaltenwertes über Eingabespalten. |
| Parameter | |
| @table | Tabellenname. |
| @fields | Eingabespalten. |
| @fieldClass | Vorhersagespalte. |
| @where | WHERE-Klausel zum Einschränken der zu untersuchenden Daten. |
| @predictionName | Name des Predictors, unter dem dieser im Repository gespeichert werden soll. |

Beispiel:

```
EXEC q.Improve.PredictorBuildZeroR
    'Sample.Persons.Customers',
    'YearOfBirthday, MaritalStatus, PostalCode',
    'Sex',
    null,
    'PredictorCustomerSex'
```


qSQL

Routine Improve.PredictorDelete

Typ Gespeicherte Prozedur

Beschreibung Löscht einen im Repository gespeicherten Predictor.

Parameter

@predictionName Name des Predictors, unter dem dieser im Repository gespeichert ist.

Beispiel:

```
EXEC q.Improve.PredictorDelete 'PredictorCustomerSex'
```

| | |
|---------------------|--|
| Routine | Improve.PredictorInfo |
| Typ | Gespeicherte Prozedur |
| Beschreibung | Gibt Informationen zu einem im Repository gespeicherten Predictor aus. |
| Parameter | |
| @predictionName | Name des Predictors, unter dem dieser im Repository gespeichert ist. |

Beispiel:

```
PRINT q.Improve.PredictorInfo( 'PredictorCity')
```

Routine Improve.PredictorPredict

Typ Skalarwertfunktion

Beschreibung Sagt auf Grundlage eines vorher erstellten Predictors den Wert für eine Zielspalte vorher.

Parameter

@predictionName Name des Predictors, unter dem dieser im Repository gespeichert werden soll.

@values Eingabewerte zur Vorhersage.

Beispiel:

```
SELECT Name, Rating,
       q.Improve.PredictorPredict(
           'PredictorCustomerRating',
           q.Tools.Param( Age, 0 ) +
           q.Tools.Param( MaritalStatus, 0 ) +
           q.Tools.Param( Sex, 0 ) +
           q.Tools.Param( State, 1 ) )
FROM   Persons.Customers
WHERE  Rating <> q.Improve.PredictorPredict(
           'PredictorCustomerRating',
           q.Tools.Param( Age, 0 ) +
           q.Tools.Param( MaritalStatus, 0 ) +
           q.Tools.Param( Sex, 0 ) +
           q.Tools.Param( State, 1 ) )
```

```
SELECT PostalCode, City,
       q.Improve.PredictorPredict(
           'PredictorCity', q.Tools.Param( PostalCode, 1 ) )
FROM   Persons.Customers
WHERE  City <> q.Improve.PredictorPredict(
           'PredictorCity', q.Tools.Param( PostalCode, 1 ) )
```

A.8 Tools

Routine Tools.AddPlugin

Typ Gespeicherte Prozedur

Beschreibung Registriert ein Plugin zur Verwendung im Datenqualitäts-Framework.

Parameter

@script Name des Predictors, unter dem dieser im Repository gespeichert werden soll.

@assemblyName Name, unter der das Skript geladen werden soll.

@references Zusätzliche Assemblies, die zum Übersetzen des Skripts benötigt werden. Bei mehreren Assemblies werden diese durch das ,|'-Zeichen getrennt.

Beispiel:

```
EXEC q.Tools.AddPlugin
    'C:\FreD\samples\Predictor.cs',
    'SamplePredictor',
    'C:\FreD\Core.dll|System.Data.dll'

EXEC q.Tools.AddPlugin
    'namespace MyEncoder
    {
        public class UpperTextPreparer :
            Fred.Text.Encoder.GenericTextPreparer
        {
            protected override string DoEncode( string text )
            {
                base.IgnoreCase = true;
                return base.DoEncode( text );
            }
        }
    }',
    'SampleEncoder',
    'C:\FreD\Core.dll'
```

| | |
|---------------------|--|
| Routine | Tools.BuildFrequencyTable |
| Typ | Gespeicherte Prozedur |
| Beschreibung | Erstellt automatisch eine Häufigkeitstabelle für alle in den Spalten vorkommenden Werte. |
| Parameter | |
| @frequencyTable | Name der Häufigkeitstabelle, die im Repository mit dem Präfix „freq“ angelegt wird. |
| @table | Tabellenname. |
| @fields | Spalten, für dessen Werte die Häufigkeitstabelle erstellt wird. |
| @where | WHERE-Klausel zum Einschränken der zu untersuchenden Daten. |
| @distinct | Wenn true, wird ein Token innerhalb eines Datensatzes nur einmal gezählt. |
| @useTokenizer | Wenn true, Verwendung der <i>GenericTokenizer</i> -Klasse, ansonsten kein Auftrennen. |

Beispiel:

```
EXEC q.Tools.BuildFrequencyTable
    'PersonsCustomers',
    'Sample.Persons.Customers',
    'Name, Street, StreetNumber, PostalCode, City',
    null, 1, 0
```

| | |
|---------------------|---|
| Routine | Tools.Log |
| Typ | Gespeicherte Prozedur |
| Beschreibung | Prozedur zum Protokollieren ins Eventlog, in die Datenbank oder in eine Datei. |
| Parameter | |
| @method | Bestimmt ob in das Eventlog, in die Datenbank oder in eine CSV-Datei protokolliert wird (Eventlog: „event:“, Datenbank: „database:“, Datei: „file:“). |
| @category | Kategorie des Protokolleintrags (z.B. Info, Warning, Error). |
| @info | Zu protokollierender Text. |

Beispiel:

```
EXEC q.Tools.Log 'file://c:\Sample.log', 'Sample', 'Duplicate found!'

EXEC q.Tools.Log 'event:', 'Sample', 'Duplicate found!'

-- in Datenbank q in Tabelle LogTable
EXEC q.Tools.Log 'database://LogTable', 'Sample', 'Duplicate found!'
```

| | |
|---------------------|--|
| Routine | Tools.Param |
| Typ | Skalarwertfunktion |
| Beschreibung | Funktion zum Übergeben mehrerer Werte über einen einzigen Übergabeparameter für die Funktion <i>Improve.PredictorPredict</i> . |
| Parameter | |
| @value | Zusammensetzende Werte. |
| @lastParam | True für den letzten Übergabewert. |

Beispiel:

```
SELECT q.Tools.Param( FirstName, 0 ) +
       q.Tools.Param( Name, 0 ) +
       q.Tools.Param( City, 1 )
FROM   Persons.Customers
```

B Klassenbibliothek

B.1 Allgemein

Anhang A enthält eine Auflistung aller Klassen, die für die Verwendung vom Framework aus durch SQL-Anweisungen relevant sind. Jede Klasse wird mit einer kurzen Beschreibung und mit dessen öffentlichen Properties, dem Assemblynamen, dem Namensraum, sowie einem kurzen Beispiel dokumentiert. Auf weitere Elemente der Klasse, wie Methoden usw. wird nicht eingegangen, da sie für die Verwendung über SQL nicht von Bedeutung sind. Das Beispiel zu jeder Klasse orientiert sich an einem einfachen Datenmodell, das dem Anhang C entnommen werden kann und sich in der Datenbank „Sample“ befindet.

B.2 Basis

B.2.1 Encoder

Namensraum FreD.Number.Encoder

Assembly Core

Klasse DecimalPlacesEncoder

Beschreibung Ermittelt die Anzahl Nachkommastellen einer Fließkommazahl.

Eigenschaften

Tokenizer Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden.

Sort Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false).

Beispiel:

```
SELECT DISTINCT q.Text.Encode(  
    ListPrice,  
    'FreD.Number.Encoder.DecimalPlacesEncoder',  
    'Core', null )  
FROM Sales.Articles
```

Namensraum FreD.Number.Encoder
Assembly Core
Klasse CheckDigitsEncoder

Beschreibung Berechnen einer Prüfziffer auf Modulo-Basis

Eigenschaften

Tokenizer Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden.

Sort Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false).

Modulo Konstante, durch die dividiert wird (Default: 10).

Weights Gewichte pro Ziffernstelle.

Mappings Umsetzen von Prüfziffern, z.B. errechnet Zahl 10 in X.

SumOfDigits Bestimmt, ob bei der Berechnung der einzelnen Ziffernstellen aus dem Ergebnis die Quersumme gebildet wird.

Beispiel:

```
-- ISBN-10 Prüfziffer
UPDATE Sales.Articles
SET    ProductNumber = q.Number.Encode( ProductNumber,
    'FreD.Number.Encoder.CheckDigitsEncoder',
    'Core',
    'Weights=10|9|8|7|6|5|4|3|2, Modulo=11, Mappings=10:X' )
```

Namensraum FreD.Number.Encoder
Assembly Core
Klasse VerhoeffCheckDigitsEncoder

Beschreibung Berechnen einer Prüfziffer nach dem Verhoeff-Algorithmus
 (Implementierung nach: J. Kirtland: Identification Numbers and Check Digit Schemes; The Mathematical Association of America; 2001; S. 152ff).

Eigenschaften

Tokenizer Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden.

Sort Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false).

Beispiel:

```
SELECT ProductNumber,
       q.Number.Encode( ProductNumber,
                        'FreD.Number.Encoder.VerhoeffCheckDigitsEncoder',
                        'Core', null)
FROM Sales.Articles

SELECT q.Number.Decode(
       '0016342765',
       'FreD.Number.Encoder.VerhoeffCheckDigitsEncoder',
       'Core', null)

SELECT q.Number.Decode( -- Ausnahme wegen falscher Prüfziffer
       '0016342761',
       'FreD.Number.Encoder.VerhoeffCheckDigitsEncoder',
       'Core', null)
```

| | |
|-------------------|---------------------|
| Namensraum | FreD.Text.Encoder |
| Assembly | Core |
| Klasse | GenericTextPreparer |

Beschreibung Filtert aus einer Zeichenkette einzelne Zeichen heraus und wandelt die Zeichenkette in Großbuchstaben (abhängig von der Eigenschaft IgnoreCase).

Eigenschaften

| | |
|--------------------|---|
| Tokenizer | Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden. |
| Sort | Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Standard: false). |
| IgnoreCase | Wandelt Zeichenkette in Großbuchstaben um (Default: false). |
| IgnoreBlank | Filtert Leerzeichen aus (Default: false). |
| IgnoreNotPrintable | Filtert nicht-druckbare Zeichen aus (Default: true). |
| IgnoreChars | Ignoriert die in der Eigenschaft übergebenen Zeichen (Default: null). |

Beispiel:

```
SELECT Description,
       q.Number.Encode( Description,
                        'FreD.Text.Encoder.GenericTextPreparer',
                        'Core',
                        'IgnoreCase=true, IgnoreBlank=true, IgnoreChars=''AEIOU.,;'' )
FROM Sales.Articles
WHERE Description IS NOT NULL
```

qSQL

| | |
|---------------------|--|
| Namensraum | FreD.Text.Encoder |
| Assembly | Core |
| Klasse | RemoveStopwordsTextPreparer |
| Beschreibung | Entfernt Stoppwörter aus einer Zeichenkette. |

Eigenschaften

| | |
|------------|--|
| Tokenizer | Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer). |
| Sort | Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false). |
| Repository | Datenbanktabelle mit Stoppwörtern (Default: „Stopwords“, entspricht Tabelle „q.Repository.stdStopwords“ mit englischsprachigen Stoppwörtern, für deutschsprachige Stoppwörter Eigenschaft auf „StopwordsGerman“ setzen). |

Beispiel:

```
SELECT Description,
       q.Number.Encode( Description,
                        'FreD.Text.Encoder.RemoveStopwordsTextPreparer',
                        'Core', null )
FROM Sales.Articles
WHERE Description IS NOT NULL
```

qSQL

| | |
|-------------------|----------------------------|
| Namensraum | FreD.Text.Encoder |
| Assembly | CaumannStemmerTextPreparer |
| Klasse | CaumannStemmerTextPreparer |

Beschreibung Führt den Stemming-Algorithmus von J. Caumann für deutschsprachigen Text durch.

Eigenschaften

| | |
|-----------|--|
| Tokenizer | Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Standard: GenericTokenizer). |
| Sort | Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false). |
| Repeat | Legt fest, wie oft der Stemming-Algorithmus für jedes einzelne Wort durchlaufen werden soll (Default: 1). |

Beispiel:¹²

```
SELECT Description,
       q.Text.Encode( Description,
                     'FreD.Text.Encoder.CaumannStemmerTextPreparer',
                     'CaumannStemmerTextPreparer',
                     'Repeat=2, Sort=true' )
FROM Sales.Articles
WHERE Description IS NOT NULL
```

¹² Das SQL-Beispiel funktioniert nur mit dem Plugin „CaumannStemmerTextPreparer“

| | |
|-------------------|----------------------------|
| Namensraum | FreD.Text.Encoder |
| Assembly | KrovetzStemmerTextPreparer |
| Klasse | KrovetzStemmerTextPreparer |

Beschreibung Führt den Stemming-Algorithmus von R. Krovetz durch.

Eigenschaften

| | |
|-----------|---|
| Tokenizer | Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer). |
| Sort | Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false). |

Beispiel:¹³

```
SELECT Description,
       q.Text.Encode( Description,
                     'FreD.Text.Encoder.KrovetzStemmerTextPreparer',
                     'KrovetzStemmerTextPreparer',
                     null )
FROM Sales.Articles
WHERE Description IS NOT NULL
```

¹³ Das SQL-Beispiel funktioniert nur mit dem Plugin „KrovetzStemmerTextPreparer“

qSQL

| | |
|-------------------|---------------------------|
| Namensraum | FreD.Text.Encoder |
| Assembly | LovinsStemmerTextPreparer |
| Klasse | LovinsStemmerTextPreparer |

Beschreibung Führt den Stemming-Algorithmus von J. Lovins durch.

Eigenschaften

| | |
|-----------|---|
| Tokenizer | Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer). |
| Sort | Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false). |
| Repeat | Legt fest, wie oft der Stemming-Algorithmus für jedes einzelne Wort durchlaufen werden soll (Default: 1). |

Beispiel:¹⁴

```
SELECT Description,
       q.Text.Encode( Description,
                     'FreD.Text.Encoder.LovinsStemmerTextPreparer',
                     'LovinsStemmerTextPreparer',
                     null )
FROM Sales.Articles
WHERE Description IS NOT NULL
```

¹⁴ Das SQL-Beispiel funktioniert nur mit dem Plugin „LovinsStemmerTextPreparer“

| | |
|---------------------|---|
| Namensraum | FreD.Text.Encoder |
| Assembly | PaiceHuskStemmerTextPreparer |
| Klasse | PaiceHuskStemmerTextPreparer |
| Beschreibung | Führt den Stemming-Algorithmus von C. Paice und G. Husk durch |

Eigenschaften

| | |
|-----------|---|
| Tokenizer | Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer). |
| Sort | Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false). |
| Repeat | Legt fest, wie oft der Stemming-Algorithmus für jedes einzelne Wort durchlaufen werden soll (Default: 1). |

Beispiel:¹⁵

```
SELECT Description,
       q.Text.Encode( Description,
                     'FreD.Text.Encoder.PaiceHuskStemmerTextPreparer',
                     'PaiceHuskStemmerTextPreparer',
                     null )
FROM Sales.Articles
WHERE Description IS NOT NULL
```

¹⁵ Das SQL-Beispiel funktioniert nur mit dem Plugin „PaiceHuskStemmerTextPreparer“

| | |
|-------------------|---------------------------|
| Namensraum | FreD.Text.Encoder |
| Assembly | PorterStemmerTextPreparer |
| Klasse | PorterStemmerTextPreparer |

Beschreibung Führt den Stemming-Algorithmus von M.F. Porter durch.

Eigenschaften

| | |
|-----------|---|
| Tokenizer | Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer). |
| Sort | Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false). |
| Repeat | Legt fest, wie oft der Stemming-Algorithmus für jedes einzelne Wort durchlaufen werden soll (Default: 1). |
| Country | Legt die Sprache des Textes fest (Default: german, andere Werte: danish, dutch, english, finnish, french, german, italian, norwegian, portuguese, russian, spanish, swedish). |

Beispiel:¹⁶

```
SELECT Description,
       q.Text.Encode( Description,
                     'FreD.Text.Encoder.PorterStemmerTextPreparer',
                     'PorterStemmerTextPreparer',
                     'Country=english' )
FROM sample.Sales.Articles
WHERE Description IS NOT NULL
```

¹⁶ Das SQL-Beispiel funktioniert nur mit dem Plugin „PorterStemmerTextPreparer“

| | |
|---------------------|---|
| Namensraum | FreD.Text.Encoder |
| Assembly | UEALiteStemmerTextPreparer |
| Klasse | UEALiteStemmerTextPreparer |
| Beschreibung | Implementiert den UEA (University of East-Anglia)-Lite Stemmer. |

Eigenschaften

| | |
|-----------|---|
| Tokenizer | Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer). |
| Sort | Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false). |
| Repeat | Legt fest, wie oft der Stemming-Algorithmus für jedes einzelne Wort durchlaufen werden soll (Default: 1). |

Beispiel:¹⁷

```
SELECT Description,
       q.Text.Encode( Description,
                     'FreD.Text.Encoder.UEALiteStemmerTextPreparer',
                     'UEALiteStemmerTextPreparer',
                     null )
FROM sample.Sales.Articles
WHERE Description IS NOT NULL
```

¹⁷ Das SQL-Beispiel funktioniert nur mit dem Plugin „UEALiteStemmerTextPreparer“

qSQL

| | |
|-------------------|------------------------|
| Namensraum | FreD.Text.Encoder |
| Assembly | Core |
| Klasse | CharSortPatternEncoder |

Beschreibung Gibt die Zeichen einer Zeichenkette sortiert zurück.

Eigenschaften

| | |
|--------------------|---|
| Tokenizer | Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer). |
| Sort | Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false). |
| IgnoreMultipleChar | Berücksichtigt das gleiche mehrfach auftretende Zeichen nur einmal (Default: true). |
| UpperCase | Konvertiert in Großbuchstaben (Default: true). |

Beispiel:

```
SELECT DISTINCT q.Text.Encode(
    Name,
    'FreD.Text.Encoder.CharSortPatternEncoder',
    'Core',
    'IgnoreMultipleChar=true, UpperCase=true' )
FROM Persons.Customers
```

qSQL

| | |
|-------------------|----------------------------------|
| Namensraum | FreD.Text.Encoder |
| Assembly | Core |
| Klasse | DataTypeRecognisedPatternEncoder |

Beschreibung Ermittelt den Datentyp eines Wertes.

Eigenschaften

| | |
|-----------|---|
| Tokenizer | Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer). |
| Sort | Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false). |
| Translate | Gibt einen .NET-Datentyp (Translate=false) oder einen SQL-Datentyp (Translate=true, Default) zurück. |

Beispiel:

```
SELECT DISTINCT q.Text.Encode (
    PostalCode,
    'FreD.Text.Encoder.DataTypeRecognisedPatternEncoder',
    'Core',
    'Translate=true' )
FROM Persons.Customers
```

| | |
|-------------------|---------------------------|
| Namensraum | FreD.Text.Encoder |
| Assembly | Core |
| Klasse | NamedEntityPatternEncoder |

Beschreibung Erkennt vordefinierte Entitäten aus einer Nachschlageliste, wobei Nachschlagetabellen sowohl normale Nachschlagetexte, aber auch reguläre Ausdrücke und auch kodierte Nachschlagewerte enthalten können.

Eigenschaften

| | |
|--------------|--|
| Tokenizer | Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer) |
| Sort | Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false). |
| Repository | Nachschlagetabelle zur Identifikation von Entitäten (Default: NamedEntity, entspricht Tabelle „Repository.stdNamedEntity“). |
| Encoder | Klassenname eines Encoders, der auf die Zeichenkette angewendet wird, um danach diesen Wert in der Nachschlagetabelle zu suchen. |
| UnkownEntity | Zeichenkette für einen nicht gefundenen Eintrag (Default: unknown). |
| ExtractValue | Extrahiert aus einer Zeichenkette nur das semantische Objekt, das in <i>ExtractValue</i> angegeben wurde. |

Beispiel:

```
SELECT DISTINCT Name,
    q.Text.Encode(Name,
        'FreD.Text.Encoder.NamedEntityPatternEncoder',
        'Core',
        'Encoder=FreD.Text.Encoder.CharSortPatternEncoder' )
FROM Persons.Customers
WHERE q.Text.Encode(Name,
    'FreD.Text.Encoder.NamedEntityPatternEncoder',
    'Core',
    'Encoder=FreD.Text.Encoder.CharSortPatternEncoder' ) <>
    '{FirstName} {LastName} '

SELECT DISTINCT Name,
    q.Text.Encode(Name,
        'FreD.Text.Encoder.NamedEntityPatternEncoder',
        'Core',
```

```
'ExtractValue=FirstName' )
FROM Persons.Customers
```

Namensraum FreD.Text.Encoder

Assembly Core

Klasse OmissionKeyPatternEncoder

Beschreibung Liefert einen Code nach dem Omission Key-Algorithmus von J. Pollock und A. Zamora zurück.

Eigenschaften

Tokenizer Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer).

Sort Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false).

Beispiel:

```
SELECT DISTINCT Name,
    q.Text.Encode( Name,
        'FreD.Text.Encoder.OmissionKeyPatternEncoder',
        'Core',
        null )
FROM Persons.Customers
ORDER BY 2
```

qSQL

| | |
|---------------------|---|
| Namensraum | FreD.Text.Encoder |
| Assembly | Core |
| Klasse | SkeletonKeyPatternEncoder |
| Beschreibung | Liefert einen Code nach dem Skeleton Key-Algorithmus von J. Pollock und A. Zamora zurück. |

Eigenschaften

| | |
|-----------|---|
| Tokenizer | Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer). |
| Sort | Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false). |

Beispiel:

```
SELECT DISTINCT Name,
    q.Text.Encode(Name,
        'FreD.Text.Encoder.SkeletonKeyPatternEncoder',
        'Core',
        null )
FROM Persons.Customers
ORDER BY 2
```


qSQL

| | |
|-------------------|----------------------------|
| Namensraum | FreD.Text.Encoder |
| Assembly | Core |
| Klasse | ReverseOrderPatternEncoder |

Beschreibung Die Zeichen einer Zeichenkette werden in umgekehrter Reihenfolge zurückgegeben.

Eigenschaften

| | |
|-----------|---|
| Tokenizer | Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer). |
| Sort | Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false). |

Beispiel:

```
SELECT DISTINCT Name,
    q.Text.Encode(Name,
        'FreD.Text.Encoder.ReverseOrderPatternEncoder',
        'Core',
        null )
FROM Persons.Customers
```

| | |
|----------------------|---|
| Namensraum | FreD.Text.Encoder |
| Assembly | Core |
| Klasse | SimpleOrderPatternEncoder |
| Beschreibung | Ersetzt jeden Groß-, Kleinbuchstaben, jede Zahlen und andere Zeichen durch ein gemeinsames Kürzel. |
| Eigenschaften | |
| Tokenizer | Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer). |
| Sort | Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false). |
| LowerLetter | Zeichenkette, die Kleinbuchstaben ersetzt (Default: ‚A‘). |
| UpperLetter | Zeichenkette, die Großbuchstaben ersetzt (Default: ‚a‘). |
| Digit | Zeichenkette, die Zahlen ersetzt (Default: ‚9‘). |
| Other | Zeichenkette, die alle anderen Zeichen außer Groß-/Kleinbuchstaben und Zahlen ersetzt (Default: ‚*‘). |

Beispiel:

```
SELECT q.Text.Encode( PostalCode,
    'FreD.Text.Encoder.SimplePatternEncoder',
    'Core',
    'Other=' ),
    COUNT(*)
FROM Persons.Customers
GROUP BY q.Text.Encode( PostalCode,
    'FreD.Text.Encoder.SimplePatternEncoder',
    'Core',
    'Other=' )
```

qSQL

| | |
|-------------------|-----------------------|
| Namensraum | FreD.Text.Encoder |
| Assembly | Core |
| Klasse | Adler32HashingEncoder |

Beschreibung Berechnet den Hashwert nach dem Adler32-Algorithmus.

Eigenschaften

| | |
|-----------|---|
| Tokenizer | Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer). |
| Sort | Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false). |

Beispiel:

```
SELECT DISTINCT q.Text.Encode( PostalCode,
    'FreD.Text.Encoder.Adler32HashingEncoder',
    'Core',
    NULL)
FROM Persons.Customers
```

qSQL

| | |
|----------------------|---|
| Namensraum | FreD.Text.Encoder |
| Assembly | Core |
| Klasse | SumHashingEncoder |
| Beschreibung | Berechnet einen Hashwert, indem es die Quersumme bildet. |
| Eigenschaften | |
| Tokenizer | Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer). |
| Sort | Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false). |

Beispiel:

```
SELECT DISTINCT q.Text.Encode( PostalCode,
    'FreD.Text.Encoder.SumHashingEncoder',
    'Core',
    NULL)
FROM Persons.Customers
```

| | |
|---------------------|--|
| Namensraum | FreD.Text.Encoder |
| Assembly | Core |
| Klasse | CaverPhoneticEncoder |
| Beschreibung | Erstellt den phonetischen Code nach dem Caverphone-Algorithmus |

Eigenschaften

| | |
|-----------|---|
| Tokenizer | Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer). |
| Sort | Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false). |
| MaxLength | Maximaler Länge des phonetischen Codes (Default: 4). |

Beispiel:

```
SELECT DISTINCT q.Text.Encode(Name,
    'FreD.Text.Encoder.CaverPhoneticEncoder',
    'Core',
    'MaxLength=6')
FROM Persons.Customers
```

qSQL

| | |
|---------------------|--|
| Namensraum | FreD.Text.Encoder |
| Assembly | Core |
| Klasse | ColognePhoneticEncoder |
| Beschreibung | Erstellt den Kölner Phonetik-Code nach H.-J. Postel. |

Eigenschaften

| | |
|-----------|---|
| Tokenizer | Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer). |
| Sort | Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false). |
| MaxLength | Maximaler Länge des phonetischen Codes (Default: 4). |

Beispiel:

```
SELECT DISTINCT Name, q.Text.Encode( Name,
    'FreD.Text.Encoder.ColognePhoneticEncoder',
    'Core',
    null)
FROM Persons.Customers
ORDER BY 2
```

| | |
|---------------------|--|
| Namensraum | FreD.Text.Encoder |
| Assembly | Core |
| Klasse | DaitchMokotoffPhoneticEncoder |
| Beschreibung | Erweiterung des Soundex-Codes von R. Daitch und G. Mokotoff. |

Eigenschaften

| | |
|------------------|---|
| Tokenizer | Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer). |
| Sort | Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false). |
| MaxLength | Maximaler Länge des phonetischen Codes (Default: 6). |
| UseAlternateCode | Falls vorhanden, wird ein zweiter alternativer Code zurückgegeben (Default: false). |

Beispiel:

```
SELECT DISTINCT Name, q.Text.Encode( Name,
    'FreD.Text.Encoder.DaitchMokotoffPhoneticEncoder',
    'Core',
    'UseAlternateCode=false')
FROM Persons.Customers
ORDER BY 2
```

| | |
|-------------------|--------------------------------|
| Namensraum | FreD.Text.Encoder |
| Assembly | Core |
| Klasse | IBMAAlphaSearchPhoneticEncoder |

Beschreibung Erzeugt einen phonetischen Code nach dem „IBM Alpha Inquiry System Personal Name Encoding“-Algorithmus. Ausnahmen für einen zweiten und dritten Durchgang wurden nicht implementiert.

Eigenschaften

| | |
|-----------|---|
| Tokenizer | Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer). |
| Sort | Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false). |
| MaxLength | Maximaler Länge des phonetischen Codes (Default: 14). |

Beispiel:

```
SELECT DISTINCT Name, q.Text.Encode( Name,
    'FreD.Text.Encoder.IBMAAlphaSearchPhoneticEncoder',
    'Core',
    null)
FROM Persons.Customers
ORDER BY 2
```


| | |
|-------------------|-----------------------|
| Namensraum | FreD.Text.Encoder |
| Assembly | Core |
| Klasse | NysiisPhoneticEncoder |

Beschreibung Erzeugt einen phonetischen Code nach dem NYSIIS-Algorithmus („New York State Identification and Intelligence System“).
Alternative erweiterte Version, siehe <http://www.dropby.com/NYSIIS.html>.

Eigenschaften

| | |
|-----------|---|
| Tokenizer | Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer). |
| Sort | Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false). |
| MaxLength | Maximaler Länge des phonetischen Codes (Default: 4). |

Beispiel:

```
SELECT DISTINCT Name, q.Text.Encode( Name,
    'FreD.Text.Encoder.NysiisPhoneticEncoder',
    'Core',
    null )
FROM Persons.Customers
ORDER BY 2
```

Namensraum FreD.Text.Encoder
Assembly Core
Klasse ONCAPhoneticEncoder

Beschreibung Erzeugt einen phonetischen Code nach dem ONCA („Oxford Name Compression Algorithm“), der NYSIIS und Soundex kombiniert.

Eigenschaften

Tokenizer Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer).

Sort Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false).

MaxLength Maximaler Länge des phonetischen Codes (Default: 4).

Beispiel:

```
SELECT DISTINCT Name, q.Text.Encode( Name,
    'FreD.Text.Encoder.ONCAPhoneticEncoder',
    'Core',
    null )
FROM Persons.Customers
ORDER BY 2
```

| | |
|---------------------|--|
| Namensraum | FreD.Text.Encoder |
| Assembly | Core |
| Klasse | PhonemPhoneticEncoder |
| Beschreibung | Erzeugt einen phonetischen Code nach dem PHONEM-Algorithmus von G. Wilde und C. Meyer. |

Eigenschaften

| | |
|-----------|---|
| Tokenizer | Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer). |
| Sort | Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false). |
| MaxLength | Maximaler Länge des phonetischen Codes (Default: 4). |

Beispiel:

```
SELECT DISTINCT Name, q.Text.Encode( Name,
    'FreD.Text.Encoder.PhonemPhoneticEncoder',
    'Core',
    null )
FROM Persons.Customers
ORDER BY 2
```

| | |
|-------------------|-----------------------|
| Namensraum | FreD.Text.Encoder |
| Assembly | Core |
| Klasse | PhonexPhoneticEncoder |

Beschreibung Erzeugt einen phonetischen Code nach dem Phonex-Algorithmus von A.J. Lait und B. Randell¹⁸, einer Variation des Soundex-Algorithmus.

Eigenschaften

| | |
|-----------|---|
| Tokenizer | Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer). |
| Sort | Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false). |
| MaxLength | Maximaler Länge des phonetischen Codes (Default: 4). |

Beispiel:

```
SELECT DISTINCT Name, q.Text.Encode( Name,
    'FreD.Text.Encoder.PhonexPhoneticEncoder',
    'Core',
    null )
FROM Persons.Customers
ORDER BY 2
```

¹⁸ Siehe Lait, A.J., Randell, B.: An Assessment of Name Matching Algorithms, Technical Report; Department of Computing Science, University of Newcastle upon Tyne; Newcastle; 1993

qSQL

| | |
|---------------------|---|
| Namensraum | FreD.Text.Encoder |
| Assembly | Core |
| Klasse | RethSchekPhoneticEncoder |
| Beschreibung | Erzeugt einen phonetischen Code für deutsche Namen nach einem Algorithmus von H.-P. von Reth und H.-J. Schek. |

Eigenschaften

| | |
|-----------|---|
| Tokenizer | Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer). |
| Sort | Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false). |
| MaxLength | Maximaler Länge des phonetischen Codes (Default: 4). |

Beispiel:

```
SELECT DISTINCT Name, q.Text.Encode( Name,
    'FreD.Text.Encoder.RethSchekPhoneticEncoder',
    'Core',
    null )
FROM Persons.Customers
ORDER BY 2
```

| | |
|-------------------|------------------------|
| Namensraum | FreD.Text.Encoder |
| Assembly | Core |
| Klasse | SoundexPhoneticEncoder |

Beschreibung Erzeugt einen phonetischen Code nach dem Soundex-Algorithmus von M. Odell und R. Russel.

Eigenschaften

| | |
|-----------|---|
| Tokenizer | Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer). |
| Sort | Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false). |
| MaxLength | Maximaler Länge des phonetischen Codes (Default: 4). |

Beispiel:

```
SELECT DISTINCT Name, q.Text.Encode( Name,  
    'FreD.Text.Encoder.SoundexPhoneticEncoder',  
    'Core',  
    null )  
FROM Persons.Customers  
ORDER BY 2
```

| | |
|----------------------|---|
| Namensraum | FreD.Text.Encoder |
| Assembly | Core |
| Klasse | SoundexPhoneticExtendedEncoder |
| Beschreibung | Erzeugt einen phonetischen Code nach dem Soundex-Algorithmus von M. Odell und R. Russel. Außerdem wird das erste Zeichen auch kodiert und es existieren zwei zusätzliche Eigenschaften. |
| Eigenschaften | |
| Tokenizer | Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer). |
| Sort | Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false). |
| MaxLength | Maximaler Länge des phonetischen Codes (Default: 4). |
| FillCode | Ergänzt den Code um das Zeichen ,0' am Ende, wenn weniger als MaxLength (default: true). |
| Mapping | 26-Zeichen lange Zeichenkette mit dem Mapping der Buchstaben von A-Z (default: „0123012-02245501262301-2-2“). |

Beispiel:

```
SELECT DISTINCT Name, q.Text.Encode( Name,
    'FreD.Text.Encoder.SoundexPhoneticExtendedEncoder',
    'Core',
    'FillCode=false, Mapping=*123*12-*22455*12623*1-2-2' )
FROM Persons.Customers
ORDER BY 2
```

B.2.2 Matching

Namensraum FreD.Date.Matching

Assembly Core

Klasse DateMatching

Beschreibung Vergleicht zwei Datumswerte und gibt deren Ähnlichkeit als Wert zwischen 0 und 1 zurück.

Eigenschaften

MaxTimeSpan Zeitintervall, der als Basis zur Berechnung des Ähnlichkeitswertes nach folgender Formel verwendet wird: $\text{Abs}(\text{Datum1} - \text{Datum2}) / \text{MaxTimeSpan}$ (Default: 7 Tage).

Beispiel:

```
SELECT OrderDate, ShipDate,
       q.Date.Match( OrderDate, ShipDate,
                     'FreD.Date.Matching.DateMatching',
                     'Core',
                     'MaxTimeSpan=14.00:00:00' )
FROM Sales.Orders
```


| | |
|-------------------|------------------------|
| Namensraum | FreD.Distance.Matching |
| Assembly | Core |
| Klasse | GeoDistanceMatching |

Beschreibung Ermittelt die Entfernung zwischen zwei Postleitzahlen über deren Geodaten (Längen-/Breitengrad) und berechnet daraufhin die Ähnlichkeit zwischen diesen beiden.

Eigenschaften

| | |
|-------------|---|
| MaxDistance | Distanz in Kilometern, die als Basis zur Berechnung des Ähnlichkeitswertes nach folgender Formel verwendet wird: Distanz zwischen den beiden Postleitzahlen / MaxDistance (Default: 100). |
| Country | Land für die übergebenen Postleitzahlen (Default: DE, andere Werte US bzw. distinkte Werte in Spalte Country der Tabelle Repository.sysGeoData). |

Beispiel:

```
SELECT a.CustomerId, b.CustomerId, a.Name, b.Name,
       a.PostalCode, b.PostalCode,
       q.Distance.Match( a.PostalCode, b.PostalCode,
                        'FreD.Distance.Matching.GeoDistanceMatching',
                        'Core',
                        'MaxDistance=20, Country=DE' )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Distance.Match( a.PostalCode, b.PostalCode,
                        'FreD.Distance.Matching.GeoDistanceMatching',
                        'Core',
                        'MaxDistance=20, Country=DE' ) > 0.0 AND
       a.CustomerId > b.CustomerId
```

Namensraum FreD.Number.Matching
Assembly Core
Klasse EuclidianDistanceMatching

Beschreibung Ermittelt die euklidische Distanz zwischen den übergebenen Zahlen und errechnet das entsprechende Ähnlichkeitsmaß zwischen 0 und 1.

Eigenschaften

Tokenizer Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden.

Beispiel:

```
SELECT a.ArticleId, b.ArticleId, a.Name, b.Name,
       a.ListPrice, b.ListPrice,
       q.Number.Match( a.ListPrice, b.ListPrice,
                       'FreD.Number.Matching.EuclidianDistanceMatching',
                       'Core', null )
FROM   Sales.Articles a, Sales.Articles b
WHERE  q.Number.Match( a.ListPrice, b.ListPrice,
                       'FreD.Number.Matching.EuclidianDistanceMatching',
                       'Core', null ) > 0.8 AND
       a.ArticleId > b.ArticleId
```

| | |
|-------------------|----------------------|
| Namensraum | FreD.Number.Matching |
| Assembly | Core |
| Klasse | SimpleNumberMatching |

Beschreibung Ermittelt die Ähnlichkeit zwischen zwei Zahlen als Wert zwischen 0 und 1.

Eigenschaften

| | |
|-------------|--|
| Tokenizer | Tokenizer zum Aufsplitten in mehrere Token. Dabei wird der Ähnlichkeitswert jedes Tokens mit jedem anderen berechnet und der größte Wert jeweils summiert. Die Summe wird durch die höhere Anzahl an Tokens dividiert. |
| MaxDistance | Maximale Distanz zwischen zwei Zahlen, die als Basis zur Berechnung des Ähnlichkeitswertes nach folgender Formel verwendet wird: $\text{Abs}(\text{Zahl1} - \text{Zahl2}) / \text{MaxDistance}$ (Default: 1). |

Beispiel:

```
SELECT a.ArticleId, b.ArticleId, a.Name, b.Name,
       a.ListPrice, b.ListPrice,
       q.Number.Match( a.ListPrice, b.ListPrice,
                       'FreD.Number.Matching.SimpleNumberMatching',
                       'Core',
                       'MaxDistance=100.0' )
FROM   Sales.Articles a, Sales.Articles b
WHERE  q.Number.Match( a.ListPrice, b.ListPrice,
                       'FreD.Number.Matching.SimpleNumberMatching',
                       'Core', 'MaxDistance=100.0' ) > 0.8 AND
       a.ArticleId > b.ArticleId
```

| | |
|-------------------|--------------------|
| Namensraum | FreD.Text.Matching |
| Assembly | Core |
| Klasse | ExactTextMatching |

Beschreibung Ermittelt die Ähnlichkeit zwischen zwei Zeichenketten zeichenbasiert. Gibt bei exakter Übereinstimmung 1, ansonsten 0 zurück.

Eigenschaften

| | |
|------------|--|
| Tokenizer | Tokenizer zum Aufsplitten in mehrere Token. Dabei wird der Ähnlichkeitswert jedes Tokens mit jedem anderen berechnet und der größte Wert jeweils summiert. Die Summe wird durch die höhere Anzahl an Tokens dividiert. |
| StartIndex | Startposition, ab der der Vergleich beginnen soll. |

Beispiel:

```
SELECT a.CustomerId, b.CustomerId, a.Name, b.Name,
       q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.ExactTextMatching',
                     'Core',
                     'StartIndex=2' )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.ExactTextMatching',
                     'Core',
                     'StartIndex=2' ) = 1.0 AND
       a.CustomerId > b.CustomerId
```

| | |
|---------------------|---|
| Namensraum | FreD.Text.Matching |
| Assembly | Core |
| Klasse | HammingApproximateMatching |
| Beschreibung | Ermittelt die Ähnlichkeit zwischen zwei Zeichenketten zeichenbasiert nach dem Algorithmus zur Berechnung der Hamming-Distanz als Wert zwischen 1 und 0. |

Eigenschaften

| | |
|-----------|--|
| Tokenizer | Tokenizer zum Aufsplitten in mehrere Token. Dabei wird der Ähnlichkeitswert jedes Tokens mit jedem anderen berechnet und der größte Wert jeweils summiert. Die Summe wird durch die höhere Anzahl an Tokens dividiert. |
|-----------|--|

Beispiel:

```
SELECT a.CustomerId, b.CustomerId, a.Name, b.Name,
       q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.HammingApproximateMatching',
                     'Core', null )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.HammingApproximateMatching',
                     'Core', null ) > 0.9 AND
       a.CustomerId > b.CustomerId
ORDER BY 5
```

qSQL

| | |
|----------------------|--|
| Namensraum | FreD.Text.Matching |
| Assembly | Core |
| Klasse | HirschbergApproximateMatching |
| Beschreibung | Ermittelt die Ähnlichkeit zwischen zwei Zeichenketten zeichenbasiert nach dem LCS-Algorithmus von Hirschberg als Wert zwischen 1 und 0. |
| Eigenschaften | |
| Tokenizer | Tokenizer zum Aufsplitten in mehrere Token. Dabei wird der Ähnlichkeitswert jedes Tokens mit jedem anderen berechnet und der größte Wert jeweils summiert. Die Summe wird durch die höhere Anzahl an Tokens dividiert. |

Beispiel:

```

SELECT a.CustomerId, b.CustomerId, a.Name, b.Name,
       q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.HirschbergApproximateMatching',
                     'Core', null )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.HirschbergApproximateMatching',
                     'Core', null ) > 0.9 AND
       a.CustomerId > b.CustomerId
ORDER BY 5

```

qSQL

| | |
|-------------------|-------------------------|
| Namensraum | FreD.Text.Matching |
| Assembly | Core |
| Klasse | JaroApproximateMatching |

Beschreibung Ermittelt die Ähnlichkeit zwischen zwei Zeichenketten zeichenbasiert nach dem Algorithmus von Jaro als Wert zwischen 1 und 0.

Eigenschaften

Tokenizer Tokenizer zum Aufsplitten in mehrere Token. Dabei wird der Ähnlichkeitswert jedes Tokens mit jedem anderen berechnet und der größte Wert jeweils summiert. Die Summe wird durch die höhere Anzahl an Tokens dividiert.

Beispiel:

```
SELECT a.CustomerId, b.CustomerId, a.Name, b.Name,
       q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.JaroApproximateMatching',
                     'Core',
                     '{Tokenizer=FreD.Text.Tokenizer.GenericTokenizer} )' )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.JaroApproximateMatching',
                     'Core',
                     '{Tokenizer=FreD.Text.Tokenizer.GenericTokenizer}' ) > 0.9
AND a.CustomerId > b.CustomerId

ORDER BY 5
```

| | |
|-------------------|--------------------------------|
| Namensraum | FreD.Text.Matching |
| Assembly | Core |
| Klasse | JaroWinklerApproximateMatching |

Beschreibung Ermittelt die Ähnlichkeit zwischen zwei Zeichenketten zeichenbasiert nach dem angepassten Algorithmus von Jaro als Wert zwischen 1 und 0, indem die Übereinstimmung der ersten n Zeichen berücksichtigt wird.

Eigenschaften

| | |
|--------------------|--|
| Tokenizer | Tokenizer zum Aufsplitten in mehrere Token. Dabei wird der Ähnlichkeitswert jedes Tokens mit jedem anderen berechnet und der größte Wert jeweils summiert. Die Summe wird durch die höhere Anzahl an Tokens dividiert. |
| CommonPrefixLength | Anzahl zu überprüfender Zeichen am Anfang der Zeichenkette (Default: 4). Die Anzahl der Übereinstimmungen geht als Korrekturfaktor in das Ähnlichkeitsmaß ein. |

Beispiel:

```
SELECT a.CustomerId, b.CustomerId, a.Name, b.Name,
       q.Text.Match( a.Name, b.Name,
                    'FreD.Text.Matching.JaroWinklerApproximateMatching',
                    'Core',
                    'CommonPrefixLength=5' )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Text.Match( a.Name, b.Name,
                    'FreD.Text.Matching.JaroWinklerApproximateMatching',
                    'Core',
                    'CommonPrefixLength=5' ) > 0.9 AND
       a.CustomerId > b.CustomerId
ORDER BY 5
```


| | |
|-------------------|-------------------------------------|
| Namensraum | FreD.Text.Matching |
| Assembly | Core |
| Klasse | JaroWinklerLynchApproximateMatching |

Beschreibung Ermittelt die Ähnlichkeit zwischen zwei Zeichenketten zeichenbasiert nach einem angepassten Algorithmus von Jaro/Winkler als Wert zwischen 1 und 0.

Eigenschaften

| | |
|--------------------|--|
| Tokenizer | Tokenizer zum Aufsplitten in mehrere Token. Dabei wird der Ähnlichkeitswert jedes Tokens mit jedem anderen berechnet und der größte Wert jeweils summiert. Die Summe wird durch die höhere Anzahl an Tokens dividiert. |
| CommonPrefixLength | Anzahl zu überprüfender Zeichen am Anfang der Zeichenkette (Default: 4). Die Anzahl der Übereinstimmungen geht als Korrekturfaktor in das Ähnlichkeitsmaß ein. |

Beispiel:

```
SELECT a.CustomerId, b.CustomerId, a.Name, b.Name,
       q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.JaroWinklerLynchApproximateMatching',
                     'Core',
                     'CommonPrefixLength=5' )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.JaroWinklerLynchApproximateMatching',
                     'Core',
                     'CommonPrefixLength=5' ) > 0.9 AND
       a.CustomerId > b.CustomerId
ORDER BY 5
```

| | |
|-------------------|--------------------------------|
| Namensraum | FreD.Text.Matching |
| Assembly | Core |
| Klasse | LevenshteinApproximateMatching |

Beschreibung Ermittelt die Distanz zwischen zwei Zeichenketten zeichenbasiert nach dem Algorithmus von V. Levenshtein und F. Damerau und errechnet daraus die Ähnlichkeit als Wert zwischen 0 und 1.

Eigenschaften

Tokenizer Tokenizer zum Aufsplitten in mehrere Token. Dabei wird der Ähnlichkeitswert jedes Tokens mit jedem anderen berechnet und der größte Wert jeweils summiert. Die Summe wird durch die höhere Anzahl an Tokens dividiert.

Beispiel:

```
SELECT a.CustomerId, b.CustomerId, a.Name, b.Name,
       q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.LevenshteinApproximateMatching',
                     'Core',
                     null )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.LevenshteinApproximateMatching',
                     'Core',
                     null ) > 0.8 AND
       a.CustomerId > b.CustomerId
ORDER BY 5
```

| | |
|----------------------|---|
| Namensraum | FreD.Text.Matching |
| Assembly | Core |
| Klasse | LevenshteinHjelmqvistApproximateMatching |
| Beschreibung | <p>Ermittelt die Lebenshtein-Distanz zwischen zwei Zeichenketten zeichenbasiert nach dem performanteren Algorithmus von S. Hjelmqvist und errechnet daraus die Ähnlichkeit als Wert zwischen 0 und 1 (siehe http://www.codeproject.com/KB/recipes/Levenshtein.aspx).</p> |
| Eigenschaften | |
| Tokenizer | Tokenizer zum Aufsplitten in mehrere Token. Dabei wird der Ähnlichkeitswert jedes Tokens mit jedem anderen berechnet und der größte Wert jeweils summiert. Die Summe wird durch die höhere Anzahl an Tokens dividiert. |

Beispiel:

```
SELECT a.CustomerId, b.CustomerId, a.Name, b.Name,
       q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.LevenshteinHjelmqvistApproximateMatching',
                     'Core',
                     null )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.LevenshteinHjelmqvistApproximateMatching',
                     'Core',
                     null ) > 0.9 AND
       a.CustomerId > b.CustomerId
ORDER BY 5
```

| | |
|-------------------|--------------------------|
| Namensraum | FreD.Text.Matching |
| Assembly | Core |
| Klasse | Sift3ApproximateMatching |

Beschreibung Ermittelt die Distanz zwischen zwei Zeichenketten zeichenbasiert nach dem Sift3-Algorithmus und errechnet daraus die Ähnlichkeit als Wert zwischen 0 und 1 (siehe <http://siderite.blogspot.com/2007/04/super-fast-and-accurate-string-distance.html>).

Eigenschaften

Tokenizer Tokenizer zum Aufsplitten in mehrere Token. Dabei wird der Ähnlichkeitswert jedes Tokens mit jedem anderen berechnet und der größte Wert jeweils summiert. Die Summe wird durch die höhere Anzahl an Tokens dividiert.

Beispiel:

```
SELECT a.CustomerId, b.CustomerId, a.Name, b.Name,
       q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.Sift3ApproximateMatching',
                     'Core',
                     null )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.Sift3ApproximateMatching',
                     'Core',
                     null ) > 0.8 AND
       a.CustomerId > b.CustomerId
ORDER BY 5
```

| | |
|----------------------|---|
| Namensraum | FreD.Text.Matching |
| Assembly | Core |
| Klasse | TypewriterApproximateMatching |
| Beschreibung | Ermittelt ein zeichenbasiertes Distanzmaß abhängig von der Entfernung unterschiedlicher Zeichen auf der Tastatur und errechnet daraus die Ähnlichkeit als Wert zwischen 0 und 1. |
| Eigenschaften | |
| Tokenizer | Tokenizer zum Aufsplitten in mehrere Token. Dabei wird der Ähnlichkeitswert jedes Tokens mit jedem anderen berechnet und der größte Wert jeweils summiert. Die Summe wird durch die höhere Anzahl an Tokens dividiert. |
| ThresholdDistance | Ist dieser Wert < 0, dann wird die Anzahl an Tasten zwischen den beiden unterschiedlichen Zeichen auf der Tastatur verwendet. Andernfalls wird überprüft, ob ThresholdDistance größer als der Tastaturabstand ist und gegebenenfalls die maximale Diszanz (errechnet sich aus dem Tastaturlayout) verwendet (Default: 4). |
| TypewriterLayout | Zeichenkette die das Tastaturlayout beschreibt. Das , '-Zeichen trennt die einzelnen Tastaturreihen (Default: „1234567890ß QWERTZUIOPÜ ASDFGHJKLÖÄ YXCVBNM;:_“). |

Beispiel:

```

SELECT a.CustomerId, b.CustomerId, a.Name, b.Name,
       q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.TypewriterApproximateMatching',
                     'Core',
                     'ThresholdDistance=10' )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.TypewriterApproximateMatching',
                     'Core',
                     'ThresholdDistance=10' ) > 0.9 AND
       a.CustomerId > b.CustomerId
ORDER BY 5

```

Namensraum FreD.Text.Matching
Assembly Core
Klasse CityBlockApproximateMatching

Beschreibung Ermittelt die Ähnlichkeit zwischen zwei Zeichenketten tokenbasiert auf Grundlage gleicher und ungleicher Token als Wert zwischen 0 und 1 über die City-Block-Distanz.

Eigenschaften

Tokenizer Tokenizer zum Aufsplitten in mehrere Token. Dabei wird der Ähnlichkeitswert jedes Tokens mit jedem anderen berechnet und der größte Wert jeweils summiert. Die Summe wird durch die höhere Anzahl an Tokens dividiert.

TokenizerTokenBased Tokenizer zum Aufsplitten in mehrere Token zur Berechnung tokenbasierter Vergleichsalgorithmen (Default: GenericTokenizer).

Beispiel:

```
SELECT a.CustomerId, b.CustomerId, a.Name, b.Name,
       q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.CityBlockApproximateMatching',
                     'Core',
                     null )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.CityBlockApproximateMatching',
                     'Core',
                     null ) >= 0.5 AND
       a.CustomerId > b.CustomerId
ORDER BY 5
```

| | |
|-------------------|---------------------------|
| Namensraum | FreD.Text.Matching |
| Assembly | Core |
| Klasse | CosineApproximateMatching |

Beschreibung Ermittelt die Ähnlichkeit zwischen zwei Zeichenketten tokenbasiert auf Grundlage gleicher und ungleicher Token als Wert zwischen 0 und 1 über das Kosinus-Maß.

Eigenschaften

| | |
|---------------------|--|
| Tokenizer | Tokenizer zum Aufsplitten in mehrere Token. Dabei wird der Ähnlichkeitswert jedes Tokens mit jedem anderen berechnet und der größte Wert jeweils summiert. Die Summe wird durch die höhere Anzahl an Tokens dividiert. |
| TokenizerTokenBased | Tokenizer zum Aufsplitten in mehrere Token zur Berechnung tokenbasierter Vergleichsalgorithmen (Default: GenericTokenizer). |

Beispiel:

```
SELECT a.CustomerId, b.CustomerId, a.Name, b.Name,
       q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.CosineApproximateMatching',
                     'Core',
                     '{TokenizerTokenBased=FreD.Text.Tokenizer.NGramTokenizer,
                      NGramSize=3}' )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.CosineApproximateMatching',
                     'Core',
                     '{TokenizerTokenBased=FreD.Text.Tokenizer.NGramTokenizer,
                      NGramSize=3}' ) >= 0.5 AND
       a.CustomerId > b.CustomerId
ORDER BY 5
```

Namensraum FreD.Text.Matching
Assembly Core
Klasse DiceApproximateMatching

Beschreibung Ermittelt die Ähnlichkeit zwischen zwei Zeichenketten tokenbasiert auf Grundlage gleicher und ungleicher Token als Wert zwischen 0 und 1 über das Dice-Maß.

Eigenschaften

Tokenizer Tokenizer zum Aufsplitten in mehrere Token. Dabei wird der Ähnlichkeitswert jedes Tokens mit jedem anderen berechnet und der größte Wert jeweils summiert. Die Summe wird durch die höhere Anzahl an Tokens dividiert.

TokenizerTokenBased Tokenizer zum Aufsplitten in mehrere Token zur Berechnung tokenbasierter Vergleichsalgorithmen (Default: GenericTokenizer).

Beispiel:

```
SELECT a.CustomerId, b.CustomerId, a.Name, b.Name,
       q.Text.Match( a.Name, b.Name,
                    'FreD.Text.Matching.DiceApproximateMatching',
                    'Core',
                    '{Tokenizer=FreD.Text.Tokenizer.NGramTokenizer,
                     NGramSize=1}' )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Text.Match( a.Name, b.Name,
                    'FreD.Text.Matching.DiceApproximateMatching',
                    'Core',
                    '{Tokenizer=FreD.Text.Tokenizer.NGramTokenizer,
                     NGramSize=1}' ) >= 0.8 AND
       a.CustomerId > b.CustomerId
ORDER BY 5
```


| | |
|----------------------|--|
| Namensraum | FreD.Text.Matching |
| Assembly | Core |
| Klasse | EuclidianDistanceApproximateMatching |
| Beschreibung | Ermittelt die Ähnlichkeit zwischen zwei Zeichenketten tokenbasiert auf Grundlage gleicher und ungleicher Token als Wert zwischen 0 und 1 über über das Euklidische Distanzmaß. |
| Eigenschaften | |
| Tokenizer | Tokenizer zum Aufsplitten in mehrere Token. Dabei wird der Ähnlichkeitswert jedes Tokens mit jedem anderen berechnet und der größte Wert jeweils summiert. Die Summe wird durch die höhere Anzahl an Tokens dividiert. |
| TokenizerTokenBased | Tokenizer zum Aufsplitten in mehrere Token zur Berechnung tokenbasierter Vergleichsalgorithmen (Default: GenericTokenizer). |

Beispiel:

```

SELECT a.CustomerId, b.CustomerId, a.Name, b.Name,
       q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.EuclidianDistanceApproximateMatching',
                     'Core',
                     null )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.EuclidianDistanceApproximateMatching',
                     'Core',
                     null ) > 0 AND
       a.CustomerId > b.CustomerId
ORDER BY 5

```

| | |
|-------------------|----------------------------------|
| Namensraum | FreD.Text.Matching |
| Assembly | Core |
| Klasse | FellegiSunterApproximateMatching |

Beschreibung Ermittelt die Ähnlichkeit zwischen zwei Zeichenketten tokenbasiert auf Grundlage gleicher und ungleicher Token als Wert zwischen 0 und 1 über den Algorithmus zur Erkennung von Duplikaten nach Fellegi und Sunter.

Eigenschaften

| | |
|---------------------|--|
| Tokenizer | Tokenizer zum Aufsplitten in mehrere Token. Dabei wird der Ähnlichkeitswert jedes Tokens mit jedem anderen berechnet und der größte Wert jeweils summiert. Die Summe wird durch die höhere Anzahl an Tokens dividiert. |
| TokenizerTokenBased | Tokenizer zum Aufsplitten in mehrere Token zur Berechnung tokenbasierter Vergleichsalgorithmen (Default: GenericTokenizer). |
| ScoreMatch | Wert zwischen 0 und 1, der bestimmt, ab wann zwei Token als identisch angesehen werden (Default: 0,8). |
| Comparator | Zeichenkettenvergleichsalgorithmus zum Vergleich der einzelnen Token (Default: JaroApproximateMatching). |
| FrequencyTable | Verweis auf Tabelle im Repository mit dem Präfix „frq“ mit den relativen Häufigkeiten einzelner Tokens (kann mit der SQL-Prozedur <i>Tools.BuildFrequencyTable</i> erzeugt werden). |

Beispiel:

```
EXEC q.Tools.BuildFrequencyTable
    'FrequencySample_Customers',
    'Sample.Persons.Customers',
    'Name', null, 1, 1

SELECT a.CustomerId, b.CustomerId, a.Name, b.Name,
    q.Text.Match( a.Name, b.Name,
        'FreD.Text.Matching.FellegiSunterApproximateMatching',
        'Core',
        'FrequencyTable=FrequencySample_Customers,
        {Comparator=FreD.Text.Matching.JaroWinklerLynchApproximateMatching}' )
FROM Persons.Customers a, Persons.Customers b
WHERE q.Text.Match( a.Name, b.Name,
```

```
'FreD.Text.Matching.FellegiSunterApproximateMatching',  
'Core',  
'FrequencyTable=FrequencySample_Customers',  
{Comparator=FreD.Text.Matching.JaroWinklerLynchApproximateMatching}'))  
    >= 0 AND a.CustomerId > b.CustomerId AND a.Name <> b.Name  
ORDER BY 5
```

Namensraum FreD.Text.Matching
Assembly Core
Klasse JaccardApproximateMatching

Beschreibung Ermittelt die Ähnlichkeit zwischen zwei Zeichenketten tokenbasiert auf Grundlage gleicher und ungleicher Token als Wert zwischen 0 und 1 über über das Jaccard-Maß.

Eigenschaften

Tokenizer Tokenizer zum Aufsplitten in mehrere Token. Dabei wird der Ähnlichkeitswert jedes Tokens mit jedem anderen berechnet und der größte Wert jeweils summiert. Die Summe wird durch die höhere Anzahl an Tokens dividiert.

TokenizerTokenBased Tokenizer zum Aufsplitten in mehrere Token zur Berechnung tokenbasierter Vergleichsalgorithmen (Default: GenericTokenizer).

Beispiel:

```
SELECT a.CustomerId, b.CustomerId, a.Name, b.Name,
       q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.JaccardApproximateMatching',
                     'Core',
                     null )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.JaccardApproximateMatching',
                     'Core',
                     null ) > 0 AND
       a.CustomerId > b.CustomerId
ORDER BY 5
```

| | |
|----------------------|--|
| Namensraum | FreD.Text.Matching |
| Assembly | Core |
| Klasse | MatchingCoefficientApproximateMatching |
| Beschreibung | Ermittelt die Ähnlichkeit zwischen zwei Zeichenketten tokenbasiert auf Grundlage gleicher und ungleicher Token als Wert zwischen 0 und 1 über über das Matching Coefficient Maß. |
| Eigenschaften | |
| Tokenizer | Tokenizer zum Aufsplitten in mehrere Token. Dabei wird der Ähnlichkeitswert jedes Tokens mit jedem anderen berechnet und der größte Wert jeweils summiert. Die Summe wird durch die höhere Anzahl an Tokens dividiert. |
| TokenizerTokenBased | Tokenizer zum Aufsplitten in mehrere Token zur Berechnung tokenbasierter Vergleichsalgorithmen (Default: GenericTokenizer). |

Beispiel:

```

SELECT a.CustomerId, b.CustomerId, a.Name, b.Name,
       q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.MatchingCoefficientApproximateMatching',
                     'Core',
                     null )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.MatchingCoefficientApproximateMatching',
                     'Core',
                     null ) > 0 AND
       a.CustomerId > b.CustomerId
ORDER BY 5

```

Namensraum FreD.Text.Matching
Assembly Core
Klasse OverlapApproximateMatching

Beschreibung Ermittelt die Ähnlichkeit zwischen zwei Zeichenketten tokenbasiert auf Grundlage gleicher und ungleicher Token als Wert zwischen 0 und 1 über über das Overlap-Maß.

Eigenschaften

Tokenizer Tokenizer zum Aufsplitten in mehrere Token. Dabei wird der Ähnlichkeitswert jedes Tokens mit jedem anderen berechnet und der größte Wert jeweils summiert. Die Summe wird durch die höhere Anzahl an Tokens dividiert.

TokenizerTokenBased Tokenizer zum Aufsplitten in mehrere Token zur Berechnung tokenbasierter Vergleichsalgorithmen (Default: GenericTokenizer).

Beispiel:

```
SELECT a.CustomerId, b.CustomerId, a.Name, b.Name,
       q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.OverlapApproximateMatching',
                     'Core',
                     null )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.OverlapApproximateMatching',
                     'Core',
                     null ) > 0 AND
       a.CustomerId > b.CustomerId
ORDER BY 5
```

| | |
|---------------------|---|
| Namensraum | FreD.Text.Matching |
| Assembly | Core |
| Klasse | SoftTFIDFApproximateMatching |
| Beschreibung | Ermittelt die Ähnlichkeit zwischen zwei Zeichenketten tokenbasiert auf Grundlage gleicher und ungleicher Token als Wert zwischen 0 und 1 über den TF-IDF-Algorithmus. |

Eigenschaften

| | |
|---------------------|--|
| Tokenizer | Tokenizer zum Aufsplitten in mehrere Token. Dabei wird der Ähnlichkeitswert jedes Tokens mit jedem anderen berechnet und der größte Wert jeweils summiert. Die Summe wird durch die höhere Anzahl an Tokens dividiert. |
| TokenizerTokenBased | Tokenizer zum Aufsplitten in mehrere Token zur Berechnung tokenbasierter Vergleichsalgorithmen (Default: GenericTokenizer) |
| ScoreMatch | Wert zwischen 0 und 1, der bestimmt, ab wann zwei Token als identisch angesehen werden (Default: 0,8). |
| Comparator | Zeichenkettenvergleichsalgorithmus zum Vergleich der einzelnen Token (Default: JaroApproximateMatching). |
| FrequencyTable | Verweis auf Tabelle im Repository mit dem Präfix „frq“ mit den relativen Häufigkeiten einzelner Tokens (kann mit der SQL-Prozedur <i>Tools.BuildFrequencyTable</i> erzeugt werden). |

Beispiel:

```
EXEC q.Tools.BuildFrequencyTable
    'FrequencySample_Customers',
    'Sample.Persons.Customers',
    'Name', null, 1, 1

SELECT a.CustomerId, b.CustomerId, a.Name, b.Name,
       q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.SoftTFIDFApproximateMatching',
                     'Core',
                     'FrequencyTable=FrequencySample_Customers' )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.SoftTFIDFApproximateMatching',
                     'Core',
                     'FrequencyTable=FrequencySample_Customers' ) >= 0.8 AND
       a.CustomerId > b.CustomerId AND a.Name <> b.Name
```

ORDER BY 5

| | |
|----------------------|---|
| Namensraum | FreD.Text.Matching |
| Assembly | CompressApproximateMatching |
| Klasse | CompressApproximateMatching |
| Beschreibung | <p>Ermittelt die Ähnlichkeit zwischen zwei Zeichenketten zeichenbasiert als Wert zwischen 0 und 1, indem die Zeichenketten verglichen werden, nachdem diese einen Komprimierungsalgorithmus durchlaufen haben.</p> |
| Eigenschaften | |
| Tokenizer | <p>Tokenizer zum Aufsplitten in mehrere Token. Dabei wird der Ähnlichkeitswert jedes Tokens mit jedem anderen berechnet und der größte Wert jeweils summiert. Die Summe wird durch die höhere Anzahl an Tokens dividiert.</p> |

Beispiel:¹⁹

```
SELECT a.CustomerId, b.CustomerId, a.Name, b.Name,
       q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.CompressApproximateMatching',
                     'CompressApproximateMatching',
                     null )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.CompressApproximateMatching',
                     'CompressApproximateMatching',
                     null ) > 0.8 AND
       a.CustomerId > b.CustomerId
ORDER BY 5
```

¹⁹ Das SQL-Beispiel funktioniert nur mit dem Plugin „CompressApproximateMatching“

| | |
|---------------------|---|
| Namensraum | FreD.Text.Matching |
| Assembly | DistanceBasedApproximateMatching |
| Klasse | NeedlemanWunschApproximateMatching |
| Beschreibung | Ermittelt die Ähnlichkeit zwischen zwei Zeichenketten zeichenbasiert als Wert zwischen 0 und 1 nach dem Needleman- Wunsch-Distanzmaß. |

Eigenschaften

| | |
|-----------|---|
| Tokenizer | Tokenizer zum Aufsplitten in mehrere Token. Dabei wird der Ähnlichkeitswert jedes Tokens mit jedem anderen berechnet und der größte Wert jeweils summiert. Die Summe wird durch die höhere Anzahl an Tokens dividiert. |
|-----------|---|

Beispiel:²⁰

```
SELECT a.CustomerId, b.CustomerId, a.Name, b.Name,
       q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.NeedlemanWunschApproximateMatching',
                     'DistanceBasedApproximateMatching',
                     null )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.NeedlemanWunschApproximateMatching',
                     'DistanceBasedApproximateMatching',
                     null ) > 0.8 AND
       a.CustomerId > b.CustomerId
ORDER BY 5
```

²⁰ Das SQL-Beispiel funktioniert nur mit dem Plugin „DistanceBasedApproximateMatching“

| | |
|----------------------|---|
| Namensraum | FreD.Text.Matching |
| Assembly | DistanceBasedApproximateMatching |
| Klasse | SmithWatermanApproximateMatching |
| Beschreibung | Ermittelt die Ähnlichkeit zwischen zwei Zeichenketten zeichenbasiert als Wert zwischen 0 und 1 nach dem Smith- Waterman-Distanzmaß. |
| Eigenschaften | |
| Tokenizer | Tokenizer zum Aufsplitten in mehrere Token. Dabei wird der Ähnlichkeitswert jedes Tokens mit jedem anderen berechnet und der größte Wert jeweils summiert. Die Summe wird durch die höhere Anzahl an Tokens dividiert. |

Beispiel:²¹

```
SELECT a.CustomerId, b.CustomerId, a.Name, b.Name,
       q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.SmithWatermanApproximateMatching',
                     'DistanceBasedApproximateMatching',
                     null )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.SmithWatermanApproximateMatching',
                     'DistanceBasedApproximateMatching',
                     null ) > 0.8 AND
       a.CustomerId > b.CustomerId
ORDER BY 5
```

²¹ Das SQL-Beispiel funktioniert nur mit dem Plugin „DistanceBasedApproximateMatching“

B.2.3 Tokenizer

Namensraum FreD.Text.Tokenizer

Assembly Core

Klasse GenericTokenizer

Beschreibung Trennt eine Zeichenkette in einzelne Tokens auf, indem es eine Liste von Trennzeichen verwendet.

Eigenschaften

Separators Trennzeichen, die bestimmen, an welchen Stellen die Zeichenkette getrennt wird

(Default: „ !\"#\$%&'()*+,-./:;<=>?@[\\]^_`{|}~\$°\t\n“).

Beispiel:

```
SELECT Name,
       q.Text.Tokenize( Name, ' | ',
                       'FreD.Text.Tokenizer.GenericTokenizer',
                       'Core', null )
FROM   Persons.Customers
```

| | |
|-------------------|---------------------|
| Namensraum | FreD.Text.Tokenizer |
| Assembly | Core |
| Klasse | NGramTokenizer |

Beschreibung Trennt eine Zeichenkette in einzelne Tokens auf, indem die Zeichenkette in gleich lange Token der Länge N aufgeteilt werden.

Eigenschaften

| | |
|-------------------|--|
| NGramSize | Länge der Token (Default: 3). |
| NormalizeHeadTail | Berücksichtigt Anfang und Ende einer Zeichenkette (Default: true) |
| PreTokenizer | Tokenizer, der vor Erstellung der N-Gram's auf die Zeichenkette angewendet wird. |

Beispiel:

```
SELECT Name,
       q.Text.Tokenize( Name, ' | ',
                       'FreD.Text.Tokenizer.NGramTokenizer',
                       'Core',
                       'NGramSize=2,
                       NormalizeHeadTail=false,
                       {PreTokenizer=FreD.Text.Tokenizer.GenericTokenizer}' )
FROM   Persons.Customers
```

| | |
|-------------------|-------------------------|
| Namensraum | FreD.Text.Tokenizer |
| Assembly | Core |
| Klasse | NGramThresholdTokenizer |

Beschreibung Bildet aus N-Gram-Token der Länge N neue zusammengesetzte Token, deren Anzahl von einem Schwellwert abhängt.

Eigenschaften

| | |
|-------------------|--|
| Threshold | Schwellwert, der die Anzahl zusammengesetzte Token bestimmt (Default: 0,8). |
| NGramSize | Länge der Token (Default: 3). |
| NormalizeHeadTail | Berücksichtigt Anfang und Ende einer Zeichenkette (Default: true) |
| PreTokenizer | Tokenizer, der vor Erstellung der N-Gram's auf die Zeichenkette angewendet wird. |

Beispiel:

```
SELECT Name,
       q.Text.Tokenize( Name, ' | ',
                       'FreD.Text.Tokenizer.NGramThresholdTokenizer',
                       'Core',
                       'Threshold=0.9,
                       NGramSize=2,
                       {PreTokenizer=FreD.Text.Tokenizer.GenericTokenizer}' )
FROM   Persons.Customers
```

| | |
|-------------------|----------------------|
| Namensraum | FreD.Text.Tokenizer |
| Assembly | Core |
| Klasse | SuffixArrayTokenizer |

Beschreibung Trennt eine Zeichenkette in einzelne Tokens auf, indem die Suffixe einer Zeichenkette bis zu einer minimalen Länge von N gebildet werden.

Eigenschaften

| | |
|-----------------|--|
| MinSuffixLength | Bestimmt die minimale Länge eines Token (Default: 3) |
|-----------------|--|

Beispiel:

```
SELECT Name,
       q.Text.Tokenize( Name, ' | ',
                       'FreD.Text.Tokenizer.SuffixArrayTokenizer',
                       'Core',
                       'MinSuffixLength=10' )
FROM   Persons.Customers
```

B.3 Verstehen

B.3.1 Daten- und Schemaeigenschaften

Namensraum FreD.Understand.Properties

Assembly Core

Klasse CountNotNullProperty

Beschreibung Ermittelt die Anzahl an NOT NULL Einträgen.

Eigenschaften

keine

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(  
    '{FreD.Understand.Properties.CountNotNullProperty|Core}'+  
    COALESCE( Color, '{NULL}') )  
FROM Sales.Articles
```

qSQL

| | |
|----------------------|---|
| Namensraum | FreD.Understand.Properties |
| Assembly | Core |
| Klasse | CountNullProperty |
| Beschreibung | Ermittelt die Anzahl an NULL Einträgen. |
| Eigenschaften | keine |

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
    '{FreD.Understand.Properties.CountNullProperty|Core}'+
    COALESCE( Color, '{NULL}') )
FROM Sales.Articles
```


qSQL

| | |
|----------------------|--|
| Namensraum | FreD.Understand.Properties |
| Assembly | Core |
| Klasse | DataTypeRecognisedProperty |
| Beschreibung | Ermittelt den Datentyp einer Tabellenspalte anhand der Dateninhalte. |
| Eigenschaften | |
| Threshold | Ausgabe als SQL-Datentyp (Default: true), ansonsten als .NET-Datentyp. |

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
    '{FreD.Understand.Properties.DataTypeRecognisedProperty|
    Core|Translate=true}' +
    COALESCE( ProductNumber, '{NULL}') )
FROM Sales.Articles
```

qSQL

| | |
|-------------------|----------------------------|
| Namensraum | FreD.Understand.Properties |
| Assembly | Core |
| Klasse | MaxProperty |

Beschreibung Ermittelt die n größten Werte einer Spalte.

Eigenschaften

MaxCount Anzahl der maximalen Werte, die zurückgegeben werden (Default: 5).

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
    '{FreD.Understand.Properties.MaxProperty|Core|MaxCount=10}' +
    COALESCE( ProductNumber, '{NULL}') )
FROM Sales.Articles
```

qSQL

Namensraum FreD.Understand.Properties

Assembly Core

Klasse MinProperty

Beschreibung Ermittelt die n kleinsten Werte einer Spalte.

Eigenschaften

MaxCount Anzahl der minimalen Werte, die zurückgegeben werden
(Default: 5).

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
    '{FreD.Understand.Properties.MinProperty|Core|MinCount=10}'+
    COALESCE( ProductNumber, '{NULL}') )
FROM Sales.Articles
```

qSQL

| | |
|-------------------|-----------------------------|
| Namensraum | FreD.Understand.Properties |
| Assembly | Core |
| Klasse | PrimaryKeyCandidateProperty |

Beschreibung Ermittelt, ob eine Spalte als Primärschlüssel geeignet ist.

Eigenschaften

Threshold Schwellwert eindeutiger Werte, ab dem eine Spalte als Primärschlüssel identifiziert wird (Default: 0.95).

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
    '{FreD.Understand.Properties.PrimaryKeyCandidateProperty|
    Core|Threshold=0.8}' +
    COALESCE( Name, '{NULL}' ) )
FROM Persons.Customers
```

qSQL

| | |
|----------------------|---|
| Namensraum | FreD.Understand.Properties |
| Assembly | Core |
| Klasse | UniqueCountProperty |
| Beschreibung | Gibt die Anzahl eindeutiger Werte zurück. |
| Eigenschaften | keine |

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
    '{FreD.Understand.Properties.UniqueCountProperty|Core}'+
    COALESCE( Name, '{NULL}') )
FROM Persons.Customers
```

qSQL

| | |
|----------------------|---|
| Namensraum | FreD.Understand.Properties |
| Assembly | Core |
| Klasse | UniqueCountProperty |
| Beschreibung | Gibt das relative Verhältnis eindeutiger Werte zur Gesamtanzahl zurück. |
| Eigenschaften | keine |

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
    '{FreD.Understand.Properties.UniquenessProperty|Core}'+
    COALESCE( Name, '{NULL}') )
FROM Persons.Customers
```

qSQL

| | |
|----------------------|--|
| Namensraum | FreD.Understand.Properties.Text |
| Assembly | Core |
| Klasse | BlankCountProperty |
| Beschreibung | Gibt die Anzahl an Werten, die leer (nicht NULL) sind. |
| Eigenschaften | keine |

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
    '{FreD.Understand.Properties.Text.BlankCountProperty|Core}'+
    COALESCE( FirstName, '{NULL}') )
FROM Persons.Customers
```

qSQL

| | |
|----------------------|---|
| Namensraum | FreD.Understand.Properties.Text |
| Assembly | Core |
| Klasse | MaxTextCountProperty |
| Beschreibung | Anzahl der Texte mit der größten Länge. |
| Eigenschaften | keine |

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
    '{FreD.Understand.Properties.Text.MaxTextCountProperty|Core}' +
    COALESCE( Name, '{NULL}') )
FROM Persons.Customers
```


qSQL

| | |
|-------------------|---------------------------------|
| Namensraum | FreD.Understand.Properties.Text |
| Assembly | Core |
| Klasse | MaxTextLengthProperty |

Beschreibung Größe des längsten Textes.

Eigenschaften

keine

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
    '{FreD.Understand.Properties.Text.MaxTextLengthProperty|Core}'+
    COALESCE( Name, '{NULL}') )
FROM Persons.Customers
```

qSQL

| | |
|----------------------|---|
| Namensraum | FreD.Understand.Properties.Text |
| Assembly | Core |
| Klasse | MinTextCountProperty |
| Beschreibung | Anzahl der Texte mit der kleinsten Länge. |
| Eigenschaften | keine |

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
    '{FreD.Understand.Properties.Text.MinTextCountProperty|Core}' +
    COALESCE( Name, '{NULL}') )
FROM Persons.Customers
```

qSQL

| | |
|-------------------|---------------------------------|
| Namensraum | FreD.Understand.Properties.Text |
| Assembly | Core |
| Klasse | MinTextLengthProperty |

Beschreibung Größe des kürzesten Textes.

Eigenschaften

keine

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
    '{FreD.Understand.Properties.Text.MinTextLengthProperty|Core}' +
    COALESCE( Name, '{NULL}' ) )
FROM Persons.Customers
```

qSQL

| | |
|----------------------|--|
| Namensraum | FreD.Understand.Properties.Number |
| Assembly | Core |
| Klasse | BenfordsLawProperty |
| Beschreibung | Gibt die Verteilung der einzelnen Ziffern nach Benford's Law und in der übergebenen Spalte aus. |
| Eigenschaften | |
| TrimmedCount | Anzahl der maximalen und minimalen Werte, die bei der Berechnung nicht berücksichtigt werden sollen. |

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
    '{FreD.Understand.Properties.Number.BenfordsLawProperty|
    Core|TrimmedCount=5}' +
    COALESCE( CAST( ListPrice AS NVARCHAR ), '{NULL}' ) )
FROM Sales.Articles
```

qSQL

| | |
|-------------------|-----------------------------------|
| Namensraum | FreD.Understand.Properties.Number |
| Assembly | Core |
| Klasse | GeometricMeanProperty |

Beschreibung Gibt den geometrischen Mittelwert zurück.

Eigenschaften

TrimmedCount Anzahl der maximalen und minimalen Werte, die bei der Berechnung nicht berücksichtigt werden sollen.

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
    '{FreD.Understand.Properties.Number.GeometricMeanProperty|Core}'+
    COALESCE( CAST( ListPrice AS NVARCHAR ), '{NULL}' ) )
FROM Sales.Articles
WHERE Listprice < 100
```

qSQL

| | |
|-------------------|-----------------------------------|
| Namensraum | FreD.Understand.Properties.Number |
| Assembly | Core |
| Klasse | HarmonicMeanProperty |

Beschreibung Gibt den harmonischen Mittelwert zurück.

Eigenschaften

TrimmedCount Anzahl der maximalen und minimalen Werte, die bei der Berechnung nicht berücksichtigt werden sollen.

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
    '{FreD.Understand.Properties.Number.HarmonicMeanProperty|Core}'+
    COALESCE( CAST( ListPrice AS NVARCHAR ), '{NULL}' ) )
FROM Sales.Articles
```

| | |
|-------------------|-----------------------------------|
| Namensraum | FreD.Understand.Properties.Number |
| Assembly | Core |
| Klasse | MeanProperty |

Beschreibung Gibt den arithmetischen Mittelwert zurück.

Eigenschaften

| | |
|--------------|--|
| TrimmedCount | Anzahl der maximalen und minimalen Werte, die bei der Berechnung nicht berücksichtigt werden sollen. |
|--------------|--|

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
    '{FreD.Understand.Properties.Number.MeanProperty|Core}'+
    COALESCE( CAST( ListPrice AS NVARCHAR ), '{NULL}') )
FROM Sales.Articles
```

Namensraum FreD.Understand.Properties.Number
Assembly Core
Klasse ProductProperty

Beschreibung Gibt das Produkt aus allen Zahlen zurück.

Eigenschaften

TrimmedCount Anzahl der maximalen und minimalen Werte, die bei der Berechnung nicht berücksichtigt werden sollen.

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
    '{FreD.Understand.Properties.Number.ProductProperty|Core}'+
    COALESCE( CAST( ListPrice AS NVARCHAR ), '{NULL}' ) )
FROM Sales.Articles
WHERE ListPrice < 10
```


qSQL

| | |
|-------------------|-----------------------------------|
| Namensraum | FreD.Understand.Properties.Number |
| Assembly | Core |
| Klasse | RangeProperty |

Beschreibung Gibt den Wertebereich aus allen Zahlen zurück.

Eigenschaften

TrimmedCount Anzahl der maximalen und minimalen Werte, die bei der Berechnung nicht berücksichtigt werden sollen.

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
    '{FreD.Understand.Properties.Number.RangeProperty|Core}'+
    COALESCE( CAST( StockReorderPoint AS NVARCHAR ), '{NULL}' ) )
FROM Sales.Articles
WHERE ListPrice < 10
```

qSQL

| | |
|-------------------|-----------------------------------|
| Namensraum | FreD.Understand.Properties.Number |
| Assembly | Core |
| Klasse | SquareSumProperty |

Beschreibung Gibt die Summe der Quadrate zurück.

Eigenschaften

TrimmedCount Anzahl der maximalen und minimalen Werte, die bei der Berechnung nicht berücksichtigt werden sollen.

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
    '{FreD.Understand.Properties.Number.SquareSumProperty|Core}'+
    COALESCE( CAST( StockReorderPoint AS NVARCHAR ), '{NULL}' ) )
FROM Sales.Articles
```

| | |
|-------------------|-----------------------------------|
| Namensraum | FreD.Understand.Properties.Number |
| Assembly | Core |
| Klasse | StdDevProperty |

Beschreibung Gibt die Standardabweichung zurück.

Eigenschaften

| | |
|--------------|--|
| TrimmedCount | Anzahl der maximalen und minimalen Werte, die bei der Berechnung nicht berücksichtigt werden sollen. |
|--------------|--|

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
    '{FreD.Understand.Properties.Number.StdDevProperty|Core}'+
    COALESCE( CAST( ListPrice AS NVARCHAR ), '{NULL}') )
FROM Sales.Articles
```

qSQL

| | |
|-------------------|-----------------------------------|
| Namensraum | FreD.Understand.Properties.Number |
| Assembly | Core |
| Klasse | StdErrProperty |

Beschreibung Gibt den Standardfehler zurück.

Eigenschaften

TrimmedCount Anzahl der maximalen und minimalen Werte, die bei der Berechnung nicht berücksichtigt werden sollen.

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
    '{FreD.Understand.Properties.Number.StdErrProperty|Core}'+
    COALESCE( CAST( ListPrice AS NVARCHAR ), '{NULL}') )
FROM Sales.Articles
```

| | |
|-------------------|-----------------------------------|
| Namensraum | FreD.Understand.Properties.Number |
| Assembly | Core |
| Klasse | SumProperty |

Beschreibung Gibt die Summe zurück.

Eigenschaften

| | |
|--------------|--|
| TrimmedCount | Anzahl der maximalen und minimalen Werte, die bei der Berechnung nicht berücksichtigt werden sollen. |
|--------------|--|

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
    '{FreD.Understand.Properties.Number.SumProperty|Core}'+
    COALESCE( CAST( ListPrice AS NVARCHAR ), '{NULL}') )
FROM Sales.Articles
```

qSQL

| | |
|-------------------|-----------------------------------|
| Namensraum | FreD.Understand.Properties.Number |
| Assembly | Core |
| Klasse | VarianceProperty |

Beschreibung Gibt die Varianz zurück.

Eigenschaften

TrimmedCount Anzahl der maximalen und minimalen Werte, die bei der Berechnung nicht berücksichtigt werden sollen.

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
    '{FreD.Understand.Properties.Number.VarianceProperty|Core}'+
    COALESCE( CAST( ListPrice AS NVARCHAR ), '{NULL}') )
FROM Sales.Articles
```

qSQL

| | |
|----------------------|---------------------------------------|
| Namensraum | FreD.Understand.Properties.Number |
| Assembly | Core |
| Klasse | IntegerCountProperty |
| Beschreibung | Gibt die Anzahl an Ganzzahlen zurück. |
| Eigenschaften | keine |

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
    '{FreD.Understand.Properties.Number.IntegerCountProperty|Core}' +
    COALESCE( PostalCode, '{NULL}') )
FROM Persons.Customers
```

qSQL

| | |
|----------------------|--|
| Namensraum | FreD.Understand.Properties.Number |
| Assembly | Core |
| Klasse | MaxDecimalPlacesProperty |
| Beschreibung | Gibt die größte Anzahl an Nachkommastellen zurück. |
| Eigenschaften | keine |

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
    '{FreD.Understand.Properties.Number.MaxDecimalPlacesProperty|
    Core}'+
    COALESCE( CAST( ListPrice AS NVARCHAR ), '{NULL}' ) )
FROM Sales.Articles
```


qSQL

| | |
|----------------------|--|
| Namensraum | FreD.Understand.Properties.Number |
| Assembly | Core |
| Klasse | MinDecimalPlacesProperty |
| Beschreibung | Gibt die kleinste Anzahl an Nachkommastellen zurück. |
| Eigenschaften | keine |

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
    '{FreD.Understand.Properties.Number.MinDecimalPlacesProperty|
    Core}' +
    COALESCE( CAST( ListPrice AS NVARCHAR ), '{NULL}' ) )
FROM Sales.Articles
```

qSQL

Namensraum FreD.Understand.Properties
Assembly Core
Klasse DataPropertyListGeneric

Beschreibung Ermittelt für Spalten einer Tabelle alle Daten-Eigenschaften aus Klassen einer Assembly, die *IDataProperty* implementieren.

Eigenschaften

keine

Beispiel:

```
SELECT *
FROM q.Understand.PropertyListData (
    'Sample.Sales.Articles',
    'Name, ListPrice, Color',
    null,
    'FreD.Understand.Properties.DataPropertyListGeneric',
    'Core', null )
ORDER BY TypeName
```

qSQL

| | |
|----------------------|---|
| Namensraum | FreD.Understand.Properties |
| Assembly | Core |
| Klasse | SchemaPropertyListGeneric |
| Beschreibung | Ermittelt für Spalten einer Tabelle alle Schema-Eigenschaften über ADO.NET. |
| Eigenschaften | keine |

Beispiel:

```
SELECT *
FROM q.Understand.PropertyListSchema (
    'Sample.Sales.Articles',
    'ArticleId, Name, Stock, ListPrice',
    'FreD.Understand.Properties.SchemaPropertyListGeneric',
    'Core', null )
ORDER BY Name
```

B.3.2 Primär- und Fremdschlüssel

Namensraum FreD.Understand.Dependencies
Assembly Core
Klasse PrimaryKeyAnalyse

Beschreibung Ermittelt mögliche Primärschlüssel (max. zwei Attribute).

Eigenschaften

Threshold Schwellwert, der angibt, wie groß der Anteil eindeutiger Werte für ein Primärschlüssel sein muss (Default: 0,8).

Beispiel:

```
SELECT *
FROM q.Understand.PrimaryKey(
    'Sample.Sales.Articles',
    '*',
    null,
    'FreD.Understand.Dependencies.PrimaryKeyAnalyse',
    'Core',
    'Threshold=0.5' )
ORDER BY Ratio DESC, ColumnsCount ASC
```

| Columns | Ratio |
|-----------------------------|-------------------|
| ArticleId | 1 |
| ProductNumber | 1 |
| Name, Color | 1 |
| Name, Stock | 1 |
| Name, StockReorderPoint | 1 |
| Name, Size | 1 |
| Name, SizeUnitMeasureCode | 1 |
| Name, WeightUnitMeasureCode | 1 |
| Name, Weight | 1 |
| Name, Description | 1 |
| Color, Weight | 0,748148148148148 |
| Stock, Weight | 0,5 |
| Size, Weight | 0,507407407407407 |

qSQL

Namensraum FreD.Understand.Dependencies
Assembly Core
Klasse ForeignKeyAnalyse

Beschreibung Ermittelt mögliche Primär-/Fremdschlüsselbeziehungen zwischen zwei Tabellen, indem die Übereinstimmung von Werten zwischen zwei Spalten überprüft wird.

Eigenschaften

Threshold Schwellwert, der angibt, wie groß der Anteil übereinstimmender Werte für ein Primär-/Fremdschlüsselbeziehung sein muss (Default: 0,0), siehe Spalte „Ratio“.

Beispiel:

```
SELECT *
FROM q.Understand.ForeignKey(
    'Sample.Sales.Articles', '*', null,
    'Sample.Sales.Orders', '*', null,
    'FreD.Understand.Dependencies.ForeignKeyAnalyse',
    'Core',
    'Threshold=0' )
```

| Relation | | Shared | Left | Right | Ratio |
|---|------|--------|------|-------|-------|
| Sample.Sales.Articles.ArticleId | <==> | 16 | 254 | 0 | 0,059 |
| Sample.Sales.Orders.ArticleNo | | | | | |
| Sample.Sales.Articles.ArticleId | <==> | 20 | 250 | 2 | 0,073 |
| Sample.Sales.Orders.CustomerNo | | | | | |
| Sample.Sales.Articles.Stock | <==> | 0 | 270 | 25 | 0 |
| Sample.Sales.Orders.ArticleNo | | | | | |
| Sample.Sales.Articles.Stock | <==> | 98 | 172 | 23 | 0,33 |
| Sample.Sales.Orders.CustomerNo | | | | | |
| Sample.Sales.Articles.StockReorderPoint | <==> | 0 | 270 | 25 | 0 |
| Sample.Sales.Orders.ArticleNo | | | | | |
| Sample.Sales.Articles.StockReorderPoint | <==> | 0 | 270 | 25 | 0 |
| Sample.Sales.Orders.CustomerNo | | | | | |

Namensraum FreD.Understand.Dependencies
Assembly Core
Klasse ForeignKeyTextMatching

Beschreibung Ermittelt mögliche Primär-/Fremdschlüsselbeziehungen zwischen zwei Tabellen, indem die Übereinstimmung anhand der Spaltennamen über ein Vergleichsalgorithmus überprüft wird.

Eigenschaften

Threshold Schwellwert, der angibt, wie groß die Übereinstimmung der Spaltennamen sein muss (Default: 0,9).
Match Textvergleichsverfahren zum Vergleichen der Spaltennamen (Default: JaroWinklerLynchApproximateMatching).

Beispiel:

```
SELECT *
FROM q.Understand.ForeignKey(
    'Sample.Sales.Articles', '*', null,
    'Sample.Sales.Orders', '*', null,
    'FreD.Understand.Dependencies.ForeignKeyTextMatching',
    'Core',
    'Threshold=0.8,
    {Match=FreD.Text.Matching.JaroApproximateMatching}' )
```

| Relation | SharedValue | LeftValue | RightValue | Ratio |
|--|-------------|-----------|------------|-------|
| Sample.Sales.Articles.ArticleId <==> Sample.Sales.Orders.ArticleNo | NULL | NULL | NULL | 0,85 |

B.3.3 Regelinduktion

Namensraum FreD.Understand.Rules

Assembly Core

Klasse OperatorRuleInduction

Beschreibung Sucht nach Größer-, Kleiner-, Gleich-Regeln in den übergebenen Spalten.

Eigenschaften

MinSupport Minimaler Support der von einer Regel erfüllt sein muss (Default: 0,8).

MinFrequency Minimale Anzahl an Datensätzen, für die die Regel gelten muss (Default: 0).

Beispiel:

```
SELECT "Rule", Support, Frequency
FROM   q.Understand.RuleInduction(
        'Sample.Sales.Orders',
        '*',
        null,
        'FreD.Understand.Rules.OperatorRuleInduction',
        'Core',
        'MinSupport=0.9, MinFrequency=10')
```

| Rule | Support | Frequency |
|--------------------------|---------|-----------|
| [OrderDate] < [ShipDate] | 0,947 | 18 |

Namensraum FreD.Understand.Rules
Assembly WekaRuleInduction
Klasse AprioriRuleInduction

Beschreibung Sucht über den Apriori-Algorithmus nach Regeln in den übergebenen Spalten.

Eigenschaften

MinSupportLowerBound Unterer Wert für den minimalen Support (Default: 0,1).
MaxSupportLowerBound Oberer Wert für den minimalen Support (Default: 1,0).
MinItemFrequency Minimale Anzahl an Datensätzen, für die die Regel gelten muss (Default: 0).
Delta Wert, um den der Support iterativ verringert wird (Default: 0,05).
NumRules Maximale Anzahl an generierten Regeln (Default: 10).
AutoPreProcess Automatisches Diskretisieren numerischer Attribute (Default: true).
WekaOptions Weka-Optionen, haben Vorrang vor den anderen Eigenschaften.

Beispiel:²²

```
SELECT "Rule", Confidence, Support, ExpectedConfidence,
      Frequency, FrequencyCondition, FrequencyResult
FROM   q.Understand.RuleInduction(
        'Sample.Persons.Customers',
        'Age, MaritalStatus, State, Sex',
        null,
        'FreD.Understand.Rules.AprioriRuleInduction',
        'WekaRuleInduction',
        'AutoPreProcess=false, MinSupportLowerBound=0.01,
        NumRules=100, MinItemFrequency=10, Delta=0.1')
```

| Rule | Confidence | Support | Expected Confidence | Frequency | Frequency Condition | Frequency Result |
|------|------------|---------|---------------------|-----------|---------------------|------------------|
|------|------------|---------|---------------------|-----------|---------------------|------------------|

²² Das SQL-Beispiel funktioniert nur mit dem Plugin „WekaRuleInduction“

| | | | | | | |
|--|------|------|------|----|----|----|
| IF [MaritalStatus] = 'widowed' THEN [Age] = 'senior' | 1 | 0,17 | 1 | 18 | 18 | 18 |
| IF [Age] = 'young' AND [State] = 'AR' THEN [Sex] = 'female' | 1 | 0,11 | 1 | 12 | 12 | 12 |
| IF [Age] = 'middle-aged' AND [Sex] = 'male' THEN [MaritalStatus] = 'married' | 1 | 0,11 | 1 | 12 | 12 | 12 |
| IF [MaritalStatus] = 'widowed' AND [Sex] = 'male' THEN [Age] = 'senior' | 1 | 0,10 | 1 | 10 | 10 | 10 |
| IF [Age] = 'middle-aged' THEN [MaritalStatus] = 'married' | 0,93 | 0,26 | 0,93 | 27 | 29 | 27 |
| IF [MaritalStatus] = 'single' THEN [Age] = 'young' | 0,91 | 0,20 | 0,91 | 21 | 23 | 21 |

Namensraum FreD.Understand.Rules
Assembly WekaRuleInduction
Klasse TertiusRuleInduction

Beschreibung Sucht über den Tertius-Algorithmus nach Regeln in den übergebenen Spalten.

Eigenschaften

AutoPreProcess Automatisches Diskretisieren numerischer Attribute (Default: true).
WekaOptions Weka-Optionen, haben Vorrang vor den anderen Eigenschaften.

Beispiel:²³

```
SELECT "Rule", Indicator
FROM   q.Understand.RuleInduction(
        'Sample.Persons.Customers',
        'Age, MaritalStatus, State, Sex, City, State',
        null,
        'FreD.Understand.Rules.TertiusRuleInduction',
        'WekaRuleInduction',
        'AutoPreProcess=false')
```

| Rule | Indicator |
|---|-----------|
| IF [Age] = 'young' THEN [MaritalStatus] = 'single' OR [City] = 'Hot Springs National Park' OR [State] = 'PR' | 0,706 |
| IF [Age] = 'young' THEN [MaritalStatus] = 'single' OR [State] = 'PR' OR [City] = 'Hot Springs National Park' | 0,706 |
| IF [MaritalStatus] = 'married' THEN [Age] = 'middle-aged' OR [State] = 'NJ' OR [City] = 'Culver' | 0,702 |
| IF [MaritalStatus] = 'married' THEN [Age] = 'middle-aged' OR [State] = 'NJ' OR [City] = 'Greenleaf' | 0,702 |

²³ Das SQL-Beispiel funktioniert nur mit dem Plugin „WekaRuleInduction“

B.4 Verbessern

B.4.1 Daten standardisieren, korrigieren, anreichern

| | |
|-------------------|-------------------------|
| Namensraum | FreD.Improve.Prediction |
| Assembly | Core |
| Klasse | LookupPredictor |

Beschreibung Ermittelt auf Basis von Eingabeattributen einer Tabelle den Wert für ein Ausgabeattribut über Nachschlagetabellen.

Eigenschaften

ResolveMultipleValues Liefert den ersten gefundenen Wert zurück (First), den letzten (Last) oder erzeugt eine Ausnahme, wenn mehr als ein Wert in der Nachschlagetabelle gefunden wurde (Default: Error).

Beispiel:

```
EXEC q.Improve.PredictorBuild
    'q.Lookup.[City.DE]',
    'Plz',
    'Ort',
    null,
    'Predict_CityDE',
    'FreD.Improve.Prediction.LookupPredictor',
    'Core',
    'ResolveMultipleValues=First'

SELECT PostalCode, City,
       q.Improve.PredictorPredict(
           'Predict_CityDE',
           q.Tools.Param( PostalCode, 1 ) )
FROM   Persons.Customers
WHERE  City <> q.Improve.PredictorPredict(
           'Predict_CityDE',
           q.Tools.Param( PostalCode, 1 ) )
```

Predict 'City' from table 'q.Lookup.[City.US]' by input fields 'PostalCode'
 SQL: SELECT City FROM q.Lookup.[City.US] WHERE PostalCode = @par1
 ResolveMultipleValues = First

qSQL

| | |
|-------------------|-------------------------|
| Namensraum | FreD.Improve.Prediction |
| Assembly | Core |
| Klasse | ZeroRPredictor |

Beschreibung Ermittelt auf Basis von Eingabeattributen einer Tabelle den Wert für ein Ausgabeattribut über den OR-Algorithmus (für numerische wird der errechnete Mittelwert, für nominale Ausgabespalten der Modus als Vorhersagewert verwendet).

Eigenschaften

keine

Beispiel:

```
EXEC q.Improve.PredictorBuild
    'Sample.Persons.Customers',
    'Sex',
    'Sex',
    null,
    'Predict_Sex',
    'FreD.Improve.Prediction.ZeroRPredictor',
    'Core',
    null

SELECT Name, Sex,
       q.Improve.PredictorPredict(
           'Predict_Sex',
           q.Tools.Param( Sex, 1 ) )
FROM   Persons.Customers
```

ZeroR predicts class value: 'female' (Modal)

| | |
|-------------------|-------------------------|
| Namensraum | FreD.Improve.Prediction |
| Assembly | WekaPrediction |
| Klasse | GenericPredictor |

Beschreibung Ermittelt auf Basis von Eingabeattributen einer Tabelle den Wert für ein Ausgabeattribut über einen beliebigen Klassifizierungs-Algorithmus, der in der Assembly Weka implementiert ist.

Eigenschaften

| | |
|---------------------------|--|
| ClassifierName | Vollständiger Java-Klassenname in der Weka-Bibliothek |
| WekaOptions | Weka-Optionen, haben Vorrang vor den anderen Eigenschaften. |
| CrossValidationEvaluation | Führt eine Kreuzvalidierung durch (Default: false). |
| ConfusionMatrix | Gibt eine Konfusionsmatrix aus (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true. |
| ClassDetails | Detaillierte Ausgabe für jede Klasse (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true. |

Beispiel:²⁴

```
EXEC q.Improve.PredictorBuild
    'Sample.Persons.Customers',
    'Sex, MaritalStatus',
    'YearOfBirthday',
    null,
    'Predict_YearOfBirthday',
    'FreD.Improve.Prediction.GenericPredictor',
    'WekaPrediction',
    'ClassifierName=weka.classifiers.trees.Id3'

SELECT Sex, MaritalStatus, YearOfBirthday,
    q.Improve.PredictorPredict(
        'Predict_YearOfBirthday',
        q.Tools.Param( Sex, 0 ) + q.Tools.Param( MaritalStatus, 1 ) )
FROM Persons.Customers
```

Id3

MaritalStatus = single

²⁴ Das SQL-Beispiel funktioniert nur mit dem Plugin „WekaRuleInduction“

```
| Sex = female: young  
| Sex = male: young  
MaritalStatus = married  
| Sex = female: senior  
| Sex = male: middle-aged  
MaritalStatus = divorced  
| Sex = female: young  
| Sex = male: senior  
MaritalStatus = widowed: senior
```

| | |
|-------------------|-------------------------|
| Namensraum | FreD.Improve.Prediction |
| Assembly | WekaPrediction |
| Klasse | Id3Predictor |

Beschreibung Ermittelt auf Basis von Eingabeattributen einer Tabelle den Wert für ein Ausgabeattribut über den ID3-Algorithmus.

Eigenschaften

| | |
|---------------------------|--|
| ClassifierName | Vollständiger Java-Klassenname in der Weka-Bibliothek (Default: weka.classifiers.trees.Id3). |
| WekaOptions | Weka-Optionen, haben Vorrang vor den anderen Eigenschaften. |
| CrossValidationEvaluation | Führt eine Kreuzvalidierung durch (Default: false). |
| ConfusionMatrix | Gibt eine Konfusionsmatrix aus (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true. |
| ClassDetails | Detaillierte Ausgabe für jede Klasse (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true. |

Beispiel:²⁵

```
EXEC q.Improve.PredictorBuild
    'Sample.Persons.Customers',
    'Sex, MaritalStatus',
    'YearOfBirthday',
    null,
    'Predict_YearOfBirthday',
    'FreD.Improve.Prediction.Id3Predictor',
    'WekaPrediction',
    'CrossValidationEvaluation=true'

SELECT Sex, MaritalStatus, Age,
    q.Improve.PredictorPredict(
        'Predict_YearOfBirthday',
        q.Tools.Param( Sex, 0 ) + q.Tools.Param( MaritalStatus, 1 ) )
FROM Persons.Customers
```

Id3

MaritalStatus = single

²⁵ Das SQL-Beispiel funktioniert nur mit dem Plugin „WekaRuleInduction“

```
| Sex = female: young
| Sex = male: young
MaritalStatus = married
| Sex = female: senior
| Sex = male: middle-aged
MaritalStatus = divorced
| Sex = female: young
| Sex = male: senior
MaritalStatus = widowed: senior
```

| | | |
|----------------------------------|-----------|-----------|
| Correctly Classified Instances | 71 | 69.6078 % |
| Incorrectly Classified Instances | 31 | 30.3922 % |
| Kappa statistic | 0.5181 | |
| Mean absolute error | 0.2038 | |
| Root mean squared error | 0.3274 | |
| Relative absolute error | 47.2603 % | |
| Root relative squared error | 70.5133 % | |
| Total Number of Instances | 102 | |

=== Detailed Accuracy By Class ===

| TP Rate | FP Rate | Precision | Recall | F-Measure | Class |
|---------|---------|-----------|--------|-----------|-------------|
| 0.483 | 0.11 | 0.636 | 0.483 | 0.549 | middle-aged |
| 0.778 | 0.053 | 0.84 | 0.778 | 0.808 | young |
| 0.783 | 0.339 | 0.655 | 0.783 | 0.713 | senior |

=== Confusion Matrix ===

```
a  b  c  <-- classified as
14  2 13 | a = middle-aged
0 21  6 | b = young
8  2 36 | c = senior
```


| | |
|-------------------|-------------------------|
| Namensraum | FreD.Improve.Prediction |
| Assembly | WekaPrediction |
| Klasse | J48Predictor |

Beschreibung Ermittelt auf Basis von Eingabeattributen einer Tabelle den Wert für ein Ausgabeattribut über den J48-Algorithmus.

Eigenschaften

| | |
|---------------------------|--|
| ClassifierName | Vollständiger Java-Klassenname in der Weka-Bibliothek (Default: weka.classifiers.trees.J48). |
| WekaOptions | Weka-Optionen, haben Vorrang vor den anderen Eigenschaften. |
| CrossValidationEvaluation | Führt eine Kreuzvalidierung durch (Default: false). |
| ConfusionMatrix | Gibt eine Konfusionsmatrix aus (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true. |
| ClassDetails | Detaillierte Ausgabe für jede Klasse (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true. |

Beispiel:²⁶

```
EXEC q.Improve.PredictorBuild
    'Sample.Persons.Customers',
    'Sex, MaritalStatus',
    'YearOfBirthday',
    null,
    'Predict_YearOfBirthday',
    'FreD.Improve.Prediction.J48Predictor',
    'WekaPrediction',
    null

SELECT Sex, MaritalStatus, YearOfBirthday,
    q.Improve.PredictorPredict(
        'Predict_YearOfBirthday',
        q.Tools.Param( Sex, 0 ) + q.Tools.Param( MaritalStatus, 1 ) )
FROM Persons.Customers
```

J48 pruned tree

MaritalStatus = single: young (23.0/2.0)

²⁶ Das SQL-Beispiel funktioniert nur mit dem Plugin „WekaRuleInduction“

```
MaritalStatus = married
|   Sex = female: senior (35.0/18.0)
|   Sex = male: middle-aged (14.0/2.0)
MaritalStatus = divorced: senior (12.0/3.0)
MaritalStatus = widowed: senior (18.0)
```

Number of Leaves : 5

Size of the tree : 7

| | | |
|----------------------------------|-----------|-----------|
| Correctly Classified Instances | 68 | 66.6667 % |
| Incorrectly Classified Instances | 34 | 33.3333 % |
| Kappa statistic | 0.4874 | |
| Mean absolute error | 0.2304 | |
| Root mean squared error | 0.3566 | |
| Relative absolute error | 53.4365 % | |
| Root relative squared error | 76.8017 % | |
| Total Number of Instances | 102 | |

| | |
|-------------------|-------------------------|
| Namensraum | FreD.Improve.Prediction |
| Assembly | WekaPrediction |
| Klasse | LogisticPredictor |

Beschreibung Ermittelt auf Basis von Eingabeattributen einer Tabelle den Wert für ein Ausgabeattribut über ein multinomiales logistisches Regressionsmodell.

Eigenschaften

| | |
|---------------------------|--|
| ClassifierName | Vollständiger Java-Klassenname in der Weka-Bibliothek (Default: weka.classifiers.functions.Logistic). |
| WekaOptions | Weka-Optionen, haben Vorrang vor den anderen Eigenschaften. |
| CrossValidationEvaluation | Führt eine Kreuzvalidierung durch (Default: false). |
| ConfusionMatrix | Gibt eine Konfusionsmatrix aus (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true. |
| ClassDetails | Detaillierte Ausgabe für jede Klasse (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true. |

Beispiel:²⁷

```
EXEC q.Improve.PredictorBuild
    'Sample.Persons.Customers',
    'Sex, MaritalStatus',
    'YearOfBirthday',
    null,
    'Predict_YearOfBirthday',
    'FreD.Improve.Prediction.LogisticPredictor',
    'WekaPrediction',
    null

SELECT Sex, MaritalStatus, YearOfBirthday,
    q.Improve.PredictorPredict(
        'Predict_YearOfBirthday',
        q.Tools.Param( Sex, 0 ) + q.Tools.Param( MaritalStatus, 1 ) )
FROM Persons.Customers
```

Logistic Regression with ridge parameter of 1.0E-8
Coefficients...

²⁷ Das SQL-Beispiel funktioniert nur mit dem Plugin „WekaRuleInduction“

| Variable | Coeff. | |
|-----------|----------|----------|
| 1 | 1.2128 | -0.7487 |
| 2 | 18.8286 | 20.0272 |
| 3 | 4.063 | 0.2705 |
| 4 | -16.423 | 1.2674 |
| 5 | -17.8746 | -25.4368 |
| Intercept | -4.0365 | -2.0268 |

Odds Ratios...

| Variable | O.R. | |
|----------|----------------|----------------|
| 1 | 3.3628 | 0.473 |
| 2 | 150371511.5031 | 498523822.1046 |
| 3 | 58.149 | 1.3106 |
| 4 | 0 | 3.5515 |
| 5 | 0 | 0 |

| | | |
|----------------------------------|-----------|-----------|
| Correctly Classified Instances | 70 | 68.6275 % |
| Incorrectly Classified Instances | 32 | 31.3725 % |
| Kappa statistic | 0.5196 | |
| Mean absolute error | 0.2212 | |
| Root mean squared error | 0.3377 | |
| Relative absolute error | 51.2925 % | |
| Root relative squared error | 72.7276 % | |
| Total Number of Instances | 102 | |

| | |
|-------------------|-------------------------------|
| Namensraum | FreD.Improve.Prediction |
| Assembly | WekaPrediction |
| Klasse | MultiLayerPerceptronPredictor |

Beschreibung Ermittelt auf Basis von Eingabeattributen einer Tabelle den Wert für ein Ausgabeattribut über ein neuronales Netz.

Eigenschaften

| | |
|---------------------------|--|
| ClassifierName | Vollständiger Java-Klassenname in der Weka-Bibliothek (Default: weka.classifiers.functions.MultilayerPerceptron). |
| WekaOptions | Weka-Optionen, haben Vorrang vor den anderen Eigenschaften. |
| CrossValidationEvaluation | Führt eine Kreuzvalidierung durch (Default: false) |
| ConfusionMatrix | Gibt eine Konfusionsmatrix aus (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true. |
| ClassDetails | Detaillierte Ausgabe für jede Klasse (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true. |

Beispiel:²⁸

```
EXEC q.Improve.PredictorBuild
    'Sample.Persons.Customers',
    'Sex, MaritalStatus',
    'YearOfBirthday',
    null,
    'Predict_YearOfBirthday',
    'FreD.Improve.Prediction.MultilayerPerceptronPredictor',
    'WekaPrediction',
    'CrossValidationEvaluation=true'

SELECT Sex, MaritalStatus, YearOfBirthday,
    q.Improve.PredictorPredict(
        'Predict_YearOfBirthday',
        q.Tools.Param( Sex, 0 ) + q.Tools.Param( MaritalStatus, 1 ) )
FROM Persons.Customers
```

Sigmoid Node 0
 Inputs Weights
 Threshold 0.99538036675303787
 Node 3 -3.2007710608453914
 Node 4 1.17192812053411

²⁸ Das SQL-Beispiel funktioniert nur mit dem Plugin „WekaRuleInduction“

```

Node 5      -1.4518174715459062
Node 6      -3.1970914276910709
Sigmoid Node 1
  Inputs      Weights
  Threshold    -0.034139192468001539
  Node 3      2.6950700652637227
  Node 4      -4.4818098768433048
  Node 5      -2.3834792005996177
  Node 6      -1.1490721730188707
Sigmoid Node 2
  Inputs      Weights
  Threshold    -3.8708761574854567
  Node 3      -0.19673472763195973
  Node 4      1.8424649387380081
  Node 5      4.0954262877598664
  Node 6      4.2545524189808122
Sigmoid Node 3
  Inputs      Weights
  Threshold     0.66570268951633971
  Attrib Sex    -0.065252494335457822
  Attrib MaritalStatus=single  1.503091439141339
  Attrib MaritalStatus=married -3.4163039668569417
  Attrib MaritalStatus=divorced  0.38509571293920936
  Attrib MaritalStatus=widowed  0.061354959834904688
Sigmoid Node 4
  Inputs      Weights
  Threshold    -0.77652801943204464
  Attrib Sex    2.7699691891519334
  Attrib MaritalStatus=single  -2.7213815471053597
  Attrib MaritalStatus=married  0.62912868785595344
  Attrib MaritalStatus=divorced  0.42342834209180563
  Attrib MaritalStatus=widowed  3.0251891148744763
Sigmoid Node 5
  Inputs      Weights
  Threshold     0.79669112088210159
  Attrib Sex    -3.0291329611001858
  Attrib MaritalStatus=single  -3.6553549005310257
  Attrib MaritalStatus=married -0.74091091905337314
  Attrib MaritalStatus=divorced  1.5726571467605397
  Attrib MaritalStatus=widowed  1.24633172228247
Sigmoid Node 6
  Inputs      Weights
  Threshold    -0.75310220400548611
  Attrib Sex    2.4804210909493083
  Attrib MaritalStatus=single  -0.87660437584765893
  Attrib MaritalStatus=married -1.7076710138669786
  Attrib MaritalStatus=divorced  0.69594415286889555
  Attrib MaritalStatus=widowed  3.39063373437556
Class middle-aged
  Input
  Node 0
Class young
  Input
  Node 1
Class senior
  Input
  Node 2

```

| | | |
|----------------------------------|-----------|-----------|
| Correctly Classified Instances | 71 | 69.6078 % |
| Incorrectly Classified Instances | 31 | 30.3922 % |
| Kappa statistic | 0.5181 | |
| Mean absolute error | 0.2101 | |
| Root mean squared error | 0.3296 | |
| Relative absolute error | 48.7187 % | |
| Root relative squared error | 70.9807 % | |
| Total Number of Instances | 102 | |

| | |
|-------------------|-------------------------|
| Namensraum | FreD.Improve.Prediction |
| Assembly | WekaPrediction |
| Klasse | NaiveBayesPredictor |

Beschreibung Ermittelt auf Basis von Eingabeattributen einer Tabelle den Wert für ein Ausgabeattribut über einen Naive Bayes Klassifizierer.

Eigenschaften

| | |
|---------------------------|--|
| ClassifierName | Vollständiger Java-Klassenname in der Weka-Bibliothek (Default: weka.classifiers.bayes.NaiveBayes). |
| WekaOptions | Weka-Optionen, haben Vorrang vor den anderen Eigenschaften. |
| CrossValidationEvaluation | Führt eine Kreuzvalidierung durch (Default: false). |
| ConfusionMatrix | Gibt eine Konfusionsmatrix aus (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true. |
| ClassDetails | Detaillierte Ausgabe für jede Klasse (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true. |

Beispiel:²⁹

```
EXEC q.Improve.PredictorBuild
    'Sample.Persons.Customers',
    'Sex, MaritalStatus',
    'YearOfBirthday',
    null,
    'Predict_YearOfBirthday',
    'FreD.Improve.Prediction.NaiveBayesPredictor',
    'WekaPrediction',
    'CrossValidationEvaluation=true'

SELECT Sex, MaritalStatus, YearOfBirthday,
    q.Improve.PredictorPredict(
        'Predict_YearOfBirthday',
        q.Tools.Param( Sex, 0 ) + q.Tools.Param( MaritalStatus, 1 ) )
FROM Persons.Customers
```

²⁹ Das SQL-Beispiel funktioniert nur mit dem Plugin „WekaRuleInduction“

Naive Bayes Classifier

Class middle-aged: Prior probability = 0.29

Sex: Discrete Estimator. Counts = 18 13 (Total = 31)

MaritalStatus: Discrete Estimator. Counts = 3 28 1 1 (Total = 33)

Class young: Prior probability = 0.27

Sex: Discrete Estimator. Counts = 22 7 (Total = 29)

MaritalStatus: Discrete Estimator. Counts = 22 4 4 1 (Total = 31)

Class senior: Prior probability = 0.45

Sex: Discrete Estimator. Counts = 29 19 (Total = 48)

MaritalStatus: Discrete Estimator. Counts = 1 20 10 19 (Total = 50)

| | | |
|----------------------------------|-----------|-----------|
| Correctly Classified Instances | 75 | 73.5294 % |
| Incorrectly Classified Instances | 27 | 26.4706 % |
| Kappa statistic | 0.6056 | |
| Mean absolute error | 0.2609 | |
| Root mean squared error | 0.3475 | |
| Relative absolute error | 60.497 % | |
| Root relative squared error | 74.8359 % | |
| Total Number of Instances | 102 | |

| | |
|---------------------------|---|
| Namensraum | FreD.Improve.Prediction |
| Assembly | WekaPrediction |
| Klasse | NaiveBayesSimplePredictor |
| Beschreibung | Ermittelt auf Basis von Eingabeattributen einer Tabelle den Wert für ein Ausgabeattribut über einen einfachen Naive Bayes Klassifizierer. |
| Eigenschaften | |
| ClassifierName | Vollständiger Java-Klassenname in der Weka-Bibliothek (Default: weka.classifiers.bayes.NaiveBayesSimple). |
| WekaOptions | Weka-Optionen, haben Vorrang vor den anderen Eigenschaften. |
| CrossValidationEvaluation | Führt eine Kreuzvalidierung durch (Default: false). |
| ConfusionMatrix | Gibt eine Konfusionsmatrix aus (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true. |
| ClassDetails | Detaillierte Ausgabe für jede Klasse (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true. |

Beispiel:³⁰

```
EXEC q.Improve.PredictorBuild
    'Sample.Persons.Customers',
    'Sex, MaritalStatus',
    'YearOfBirthday',
    null,
    'Predict_YearOfBirthday',
    'FreD.Improve.Prediction.NaiveBayesSimplePredictor',
    'WekaPrediction',
    'CrossValidationEvaluation=true'

SELECT Sex, MaritalStatus, YearOfBirthday,
    q.Improve.PredictorPredict(
        'Predict_YearOfBirthday',
        q.Tools.Param( Sex, 0 ) + q.Tools.Param( MaritalStatus, 1 ) )
FROM Persons.Customers
```

³⁰ Das SQL-Beispiel funktioniert nur mit dem Plugin „WekaRuleInduction“

Naive Bayes (simple)

Class middle-aged: $P(C) = 0.28571429$

Attribute Sex

female male
0.58064516 0.41935484

Attribute MaritalStatus

single marrieddivorced widowed
0.09090909 0.84848485 0.03030303 0.03030303

Class young: $P(C) = 0.26666667$

Attribute Sex

female male
0.75862069 0.24137931

Attribute MaritalStatus

single marrieddivorced widowed
0.70967742 0.12903226 0.12903226 0.03225806

Class senior: $P(C) = 0.44761905$

Attribute Sex

female male
0.60416667 0.39583333

Attribute MaritalStatus

single marrieddivorced widowed
0.02 0.4 0.2 0.38

| | | |
|----------------------------------|-----------|-----------|
| Correctly Classified Instances | 75 | 73.5294 % |
| Incorrectly Classified Instances | 27 | 26.4706 % |
| Kappa statistic | 0.6056 | |
| Mean absolute error | 0.2609 | |
| Root mean squared error | 0.3475 | |
| Relative absolute error | 60.497 % | |
| Root relative squared error | 74.8359 % | |
| Total Number of Instances | 102 | |

| | |
|-------------------|-------------------------|
| Namensraum | FreD.Improve.Prediction |
| Assembly | WekaPrediction |
| Klasse | OneRPredictor |

Beschreibung Ermittelt auf Basis von Eingabeattributen einer Tabelle den Wert für ein Ausgabeattribut über den 1R-Algorithmus.

Eigenschaften

| | |
|---------------------------|--|
| ClassifierName | Vollständiger Java-Klassenname in der Weka-Bibliothek (Default: weka.classifiers.rules.OneR). |
| WekaOptions | Weka-Optionen, haben Vorrang vor den anderen Eigenschaften. |
| CrossValidationEvaluation | Führt eine Kreuzvalidierung durch (Default: false). |
| ConfusionMatrix | Gibt eine Konfusionsmatrix aus (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true. |
| ClassDetails | Detaillierte Ausgabe für jede Klasse (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true. |

Beispiel:³¹

```
EXEC q.Improve.PredictorBuild
    'Sample.Persons.Customers',
    'Sex, MaritalStatus',
    'YearOfBirthday',
    null,
    'Predict_YearOfBirthday',
    'FreD.Improve.Prediction.OneRPredictor',
    'WekaPrediction',
    'CrossValidationEvaluation=true'

SELECT Sex, MaritalStatus, YearOfBirthday,
    q.Improve.PredictorPredict(
        'Predict_YearOfBirthday',
        q.Tools.Param( Sex, 0 ) + q.Tools.Param( MaritalStatus, 1 ) )
FROM Persons.Customers
```

³¹ Das SQL-Beispiel funktioniert nur mit dem Plugin „WekaRuleInduction“

MaritalStatus:

single -> young
married-> middle-aged
divorced -> senior
widowed-> senior

(75/102 instances correct)

| | | |
|----------------------------------|-----------|-----------|
| Correctly Classified Instances | 75 | 73.5294 % |
| Incorrectly Classified Instances | 27 | 26.4706 % |
| Kappa statistic | 0.6056 | |
| Mean absolute error | 0.1765 | |
| Root mean squared error | 0.4201 | |
| Relative absolute error | 40.9208 % | |
| Root relative squared error | 90.4729 % | |
| Total Number of Instances | 102 | |

| | |
|-------------------|-------------------------|
| Namensraum | FreD.Improve.Prediction |
| Assembly | WekaPrediction |
| Klasse | BaggingPredictor |

Beschreibung Ermittelt auf Basis von Eingabeattributen einer Tabelle den Wert für ein Ausgabeattribut über das Bagging-Verfahren.

Eigenschaften

| | |
|---------------------------|--|
| ClassifierName | Vollständiger Java-Klassenname in der Weka-Bibliothek (Default: weka.classifiers.meta.Bagging). |
| WekaOptions | Weka-Optionen, haben Vorrang vor den anderen Eigenschaften. |
| CrossValidationEvaluation | Führt eine Kreuzvalidierung durch (Default: false). |
| ConfusionMatrix | Gibt eine Konfusionsmatrix aus (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true. |
| ClassDetails | Detaillierte Ausgabe für jede Klasse (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true. |
| ClassifierNameIntern | Weka-Klassifizierer, der im Bagging-Verfahren verwendet wird (Default: weka.classifiers.rules.ZeroR). |

Beispiel:³²

```
EXEC q.Improve.PredictorBuild
    'Sample.Persons.Customers',
    'Sex, MaritalStatus',
    'YearOfBirthday',
    null,
    'Predict_YearOfBirthday',
    'FreD.Improve.Prediction.BaggingPredictor',
    'WekaPrediction',
    'ClassifierNameIntern=weka.classifiers.rules.OneR,
    CrossValidationEvaluation=true'

SELECT Sex, MaritalStatus, YearOfBirthday,
    q.Improve.PredictorPredict(
        'Predict_YearOfBirthday',
        q.Tools.Param( Sex, 0 ) + q.Tools.Param( MaritalStatus, 1 ) )
FROM Persons.Customers
```

³² Das SQL-Beispiel funktioniert nur mit dem Plugin „WekaRuleInduction“

All the base classifiers:

```
MaritalStatus:
  single -> young
  married-> middle-aged
  divorced -> senior
  widowed-> senior
(73/102 instances correct)
```

```
MaritalStatus:
  single -> young
  married-> middle-aged
  divorced -> senior
  widowed-> senior
(73/102 instances correct)
```

```
MaritalStatus:
  single -> young
  married-> middle-aged
  divorced -> senior
  widowed-> senior
(76/102 instances correct)
```

...

```
MaritalStatus:
  single -> young
  married-> middle-aged
  divorced -> senior
  widowed-> senior
(72/102 instances correct)
```

| | | |
|----------------------------------|-----------|-----------|
| Correctly Classified Instances | 75 | 73.5294 % |
| Incorrectly Classified Instances | 27 | 26.4706 % |
| Kappa statistic | 0.6056 | |
| Mean absolute error | 0.1882 | |
| Root mean squared error | 0.4114 | |
| Relative absolute error | 43.6488 % | |
| Root relative squared error | 88.6107 % | |
| Total Number of Instances | 102 | |

qSQL

| | |
|-------------------|---------------------------|
| Namensraum | FreD.Improve.Prediction |
| Assembly | WekaPrediction |
| Klasse | LinearRegressionPredictor |

Beschreibung Ermittelt auf Basis von Eingabeattributen einer Tabelle den Wert für ein Ausgabeattribut über ein lineares Regressionsmodell. Unterstützt auch nominale Eingabeattribute.

Eigenschaften

| | |
|---------------------------|--|
| ClassifierName | Vollständiger Java-Klassenname in der Weka-Bibliothek (Default: weka.classifiers.functions.LinearRegression). |
| WekaOptions | Weka-Optionen, haben Vorrang vor den anderen Eigenschaften. |
| CrossValidationEvaluation | Führt eine Kreuzvalidierung durch (Default: false). |
| ConfusionMatrix | Gibt eine Konfusionsmatrix aus (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true. |
| ClassDetails | Detaillierte Ausgabe für jede Klasse (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true. |

Beispiel:³³

```
EXEC q.Improve.PredictorBuild
    'Sample.Sales.Articles',
    'Stock, Size, Weight',
    'ListPrice',
    'Size IS NOT NULL AND Weight IS NOT NULL',
    'Predict_ListPrice',
    'FreD.Improve.Prediction.LinearRegressionPredictor',
    'WekaPrediction',
    'CrossValidationEvaluation=true'

SELECT Stock, Size, Weight, ListPrice,
    q.Improve.PredictorPredict(
        'Predict_ListPrice',
        q.Tools.Param( Stock, 0 ) +
        q.Tools.Param( Size, 0 ) +
        q.Tools.Param( Weight, 1 ) )
FROM Sales.Articles
```

³³ Das SQL-Beispiel funktioniert nur mit dem Plugin „WekaRuleInduction“

Linear Regression Model

ListPrice =

```

        665.7298 * Size=58,62,52,60,50,54,44,48,42,46,38,56 +
        170.7948 * Size=62,52,60,50,54,44,48,42,46,38,56 +
    -836.8154 * Size=52,60,50,54,44,48,42,46,38,56 +
        479.9273 * Size=60,50,54,44,48,42,46,38,56 +
        657.5633 * Size=50,54,44,48,42,46,38,56 +
    -397.5503 * Size=54,44,48,42,46,38,56 +
    -739.6495 * Size=44,48,42,46,38,56 +
        408.1198 * Size=48,42,46,38,56 +
    -196.7499 * Size=42,46,38,56 +
        444.6499 * Size=46,38,56 +
    -444.6499 * Size=38,56 +
        454.3599 * Size=56 +
    -211.3698 *
Weight=2.92,3.02,3.10,3.20,3.14,3.16,2.32,2.40,2.48,2.46,2.50,2.73,2.77,2.81,2.85,2.36,20.79,
20.13,19.77,20.42,28.68,27.77,28.42,28.13,27.35,2.18,2.38,2.22,2.34,2.26,3.00,2.96,3.04,29.68
,30.00,28.77,29.90,29.42,29.79,29.13,26.77,26.35,18.77,20.00,19.42,19.13,19.79,19.90,27.13,27
.42,3.08,2.30,25.77,25.35,26.42,26.13,17.77,18.13,17.35,18.68,18.42,27.68,27.90,2.80,2.76,2.6
8,2.84,2.72,2.12,2.24,2.20,2.16,17.13,16.77,17.42,17.79,17.90,15.77,15.35,16.42,16.13,23.77,2
3.35,24.13,25.68,25.42,25.90,25.13,14.77,15.13,15.68,15.79,15.42,21.13,20.77,20.35,21.42,14.4
2,14.13,14.68,15.00,13.77 +
        287.8698 *

```

...

```

Weight=14.77,15.13,15.68,15.79,15.42,21.13,20.77,20.35,21.42,14.42,14.13,14.68,15.00,13.77 +
    -408.1198 *
Weight=15.13,15.68,15.79,15.42,21.13,20.77,20.35,21.42,14.42,14.13,14.68,15.00,13.77 +
    -257.61 * Weight=15.68,15.79,15.42,21.13,20.77,20.35,21.42,14.42,14.13,14.68,15.00,13.77
+
        666.0206 * Weight=15.42,21.13,20.77,20.35,21.42,14.42,14.13,14.68,15.00,13.77 +
        943.8492 * Weight=21.13,20.77,20.35,21.42,14.42,14.13,14.68,15.00,13.77 +
    -211.3698 * Weight=20.77,20.35,21.42,14.42,14.13,14.68,15.00,13.77 +
    -196.7499 * Weight=21.42,14.42,14.13,14.68,15.00,13.77 +
        599.1906 * Weight=14.42,14.13,14.68,15.00,13.77 +
    -408.4106 * Weight=14.13,14.68,15.00,13.77 +
    -257.61 * Weight=14.68,15.00,13.77 +
    -170.7948 * Weight=15.00,13.77 +
        836.5246 * Weight=13.77 +
        256.92

```

| | |
|-----------------------------|-----------|
| Correlation coefficient | 0.8119 |
| Mean absolute error | 241.1559 |
| Root mean squared error | 538.4216 |
| Relative absolute error | 33.3774 % |
| Root relative squared error | 58.8974 % |
| Total Number of Instances | 176 |

| | |
|-------------------|-------------------------|
| Namensraum | FreD.Improve.Prediction |
| Assembly | WekaPrediction |
| Klasse | M5RulesPredictor |

Beschreibung Ermittelt auf Basis von Eingabeattributen einer Tabelle den Wert für ein Ausgabeattribut über den M5-Algorithmus.

Eigenschaften

| | |
|---------------------------|--|
| ClassifierName | Vollständiger Java-Klassenname in der Weka-Bibliothek (Default: weka.classifiers.meta.Bagging). |
| WekaOptions | Weka-Optionen, haben Vorrang vor den anderen Eigenschaften. |
| CrossValidationEvaluation | Führt eine Kreuzvalidierung durch (Default: false). |
| ConfusionMatrix | Gibt eine Konfusionsmatrix aus (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true. |
| ClassDetails | Detaillierte Ausgabe für jede Klasse (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true. |

Beispiel:³⁴

```
EXEC q.Improve.PredictorBuild
    'Sample.Sales.Articles',
    'Stock, Size, Weight',
    'ListPrice',
    'Size IS NOT NULL AND Weight IS NOT NULL',
    'Predict_ListPrice',
    'FreD.Improve.Prediction.M5RulesPredictor',
    'WekaPrediction',
    'CrossValidationEvaluation=true'

SELECT Stock, Size, Weight, ListPrice,
    q.Improve.PredictorPredict(
        'Predict_ListPrice',
        q.Tools.Param( Stock, 0 ) +
        q.Tools.Param( Size, 0 ) +
        q.Tools.Param( Weight, 1 ) )
FROM Sales.Articles
```

³⁴ Das SQL-Beispiel funktioniert nur mit dem Plugin „WekaRuleInduction“

```

M5 pruned model rules
(using smoothed linear models) :
Number of Rules : 9

Rule: 1
IF
    Weight=15.77,15.35,16.42,16.13,23.77,23.35,24.13,25.68,25.42,25.90,25.13,14.77,15.13,15
    .68,15.79,15.42,21.13,20.77,20.35,21.42,14.42,14.13,14.68,15.00,13.77 > 0.5
THEN

ListPrice =
    -28.0914 * Size=52,60,50,54,44,48,42,46,38,56
    + 33.8038 * Size=60,50,54,44,48,42,46,38,56
    + 23.7558 * Size=50,54,44,48,42,46,38,56
    - 49.5578 * Size=44,48,42,46,38,56
    + 22.4534 * Size=46,38,56
    + 35.8865 *
Weight=2.36,20.79,20.13,19.77,20.42,28.68,27.77,28.42,28.13,27.35,2.18,2.38,2.22,2.34,2.26,3.0
0,2.96,3.04,29.68,30.00,28.77,29.90,29.42,29.79,29.13,26.77,26.35,18.77,20.00,19.42,19.13,19.7
9,19.90,27.13,27.42,3.08,2.30,25.77,25.35,26.42,26.13,17.77,18.13,17.35,18.68,18.42,27.68,27.9
0,2.80,2.76,2.68,2.84,2.72,2.12,2.24,2.20,2.16,17.13,16.77,17.42,17.79,17.90,15.77,15.35,16.42
,16.13,23.77,23.35,24.13,25.68,25.42,25.90,25.13,14.77,15.13,15.68,15.79,15.42,21.13,20.77,20.
35,21.42,14.42,14.13,14.68,15.00,13.77
    + 42.5333 *

...

Rule: 9

ListPrice =
    -230.19 * Size=62,52,60,50,54,44,48,42,46,38,56
    + 567.41 [4/16.846%]

Correlation coefficient          0.8181
Mean absolute error             269.5738
Root mean squared error        525.1147
Relative absolute error         37.3106 %
Root relative squared error     57.4418 %
Total Number of Instances      176

```

| | |
|-------------------|---------------------------|
| Namensraum | FreD.Improve.Prediction |
| Assembly | WekaPrediction |
| Klasse | LinearRegressionPredictor |

Beschreibung Ermittelt auf Basis von Eingabeattributen einer Tabelle den Wert für ein Ausgabeattribut über ein einfaches lineares Regressionsmodell.

Eigenschaften

| | |
|---------------------------|--|
| ClassifierName | Vollständiger Java-Klassenname in der Weka-Bibliothek (Default:weka.classifiers.functions.SimpleLinearRegression). |
| WekaOptions | Weka-Optionen, haben Vorrang vor den anderen Eigenschaften. |
| CrossValidationEvaluation | Führt eine Kreuzvalidierung durch (Default: false). |
| ConfusionMatrix | Gibt eine Konfusionsmatrix aus (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true. |
| ClassDetails | Detaillierte Ausgabe für jede Klasse (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true. |

Beispiel:³⁵

```
EXEC q.Improve.PredictorBuild
    'Sample.Sales.Articles',
    'Stock, StockReorderPoint',
    'ListPrice',
    'Size IS NOT NULL AND Weight IS NOT NULL',
    'Predict_ListPrice',
    'FreD.Improve.Prediction.SimpleLinearRegressionPredictor',
    'WekaPrediction',
    'CrossValidationEvaluation=true'

SELECT Stock, StockReorderPoint, ListPrice,
    q.Improve.PredictorPredict(
        'Predict_ListPrice',
        q.Tools.Param( Stock, 0 ) +
        q.Tools.Param( StockReorderPoint, 1 ) )
FROM Sales.Articles
```

³⁵ Das SQL-Beispiel funktioniert nur mit dem Plugin „WekaRuleInduction“

Linear regression on Stock

$-2.21 * \text{Stock} + 1809.03$

| | |
|-----------------------------|-----------|
| Correlation coefficient | 0.4561 |
| Mean absolute error | 681.5161 |
| Root mean squared error | 809.9723 |
| Relative absolute error | 94.3259 % |
| Root relative squared error | 88.6021 % |
| Total Number of Instances | 176 |

B.4.2 Fusion

Namensraum FreD.Improve.Deduplication.Fuse

Assembly Core

Klasse Concat

Beschreibung Aggregatfunktion, die mehrere Werte zu einer gemeinsamen Zeichenkette verbindet.

Eigenschaften

keine

Beispiel:

```
SELECT q.Improve.DeduplicationFuse(
    '{FreD.Improve.Deduplication.Fuse.Concat|Core}'+
    COALESCE( Name, '{NULL}') )
FROM   Persons.Customers
WHERE  [q.Group] IS NOT NULL
GROUP BY [q.Group]
```

qSQL

| | |
|----------------------|--|
| Namensraum | FreD.Improve.Deduplication.Fuse |
| Assembly | Core |
| Klasse | Discard |
| Beschreibung | Aggregatfunktion, die nur dann einen Wert zurückgibt, wenn alle Werte der Gruppe identisch sind. |
| Eigenschaften | keine |

Beispiel:

```
SELECT q.Improve.DeduplicationFuse(
    '{FreD.Improve.Deduplication.Fuse.Discard|Core}'+
    COALESCE( Name, '{NULL}') )
FROM   Persons.Customers
WHERE  [q.Group] IS NOT NULL
GROUP BY [q.Group]
```

qSQL

| | |
|----------------------|--|
| Namensraum | FreD.Improve.Deduplication.Fuse |
| Assembly | Core |
| Klasse | First |
| Beschreibung | Aggregatfunktion, die den ersten Wert einer Gruppe zurückgibt. |
| Eigenschaften | keine |

Beispiel:

```
SELECT q.Improve.DeduplicationFuse(
    '{FreD.Improve.Deduplication.Fuse.First|Core}'+
    COALESCE( Name, '{NULL}') )
FROM   Persons.Customers
WHERE  [q.Group] IS NOT NULL
GROUP BY [q.Group]
```

qSQL

| | |
|-------------------|---------------------------------|
| Namensraum | FreD.Improve.Deduplication.Fuse |
| Assembly | Core |
| Klasse | Last |

Beschreibung Aggregatfunktion, die den letzten Wert einer Gruppe zurückgibt.

Eigenschaften

keine

Beispiel:

```
SELECT q.Improve.DeduplicationFuse(
    '{FreD.Improve.Deduplication.Fuse.Last|Core}'+
    COALESCE( Name, '{NULL}') )
FROM   Persons.Customers
WHERE  [q.Group] IS NOT NULL
GROUP BY [q.Group]
```


qSQL

| | |
|----------------------|---|
| Namensraum | FreD.Improve.Deduplication.Fuse |
| Assembly | Core |
| Klasse | Max |
| Beschreibung | Aggregatfunktion, die den größten Wert einer Gruppe zurückgibt. |
| Eigenschaften | keine |

Beispiel:

```
SELECT q.Improve.DeduplicationFuse(
    '{FreD.Improve.Deduplication.Fuse.Max|Core}'+
    COALESCE( Name, '{NULL}') )
FROM   Persons.Customers
WHERE  [q.Group] IS NOT NULL
GROUP BY [q.Group]
```

qSQL

| | |
|----------------------|---|
| Namensraum | FreD.Improve.Deduplication.Fuse |
| Assembly | Core |
| Klasse | Min |
| Beschreibung | Aggregatfunktion, die den kleinsten Wert einer Gruppe zurückgibt. |
| Eigenschaften | keine |

Beispiel:

```
SELECT q.Improve.DeduplicationFuse(
    '{FreD.Improve.Deduplication.Fuse.Min|Core}'+
    COALESCE( Name, '{NULL}') )
FROM   Persons.Customers
WHERE  [q.Group] IS NOT NULL
GROUP BY [q.Group]
```

qSQL

| | |
|----------------------|--|
| Namensraum | FreD.Improve.Deduplication.Fuse |
| Assembly | Core |
| Klasse | MaxFrequency |
| Beschreibung | Aggregatfunktion, die den am häufigsten vorkommenden Wert der Gruppe zurückgibt. |
| Eigenschaften | keine |

Beispiel:

```
SELECT q.Improve.DeduplicationFuse(
    '{FreD.Improve.Deduplication.Fuse.MaxFrequency|Core}'+
    COALESCE( Name, '{NULL}') )
FROM   Persons.Customers
WHERE  [q.Group] IS NOT NULL
GROUP BY [q.Group]
```

| | |
|----------------------|---|
| Namensraum | FreD.Improve.Deduplication.Fuse |
| Assembly | Core |
| Klasse | LongestText |
| Beschreibung | Aggregatfunktion, die die längste Zeichenkette der Gruppe zurückgibt. |
| Eigenschaften | keine |

Beispiel:

```
SELECT q.Improve.DeduplicationFuse(
    '{FreD.Improve.Deduplication.Fuse.LongestText|Core}'+
    COALESCE( Name, '{NULL}') )
FROM   Persons.Customers
WHERE  [q.Group] IS NOT NULL
GROUP BY [q.Group]
```

qSQL

| | |
|----------------------|--|
| Namensraum | FreD.Improve.Deduplication.Fuse |
| Assembly | Core |
| Klasse | ShortestText |
| Beschreibung | Aggregatfunktion, die die kürzeste Zeichenkette der Gruppe zurückgibt. |
| Eigenschaften | keine |

Beispiel:

```
SELECT q.Improve.DeduplicationFuse(
    '{FreD.Improve.Deduplication.Fuse.ShortestText|Core}'+
    COALESCE( Name, '{NULL}') )
FROM   Persons.Customers
WHERE  [q.Group] IS NOT NULL
GROUP BY [q.Group]
```

C Datenmodell für Beispieldatenbank

