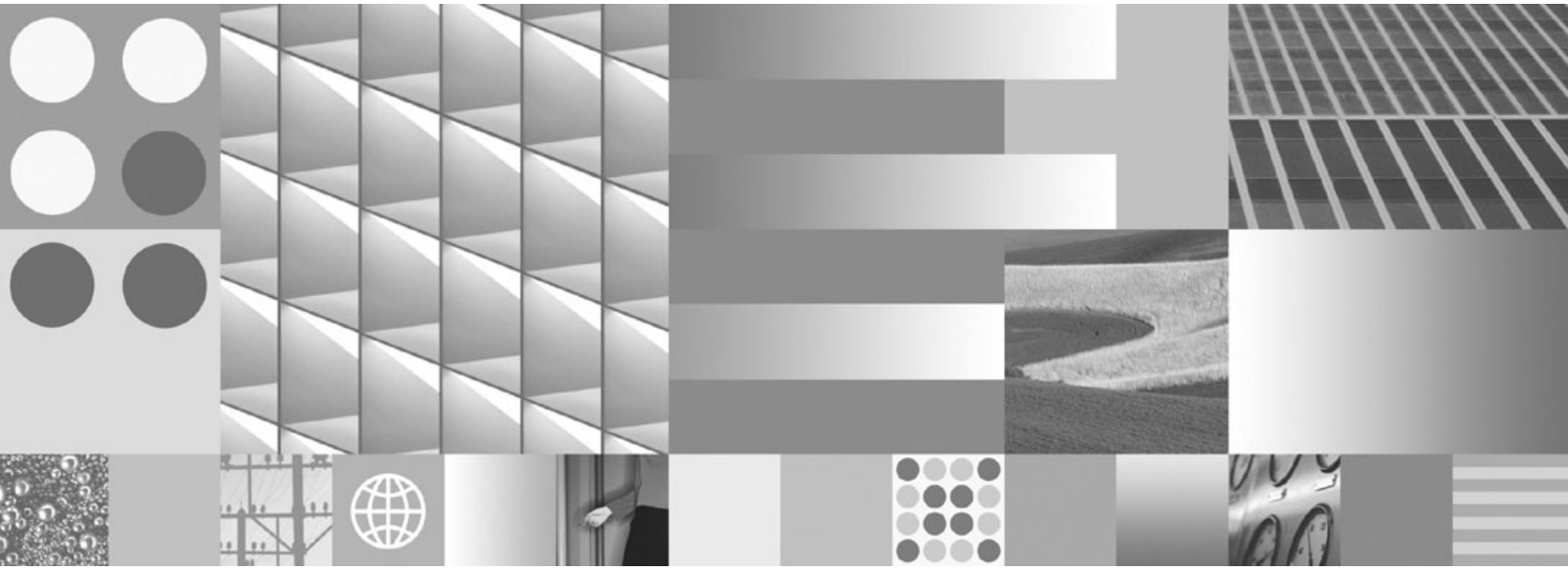
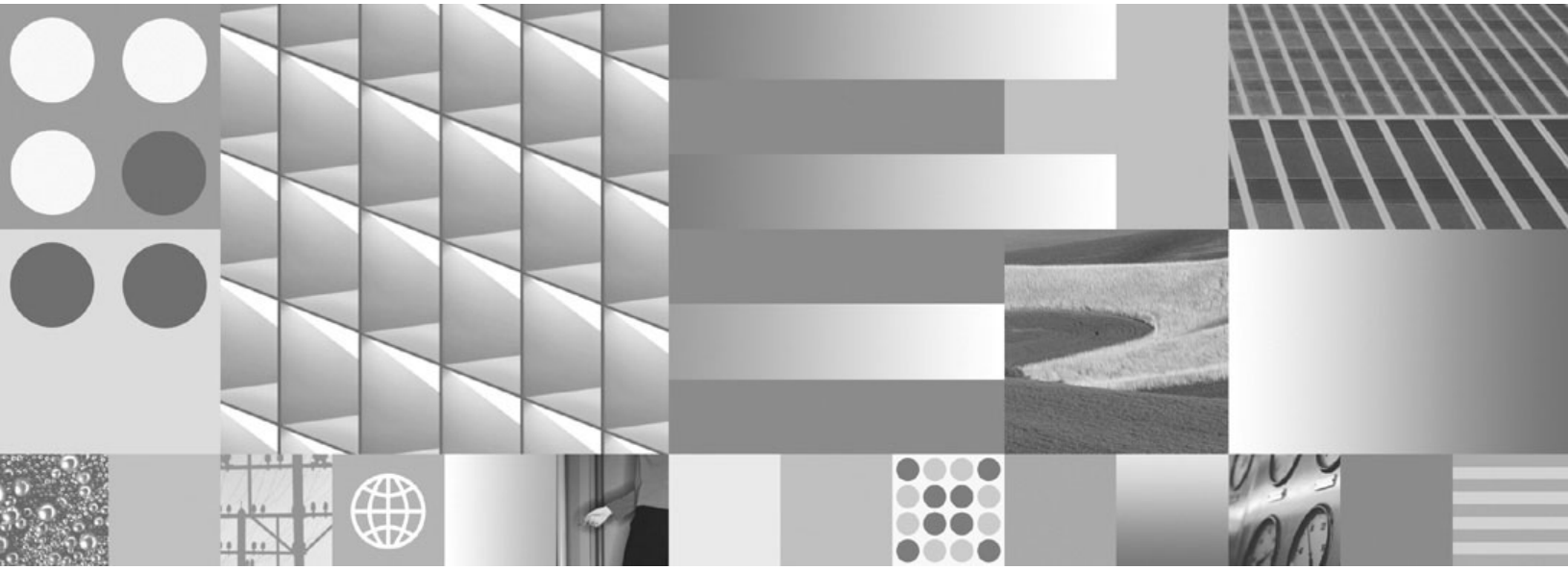
DB2 Version 9.5
für Linux, UNIX und Windows



# XQuery - Referenz

SC12-3931-02

DB2 Version 9.5
für Linux, UNIX und Windows



# XQuery - Referenz

SC12-3931-02

**Hinweis**

Vor Verwendung dieser Informationen und des darin beschriebenen Produkts sollten die allgemeinen Informationen in Anhang B, „Bemerkungen", auf Seite 251 gelesen werden.

Diese Veröffentlichung ist eine Übersetzung des Handbuchs
*IBM DB2 Version 9.5 for Linux, UNIX, and Windows, XQuery Reference*,
IBM Form SC23-5872-02,
herausgegeben von International Business Machines Corporation, USA

© Copyright International Business Machines Corporation 2006, 2009
© Copyright IBM Deutschland GmbH 2009

Informationen, die nur für bestimmte Länder Gültigkeit haben und für Deutschland, Österreich und die Schweiz nicht zutreffen, wurden in dieser Veröffentlichung im Originaltext übernommen.

Möglicherweise sind nicht alle in dieser Übersetzung aufgeführten Produkte in Deutschland angekündigt und verfügbar; vor Entscheidungen empfiehlt sich der Kontakt mit der zuständigen IBM Geschäftsstelle.

Änderung des Textes bleibt vorbehalten.

Herausgegeben von:
SW TSC Germany
Kst. 2877
April 2009

# Inhaltsverzeichnis


**Zu diesem Handbuch . . . . . vii**

**Kapitel 1. DB2 XQuery - Konzepte . . . 1**
Einführung zu XQuery . . . . . 1
Vergleich zwischen XQuery und SQL . . . . . 2
Abrufen von DB2-Daten mit XQuery-Funktionen . . 3
XQuery- und XPath-Datenmodell . . . . . 5
Sequenzen und Elemente . . . . . 5
Atomare Werte . . . . . 6
Knotenhierarchien . . . . . 6
Knotenmerkmale . . . . . 8
Knotensorten . . . . . 9
Dokumentreihenfolge der Knoten . . . . . 11
Knotenidentität . . . . . 11
Typisierte Werte und Zeichenfolgewerte von Knoten . . . . . 12
Serialisierung des XQuery- und XPath-Datenmodells (XDM) . . . . . 13
XML-Namensbereiche und qualifizierte Namen (QNames) . . . . . 14
Qualifizierte Namen (QNames) . . . . . 14
Statisch bekannte Namensbereiche . . . . . 15
Sprachkonventionen . . . . . 16
Groß-/Kleinschreibung . . . . . 16
Leerzeichen . . . . . 16
Kommentare . . . . . 16
Weitere Informationen zu XQuery . . . . . 17

**Kapitel 2. Typsystem . . . . . 19**
Die Typhierarchie . . . . . 19
Datentypen nach Kategorie . . . . . 20
Konstruktorfunktionen für integrierte Datentypen 26
Typumsetzung . . . . . 27
Datentyp 'anyAtomicType' . . . . . 30
Datentyp 'anySimpleType' . . . . . 30
Datentyp 'anyType' . . . . . 30
Datentyp 'anyURI' . . . . . 30
Datentyp 'base64Binary' . . . . . 30
Datentyp 'boolean' . . . . . 31
Datentyp 'byte' . . . . . 31
Datentyp 'date' . . . . . 31
Datentyp 'dateTime' . . . . . 32
Datentyp 'dayTimeDuration' . . . . . 33
Datentyp 'decimal' . . . . . 35
Datentyp 'double' . . . . . 35
Datentyp 'duration' . . . . . 35
Datentyp ENTITY . . . . . 37
Datentyp 'float' . . . . . 37
Datentyp 'gDay' . . . . . 38
Datentyp 'gMonth' . . . . . 38
Datentyp 'gMonthDay' . . . . . 39
Datentyp 'gYear' . . . . . 39
Datentyp 'gYearMonth' . . . . . 40
Datentyp 'hexBinary' . . . . . 40
Datentyp 'ID' . . . . . 40
Datentyp 'IDREF' . . . . . 40
Datentyp 'int' . . . . . 41
Datentyp 'integer' . . . . . 41
Datentyp 'language' . . . . . 41
Datentyp 'long' . . . . . 41
Datentyp 'Name' . . . . . 41
Datentyp 'NCName' . . . . . 42
Datentyp 'negativeInteger' . . . . . 42
Datentyp NMTOKEN . . . . . 42
Datentyp 'nonNegativeInteger' . . . . . 42
Datentyp 'nonPositiveInteger' . . . . . 42
Datentyp 'normalizedString' . . . . . 43
Datentyp NOTATION . . . . . 43
Datentyp 'positiveInteger' . . . . . 43
Datentyp 'QName' . . . . . 43
Datentyp 'short' . . . . . 44
Datentyp 'string' . . . . . 44
Datentyp 'time' . . . . . 44
Datentyp 'token' . . . . . 45
Datentyp 'unsignedByte' . . . . . 45
Datentyp 'unsignedInt' . . . . . 45
Datentyp 'unsignedLong' . . . . . 45
Datentyp 'unsignedShort' . . . . . 45
Datentyp 'untyped' . . . . . 46
Datentyp 'untypedAtomic' . . . . . 46
Datentyp 'yearMonthDuration' . . . . . 46

**Kapitel 3. Prolog . . . . . 49**
Versionsdeklaration . . . . . 50
Deklaration 'boundary-space' . . . . . 50
Deklaration 'construction' . . . . . 51
Deklaration 'copy-namespaces' . . . . . 51
Deklaration 'default element/type namespace' . . . 52
Deklaration 'default function namespace' . . . . 53
Deklaration 'empty order' . . . . . 54
Sortiermodusdeklaration . . . . . 54
Namensbereichsdeklaration . . . . . 55

**Kapitel 4. Ausdrücke . . . . . 57**
Ausdrücke auswerten und verarbeiten . . . . . 57
Dynamischer Kontext und Fokus . . . . . 57
Vorrangstellung . . . . . 57
Reihenfolge der Ergebnisse in XQuery-Ausdrücken . . . . . 58
Atomisierung . . . . . 61
Subtypsubstitution . . . . . 62
Typumstufung . . . . . 62
Effektiver Boolescher Wert . . . . . 63
Primärausdrücke . . . . . 64
Literale . . . . . 64
Variablenverweise . . . . . 66
Klammerausdruck . . . . . 67
Kontextelementausdrücke . . . . . 67
Funktionsaufrufe . . . . . 67
Pfadausdrücke . . . . . 68

Syntax von Pfadausdrücken . . . . . 69
Achsenschritte . . . . . 70
Abgekürzte Syntax für Pfadausdrücke . . . . 74
Vergleichselemente . . . . . 77
Sequenzausdrücke . . . . . 78
Ausdrücke zum Erstellen von Sequenzen . . . 78
Filterausdrücke . . . . . 79
Ausdrücke zum Kombinieren von Knotensequenzen . . . . . 80
Arithmetische Ausdrücke . . . . . 81
Vergleichsausdrücke . . . . . 84
Wertevergleiche . . . . . 84
Allgemeine Vergleiche . . . . . 86
Knotenvergleiche . . . . . 88
Logische Ausdrücke . . . . . 89
Konstruktoren . . . . . 90
Eingeschlossene Ausdrücke in Konstruktoren . . 91
Direkte Elementkonstruktoren . . . . . 92
Berechnete Elementkonstruktoren . . . . . 100
Berechnete Attributkonstruktoren. . . . . 101
Dokumentknotenkonstruktoren . . . . . 102
Textknotenkonstruktoren . . . . . 103
Verarbeitungsanweisungskonstruktoren . . . 104
Kommentarkonstruktoren . . . . . 105
FLWOR-Ausdrücke . . . . . 106
Syntax von FLWOR-Ausdrücken . . . . . 106
Klauseln **FOR** und **LET** . . . . . 108
Klauseln **WHERE** . . . . . 112
Klauseln **ORDER BY** . . . . . 112
Klauseln **RETURN** . . . . . 115
FLWOR-Beispiele . . . . . 115
Bedingungsausdrücke . . . . . 119
Quantifizierte Ausdrücke . . . . . 120
Umsetzungsausdrücke . . . . . 121
Ausdruck 'castable' . . . . . 122
Umsetzungsausdruck und Aktualisierungsausdrücke . . . . . 124
Aktualisierungsausdrücke in einem Umsetzungsausdruck verwenden. . . . . 124
Umsetzungsausdruck. . . . . 128
Grundlegende Aktualisierungsausdrücke . . . 131

**Kapitel 5. Integrierte Funktionen . . . 143**

Funktionen nach Kategorie . . . . . 143
Funktion 'adjust-date-to-timezone' . . . . . 150
Funktion 'adjust-dateTime-to-timezone'. . . . . 152
Funktion 'adjust-time-to-timezone' . . . . . 154
Funktion 'abs' . . . . . 155
Funktion 'avg' . . . . . 156
Funktion 'boolean' . . . . . 157
Funktion 'ceiling' . . . . . 158
Funktion 'codepoints-to-string' . . . . . 159
Funktion 'compare' . . . . . 160
Funktion 'concat' . . . . . 161
Funktion 'contains' . . . . . 161
Funktion 'count' . . . . . 162
Funktion 'current-date' . . . . . 162
Funktion 'current-dateTime' . . . . . 163
Funktion 'current-time' . . . . . 163
Funktion 'data' . . . . . 164
Funktion 'dateTime' . . . . . 164
Funktion 'day-from-date' . . . . . 165
Funktion 'day-from-dateTime' . . . . . 166
Funktion 'days-from-duration'. . . . . 166
Funktion 'deep-equal' . . . . . 167
Funktion 'default-collation' . . . . . 169
Funktion 'distinct-values' . . . . . 170
Funktion 'empty' . . . . . 170
Funktion 'ends-with' . . . . . 171
Funktion 'exactly-one' . . . . . 172
Funktion 'exists' . . . . . 172
Funktion 'false'. . . . . 172
Funktion 'floor'. . . . . 173
Funktion 'hours-from-dateTime' . . . . . 174
Funktion 'hours-from-duration' . . . . . 174
Funktion 'hours-from-time' . . . . . 175
Funktion 'implicit-timezone' . . . . . 176
Funktion 'in-scope-prefixes' . . . . . 176
Funktion 'index-of' . . . . . 177
Funktion 'insert-before' . . . . . 178
Funktion 'last' . . . . . 179
Funktion 'local-name' . . . . . 179
Funktion 'local-name-from-QName' . . . . . 180
Funktion 'lower-case' . . . . . 180
Funktion 'matches' . . . . . 181
Funktion 'max' . . . . . 183
Funktion 'min' . . . . . 184
Funktion 'minutes-from-dateTime' . . . . . 186
Funktion 'minutes-from-duration' . . . . . 186
Funktion 'minutes-from-time' . . . . . 187
Funktion 'month-from-date' . . . . . 188
Funktion 'month-from-dateTime' . . . . . 188
Funktion 'months-from-duration' . . . . . 189
Funktion 'name' . . . . . 190
Funktion 'namespace-uri' . . . . . 191
Funktion 'namespace-uri-for-prefix' . . . . . 192
Funktion 'namespace-uri-from-QName'. . . . . 192
Funktion 'node-name' . . . . . 193
Funktion 'normalize-space' . . . . . 193
Funktion 'normalize-unicode' . . . . . 194
Funktion 'not' . . . . . 195
Funktion 'number' . . . . . 196
Funktion 'one-or-more' . . . . . 196
Funktion 'position' . . . . . 197
Funktion 'QName'. . . . . 197
Funktion 'remove' . . . . . 198
Funktion 'replace' . . . . . 199
Funktion 'resolve-QName' . . . . . 201
Funktion 'reverse' . . . . . 202
Funktion 'root' . . . . . 203
Funktion 'round' . . . . . 203
Funktion 'round-half-to-even' . . . . . 204
Funktion 'seconds-from-dateTime' . . . . . 206
Funktion 'seconds-from-duration' . . . . . 206
Funktion 'seconds-from-time' . . . . . 207
Funktion 'sqlquery' . . . . . 208
Funktion 'starts-with'. . . . . 211
Funktion 'string' . . . . . 212
Funktion 'string-join' . . . . . 213
Funktion 'string-length' . . . . . 213
Funktion 'string-to-codepoints' . . . . . 214
Funktion 'subsequence' . . . . . 214

Funktion 'substring' . . . . . 215
Funktion 'substring-after' . . . . . 216
Funktion 'substring-before' . . . . . 217
Funktion 'sum' . . . . . 218
Funktion 'timezone-from-date' . . . . . 219
Funktion 'timezone-from-dateTime' . . . . . 220
Funktion 'timezone-from-time' . . . . . 220
Funktion 'tokenize' . . . . . 221
Funktion 'translate' . . . . . 223
Funktion 'true' . . . . . 224
Funktion 'unordered' . . . . . 224
Funktion 'upper-case' . . . . . 225
Funktion 'xmlcolumn' . . . . . 226
Funktion 'year-from-date' . . . . . 227
Funktion 'year-from-dateTime' . . . . . 228
Funktion 'years-from-duration' . . . . . 228
Funktion 'zero-or-one' . . . . . 229

**Kapitel 6. Reguläre Ausdrücke . . . . 231**

**Kapitel 7. Grenzwerte . . . . . 239**
Begrenzungen für XQuery-Datentypen . . . . . 239
Größenbegrenzungen . . . . . 240

**Anhang A. Übersicht über die technischen Informationen zu DB2 . . . . . 241**
Bibliothek mit technischen Informationen zu DB2 im Hardcopy- oder PDF-Format . . . . . 242
Bestellen gedruckter DB2-Bücher . . . . . 244
Aufrufen der Hilfe für den SQL-Status über den Befehlszeilenprozessor . . . . . 245
Zugriff auf verschiedene Versionen der DB2-Informationszentrale . . . . . 245
Anzeigen von Themen in der gewünschten Sprache in der DB2-Informationszentrale . . . . . 246
Aktualisieren der auf Ihrem Computer oder Intranet-Server installierten DB2-Informationszentrale . 247
DB2-Lernprogramme . . . . . 249
Informationen zur Fehlerbehebung in DB2 . . . 249
Bedingungen . . . . . 250

**Anhang B. Bemerkungen . . . . . 251**

**Index . . . . . 255**

# Zu diesem Handbuch

Im Handbuch 'XQuery - Referenz' wird die XQuery-Sprache beschrieben, die von einer DB2-Datenbank zur Bearbeitung von XML-Daten verwendet wird. Das Handbuch enthält Informationen zu XQuery-Konzepten, Datentypen, Sprachelementen, in XQuery definierten Funktionen und in DB2 definierten Funktionen. Darüber hinaus sind Informationen zu DB2-XQuery-Größenbegrenzungen sowie zu Begrenzungen für XQuery-Datentypen enthalten.

# Kapitel 1. DB2 XQuery - Konzepte

In den folgenden Abschnitten werden die Basiskonzepte von XQuery vorgestellt, und es wird beschrieben, wie XQuery mit einer DB2-Datenbank zusammenarbeitet.

## Einführung zu XQuery

Bei XQuery handelt es sich um eine funktionsorientierte Programmiersprache, die von W3C (World Wide Web Consortium) entwickelt wurde, um die speziellen Anforderungen zu erfüllen, die beim Abfragen und Ändern von XML-Daten gelten.

Anders als bei relationalen Daten, die vorhersehbar sind und über eine reguläre Struktur verfügen, sind XML-Daten sehr variabel strukturiert. XML-Daten können häufig nicht genau vorhergesehen werden, enthalten nur wenige Informationen und sind selbstbeschreibend.

Da die Struktur von XML-Daten nicht genau vorhersehbar ist, unterscheiden sich auch die Abfragen, die für XML-Daten ausgeführt werden müssen, häufig von normalen Abfragen für relationale Daten. Die XQuery-Sprache bietet die Flexibilität, die zur Ausführung dieser Operationen erforderlich ist. So kann es beispielsweise erforderlich sein, dass Sie die Sprache XQuery zur Ausführung der folgenden Operationen verwenden müssen:

- Durchsuchen von XML-Daten nach Objekten, die sich innerhalb der Hierarchie auf einer nicht bekannten Ebene befinden.
- Ausführen struktureller Konvertierungen für die Daten (z. B. zur Umkehrung einer Hierarchie).
- Zurückgeben von Ergebnissen mit gemischten Typen.
- Aktualisieren vorhandener XML-Daten.

### Komponenten einer XQuery-Abfrage

In XQuery werden Abfragen auf der Basis von Ausdrücken erstellt. Diese Ausdrücke können verschachtelt sein und bilden den Hauptteil einer Abfrage. Eine Abfrage kann außerdem über einen Prolog verfügen, der diesem Hauptteil vorangestellt ist. Der *Prolog* enthält eine Reihe von Deklarationen, mit denen die Verarbeitungsumgebung der Abfrage definiert wird. Der *Abfragehauptteil* besteht aus einem Ausdruck, der die Ergebnisse der Abfrage definiert. Dieser Ausdruck kann aus mehreren XQuery-Ausdrücken zusammengesetzt sein, die mithilfe von Operatoren oder Schlüsselwörtern kombiniert werden.

Abb. 1 auf Seite 2 zeigt die Struktur einer typischen Abfrage. In diesem Beispiel enthält der Prolog zwei Deklarationen: Eine Versionsdeklaration (version), in der die Version der XQuery-Syntax angegeben ist, die zur Verarbeitung der Abfrage verwendet werden soll, und eine Standarddeklaration für den Namensbereich (namespace), in der die Namensbereichs-URI angegeben ist, die für Element- und Typnamen ohne Präfix verwendet werden kann. Der Abfragehauptteil enthält einen Ausdruck, mit dem ein Element `price_list` erstellt werden kann. Der Inhalt des Elements `price_list` umfasst eine Liste mit `product`-Elementen, die in absteigender Reihenfolge nach Preis (price) sortiert sind.

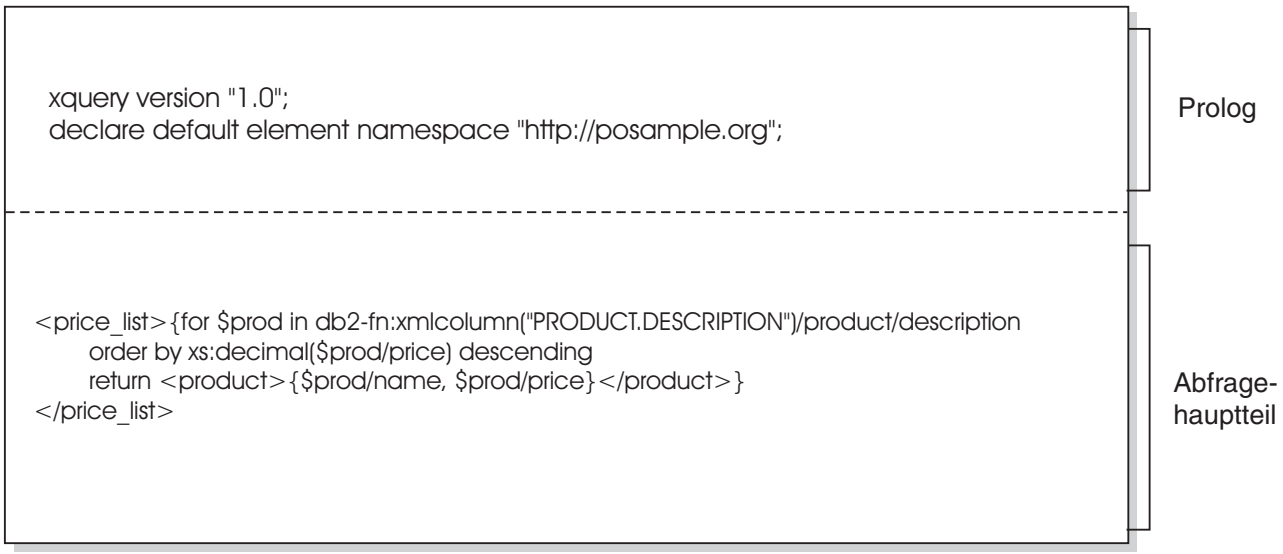



*Abbildung 1. Struktur einer typischen XQuery-Abfrage*

## Vergleich zwischen XQuery und SQL

DB2-Datenbanken unterstützen das Speichern korrekt formatierter XML-Daten in einer Spalte einer Tabelle und das Abrufen der XML-Daten aus der Datenbank mithilfe von SQL und/oder XQuery. Beide Sprachen werden als primäre Abfragesprachen unterstützt, und beide Sprachen bieten Funktionen zum Aufrufen der jeweils anderen Sprache.

**XQuery**

Eine Abfrage, die XQuery direkt aufruft, beginnt mit dem Schlüsselwort XQUERY. Dieses Schlüsselwort gibt an, dass XQuery verwendet wird und dass der DB2-Server aus diesem Grund die Regeln zur Beachtung der Groß-/Kleinschreibung verwenden muss, die für die XQuery-Sprache gelten. Die Fehlerbehandlung basiert auf den Schnittstellen, die zur Verarbeitung der XQuery-Ausdrücke verwendet werden. XQuery-Fehler werden in derselben Weise wie SQL-Fehler mit einem SQLCODE-Wert und einem SQLSTATE-Wert gemeldet. Bei der Verarbeitung von XQuery-Ausdrücken werden keine Warnungen ausgegeben. XQuery fordert Daten durch Aufruf von Funktionen an, mit denen XML-Daten aus DB2-Tabellen und -Sichten extrahiert werden. XQuery kann darüber hinaus auch über eine SQL-Abfrage aufgerufen werden. In diesem Fall kann die SQL-Abfrage XML-Daten in Form gebundener Variablen an XQuery übergeben. XQuery unterstützt verschiedene Ausdrücke für die Verarbeitung von XML-Daten und zum Erstellen neuer XML-Objekte wie z. B. Elemente und Attribute. Die Programmierschnittstelle für XQuery bietet Funktionen, die Ähnlichkeiten mit den entsprechenden SQL-Funktionen aufweisen und zur Vorbereitung von Abfragen und zum Abrufen von Abfrageergebnissen benutzt werden können.

**SQL** SQL bietet Funktionen, mit denen Werte des Datentyps XML definiert und instanziiert werden können. Zeichenfolgen, die korrekt formatierte XML-Dokumente enthalten, können über eine Syntaxanalyse in XML-Werte umgesetzt werden, optional auf der Basis eines XML-Schemas auf Ihre Gültigkeit überprüft und in Tabellen eingefügt oder in diesen aktualisiert werden. Alternativ hierzu können XML-Werte auch mithilfe einer SQL-

Konstruktorfunktion erstellt werden, mit der andere relationale Daten in XML-Werte umgesetzt werden. Das System bietet außerdem Funktionen zum Abfragen von XML-Daten mithilfe von XQuery und zum Konvertieren von XML-Daten in eine relationale Tabelle, die dann in einer SQL-Abfrage verwendet werden kann. Zusätzlich zur Serialisierung von XML-Werten in Zeichenfolgewerte können die Daten zwischen den Datentypen von SQL und XML umgesetzt werden.

SQL/XML bietet die folgenden Funktionen und Vergleichselemente für das Aufrufen von XQuery über SQL:

**XMLQUERY**
Bei XMLQUERY handelt es sich um eine Skalarfunktion, die einen XQuery-Ausdruck als Argument verwendet und eine XML-Sequenz zurückgibt. Die Funktion umfasst optionale Parameter, die zum Übergeben von SQL-Werten als XQuery-Variablen an den XQuery-Ausdruck verwendet werden können. Die XML-Werte, die von XMLQUERY zurückgegeben werden, können im Kontext der SQL-Abfrage weiter verarbeitet werden.

**XMLTABLE**
Bei XMLTABLE handelt es sich um eine Tabellenfunktion, die XQuery-Ausdrücke zum Generieren einer SQL-Tabelle auf der Basis von XML-Daten verwendet, die mit SQL weiter verarbeitet werden können.

**XMLEXISTS**
XMLEXISTS ist ein SQL-Vergleichselement, mit dem festgestellt werden kann, ob ein XQuery-Ausdruck eine Sequenz von einem oder mehreren Elementen (und keine leere Sequenz) zurückgibt.

## Abrufen von DB2-Daten mit XQuery-Funktionen

In XQuery kann eine Abfrage eine der folgenden Funktionen aufrufen, um XML-Eingabedaten aus einer DB2-Datenbank abzurufen: 'db2-fn:sqlquery' und 'db2-fn:xmlcolumn'.

Die Funktion 'db2-fn:xmlcolumn' ruft eine vollständige XML-Spalte ab, während die Funktion 'db2-fn:sqlquery' XML-Werte abruft, die auf einer SQL-Fullselect-Operation beruhen.

**db2-fn:xmlcolumn**
Die Funktion db2-fn:xmlcolumn verwendet ein Zeichenfolgeliteralargument, in dem eine XML-Spalte in einer Tabelle oder einer Sicht angegeben ist, und gibt eine Sequenz von XML-Werten zurück, die sich in dieser Spalte befinden. Beim Argument dieser Funktion muss die Groß-/Kleinschreibung beachtet werden. Das Zeichenfolgeliteralargument muss einen qualifizierten Spaltennamen des Typs XML angeben. Diese Funktion ermöglicht Ihnen das Extrahieren einer vollständigen Spalte von XML-Daten, ohne dass hierzu eine Suchbedingung angewendet werden muss.

Im folgenden Beispiel verwendet die Abfrage die Funktion db2-fn:xmlcolumn, um alle Bestellungen in der Spalte PURCHASE_ORDER der Tabelle BUSINESS.ORDERS abzurufen. Die Abfrage verwendet dann diese Eingabedaten, um die Städte aus der Versandadresse dieser Bestellungen zu extrahieren. Das Ergebnis der Abfrage ist eine Liste aller Städte, in die Bestellungen verschickt wurden.

```
db2-fn:xmlcolumn('BUSINESS.ORDERS.PURCHASE_ORDER')/shipping_address/city
```

**db2-fn:sqlquery**

Die Funktion db2-fn:sqlquery verwendet ein Zeichenfolgeargument, das eine Fullselect-Operation darstellt, und gibt eine XML-Sequenz zurück, die eine Verknüpfung der XML-Werte darstellt, die von der Fullselect-Operation zurückgegeben werden. Die Fullselect-Operation muss eine aus einer Zeile bestehende Ergebnismenge angeben, wobei diese Spalte den Datentyp XML aufweisen muss. Durch die Angabe einer Fullselect-Operation können Sie die Leistungsfähigkeit von SQL zur Bereitstellung von XML-Daten für XQuery nutzen. Die Funktion unterstützt die Verwendung von Parametern zum Übergeben von Werten an die SQL-Anweisung.

Im folgenden Beispiel enthält die Tabelle BUSINESS.ORDERS eine XML-Spalte mit dem Namen PURCHASE_ORDER. Die Abfrage in diesem Beispiel verwendet die Funktion db2-fn:sqlquery, um SQL aufzurufen und alle Bestellungen abzurufen, die das Versanddatum 15. Juni 2005 aufweisen. Die Abfrage verwendet dann diese Eingabedaten, um die Städte aus den Versandadressen dieser Bestellungen zu extrahieren. Das Ergebnis der Abfrage ist eine Liste aller Städte, in die am 15. Juni Bestellungen verschickt wurden.

```
db2-fn:sqlquery("
SELECT purchase_order FROM business.orders
WHERE ship_date = '2005-06-15' ")/shipping_address/city
```

**Wichtig:** Eine XML-Sequenz, die von der Funktion db2-fn:sqlquery oder db2-fn:xmlcolumn zurückgegeben wird, kann alle XML-Werte einschließlich atomarer Werte und Knoten enthalten. Diese Funktionen geben nicht immer eine Sequenz von korrekt formatierten Dokumenten zurück. Die Funktion kann z. B. einen einzelnen atomaren Wert (z. B. 36) als Instanz des Datentyps XML zurückgeben.

SQL und XQuery verwenden bei der Beachtung der Groß-/Kleinschreibung unterschiedliche Konventionen. Sie müssen diese Unterschiede berücksichtigen, wenn Sie die Funktionen db2-fn:sqlquery und db2-fn:xmlcolumn verwenden.

**Bei SQL muss die Groß-/Kleinschreibung nicht beachtet werden.**

Standardmäßig werden alle Standardbezeichner, die in SQL-Anweisungen verwendet werden, automatisch in Großschreibung umgesetzt. Aus diesem Grund werden die Namen von SQL-Tabellen und -Spalten normalerweise in Großbuchstaben angezeigt. In den obigen Beispielen trifft dies z. B. für BUSINESS.ORDERS und PURCHASE_ORDER zu. In einer SQL-Anweisung können Sie mit Kleinbuchstaben auf diese Spalten verweisen und z. B. business.orders und purchase_order verwenden, da diese Angaben automatisch während der Verarbeitung der SQL-Anweisung in Großbuchstaben umgesetzt werden. (Sie können auch einen Namen erstellen, bei dem die Groß-/Kleinschreibung beachtet werden muss und der in SQL als *begrenzter Bezeichner* bezeichnet wird. Setzen Sie hierzu den Namen in doppelte Anführungszeichen.)

**Bei XQuery muss die Groß-/Kleinschreibung beachtet werden.**

Namen in Kleinbuchstaben werden von XQuery nicht in Großschreibung umgesetzt. Dieser Unterschied kann bei der gemeinsamen Verwendung von XQuery und SQL zu Problemen führen. Die Zeichenfolge, die an db2-fn:sqlquery übergeben wird, wird als SQL-Abfrage interpretiert und vom SQL-Parser syntaktisch analysiert. Hierbei werden alle Namen in Großbuchstaben umgesetzt. Im Beispiel mit db2-fn:sqlquery können der Tabellenname business.orders und die Spaltennamen purchase_order und ship_date sowohl in Groß- als auch in Kleinschreibung angegeben werden. Der Operand von db2-fn:xmlcolumn ist allerdings keine SQL-Abfrage. Hierbei handelt es sich vielmehr um ein XQuery-Zeichenfolgeliteral, bei

dem die Groß-/Kleinschreibung beachtet werden muss und das für den Namen einer Spalte steht. Da der tatsächliche Name der Spalte BUSINESS-.ORDERS.PURCHASE_ORDER lautet, muss dieser Name im Operanden von db2-fn:xmlcolumn in Großbuchstaben angegeben werden.

# XQuery- und XPath-Datenmodell

XQuery-Ausdrücke arbeiten mit Instanzen des XQuery- und des XPath-Datenmodells (XDM) und geben Instanzen des Datenmodells zurück. Das XQuery- und XPath-Datenmodell stellt eine abstrakte Darstellung eines oder mehrerer XML-Dokumente oder -Fragmente zur Verfügung. Das Datenmodell definiert alle gültigen Werte von Ausdrücken in XQuery, einschließlich der Werte, die während der Zwischenberechnungen verwendet werden.

Das Parsing (Syntaxanalyse) von XML-Daten in das XQuery- und XPath-Datenmodell und die Gültigkeitsprüfung der Daten auf der Basis eines Schemas werden vor der Verarbeitung der Daten durch XQuery ausgeführt. Während der Datenmodellgenerierung wird das XML-Eingabedokument syntaktisch analysiert und in eine Instanz des XQuery- und XPath-Datenmodells konvertiert. Das Dokument kann mit oder ohne Gültigkeitsprüfung syntaktisch analysiert werden.

Das XQuery- und XPath-Datenmodell wird anhand von Sequenzen aus atomaren Werten und Knoten beschrieben.

## Sequenzen und Elemente

Eine Instanz des XQuery- und XPath-Datenmodells (XDM) ist eine Sequenz. Eine *Sequenz* ist eine sortierte Gruppe aus null oder mehr Elementen. Ein *Element* ist entweder ein atomarer Wert oder ein Knoten.

Eine Sequenz kann Knoten, atomare Werte oder eine Kombination aus Knoten und atomaren Werten enthalten. Jeder Eintrag in der folgenden Liste beispielsweise ist eine Sequenz:

- `36`
- `<dog/>`
- `(2, 3, 4)`
- `(36, <dog/>, "cat")`
- `()`

Zusätzlich zu den Einträgen in der Liste, ist auch ein XML-Dokument, das in einer XML-Spalte in einer DB2-Datenbank gespeichert ist, eine Sequenz.

In den Beispielen wird zur Darstellung von Sequenzen eine Notation verwendet, die der Syntax entspricht, die zum Erstellen von Sequenzen in XQuery verwendet wird:

- Jedes Element in der Sequenz wird durch ein Komma von den restlichen Elementen getrennt.
- Eine ganze Sequenz wird in runde Klammern eingeschlossen.
- Zur Darstellung einer leeren Sequenz wird ein Paar leerer runder Klammern verwendet.
- Ein einzelnes Element, das alleine steht, entspricht einer Sequenz, die ein Element enthält.

  Es gibt z. B. keinen Unterschied zwischen der Sequenz (36) und dem atomaren Wert 36.

Sequenzen dürfen nicht verschachtelt sein. Wenn zwei Sequenzen kombiniert werden, dann wird als Ergebnis immer eine abgewickelte Sequenz von Knoten und atomaren Werten generiert. Wenn Sie z. B. die Sequenz (2, 3) an die Sequenz (3, 5, 6) anfügen, dann wird eine Sequenz generiert, die (3, 5, 6, 2, 3) lautet. Durch die Kombination dieser Sequenzen kann nicht die Sequenz (3, 5, 6, (2, 3)) generiert werden, da verschachtelte Sequenzen niemals auftreten.

Eine Sequenz, die null Elemente enthält, wird als *leere Sequenz* bezeichnet. Leere Sequenzen können zur Darstellung fehlender oder unbekannter Informationen verwendet werden.

## Atomare Werte

Als *atomarer Wert* wird eine Instanz eines der integrierten atomaren Datentypen bezeichnet, die mit XML-Schemata definiert werden können. Diese Datentypen umfassen Zeichenfolgen, ganze Zahlen (Integer), Dezimalzahlen, Datumsangaben und weitere atomare Typen. Diese Typen werden als atomar bezeichnet, weil sie nicht weiter in kleinere Einheiten unterteilt werden können.

Im Gegensatz zu Knoten verfügen atomare Werte nicht über eine Identität. Jede Instanz eines atomaren Wertes (z. B. die ganze Zahl 7) ist identisch mit allen anderen Instanzen dieses Wertes.

Die folgenden Beispiele zeigen verschiedene Möglichkeiten zur Erstellung von atomaren Werten:

- Extraktion aus Knoten über den so genannten Atomisierungsprozess. Die Atomisierung wird von Ausdrücken immer dann verwendet, wenn eine Sequenz atomarer Werte erforderlich ist.
- Angabe als numerischer oder Zeichenfolgeliteralwert. Literalwerte werden von XQuery als atomare Werte interpretiert. Die folgenden Literalwerte werden z. B. als atomare Werte interpretiert:
  - ″dies ist eine zeichenfolge″ (Der Typ lautet xs:string)
  - 45 (Der Typ lautet xs:integer)
  - 1,44 (Der Typ lautet xs:decimal)
- Berechnung mit Konstruktorfunktionen. Die folgende Konstruktorfunktion beispielsweise kann zur Erstellung eines Wertes des Typs 'xs:date' auf der Basis der Zeichenfolge ″2005-01-01″ verwendet werden:
  ```
  xs:date("2005-01-01")
  ```
- Rückgabe durch die integrierten Funktionen fn:true() und fn:false(). Diese Funktionen geben die Booleschen Werte `true` (wahr) und `false` (falsch) zurück. Diese Werte können nicht als Literalwerte dargestellt werden.
- Rückgabe durch viele Arten von Ausdrücken wie z. B. arithmetischen Ausdrücken und logischen Ausdrücken.

## Knotenhierarchien

Die Knoten einer Sequenz bilden mindestens eine *Hierarchie* oder *Baumstruktur*, die aus einem Rootknoten und allen anderen Knoten besteht, die über den Rootknoten direkt oder indirekt aufgerufen werden können. Jeder Knoten gehört zu genau einer Hierarchie, und jede Hierarchie verfügt über genau einen Rootknoten. DB2 unterstützt sechs verschiedene Knotensorten: Dokument-, Element-, Attribut-, Text-, Verarbeitungsanweisungs- und Kommentarknoten.

Im Folgenden ist das XML-Dokument products.xml aufgeführt, das ein Rootelement mit dem Namen `products` (Produkte) enthält, das verschiedene Produkt-

elemente (product) umfasst. Jedes dieser product-Elemente verfügt über ein Attribut mit dem Namen pid (Produkt-ID) und über ein untergeordnetes Element (Kind) mit dem Namen description (Beschreibung). Das Element description enthält untergeordnete Elemente mit dem Namen name und price (Preis).

```
<products>
  <product xmlns="http://posample.org" pid="10">
    <description>
      <name>Fleecejacke</name>
      <price>19,99</price>
    </description>
  </product>
  <product xmlns="http://posample.org" pid="11">
    <description>
      <name>Nylonstrumpfhosen</name>
      <price>9,99</price>
    </description>
  </product>
</products>
```

Abb. 2 zeigt eine vereinfachte Darstellung des Datenmodells für products.xml. Die Abbildung umfasst einen Dokumentknoten (D), Elementknoten (E), Attributknoten (A) sowie Textknoten (T).

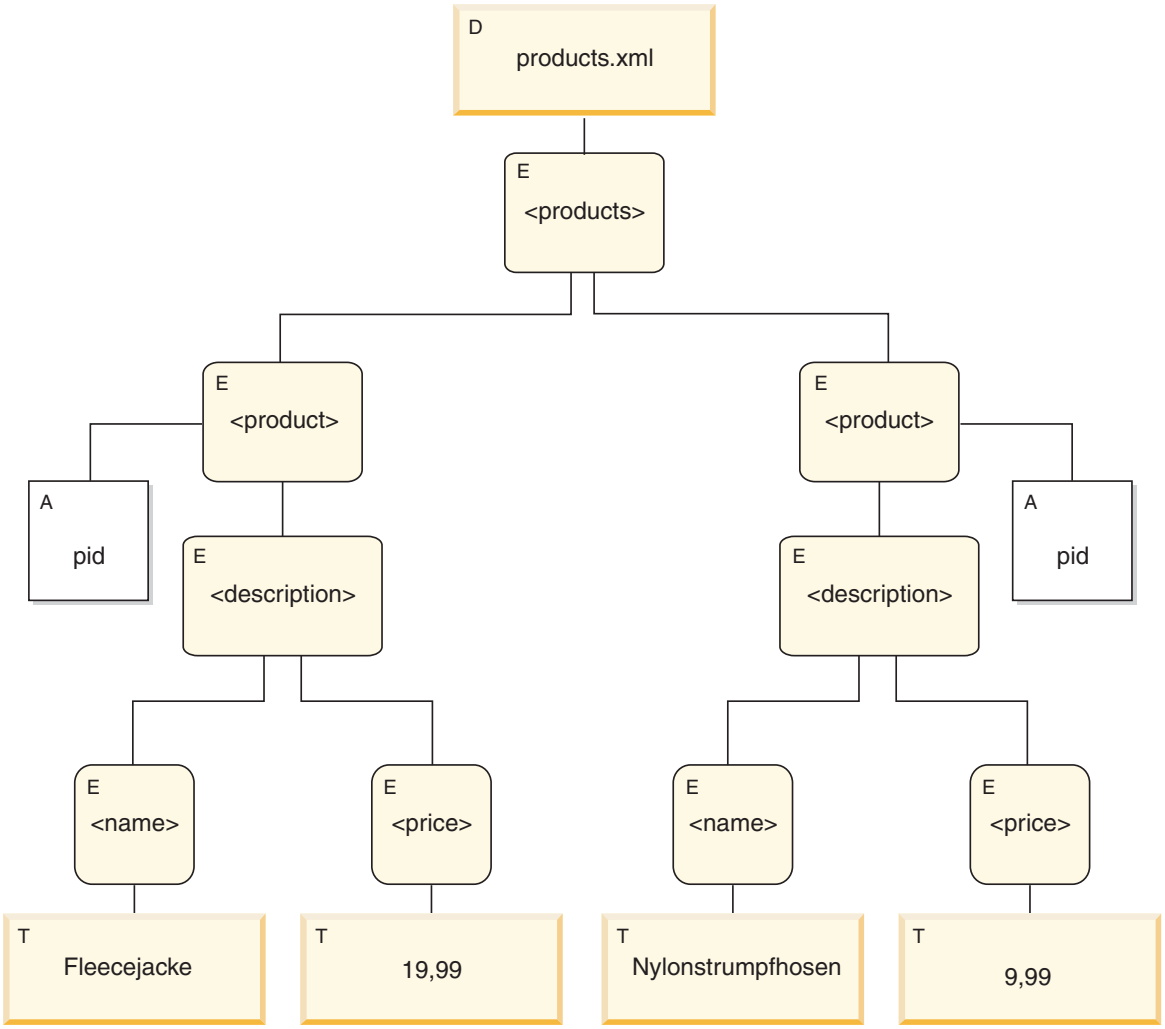



*Abbildung 2. Datenmodelldiagramm für das Dokument products.xml*

Wie im Beispiel dargestellt, kann ein Knoten weitere Knoten als untergeordnete Elemente umfassen und so eine oder mehrere *Knotenhierarchien* bilden. Im vorliegenden Beispiel stellt das Element `product` ein untergeordnetes Element von `products` dar. Das Element `description` ist ein untergeordnetes Element von `product`. Die Elemente `name` und `price` sind untergeordnete Elemente des Elements `description`. Der Textknoten mit dem Wert `Fleecejacke` ist ein untergeordnetes Element des Elementknotens `name`, und der Textknoten `19,99` ist ein untergeordnetes Element von `price`.

## Knotenmerkmale

Jeder Knoten verfügt über bestimmte *Merkmale*, in denen die Kenndaten dieses Knotens beschrieben sind. In den Knotenmerkmalen können z. B. der Knotenname, seine untergeordneten und übergeordneten Elemente, Attribute und weitere Informationen festgelegt sein, die den Knoten beschreiben. Die Knotensorte legt hierbei fest, welche Merkmale für bestimmte Knoten definiert sind.

Ein Knoten kann über eines oder mehrere der nachfolgend aufgeführten Merkmale verfügen:

**Knotenname (node-name)**
Der Name des Knotens in Form eines QName.

**Übergeordnetes Element (parent)**
Der Knoten, der als übergeordnetes Element des aktuellen Knotens definiert ist.

**Typname (type-name)**
Der dynamische Typ (Laufzeittyp) des Knotens (wird auch als *Typenannotation*) bezeichnet.

**Untergeordnete Elemente (children)**
Die Knotensequenz, die untergeordnete Elemente des aktuellen Knotens darstellen.

**Attribute**
Die Gruppe der Attributknoten, die zum aktuellen Knoten gehören.

**Zeichenfolgewert (string-value)**
Ein Zeichenfolgewert, der aus dem Knoten extrahiert werden kann.

**Typisierter Wert (typed-value)**
Eine Sequenz von null oder mehr atomaren Werten, die aus dem Knoten extrahiert werden können.

**Gültige Namensbereiche (in-scope namespaces)**
Die gültigen Namensbereiche, die dem Knoten zugeordnet sind.

**Inhalt (content)**
Der Inhalt des Knotens.

## Knotensorten

DB2 unterstützt sechs verschiedene Knotensorten: Dokument-, Element-, Attribut-, Text-, Verarbeitungsanweisungs- und Kommentarknoten.

### Dokumentknoten

Ein Dokumentknoten dient zur Einbindung eines XML-Dokuments.

Ein Dokumentknoten kann null oder mehr untergeordnete Elemente (Kinder) aufweisen. Diese untergeordneten Elemente können Elementknoten, Verarbeitungsanweisungsknoten, Kommentar- und Textknoten umfassen.

Der Zeichenfolgewert eines Dokumentknotens entspricht dem verknüpften Inhalt aller untergeordneten Textknoten in der Reihenfolge der Dokumente. Der Typ des Zeichenfolgewertes lautet xs:string. Der typisierte Wert eines Dokumentknotens stimmt mit seinem Zeichenfolgewert überein. Eine Ausnahme bildet hierbei allerdings die Tatsache, dass der Typ des typisierten Wertes xdt:untypedAtomic lautet.

Ein Dokumentknoten verfügt über die folgenden Knotenmerkmale:

- Untergeordnete Elemente (children), möglicherweise nicht angegeben
- Zeichenfolgewert (string-value)
- Typisierter Wert (typed-value)

Dokumentknoten können in XQuery-Ausdrücken mithilfe berechneter Konstruktoren erstellt werden. Eine Sequenz aus Dokumentknoten kann auch von der Funktion db2-fn:xmlcolumn zurückgegeben werden.

### Elementknoten

Ein Elementknoten dient zur Einbindung eines XML-Elements.

Ein Elementknoten verfügt über null übergeordnete Elemente bzw. ein übergeordnetes Element sowie über null oder mehr untergeordnete Elemente. Diese untergeordneten Elemente können Elementknoten, Verarbeitungsanweisungsknoten, Kommentar- und Textknoten umfassen. Dokument- und Attributknoten werden nie als untergeordnete Elemente von Elementknoten verwendet. Allerdings wird ein Elementknoten als übergeordnetes Element seiner Attribute interpretiert. Die Attribute eines Elementknotens müssen über eindeutige QNames verfügen.

Ein Elementknoten weist die folgenden Knotenmerkmale auf:

- Knotenname (node-name)
- Übergeordnetes Element (parent), möglicherweise nicht angegeben
- Typname (type-name)
- Untergeordnete Elemente (children), möglicherweise nicht angegeben
- Attribute (attributes), möglicherweise nicht angegeben
- Zeichenfolgewert (string-value)
- Typisierter Wert (typed-value)
- Bereichsinterne Namensbereiche (in-scope-namespaces)

Elementknoten können in XQuery-Ausdrücken mithilfe direkter oder berechneter Konstruktoren erstellt werden.

Das Merkmal 'type-name' eines Elementknotens gibt die Beziehung zwischen seinem typisierten Wert und seinem Zeichenfolgewert an. Wenn ein Elementknoten z. B. das type-name-Merkmal xs:decimal und den Zeichenfolgewert '47.5' aufweist,

dann wird als typisierter Wert der Dezimalwert 47.5 verwendet. Wenn das type-name-Merkmal eines Elementknotens xdt:untyped lautet, dann stimmt der typisierte Wert des Elements mit seinem Zeichenfolgewert überein und weist den Typ xdt:untypedAtomic auf.

### Attributknoten

Ein Attributknoten stellt ein XML-Attribut dar.

Ein Attributknoten verfügt entweder über kein oder ein übergeordnetes Element. Der Elementknoten, der als Eigner eines Attributs definiert ist, wird als übergeordnetes Element behandelt, obwohl ein Attributknoten kein untergeordnetes Element dieses Elements ist.

Ein Attributknoten weist die folgenden Knotenmerkmale auf:

- Knotenname (node-name)
- Übergeordnetes Element (parent), möglicherweise nicht angegeben
- Typname (type-name)
- Zeichenfolgewert (string-value)
- Typisierter Wert (typed-value)

Attributknoten können in XQuery-Ausdrücken mithilfe direkter oder berechneter Konstruktoren erstellt werden.

Das Merkmal 'type-name' eines Attributknotens gibt die Beziehung zwischen seinem typisierten Wert und seinem Zeichenfolgewert an. Wenn ein Attributknoten z. B. das type-name-Merkmal xs:decimal und den Zeichenfolgewert ″47,5″ aufweist, dann wird als typisierter Wert der Dezimalwert 47,5 verwendet.

### Textknoten

Ein Textknoten dient zur Einbindung von Inhalten mit XML-Zeichen.

Ein Textknoten kann null oder ein übergeordnetes Element aufweisen. Textknoten, bei denen es sich um untergeordnete Elemente eines Dokument- oder Elementknotens handelt, treten niemals als benachbarte gleichgeordnete Elemente auf. Wenn ein Dokument- oder Elementknoten erstellt wird, werden alle benachbarten, gleichgeordneten Textknoten zu einem einzigen Textknoten kombiniert. Wenn der resultierende Textknoten leer ist, wird er gelöscht.

Textknoten weisen die folgenden Knotenmerkmale auf:

- Inhalt (content), möglicherweise nicht angegeben
- Übergeordnetes Element (parent), möglicherweise nicht angegeben

Textknoten können in XQuery-Ausdrücken mithilfe berechneter Konstruktoren oder mit der Aktion eines direkten Elementkonstruktors erstellt werden.

### Verarbeitungsanweisungsknoten

Ein Verarbeitungsanweisungsknoten dient zum Einbinden einer XML-Verarbeitungsanweisung.

Ein Verarbeitungsanweisungsknoten kann null übergeordnete Elemente oder ein übergeordnetes Element aufweisen. Die Verwendung der Zeichenfolge ?> ist in einer Verarbeitungsanweisung nicht zulässig. Als Zieleinheit einer Verarbeitungsan-

weisung muss ein Name ohne Doppelpunkte (NCName) angegeben sein. Die Zieleinheit wird verwendet, um die Anwendung zu identifizieren, für die die Anweisung bestimmt ist.

Ein Verarbeitungsanweisungsknoten verfügt über die folgenden Knotenmerkmale:

- Ziel (target)
- Inhalt (content)
- Übergeordnetes Element (parent), möglicherweise nicht angegeben

Verarbeitungsanweisungsknoten können in XQuery-Ausdrücken mithilfe direkter oder berechneter Konstruktoren erstellt werden.

### Kommentarknoten

Ein Kommentarknoten dient zur Einbindung eines XML-Kommentars.

Ein Kommentarknoten kann null übergeordnete Elemente oder ein übergeordnetes Element aufweisen. Der Inhalt eines Kommentarknotens darf weder die Zeichenfolge "--" (zwei Silbentrennstriche) noch einen Silbentrennstrich (-) als letztes Zeichen enthalten.

Ein Kommentarknoten verfügt über die folgenden Knotenmerkmale:

- Inhalt (content)
- Übergeordnetes Element (parent), möglicherweise leer

Kommentarknoten können in XQuery-Ausdrücken mithilfe direkter oder berechneter Konstruktoren erstellt werden.

## Dokumentreihenfolge der Knoten

Alle Knoten in einer Hierarchie entsprechen einer Reihenfolge, die als *Dokumentreihenfolge* bezeichnet wird und in der alle Knoten vor den zugehörigen untergeordneten Elementen dargestellt werden. Die Dokumentreihenfolge entspricht der Reihenfolge, in der die Knoten angezeigt werden, wenn die Knotenhierarchie im serialisierten XML-Format dargestellt wird.

Knoten in einer Hierarchie werden in der folgenden Reihenfolge dargestellt:

- Der Rootknoten ist der erste Knoten.
- Elementknoten werden vor den zugehörigen untergeordneten Elementen dargestellt.
- Attributknoten werden direkt nach dem Elementknoten dargestellt, dem sie zugeordnet sind. Die relative Reihenfolge der Attributknoten ist beliebig, diese Reihenfolge wird jedoch während der Verarbeitung einer Abfrage nicht geändert.
- Die relative Reihenfolge der gleichgeordneten Elemente wird von ihrer Reihenfolge in der Knotenhierarchie bestimmt.
- Untergeordnete Elemente und Nachkommen eines Knotens werden vor den gleichgeordneten Elementen dargestellt, die auf den Knoten folgen.

## Knotenidentität

Jeder Knoten hat eine eindeutige Identität. Zwei Knoten können anhand dieser Identität auch dann unterschieden werden, wenn sie identische Namen und Werte aufweisen. Im Gegensatz hierzu verfügen atomare Werte nicht über eine Identität.

Die Knotenidentität ist nicht gleichbedeutend mit einem Attribut vom Typ ID. Einem Element in einem XML-Dokument kann vom Dokumentautor ein Attribut

vom Typ ID zugeordnet werden. Eine Knotenidentität wird hingegen jedem Knoten vom System automatisch zugeordnet, kann vom Benutzer jedoch nicht direkt angezeigt werden.

Die Knotenidentität wird zur Verarbeitung folgender Ausdruckstypen verwendet:

- Knotenvergleiche. Die Knotenidentität wird vom Operator **is** verwendet, um festzustellen, ob zwei Knoten über dieselbe Identität verfügen.
- Pfadausdrücke. Die Knotenidentität wird von Pfadausdrücken verwendet, um doppelte Knoten zu eliminieren.
- Sequenzausdrücke. Die Knotenidentität wird vom Operator **union**, **intersect** oder **except** verwendet, um doppelte Knoten zu eliminieren.

## Typisierte Werte und Zeichenfolgewerte von Knoten

Jeder Knoten verfügt sowohl über einen *typisierten Wert* als auch über einen *Zeichenfolgewert*. Diese beiden Knotenmerkmale werden in den Definitionen bestimmter XQuery-Operationen (beispielsweise bei der Atomisierung) und -Funktionen (beispielsweise fn:data, fn:string und fn:deep-equal) verwendet.

*Tabelle 1. Zeichenfolgewerte und typisierte Werte von Knoten*

| Knotensorte | Zeichenfolgewert | Typisierter Wert |
|---|---|---|
| Dokument | Eine Instanz des Datentyps xs:string, die den verknüpften Inhalt aller untergeordneten Textknoten in Dokumentreihenfolge enthält. | Eine Instanz des Datentyps xdt:untypedAtomic, die den verknüpften Inhalt aller untergeordneten Textknoten in Dokumentreihenfolge enthält. |
| Element in einem geprüften Dokument | • Wenn dem Elementknoten bei der Gültigkeitsprüfung ein einfacher Datentyp (z. B. xs:decimal) zugeordnet wird oder ein Typ, für den ein einfacher Inhalt definiert ist (z. B. der Typ "temperature", dessen Inhalt 'xs:decimal' lautet), gibt der Zeichenfolgewert die Zeichenfolge an, die den Wert des Elements im ursprünglichen XML-Dokument ausdrückt.<br>• Wenn dem Elementknoten bei der Gültigkeitsprüfung ein Typ zugeordnet wird, bei dem ein gemischter Inhalt (sowohl Textelemente als auch untergeordnete Elemente) zulässig ist, dann stellt der Zeichenfolgewert eine Instanz des Datentyps xs:string dar, der den verknüpften Inhalt aller untergeordneten Textknoten in Dokumentreihenfolge angibt.<br>• Wenn dem Elementknoten bei der Gültigkeitsprüfung ein Typ zugeordnet wird, für den kein Inhalt zulässig ist (weder Textelemente noch untergeordnete Elemente), dann wird als Zeichenfolgewert des Elements eine leere Zeichenfolge angegeben.<br>• Wenn dem Elementknoten bei der Gültigkeitsprüfung ein Typ zugeordnet wird, der vorgibt, dass dieses nur untergeordnete Elemente (keinen Text) enthalten darf, dann besteht der Zeichenfolgewert des Elements aus den verknüpften Zeichenfolgewerten aller untergeordneten Textknoten in Dokumentreihenfolge. | • Wenn dem Elementknoten bei der Gültigkeitsprüfung ein einfacher Datentyp (z. B. xs:decimal) zugeordnet wird oder ein Typ, für den ein einfacher Inhalt definiert ist (z. B. der Typ "temperature", dessen Inhalt 'xs:decimal' lautet), gibt der typisierte Wert das Ergebnis der Umsetzung des Zeichenfolgewertes in den einfachen Typ an, der während der Gültigkeitsprüfung zugeordnet wurde (z. B. xs:decimal).<br>• Wenn dem Elementknoten bei der Gültigkeitsprüfung ein Typ zugeordnet wird, bei dem ein gemischter Inhalt (sowohl Textelemente als auch untergeordnete Elemente) zulässig ist, dann stellt der typisierte Wert eine Instanz des Datentyps xdt:untypedAtomic dar, der den verknüpften Inhalt aller untergeordneten Textknoten in Dokumentreihenfolge angibt.<br>• Wenn dem Elementknoten bei der Gültigkeitsprüfung ein Typ zugeordnet wird, für den kein Inhalt zulässig ist (weder Textelemente noch untergeordnete Elemente), dann wird als typisierter Wert eine leere Sequenz angegeben.<br>• Wenn dem Elementknoten bei der Gültigkeitsprüfung ein Typ zugeordnet wird, der vorgibt, dass dieses nur untergeordnete Elemente (keinen Text) enthalten darf, dann verfügt das Element über keinen typisierten Wert und ein Versuch, den typisierten Wert zu extrahieren (z. B. mit der Funktion fn:data) führt zu einem Fehler. |

*Tabelle 1. Zeichenfolgewerte und typisierte Werte von Knoten (Forts.)*

| Knotensorte | Zeichenfolgewert | Typisierter Wert |
|---|---|---|
| Element in einem nicht geprüften Dokument | Eine Instanz des Datentyps xs:string, die den verknüpften Inhalt aller untergeordneten Textknoten in Dokumentreihenfolge enthält. | Eine Instanz des Datentyps xdt:untypedAtomic, die den verknüpften Inhalt aller untergeordneten Textknoten in Dokumentreihenfolge enthält. |
| Attribut in einem geprüften Dokument | Eine Instanz des Datentyps xs:string, die den Attributwert im ursprünglichen XML-Dokument darstellt. | Das Ergebnis der Umsetzung des Zeichenfolgewerts in den Typ, der dem Attribut während der Gültigkeitsprüfung zugeordnet wurde. Wenn für ein Attribut z. B. bei der Gültigkeitsprüfung der Typ 'xs:decimal' festgestellt wurde, dann kann als Zeichenfolgewert beispielsweise "74,8" und als typisierter Wert beispielweise 74,8 als Dezimalzahl ermittelt werden. |
| Attribut in einem nicht geprüften Dokument | Eine Instanz des Datentyps xs:string, die den Attributwert im ursprünglichen XML-Dokument darstellt. | Eine Instanz des Datentyps xdt:untypedAtomic, die den Attributwert im ursprünglichen XML-Dokument darstellt. |
| Text | Der Inhalt als Instanz des Datentyps xs:string. | Der Inhalt als Instanz des Datentyps xdt:untypedAtomic. |
| Kommentar | Der Inhalt als Instanz des Datentyps xs:string. | Der Inhalt als Instanz des Datentyps xs:string. |
| Verarbeitungsanweisung | Der Inhalt als Instanz des Datentyps xs:string. | Der Inhalt als Instanz des Datentyps xs:string. |

## Serialisierung des XQuery- und XPath-Datenmodells (XDM)

Das Ergebnis eines XQuery-Ausdrucks, das eine XDM-Instanz ist, kann mithilfe eines als *Serialisierung* bezeichneten Prozesses in eine XML-Darstellung umgesetzt werden.

Bei der Serialisierung wird die Sequenz aus Knoten und atomaren Werten (die XDM-Instanz) in eine XML-Darstellung umgesetzt. Das Ergebnis der Serialisierung ist nicht immer ein korrekt formatiertes Dokument. Tatsächlich kann die Serialisierung in einem einzelnen atomaren Wert (beispielsweise 17) oder in einer Sequenz aus Elementen ohne ein gemeinsames übergeordnetes Element resultieren.

XQuery stellt keine Funktion zum Serialisieren des XDMs bereit. Die Art der Serialisierung des XDMs zur Umsetzung in XML-Daten hängt von der Umgebung ab, in der die Abfrage ausgeführt wird. Der Befehlszeilenprozessor beispielsweise gibt eine Sequenz aus serialisierten Elementen zurück, in der jedes Element jeweils als eine Zeile im Ergebnis zurückgegeben wird. Beispiel: Bei Eingabe der Abfrage `XQUERY (1, 2, 3)` über den Befehlszeilenprozessor wird das folgende Ergebnis zurückgegeben:

```
1
2
3
```

Eine Serialisierung kann auch mithilfe der SQL/XML-Funktion XMLSERIALIZE durchgeführt werden.

# XML-Namensbereiche und qualifizierte Namen (QNames)

XML-Namensbereiche verhindern Kollisionen bei der Benennung. Ein *XML-Namensbereich* ist eine Gruppe von Namen, die durch eine Namensbereichs-URI identifiziert wird. Namensbereiche sind eine Möglichkeit zur Qualifizierung von Namen, die für Elemente, Attribute, Datentypen und Funktionen in XQuery verwendet werden. Ein Name, der durch ein Namensbereichspräfix qualifiziert ist, wird als *qualifizierter Name* (*QName*) bezeichnet.

## Qualifizierte Namen (QNames)

Ein *QName* besteht aus einem optionalen Namensbereichspräfix und einem lokalen Namen. Das Namensbereichspräfix und der lokale Name werden durch einen Doppelpunkt voneinander getrennt. Das Namensbereichspräfix (sofern vorhanden) ist an eine URI (Universal Resource Identifier) gebunden und stellt eine Kurzform der URI dar.

Während der Abfrageverarbeitung erweitert XQuery den qualifizierten Namen (QName) und löst die URI auf, die an das Namensbereichspräfix gebunden ist. Der erweiterte QName umfasst die Namensbereichs-URI und einen lokalen Namen. Zwei qualifizierte Namen sind identisch, wenn sie dieselbe Namensbereichs-URI und denselben lokalen Namen aufweisen. Dies bedeutet, dass zwei qualifizierte Namen einander selbst dann entsprechen können, wenn sie unterschiedliche Präfixe aufweisen. Voraussetzung hierfür ist, dass die betreffenden Präfixe an dieselbe Namensbereichs-URI gebunden sind.

Das nachstehende Beispiel enthält folgende qualifizierte Namen:

- `ns1:name`
- `ns2:name`
- `name`

In diesem Beispiel ist `ns1` ein Präfix, das an die URI `http://posample.org` gebunden ist. Das Präfix `ns2` ist an die URI `http://mycompany.com` gebunden. Der Standardnamensbereich für Elemente ist eine andere URI als die URIs, die den Präfixen ns1 und ns2 zugeordnet sind. Der lokale Name aller drei Elemente ist `name`.

```
<ns1:name>Dieser Text ist ein Element
namens "name" das durch das Präfix "ns1" qualifiziert ist.</ns1:name>
<ns2:name>Dieser Text ist ein Element
namens "name" das durch das Präfix "ns2" qualifiziert ist.</ns2:name>
<name>Dieser Text ist ein Element
namens "name", das sich im Standardnamensbereich für Elemente befindet.</name>
```

Die Elemente in diesem Beispiel verwenden zwar gemeinsam denselben lokalen Namen (`name`), doch kommt es zu keiner Namensunverträglichkeiten, da sich die Elemente in unterschiedlichen Namensbereichen befinden. Während der Ausdrucksverarbeitung wird der Name `ns1:name` in einen Namen erweitert, der die an `ns1` gebundene URI sowie den lokalen Namen (`name`) umfasst. Ebenso wird der Name `ns2:name` in einen Namen erweitert, der die an `ns2` gebundene URI sowie den lokalen Namen (`name`) umfasst. Das Element `name`, das ein leeres Präfix aufweist, ist an den Standardnamensbereich für Elemente gebunden, da kein Präfix angegeben ist. Wenn für einen Namen ein Präfix verwendet wird, das nicht an eine URI gebunden ist, wird ein Fehler zurückgegeben.

Qualifizierte Namen (QNames) unterliegen der in der W3C-Empfehlung *Namespaces in XML* definierten Syntax.

## Statisch bekannte Namensbereiche

Namensbereichspräfixe werden mithilfe von Namensbereichsdeklarationen an URIs gebunden. Die Gruppe der Namensbereichsbindungen, die die Interpretation qualifizierter Namen (QNames) in einem Abfrageausdruck steuert, wird als statisch bekannte Namensbereiche bezeichnet. Statisch bekannte Namensbereiche sind Merkmale eines Abfrageausdrucks und hängen von den vom jeweiligen Ausdruck verarbeiteten Daten ab.

Einige Namensbereichspräfixe sind bereits vorab deklariert; andere können mithilfe von Deklarationen im Abfrageprolog oder in einem Elementkonstruktor hinzugefügt werden. DB2 XQuery umfasst die in folgender Tabelle beschriebenen vordeklarierten Namensbereichspräfixe.

*Tabelle 2. Vordeklarierte Namensbereiche in DB2 XQuery*

| Präfix | URI | Beschreibung |
|---|---|---|
| xml | `http://www.w3.org/XML/1998/namespace` | Für XML reservierter Namensbereich |
| xs | `http://www.w3.org/2001/XMLSchema` | Namensbereich des XML-Schemas |
| xsi | `http://www.w3.org/2001/XMLSchema-instance` | Namensbereich der XML-Schemainstanz |
| fn | `http://www.w3.org/2005/xpath-functions` | Standardfunktionsnamensbereich |
| xdt | `http://www.w3.org/2005/xpath-datatypes` | Namensbereich für XQuery-Typen |
| db2-fn | `http://www.ibm.com/xmlns/prod/db2/functions` | DB2-Funktionsnamensbereich |

Zusätzlich zu den vordeklarierten Namensbereichen kann mit einer der folgenden Methoden eine Gruppe statisch bekannter Namensbereiche bereitgestellt werden:

- Im Abfrageprolog mithilfe einer Namensbereichsdeklaration oder einer Standardnamensbereichsdeklaration deklariert. Das folgende Beispiel einer Namensbereichsdeklaration ordnet das Namensbereichspräfix 'ns1' der URI 'http://mycompany.com' zu:

  ```
  declare namespace ns1 = "http://mycompany.com";
  ```

  Das folgende Beispiel einer Standardnamensbereichsdeklaration für Elemente/Typen setzt die URI für Elementnamen in der Abfrage, die keine Präfixe aufweisen:

  ```
  declare default element namespace "http://posample.org";
  ```

- Deklariert durch ein Deklarationsattribut für einen Namensbereich in einem Elementkonstruktor. Das folgende Beispiel ist ein Elementkonstruktor, der ein Deklarationsattribut für einen Namensbereich enthält, das das Präfix 'ns2' an die URI 'http://mycompany.com' im Geltungsbereich des erstellten Elements bindet:

  ```
  <ns2:price xmlns:ns2="http://mycompany.com">14.99</ns2:price>
  ```

- Von SQL/XML bereitgestellt. SQL/XML kann die folgende Gruppe von Namensbereichen bereitstellen:
  - Vordeklarierte SQL/XML-Namensbereiche
  - Namensbereiche, die innerhalb von SQL/XML-Konstruktoren und anderen SQL/XML-Ausdrücken deklariert sind.

  Namensbereiche, die von SQL/XML bereitgestellt werden, können durch Namensbereichsdeklarationen im Prolog bzw. durch nachfolgende Deklarations-

attribute in Elementkonstruktoren überschrieben werden. Namensbereiche, die im Prolog deklariert werden, können von Deklarationsattributen für Namensbereiche in Elementkonstruktoren überschrieben werden.

# Sprachkonventionen

Die XQuery-Sprachkonventionen werden in den folgenden Abschnitten beschrieben.

## Groß-/Kleinschreibung

XQuery ist eine von der Groß-/Kleinschreibung abhängige Sprache.

Schlüsselwörter in XQuery werden in Kleinbuchstaben angegeben und sind nicht reserviert. Namen in XQuery-Ausdrücken können mit Schlüsselwörtern der Sprache übereinstimmen.

## Leerzeichen

Leerzeichen sind in den meisten XQuery-Ausdrücken zulässig, um die Lesbarkeit zu verbessern, auch wenn Leerzeichen nicht zur Syntax des Ausdrucks gehören. Zur Zeichenklasse der Leerzeichen gehören einzelne Leerzeichen (X'20'), Rücklaufzeichen (X'0D'), Zeilenvorschubzeichen (X'0A') und Tabulatorzeichen (X'09').

Im Allgemeinen haben Leerzeichen in einer Abfrage keine Bedeutung. Hiervon ausgenommen sind folgende Situationen, in denen Leerzeichen beibehalten werden:

- Die Leerzeichen befinden sich in einem Zeichenfolgeliteral.
- Die Leerzeichen verdeutlichen einen Ausdruck, indem sie verhindern, dass der Parser zwei benachbarte Token als einen erkennt.
- Die Leerzeichen befinden sich in einem Elementkonstruktor. Die Deklaration 'boundary-space' im Prolog legt fest, ob Leerzeichen in Elementkonstruktoren beibehalten oder verworfen werden sollen.

Die folgenden Ausdrücke beispielsweise machen Leerzeichen zur Verdeutlichung erforderlich:

- `name- name` führt zu einem Fehler. Der Parser erkennt 'name-' als einzelnen qualifizierten Namen (QName) und gibt einen Fehler zurück, wenn kein Operator gefunden wird.
- `name -name` verursacht keinen Fehler. Der Parser erkennt den ersten Namen (`name`) als einen qualifizierten Namen, das Minuszeichen (-) als Operator und den zweiten Namen (`name`) als einen weiteren qualifizierten Namen.
- `name-name` verursacht auch keinen Fehler. Der Ausdruck wird syntaktisch jedoch als einzelner qualifizierter Name analysiert, da ein Bindestrich (-) ein gültiges Zeichen in einem qualifizierten Namen ist.
- Die folgenden beiden Ausdrücke resultieren in einem Fehler:
  - `10 div3`
  - `10div3`

  In diesen Ausdrücken sind Leerzeichen erforderlich, damit der Parser die einzelnen Token separat erkennt.

## Kommentare

Kommentare sind sowohl im Prolog als auch im Abfragehauptteil zulässig. Sie haben keinen Einfluss auf die Abfrageverarbeitung.

Ein Kommentar besteht aus einer durch die Symbole '(:' und ':)' begrenzten Zeichenfolge. Das folgende Beispiel zeigt einen Kommentar in XQuery:

```
(: Ein Kommentar. Sie können Kommentare verwenden, um Ihren Code leichter
verständlich zu machen. :)
```

Die folgenden allgemeinen Regeln gelten für die Verwendung von Kommentaren in DB2 XQuery:

- Kommentare können überall dort eingesetzt werden, wo nicht relevante Leerzeichen zulässig sind. *Nicht relevante Leerzeichen* sind Leerzeichen, die für die Ergebnisse von Ausdrücken ohne Bedeutung sind.
- Kommentare im Konstruktorinhalt sind unzulässig.
- Kommentare können ineinander verschachtelt werden. Hierbei muss jedoch jeder verschachtelte Kommentar über einen einleitenden und abschließenden Begrenzer verfügen: '(:' und ':)'.

Die folgenden Beispiele zeigen sowohl gültige Kommentare als auch Kommentare, die zu einem Fehler führen:

- `(: Ist dies ein Kommentar? ::)` ist ein gültiger Kommentar.
- `(: Ist dies ein Kommentar? ::) oder ein Fehler? :)` führt zu einem Fehler, weil die einleitenden und abschließenden Begrenzer '(:' und ':)' bei der Verschachtelung unpaarig sind und daher nicht den Regeln entsprechen.
- `(: Einen (: Kommentar :) auf Kommentar zu setzen, ist möglicherweise kompliziert, aber häufig nützlich :)` ist ein gültiger Kommentar, weil die paarige Verschachtelung der Kommentare hier den Regeln entspricht und daher zulässig ist.
- `"Dies ist nur eine Zeichenfolge :)"` ist ein gültiger Ausdruck.
- `(: "Dies ist nur eine Zeichenfolge :)" :)` führt zu einem Fehler. Der Ausdruck `"Dies ist eine weitere Zeichenfolge (:"` ist ebenfalls gültig, während der Ausdruck `(: "Dies ist eine weitere Zeichenfolge (:" :)` zu einem Fehler führt. Literale im Inhalt können zu einer unpaarigen Verschachtelung von Kommentaren führen.

## Weitere Informationen zu XQuery

Die folgenden Ressourcen enthalten weitere Informationen zu den Spezifikationen, auf denen DB2 XQuery basiert.

- **XQuery 1.0**

  World Wide Web Consortium. *XQuery 1.0: An XML Query Language*. W3C Recommendation vom 23. Januar 2007. Siehe www.w3.org/TR/2007/REC-xquery-20070123/.
- **XQuery 1.0 and XPath 2.0 Functions and Operators**

  World Wide Web Consortium. *XQuery 1.0 and XPath 2.0 Functions and Operators*. W3C Recommendation vom 23. Januar 2007. Siehe www.w3.org/TR/2007/REC-xpath-functions-20070123/.
- **XQuery 1.0 and XPath 2.0 Data Model**

  World Wide Web Consortium. *XQuery 1.0 and XPath 2.0 Data Model*. W3C Recommendation vom 23. Januar 2007. Siehe www.w3.org/TR/2007/REC-xpath-datamodel-20070123/.
- **XML Query Use Cases**

  World Wide Web Consortium. *XML Query Use Cases*. W3C Working Group Note vom 23. März 2007. Siehe www.w3.org/TR/2007/NOTE-xquery-use-cases-20070323/.

- **XML Schema**

  World Wide Web Consortium. *XML Schema, Parts 0, 1, and 2*. W3C Recommendation vom 28. Oktober 2004. Siehe www.w3.org/TR/2004/REC-xmlschema-0-20041028/, www.w3.org/TR/2004/REC-xmlschema-1-20041028/, and www.w3.org/TR/2004/REC-xmlschema-2-20041028/.
- **XML Names**

  World Wide Web Consortium. *Namespaces in XML 1.0 (Second Edition)*. W3C Recommendation vom 16. August 2006. Siehe www.w3.org/TR/2006/REC-xml-names-20060816/.
- **Updating XML**

  World Wide Web Consortium. *XQuery Update Facility*. W3C Working Draft vom 11. Juli 2006. Siehe www.w3.org/TR/2006/WD-xqupdate-20060711/.

# Kapitel 2. Typsystem

XQuery ist eine stark typisierte Sprache, in der die Operanden der verschiedenen Ausdrücke, Operatoren und Funktionen den jeweils erwarteten Typen entsprechen müssen. Das Typsystem für DB2 XQuery umfasst die integrierten Typen des XML-Schemas und die vordefinierten Typen von XQuery.

Die integrierten Typen des XML-Schemas befinden sich im Namensbereich `http://www.w3.org/2001/XMLSchema`, der das vordeklarierte Namensbereichspräfix `xs` besitzt. Zu den integrierten Schematypen gehören u. a. beispielsweise 'xs:integer', 'xs:string' und 'xs:date'.

Die vordefinierten Typen von XQuery befinden sich im Namensbereich `http://www.w3.org/2005/xpath-datatypes`, der das vordeklarierte Namensbereichspräfix `xdt` besitzt. Zu den vordefinierten Typen von XQuery gehören u. a. beispielsweise 'xdt:untypedAtomic', 'xdt:yearMonthDuration' und 'xdt:dayTimeDuration'.

Jeder Datentyp hat ein lexikalisches Format, bei dem es sich um eine Zeichenfolge handelt, die in den betreffenden Typ umgesetzt werden kann oder die verwendet werden kann, um einen Wert des betreffenden Datentyps nach der Serialisierung darzustellen.

## Die Typhierarchie

Die Typhierarchie von DB2 XQuery bietet eine Übersicht über alle Typen, die in XQuery-Ausdrücken verwendet werden können.

Die Hierarchie in Abb. 3 auf Seite 20 enthält abstrakte Basistypen und abgeleitete Typen. Alle atomaren Typen werden vom Datentyp 'xdt:anyAtomicType' abgeleitet. Durchgezogene Linien verbinden jeden abgeleiteten Datentyp mit dem ihm zugrunde liegenden Basistyp.

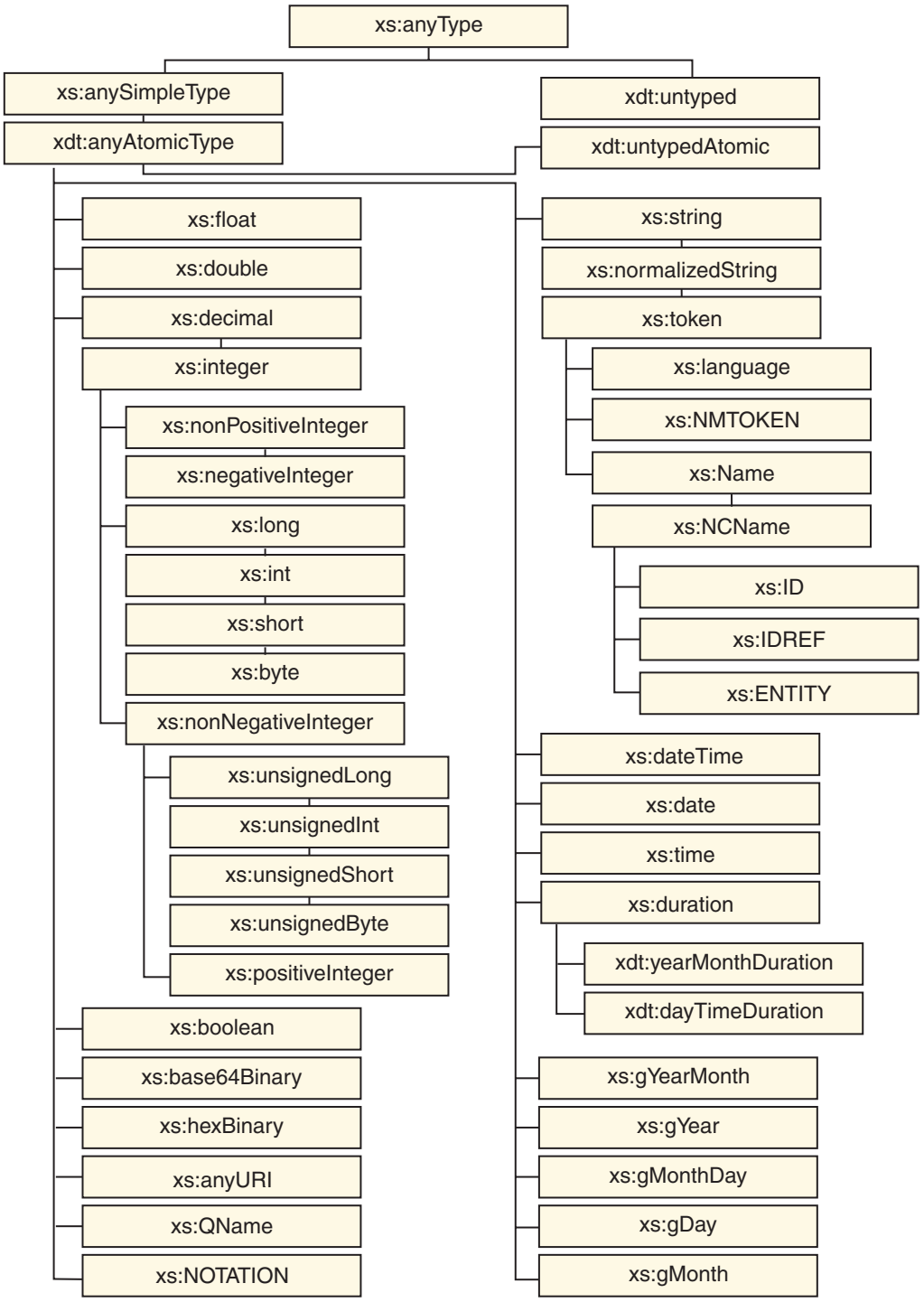



*Abbildung 3. DB2 XQuery-Typhierarchie*

## Datentypen nach Kategorie

DB2 XQuery weist die folgenden Kategorien von Datentypen auf: generische Datentypen, nicht typisierte Datentypen, Zeichenfolgedatentypen, numerische Datentypen, Datentypen für Datumsangaben, Uhrzeit und Dauer sowie sonstige Datentypen.

## Generische Datentypen

*Tabelle 3. Generische Datentypen*

| Typ | Beschreibung |
|---|---|
| „Datentyp 'anyType'" auf Seite 30 | Der Datentyp 'xs:anyType' umfasst eine beliebige Sequenz aus null oder mehr Knoten und null oder mehr atomaren Werten. |
| „Datentyp 'anySimpleType'" auf Seite 30 | Der Datentyp 'xs:anySimpleType' bezeichnet einen Kontext, in dem ein beliebiger einfacher Typ verwendet werden kann. Dieser Datentyp dient als Basistyp für alle einfachen Typen. Eine Instanz eines einfachen Typs kann eine beliebige Sequenz aus atomaren Werten sein. Dieser Datentyp ist vom Datentyp 'xs:anyType' abgeleitet. |
| „Datentyp 'anyAtomicType'" auf Seite 30 | Der Datentyp 'xdt:anyAtomicType' bezeichnet einen Kontext, in dem ein beliebiger atomarer Typ verwendet werden kann. Dieser Datentyp dient als Basistyp für alle atomaren Typen. Eine Instanz eines atomaren Typs ist ein einzelner, nicht zerlegbarer Wert wie beispielsweise eine ganze Zahl, eine Zeichenfolge oder ein Datum. Dieser Datentyp ist vom Datentyp 'xs:anySimpleType' abgeleitet. |

## Nicht typisierte Datentypen

*Tabelle 4. Nicht typisierte Datentypen*

| Typ | Beschreibung |
|---|---|
| „Datentyp 'untyped'" auf Seite 46 | Der Datentyp 'xdt:untyped' bezeichnet einen Knoten, der nicht von einem XML-Schema geprüft worden ist. Dieser Datentyp ist vom Datentyp 'xs:anyType' abgeleitet. |
| „Datentyp 'untypedAtomic'" auf Seite 46 | Der Datentyp 'xdt:untypedAtomic' bezeichnet einen atomaren Wert, der nicht von einem XML-Schema geprüft worden ist. Dieser Datentyp ist vom Datentyp 'xdt:anyAtomicType' abgeleitet. |

## Zeichenfolgedatentypen

*Tabelle 5. Zeichenfolgedatentypen*

| Typ | Beschreibung |
|---|---|
| „Datentyp 'string'" auf Seite 44 | Der Datentyp 'xs:string' stellt eine Zeichenfolge dar. Dieser Datentyp ist vom Datentyp 'xdt:anyAtomicType' abgeleitet. |
| „Datentyp 'normalizedString'" auf Seite 43 | Der Datentyp 'xs:normalizedString' stellt eine um Zeichen aus der Leerzeichenklasse normalisierte Zeichenfolge dar. Dieser Datentyp ist vom Datentyp 'xs:string' abgeleitet. |

*Tabelle 5. Zeichenfolgedatentypen (Forts.)*

| Typ | Beschreibung |
|---|---|
| „Datentyp 'token'" auf Seite 45 | Der Datentyp 'xs:token' stellt eine mit einem Token versehene Zeichenfolge dar. Dieser Datentyp ist vom Datentyp 'xs:normalizedString' abgeleitet. |
| „Datentyp 'language'" auf Seite 41 | Der Datentyp 'xs:language' stellt die Kennung einer natürlichen Sprache gemäß Definition nach *RFC 3066* dar. Dieser Datentyp ist vom Datentyp 'xs:token' abgeleitet. |
| „Datentyp NMTOKEN" auf Seite 42 | Der Datentyp 'xs:NMTOKEN' stellt den Attributtyp NMTOKEN aus *XML 1.0 (Third Edition)* dar. Dieser Datentyp ist vom Datentyp 'xs:token' abgeleitet. |
| „Datentyp 'Name'" auf Seite 41 | Der Datentyp 'xs:Name' stellt einen XML-Namen dar. Dieser Datentyp ist vom Datentyp 'xs:token' abgeleitet. |
| „Datentyp 'NCName'" auf Seite 42 | Der Datentyp 'xs:NCName' stellt einen XML-Namen ohne Doppelpunkte dar. Dieser Datentyp ist vom Datentyp 'xs:Name' abgeleitet. |
| „Datentyp 'ID'" auf Seite 40 | Der Datentyp 'xs:ID' stellt den Attributtyp ID aus *XML 1.0 (Third Edition)* dar. Dieser Datentyp ist vom Datentyp 'xs:NCName' abgeleitet. |
| „Datentyp 'IDREF'" auf Seite 40 | Der Datentyp 'xs:IDREF' stellt den Attributtyp IDREF aus *XML 1.0 (Third Edition)* dar. Dieser Datentyp ist vom Datentyp 'xs:NCName' abgeleitet. |
| „Datentyp ENTITY" auf Seite 37 | Der Datentyp 'xs:ENTITY' stellt den Attributtyp ENTITY aus *XML 1.0 (Third Edition)* dar. Dieser Datentyp ist vom Datentyp 'xs:NCName' abgeleitet. |

## Numerische Datentypen

*Tabelle 6. Numerische Datentypen*

| Typ | Beschreibung |
|---|---|
| „Datentyp 'decimal'" auf Seite 35 | Der Datentyp 'xs:decimal' stellt eine Untergruppe der reellen Zahlen dar, die durch Dezimalnumerale dargestellt werden können. Dieser Datentyp ist vom Datentyp 'xdt:anyAtomicType' abgeleitet. |
| „Datentyp 'float'" auf Seite 37 | Der Datentyp 'xs:float' entspricht dem Muster des IEEE-32-Bit-Gleitkommatyps mit einfacher Genauigkeit. Dieser Datentyp ist vom Datentyp 'xdt:anyAtomicType' abgeleitet. |
| „Datentyp 'double'" auf Seite 35 | Der Datentyp 'xs:double' entspricht dem Muster des IEEE-64-Bit-Gleitkommatyps mit doppelter Genauigkeit. Dieser Datentyp ist vom Datentyp 'xdt:anyAtomicType' abgeleitet. |

*Tabelle 6. Numerische Datentypen (Forts.)*

| Typ | Beschreibung |
|---|---|
| „Datentyp 'int'" auf Seite 41 | Der Datentyp 'xs:int' stellt eine ganze Zahl dar, die kleiner-gleich 2.147.483.647 und größer-gleich -2.147.483.648 ist. Dieser Datentyp ist vom Datentyp 'xs:long' abgeleitet. |
| „Datentyp 'nonPositiveInteger'" auf Seite 42 | Der Datentyp 'xs:nonPositiveInteger' stellt eine ganze Zahl dar, die kleiner-gleich null ist. Dieser Datentyp ist vom Datentyp 'xs:integer' abgeleitet. |
| „Datentyp 'negativeInteger'" auf Seite 42 | Der Datentyp 'xs:negativeInteger' stellt eine ganze Zahl dar, die kleiner als null ist. Dieser Datentyp ist vom Datentyp 'xs:nonPositiveInteger' abgeleitet. |
| „Datentyp 'nonNegativeInteger'" auf Seite 42 | Der Datentyp 'xs:nonNegativeInteger' stellt eine ganze Zahl dar, die größer-gleich null ist. Dieser Datentyp ist vom Datentyp 'xs:integer' abgeleitet. |
| „Datentyp 'long'" auf Seite 41 | Der Datentyp 'xs:long' stellt eine ganze Zahl dar, die kleiner-gleich 9.223.372.036.854.775.807 und größer-gleich -9.223.372.036.854.775 808 ist. Dieser Datentyp ist vom Datentyp 'xs:integer' abgeleitet. |
| „Datentyp 'integer'" auf Seite 41 | Der Datentyp 'xs:integer' stellt eine Zahl dar, die kleiner-gleich 9.223.372.036.854.775.807 und größer-gleich -9.223.372.036.854.775 808 ist. Dieser Datentyp ist vom Datentyp 'xs:decimal' abgeleitet. |
| „Datentyp 'short'" auf Seite 44 | Der Datentyp 'xs:short' stellt eine ganze Zahl dar, die kleiner-gleich 32.767 und größer-gleich -32.768 ist. Dieser Datentyp ist vom Datentyp 'xs:int' abgeleitet. |
| „Datentyp 'byte'" auf Seite 31 | Der Datentyp 'xs:byte' stellt eine ganze Zahl dar, die kleiner-gleich 127 und größer-gleich -128 ist. Dieser Datentyp ist vom Datentyp 'xs:short' abgeleitet. |
| „Datentyp 'unsignedLong'" auf Seite 45 | Der Datentyp 'xs:unsignedLong' stellt eine ganze Zahl ohne Vorzeichen dar, die kleiner-gleich 9.223.372.036.854.775.807 ist. Dieser Datentyp ist vom Datentyp 'xs:nonNegativeInteger' abgeleitet. |
| „Datentyp 'unsignedInt'" auf Seite 45 | Der Datentyp 'xs:unsignedInt' stellt eine ganze Zahl ohne Vorzeichen dar, die kleiner-gleich 4.294.967.295 ist. Dieser Datentyp ist vom Datentyp 'xs:unsignedLong' abgeleitet. |
| „Datentyp 'unsignedShort'" auf Seite 45 | Der Datentyp 'xs:unsignedShort' stellt eine ganze Zahl ohne Vorzeichen dar, die kleiner-gleich 65.535 ist. Dieser Datentyp ist vom Datentyp 'xs:unsignedInt' abgeleitet. |
| „Datentyp 'unsignedByte'" auf Seite 45 | Der Datentyp 'xs:unsignedByte' stellt eine ganze Zahl ohne Vorzeichen dar, die kleiner-gleich 255 ist. Dieser Datentyp ist vom Datentyp 'xs:unsignedShort' abgeleitet. |

*Tabelle 6. Numerische Datentypen (Forts.)*

| Typ | Beschreibung |
|---|---|
| „Datentyp 'positiveInteger'" auf Seite 43 | Der Datentyp 'xs:positiveInteger' stellt eine positive ganze Zahl dar, die größer-gleich 1 ist. Dieser Datentyp ist vom Datentyp 'xs:nonNegativeInteger' abgeleitet. |

## Datentypen für Datumsangaben, Uhrzeit und Dauer

*Tabelle 7. Datentypen für Datumsangaben, Uhrzeit und Dauer*

| Typ | Beschreibung |
|---|---|
| „Datentyp 'duration'" auf Seite 35 | Der Datentyp 'xs:duration' stellt eine Zeitdauer dar, die durch die Gregorianischen Komponenten für Jahr, Monat, Tag, Stunde, Minute und Sekunde ausgedrückt wird. Dieser Datentyp ist vom Datentyp 'xdt:anyAtomicType' abgeleitet. |
| „Datentyp 'yearMonthDuration'" auf Seite 46 | Der Datentyp 'xdt:yearMonthDuration' stellt eine Zeitdauer dar, die durch die Gregorianischen Komponenten für Jahr und Monat ausgedrückt wird. Dieser Datentyp ist vom Datentyp 'xs:duration' abgeleitet. |
| „Datentyp 'dayTimeDuration'" auf Seite 33 | Der Datentyp 'xdt:dayTimeDuration' stellt eine Zeitdauer dar, die durch die Komponenten für Tag, Stunde, Minute und Sekunden ausgedrückt wird. Dieser Datentyp ist vom Datentyp 'xs:duration' abgeleitet. |
| „Datentyp 'dateTime'" auf Seite 32 | Der Datentyp 'xs:dateTime' stellt einen Zeitpunkt mit folgenden Merkmalen dar: den Merkmalen für Jahr, Monat, Tag, Stunde und Minute, die durch ganzzahlige Werte ausgedrückt werden, einem Merkmal für Sekunden, das als Dezimalwert ausgedrückt wird, und einem optionalen Bezugswert für eine Zeitzone. Dieser Datentyp ist vom Datentyp 'xdt:anyAtomicType' abgeleitet. |
| „Datentyp 'date'" auf Seite 31 | Der Datentyp 'xs:date' stellt ein Intervall mit einer Dauer von genau einem Tag dar, das mit dem ersten Moment eines bestimmten Tages beginnt. Der Datentyp 'xs:date' besteht aus den Merkmalen für Jahr, Monat und Tag, die als ganzzahlige Werte ausgedrückt werden, sowie aus einem optionalen Bezugswert für eine Zeitzone. Dieser Datentyp ist vom Datentyp 'xdt:anyAtomicType' abgeleitet. |
| „Datentyp 'time'" auf Seite 44 | Der Datentyp 'xs:time' stellt einen Zeitpunkt dar, der jeden Tag wiederkehrt. Dieser Datentyp ist vom Datentyp 'xdt:anyAtomicType' abgeleitet. |

*Tabelle 7. Datentypen für Datumsangaben, Uhrzeit und Dauer (Forts.)*

| Typ | Beschreibung |
|---|---|
| „Datentyp 'gYearMonth'" auf Seite 40 | Der Datentyp 'xs:gYearMonth' stellt einen bestimmten Gregorianischen Monat in einem bestimmten Gregorianischen Jahr dar. Die Monate und Jahre des Gregorianischen Kalenders sind nach *ISO 8601* definiert. Dieser Datentyp ist vom Datentyp 'xdt:anyAtomicType' abgeleitet. |
| „Datentyp 'gYear'" auf Seite 39 | Der Datentyp 'xs:gYear' stellt ein Jahr aus dem Gregorianischen Kalender dar. Die Jahre des Gregorianischen Kalenders sind nach *ISO 8601* definiert. Dieser Datentyp ist vom Datentyp 'xdt:anyAtomicType' abgeleitet. |
| „Datentyp 'gMonthDay'" auf Seite 39 | Der Datentyp 'xs:gMonthDay' stellt ein wiederkehrendes Gregorianisches Datum dar. Die Daten des Gregorianischen Kalenders sind nach *ISO 8601* definiert. Dieser Datentyp ist vom Datentyp 'xdt:anyAtomicType' abgeleitet. |
| „Datentyp 'gDay'" auf Seite 38 | Der Datentyp 'xs:gDay' stellt einen wiederkehrenden Tag aus dem Gregorianischen Kalender dar. Die Tage des Gregorianischen Kalenders sind nach *ISO 8601* definiert. Dieser Datentyp ist vom Datentyp 'xdt:anyAtomicType' abgeleitet. |
| „Datentyp 'gMonth'" auf Seite 38 | Der Datentyp 'xs:gMonth' stellt einen Gregorianischen Monat dar, der jedes Jahr wiederkehrt. Die Monate des Gregorianischen Kalenders sind nach *ISO 8601* definiert. Dieser Datentyp ist vom Datentyp 'xdt:anyAtomicType' abgeleitet. |

## Sonstige Datentypen

*Tabelle 8. Sonstige Datentypen*

| Typ | Beschreibung |
|---|---|
| „Datentyp 'boolean'" auf Seite 31 | Der Datentyp 'xs:boolean' unterstützt das mathematische Konzept einer binär ausgewerteten Logik mit den Werten 'true' für wahr und 'false' für falsch. Dieser Datentyp ist vom Datentyp 'xdt:anyAtomicType' abgeleitet. |
| „Datentyp 'anyURI'" auf Seite 30 | Der Datentyp 'xs:anyURI' stellt eine URI (Uniform Resource Identifier, einheitliche Ressourcen-ID) dar. Dieser Datentyp ist vom Datentyp 'xdt:anyAtomicType' abgeleitet. |

*Tabelle 8. Sonstige Datentypen (Forts.)*

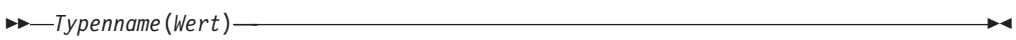
| Typ | Beschreibung |
|---|---|
| „Datentyp 'QName'" auf Seite 43 | Der Datentyp 'xs:QName' stellt einen qualifzierten XML-Namen (QName) dar. Ein QName umfasst ein optionales Namensbereichspräfix, eine URI zur Identifizierung des XML-Namensbereichs und einen lokalen Teil, bei dem es sich um einen Namen ohne Doppelpunkte (NCName) handelt. Dieser Datentyp ist vom Datentyp 'xdt:anyAtomicType' abgeleitet. |
| „Datentyp NOTATION" auf Seite 43 | Der Datentyp 'xs:NOTATION' stellt den Attributtyp NOTATION aus *XML 1.0 (Third Edition)* dar. Dieser Datentyp ist vom Datentyp 'xdt:anyAtomicType' abgeleitet. |
| „Datentyp 'hexBinary'" auf Seite 40 | Der Datentyp 'xs:hexBinary' stellt Binärdaten mit Hexadezimalcodierung dar. Dieser Datentyp ist vom Datentyp 'xdt:anyAtomicType' abgeleitet. |
| „Datentyp 'base64Binary'" auf Seite 30 | Der Datentyp 'xs:base64Binary' stellt Binärdaten mit base64-Codierung dar. Dieser Datentyp ist vom Datentyp 'xdt:anyAtomicType' abgeleitet. |

## Konstruktorfunktionen für integrierte Datentypen

*Konstruktorfunktionen* wandeln eine Instanz eines atomaren Typs in eine Instanz eines anderen atomaren Typs um. Eine implizit definierte Konstruktorfunktion ist für jeden der im XML-Schema definierten integrierten atomaren Typen vorhanden. Konstruktorfunktionen gibt es auch für den Datentyp 'xdt:untypedAtomic' und die beiden abgeleiteten Datentypen 'xdt:yearMonthDuration' und 'xdt:dayTimeDuration'.

Für die Datentypen 'xs:NOTATION', 'xs:anyType', 'xs:anySimpleType' und 'xdt:anyAtomicType' stehen keine Konstruktorfunktionen zur Verfügung.

Alle Konstruktorfunktionen für integrierte Typen verwenden die folgende generische Syntax:




**Anmerkung:** Die Semantik der Konstruktorfunktion *Typenname*(*Wert*) ist gemäß Definition äquivalent zum Ausdruck (*Wert* cast as *Typenname*?).

*Typenname*
Der qualifizierte Name (QName) des Zieldatentyps.

*Wert*
Der Wert, der als eine Instanz vom Zieldatentyp erstellt werden soll. Der Wert wird atomisiert. Ist das Ergebnis der Atomisierung eine leere Sequenz, wird die leere Sequenz zurückgegeben. Ergibt die Atomisierung eine Sequenz mit mehr als einem Element, tritt ein Fehler auf. Andernfalls wird der resultierende atomare Wert in den Zieltyp umgesetzt. Der Abschnitt „Typumsetzung" auf Seite 27 enthält Informationen dazu, welche Typen jeweils in welche anderen Typen umgesetzt werden können.

Das folgende Diagramm beispielsweise stellt die Syntax der Konstruktorfunktion für den Datentyp 'xs:unsignedInt' des XML-Schemas dar:




An diese Konstruktorfunktion kann ein beliebiger atomarer Wert übergeben werden, der ordnungsgemäß in den Zieldatentyp umgesetzt werden kann. Die folgenden Aufrufe dieser Funktion geben dasselbe Ergebnis zurück - den Wert 12 vom Typ 'xs:unsignedInt':

```
xs:unsignedInt(12)
xs:unsignedInt("12")
```

Im ersten Beispiel wird das numerische Literal 12 an die Konstruktorfunktion übergeben. Da das Literal kein Dezimalzeichen enthält, wird es als Datentyp 'xs:integer' verarbeitet, und der Wert vom Typ 'xs:integer' wird in den Typ 'xs:unsignedInt' umgesetzt. Im zweiten Beispiel wird das Zeichenfolgeliteral ″12″ an die Konstruktorfunktion übergeben. Das Zeichenfolgeliteral wird als Datentyp 'xs:string' verarbeitet, und der Wert vom Typ 'xs:string' wird in den Typ 'xs:unsignedInt' umgesetzt.

Eine Konstruktorfunktion kann auch mit einem Knoten als Argument aufgerufen werden. In diesem Fall wird der Knoten von DB2 XQuery zunächst atomisiert, um dessen typisierten Wert zu extrahieren, und anschließend wird der Konstruktor mit dem betreffenden Wert aufgerufen. Wenn der an einen Konstruktor übergebene Wert nicht in den Zieldatentyp umgesetzt werden kann, wird ein Fehler zurückgegeben.

Die Konstruktorfunktion für 'xs:QName' unterscheidet sich dadurch von der generischen Syntax für Konstruktorfunktionen, dass die Konstruktorfunktion auf ein Zeichenfolgeliteral als Argument beschränkt ist.

Beim Umsetzen eines Wertes in einen Datentyp können Sie den Ausdruck 'castable' verwenden, um zu testen, ob der Wert tatsächlich in den betreffenden Datentyp umgesetzt werden kann.

## Typumsetzung

Eine Typumsetzung wird zwischen den Datentypen 'xdt:untypedAtomic', 'xs:integer', den beiden von 'xs:duration' abgeleiteten Typen 'xdt:yearMonthDuration' und 'xdt:dayTimeDuration' sowie den im XML-Schema definierten neunzehn primitiven Typen unterstützt. Typumsetzungen werden in Umsetzungsausdrücken und Typkonstruktoren verwendet.

In den nachstehenden Tabellen werden die unterstützten Typumsetzungen angegeben. Jede Tabelle zeigt links (von oben nach unten) die primitiven Typen, die die Quelle der Typumsetzung darstellen, und oben (von links nach rechts) die primitiven Typen, die das Ziel der Typumsetzung darstellen. Die erste Tabelle enthält die Ziele bei der Umsetzung von 'xdt:untypedAtomic' in 'xs:dateTime', und die zweite Tabelle enthält die Ziele bei der Umsetzung von 'xs:time' in 'xs:NOTATION'.

Die Zellen in den Tabellen enthalten jeweils eines der folgenden drei Zeichen:

**J** Ja. Gibt an, dass eine Umsetzung von Werten des Quellentyps in den Zieltyp unterstützt wird.

**N** Nein. Gibt an, dass eine Umsetzung von Werten des Quellentyps in den Zieltyp nicht unterstützt wird.

**M** Möglicherweise. Gibt an, dass eine Umsetzung von Werten des Quellentyps in den Zieltyp möglicherweise bei einigen Werten funktioniert, aber bei anderen Werten fehlschlägt.

Eine explizite Typumsetzung (Casting) wird von/in 'xs:anySimpleType' bzw. von/in 'xdt:anyAtomicType' nicht unterstützt.

Wird eine nicht unterstützte Umsetzung versucht, wird ein Fehler zurückgegeben.

*Tabelle 9. Umsetzung primitiver Typen, Teil 1 (Ziele bei Umsetzung von 'xdt:untypedAtomic' in 'xs:dateTime')*

| Quellen-datentyp | Zieldatentyp | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **uA** | **string** | **float** | **double** | **decimal** | **integer** | **dur** | **yMD** | **dTD** | **dT** |
| uA | J | J | M | M | M | M | M | M | M | M |
| string | J | J | M | M | M | M | M | M | M | M |
| float | J | J | J | J | M | M | N | N | N | N |
| double | J | J | M | J | M | M | N | N | N | N |
| decimal | J | J | J | J | J | M | N | N | N | N |
| integer | J | J | J | J | J | J | N | N | N | N |
| dur | J | J | N | N | N | N | J | J | J | N |
| yMD | J | J | N | N | N | N | J | J | N | N |
| dTD | J | J | N | N | N | N | J | N | J | N |
| dT | J | J | N | N | N | N | N | N | N | J |
| time | J | J | N | N | N | N | N | N | N | N |
| date | J | J | N | N | N | N | N | N | N | J |
| gYM | J | J | N | N | N | N | N | N | N | N |
| gYr | J | J | N | N | N | N | N | N | N | N |
| gMD | J | J | N | N | N | N | N | N | N | N |
| gDay | J | J | N | N | N | N | N | N | N | N |
| gMon | J | J | N | N | N | N | N | N | N | N |
| bool | J | J | J | J | J | J | N | N | N | N |
| b64 | J | J | N | N | N | N | N | N | N | N |
| hxB | J | J | N | N | N | N | N | N | N | N |
| aURI | J | J | N | N | N | N | N | N | N | N |
| QN | J | J | N | N | N | N | N | N | N | N |
| NOT | J | J | N | N | N | N | N | N | N | N |

*Tabelle 10. Casting primitiver Typen, Teil 2 (Ziele bei Umsetzung von 'xs:time' in 'xs:NOTATION')*

| Quellen-datentyp | Zieldatentyp | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **time** | **date** | **gYM** | **gYr** | **gMD** | **gDay** | **gMon** | **bool** | **b64** | **hxB** | **aURI** | **QN** | **NOT** |
| uA | M | M | M | M | M | M | M | M | M | M | M | N | N |
| string | M | M | M | M | M | M | M | M | M | M | M | M | M |
| float | N | N | N | N | N | N | N | J | N | N | N | N | N |
| double | N | N | N | N | N | N | N | J | N | N | N | N | N |

*Tabelle 10. Casting primitiver Typen, Teil 2 (Ziele bei Umsetzung von 'xs:time' in 'xs:NOTATION') (Forts.)*

| Quellen-datentyp | Zieldatentyp | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **time** | **date** | **gYM** | **gYr** | **gMD** | **gDay** | **gMon** | **bool** | **b64** | **hxB** | **aURI** | **QN** | **NOT** |
| decimal | N | N | N | N | N | N | N | J | N | N | N | N | N |
| integer | N | N | N | N | N | N | N | J | N | N | N | N | N |
| dur | N | N | N | N | N | N | N | N | N | N | N | N | N |
| yMD | N | N | N | N | N | N | N | N | N | N | N | N | N |
| dTD | N | N | N | N | N | N | N | N | N | N | N | N | N |
| dT | J | J | J | J | J | J | J | N | N | N | N | N | N |
| time | J | N | N | N | N | N | N | N | N | N | N | N | N |
| date | N | J | J | J | J | J | J | N | N | N | N | N | N |
| gYM | N | N | J | N | N | N | N | N | N | N | N | N | N |
| gYr | N | N | N | J | N | N | N | N | N | N | N | N | N |
| gMD | N | N | N | N | J | N | N | N | N | N | N | N | N |
| gDay | N | N | N | N | N | J | N | N | N | N | N | N | N |
| gMon | N | N | N | N | N | N | J | N | N | N | N | N | N |
| bool | N | N | N | N | N | N | N | J | N | N | N | N | N |
| b64 | N | N | N | N | N | N | N | N | J | J | N | N | N |
| hxB | N | N | N | N | N | N | N | N | J | J | N | N | N |
| aURI | N | N | N | N | N | N | N | N | N | N | J | N | N |
| QN | N | N | N | N | N | N | N | N | N | N | N | N | N |
| NOT | N | N | N | N | N | N | N | N | N | N | N | N | M |

Die Spalten und Zeilen geben den abgekürzten Code für die folgenden Typen an:

- uA = xdt:untypedAtomic
- string = xs:string
- float = xs:float
- double = xs:double
- decimal = xs:decimal
- integer = xs:integer
- dur = xs:duration
- yMD = xdt:yearMonthDuration
- dTD = xdt:dayTimeDuration
- dT = xs:dateTime
- time = xs:time
- date = xs:date
- gYM = xs:gYearMonth
- gYr = xs:gYear
- gMD = xs:gMonthDay
- gDay = xs:gDay
- gMon = xs:gMonth
- bool = xs:boolean
- b64 = xs:base64Binary

- hxB = xs:hexBinary
- aURI = xs:anyURI
- QN = xs:QName
- NOT = xs:NOTATION

## Datentyp 'anyAtomicType'

Der Datentyp 'xdt:anyAtomicType' bezeichnet einen Kontext, in dem ein beliebiger atomarer Typ verwendet werden kann. Dieser Datentyp dient als Basistyp für alle atomaren Typen. Eine Instanz eines atomaren Typs ist ein einzelner, nicht zerlegbarer Wert wie beispielsweise eine ganze Zahl, eine Zeichenfolge oder ein Datum. Dieser Datentyp ist vom Datentyp 'xs:anySimpleType' abgeleitet.

Der Datentyp 'xdt:anyAtomicType' verfügt über ein uneingeschränktes lexikalisches Format.

Eine explizite Typumwandlung (Casting) in den oder aus dem Datentyp 'xdt:anyAtomicType' wird nicht unterstützt.

## Datentyp 'anySimpleType'

Der Datentyp 'xs:anySimpleType' bezeichnet einen Kontext, in dem ein beliebiger einfacher Typ verwendet werden kann. Dieser Datentyp dient als Basistyp für alle einfachen Typen. Eine Instanz eines einfachen Typs kann eine beliebige Sequenz aus atomaren Werten sein. Dieser Datentyp ist vom Datentyp 'xs:anyType' abgeleitet.

Der Datentyp 'xs:anySimpleType' verfügt über ein uneingeschränktes lexikalisches Format.

Eine explizite Typumwandlung (Casting) in den oder aus dem Datentyp 'xs:anySimpleType' wird nicht unterstützt.

## Datentyp 'anyType'

Der Datentyp 'xs:anyType' umfasst eine beliebige Sequenz aus null oder mehr Knoten und null oder mehr atomaren Werten.

## Datentyp 'anyURI'

Der Datentyp 'xs:anyURI' stellt eine URI (Uniform Resource Identifier, einheitliche Ressourcen-ID) dar. Dieser Datentyp ist vom Datentyp 'xdt:anyAtomicType' abgeleitet.

Das lexikalische Format des Datentyps 'xs:anyURI' ist eine Zeichenfolge, bei der es sich um eine gültige URI entsprechend der Definition von *RFC 2396* und der aktualisierten Fassung von *RFC 2732* handelt. Die Verwendung von Leerzeichen in Werten dieses Typs ist zu vermeiden, sofern diese Leerzeichen nicht mit %20 codiert sind.

## Datentyp 'base64Binary'

Der Datentyp 'xs:base64Binary' stellt Binärdaten mit base64-Codierung dar. Dieser Datentyp ist vom Datentyp 'xdt:anyAtomicType' abgeleitet.

Bei mit base64 codierten Binärdaten wird der gesamte binäre Datenstrom mithilfe des base64-Alphabets codiert. Das base64-Alphabet wird in *RFC 2045* beschrieben.

Das lexikalische Format von 'xs:base64Binary' ist auf die 65 Zeichen des in *RFC 2045* definierten base64-Alphabets beschränkt. Gültige Zeichen sind a-z, A-Z, 0-9, das Pluszeichen (+), der Schrägstrich (/), das Gleichheitszeichen (=) sowie die in *XML 1.0 (Third Edition)* als Leerzeichen definierten Zeichen. Weitere Zeichen sind nicht zulässig.

## Datentyp 'boolean'

Der Datentyp 'xs:boolean' unterstützt das mathematische Konzept einer binär ausgewerteten Logik mit den Werten 'true' für wahr und 'false' für falsch. Dieser Datentyp ist vom Datentyp 'xdt:anyAtomicType' abgeleitet.

Das lexikalische Format des Datentyps 'xs:boolean' ist auf die folgenden Werte beschränkt: true (wahr), false (falsch), 1 und 0.

## Datentyp 'byte'

Der Datentyp 'xs:byte' stellt eine ganze Zahl dar, die kleiner-gleich 127 und größergleich -128 ist. Dieser Datentyp ist vom Datentyp 'xs:short' abgeleitet.

Das lexikalische Format des Datentyps 'xs:byte' ist ein optionales Zeichen, auf das eine Sequenz aus Dezimalziffern mit begrenzter Länge folgt. Wird kein Zeichen angegeben, wird von einem Pluszeichen (+) ausgegangen. Die folgenden Zahlen sind gültige Beispiele dieses Datentyps: `-1, 0, 126 und +100`.

## Datentyp 'date'

Der Datentyp 'xs:date' stellt ein Intervall mit einer Dauer von genau einem Tag dar, das mit dem ersten Moment eines bestimmten Tages beginnt. Der Datentyp 'xs:date' besteht aus den Merkmalen für Jahr, Monat und Tag, die als ganzzahlige Werte ausgedrückt werden, sowie aus einem optionalen Bezugswert für eine Zeitzone. Dieser Datentyp ist vom Datentyp 'xdt:anyAtomicType' abgeleitet.

Bei Werten vom Typ 'xs:date' mit Zeitzone wird der Moment des Tagesbeginns gemäß der entsprechenden Zeitzone aufgezeichnet. Der erste Moment des Tages beginnt um 00:00:00, und der Tag verläuft bis, aber nicht einschließlich, 24:00:00, was wiederum der erste Moment des Folgetags ist. Der erste Moment des Datums 2002-10-10+13:00 beispielsweise ist der Wert 2002-10-10T00:00:00+13:00. Dieser Wert entspricht dem Wert 2002-10-09T11:00:00Z, der auch den ersten Moment von 2002-10-09-11:00 darstellt. Daher geben die Werte 2002-10-10+13:00 und 2002-10-09-11:00 dasselbe Intervall an.

Das lexikalische Format von 'xs:date' ist eine Zeichenfolge mit beschränkter Länge in folgendem Format: *jjjj-mm-ttzzzzzz*. Negative Datumsangaben sind nicht zulässig. Zur Beschreibung dieses Formats werden die folgenden Abkürzungen verwendet:

*jjjj* Ein vierstelliges Numeral zur Darstellung des Jahres. Gültige Werte sind 0001 bis 9999 einschließlich. Ein Pluszeichen (+) ist nicht zulässig.

*mm*
Ein zweistelliges Numeral zur Darstellung des Monats.

*tt* Ein zweistelliges Numeral zur Darstellung des Tages.

*zzzzzz*
Optional. Sofern vorhanden, handelt es sich um die Zeitzone. Der Abschnitt „Zeitzonenbezugswert" auf Seite 33 enthält weitere Informationen zum Format dieses Merkmals.

## Datentyp 'dateTime'

Der Datentyp 'xs:dateTime' stellt einen Zeitpunkt mit folgenden Merkmalen dar: den Merkmalen für Jahr, Monat, Tag, Stunde und Minute, die durch ganzzahlige Werte ausgedrückt werden, einem Merkmal für Sekunden, das als Dezimalwert ausgedrückt wird, und einem optionalen Bezugswert für eine Zeitzone. Dieser Datentyp ist vom Datentyp 'xdt:anyAtomicType' abgeleitet.

Gültige lexikalische Darstellungen von 'xs:dateTime' verfügen nicht unbedingt über eine explizite Zeitzone. Bei Darstellungen ohne explizite Zeitzone wird die implizite Zeitzone UTC (Weltzeit, auch Greenwich Mean Time (GMT) bzw. Westeuropäische Zeit genannt) verwendet. Jedes als numerischer Wert ausgedrückte Merkmal ist auf den Maximalwert in dem Intervall beschränkt, das durch das nächsthöhere Merkmal festgelegt wird. Der Wert für einen Tag beispielsweise kann niemals 32 lauten und nicht einmal 29 bei einem Wert von 02 für den Monat und 2002 für das Jahr (Februar 2002).

Das lexikalische Format von 'xs:dateTime' ist eine Zeichenfolge mit beschränkter Länge in folgendem Format: *jjjj-mm-tt*T*hh*:*mm*:*ss.ssssssszzzzzz*. Negative Datumsangaben sind nicht zulässig. Zur Beschreibung dieses Formats werden die folgenden Abkürzungen verwendet:

*jjjj* Ein vierstelliges Numeral zur Darstellung des Jahres. Gültige Werte sind 0001 bis 9999 einschließlich. Ein Pluszeichen (+) ist nicht zulässig.

**-** Trennzeichen zwischen den Teilen des Datumsabschnitts.

*mm*
Ein zweistelliges Numeral zur Darstellung des Monats.

*tt* Ein zweistelliges Numeral zur Darstellung des Tages.

**T** Ein Trennzeichen, der angibt, dass die Tageszeit folgt.

*hh* Ein zweistelliges Numeral zur Darstellung der Stunde. Der Wert 24 ist nur dann zulässig, wenn die Minuten und Sekunden durch Nullen dargestellt werden. Eine Abfrage mit der Zeitangabe 24:00:00 wird wie die Zeitangabe 00:00:00 des nächsten Tages behandelt.

**:** Ein Trennzeichen zwischen den Teilen des Zeitabschnitts.

*mm*
Ein zweistelliges Numeral zur Darstellung der Minute.

*ss* Ein zweistelliges Numeral zur Darstellung der Sekunde.

**.***ssssss*
Optional. Sofern vorhanden, handelt es sich um ein ein- bis sechsstelliges Numeral zur Darstellung der Sekundenbruchteile.

*zzzzzz*
Optional. Sofern vorhanden, handelt es sich um die Zeitzone. Der Abschnitt „Zeitzonenbezugswert" auf Seite 33 enthält weitere Informationen zum Format dieses Merkmals.

Das folgende Format beispielsweise gibt die Mittagszeit am 10. Oktober 2005 nach Eastern Standard Time in den Vereinigte Staaten an:

```
2005-10-10T12:00:00-05:00
```

Gemäß UTC wäre dies `2002-10-10T17:00:00Z`.

### Zeitzonenbezugswert

Das lexikalische Format für den Zeitzonenbezugswert ist eine Zeichenfolge, die eines der beiden folgenden Formate einschließt:

- Ein Pluszeichen (+) oder Minuszeichen (-), gefolgt von `hh:mm`. Diese Abkürzungen bedeuten Folgendes:

  *hh* Ein zweistelliges Numeral (ggf. mit führenden Nullen) zur Darstellung der Stunden. Derzeit gibt es keine offiziell gültigen Zeitzonen mit einem Intervall von mehr als 24 Stunden. Daher ist der Wert 24 für die Stunden (hh) nur dann zulässig, wenn die Minuten (mm) durch Nullen dargestellt werden.

  *mm*
  Ein zweistelliges Numeral zur Darstellung der Minuten. Der Wert für die Minuten (mm) muss null sein, wenn für die Stunden (hh) der Wert 24 angegeben ist.

  **+** Gibt an, dass sich der angegebene Zeitpunkt in einer Zeitzone befindet, die um *hh* Stunden und *mm* Minuten nach UTC liegt.

  **-** Gibt an, dass sich der angegebene Zeitpunkt in einer Zeitzone befindet, die um *hh* Stunden und *mm* Minuten vor UTC liegt.
- Das Literal `Z`, das die Zeit gemäß UTC darstellt (`Z` steht für Zulu-Zeit, die äquivalent zu UTC ist). Die Angabe von Z zur Darstellung der Zeitzone entspricht der Angabe von `+00:00` oder `-00:00`.

## Datentyp 'dayTimeDuration'

Der Datentyp 'xdt:dayTimeDuration' stellt eine Zeitdauer dar, die durch die Komponenten für Tag, Stunde, Minute und Sekunden ausgedrückt wird. Dieser Datentyp ist vom Datentyp 'xs:duration' abgeleitet.

Der Bereich, der von diesem Datentyp dargestellt werden kann, erstreckt sich von '-P83333333333333Y3M11574074074DT1H46M39.999999S' bis 'P83333333333333Y3M11574074074DT1H46M39.999999S' bzw. von '-999999999999999 Monaten und -999999999999999.999999 Sekunden' bis '999999999999999 Monaten und 999999999999999.999999 Sekunden'.

Das lexikalische Format von 'xdt:dayTimeDuration' lautet `P`*n*`DT`*n*`H`*n*`M`*n*`S`, wobei es sich um eine Abkürzung des *ISO 8601*-Formats handelt. Zur Beschreibung dieses Formats werden die folgenden Abkürzungen verwendet:

**P** Die Bezeichnung für eine Dauer.

*n***D**
*n* ist eine ganze Zahl ohne Vorzeichen zur Darstellung der Anzahl der Tage (D).

**T** Das Datums- und Zeittrennzeichen.

*n***H**
*n* ist eine ganze Zahl ohne Vorzeichen zur Darstellung der Anzahl der Stunden (H).

*n***M**
*n* ist eine ganze Zahl ohne Vorzeichen zur Darstellung der Anzahl der Minuten (M).

*n***S** *n* ist eine ganze Zahl ohne Vorzeichen zur Darstellung der Anzahl der Sekunden (S). Wird ein Dezimalzeichen angezeigt, müssen anschließend ein bis sechs Ziffern folgen, die die Sekundenbruchteile darstellen.

Das folgende Format beispielsweise gibt eine Dauer von 3 Tagen, 10 Stunden und 30 Minuten an:

```
P3DT10H30M
```

Das folgende Format gibt eine Dauer von negativen 120 Tagen an:

```
-P120D
```

Ein optionales führendes Minuszeichen (-) gibt eine negative Dauer an. Wird kein Zeichen angegeben, wird von einer positiven Dauer ausgegangen.

Eine geringere Genauigkeit und abgeschnittene Darstellungen dieses Formats sind zulässig, müssen jedoch die folgenden Anforderungen erfüllen:

- Wenn die Anzahl der Tage, Stunden, Minuten oder Sekunden in einem Ausdruck null entspricht, können die Zahl und die entsprechende Bezeichnung ausgelassen werden. Es muss jedoch mindestens eine Zahl samt Bezeichnung vorhanden sein.
- Der Sekundenteil kann einen Dezimalbruchteil aufweisen.
- Die Bezeichnung `T` darf ausschließlich dann fehlen, wenn sämtliche Zeitelemente fehlen. Die Bezeichnung `P` muss stets vorhanden sein.

So sind beispielsweise die folgenden Formate zulässig:

```
P13D
PT47H
P3DT2H
-PT35.89S
P4DT251M
```

Das Format `P-134D` ist nicht zulässig, wohingegen das Format `-P1347D` gültig ist.

DB2 speichert Werte vom Typ 'xdt:dayTimeDuration' in einem normalisierten Format. Im normalisierten Format sind die Werte für Sekunden und Minuten kleiner als 60, und die Werte für Stunden sind kleiner als 24. Jedes Vielfache von 60 Sekunden wird in eine Minute umgewandelt, jedes Vielfache von 60 Minuten wird in eine Stunde umgewandelt, und jedes Vielfache von 24 Stunden wird in einen Tag umgewandelt. Der folgende XQuery-Ausdruck beispielsweise ruft eine Konstruktorfunktion auf, die einen Wert des Typs 'dayTimeDuration' von 63 Tagen, 55 Stunden und 81 Sekunden angibt:

```
  xquery
xdt:dayTimeDuration("P63DT55H81S")
```

Bei der Angabe der Dauer werden die 55 Stunden in 2 Tage und 7 Stunden und die 81 Sekunden in eine Minute und 21 Sekunden umgewandelt. Demnach gibt der Ausdruck den normalisierten dayTimeDuration-Wert '65DT7H1M21S' zurück.

## Datentyp 'decimal'

Der Datentyp 'xs:decimal' stellt eine Untergruppe der reellen Zahlen dar, die durch Dezimalnumerale dargestellt werden können. Dieser Datentyp ist vom Datentyp 'xdt:anyAtomicType' abgeleitet.

Das lexikalische Format von 'xs:decimal' ist eine Sequenz aus Dezimalziffern mit begrenzter Länge, in der als Dezimalzeichen ein Punkt verwendet wird. Ein optionales Vorzeichen ist zulässig. Wird kein Zeichen angegeben, wird von einem Pluszeichen (+) ausgegangen. Führende und abschließende Nullen sind optional. Ist der Bruchteil der Dezimalziffer null, können das Dezimalzeichen und alle folgenden Nullen ausgelassen werden. Die folgenden Zahlen sind gültige Beispiele dieses Datentyps:

```
-1.23
12678967.543233
+100000.00
210
```

## Datentyp 'double'

Der Datentyp 'xs:double' entspricht dem Muster des IEEE-64-Bit-Gleitkommatyps mit doppelter Genauigkeit. Dieser Datentyp ist vom Datentyp 'xdt:anyAtomicType' abgeleitet.

Der Basiswertebereich von 'xs:double' besteht aus Werten, die von -1.7976931348623158e+308 bis -2.2250738585072014e-308 einschließlich und von +2.2250738585072014e-308 bis +1.7976931348623158e+308 einschließlich reichen. Der Wertebereich von 'xs:double' umfasst darüber hinaus auch folgende Sonderwerte: positive Unendlichkeit, negative Unendlichkeit, positive Null, negative Null und Nichtzahlen (NaN).

Das lexikalische Format von 'xs:double' ist eine Mantisse, optional gefolgt vom Zeichen `E` oder `e`, gefolgt von einem Exponenten. Der Exponent muss eine ganze Zahl (Integer) sein. Die Mantisse muss eine Dezimalzahl sein. Die Darstellungen des Exponenten und der Mantisse müssen den lexikalischen Regeln für die Datentypen 'xs:integer' und 'xs:decimal' folgen. Werden der Buchstabe `E` bzw. `e` und der darauf folgende Exponent ausgelassen, wird von einem Wert 0 für den Exponenten ausgegangen.

Die lexikalischen Formate für null können ein positives oder negatives Vorzeichen aufweisen. Die folgenden Literale sind gültige Beispiele dieses Datentyps: `-1E4`, `1267.43233E12`, `12.78e-2`, `12 `, `-0` und `0`.

Die Sonderwerte 'positive Unendlichkeit', 'negative Unendlichkeit' und Nichtzahlen haben das lexikalische Format INF, -INF bzw. NaN. Das lexikalische Format für 'positive Unendlichkeit' darf kein positives Vorzeichen haben.

**Tipp:** Für die Sonderwerte INF, -INF und NaN gibt es kein Literal. Sie können die Werte INF, -INF und NaN anhand entsprechender Zeichenfolgen mithilfe des Konstruktors des Datentyps 'xs:double' erstellen. Beispiel: `xs:double("INF")`.

## Datentyp 'duration'

Der Datentyp 'xs:duration' stellt eine Zeitdauer dar, die durch die Gregorianischen Komponenten für Jahr, Monat, Tag, Stunde, Minute und Sekunde ausgedrückt wird. Dieser Datentyp ist vom Datentyp 'xdt:anyAtomicType' abgeleitet.

Der Bereich, der von diesem Datentyp dargestellt werden kann, erstreckt sich von '-P83333333333333Y3M11574074074DT1H46M39.999999S' bis 'P83333333333333Y3M11574074074DT1H46M39.999999S' bzw. von '-999999999999999 Monaten und -999999999999999.999999 Sekunden' bis '999999999999999 Monaten und 999999999999999.999999' Sekunden.

Das lexikalische Format von 'xs:duration' entspricht dem erweiterten *ISO 8601*-Format `PnYnMnDTnHnMnS`. Zur Beschreibung des erweiterten Formats werden die folgenden Abkürzungen verwendet:

**P** Die Bezeichnung für eine Dauer.

*n***Y** *n* ist eine ganze Zahl ohne Vorzeichen zur Darstellung der Anzahl der Jahre (Y).

*n***M**
*n* ist eine ganze Zahl ohne Vorzeichen zur Darstellung der Anzahl der Monate (M).

*n***D**
*n* ist eine ganze Zahl ohne Vorzeichen zur Darstellung der Anzahl der Tage (D).

**T** Das Datums- und Zeittrennzeichen.

*n***H**
*n* ist eine ganze Zahl ohne Vorzeichen zur Darstellung der Anzahl der Stunden (H).

*n***M**
*n* ist eine ganze Zahl ohne Vorzeichen zur Darstellung der Anzahl der Minuten (M).

*n***S** *n* ist eine ganze Zahl ohne Vorzeichen zur Darstellung der Anzahl der Sekunden (S). Wird ein Dezimalzeichen angezeigt, müssen anschließend ein bis sechs Ziffern folgen, die die Sekundenbruchteile darstellen.

Das folgende Format beispielsweise gibt eine Dauer von 1 Jahr, 2 Monaten, 3 Tagen, 10 Stunden und 30 Minuten an:

```
P1Y2M3DT10H30M
```

Das folgende Format gibt eine Dauer von negativen 120 Tagen an:

```
-P120D
```

Ein optionales führendes Minuszeichen (-) gibt eine negative Dauer an. Wird kein Zeichen angegeben, wird von einer positiven Dauer ausgegangen.

Eine geringere Genauigkeit und abgeschnittene Darstellungen dieses Formats sind zulässig, müssen jedoch die folgenden Anforderungen erfüllen:

- Wenn die Anzahl der Jahre, Monate Tage, Stunden, Minuten oder Sekunden in einem Ausdruck null entspricht, können die Zahl und die entsprechende Bezeichnung ausgelassen werden. Es muss jedoch mindestens eine Zahl samt Bezeichnung vorhanden sein.
- Der Sekundenteil kann einen Dezimalbruchteil aufweisen.
- Die Bezeichnung `T` darf ausschließlich dann fehlen, wenn sämtliche Zeitelemente fehlen.
- Die Bezeichnung `P` muss stets vorhanden sein.

So sind beispielsweise die folgenden Formate zulässig:

```
P1347Y
P1347M
P1Y2MT2H
P0Y1347M
P0Y1347M0D
```

Das Format `P1Y2MT` ist nicht zulässig, da keine Zeitelemente vorhanden sind. Das Format `P-1347M` ist nicht zulässig, wohingegen das Format `-P1347M` gültig ist.

DB2 speichert Werte vom Typ 'xs:duration' in einem normalisierten Format. Im normalisierten Format sind die Werte für Sekunden und Minuten kleiner als 60, die Werte für Stunden sind kleiner als 24, und die Werte für Monate sind kleiner als 12. Jedes Vielfache von 60 Sekunden wird in eine Minute umgewandelt, jedes Vielfache von 60 Minuten wird in eine Stunde umgewandelt, jedes Vielfache von 24 Stunden wird in einen Tag umgewandelt, und jedes Vielfache von 12 Monaten wird in ein Jahr umgewandelt. Der folgende XQuery-Ausdruck beispielsweise ruft eine Konstruktorfunktion auf, die eine Dauer von 2 Monaten, 63 Tagen, 55 Stunden und 91 Minuten angibt:

```
  xquery
xs:duration("P2M63DT55H91M")
```

Bei der Angabe der Dauer werden die 55 Stunden in 2 Tage und 7 Stunden und die 91 Minuten in eine Stunde und 31 Minuten umgewandelt. Demnach gibt der Ausdruck den normalisierten Wert 'P2M65DT8H31M' für die Dauer zurück.

## Datentyp ENTITY

Der Datentyp 'xs:ENTITY' stellt den Attributtyp ENTITY aus *XML 1.0 (Third Edition)* dar. Dieser Datentyp ist vom Datentyp 'xs:NCName' abgeleitet.

Das lexikalische Format von 'xs:ENTITY' ist ein XML-Name, der keinen Doppelpunkt enthält (NCName).

## Datentyp 'float'

Der Datentyp 'xs:float' entspricht dem Muster des IEEE-32-Bit-Gleitkommatyps mit einfacher Genauigkeit. Dieser Datentyp ist vom Datentyp 'xdt:anyAtomicType' abgeleitet.

Der Basiswertebereich von 'xs:float' besteht aus Werten, die von -3.4028234663852886e+38 bis -1.1754943508222875e-38 einschließlich und von +1.1754943508222875e-38 bis +3.4028234663852886e+38 einschließlich reichen. Der Wertebereich von 'xs:float' umfasst darüber hinaus auch folgende Sonderwerte: positive Unendlichkeit, negative Unendlichkeit, positive Null, negative Null und Nichtzahlen (NaN).

Das lexikalische Format von 'xs:float' ist eine Mantisse, optional gefolgt vom Zeichen `E` oder `e`, gefolgt von einem Exponenten. Der Exponent muss eine ganze Zahl (Integer) sein. Die Mantisse muss eine Dezimalzahl sein. Die Darstellungen des Exponenten und der Mantisse müssen den lexikalischen Regeln für die Datentypen 'xs:integer' und 'xs:decimal' folgen. Werden der Buchstabe `E` bzw. `e` und der darauf folgende Exponent ausgelassen, wird von einem Wert 0 für den Exponenten ausgegangen.

Die lexikalischen Formate für null können ein positives oder negatives Vorzeichen aufweisen. Die folgenden Literale sind gültige Beispiele dieses Datentyps: `-1E4, 1267.43233E12, 12.78e-2, 12 , -0 und 0`.

Die Sonderwerte 'positive Unendlichkeit', 'negative Unendlichkeit' und Nichtzahlen haben das lexikalische Format INF, -INF bzw. NaN. Das lexikalische Format für 'positive Unendlichkeit' darf kein positives Vorzeichen haben.

**Tipp:** Für die Sonderwerte INF, -INF und NaN gibt es kein Literal. Sie können die Werte INF, -INF und NaN anhand entsprechender Zeichenfolgen mithilfe des Konstruktors des Datentyps 'xs:float' erstellen. Beispiel: `xs:float("INF")`.

## Datentyp 'gDay'

Der Datentyp 'xs:gDay' stellt einen wiederkehrenden Tag aus dem Gregorianischen Kalender dar. Die Tage des Gregorianischen Kalenders sind nach *ISO 8601* definiert. Dieser Datentyp ist vom Datentyp 'xdt:anyAtomicType' abgeleitet.

Dieser Datentyp stellt einen bestimmten Tag des Monats dar. So könnte dieser Datentyp beispielsweise verwendet werden um anzugeben, dass der Zahltag jeweils der 15. Tag eines Monats ist.

Das lexikalische Format des Datentyps 'xs:gDay' lautet `---`*`ttzzzzzz`*, wobei es sich um eine abgeschnittene Darstellung des Datentyps 'xs:date' handelt, in der der Monat und das Jahr nicht enthalten sind. Vorzeichen sind unzulässig. Ebenso sind auch keine anderen Formate zulässig. Zur Beschreibung dieses Formats werden die folgenden Abkürzungen verwendet:

*tt* Ein zweistelliges Numeral zur Darstellung des Tages.

*zzzzzz*
Optional. Sofern vorhanden, handelt es sich um die Zeitzone. Der Abschnitt „Zeitzonenbezugswert" auf Seite 33 enthält weitere Informationen zum Format dieses Merkmals.

Das folgende Format beispielsweise gibt den 16. Tag des Monats an - ein Tag, der jeden Monat wiederkehrt:

```
---16
```

## Datentyp 'gMonth'

Der Datentyp 'xs:gMonth' stellt einen Gregorianischen Monat dar, der jedes Jahr wiederkehrt. Die Monate des Gregorianischen Kalenders sind nach *ISO 8601* definiert. Dieser Datentyp ist vom Datentyp 'xdt:anyAtomicType' abgeleitet.

Dieser Datentyp stellt einen bestimmten Monat des Jahres dar. So könnte dieser Datentyp beispielsweise verwendet werden um anzugeben, dass Weihnachten im Monat Dezember gefeiert wird.

Das lexikalische Format des Datentyps 'xs:gMonth' lautet `--`*`mmzzzzzz`*, wobei es sich um eine abgeschnittene Darstellung des Datentyps 'xs:date' handelt, in der das Jahr und der Tag nicht enthalten sind. Vorzeichen sind unzulässig. Ebenso sind keine anderen Formate zulässig. Zur Beschreibung dieses Formats werden die folgenden Abkürzungen verwendet:

*mm*
Ein zweistelliges Numeral zur Darstellung des Monats.

*zzzzzz*
Optional. Sofern vorhanden, handelt es sich um die Zeitzone. Der Abschnitt „Zeitzonenbezugswert" auf Seite 33 enthält weitere Informationen zum Format dieses Merkmals.

Das folgende Format beispielsweise gibt den Monat Dezember an - einen bestimmten Monat, der jedes Jahr wiederkehrt:

```
--12
```

## Datentyp 'gMonthDay'

Der Datentyp 'xs:gMonthDay' stellt ein wiederkehrendes Gregorianisches Datum dar. Die Daten des Gregorianischen Kalenders sind nach *ISO 8601* definiert. Dieser Datentyp ist vom Datentyp 'xdt:anyAtomicType' abgeleitet.

Dieser Datentyp stellt einen bestimmten Tag des Jahres dar. So könnte dieser Datentyp beispielsweise verwendet werden, um einen Geburtstag anzugeben, der jedes Jahr am 16. April wiederkehrt.

Das lexikalische Format des Datentyps 'xs:gMonthDay' lautet *--mm-ttzzzzzz*, wobei es sich um eine abgeschnittene Darstellung des Datentyps 'xs:date' handelt, in der das Jahr nicht enthalten ist. Vorzeichen sind unzulässig. Ebenso sind keine anderen Formate zulässig. Zur Beschreibung dieses Formats werden die folgenden Abkürzungen verwendet:

*mm*
: Ein zweistelliges Numeral zur Darstellung des Monats.

*tt*
: Ein zweistelliges Numeral zur Darstellung des Tages.

*zzzzzz*
: Optional. Sofern vorhanden, handelt es sich um die Zeitzone. Der Abschnitt „Zeitzonenbezugswert" auf Seite 33 enthält weitere Informationen zum Format dieses Merkmals.

Das folgende Format beispielsweise gibt den 16. April an - einen bestimmten Tag, der jedes Jahr wiederkehrt:

```
--04-16
```

## Datentyp 'gYear'

Der Datentyp 'xs:gYear' stellt ein Jahr aus dem Gregorianischen Kalender dar. Die Jahre des Gregorianischen Kalenders sind nach *ISO 8601* definiert. Dieser Datentyp ist vom Datentyp 'xdt:anyAtomicType' abgeleitet.

Das lexikalische Format des Datentyps 'xs:gYear' lautet *jjjjzzzzzz*. Dieses Format ist eine abgeschnittene Darstellung des Datentyps 'xs:dateTime', in der der Monat, der Tag und die Uhrzeit nicht enthalten sind. Negative Datumsangaben sind nicht zulässig. Zur Beschreibung dieses Formats werden die folgenden Abkürzungen verwendet:

*jjjj* Ein vierstelliges Numeral zur Darstellung des Jahres. Gültige Werte sind 0001 bis 9999 einschließlich. Ein Pluszeichen (+) ist nicht zulässig.

*zzzzzz*
: Optional. Sofern vorhanden, handelt es sich um die Zeitzone. Der Abschnitt „Zeitzonenbezugswert" auf Seite 33 enthält weitere Informationen zum Format dieses Merkmals.

Das folgende Format beispielsweise stellt das Gregorianische Jahr 2005 dar: `2005`.

## Datentyp 'gYearMonth'

Der Datentyp 'xs:gYearMonth' stellt einen bestimmten Gregorianischen Monat in einem bestimmten Gregorianischen Jahr dar. Die Monate und Jahre des Gregorianischen Kalenders sind nach *ISO 8601* definiert. Dieser Datentyp ist vom Datentyp 'xdt:anyAtomicType' abgeleitet.

Das lexikalische Format des Datentyps 'xs:gYearMonth' lautet *jjjj-mmzzzzzz*. Dieses Format ist eine abgeschnittene Darstellung des Datentyps 'xs:dateTime', in der die Uhrzeit nicht enthalten ist. Negative Datumsangaben sind nicht zulässig. Zur Beschreibung dieses Formats werden die folgenden Abkürzungen verwendet:

*jjjj* Ein vierstelliges Numeral zur Darstellung des Jahres. Gültige Werte sind 0001 bis 9999 einschließlich. Ein Pluszeichen (+) ist nicht zulässig.

*mm*
Ein zweistelliges Numeral zur Darstellung des Monats.

*zzzzzz*
Optional. Sofern vorhanden, handelt es sich um die Zeitzone. Der Abschnitt „Zeitzonenbezugswert" auf Seite 33 enthält weitere Informationen zum Format dieses Merkmals.

Das folgende Format beispielsweise, das keinen optionalen Bezugswert für die Zeitzone enthält, gibt den Monat Oktober im Jahr 2005 an:

```
2005-10
```

## Datentyp 'hexBinary'

Der Datentyp 'xs:hexBinary' stellt Binärdaten mit Hexadezimalcodierung dar. Dieser Datentyp ist vom Datentyp 'xdt:anyAtomicType' abgeleitet.

Das lexikalische Format des Datentyps 'xs:hexBinary' ist eine Sequenz aus Zeichen, in der jedes binäre Oktett durch zwei Hexadezimalziffern dargestellt wird. Das folgende Format beispielsweise ist eine hexadezimale Verschlüsselung für die 16-Bit-Ganzzahl 4023, deren binäre Darstellung '111110110111' ist: `0FB7`.

## Datentyp 'ID'

Der Datentyp 'xs:ID' stellt den Attributtyp ID aus *XML 1.0 (Third Edition)* dar. Dieser Datentyp ist vom Datentyp 'xs:NCName' abgeleitet.

Das lexikalische Format des Datentyps 'xs:ID' ist ein XML-Name, der keinen Doppelpunkt enthält (NCName).

## Datentyp 'IDREF'

Der Datentyp 'xs:IDREF' stellt den Attributtyp IDREF aus *XML 1.0 (Third Edition)* dar. Dieser Datentyp ist vom Datentyp 'xs:NCName' abgeleitet.

Das lexikalische Format des Datentyps 'xs:IDREF' ist ein XML-Name, der keinen Doppelpunkt enthält (NCName).

## Datentyp 'int'

Der Datentyp 'xs:int' stellt eine ganze Zahl dar, die kleiner-gleich 2.147.483.647 und größer-gleich -2.147.483.648 ist. Dieser Datentyp ist vom Datentyp 'xs:long' abgeleitet.

Das lexikalische Format des Datentyps 'xs:int' ist ein optionales Zeichen, auf das eine Sequenz aus Dezimalziffern mit begrenzter Länge folgt. Wird kein Zeichen angegeben, wird von einem Pluszeichen (+) ausgegangen. Die folgenden Zahlen sind gültige Beispiele dieses Datentyps: `-1, 0, 126789675 und +100000`.

## Datentyp 'integer'

Der Datentyp 'xs:integer' stellt eine Zahl dar, die kleiner-gleich 9.223.372.036.854.775.807 und größer-gleich -9.223.372.036.854.775 808 ist. Dieser Datentyp ist vom Datentyp 'xs:decimal' abgeleitet.

Das lexikalische Format des Datentyps 'xs:integer' ist eine Sequenz aus Dezimalziffern mit begrenzter Länge mit optionalem Vorzeichen. Wird kein Zeichen angegeben, wird von einem Pluszeichen (+) ausgegangen. Die folgenden Zahlen sind gültige Beispiele dieses Datentyps: `-1, 0, 12678967543233 und +100000`.

## Datentyp 'language'

Der Datentyp 'xs:language' stellt die Kennung einer natürlichen Sprache gemäß Definition nach *RFC 3066* dar. Dieser Datentyp ist vom Datentyp 'xs:token' abgeleitet.

Das lexikalische Format des Datentyps 'xs:language' besteht aus Zeichenfolgen von Markierungen, die durch Bindestriche miteinander verbunden sind. Jede Markierung enthält höchstens acht Zeichen. Die erste Markierung darf nur alphabetische Zeichen enthalten, während anschließend folgende Markierungen sowohl alphabetische als auch numerische Zeichen enthalten können. Der Wert 'en-US' beispielsweise steht für US-amerikanisches Englisch. Die Zeichenfolge entspricht dem Muster `[a-zA-Z]{1,8}(-[a-zA-Z0-9]{1,8})*`.

## Datentyp 'long'

Der Datentyp 'xs:long' stellt eine ganze Zahl dar, die kleiner-gleich 9.223.372.036.854.775.807 und größer-gleich -9.223.372.036.854.775 808 ist. Dieser Datentyp ist vom Datentyp 'xs:integer' abgeleitet.

Das lexikalische Format des Datentyps 'xs:long' ist ein optionales Zeichen, auf das eine Sequenz aus Dezimalziffern mit begrenzter Länge folgt. Wird kein Zeichen angegeben, wird von einem Pluszeichen (+) ausgegangen. Die folgenden Zahlen sind gültige Beispiele dieses Datentyps: `-1, 0, 12678967543233 und +100000`.

## Datentyp 'Name'

Der Datentyp 'xs:Name' stellt einen XML-Namen dar. Dieser Datentyp ist vom Datentyp 'xs:token' abgeleitet.

Das lexikalische Format des Datentyps 'xs:Name' ist eine Zeichenfolge, die dem gemäß *XML 1.0 (Third Edition)* erstellten Namen entspricht.

## Datentyp 'NCName'

Der Datentyp 'xs:NCName' stellt einen XML-Namen ohne Doppelpunkte dar. Dieser Datentyp ist vom Datentyp 'xs:Name' abgeleitet.

Das lexikalische Format des Datentyps 'xs:NCName' ist ein XML-Name, der keinen Doppelpunkt enthält.

## Datentyp 'negativeInteger'

Der Datentyp 'xs:negativeInteger' stellt eine ganze Zahl dar, die kleiner als null ist. Dieser Datentyp ist vom Datentyp 'xs:nonPositiveInteger' abgeleitet.

Das lexikalische Format des Datentyps 'xs:negativeInteger' ist ein negatives Vorzeichen (-), auf das eine Sequenz aus Dezimalziffern mit begrenzter Länge folgt. Der Bereich, der von diesem Datentyp dargestellt werden kann, erstreckt sich von -9223372036854775808 bis -1. Die folgenden Zahlen sind gültige Beispiele dieses Datentyps: `-1, -12678967543233 und -100000`.

## Datentyp NMTOKEN

Der Datentyp 'xs:NMTOKEN' stellt den Attributtyp NMTOKEN aus *XML 1.0 (Third Edition)* dar. Dieser Datentyp ist vom Datentyp 'xs:token' abgeleitet.

Das lexikalische Format des Datentyps 'xs:NMTOKEN' ist eine Zeichenfolge, die dem gemäß *XML 1.0 (Third Edition)* erstellten Attributtyp 'Nmtoken' entspricht.

## Datentyp 'nonNegativeInteger'

Der Datentyp 'xs:nonNegativeInteger' stellt eine ganze Zahl dar, die größer-gleich null ist. Dieser Datentyp ist vom Datentyp 'xs:integer' abgeleitet.

Das lexikalische Format des Datentyps 'xs:nonNegativeInteger' ist ein optionales Vorzeichen, auf das eine Sequenz aus Dezimalziffern mit begrenzter Länge folgt. Wird kein Vorzeichen angegeben, wird von einem Pluszeichen (+) ausgegangen. Bei lexikalischen Formaten, die null angeben, kann ein positives (+) oder negatives (-) Vorzeichen verwendet werden. Bei allen anderen lexikalischen Formaten darf ausschließlich ein positives (+) Vorzeichen vorhanden sein, sofern ein Vorzeichen verwendet wird. Der Bereich, der von diesem Datentyp dargestellt werden kann, erstreckt sich von 0 bis +9223372036854775807. Die folgenden Zahlen sind gültige Beispiele dieses Datentyps: `1, 0, 12678967543233 und +100000`.

## Datentyp 'nonPositiveInteger'

Der Datentyp 'xs:nonPositiveInteger' stellt eine ganze Zahl dar, die kleiner-gleich null ist. Dieser Datentyp ist vom Datentyp 'xs:integer' abgeleitet.

Das lexikalische Format des Datentyps 'xs:nonPositiveInteger' ist ein optionales Vorzeichen, auf das eine Sequenz aus Dezimalziffern mit begrenzter Länge folgt. Bei lexikalischen Formaten, die null angeben, kann ein negatives (-) oder auch gar kein Vorzeichen verwendet werden. Bei allen anderen lexikalischen Formaten muss das negative Vorzeichen (-) vorhanden sein. Der Bereich, der mit diesem Datentyp dargestellt werden kann, erstreckt sich von -9223372036854775808 bis 0. Die folgenden Zahlen sind gültige Beispiele dieses Datentyps: `-1, 0, -12678967543233 und -100000`.

## Datentyp 'normalizedString'

Der Datentyp 'xs:normalizedString' stellt eine um Zeichen aus der Leerzeichenklasse normalisierte Zeichenfolge dar. Dieser Datentyp ist vom Datentyp 'xs:string' abgeleitet.

Das lexikalische Format des Datentyps 'xs:normalizedString' ist eine Zeichenfolge ohne Rücklaufzeichen (X'0D'), ohne Zeilenvorschubzeichen (X'0A') und ohne Tabulatorzeichen (X'09').

## Datentyp NOTATION

Der Datentyp 'xs:NOTATION' stellt den Attributtyp NOTATION aus *XML 1.0 (Third Edition)* dar. Dieser Datentyp ist vom Datentyp 'xdt:anyAtomicType' abgeleitet.

Das lexikalische Format des Datentyps 'xs:NOTATION' entspricht dem lexikalischen Format des Datentyps 'xs:QName'.

## Datentyp 'positiveInteger'

Der Datentyp 'xs:positiveInteger' stellt eine positive ganze Zahl dar, die größergleich 1 ist. Dieser Datentyp ist vom Datentyp 'xs:nonNegativeInteger' abgeleitet.

Das lexikalische Format des Datentyps 'xs:positiveInteger' ist ein optionales positives Vorzeichen (+), auf das eine Sequenz aus Dezimalziffern mit begrenzter Länge folgt. Der Bereich, der von diesem Datentyp dargestellt werden kann, erstreckt sich von +1 bis +9223372036854775807. Die folgenden Zahlen sind gültige Beispiele dieses Datentyps: `1`, `12678967543233` und `+100000`.

## Datentyp 'QName'

Der Datentyp 'xs:QName' stellt einen qualifzierten XML-Namen (QName) dar. Ein QName umfasst ein optionales Namensbereichspräfix, eine URI zur Identifizierung des XML-Namensbereichs und einen lokalen Teil, bei dem es sich um einen Namen ohne Doppelpunkte (NCName) handelt. Dieser Datentyp ist vom Datentyp 'xdt:anyAtomicType' abgeleitet.

Das lexikalische Format des Datentyps 'xs:QName' ist eine Zeichenfolge mit folgendem Format: *`Präfix:lokaler_Name`*. Zur Beschreibung dieses Formats werden die folgenden Abkürzungen verwendet:

*Präfix*
: Optional. Ein Namensbereichspräfix. Das Namensbereichspräfix muss durch eine Namensbereichsdeklaration an einen URI-Verweis gebunden worden sein. Das Präfix fungiert lediglich als Platzhalter für den Namen eines Namensbereichs. Wird kein Präfix angegeben, wird die URI des Standardnamensbereichs für Elemente/Typen verwendet.

*lokaler_Name*
: Ein Name ohne Doppelpunkte (NCName), bei dem es sich um den lokalen Teil des qualifizierten Namens handelt. Ein NCName ist ein XML-Name, der keinen Doppelpunkt enthält.

Die folgende Zeichenfolge beispielsweise zeigt ein gültiges lexikalisches Format eines qualifizierten Namens, der ein Präfix enthält:

```
ns1:emp
```

## Datentyp 'short'

Der Datentyp 'xs:short' stellt eine ganze Zahl dar, die kleiner-gleich 32.767 und größer-gleich -32.768 ist. Dieser Datentyp ist vom Datentyp 'xs:int' abgeleitet.

Das lexikalische Format des Datentyps 'xs:short' ist ein optionales Zeichen, auf das eine Sequenz aus Dezimalziffern mit begrenzter Länge folgt. Wird kein Zeichen angegeben, wird von einem Pluszeichen (+) ausgegangen. Die folgenden Zahlen sind gültige Beispiele dieses Datentyps: `-1, 0, 12678 und +10000`.

## Datentyp 'string'

Der Datentyp 'xs:string' stellt eine Zeichenfolge dar. Dieser Datentyp ist vom Datentyp 'xdt:anyAtomicType' abgeleitet.

Das lexikalische Format des Datentyps 'xs:string' ist eine Sequenz aus beliebigen Zeichen aus dem für XML gültigen Zeichensatz.

## Datentyp 'time'

Der Datentyp 'xs:time' stellt einen Zeitpunkt dar, der jeden Tag wiederkehrt. Dieser Datentyp ist vom Datentyp 'xdt:anyAtomicType' abgeleitet.

Das lexikalische Format des Datentyps 'xs:time' lautet *hh*:*mm*:*ss*.*ssssssszzzzz*. Dieses Format ist eine abgeschnittene Darstellung des Datentyps 'xs:dateTime', in der das Jahr, der Tag bzw. der Monat nicht enthalten sind. Zur Beschreibung dieses Formats werden die folgenden Abkürzungen verwendet:

*hh* Ein zweistelliges Numeral zur Darstellung der Stunde. Der Wert 24 ist nur dann zulässig, wenn die Minuten und Sekunden durch Nullen dargestellt werden. Eine Abfrage mit der Zeitangabe 24:00:00 wird wie die Zeitangabe 00:00:00 des nächsten Tages behandelt.

**:** Ein Trennzeichen zwischen den Teilen des Zeitabschnitts.

*mm*
Ein zweistelliges Numeral zur Darstellung der Minute.

*ss* Ein zweistelliges Numeral zur Darstellung der Sekunde.

**.***ssssss*
Optional. Sofern vorhanden, handelt es sich um ein ein- bis sechsstelliges Numeral zur Darstellung der Sekundenbruchteile.

*zzzzzz*
Optional. Sofern vorhanden, handelt es sich um die Zeitzone. Der Abschnitt „Zeitzonenbezugswert" auf Seite 33 enthält weitere Informationen zum Format dieses Merkmals.

Das folgende Format beispielsweise, das einen optionalen Bezugswert für die Zeitzone enthält, gibt 13:20 Uhr nach Eastern Standard Time an, die im Vergleich zur Weltzeit (UTC) 5 Stunden zurückliegt:

```
13:20:00-05:00
```

## Datentyp 'token'

Der Datentyp 'xs:token' stellt eine mit einem Token versehene Zeichenfolge dar. Dieser Datentyp ist vom Datentyp 'xs:normalizedString' abgeleitet.

Das lexikalische Format des Datentyps 'xs:token' ist eine Zeichenfolge, die keines der folgenden Zeichen enthält:

- Rücklaufzeichen (X'0D')
- Zeilenvorschubzeichen (X'0A')
- Tabulatorzeichen (X'09')
- Führende oder folgende Leerzeichen (X'20')
- Interne Sequenzen von zwei und mehr Leerzeichen

## Datentyp 'unsignedByte'

Der Datentyp 'xs:unsignedByte' stellt eine ganze Zahl ohne Vorzeichen dar, die kleiner-gleich 255 ist. Dieser Datentyp ist vom Datentyp 'xs:unsignedShort' abgeleitet.

Das lexikalische Format des Datentyps 'xs:unsignedByte' ist eine Sequenz aus Dezimalziffern mit begrenzter Länge. Die folgenden Zahlen sind gültige Beispiele dieses Datentyps: 0, 126 und 100.

## Datentyp 'unsignedInt'

Der Datentyp 'xs:unsignedInt' stellt eine ganze Zahl ohne Vorzeichen dar, die kleiner-gleich 4.294.967.295 ist. Dieser Datentyp ist vom Datentyp 'xs:unsignedLong' abgeleitet.

Das lexikalische Format des Datentyps 'xs:unsignedInt' ist eine Sequenz aus Dezimalziffern mit begrenzter Länge. Die folgenden Zahlen sind gültige Beispiel dieses Datentyps: `0, 1267896754 und 100000`.

## Datentyp 'unsignedLong'

Der Datentyp 'xs:unsignedLong' stellt eine ganze Zahl ohne Vorzeichen dar, die kleiner-gleich 9.223.372.036.854.775.807 ist. Dieser Datentyp ist vom Datentyp 'xs:nonNegativeInteger' abgeleitet.

Das lexikalische Format des Datentyps 'xs:unsignedLong' ist eine Sequenz aus Dezimalziffern mit begrenzter Länge. Die folgenden Zahlen sind gültige Beispiel dieses Datentyps: `0, 12678967543233 und 100000`.

## Datentyp 'unsignedShort'

Der Datentyp 'xs:unsignedShort' stellt eine ganze Zahl ohne Vorzeichen dar, die kleiner-gleich 65.535 ist. Dieser Datentyp ist vom Datentyp 'xs:unsignedInt' abgeleitet.

Das lexikalische Format des Datentyps 'xs:unsignedShort' ist eine Sequenz aus Dezimalziffern mit begrenzter Länge. Die folgenden Zahlen sind gültige Beispiel dieses Datentyps: `0, 12678 und 10000`.

## Datentyp 'untyped'

Der Datentyp 'xdt:untyped' bezeichnet einen Knoten, der nicht von einem XML-Schema geprüft worden ist. Dieser Datentyp ist vom Datentyp 'xs:anyType' abgeleitet.

Wenn in der Annotation eines Elementknotens der Datentyp 'xdt:untyped' angegeben ist, enthält die Annotation aller untergeordneten Elemente ebenfalls den Datentyp 'xdt:untyped'.

## Datentyp 'untypedAtomic'

Der Datentyp 'xdt:untypedAtomic' bezeichnet einen atomaren Wert, der nicht von einem XML-Schema geprüft worden ist. Dieser Datentyp ist vom Datentyp 'xdt:anyAtomicType' abgeleitet.

Der Datentyp 'xdt:untypedAtomic' hat ein uneingeschränktes lexikalisches Format.

## Datentyp 'yearMonthDuration'

Der Datentyp 'xdt:yearMonthDuration' stellt eine Zeitdauer dar, die durch die Gregorianischen Komponenten für Jahr und Monat ausgedrückt wird. Dieser Datentyp ist vom Datentyp 'xs:duration' abgeleitet.

Der Bereich, der von diesem Datentyp dargestellt werden kann, erstreckt sich von '-P83333333333333Y3M' bis 'P83333333333333Y3M' (oder von '-999999999999999' bis '999999999999999' Monaten).

Das lexikalische Format des Datentyps 'xdt:yearMonthDuration' lautet `PnYnM`, wobei es sich um eine Abkürzung des *ISO 8601*-Formats handelt. Zur Beschreibung dieses Formats werden die folgenden Abkürzungen verwendet:

*n***Y** *n* ist eine ganze Zahl ohne Vorzeichen zur Darstellung der Anzahl der Jahre (Y).

*n***M**
*n* ist eine ganze Zahl ohne Vorzeichen zur Darstellung der Anzahl der Monate (M).

Ein optionales führendes Minuszeichen (-) gibt eine negative Dauer an. Wird kein Zeichen angegeben, wird von einer positiven Dauer ausgegangen.

Das folgende Format beispielsweise gibt eine Dauer von 1 Jahr und 2 Monaten an:

```
P1Y2M
```

Das folgende Format gibt eine Dauer von negativen 13 Monaten an:

```
-P13M
```

Eine geringere Genauigkeit und abgeschnittene Darstellungen dieses Formats sind zulässig, müssen jedoch die folgenden Anforderungen erfüllen:

- Die Bezeichnung `P` muss stets vorhanden sein.
- Wenn die Anzahl der Jahre oder Monate in einem Ausdruck null entspricht, können die Zahl und die entsprechende Bezeichnung ausgelassen werden. Es muss jedoch mindestens eine Zahl samt Bezeichnung (`Y` oder `M`) vorhanden sein.

So sind beispielsweise die folgenden Formate zulässig:

```
P1347Y
P1347M
```

Das Format `P-1347M` ist nicht zulässig, wohingegen das Format `-P1347M` gültig ist. Die Formate `P24YM` und `PY43M` sind nicht zulässig, da `Y` und `M` jeweils mindestens eine führende Ziffer haben müssen.

DB2 speichert Werte vom Typ 'xdt:yearMonthDuration' in einem normalisierten Format. Im normalisierten Format sind die Werte der Monatskomponente kleiner als 12. Jedes Vielfache von 12 Monaten wird in ein Jahr umgewandelt. Der folgende XQuery-Ausdruck beispielsweise ruft eine Konstruktorfunktion auf, die einen Wert vom Typ 'yearMonthDuration' von 20 Jahren und 30 Monaten angibt:

```
  xquery
xdt:yearMonthDuration("P20Y30M")
```

Bei der Angabe der Dauer werden die 30 Monate in 2 Jahre und 6 Monate umgewandelt. Demnach gibt der Ausdruck den normalisierten yearMonthDuration-Wert 'P22Y6M' zurück.

# Kapitel 3. Prolog

Der *Prolog* ist eine Reihe von Deklarationen, mit denen die Verarbeitungsumgebung einer Abfrage definiert wird. Auf jede Deklaration im Prolog folgt ein Semikolon (;). Der Prolog ist ein optionaler Teil der Abfrage. Eine gültige Abfrage kann auch lediglich aus einem Abfragehauptteil ohne Prolog bestehen.

Der Prolog umfasst eine optionale Versionsdeklaration ('version', Namensbereichsdeklarationen ('namespace') sowie *Setter*, bei denen es sich um optionale Deklarationen handelt, die Werte für Merkmale setzen, die einen Einfluss auf die Abfrageverarbeitung haben.

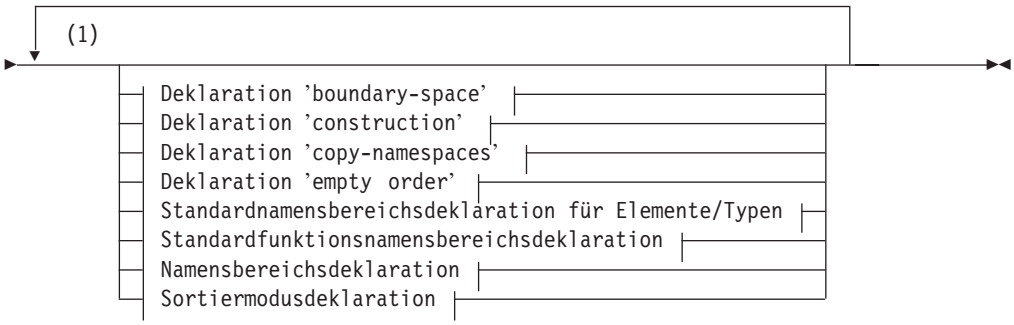
DB2 XQuery unterstützt die Deklaration 'boundary-space', mit deren Hilfe die Art der Abfrageverarbeitung geändert werden kann. Der Prolog enthält auch Namensbereichsdeklarationen und Standardnamensbereichsdeklarationen.

Darüber hinaus unterstützt DB2 XQuery die nachstehenden Setter. Diese Setter ändern jedoch nicht die Verarbeitungsumgebung, da DB2 XQuery in jedem Fall lediglich eine Option unterstützt:

- Deklaration 'construction'
- Deklaration 'copy-namespaces'
- Deklaration 'empty order'
- Sortiermodusdeklaration 'ordering'

Die Versionsdeklaration (sofern vorhanden) muss im Prolog an erster Stelle stehen. Setter und andere Deklarationen können im Prolog in beliebiger Reihenfolge nach der Versionsdeklaration stehen.

## Syntax




**Anmerkungen:**

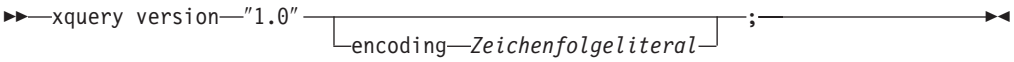
1 Mit Ausnahme der Namensbereichsdeklaration kann jede Deklaration nur einmal angegeben werden.

## Versionsdeklaration

Eine Versionsdeklaration steht am Anfang einer Abfrage, um die Version der XQuery-Syntax und der -Semantik anzugeben, die für die Verarbeitung der Abfrage erforderlich sind. Die Versionsdeklaration kann eine Deklaration vom Typ 'encoding' (Verschlüsselung) enthalten; die Deklaration 'encoding' wird jedoch von DB2 XQuery ignoriert.

Ist die Versionsdeklaration vorhanden, muss sie am Anfang des Prologs stehen. Die Version "1.0" ist die einzige Version, die von DB2 XQuery unterstützt wird.

### Syntax




**1.0** Gibt an, dass zur Verarbeitung der Abfrage die XQuery-Syntax und -Semantik der Version 1.0 erforderlich sind.

*Zeichenfolgeliteral*
Gibt ein Zeichenfolgeliteral an, das den Verschlüsselungsnamen darstellt. Die Angabe einer Deklaration vom Typ 'encoding' hat keine Auswirkung auf die Abfrage, da der Wert von *Zeichenfolgeliteral* ignoriert wird. Bei DB2 XQuery wird stets vorausgesetzt, dass die Verschlüsselung UTF-8 ist.

### Beispiel

Die folgende Versionsdeklaration gibt an, dass die Abfrage von einer Implementierung ausgeführt werden muss, die XQuery Version 1.0 unterstützt:

```
xquery version "1.0";
```

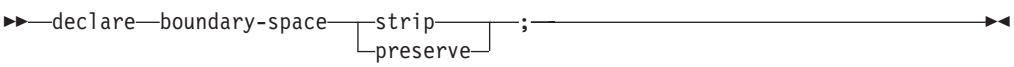
## Deklaration 'boundary-space'

Mithilfe der Deklaration 'boundary-space' im Abfrageprolog wird die Richtlinie für Begrenzungsleerzeichen für die Abfrage festgelegt. Mithilfe der *Richtlinie für Begrenzungsleerzeichen (boundary-space)* wird gesteuert, wie Begrenzungsleerzeichen von Elementkonstruktoren verarbeitet werden sollen. Zu den *Begrenzungsleerzeichen* gehören alle Leerzeichen, die eigenständig in den Begrenzungen zwischen Befehlen oder eingeschlossenen Ausdrücken in Elementkonstruktoren vorkommen.

Die Richtlinie für Begrenzungsleerzeichen (boundary-space) gibt an, ob Begrenzungsleerzeichen beim Erstellen von Elementen beibehalten oder verworfen (entfernt) werden sollen. Wird keine Deklaration 'boundary-space' angegeben, werden Begrenzungsleerzeichen während der Elementerstellung standardmäßig verworfen.

Der Prolog kann für eine Abfrage jeweils nur eine einzige Deklaration vom Typ 'boundary-space' enthalten.

### Syntax



**strip**
Gibt an, dass Begrenzungsleerzeichen beim Erstellen von Elementen entfernt werden sollen.

**preserve**
Gibt an, dass Begrenzungsleerzeichen beim Erstellen von Elementen beibehalten werden sollen.

### Beispiel

Die folgende Deklaration vom Typ 'boundary-space' gibt an, dass Begrenzungsleerzeichen beim Erstellen von Elementen beibehalten werden sollen:

```
declare boundary-space preserve;
```

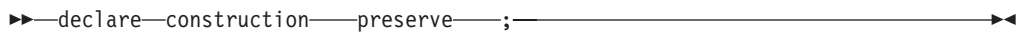
## Deklaration 'construction'

Mit der Deklaration 'construction' im Abfrageprolog wird der Erstellungsmodus für die Abfrage festgelegt. Der *Erstellungsmodus* steuert, wie Typenannotationen zu Element- und Attributknoten zugeordnet werden, die als Inhalt des neu erstellten Knotens kopiert werden.

In DB2 XQuery ist der Erstellungsmodus für erstellte Elementknoten stets **preserve**. Beim Erstellungsmodus **preserve** behalten die kopierten Attribute und untergeordneten Elemente (Nachkommen) des erstellten Elements jeweils ihren ursprünglichen Typ bei.

Eine Deklaration vom Typ 'construction', in der ein anderer Wert als **preserve** angegeben wird, führt zu einem Fehler. Der Prolog kann für eine Abfrage jeweils nur eine einzige Deklaration vom Typ 'construction' enthalten.

### Syntax




**preserve**
Gibt an, dass die kopierten Attribute und untergeordneten Elemente des erstellten Elements jeweils ihren ursprünglichen Typ beibehalten.

### Beispiel

Die folgende Deklaration 'construction' ist zwar gültig, ändert jedoch nicht das Standardverhalten für die Elementerstellung:

```
declare construction preserve;
```

## Deklaration 'copy-namespaces'

Mit dem Modus 'copy-namespaces' werden die Namensbereichsbindungen gesteuert, die zugeordnet werden, wenn ein vorhandener Elementknoten von einem Elementkonstruktor kopiert wird.

In DB2 XQuery ist der Modus 'copy-namespaces' stets auf **preserve** und **inherit** eingestellt. Die Einstellung **preserve** gibt an, das alle gültigen Namensbereiche des ursprünglichen Elements in der neuen Kopie beibehalten werden. Der Standardnamensbereich wird wie alle anderen Namensbereichsbindungen behandelt: der kopierte Knoten behält seinen Standardnamensbereich bzw. dessen Nicht-

vorhandensein bei. Die Einstellung **inherit** gibt an, dass der kopierte Knoten die gültigen Namensbereiche vom erstellten Knoten übernimmt. Im Falle eines Konflikts haben die beibehaltenen Namensbereichsbindungen des ursprünglichen Knotens Vorrang.

Eine Deklaration vom Typ 'copy-namespaces', die andere Werte als **preserve** und **inherit** angibt, führt zu einem Fehler. Der Prolog kann für eine Abfrage jeweils nur eine einzige Deklaration vom Typ 'copy-namespaces' enthalten.

### Syntax




**preserve**
Gibt an, dass alle gültigen Namensbereiche des ursprünglichen Elements in der neuen Kopie beibehalten werden.

**inherit**
Gibt an, dass der kopierte Knoten die gültigen Namensbereiche vom erstellten Knoten übernimmt.

### Beispiel

Die folgende Deklaration 'copy-namespace' ist zwar gültig, ändert jedoch nicht das Standardverhalten für die Elementerstellung:

```
declare copy-namespaces preserve, inherit;
```

## Deklaration 'default element/type namespace'

Die Deklaration 'default element/type namespace' im Abfrageprolog gibt den Namensbereich an, der für die qualifizierten Namen (QNames) von Element- und Typnamen, die kein Präfix haben, verwendet werden soll.

Der Abfrageprolog darf lediglich eine Deklaration vom Typ 'default element/type namespace' enthalten. Diese Deklaration ist während der gesamten Abfrage, in der sie angegeben wird, gültig, sofern sie nicht von einem Deklarationsattribut für einen Namensbereich in einem direkten Elementkonstruktor überschrieben wird. Wird kein Standardnamensbereich für Elemente/Typen angegeben, befinden sich die Element- und Typnamen ohne Präfix in keinem Namensbereich.

Der Standardnamensbereich für Elemente/Typen gilt nicht für unqualifizierte Attributnamen. Attributnamen und Variablennamen ohne Präfix befinden sich in keinem Namensbereich.

### Syntax




**element**
Gibt an, dass die Deklaration vom Typ 'default element/type namespace' ist.

*URI-Literal*
Gibt ein Zeichenfolgeliteral an, das die URI für den Namensbereich darstellt. Das Zeichenfolgeliteral muss eine gültige URI oder eine Zeichenfolge mit Nulllänge sein. Ist das Zeichenfolgeliteral in einer Deklaration 'default element/

type namespace' eine Zeichenfolge mit Nulllänge, befinden sich die Element- und Typnamen ohne Präfix in keinem Namensbereich.

### Beispiel

Die folgende Deklaration gibt an, dass der Standardnamensbereich für Element- und Typnamen derjenige Namensbereich ist, der der URIhttp://posample.ibm.org zugeordnet ist:

```
declare default element namespace "http://posample.org";
<name>Snow boots</name>
```

Bei Ausführung der Abfrage im Beispiel befindet sich der neu erstellte Knoten (ein Elementknoten namens name) in dem Namensbereich, der der Namensbereich-URI http://posample.org zugeordnet ist.

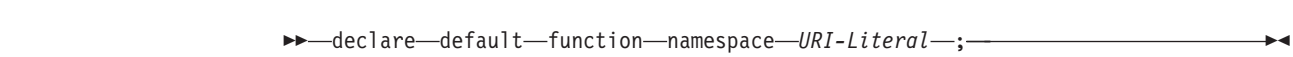
## Deklaration 'default function namespace'

Die Deklaration 'default function namespace' im Abfrageprolog gibt eine Namensbereichs-URI an, die für Funktionsnamen ohne Präfix in Funktionsaufrufen verwendet wird.

Der Abfrageprolog darf lediglich eine Deklaration vom Typ 'default function namespace' (Standardfunktionsnamensbereich) enthalten. Wird kein Standardfunktionsnamensbereich 'default function namespace' deklariert, wird als Standardfunktionsnamensbereich der Namensbereich von XPath- und XQuery-Funktionen (http://www.w3.org/2005/xpath-functions) verwendet. Wird ein Standardfunktionsnamensbereich angegeben, kann jede Funktion in diesem Namensbereich ohne Angabe eines Präfixes aufgerufen werden.

DB2 XQuery gibt einen Fehler zurück, wenn der lokale Name eines Funktionsaufrufs ohne Präfix keiner Funktion im Standardfunktionsnamensbereich entspricht.

### Syntax




**function**
Gibt an, dass die Deklaration vom Typ 'default function namespace' ist.

*URI-Literal*
Gibt ein Zeichenfolgeliteral an, das die URI für den Namensbereich darstellt. Das Zeichenfolgeliteral muss eine gültige URI oder eine Zeichenfolge mit Nulllänge sein. Handelt es sich beim Zeichenfolgeliteral in einer Deklaration vom Typ 'default function namespace' um eine Zeichenfolge mit Nulllänge, müssen bei allen Funktionsaufrufen Funktionsnamen mit Präfixen verwendet werden, da sich jede Funktion in einem beliebigen Namensbereich befinden kann.

### Beispiel

Die folgende Deklaration gibt an, dass der Standardfunktionsnamensbereich der URI http://www.ibm.com/xmlns/prod/db2/functions zugeordnet ist:

```
declare default function namespace "http://www.ibm.com/xmlns/prod/db2/functions";
```

Innerhalb des Abfragehauptteils dieses Beispiels könnte auf eine beliebige Funktion im Standardfunktionsnamensbereich verwiesen werden, ohne dass im Funktionsnamen ein Präfix angegeben werden müsste. Dieser Standardfunktionsnamens-

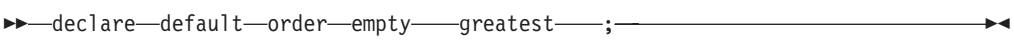
bereich enthält beispielsweise die Funktion `xmlcolumn`, sodass Sie `xmlcolumn('T1.MYDOC')` anstelle von `db2-fn:xmlcolumn('T1.MYDOC')` eingeben können. Da der Standardfunktionsnamensbereich in diesem Beispiel jedoch nicht mehr dem Namensbereich für XQuery-Funktionen zugeordnet ist, müsste beim Aufrufen integrierter XQuery-Funktionen ein Präfix angegeben werden. So müsste beispielsweise `fn:current-date()` anstelle von `current-date()` eingegeben werden.

## Deklaration 'empty order'

Die Deklaration 'empty order' im Abfrageprolog steuert, ob eine leere Sequenz bzw. eine Nichtzahl (NaN) bei der Verarbeitung einer Klausel **ORDER BY** in einem FLWOR-Ausdruck als größter oder kleinster Wert interpretiert wird.

In DB2 XQuery wird eine leere Sequenz bei der Verarbeitung einer Klausel **ORDER BY** in einem FLWOR-Ausdruck stets als größter Wert interpretiert. Eine Nichtzahl wird als ein Wert interpretiert, der größer als alle anderen Werte ist, mit Ausnahme einer leeren Sequenz. Diese Einstellung kann nicht überschrieben werden. Eine Deklaration vom Typ 'empty order', in der ein anderer Wert als **empty greatest** angegeben wird, führt zu einem Fehler. Der Abfrageprolog kann für eine Abfrage jeweils nur eine einzige Deklaration vom Typ 'empty order' enthalten.

### Syntax




**greatest**
Gibt an, dass eine leere Sequenz bei der Verarbeitung einer Klausel **ORDER BY** in einem FLWOR-Ausdruck stets als größter Wert interpretiert wird. Eine Nichtzahl wird als ein Wert interpretiert, der größer als alle anderen Werte ist, mit Ausnahme einer leeren Sequenz.

### Beispiel

Die folgende Deklaration vom Typ 'empty order' ist gültig:

```
declare default order empty greatest;
```

## Sortiermodusdeklaration

Mit einer *Sortiermodusdeklaration* im Abfrageprolog wird der Sortiermodus für die Abfrage festgelegt. Der *Sortiermodus* definiert die Sortierreihenfolge von Knoten im Abfrageergebnis.

Da DB2 XQuery nicht den Sortiermodus 'ordered' (sortiert) gemäß Definition in *XQuery 1.0: An XML Query Language* unterstützt, muss die Sortiermodusdeklaration (sofern vorhanden) den Modus 'unordered' (unsortiert) angeben. Der Abschnitt „Reihenfolge der Ergebnisse in XQuery-Ausdrücken" auf Seite 58 enthält die Regeln, denen die Sortierung von Abfrageergebnissen in DB2 XQuery unterliegt.

Der Abfrageprolog darf lediglich eine Sortiermodusdeklaration enthalten. Eine Sortiermodusdeklaration, die einen anderen Wert als 'unordered' (unsortiert) angibt, führt zu einem Fehler.

### Syntax

*unordered*
Gibt an, dass die Regeln des Modus 'ordered' (sortiert) gemäß *XQuery 1.0: An XML Query Language* nicht gelten. Der Abschnitt „Reihenfolge der Ergebnisse in XQuery-Ausdrücken" auf Seite 58 enthält die Regeln, denen die Sortierung von Abfrageergebnissen in DB2 XQuery unterliegt.

### Beispiel

Die folgende Deklaration ist zwar gültig, ändert jedoch nicht das Standardverhalten bei der Sortierung, da DB2 XQuery ohnehin nur den Modus 'unordered' (unsortiert) unterstützt:

```
declare ordering unordered;
```

## Namensbereichsdeklaration

Eine Namensbereichsdeklaration im Abfrageprolog deklariert ein Namensbereichspräfix und ordnet dem Präfix eine Namensbereichs-URI zu. Eine Zuordnung zwischen einem Präfix und einer Namensbereichs-URI wird als *Namensbereichsbindung* bezeichnet. Ein Namensbereich, der in einer Namensbereichsdeklaration gebunden wird, wird den statisch bekannten Namensbereichen hinzugefügt. Die *statisch bekannten Namensbereiche* umfassen alle Namensbereichsbindungen, die verwendet werden können, um Namensbereichspräfixe bei der Verarbeitung einer Abfrage aufzulösen.

Die Namensbereichsdeklaration ist während der gesamten Abfrage, in der sie angegeben wird, gültig, sofern sie nicht von einem Deklarationsattribut für einen Namensbereich in einem direkten Elementkonstruktor überschrieben wird. Mehrere Deklarationen desselben Namensbereichspräfixes im Abfrageprolog führen zu einem Fehler.

### Syntax

*Präfix*
Gibt ein Namensbereichspräfix an, das an die von *URI-Literal* angegebene URI gebunden wird. Das Namensbereichspräfix wird in qualifizierten Namen (QNames) verwendet, um den Namensbereich für ein Element, ein Attribut, einen Datentyp oder eine Funktion zu identifizieren.

Die Präfixe `xmlns` und `xml` sind reserviert und dürfen nicht als Präfixe in Namensbereichsdeklarationen angegeben werden.

*URI-Literal*
Gibt die URI an, an die das Präfix gebunden ist. *URI-Literal* muss eine Literalzeichenfolge mit einer Länge größer null sein, die eine gültige URI enthält.

### Beispiel

Die folgende Abfrage umfasst eine Namensbereichsdeklaration, mit der das Namensbereichspräfix `ns1` deklariert wird, dem die Namensbereichs-URI `http://posample.org` zugeordnet wird:

```
declare namespace ns1 = "http://posample.org";
<ns1:name>Thermal gloves</ns1:name>
```

Bei Ausführung der Abfrage im Beispiel befindet sich der neu erstellte Knoten (ein Elementknoten namens `name`) in dem Namensbereich, der der Namensbereich-URI `http://posample.org` zugeordnet ist.

## Vordeklarierte Namensbereichspräfixe

In XQuery gibt es mehrere vordeklarierte Namensbereichspräfixe, die bereits vor der Verarbeitung der einzelnen Abfragen in den statisch bekannten Namensbereichen enthalten sind. Sämtliche vordeklarierten Präfixe können ohne explizite Deklaration verwendet werden. Die vordeklarierten Namensbereichspräfixe für DB2 XQuery umfassen die in folgender Tabelle aufgeführten Paare aus Präfix und URI:

*Tabelle 11. Vordeklarierte Namensbereiche in DB2 XQuery*

| Präfix | URI | Beschreibung |
|---|---|---|
| xml | `http://www.w3.org/XML/1998/namespace` | Für XML reservierter Namensbereich |
| xs | `http://www.w3.org/2001/XMLSchema` | Namensbereich des XML-Schemas |
| xsi | `http://www.w3.org/2001/XMLSchema-instance` | Namensbereich der XML-Schemainstanz |
| fn | `http://www.w3.org/2005/xpath-functions` | Standardfunktionsnamensbereich |
| xdt | `http://www.w3.org/2005/xpath-datatypes` | Namensbereich für XQuery-Typen |
| db2-fn | `http://www.ibm.com/xmlns/prod/db2/functions` | DB2-Funktionsnamensbereich |

Vordeklarierte Namensbereichspräfixe können durch Angabe einer Namensbereichsdeklaration in einem Abfrageprolog überschrieben werden. Die dem Präfix `xml` zugeordnete URI kann jedoch nicht überschrieben werden.

# Kapitel 4. Ausdrücke

Ausdrücke sind die Grundbausteine einer Abfrage. Ausdrücke können entweder alleine oder in Kombination mit anderen Ausdrücken für komplexe Abfragen verwendet werden. DB2 XQuery unterstützt verschiedene Ausdruckstypen für das Arbeiten mit XML-Daten.

## Ausdrücke auswerten und verarbeiten

Häufig umfasst die Verarbeitung von Ausdrücken eine Reihe von Operationen. Zu diesen Operationen gehören beispielsweise das Extrahieren von atomaren Werten aus Knoten, die Verwendung der Typenumstufung und Subtypsubstitution zum Abrufen von Werten eines erwarteten Typs und das Berechnen des Booleschen Wertes einer Sequenz.

In DB2 XQuery können Aktualisierungsausdrücke nur innerhalb der Klausel **MODIFY** eines Umsetzungsausdrucks verwendet werden. Die Abschnitte „Umsetzungsausdruck" auf Seite 128 und „Aktualisierungsausdrücke in einem Umsetzungsausdruck verwenden" auf Seite 124 enthalten Informationen zum XQuery-Umsetzungsausdruck sowie zur Verarbeitung von Aktualisierungsausdrücken.

### Dynamischer Kontext und Fokus

Der dynamische Kontext eines Ausdrucks besteht aus den Informationen, die zum Zeitpunkt der Auswertung des Ausdrucks verfügbar sind. Der Fokus, der aus dem Kontextelement, der Kontextposition und der Kontextgröße besteht, ist ein wichtiger Teil des dynamischen Kontextes.

Wenn DB2 XQuery die einzelnen Elemente in einer Sequenz verarbeitet, ändert sich im Zuge dieser Verarbeitung auch der Fokus. Der Fokus besteht aus den folgenden Informationen:

**Kontextelement**
Der atomare Wert oder Knoten, der zum jeweiligen Zeitpunkt verarbeitet wird. Das Kontextelement kann vom Kontextelementausdruck, der aus einem einzigen Punkt (`.` ) besteht, abgerufen werden.

**Kontextposition**
Die Position des Kontextelements in der Sequenz, die zum jeweiligen Zeitpunkt verarbeitet wird. Das Kontextelement kann von der Funktion 'fn:position()' abgerufen werden.

**Kontextgröße**
Die Anzahl der Elemente in der Sequenz, die zum jeweiligen Zeitpunkt verarbeitet wird. Die Kontextgröße kann von der Funktion 'fn:last()' abgerufen werden.

### Vorrangstellung

Die Grammatik von XQuery definiert eine integrierte Vorrangstellung unter den Operatoren und Ausdrücken. Wenn ein Ausdruck mit geringerer Vorrangstellung als Operand eines Ausdrucks mit höherer Vorangestellung verwendet wird, muss der Ausdruck mit der geringeren Vorrangstellung in runde Klammern eingeschlossen werden.

Die folgende Tabelle enthält eine Liste der XQuery-Operatoren und -Ausdrücke in der Reihenfolge ihrer Vorrangstellung von der niedrigsten zur höchsten Stellung. Die Spalte 'Assoziativität' gibt die Reihenfolge an, in der Operatoren bzw. Ausdrücke gleicher Vorrangstellung angewandt werden.

*Tabelle 12. Vorrangstellung in DB2 XQuery*

| Operator oder Ausdruck | Assoziativität |
|---|---|
| , (Komma) | Von links nach rechts |
| := (Zuordnung) | Von rechts nach links |
| FLWOR, some, every, if | Von links nach rechts |
| or | Von links nach rechts |
| and | Von links nach rechts |
| eq, ne, lt, le, gt, ge, =, !=, <, <=, >, >=, is, <<, >> | Von links nach rechts |
| to | Von links nach rechts |
| +, - | Von links nach rechts |
| *, div, idiv, mod | Von links nach rechts |
| union, \| | Von links nach rechts |
| intersect, except | Von links nach rechts |
| castable | Von links nach rechts |
| cast | Von links nach rechts |
| -(unary), +(unary) | Von rechts nach links |
| ? | Von links nach rechts |
| /, // | Von links nach rechts |
| [ ], ( ), {} | Von links nach rechts |

## Reihenfolge der Ergebnisse in XQuery-Ausdrücken

Bei der Verwendung von DB2 XQuery geben einige Typen von XQuery-Ausdrücken Sequenzen in einer deterministischen Reihenfolge zurück, während andere Typen von Ausdrücken Sequenzen in einer nicht deterministischen Reihenfolge zurückgeben.

Die folgenden Typen von Ergebnissen geben Sequenzen in einer deterministischen Reihenfolge zurück:

- FLWOR-Ausdrücke, die eine explizite Klausel vom Typ **ORDER BY** enthalten, geben Ergebnisse in der angegebenen Reihenfolge zurück. Der folgende Ausdruck beispielsweise gibt eine Sequenz aus Produktelementen (product) in aufsteigender Reihenfolge nach Preis zurück:

  ```
  for $p in /product
  order by $p/price
  return $p
  ```

- Aktualisierungsausdrücke, die Elemente hinzufügen und explizite Positionsschlüsselwörter verwenden, geben Ergebnisse zurück, bei denen sich die hinzugefügten Elemente an den angegebenen Positionen befinden.

Im folgenden Aktualisierungsausdruck beispielsweise werden die Schlüsselwörter **as first into** verwendet, und das Element `<status>current</status>` wird als erstes Element in das Element 'customerinfo' eingefügt:

```
xquery declare default element namespace 'http://posample.org';
transform
copy $mycust := db2-fn:sqlquery('select info from customer where cid = 1001')
modify
  do insert <status>current</status> as first into $mycust/customerinfo
return $mycust
```

- Ausdrücke, die Sequenzen mithilfe des Operators **union**, **intersect** oder **except** verbinden, geben die Ergebnisse in Dokumentreihenfolge zurück.
- Pfadausdrücke, die die folgenden Bedingungen erfüllen, geben die Ergebnisse in Dokumentreihenfolge zurück:
  - Der Pfadausdruck enthält ausschließlich Vorwärtsachsenschritte.
  - Der Pfadausdruck stammt aus einem einzigen Knoten, der beispielsweise aus einem Funktionsaufruf oder einem Variablenverweis resultiert.
  - Kein Schritt in dem Pfadausdruck enthält mehr als ein einzelnes Vergleichselement.
  - Der Pfadausdruck enthält keinen Funktionsaufruf vom Typ 'fn:position' und keinen Funktionsaufruf vom Typ 'fn:last'.

  Das folgende Beispiel ist ein Pfadausdruck, der Ergebnisse in Dokumentreihenfolge zurückgibt, wobei davon ausgegangen wird, dass die Variable *$bib* an ein einzelnes Element gebunden ist:

  ```
  $bib/book[title eq "War and Peace"]/chapter
  ```
- Bereichsausdrücke, bei denen es sich um Ausdrücke handelt, die den Operator **to** enthalten, geben Sequenzen aus ganzen Zahlen in aufsteigender Reihenfolge zurück. Beispiel: `15 to 25`.
- Ausdrücke, die Kommaoperatoren enthalten, geben die Ergebnisse in der Reihenfolge ihrer Operanden zurück, wenn alle Operanden Sequenzen mit deterministischer Reihenfolge darstellen. Der folgende Ausdruck beispielsweise gibt die Sequenz (5, 10, 15, 16, 17, 18, 19, 20, 25) zurück:

  ```
  (5, 10, 15 to 20, 25)
  ```
- Andere Ausdrücke, die Operandenausdrücke enthalten, die Ergebnisse in deterministischer Reihenfolge zurückgeben, geben die Ergebnisse in einer deterministischen Reihenfolge zurück. Beispiel: Unter der Voraussetzung, dass die Variable *$pub* an ein einzelnes Element gebunden ist, gibt der folgende Bedingungsausdruck sortierte Ergebnisse zurück, da die Pfadausdrücke in den Klauseln vom Typ THEN und ELSE sortierte Ergebnisse zurückgeben:

  ```
  if ($pub/type eq "journal")
    then $pub/editor
    else $pub/author
  ```

Gibt ein Ausdruck, der in der vorstehenden Liste nicht enthalten ist, mehr als ein Element zurück, dann befinden sich die Elemente in der Sequenz in einer nicht deterministischen Reihenfolge.

*Tabelle 13. Zusammenfassung der Sortierung von Ergebnissen in XQuery-Ausdrücken*

| Ausdruckstyp | Bedingungen für eine deterministische Sortierung | Sortierung von Ergebnissen | Beispiel |
|---|---|---|---|
| FLWOR | Explizite Klausel **ORDER BY** | Festgelegt durch Klausel **ORDER BY** | Der folgende Ausdruck gibt eine Sequenz aus Produktelementen (product) in aufsteigender Reihenfolge nach Preis zurück:<br>`for $p in /product order by $p/price return $p` |
| Aktualisierungsausdrücke | Verwendung von Schlüsselwörtern, die beim Hinzufügen eines Elements eine Position angeben | Festgelegt durch die Schlüsselwörter der Aktualisierungsausdrücke | Beim Einfügen eines Elements mithilfe der Schlüsselwörter **as last into** wird das Element als letztes untergeordnetes Element (Kind) des angegebenen Knotens hinzugefügt. |
| Ausdrücke mit den Operatoren **union**, **intersect** oder **except** | Keine | Dokumentreihenfolge | `$managers union $students` |
| Pfadausdrücke | • Der Pfadausdruck enthält ausschließlich Vorwärtsachsenschritte.<br>• Der Pfadausdruck stammt aus einem einzigen Knoten, der beispielsweise aus einem Funktionsaufruf oder einem Variablenverweis resultiert.<br>• Kein Schritt in dem Pfadausdruck enthält mehr als ein einzelnes Vergleichselement.<br>• Der Pfadausdruck enthält keinen Funktionsaufruf vom Typ 'fn:position' und keinen Funktionsaufruf vom Typ 'fn:last'. | Dokumentreihenfolge | Das folgende Beispiel ist ein Pfadausdruck, der Ergebnisse in Dokumentreihenfolge zurückgibt, wobei davon ausgegangen wird, dass die Variable *$bib* an ein einzelnes Element gebunden ist:<br>`$bib/book [title eq "War and Peace"] /chapter` |
| Bereichsausdrücke, bei denen es sich um Ausdrücke mit dem Operator **to** handelt | Keine | Sequenz aus ganzen Zahlen in aufsteigender Reihenfolge | `15 to 25` |

*Tabelle 13. Zusammenfassung der Sortierung von Ergebnissen in XQuery-Ausdrücken (Forts.)*

| Ausdruckstyp | Bedingungen für eine deterministische Sortierung | Sortierung von Ergebnissen | Beispiel |
|---|---|---|---|
| Ausdrücke mit Kommaoperatoren | Alle Operanden sind Sequenzen mit deterministischer Reihenfolge | Die zurückgegebenen Ergebnisse liegen in der Reihenfolge ihrer Operanden vor | `(5, 10, 15 to 20, 25)` |
| Sonstige Ausdrücke | Alle Operandenausdrücke geben Ergebnisse zurück, die in einer deterministischen Reihenfolge vorliegen | Festgelegt durch die Sortierung der Ergebnisse der verschachtelten Ausdrücke | Unter der Voraussetzung, dass die Variable *$pub* an ein einzelnes Element gebunden ist, gibt der folgende Bedingungsausdruck sortierte Ergebnisse zurück, da die Pfadausdrücke in den Klauseln vom Typ THEN und ELSE sortierte Ergebnisse zurückgeben:<br>`if ($pub/type eq "journal") then $pub/editor else $pub/author` |

**Anmerkung:** Wird ein positionsgebundenes Vergleichselement auf eine Sequenz angewandt, die keine deterministische Reihenfolge aufweist, liegt das Ergebnis in einer nicht deterministischen Reihenfolge vor, sodass ein beliebiges Element in der Sequenz ausgewählt werden kann.

## Atomisierung

Der Begriff *Atomisierung* bezeichnet den Prozess der Konvertierung einer Sequenz aus Elementen in eine Sequenz aus atomaren Werten. Die Atomisierung wird von Ausdrücken immer dann verwendet, wenn eine Sequenz atomarer Werte erforderlich ist.

Jedes Element in einer Sequenz wird unter Beachtung der folgenden Regeln in einen atomaren Wert umgewandelt:

- Handelt es sich bei dem Element um einen atomaren Wert, wird der atomare Wert zurückgegeben.
- Handelt es sich bei dem Element um einen Knoten, wird dessen typisierter Wert zurückgegeben. Der *typisierte Wert* eines Knotens ist eine Sequenz von null oder mehr atomaren Werten, die aus dem Knoten extrahiert werden können. Verfügt der Knoten über keinen typisierten Wert, wird ein Fehler zurückgegeben.

Bei einer impliziten Atomisierung einer Sequenz wird das gleiche Ergebnis generiert wie beim expliziten Aufrufen der Funktion 'fn:data' für eine Sequenz.

Die folgende Sequenz beispielsweise enthält eine Kombination aus Knoten und atomaren Werten:

```
("Some text",<anElement xsi:type="string">More text</anElement>,
<anotherElement xsi:type="decimal">1.23</anotherElement>,1001)
```

Eine Atomisierung dieser Sequenz ergibt die folgende Sequenz aus atomaren Werten:

```
("Some text", "More text", 1.23, 1001)
```

Die folgenden XQuery-Ausdrücke verwenden die Atomisierung zum Konvertieren von Elementen in atomare Werte:

- Arithmetische Ausdrücke
- Vergleichsausdrücke
- Funktionsaufrufe mit Argumenten, deren erwartete Werte atomar sind
- Umsetzungsausdrücke
- Konstruktorausdrücke für unterschiedliche Knotensorten
- Klauseln **ORDER BY** in FLWOR-Ausdrücken
- Typkonstruktorfunktionen

## Subtypsubstitution

Bei der *Subtypsubstitution* handelt es sich um die Verwendung eines Wertes, dessen dynamischer Typ von einem erwarteten Typ abgeleitet wird.

Bei der Subtypsubstitution wird der tatsächliche Typ eines Wertes nicht geändert. Beispiel: Wird ein Wert vom Typ 'xs:integer' verwendet, wo ein Wert vom Typ 'xs:decimal' erwartet wird, behält der Wert den Typ 'xs:integer' bei.

Im folgenden Beispiel vergleicht die Funktion 'fn:compare' einen Wert vom Typ 'xs:string' mit einem Wert vom Typ 'xs:NCName':

```
fn:compare("product", xs:NCName("product"))
```

Der zurückgegebene Wert ist 0, was bedeutet, dass sich die Argumente beim Vergleich als identisch herausgestellt haben. Obwohl die Funktion 'fn:compare' Argumente vom Typ 'xs:string' erwartet, kann die Funktion mit einem Wert vom Typ 'xs:NCNAME' aufgerufen werden, da dieser Typ von 'xs:string' abgeleitet ist.

Die Subtypsubstitution wird immer dann verwendet, wenn an einen Ausdruck ein Wert übergeben wird, der von einem erwarteten Typ abgeleitet ist.

## Typumstufung

Bei der *Typumstufung* wird ein atomarer Wert von seinem ursprünglichen Typ in den von einem Ausdruck erwarteten Wert umgesetzt. XQuery verwendet die Typumstufung während der Auswertung von Funktionsaufrufen, von Klauseln des Typs **ORDER BY** und von Operatoren, die sowohl numerische Operanden als auch Zeichenfolgeoperanden als Eingabe akzeptieren.

Die folgenden Typumstufungen sind in XQuery zulässig:

**Numerische Typumstufung:**

Ein Wert vom Typ 'xs:float' (oder einem beliebigen durch RESTRICTION von 'xs:float' abgeleiteten Typ) wird in den Typ 'xs:double' umgestuft. Das Ergebnis ist ein Wert vom Typ 'xs:double', der dem ursprünglichen Wert entspricht.

Ein Wert vom Typ 'xs:decimal' (oder einem beliebigen durch RESTRICTION von 'xs:decimal' abgeleiteten Typ) kann entweder in den Datentyp 'xs:float' oder 'xs:double' umgestuft werden. Das Ergebnis dieser Umstufung wird erstellt, indem der ursprüngliche Wert in den erforderlichen Typ umgesetzt wird. Diese Art der Umstufung kann zur einem

Verlust an Genauigkeit führen.
Im folgenden Beispiel wird eine Sequenz, die den Wert 13.54e-2 vom Typ 'xs:double' und den Wert 100 vom Typ 'xs:decimal' enthält, an die Funktion 'fn:sum' übergeben, die einen Wert vom Typ 'xs:double' zurückgibt:

```
fn:sum(xs:double(13.54e-2), xs:decimal(100))
```

**URI-Typumstufung:**
Ein Wert vom Typ 'xs:anyURI' (oder einem beliebigen durch RESTRICTION von 'xs:anyURI' abgeleiteten Typ) kann in den Typ 'xs:string' umgestuft werden. Das Ergebnis dieser Umstufung wird erstellt, indem der ursprüngliche Wert in den Typ 'xs:string' umgesetzt wird.

Im folgenden Beispiel wird der URI-Wert in den erwarteten Typ 'xs:string' umgesetzt, und die Funktion gibt den Wert 18 zurück:

```
fn:string-length(xs:anyURI("http://example.com"))
```

Hierbei ist zu beachten, dass sich die Typumstufung und die Subtypsubstitution in folgenden Aspekten unterscheiden:

- Bei der Typumstufung wird der atomare Wert tatsächlich von seinem ursprünglichen Typ in den von einem Ausdruck erwarteten Typ umgesetzt.
- Bei der Subtypsubstitution kann ein Ausdruck, der einen bestimmten Typ erwartet, mit einem Wert aufgerufen werden, der von dem betreffenden Typ abgeleitet ist. Der Wert behält seinen ursprünglichen Typ jedoch bei.

## Effektiver Boolescher Wert

Der *effektive Boolesche Wert (EBW)* einer Sequenz wird implizit berechnet, während Ausdrücke, für die Boolesche Werte erforderlich sind, verarbeitet werden. Der EBW eines Werts wird ermittelt, indem die Funktion 'fn:boolean' auf den entsprechenden Wert angewandt wird.

In der folgenden Tabelle werden die effektiven Booleschen Werte beschrieben, die für bestimmte Wertetypen zurückgegeben werden.

*Tabelle 14. Für bestimmte Wertetypen in XQuery zurückgegebene effektive Boolesche Werte*

| Beschreibung des Wertes | Zurückgegebener EBW |
|---|---|
| Eine leere Sequenz | false (falsch) |
| Eine Sequenz mit einem Knoten als erstem Element | true (wahr) |
| Ein Einzelwert vom Typ 'xs:boolean' (oder von 'xs:boolean' abgeleitet) | false (falsch) - sofern der Wert von 'xs:boolean' 'false' ist |
| | true (wahr) - sofern der Wert von 'xs:boolean' 'true' ist |
| Ein Einzelwert vom Typ 'xs:string' oder 'xdt:untypedAtomic' (bzw. von einem von diesen Typen abgeleiteten Typ) | false (falsch) - sofern der Wert eine Nulllänge aufweist |
| | true (wahr) - sofern die Länge des Wertes größer als null ist |
| Ein Einzelwert eines numerischen Typs (oder von einem von einem numerischen Typ abgeleiteten Typ) | false (falsch) - sofern der Wert eine Nichtzahl (NaN) ist oder numerisch gleich null ist |
| | true (wahr) - sofern der Wert numerisch ungleich null ist |
| Alle anderen Werte | error (Fehler) |

*Tabelle 14. Für bestimmte Wertetypen in XQuery zurückgegebene effektive Boolesche Werte (Forts.)*

| Beschreibung des Wertes | Zurückgegebener EBW |
|---|---|
| **Anmerkung:** Der effektive Boolesche Wert einer Sequenz, die mindestens einen Knoten und mindestens einen atomaren Wert enthält, ist in einer Abfrage mit unvorhersehbarer Reihenfolge nicht deterministisch. | |

Der effektive Boolesche Wert einer Sequenz wird implizit berechnet, wenn die folgenden Ausdruckstypen verarbeitet werden:

- Logische Ausdrücke (**and**, **or**)
- Die Funktion 'fn:not'
- Die Klausel **WHERE** eines FLWOR-Ausdrucks
- Bestimmte Vergleichselementtypen wie beispielsweise `a[b]`
- Bedingungsausdrücke (**if**)
- Qualifizierte Ausdrücke (**some**, **every**)

# Primärausdrücke

Primärausdrücke sind die Basiselemente der Sprache. Zu ihnen gehören Literale, Variablenverweise, Klammerausdrücke, Kontextelementausdrücke, Konstruktoren und Funktionsaufrufe.

## Literale

Ein *Literal* ist eine direkte syntaktische Darstellung eines atomaren Wertes. DB2 XQuery unterstützt zwei Typen von Literalen: numerische Literale und Zeichenfolgeliterale.

Ein *numerisches Literal* ist ein atomarer Wert vom Datentyp 'xs:integer', 'xs:decimal' oder 'xs:double':

- Ein numerisches Literal, das keinen Punkt (.) als Dezimalzeichen und kein Zeichen 'e' oder 'E' enthält, ist ein atomarer Wert vom Typ 'xs:integer'. Der Wert `12` beispielsweise ist ein numerisches Literal.
- Ein numerisches Literal, das einen Punkt (.) als Dezimalzeichen, aber kein Zeichen 'e' oder 'E' enthält, ist ein atomarer Wert vom Typ 'xs:decimal'. Der Wert `12.5` beispielsweise ist ein numerisches Literal.
- Ein numerisches Literal, das ein Zeichen 'e' oder 'E' enthält, ist ein atomarer Wert vom Typ 'xs:double'. Der Wert `125E2` beispielsweise ist ein numerisches Literal.

Werte numerischer Literale werden gemäß den Regeln von XML-Schemata interpretiert.

Ein *Zeichenfolgeliteral* ist ein atomarer Wert vom Typ 'xs:string', der in einfache Anführungszeichen (') oder doppelte Anführungszeichen (") als Begrenzer eingeschlossen ist. Zeichenfolgeliterale können vordefinierte Entitätsverweise und Zeichenverweise enthalten. Die folgenden Zeichenfolgen beispielsweise sind gültige Zeichenfolgeliterale:

```
"12.5"
"He said, ""Let it be."""
'She said: "Why should I?"'
"Ben &amp;amp; Jerry&amp;apos;s"
"&amp;#8364;65.50" (: denotes the string €65.50 :)
```

**Tipp:** Um ein einfaches Anführungszeichen in ein Zeichenfolgeliteral einzuschließen, das durch einfache Anführungszeichen begrenzt ist, müssen zwei benachbarte einfache Anführungszeichen angegeben werden. Ebenso gilt: Um ein doppeltes Anführungszeichen in ein Zeichenfolgeliteral einzuschließen, das durch doppelte Anführungszeichen begrenzt ist, müssen zwei benachbarte doppelte Anführungszeichen angegeben werden.

Innerhalb eines Zeichenfolgeliterals werden Zeilenenden gemäß den Regeln von *XML 1.0 (Third Edition)* normalisiert. Jede Sequenz aus zwei Zeichen, die ein Rücklaufzeichen (X'0D') gefolgt von einem Zeilenvorschubzeichen (X'0A') enthält, wird in ein einzelnes Zeilenvorschubzeichen (X'0A') umgesetzt. Jedes Rücklaufzeichen (X'0D'), auf das kein Zeilenvorschubzeichen (X'0A') folgt, wird in ein einzelnes Zeilenvorschubzeichen (X'0A') umgesetzt.

Verfügt der Wert, der instanziiert werden soll, über keine Literaldarstellung, können Sie eine Konstruktorfunktion oder integrierte Funktion verwenden, um den Wert zurückzugeben. Die folgenden Funktionen und Konstruktoren geben Werte zurück, die keine Literaldarstellung aufweisen:

- Die integrierten Funktionen 'fn:true()' und 'fn:false()' geben die Booleschen Werte `true` (wahr) bzw. `false` (falsch) zurück. Diese Werte können auch von den Konstruktorfunktionen 'xs:boolean(″false″)' bzw. 'xs:boolean(″true″)' zurückgegeben werden.
- Die Konstruktorfunktion `xs:date("2005-04-16")` gibt ein Element vom Datentyp 'xs:date' zurück, dessen Wert das Datum 16. April 2005 darstellt.
- Die Konstruktorfunktion `xdt:dayTimeDuration("PT4H")` gibt ein Element vom Datentyp 'xdt:dayTimeDuration' zurück, dessen Wert eine Dauer von vier Stunden darstellt.
- Die Konstruktorfunktion `xs:float("NaN")` gibt den Sondergleitkommawert ″Not a Number″ (NaN) zurück.
- Die Konstruktorfunktion `xs:double("INF")` gibt den Sonderwert mit doppelter Genauigkeit ″positive infinity″ zurück.

## Vordefinierte Entitätsverweise

Ein *vordefinierter Entitätsverweis* ist eine kurze Zeichenfolge, die ein Zeichen darstellt, das in DB2 XQuery von syntaktischer Bedeutung ist. Ein vordefinierter Entitätsverweis beginnt mit einem Et-Zeichen (&) und endet mit einem Semikolon (;). Bei der Verarbeitung eines Zeichenfolgeliterals wird jeder vordefinierte Entitätsverweis durch die von ihm dargestellten Zeichen ersetzt.

Die folgende Tabelle enthält eine Liste der vordefinierten Entitätsverweise, die von DB2 XQuery erkannt werden.

*Tabelle 15. Vordefinierte Entitätsverweise in DB2 XQuery*

| Entitätsverweis | Dargestellte Zeichen |
|---|---|
| < | < |
| > | > |
| & | & |
| " | ″ |
| ' | ' |

### Zeichenverweise

Ein *Zeichenverweis* ist ein XML-Verweis auf ein Unicode-Zeichen, das durch seinen dezimalen oder hexadezimalen Codepunkt identifiziert wird.

Ein Zeichenverweis beginnt entweder mit `&#x` oder `&#` und endet mit einem Semikolon (;). Beginnt der Zeichenverweis mit `&#x`, liefern die Ziffern und Buchstaben vor dem abschließenden Semikolon (;) eine hexadezimale Darstellung des Codepunktes des Zeichens entsprechend der Norm *ISO/IEC 10646*. Beginnt der Zeichenverweis mit `&#`, liefern die Ziffern und Buchstaben vor dem abschließenden Semikolon (;) eine dezimale Darstellung des Codepunktes des Zeichens.

#### Beispiel

Der Zeichenverweis `€` oder `€` stellt das Euro-Symbol (€) dar.

## Variablenverweise

Ein Variablenverweis ist ein Name ohne Doppelpunkte (NCName), vor dem ein Dollarzeichen ($) steht. Bei der Auswertung einer Abfrage wird jeder Variablenverweis in den an die betreffende Variable gebundenen Wert aufgelöst. Jeder Variablenverweis muss zum Zeitpunkt des Verweises einem Namen unter den gültigen Variablen entsprechen.

Es gibt folgende Möglichkeiten, um Variablen den gültigen Variablen hinzuzufügen:

- Eine Variable kann den gültigen Variablen mithilfe der Funktion XMLQUERY, XMLTABLE oder XMLEXISTS über die Hostprogrammiersprachenumgebung SQL/XML hinzugefügt werden. Eine Variable, die über SQL/XML hinzugefügt wird, gilt für die gesamte Abfrage, sofern die Variable nicht durch eine andere Bindung derselben Variablen in einem XQuery-Ausdruck überschrieben wird.
- Eine Variable kann von einem XQuery-Ausdruck an einen Wert gebunden werden. Variablen können von FLWOR-Ausdrücken und von quantifizierten Ausdrücken gebunden werden. Funktionsaufrufe binden Variablen auch an die Formalparameter von Funktionen, bevor der Funktionsteil ausgeführt wird. Eine Variable, die von einem XQuery-Ausdruck gebunden wird, gilt für den gesamten Ausdruck, für den sie gebunden ist.

  Ein Variablenname darf in einem FLWOR-Ausdruck jeweils nur einmal deklariert werden. So wird beispielsweise der folgende Ausdruck von DB2 XQuery nicht unterstützt:

  ```
  for $i in (1, 2)
  for $i in ("a", "b")
     return $i
  ```

Stimmt ein Variablenverweis mit zwei oder mehr gültigen Variablenbindungen überein, dann bezieht sich der Verweis auf die innere Bindung (die Bindung mit dem kleineren Geltungsbereich).

**Tipp:** Um die Lesbarkeit von Code zu verbessern, sollten für Variablen in einer Abfrage stets eindeutige Namen verwendet werden.

### Beispiel

Im folgenden Beispiel bindet ein FLWOR-Ausdruck die Variable `$seq` an die Sequenz (10, 20, 30):

```
let $seq := (10, 20, 30)
return $seq[2];
```

Der zurückgegebene Wert lautet '20'.

## Klammerausdruck

Mithilfe von runden Klammern kann in Ausdrücken, die mehrere Operatoren enthalten, eine bestimmte Reihenfolge bei der Auswertung erzwungen werden.

Beispiel: Der Ausdruck `(2 + 4) * 5` ergibt bei der Auswertung den Wert 30, da der Klammerausdruck `(2 + 4)` zuerst ausgewertet und dessen Ergebnis anschließend mit fünf multipliziert wird. Ohne runde Klammern würde der Ausdruck `2 + 4 * 5` den Wert 22 ergeben, da der Multiplikationsoperator eine höhere Vorrangstellung hat als der Additionsoperator.

Leere runde Klammern geben eine leere Sequenz an.

## Kontextelementausdrücke

Ein Kontextelementausdruck besteht aus einem einzelnen Punkt (.). Ein Kontextelementausdruck wertet jeweils das momentan verarbeitete Element aus, das als *Kontextelement* bezeichnet wird. Das Kontextelement kann entweder ein Knoten oder ein atomarer Wert sein. Kontextelemente werden nur in Pfadausdrücken und Vergleichselementausdrücken definiert.

### Beispiel

Das folgende Beispiel enthält einen Kontextelementausdruck, der den Modulusoperator für jedes Element in der Sequenz aufruft, das vom Bereichsausdruck `1 to 100` zurückgegeben wird:

```
(1 to 100)[. mod 5 eq 0]
```

Das Ergebnis dieses Beispiels ist eine Sequenz von ganzen Zahlen (Integern) zwischen 1 und 100, die glatt durch 5 teilbar sind.

## Funktionsaufrufe

Ein Funktionsaufruf besteht aus einem qualifizierten Namen (QName), auf den eine in runde Klammern eingeschlossene Liste aus null oder mehr Ausdrücken - den so genannten Argumenten - folgt. DB2 XQuery unterstützt Aufrufe an integrierte XQuery-Funktionen sowie an DB2-spezifische Funktionen.

Integrierte XQuery-Funktionen befinden sich im Namensbereich `http://www.w3.org/2005/xpath-functions`, der an das Präfix `fn` gebunden ist. DB2-spezifische Funktionen befinden sich im Namensbereich `http://www.ibm.com/xmlns/prod/db2/functions`, der an das Präfix `db2-fn` gebunden ist. Verfügt der im Funktionsaufruf angegebene QName über kein Namensbereichspräfix, muss sich die Funktion im Standardfunktionsnamensbereich befinden. Der Standardfunktionsnamensbereich ist der Namensbereich der integrierten XQuery-Funktionen (die an das Präfix `fn` gebunden sind), sofern der Namensbereich nicht durch die Deklaration einer Standardfunktion im Abfrageprolog überschrieben wird.

**Wichtig:** Da die Argumente eines Funktionsaufrufs durch Kommata getrennt werden, müssen die Argumentausdrücke, die Kommaoperatoren auf erster Ebene enthalten, in runde Klammern eingeschlossen werden.

Anhand der folgenden Schritte wird erläutert, wie DB2 XQuery bei der Auswertung von Funktionen vorgeht:

1. DB2 XQuery wertet jeden Ausdruck aus, der als Argument im Funktionsaufruf übergeben wird, und gibt für jeden Ausdruck einen Wert zurück.
2. Der für jedes Argument zurückgegebene Wert wird in den Datentyp konvertiert, der von dem betreffenden Argument erwartet wird. Wenn das Argument einen atomaren Wert oder eine Sequenz aus atomaren Werten erwartet, verwendet DB2 XQuery die folgenden Regeln, um den Wert des Arguments in den entsprechenden erwarteten Typ zu konvertieren:
   a. Auf den betreffenden Wert wird die Atomisierung angewandt. Dies führt zu einer Sequenz aus atomaren Werten.
   b. Jedes Element in der atomaren Sequenz, das den Typ 'xdt:untypedAtomic' aufweist, wird in den erwarteten atomaren Typ umgesetzt. Für integrierte Funktionen, die numerische Argumente erwarten, werden Argumente vom Typ 'xdt:untypedAtomic' in den Typ 'xs:double' umgesetzt.
   c. Auf jedes numerische Element in der atomaren Sequenz, das in den erwarteten atomaren Typ umgestuft werden kann, wird die numerische Typumstufung angewandt. Zu den numerischen Elementen gehören Elemente des Typs 'xs:integer' (oder eines von 'xs:integer' abgeleiteten Typs), 'xs:decimal', 'xs:float' und 'xs:double'.
   d. Ist der erwartete Typ 'xs:string' wird jedes Element in der atomaren Sequenz, das vom Typ 'xs:anyURI' (oder von einem von 'xs:anyURI' abgeleiteten Typ) ist, in den Typ 'xs:string' umgestuft.
3. Die Funktion wird unter Verwendung der konvertierten Werte ihrer Argumente ausgewertet. Das Ergebnis des Funktionsaufrufs ist entweder eine Instanz des deklarierten Rückgabetyps der Funktion oder ein Fehler.

### Beispiele

**Funktionsaufruf mit einem Zeichenfolgeargument:** Der folgende Funktionsaufruf enthält ein Argument vom Typ 'xs:string' und gibt einen Wert vom Typ 'xs:string' zurück, in dem alle Zeichen in Großschreibung angegeben werden:

```
fn:upper-case($ns1_customerinfo/ns1:addr/@country)
```

In diesem Beispiel ist das an die Funktion 'fn:upper-case' übergebene Argument ein Pfadausdruck. Beim Aufrufen der Funktion wird der Pfadausdruck ausgewertet, und die resultierende Knotensequenz wird atomisiert. Jeder atomare Wert in der Sequenz wird in den erwarteten Datentyp 'xs:string' umgesetzt. Die Funktion wird ausgewertet und gibt eine Sequenz aus atomaren Werten des Typs 'xs:string' zurück.

**Funktionsaufruf mit einem Sequenzargument:** Die folgende Funktion enthält als einziges Argument die Sequenz (1, 2, 3):

```
fn:max((1, 2, 3))
```

Da die Funktion 'fn:max' ein einzelnes Argument in Form einer Sequenz aus atomaren Werten erwartet, sind verschachtelte runde Klammern erforderlich. Der zurückgegebene Wert ist 3.

## Pfadausdrücke

*Pfadausdrücke* identifizieren Knoten innerhalb einer XML-Baumstruktur. Pfadausdrücke in DB2 XQuery basieren auf der Syntax von XPath 2.0.

Ein Pfadausdruck besteht aus einem Schritt oder aus mehreren Schritten, die durch einen einfachen Schrägstrich (/) oder doppelte Schrägstriche (//) voneinander

getrennt werden. Ein Pfadausdruck kann mit einem Schritt oder mit einem Schrägstrich bzw. doppeltem Schrägstrich beginnen. Jeder Schritt vor dem abschließenden letzten Schritt generiert eine Sequenz aus Knoten, die als Kontextknoten für den jeweils folgenden Schritt verwendet werden.

Der erste Schritt gibt den Ausgangspunkt des Pfads an. Hierfür wird häufig ein Funktionsaufruf oder ein Variablenverweis verwendet, der einen Knoten oder eine Knotensequenz zurückgibt. Ein Schrägstrich ("/") am Anfang gibt an, dass der Pfad am Rootknoten der Baumstruktur beginnt, die den Kontextknoten enthält. Ein doppelter Schrägstrich ("//") am Anfang gibt an, dass der Pfad mit einer anfänglichen Knotensequenz beginnt, die aus dem Rootknoten der Baumstruktur, die den Kontextknoten enthält, sowie aus allen untergeordneten Elementen (Nachkommen) des Rootknotens besteht.

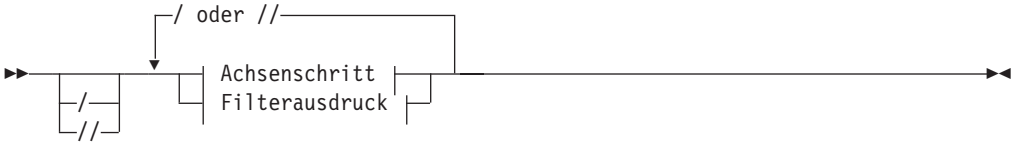
Jeder Schritt wird wiederholt ausgeführt - einmal für jeden Kontextknoten, der vom jeweils vorherigen Schritt generiert wurde. Die Ergebnisse dieser wiederholten Ausführungen werden anschließend kombiniert und bilden die Sequenz aus Kontextknoten für den jeweils folgenden Schritt. Doppelte Knoten werden aufgrundlage der Knoten-ID aus dieser kombinierten Sequenz entfernt.

Der Wert des Pfadausdrucks entspricht der kombinierten Elementsequenz, die aus dem letzten Schritt im Pfad resultiert. Dieser Wert kann entweder eine Sequenz aus Knoten oder eine Sequenz aus atomaren Werten sein. Da jeder Schritt im Pfad Kontextknoten für den jeweils folgenden Schritt bereitstellt, ist der letzte Schritt im Pfad der einzige Schritt, der eine Sequenz aus atomaren Werten zurückgeben kann. Ein Pfadausdruck, der eine Mischung aus Knoten und atomaren Werten zurückgibt, verursacht einen Fehler.

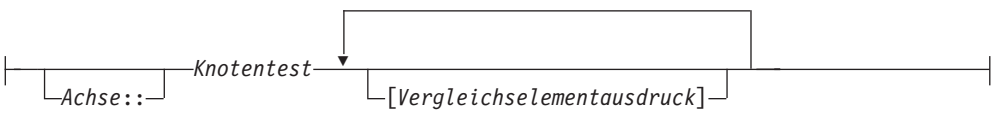
Die Knotensequenz, die aus einem Pfadausdruck resultiert, liegt nicht notwendigerweise in einer bestimmten Reihenfolge vor. Der Abschnitt zur Reihenfolge von Ergebnissen in XQuery-Ausdrücken enthält Informationen dazu, wann ein Pfadausdruck sortierte Ergebnisse zurückgibt.

## Syntax von Pfadausdrücken

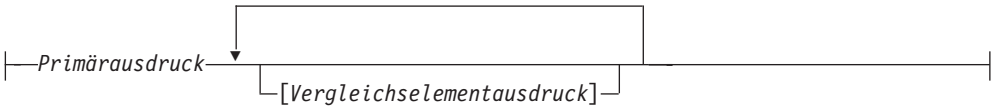
Jeder Schritt eines Pfadausdrucks ist entweder ein Achsenschritt oder ein Filterausdruck. Ein *Achsenschritt* gibt eine Sequenz aus Knoten zurück, die über eine angegebene Achse vom Kontextknoten aus erreichbar sind. Ein *Filterausdruck* besteht aus einem Primärausdruck, auf den null oder mehr Vergleichselemente folgen.




**Achsenschritt:**



**Filterausdruck:**




/ Ein Schrägstrich (/) am Anfang gibt an, dass der Pfad am Rootknoten (dies muss ein Dokumentknoten sein) der Baumstruktur beginnt, die den Kontextknoten enthält. Schrägstriche innerhalb eines Pfadausdrucks dienen zum Trennen einzelner Schritte.

// Ein doppelter Schrägstrich (//) am Anfang gibt an, dass der Pfad mit einer anfänglichen Knotensequenz beginnt, die aus dem Rootknoten (dies muss ein Dokumentknoten sein) der Baumstruktur, die den Kontextknoten enthält, sowie aus allen untergeordneten Elementen (Nachkommen) des Rootknotens besteht. Informationen zur Bedeutung eines doppelten Schrägstrichs zwischen einzelnen Schritten enthält der Abschnitt zur abgekürzten Syntax.

*Achse*
Gibt eine Bewegungsrichtung durch ein XML-Dokument oder -Fragment an. Folgende Achsen werden unterstützt: 'child', 'descendant', 'attribute', 'self', 'descendant-or-self' und 'parent'. Einige dieser Achsen können durch eine abgekürzte Syntax dargestellt werden.

*Knotentest*
Eine Bedingung, die für jeden Knoten, der von einem Achsenschritt ausgewählt ist, als wahr ('true') zutreffen muss. Der Test kann entweder ein Namenstest sein, bei dem Knoten aufgrundlage ihres Namens ausgewählt werden, oder ein Sortentest, bei dem Knoten aufgrundlage ihrer Sorte ausgewählt werden.

*Primärausdruck*
Ein Primärausdruck.

*Vergleichselementausdruck*
Ein Ausdruck, der festlegt, ob Elemente der Sequenz beibehalten oder verworfen werden.

### Beispiele

Das nachstehende Beispiel zeigt einen Achsenschritt mit zwei Vergleichselementen. Dieser Schritt wählt aus dem Kontextknoten alle direkt untergeordneten Elemente (Kinder) vom Typ `employee` aus, die sowohl ein Kindelement `secretary` als auch ein Kindelement `assistant` aufweisen:

```
child::employee[secretary][assistant]
```

In nachstehendem Beispiel wird ein Filterausdruck als Schritt in einem Pfadausdruck verwendet. Der Ausdruck gibt aus einem angegebenen Buch (book) jedes Kapitel (chapter) bzw. jeden Anhang (appendix) zurück, das. bzw. der mehr als eine Fußnote (footnote) enthält:

```
$book/(chapter | appendix)[fn:count(footnote)> 1]
```

## Achsenschritte

Ein *Achsenschritt* besteht aus drei Teilen: einer optionalen *Achse*, die eine Bewegungsrichtung angibt, einem *Knotentest*, der die zur Auswahl von Knoten ver-

wendeten Kriterien angibt, und null oder mehr *Vergleichselemente* zum Filtern der vom Schritt zurückgegebenen Sequenz. Das Ergebnis eines Achsenschritts ist stets eine Sequenz aus null oder mehr Knoten.

Ein Achsenschritt kann entweder ein *Vorwärtsschritt* sein, der am Kontextknoten beginnt und in Dokumentreihenfolge durch die XML-Baumstruktur verläuft, oder ein *Rückwärtsschritt*, der am Kontextknoten beginnt und entgegen der Dokumentreihenfolge durch die XML-Baumstruktur verläuft. Wenn das Kontextelement kein Knoten ist, führt der Ausdruck zu einem Fehler.

Die nicht abgekürzte Syntax für einen Achsenschritt besteht aus einem Achsennamen und einem Knotentest, die durch einen Doppelpunkt getrennt sind, gefolgt von null oder mehr Vergleichselementen. Die Syntax eines Achsenausdrucks kann abgekürzt werden, indem die Achse ausgelassen und eine verkürzte Schreibweise verwendet wird.

In folgendem Beispiel ist `child` der Name der Achse und `para` ist der Name des Elementknotens, der in dieser Achse ausgewählt werden soll.

```
child::para
```

Der Achsenschritt in diesem Beispiel wählt alle Elemente `para` aus, die Kinder des Kontextknotens sind.

## Achsen

Eine *Achse* ist Teil eines Achsenschritts und gibt die Bewegungsrichtung durch ein XML-Dokument an.

Eine Achse kann entweder eine Vorwärts- oder Rückwärtsachse sein. Eine *Vorwärtsachse* enthält den Kontextknoten sowie Knoten, die in der Dokumentreihenfolge nach dem Kontextknoten stehen. Eine *Vorwärtsachse* enthält den Kontextknoten sowie Knoten, die in der Dokumentreihenfolge vor dem Kontextknoten stehen.

In der folgenden Tabelle werden die in DB2 XQuery unterstützten Achsen beschrieben.

*Tabelle 16. In DB2 XQuery unterstützte Achsen*

| Achse | Beschreibung | Richtung | Anmerkungen |
|---|---|---|---|
| child | Gibt die (direkt) untergeordneten Elemente (Kinder) des Kontextknotens zurück. | Vorwärts | Dokumentknoten und Elementknoten sind die einzigen Knoten mit Kindern. Weist der Kontextknoten eine andere Knotensorte auf oder ist der Kontextknoten ein Dokument bzw. Element ohne Kinder, ist die Kindachse 'child' eine leere Sequenz. Die Kinder eines Dokument- oder Elementknotens können Element-, Verarbeitungsanweisungs-, Kommentar- oder Textknoten sein. Attribut- und Dokumentknoten können nie als Kinder angezeigt werden. |
| descendant | Gibt die untergeordneten Elemente (Nachkommen) des Kontextknotens an (die Kinder, die Kindeskinder usw.). | Vorwärts | |
| attribute | Gibt die Attribute des Kontextknotens zurück. | Vorwärts | Diese Achse ist leer, wenn der Kontextknoten kein Elementknoten ist. |

*Tabelle 16. In DB2 XQuery unterstützte Achsen (Forts.)*

| Achse | Beschreibung | Richtung | Anmerkungen |
|---|---|---|---|
| self | Gibt ausschließlich den Kontextknoten selbst zurück. | Vorwärts | |
| descendant-or-self | Gibt den Kontextknoten selbst und die Nachkommen des Kontextknotens zurück. | Vorwärts | |
| parent | Gibt das übergeordnete Element (Elter) des Kontextknotens zurück bzw. eine leere Sequenz, wenn der Kontextknoten kein übergeordnetes Element aufweist. | Rückwärts | Ein Elementknoten kann das übergeordnete Element eines Attributknotens sein, wenngleich ein Attributknoten niemals ein Kind eines Elementknotens ist. |

Wenn ein Achsenschritt eine Knotensequenz auswählt, wird jedem Knoten eine Kontextposition zugeordnet, die der Position des betreffenden Knotens in der Sequenz entspricht. Bei Vorwärtsachsen werden die Kontextpositionen den Knoten in Dokumentreihenfolge zugeordnet, beginnend mit 1. Bei Rückwärtsachsen werden die Kontextpositionen den Knoten in umgekehrter Dokumentreihenfolge zugeordnet, beginnen mit 1. Anhand der Zuordnungen von Kontextpositionen können Sie einen Knoten durch Angabe seiner entsprechenden Position aus der Sequenz auswählen.

## Knotentests

Ein *Knotentest* ist eine Bedingung, die für jeden Knoten, der von einem Achsenschritt ausgewählt ist, als wahr ('true') zutreffen muss. Ein Knotentest kann entweder als Namenstest oder als Sortentest ausgedrückt werden. Bei einem *Namenstest* werden Knoten anhand ihres Namens ausgewählt. Bei einem *Sortentest* werden Knoten anhand ihrer Sorte ausgewählt.

### Namenstests

Ein Namenstest besteht aus einem qualifizierten Namen (QName) oder einem Platzhalterzeichen. Wird ein Namenstest in einem Achsenschritt angegeben, wählt der Schritt diejenigen Knoten auf der angegebenen Achse aus, die dem qualifizierten Namen oder dem Platzhalterzeichen entsprechen. Wird der Namenstest auf der Attributachse angegeben, wählt der Schritt alle Attribute aus, die mit dem Namenstest übereinstimmen. Auf allen anderen Achsen wählt der Schritt beliebige Elemente aus, die mit dem Namenstest übereinstimmen. Die qualifizierten Namen stimmen überein, wenn der erweiterte qualifizierte Name des Knoten dem im Namenstest angegebenen erweiterten qualifizierten Namen auf Codepunktbasis entspricht. Zwei erweiterte qualifizierte Namen stimmen überein, wenn ihre Namensbereichs-URIs und ihre lokalen Namen einander jeweils entsprechen (selbst wenn ihre Namensbereichspräfixe nicht identisch sind).

**Wichtig:** Jedes in einem Namenstest angegebene Präfix muss einem der statisch bekannten Namensbereiche des Ausdrucks entsprechen. Bei Namenstests, die auf der Attributachse ausgeführt werden, haben qualifizierte Namen ohne Präfix keine Namensbereichs-URI. Bei Namenstests, die auf anderen Achsen ausgeführt werden, haben qualifizierte Namen ohne Präfix die Namensbereichs-URI des Standardnamensbereichs für Elemente/Typen.

In der folgenden Tabelle werden die in DB2 XQuery unterstützten Namenstests beschrieben.

*Tabelle 17. In DB2 XQuery unterstützte Namenstests*

| Test | Beschreibung | Beispiele |
|---|---|---|
| *QName* | Entspricht allen Knoten (auf der angegebenen Achse), deren qualifizierter Name (QName) mit dem angegebenen qualifizierten Namen übereinstimmt. Handelt es sich bei der Achse um eine Attributachse, werden mit diesem Test Attributknoten abgeglichen. Auf allen anderen Achsen werden mit diesem Test Elementknoten abgeglichen. | Im Ausdruck `child::para` wählt der Namenstest `para` alle Elemente vom Typ `para` auf der Kindachse (child) aus. |
| * | Entspricht allen Knoten auf der angegebenen Achse. Handelt es sich bei der Achse um eine Attributachse, werden mit diesem Test alle Attributknoten abgeglichen. Auf allen anderen Achsen werden mit diesem Test alle Elementknoten abgeglichen. | Im Ausdruck `child::*` entspricht der Namenstest * allen Elementen auf der Kindachse (child). |
| *NCName*:* | Gibt einen Namen ohne Doppelpunkte (*NCName*) an, der den Abschnitt mit dem Präfix eines qualifizierten Namens (QName) darstellt. Dieser Namenstest entspricht allen Knoten (auf der angegebenen Achse), deren Namensbereichs-URI mit der Namensbereichs-URI übereinstimmt, an die das Präfix gebunden ist. Handelt es sich bei der Achse um eine Attributachse, werden mit diesem Test Attributknoten abgeglichen. Auf allen anderen Achsen werden mit diesem Test Elementknoten abgeglichen. | Im Ausdruck `child::ns1:*` entspricht der Namenstest `ns1:*` allen Elementen auf der Kindachse (child), die dem Namensbereich zugeordnet sind, an den das Präfix `ns1` gebunden ist. |
| *:*NCName* | Gibt einen Namen ohne Doppelpunkte (*NCName*) an, der den Abschnitt mit dem lokalen Namen eines qualifizierten Namens (QName) darstellt. Dieser Namenstest entspricht allen Knoten (auf der angegebenen Achse), deren lokaler Name mit *NCName* übereinstimmt. Handelt es sich bei der Achse um eine Attributachse, werden mit diesem Test Attributknoten abgeglichen. Auf allen anderen Achsen werden mit diesem Test Elementknoten abgeglichen. | Im Ausdruck `child::*:customerinfo` entspricht der Namenstest `*:customerinfo` allen Elementen auf der Kindachse (child), die den lokalen Namen `customerinfo` aufweisen. Der dem Elementnamen zugeordnete Namensbereich spielt hierbei keine Rolle. |

## Sortentests

Wird ein Sortentest in einem Achsenschritt angegeben, wählt der Schritt lediglich diejenigen Knoten auf der angegebenen Achse aus, die dem Sortentest entsprechen. In der folgenden Tabelle werden die in DB2 XQuery unterstützten Sortentests beschrieben.

*Tabelle 18. In DB2 XQuery unterstützte Sortentests*

| Test | Beschreibung | Beispiele |
|---|---|---|
| node() | Entspricht allen Knoten auf der angegebenen Achse. | Im Ausdruck `child::node()` wählt der Sortentest `node()` alle Knoten auf der Kindachse (child) aus. |
| text() | Entspricht allen Testknoten auf der angegebenen Achse. | Im Ausdruck `child::text()` wählt der Sortentest `text()` alle Textknoten auf der Kindachse (child) aus. |
| comment() | Entspricht allen Kommentarknoten auf der angegebenen Achse. | Im Ausdruck `child::comment()` wählt der Sortentest `comment()` alle Kommentarknoten auf der Kindachse (child) aus. |
| processing-instruction() | Entspricht allen Verarbeitungsanweisungsknoten auf der angegebenen Achse. | Im Ausdruck `child::processing-instruction()` wählt der Sortentest `processing-instruction()` alle Verarbeitungsanweisungsknoten auf der Kindachse (child) aus. |
| element() oder element(*) | Entspricht allen Elementknoten auf der angegebenen Achse. | Im Ausdruck `child::element()` wählt der Sortentest `element()` alle Elementknoten auf der Kindachse (child) aus. Im Ausdruck `child::element(*)` wählt der Sortentest `element(*)` ebenfalls alle Elementknoten auf der Kindachse (child) aus. |
| attribute() oder attribute(*) | Entspricht allen Attributknoten auf der angegebenen Achse. | Im Ausdruck `child::attribute()` wählt der Sortentest `attribute()` alle Attributknoten auf der Kindachse (child) aus. Im Ausdruck `child::attribute(*)` wählt der Sortentest `attribute(*)` ebenfalls alle Attributknoten auf der Kindachse (child) aus. |
| document-node() | Entspricht allen Dokumentknoten auf der angegebenen Achse. | Im Ausdruck `self::document-node()` wählt der Sortentest `document-node()` einen Dokumentknoten aus, der sich im Kontextknoten befindet. |

## Abgekürzte Syntax für Pfadausdrücke

XQuery bietet eine abgekürzte Syntax zur Angabe von Achsen in Pfadausdrücken.

In der folgenden Tabellen werden die in Pfadausdrücken zulässigen Abkürzungen beschrieben.

*Tabelle 19. Abgekürzte Syntax für Pfadausdrücke*

| Abgekürzte Syntax | Beschreibung | Beispiele |
|---|---|---|
| Keine Angabe von Achsen | Kurzform für `child::`, es sei denn, im Achsenschritt wird `attribute()` für den Knotentest angegeben. Wenn im Achsenschritt ein Attributtest angegeben wird, ist eine ausgelassene Achse die Kurzform für `attribute::`. | Der Pfadausdruck `section/para` ist eine Abkürzung für `child::section/child::para`. Der Pfadausdruck `section/attribute()` ist eine Abkürzung für `child::section/attribute::attribute()`. |
| @ | Kurzform für `attribute::` . | Der Pfadausdruck `section/@id` ist eine Abkürzung für `child::section/attribute::id`. |
| // | Kurzform für `/descendant-or-self::node()/`, es sei denn, diese Abkürzung steht am Anfang des Pfadausdrucks.<br><br>Bei Angabe dieser Abkürzung am Anfang des Pfadausdrucks wählt der Achsenschritt eine Anfangsknotensequenz aus, die das Rootelement der Baumstruktur, in dem sich der Kontextknoten befindet, sowie alle diesem Rootelement untergeordneten Knoten enthält. Dieser Ausdruck gibt einen Fehler zurück, wenn es sich bei dem Rootknoten nicht um einen Dokumentknoten handelt. | Der Pfadausdruck `div1//para` ist eine Abkürzung für `child::div1/descendant-or-self::node()/child::para` . |
| .. | Kurzform für `parent::node()`. | Der Pfadausdruck `../title` ist eine Abkürzung für `parent::node()/child::title`. |

## Beispiele für abgekürzte und nicht abgekürzte Syntax

Die folgende Tabelle enthält Beispiele für abgekürzte und nicht abgekürzte Syntax.

*Tabelle 20. Nicht abgekürzte und abgekürzte Syntax*

| Nicht abgekürzte Syntax | Abgekürzte Syntax | Ergebnis |
|---|---|---|
| `child::para` | `para` | Wählt die `para`-Elemente aus, die direkt untergeordnete Elemente (Kinder) des Kontextknotens sind. |
| `child::*` | `*` | Wählt alle Elemente aus, die direkt untergeordnete Elemente (Kinder) des Kontextknotens sind. |
| `child::text()` | `text()` | Wählt alle Textknoten aus, die direkt untergeordnete Elemente (Kinder) des Kontextknotens sind. |
| `child::node()` | `node()` | Wählt alle direkt untergeordneten Elemente (Kinder) des Kontextknotens aus. Dieser Ausdruck gibt keine Attributknoten zurück, da Attributknoten nicht als Kinder eines Knotens gelten. |
| `attribute::name` | `@name` | Wählt das Attribut `name` des Kontextknotens aus. |

*Tabelle 20. Nicht abgekürzte und abgekürzte Syntax (Forts.)*

| Nicht abgekürzte Syntax | Abgekürzte Syntax | Ergebnis |
|---|---|---|
| `attribute::*` | `@*` | Wählt alle Attribute des Kontextknotens aus. |
| `child::para[fn:position() = 1]` | `para[1]` | Wählt das erste Element `para` aus, das ein direkt untergeordnetes Element (Kind) des Kontextknotens ist. |
| `child::para[fn:position() = fn:last()]` | `para[fn:last()]` | Wählt das letzte Element `para` aus, das ein direkt untergeordnetes Element (Kind) des Kontextknotens ist. |
| `/child::book/child::chapter[fn:position() = 5] /child::section[fn:position() = 2]` | `/book/chapter[5]/section[2]` | Wählt den zweiten Abschnitt (`section`) des fünften Kapitels (`chapter`) des Buches (`book`) aus, dessen übergeordnetes Element (Elter) der Dokumentknoten ist, der den Kontextknoten enthält. |
| `child::para[attribute::type="warning"]` | `para[@type="warning"]` | Wählt alle direkt untergeordneten Elemente (Kinder ) vom Typ `para` des Kontextknotens aus, die über ein Typattribut mit dem Wert `warning` für Warnung verfügen. |
| `child::para[attribute::type='warning'] [fn:position() = 5]` | `para[@type="warning"][5]` | Wählt das fünfte direkt untergeordnete Element (Kind) vom Typ `para` des Kontextknotens aus, das über ein Typattribut mit dem Wert `warning` für Warnung verfügt. |
| `child::para[fn:position() = 5] [attribute::type="warning"]` | `para[5][@type="warning"]` | Wählt das fünfte direkt untergeordnete Element (Kind) vom Typ `para` des Kontextknotens aus, sofern dieses Kind über ein Typattribut mit dem Wert `warning` für Warnung verfügt. |
| `child::chapter[child::title='Introduction']` | `chapter[title="Introduction"]` | Wählt die direkt untergeordneten Elemente (Kinder) vom Typ `chapter` (Kapitel) des Kontextknotens aus, die mindestens ein Kind vom Typ `title` (Titel) haben, dessen Typattributwert der Zeichenfolge `Introduction` (Einführung) entspricht. |

*Tabelle 20. Nicht abgekürzte und abgekürzte Syntax (Forts.)*

| Nicht abgekürzte Syntax | Abgekürzte Syntax | Ergebnis |
|---|---|---|
| `child::chapter[child::title]` | `chapter[title]` | Wählt die direkt untergeordneten Elemente (Kinder) vom Typ `chapter` (Kapitel) des Kontexknotens aus, die mindestens ein Kind vom Typ `title` (Titel) haben. |

## Vergleichselemente

Ein *Vergleichselement* dient zum Filtern einer Sequenz, indem die zulässigen Elemente zurückbehalten werden. Ein Vergleichselement besteht aus einem Ausdruck, der als Vergleichselementausdruck bezeichnet wird und in eckigen Klammern ([]) eingeschlossen ist. Der Vergleichselementausdruck wird einmal für jedes Element in der Sequenz ausgewertet, wobei das ausgewählte Element als Kontextelement verwendet wird. Jede Auswertung des Vergleichselementausdrucks gibt einen Wert vom Datentyp 'xs:boolean' zurück. Dieser Wert wird als *Wahrheitswert des Vergleichselements* bezeichnet. Diejenigen Elemente, bei denen der Wahrheitswert des Vergleichselements 'true' (wahr) lautet, werden zurückbehalten, und diejenigen Elemente, bei denen der Wahrheitswert des Vergleichselements 'false' (falsch) lautet, werden verworfen.

Der Wahrheitswert des Vergleichselements wird anhand der folgenden Regeln ermittelt:

- Wenn der Vergleichselementausdruck einen nicht numerischen Wert zurückgibt, ist der Wahrheitswert des Vergleichselements der effektive Boolesche Wert des Vergleichselementausdrucks.
- Wenn der Vergleichselementausdruck einen numerischen Wert zurückgibt, ist der Wahrheitswert des Vergleichselements nur für das Element wahr ('true'), dessen Position in der Sequenz mit dem numerischen Wert übereinstimmt. Für alle anderen Elemente ist der Wahrheitswert des Vergleichselements falsch ('false'). Diese Sorte von Vergleichselement wird als *numerisches Vergleichselement* oder *positionsgebundenes Vergleichselement* bezeichnet. In dem Ausdruck `$products[5]` beispielsweise behält das numerische Vergleichselement `[5]` lediglich das fünfte Element in der an die Variable `$products` gebundene Sequenz zurück.

  **Wichtig:** Das Element, das von einem numerischen Vergleichselement aus einer Sequenz ausgewählt wird, ist nur dann deterministisch, wenn die Sequenz in einer deterministischen Reihenfolge vorliegt.

**Tipp:** Das Verhalten eines Vergleichselements hängt davon ab, ob der Vergleichselementausdruck einen numerischen Wert zurückgibt oder nicht, was nicht unbedingt aus der Betrachtung des Vergleichselementausdrucks hervorgeht. Mithilfe der Funktion 'fn:boolean' kann erzwungen werden, dass ein Vergleichselement einen effektiven Booleschen Wert verwendet. Beispiel: `[fn:boolean(`*`Vergleichselementausdruck`*`)]`. Mithilfe der Funktion 'fn:position' hingegen kann erzwungen werden, dass sich ein Vergleichselement wie ein positionsgebundenes Vergleichselement verhält. Beispiel: `[fn:position() eq `*`Vergleichselementausdruck`*`]`.

Die folgenden Beispiele enthalten Vergleichselemente:

- `chapter[2]` wählt das zweite Element `chapter` aus, das ein direktes untergeordnetes Element (Kind) des Kontextknotens ist.
- `descendant::toy[@color = "Red"]` wählt alle untergeordneten Elemente (Nachkommen) des Kontextknotens aus, bei denen es sich um Elemente namens `toy` handelt, deren Farbattribut (color) den Wert ″Red″ für rot aufweist.
- `employee[secretary][assistant]` wählt alle direkt untergeordneten Elemente (Kinder) vom Typ 'employee' des Kontextknotens aus, die sowohl ein Kindelement 'secretary' als auch ein Kindelement 'assistant' aufweisen.
- `(<cat />, <dog />, 47, <zebra />)[2]` gibt das Element `<dog />` zurück.

# Sequenzausdrücke

Mithilfe von Sequenzausdrücken werden Sequenzen aus Elementen erstellt, gefiltert und kombiniert. Sequenzen sind nie verschachtelt. Beispiel: Die Kombination der Werte 1, (2, 3) und ( ) zu einer einzigen Sequenz ergibt die Sequenz '(1, 2, 3)'.

## Ausdrücke zum Erstellen von Sequenzen

Sequenzen können entweder mithilfe des Kommaoperators oder eines Bereichsausdrucks erstellt werden.

### Kommaoperatoren

Um eine Sequenz mithilfe des Kommaoperators zu erstellen, geben Sie mindestens zwei Operanden (Ausdrücke) an, die durch Kommata getrennt sind. Bei der Auswertung des Sequenzausdrucks evaluiert der Kommaoperator jeden seiner Operanden und verknüpft die sich ergebenden Sequenzen in ihrer Reihenfolge zu einer einzigen Ergebnissequenz. Der folgende Ausdruck beispielsweise ergibt eine Sequenz mit fünf ganzen Zahlen:

```
(15, 1, 3, 5, 7)
```

Eine Sequenz kann doppelte atomare Werte und Knoten enthalten. Eine Sequenz ist jedoch niemals ein Element in einer anderen Sequenz. Wird eine neue Sequenz durch Verknüpfung von mindestens zwei Eingabesequenzen erstellt, enthält die neue Sequenz alle Elemente der Eingabesequenzen, und die Länge der Sequenz ist die Summe der Längen der Eingabesequenzen.

Die folgenden Ausdrücke verwenden für die Sequenzerstellung den Kommaoperator:

- Dieser Ausdruck kombiniert vier Sequenzen mit einer Länge von eins, zwei, null bzw. zwei zu einer einzigen Sequenz mit einer Länge von fünf. Das Ergebnis dieses Ausdrucks ist die Sequenz 10, 1, 2, 3, 4.

  ```
  (10, (1, 2), (), (3, 4))
  ```
- Das Ergebnis dieses Ausdrucks ist eine Sequenz, die alle Elemente vom Typ `salary` (Gehalt) enthält, die Kinder des Kontextknotens sind, gefolgt von allen Elementen vom Typ `bonus`, die Kinder des Kontextknotens sind.

  ```
  (salary, bonus)
  ```
- Unter der Voraussetzung, dass die Variable '$price' an den Wert 10,50 gebunden ist, ergibt dieser Ausdruck die Sequenz 10,50, 10,50.

  ```
  ($price, $price)
  ```

### Bereichsausdrücke

Bereichsausdrücke erstellen eine Sequenz aufeinanderfolgender ganzer Zahlen. Ein Bereichsausdruck besteht aus zwei Operanden (Ausdrücken), die durch den Operator **to** (bis) getrennt werden. Der Wert jedes Operanden muss in einen Wert vom Typ 'xs:integer' konvertierbar sein. Handelt es sich bei einem der Operanden um eine leere Sequenz oder ist die vom ersten Operanden abgeleitete ganze Zahl größer als die vom zweiten Operanden abgeleitete ganze Zahl, ist das Ergebnis des Bereichsausdrucks eine leere Sequenz. Andernfalls ist das Ergebnis eine Sequenz, die die beiden ganzen Zahlen enthält, die von den Operanden abgeleitet werden, sowie jede ganze Zahl, die zwischen den betreffenden beiden ganzen Zahlen liegt (in aufsteigender Reihenfolge). Der folgende Bereichsausdruck beispielsweise ergibt bei der Auswertung die Sequenz 1, 2, 3, 4:

```
(1 to 4)
```

In den folgenden Beispielen werden für die Sequenzerstellung Bereichsausdrücke verwendet:

- Mit diesem Beispiel wird ein Bereichsausdruck als ein Operand bei der Erstellung einer Sequenz verwendet. Der Sequenzausdruck ergibt bei der Auswertung die Sequenz 10, 1, 2, 3, 4.

  ```
  (10, 1 to 4)
  ```

- Mit diesem Beispiel wird eine Sequenz mit der Länge eins erstellt, die lediglich die ganze Zahl 10 enthält.

  ```
  10 to 10
  ```

- Das Ergebnis dieses Beispiels ist eine Sequenz mit der Länge null.

  ```
  15 to 10
  ```

- In diesem Beispiel wird die Funktion 'fn:reverse' verwendet, um eine Sequenz mit sechs ganzen Zahlen in absteigender Reihenfolge zu erstellen. Dieser Sequenzausdruck ergibt bei der Auswertung die Sequenz 15, 14, 13, 12, 11, 10.

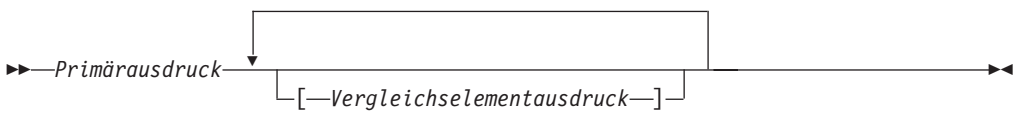
  ```
  fn:reverse(10 to 15)
  ```

## Filterausdrücke

Ein Filterausdruck besteht aus einem Primärausdruck, auf den null oder mehr Vergleichselemente folgen. Die Vergleichselemente (sofern vorhanden) filtern die vom Primärausdruck zurückgegebene Sequenz.

Das Ergebnis des Filterausdrucks besteht aus allen vom Primärausdruck zurückgegebenen Elementen, bei denen der Wahrheitswert der Vergleichselemente 'true' (wahr) lautet. Sind keine Vergleichselemente angegeben, entspricht das Ergebnis einfach dem Ergebnis des Primärausdrucks. Die Elemente in der Ergebnissequenz liegen in derselben Reihenfolge vor wie die vom Primärausdruck zurückgegebenen Elemente. Während der Auswertung eines Vergleichselements hat jedes Element eine Kontextposition, die die Position des Elements in der vom betreffenden Vergleichselement gefilterten Sequenz darstellt. Die erste Kontextposition ist 1.

### Syntax



*Primärausdruck*
Ein Primärausdruck.

*Vergleichselementausdruck*
Ein Ausdruck, der festlegt, ob Elemente der Sequenz beibehalten oder verworfen werden.

### Beispiele

In den folgenden Beispielen werden Filterausdrücke verwendet, um eine gefilterte Sequenz zurückzugeben:

- Bei Vorliegen einer an eine Variable gebundenen Sequenz aus Produkten (products) gibt dieser Ausdruck nur diejenigen Produkte mit einem Preis von mehr als 100 zurück:

  ```
  $products[price gt 100]
  ```

- In diesem Ausdruck wird ein Bereichsausdruck mit einem Vergleichselement verwendet, um alle ganzen Zahlen von 1 bis 100 aufzulisten, die durch 5 teilbar sind. Der Bereichsausdruck wird als Primärausdruck verarbeitet, da er in runde Klammern eingeschlossen ist:

  ```
  (1 to 100)[. mod 5 eq 0]
  ```

- Dieser Ausdruck gibt als Ergebnis die ganze Zahl 5 zurück:

  ```
  (1 to 21)[5]
  ```

- In diesem Ausdruck wird ein Filterausdruck als Schritt in einem Pfadausdruck verwendet. Der Ausdruck gibt das letzte Kapitel (chapter) oder den letzten Anhang (appendix) eines Buches zurück, das an die Variable '$book' gebunden ist:

  ```
  $book/(chapter | appendix)[fn:last()]
  ```

## Ausdrücke zum Kombinieren von Knotensequenzen

Mit DB2 XQuery werden Operatoren zum Kombinieren von Knotensequenzen bereitgestellt. Zu diesen Operatoren gehören **union**, **intersect** und **except**.

In der folgenden Tabelle werden die Operatoren beschrieben, die zum Kombinieren von Knotensequenzen zur Verfügung stehen.

*Tabelle 21. XQuery-Operatoren zum Kombinieren von Knotensequenzen*

| Operator | Beschreibung |
|---|---|
| **union** oder \| | Verwendet zwei Knotensequenzen als Operanden und gibt eine Sequenz mit sämtlichen Knoten zurück, die in den beiden Operanden vorkommen. Das Schlüsselwort **union** und das Zeichen \| sind äquivalent. |
| **intersect** | Verwendet zwei Knotensequenzen als Operanden und gibt eine Sequenz mit allen Knoten zurück, die in beiden Operanden gleichzeitig vorkommen. |
| **except** | Verwendet zwei Knotensequenzen als Operanden und gibt eine Sequenz mit allen Knoten zurück, die im ersten, aber nicht im zweiten Operanden vorkommen. |

Bei allen Operatoren werden doppelte Knoten anhand der Knoten-ID aus den Ergebnissequenzen entfernt. Die resultierenden Sequenzen werden jeweils in Dokumentreihenfolge zurückgegeben.

Die Operanden der Operatoren **union**, **intersect** bzw. **except** müssen sich in Sequenzen auflösen lassen, die ausschließlich Knoten enthalten. Wenn ein Operand außer Knoten noch einen anderen Elementtyp enthält, wird ein Fehler zurückgegeben.

Zusätzlich zu den in vorliegendem Abschnitt erörterten Operatoren stellt DB2 XQuery Funktionen für den indexierten Zugriff auf Elemente oder Untersequenzen einer Sequenz (fn:index-of), für das indexierte Einfügen oder Entfernen von Elementen in eine bzw. aus einer Sequenz ('fn:insert-before' und 'fn:remove') sowie für das Entfernen doppelter Elemente aus einer Sequenz (fn:distinct-values) zur Verfügung.

### Beispiele

In den nachstehenden Beispielen wird davon ausgegangen, dass die Variable `$managers` an eine Gruppe von Knoten des Typs 'employee' gebunden ist, die die Mitarbeiter mit der Funktion eines Managers darstellen, und dass die Variable `$students` an eine Gruppe von Knoten des Typs 'employee' gebunden ist, die die studentischen Mitarbeiter darstellen.

Die folgenden Ausdrücke sind alle gültige Beispiele mit Operatoren zum Kombinieren von Knotensequenzen:

- Der Ausdruck `$managers union $students` gibt die Gruppe derjenigen Knoten zurück, die Mitarbeiter darstellen, die entweder Manager oder Studenten sind.
- Der Ausdruck `$managers intersect $students` gibt die Gruppe derjenigen Knoten zurück, die Mitarbeiter darstellen, die sowohl Manager als auch Studenten sind.
- Der Ausdruck `$managers except $students` gibt die Gruppe derjenigen Knoten zurück, die Mitarbeiter darstellen, die Manager, aber keine Studenten sind.

## Arithmetische Ausdrücke

Arithmetische Ausdrücke führen Operationen wie beispielsweise Addition, Subtraktion, Multiplikation, Division und Modulus aus.

In der nachstehenden Tabelle werden die arithmetischen Operatoren beschrieben und anhand der Vorrangstellung für Operatoren von hoch nach niedrig sortiert angegeben. Monadische Operatoren (unary) haben eine höhere Vorrangstellung als Binäroperatoren, sofern keine runden Klammern verwendet werden, um die Auswertung eines Binäroperators zu erzwingen.

*Tabelle 22. Arithmetische Operatoren in XQuery*

| Operator | Zweck | Assoziativität |
|---|---|---|
| -(unary), +(unary) | Negiert bzw. erhält den Operandenwert | Von rechts nach links |
| *, div, idiv, mod | Multiplikation, Division, Ganzzahldivision, Modulus | Von links nach rechts |
| +, - | Addition, Subtraktion | Von links nach rechts |

**Anmerkung:** Vor einem Subtraktionsoperator muss ein Leerzeichen stehen, wenn er andernfalls als Teil eines vorherigen Tokens interpretiert werden könnte. Beispiel: `a-b` wird als Name interpretiert, während `a - b` und `a -b` als Rechenoperationen interpretiert werden.

Das Ergebnis eines arithmetischen Ausdrucks ist entweder ein numerischer Wert, eine leere Sequenz oder ein Fehler. Bei der Auswertung eines arithmetischen Ausdrucks wird jeder Operand in einen atomaren Wert konvertiert (atomisiert), und es werden die folgenden Regeln angewandt:

- Handelt es sich bei dem atomisierten Operanden um eine leere Sequenz, ist das Ergebnis des arithmetischen Ausdrucks eine leere Sequenz.
- Handelt es sich bei dem atomisierten Operanden um eine Sequenz mit mehr als einem Wert, wird ein Fehler zurückgegeben.
- Handelt es sich bei dem atomisierten Operanden um einen atomaren Wert ohne Datentyp (xdt:untypedAtomic), wird der Wert in 'xs:double' konvertiert. Schlägt die Konvertierung fehl, wird ein Fehler zurückgegeben.

Stellen die Typen der Operanden nach der Auswertung eine gültige Kombination für den arithmetischen Operator dar, wird der betreffende Operator auf den atomisierten Operanden angewandt, und das Ergebnis dieser Operation ist ein atomarer Wert oder ein Fehler (der beispielweise aus einer Division durch null resultiert). Handelt es sich bei den Typen der Operanden um keine gültige Kombination für den arithmetischen Operator, wird ein Fehler zurückgegeben.

Tabelle 23 zeigt gültige Kombinationen von Typen für arithmetische Operatoren. In dieser Tabelle stellt der Buchstabe A den ersten Operanden im Ausdruck dar, und der Buchstabe B steht für den zweiten Operanden. Der Begriff 'numerisch' bezeichnet die Typen 'xs:integer', 'xs:decimal', 'xs:float', 'xs:double' oder einen beliebigen Typ, der aus diesen Typen abgeleitet ist. Wird der Ergebnistyp eines Operators als 'numerisch' aufgeführt, entspricht der Ergebnistyp dem ersten Typ in der sortierten Liste (xs:integer, xs:decimal, xs:float, xs:double), in den alle Operanden durch Subtypsubstitution und Typumstufung konvertiert werden können.

*Tabelle 23. Gültige Typen für Operanden von arithmetischen Ausdrücken*

| Operator mit Operanden | Typ des Operanden A | Typ des Operanden B | Ergebnistyp |
|---|---|---|---|
| A + B | numerisch | numerisch | numerisch |
| | xs:date | xdt:yearMonthDuration | xs:date |
| | xdt:yearMonthDuration | xs:date | xs:date |
| | xs:date | xdt:dayTimeDuration | xs:date |
| | xdt:dayTimeDuration | xs:date | xs:date |
| | xs:time | xdt:dayTimeDuration | xs:time |
| | xdt:dayTimeDuration | xs:time | xs:time |
| | xs:dateTime | xdt:yearMonthDuration | xs:dateTime |
| | xdt:yearMonthDuration | xs:dateTime | xs:dateTime |
| | xs:dateTime | xdt:dayTimeDuration | xs:dateTime |
| | xdt:dayTimeDuration | xs:dateTime | xs:dateTime |
| | xdt:yearMonthDuration | xdt:yearMonthDuration | xdt:yearMonthDuration |
| | xdt:dayTimeDuration | xdt:dayTimeDuration | xdt:dayTimeDuration |

*Tabelle 23. Gültige Typen für Operanden von arithmetischen Ausdrücken (Forts.)*

| Operator mit Operanden | Typ des Operanden A | Typ des Operanden B | Ergebnistyp |
|---|---|---|---|
| A - B | numerisch | numerisch | numerisch |
| | xs:date | xs:date | xdt:dayTimeDuration |
| | xs:date | xdt:yearMonthDuration | xs:date |
| | xs:date | xdt:dayTimeDuration | xs:date |
| | xs:time | xs:time | xdt:dayTimeDuration |
| | xs:time | xdt:dayTimeDuration | xs:time |
| | xs:dateTime | xs:dateTime | xdt:dayTimeDuration |
| | xs:dateTime | xdt:yearMonthDuration | xs:dateTime |
| | xs:dateTime | xdt:dayTimeDuration | xs:dateTime |
| | xdt:yearMonthDuration | xdt:yearMonthDuration | xdt:yearMonthDuration |
| | xdt:dayTimeDuration | xdt:dayTimeDuration | xdt:dayTimeDuration |
| A * B | numerisch | numerisch | numerisch |
| | xdt:yearMonthDuration | numerisch | xdt:yearMonthDuration |
| | numerisch | xdt:yearMonthDuration | xdt:yearMonthDuration |
| | xdt:dayTimeDuration | numerisch | xdt:dayTimeDuration |
| | numerisch | xdt:dayTimeDuration | xdt:dayTimeDuration |
| A idiv B | numerisch | numerisch | xs:integer |
| A div B | numerisch | numerisch | numerisch, aber 'xs:decimal', wenn beide Operanden vom Typ 'xs:integer' sind |
| | xdt:yearMonthDuration | numerisch | xdt:yearMonthDuration |
| | xdt:dayTimeDuration | numerisch | xdt:dayTimeDuration |
| | xdt:yearMonthDuration | xdt:yearMonthDuration | xs:decimal |
| | xdt:dayTimeDuration | xdt:dayTimeDuration | xs:decimal |
| A mod B | numerisch | numerisch | numerisch |

## Beispiele

- Der erste nachstehende Ausdruck gibt den Wert -1,5 vom Typ 'xs:decimal' zurück, und der zweite nachstehende Ausdruck gibt den Wert -1 vom Typ 'xs:integer' zurück.

```
-3 div 2
-3 idiv 2
```

- Im folgenden Ausdruck ergibt die Subtraktion von zwei Datumswerten einen Wert vom Typ 'xdt:dayTimeDuration' :

```
$emp/hiredate - $emp/birthdate
```

- Das folgende Beispiel veranschaulicht den Unterschied zwischen einem Subtraktionsoperator und Bindestrichen, die in den Variablennamen `unit-price` und `unit-discount` verwendet werden:

```
$unit-price - $unit-discount
```

# Vergleichsausdrücke

Vergleichsausdrücke dienen dem Vergleich zweier Werte. XQuery stellt drei Typen von Vergleichsausdrücken bereit: Wertevergleiche, allgemeine Vergleiche und Knotenvergleiche.

## Wertevergleiche

Bei Wertevergleichen werden zwei atomare Werte miteinander verglichen. Zu den Operatoren von Wertevergleichen gehören **eq**, **ne**, **lt**, **le**, **gt** und **ge**.

Die folgende Tabelle enthält eine Beschreibung der einzelnen Operatoren.

*Tabelle 24. Vergleichsoperatoren für Werte in XQuery*

| Operator | Zweck |
|---|---|
| eq | Gibt 'true' (wahr) zurück, wenn der erste Wert mit dem zweiten Wert übereinstimmt. |
| ne | Gibt 'true' (wahr) zurück, wenn der erste Wert nicht mit dem zweiten Wert übereinstimmt. |
| lt | Gibt 'true' (wahr) zurück, wenn der erste Wert kleiner als der zweite Wert ist. |
| le | Gibt 'true' (wahr) zurück, wenn der erste Wert kleiner-gleich dem zweiten Wert ist. |
| gt | Gibt 'true' (wahr) zurück, wenn der erste Wert größer als der zweite Wert ist. |
| ge | Gibt 'true' (wahr) zurück, wenn der erste Wert größer-gleich dem zweiten Wert ist. |

Zwei Werte können miteinander verglichen werden, wenn sie denselben Typ haben oder wenn der Typ des einen Operanden ein Subtyp des Typs des anderen Operanden ist. Zwei Operanden eines numerischen Typs ('xs:float', 'xs:integer', 'xs:decimal', 'xs:double' und eines Typs, der von diesen Typen abgeleitet ist) können miteinander verglichen werden. Ebenso können Werte der Typen 'xs:string' und 'xs:anyURI' miteinander verglichen werden.

**Sonderwerte:** Bei Werten vom Typ 'xs:float' und 'xs:double' sind eine positive und eine negative Null äquivalent. INF und INF sind gleiche Werte, ebenso -INF und -INF. NaN ist ungleich sich selbst. Die positive Unendlichkeit ist größer als alle anderen Nicht-NaN-Werte. Die negative Unendlich ist kleiner als alle anderen Nicht-NaN-Werte. 'NaN **ne** NaN' ergibt 'true' (wahr), und alle anderen Vergleiche mit einem NaN-Wert ergeben 'false' (falsch). Zwei Werte vom Typ 'xs:QName' gelten als gleich, wenn ihre Namensbereichs-URIs gleich sind und ihre lokalen Namen gleich sind (Namensbereichspräfixe sind bedeutungslos).

Das Ergebnis eines Wertevergleichs kann ein Boolescher Wert, eine leere Sequenz oder ein Fehler sein. Bei der Auswertung eines Wertevergleichs wird jeder Operand in einen atomaren Wert konvertiert (atomisiert), und es werden die folgenden Regeln angewandt:

- Handelt es sich bei einem der atomisierten Operanden um eine leere Sequenz, ist das Ergebnis des Wertevergleichs eine leere Sequenz.
- Handelt es sich bei einem der atomisierten Operanden um eine Sequenz mit mehr als einem Wert, wird ein Fehler zurückgegeben.

- Handelt es sich bei einem der atomisierten Operanden um einen atomaren Wert ohne Datentyp (xdt:untypedAtomic), wird der Wert in 'xs:string' umgesetzt.

  Das Umsetzen von Werten des Typs 'xdt:untypedAtomic' in den Typ 'xs:string' ermöglicht transitive Wertevergleiche. Im Gegensatz dazu unterliegen allgemeine Vergleiche einer anderen Regel zum Umsetzen nicht typisierter Daten und sind daher nicht transitiv. Die Transitivität eines Wertevergleichs kann durch einen Verlust an Genauigkeit während der Typumsetzung beeinträchtigt werden. Zwei Werte vom Typ 'xs:integer' beispielsweise, die sich geringfügig unterscheiden, werden unter Umständen beide als mit demselben Wert vom Typ 'xs:float' übereinstimmend betrachtet, weil der Datentyp 'xs:float' eine geringere Genauigkeit aufweist als der Datentyp 'xs:integer'.
- Stellen die Typen der Operanden nach der Auswertung eine gültige Kombination für den Operator dar, wird der betreffende Operator auf die atomisierten Operanden angewandt, und das Ergebnis des Vergleichs ist entweder 'true' (wahr) oder 'false' (falsch). Handelt es sich bei den Typen der Operanden um keine gültige Kombination für den Vergleichsoperator, wird ein Fehler zurückgegeben.

Die nachstehenden Typen können mithilfe des Operators **eq** oder **ne** verglichen werden. Der Begriff 'Gregorianisch' bezieht sich auf die Typen 'xs:gYearMonth', 'xs:gYear', 'xs:gMonthDay', 'xs:gDay' und 'xs:gMonth'. Bei Binäroperatoren, die zwei Operanden mit einem Gregorianischen Typ akzeptieren, müssen beide Operanden denselben Typ aufweisen (wenn beispielsweise ein Operand den Typ 'xs:gDay' hat, muss der andere Operand ebenfalls den Typ 'xs:gDay' haben). Der Begriff 'numerisch' bezieht sich auf die Typen 'xs:integer', 'xs:decimal', 'xs:float', 'xs:double' und einen beliebigen Typ, der von einem dieser Typen abgeleitet ist. Bei Vergleichen mit numerischen Werten werden die Subtypsubstitution und die Umstufung numerischer Typen verwendet, um die Operanden in den ersten Typ in der sortierten Liste (xs:integer, xs:decimal, xs:float, xs:double) zu konvertieren, in den alle Operanden konvertiert werden können.

- Numerisch
- xs:boolean
- xs:string
- xs:date
- xs:time
- xs:dateTime
- xs:duration
- xdt:yearMonthDuration
- xdt:dayTimeDuration
- Gregorianisch
- xs:hexBinary
- xs:base64Binary
- xs:QName
- xs:NOTATION

Die nachstehenden Typen können mithilfe der Operatoren **gt**, **lt**, **ge** und **le** verglichen werden. Der Begriff 'numerisch' bezieht sich auf die Typen 'xs:integer', 'xs:decimal', 'xs:float', 'xs:double'. Bei Vergleichen mit numerischen Werten werden die Subtypsubstitution und die Umstufung numerischer Typen verwendet, um die Operanden in den ersten Typ in der sortierten Liste (xs:integer, xs:decimal, xs:float, xs:double) zu konvertieren, in den alle Operanden konvertiert werden können.

- Numerisch

- xs:boolean
- xs:string
- xs:date
- xs:time
- xs:dateTime
- xdt:yearMonthDuration
- xdt:dayTimeDuration

### Beispiele

- Bei folgendem Vergleich werden die vom Ausdruck '$book/author' zurückgegebenen Knoten atomisiert. Der Vergleich ergibt nur dann den Wert 'true' (wahr), wenn das Ergebnis der Atomisierung "Kennedy" lautet und eine Instanz des Datentyps 'xs:string' oder 'xdt:untypedAtomic' ist. Handelt es sich beim Ergebnis der Atomisierung um eine Sequenz mit mehr als einem Wert, wird ein Fehler zurückgegeben:

  ```
  $book1/author eq "Kennedy"
  ```

- Der nachstehende Pfadausdruck enthält ein Vergleichselement zur Auswahl von Produkten (product) mit einem Gewicht (weight) von mehr als 100. Bei allen Produkten, die kein Unterelement 'weight' aufweisen, ist der Wert des Vergleichselements eine leere Sequenz, und das Produkt wird nicht ausgewählt:

  ```
  //product[weight gt 100]
  ```

- Die folgenden Vergleiche ergeben 'true' (wahr), weil die beiden erstellten Knoten in allen Fällen nach der Atomisierung denselben Wert aufweisen, obwohl sie unterschiedliche Identitäten oder Namen haben:

  ```
  <a>5</a> eq <a>5</a>
  <a>5</a> eq <b>5</b>
  ```

## Allgemeine Vergleiche

Bei allgemeinen Vergleichen werden zwei Sequenzen einer beliebigen Länge miteinander verglichen, um zu ermitteln, ob mindestens ein Element in der ersten Sequenz und ein Element in der zweiten Sequenz die angegebene Vergleichsbedingung erfüllen. Die allgemeinen Vergleichsoperatoren umfassen die Operatoren '=', '!=', '<', '<=,>' und '>='.

Die folgende Tabelle enthält eine Beschreibung der einzelnen Operatoren.

*Tabelle 25. Vergleichsoperatoren für Werte in XQuery*

| Operator | Zweck |
|---|---|
| = | Gibt 'true' (wahr) zurück, wenn ein Wert in der ersten Sequenz mit einem Wert in der zweiten Sequenz übereinstimmt. |
| != | Gibt 'true' (wahr) zurück, wenn ein Wert in der ersten Sequenz nicht mit einem Wert in der zweiten Sequenz übereinstimmt. |
| < | Gibt 'true' (wahr) zurück, wenn ein Wert in der ersten Sequenz kleiner als ein Wert in der zweiten Sequenz ist. |
| <= | Gibt 'true' (wahr) zurück, wenn ein Wert in der ersten Sequenz kleinergleich einem Wert in der zweiten Sequenz ist. |
| > | Gibt 'true' (wahr) zurück, wenn ein Wert in der ersten Sequenz größer als ein Wert in der zweiten Sequenz ist. |
| >= | Gibt 'true' (wahr) zurück, wenn ein Wert in der ersten Sequenz größergleich einem Wert in der zweiten Sequenz ist. |

Das Ergebnis eines allgemeinen Vergleichs ist entweder ein Boolescher Wert oder ein Fehler. Bei der Auswertung eines allgemeinen Vergleichs wird jeder Operand atomisiert, d. h. in eine Sequenz aus atomaren Werten konvertiert. Die erste Sequenz wird mit der zweiten Sequenz verglichen, um zu ermitteln, ob mindestens ein Element in der ersten Sequenz und mindestens ein Element in der zweiten Sequenz die angegebene Vergleichsbedingung erfüllen. Beim Vergleich einzelner atomarer Werte werden die folgenden Regeln angewandt:

- Wenn einer der atomaren Werte eine Instanz vom Datentyp 'xdt:untypedAtomic' ist und der andere Wert eine Instanz eines numerischen Datentyps ist, dann wird der nicht typisierte Wert in den Datentyp 'xs:double' umgesetzt.
- Wenn einer der atomaren Werte ein Exemplar vom Datentyp 'xdt:untypedAtomic' ist und der andere Wert ein Exemplar vom Datentyp 'xdt:untypedAtomic' oder 'xs:string' ist, dann werden die Werte vom Typ 'xdt:untypedAtomic' in den Datentyp 'xs:string' umgesetzt.
- Wenn einer der atomaren Wert ein Exemplar vom Typ 'xdt:untypedAtomic' und der andere Wert keine Instanz vom Datentyp 'xs:string', 'xdt:untypedAtomic' oder einem numerischen Datentyp ist, wird der Wert vom Datentyp 'xdt:untypedAtomic' in den Typ des anderen Werts umgesetzt.

Nach dem Umsetzen der Typen wie oben beschrieben werden die atomaren Werte mithilfe eines der Vergleichsoperatoren für Werte (**eq**, **ne**, **lt**, **le**, **gt** oder **ge**) miteinander verglichen.

Das Ergebnis des Vergleichs ist wahr ('true'), wenn ein Paar atomarer Werte vorliegt (ein Wert in der ersten Operandensequenz und der andere in der zweiten Operandensequenz), auf das die Vergleichsbedingung als wahr zutrifft.

**Anmerkung:** Beim Auftreten von Fehlern kann das Ergebnis eines allgemeinen Vergleichs entweder ein Boolescher Wert oder ein Fehler sein. Der Vergleich `(1, 2) = (2, "Hello")` beispielsweise könnte den Wert 'true' (wahr) zurückgeben, weil `2 eq 2` wahr ist, oder aber auch einen Fehler, da der Wert 1 nicht mit dem Wert ″Hello″ vergleichbar ist.

**Tipp:** Um zwei Sequenzen Element für Element miteinander zu vergleichen, verwenden Sie die XQuery-Funktion 'fn:deep-equal'.

## Beispiele

- Der folgende Vergleich ergibt den Wert 'true' (wahr), wenn der typisierte Wert eines Unterelements 'author' von `$book1` ″Kennedy″ lautet und eine Instanz des Datentyps 'xs:string' oder 'xdt:untypedAtomic' ist:

  ```
  $book1/author = "Kennedy"
  ```

- Das nachstehende Beispiel enthält drei allgemeine Vergleiche. Der Wert der ersten beiden Vergleiche ist 'true' (wahr), und der Wert des dritten Vergleichs ist 'false' (falsch). Dieses Beispiel veranschaulicht die Tatsache, dass allgemeine Vergleiche nicht transitiv sind:

  ```
  (1, 2) = (2, 3)
  (2, 3) = (3, 4)
  (1, 2) = (3, 4)
  ```

- Das nachstehende Beispiel enthält zwei allgemeine Vergleiche, die beide den Wert 'true' (wahr) ergeben. Dieses Beispiel veranschaulicht die Tatsache, dass die Operatoren '=' und '!=' einander nicht entgegengesetzt sind:

  ```
  (1, 2) = (2, 3)
  (1, 2) != (2, 3)
  ```

- In dem nachstehenden Beispiel sind die Variablen '$a', '$b' und '$c' an Elementknoten mit der Typenannotation 'xdt:untypedAtomic' gebunden. Die Elementknoten enthalten die Zeichenfolgewerte ″1″, ″2″ bzw. ″2.0″. In diesem Beispiel gibt der folgende Ausdruck den Wert 'false' (falsch) zurück, weil die Werte, die an '$b' und '$c' gebunden sind (″2″ und ″2.0″), als Zeichenfolgen verglichen werden:

  ```
  ($a, $b) = ($c, 3.0)
  ```

  Der folgende Ausdruck hingegen gibt den Wert 'true' (wahr) zurück, weil der Wert, der an '$b' gebunden ist (″2″), und der Wert 2.0 als Zahlen verglichen werden:

  ```
  ($a, $b) = ($c, 2.0)
  ```

## Knotenvergleiche

Bei Knotenvergleichen werden jeweils zwei Knoten miteinander verglichen. Mithilfe eines Knotenvergleichs kann ermittelt werden, ob die betreffenden Knoten dieselbe ID gemeinsam nutzen oder ob einer der Knoten vor oder hinter dem anderen Knoten in der Dokumentreihenfolge steht.

Die folgende Tabelle enthält eine Beschreibung der Knotenvergleichsoperatoren, die in XQuery zur Verfügung stehen.

*Tabelle 26. Knotenvergleichsoperatoren in XQuery*

| Operator | Zweck |
|---|---|
| is | Gibt 'true' (wahr) zurück, wenn die beiden verglichenen Knoten dieselbe ID aufweisen. |
| << | Gibt 'true' (wahr) zurück, wenn der erste Operandenknoten in der Dokumentreihenfolge vor dem zweiten Operandenknoten steht. |
| >> | Gibt 'true' (wahr) zurück, wenn der erste Operandenknoten in der Dokumentreihenfolge nach dem zweiten Operandenknoten steht. |

Das Ergebnis eines Knotenvergleichs ist entweder ein Boolescher Wert, eine leere Sequenz oder ein Fehler. Das Ergebnis eines Knotenvergleichs unterliegt den folgenden Regeln:

- Jeder Operand muss entweder ein einzelner Knoten oder eine leere Sequenz sein. Andernfalls wird ein Fehler zurückgegeben.
- Wenn einer der Operanden eine leere Sequenz ist, wird als Ergebnis des Vergleichs eine leere Sequenz zurückgegeben.
- Ein Vergleich mithilfe des Operators **is** ergibt den Wert 'true' (wahr), wenn die beiden miteinander verglichenen Knoten dieselbe ID aufweisen. Andernfalls ergibt der Vergleich den Wert 'false' (falsch).
- Ein Vergleich mithilfe des Operators << ergibt den Wert 'true' (wahr), wenn der linke Operandenknoten in der Dokumentreihenfolge vor dem rechten Operandenknoten steht. Andernfalls ergibt der Vergleich den Wert 'false' (falsch).
- Ein Vergleich mithilfe des Operators >> ergibt den Wert 'true' (wahr), wenn der linke Operandenknoten in der Dokumentreihenfolge auf den rechten Operandenknoten folgt. Andernfalls ergibt der Vergleich den Wert 'false' (falsch).

### Beispiele

- Der folgende Vergleich ergibt nur dann den Wert 'true' (wahr), wenn sowohl der linke als auch der rechte Operand bei der Auswertung denselben Knoten aufweisen:

  ```
  /books/book[isbn="1558604820"] is /books/book[call="QA76.9 C3845"]
  ```

- Der folgende Vergleich ergibt den Wert 'false' (falsch), weil jeder erstellte Knoten eine eigene ID aufweist:

  ```
  <a>5</a> is <a>5</a>
  ```

- Der folgende Vergleich ergibt nur dann den Wert 'true' (wahr), wenn der durch den linken Operanden identifizierte Knoten in der Dokumentreihenfolge vor dem durch den rechten Operanden identifizierten Knoten steht:

  ```
  /transactions/purchase[parcel="28-451"] << /transactions/sale[parcel="33-870"]
  ```

## Logische Ausdrücke

Logische Ausdrücke verwenden die Operatoren **and** und **or** zur Berechnung eines Booleschen Werts ('true' für wahr oder 'false' für falsch).

In der folgenden Tabelle werden diese Operatoren beschrieben und in der Reihenfolge der Vorrangstellung für Operatoren von hoch nach niedrig aufgelistet.

*Tabelle 27. Operatoren logischer Ausdrücke in XQuery*

| Operator | Zweck |
|---|---|
| and | Gibt 'true' für wahr zurück, wenn beide Ausdrücke wahr sind. |
| or | Gibt 'true' für wahr zurück, wenn einer der Ausdrücke wahr ist oder beide Ausdrücke wahr sind. |

Das Ergebnis eines logischen Ausdrucks ist entweder ein Boolescher Wert ('true' oder 'false') oder ein Fehler. Bei der Auswertung eines logischen Ausdrucks wird der effektive Boolescher Wert (EBW) jedes Operanden ermittelt. Anschließend wird der Operator auf die effektiven Booleschen Werte des Operanden angewandt, und das Ergebnis ist entweder ein Boolescher Wert oder ein Fehler. Ist der EBW eines Operanden ein Fehler, führt der logische Ausdruck unter Umständen zu einem Fehler. Die folgende Tabelle zeigt die Ergebnisse, die von einem logischen Ausdruck auf der Grundlage der effektiven Booleschen Werte seiner Operanden zurückgegeben werden.

*Tabelle 28. Ergebnisse logischer Ausdrücke auf der Grundlage der effektiven Booleschen Werte von Operanden*

| EBW von Operand 1 | Operator | EBW von Operand 2 | Ergebnis |
|---|---|---|---|
| true | and | true | true |
| true | and | false | false |
| false | and | true | false |
| false | and | false | false |
| true | and | error (Fehler) | error (Fehler) |
| error (Fehler) | and | true | error (Fehler) |
| false | and | error (Fehler) | false oder error (Fehler) |
| error (Fehler) | and | false | false oder error (Fehler) |

*Tabelle 28. Ergebnisse logischer Ausdrücke auf der Grundlage der effektiven Booleschen Werte von Operanden (Forts.)*

| EBW von Operand 1 | Operator | EBW von Operand 2 | Ergebnis |
|---|---|---|---|
| error (Fehler) | and | error (Fehler) | error (Fehler) |
| true | or | true | true |
| false | or | false | false |
| true | or | false | true |
| false | or | true | true |
| true | or | error (Fehler) | true oder error (Fehler) |
| error (Fehler) | or | true | true oder error (Fehler) |
| false | or | error (Fehler) | error (Fehler) |
| error (Fehler) | or | false | error (Fehler) |
| error (Fehler) | or | error (Fehler) | error (Fehler) |

**Tipp:** Zusätzlich zu logischen Ausdrücken stellt XQuery eine Funktion namens 'fn:not' bereit, die eine allgemeine Sequenz als Parameter angibt und einen Booleschen Wert zurückgibt.

### Beispiele

- Die folgenden Ausdrücke geben den Wert 'true' für wahr zurück:

```
1 eq 1 and 2 eq 2
1 eq 1 or 2 eq 3
```

- Der folgende Ausdruck kann entweder den Wert 'false' für falsch oder einen Fehler (error) zurückgeben:

```
1 eq 2 and 3 idiv 0 = 1
```

- Der folgende Ausdruck kann entweder den Wert 'true' für wahr oder einen Fehler (error) zurückgeben:

```
1 eq 1 or 3 idiv 0 = 1
```

- Der folgende Ausdruck gibt einen Fehler (error) zurück:

```
1 eq 1 and 3 idiv 0 = 1
```

## Konstruktoren

Konstruktoren erstellen XML-Strukturen innerhalb einer Abfrage. XQuery bietet Konstruktoren zum Erstellen von Elementknoten, Attributknoten, Dokumentknoten, Textknoten, Verarbeitungsanweisungsknoten und Kommentarknoten. XQuery stellt zwei Typen von Konstruktoren bereit: direkte Konstruktoren und berechnete Konstruktoren.

*Direkte Konstruktoren* verwenden eine XML-artige Notation zum Erstellen von XML-Strukturen innerhalb einer Abfrage. XQuery bietet direkte Konstruktoren zum Erstellen von Elementknoten (die Attributknoten, Textknoten und verschachtelte Elementknoten umfassen können), Verarbeitungsanweisungsknoten und Kommentarknoten. Der folgende Konstruktor beispielsweise erstellt ein Element vom Typ `book` (Buch), das ein Attribut und einige verschachtelte Elemente enthält:

```
<book isbn="isbn-0060229357">
  <title>Harold and the Purple Crayon</title>
  <author>
```

```
    <first>Crockett</first>
    <last>Johnson</last>
  </author>
</book>
```

*Berechnete Konstruktoren* verwenden eine Notation auf der Grundlage von eingeschlossenen Ausdrücken, um XML-Strukturen innerhalb einer Abfrage zu erstellen. Ein berechneter Konstruktor beginnt mit einem Schlüsselwort, das den Typ des zu erstellenden Knotens identifiziert, gefolgt vom Namen des Knotens (sofern zutreffend) und einem eingeschlossenen Ausdruck, der den Inhalt des Knotens berechnet. XQuery bietet berechnete Konstruktoren zum Erstellen von Elementknoten, Attributknoten, Dokumentknoten, Textknoten, Verarbeitungsanweisungsknoten und Kommentarknoten. Die folgende Abfrage beispielsweise enthält berechnete Konstruktoren, die dasselbe Ergebnis generieren wie der vorstehend beschriebene direkte Konstruktor:

```
element book {
  attribute isbn {"isbn-0060229357" },
  element title { "Harold and the Purple Crayon"},
  element author {
    element first { "Crockett" },
    element last {"Johnson" }
  }
}
```

## Eingeschlossene Ausdrücke in Konstruktoren

Eingeschlossene Ausdrücke werden in Konstruktoren verwendet, um berechnete Werte für Element- und Attributinhalte bereitzustellen. Diese Ausdrücke werden bei der Verarbeitung des Konstruktors ausgewertet und durch den ihnen zugeordneten Wert ersetzt. Eingeschlossene Ausdrücke werden in geschweifte Klammern ({}) gesetzt, um sie von Literaltext unterscheiden zu können.

Eingeschlossene Ausdrücke können in den folgenden Konstruktoren verwendet werden, um berechnete Werte bereitzustellen:

- In direkten Elementkonstruktoren:
  - Ein Attributwert im Startbefehl eines direkten Elementkonstruktors kann einen eingeschlossenen Ausdruck umfassen.
  - Der Inhalt eines direkten Elementkonstruktors kann einen eingeschlossenen Ausdruck umfassen, der sowohl den Inhalt als auch die Attribute des erstellten Knotens berechnet.
- In berechneten Konstruktoren:
  - Ein eingeschlossener Ausdruck kann verwendet werden, um den Inhalt des Knotens zu generieren.

Der folgende direkte Elementkonstruktor beispielsweise umfasst einen eingeschlossenen Ausdruck:

```
<example>
  <p> Here is a query. </p>
  <eg> $b/title </eg>
  <p> Here is the result of the query. </p>
  <eg>{ $b/title }</eg>
</example>
```

Bei der Auswertung dieses Konstruktors könnte das folgende Ergebnis generiert werden (zur besseren Lesbarkeit werden diesem Beispiel Leerzeichen hinzugefügt):

```
<example>
  <p> Here is a query. </p>
  <eg> $b/title </eg>
  <p> Here is the result of the query. </p>
  <eg><title>Harold and the Purple Crayon</title></eg>
</example>
```

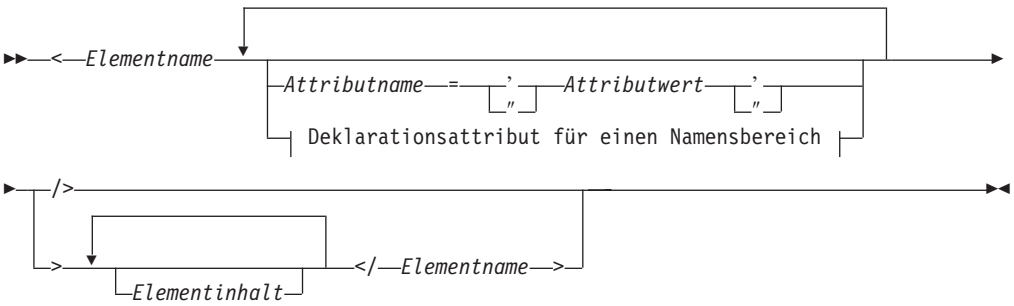
**Tipp:** Um eine geschweifte Klammer als ordentliches Zeichen innerhalb des Inhalts eines Elements oder Attributs verwenden zu können, haben Sie die Möglichkeit, entweder ein Paar identischer geschweifter Klammern einzuschließen oder Zeichenverweise zu verwenden. So können Sie beispielsweise das Paar '{{' zur Darstellung des Zeichens '{' verwenden. Ebenso können Sie das Paar '}}' zur Darstellung des Zeichens '}' verwenden. Alternativ können Sie für die Angabe von geschweiften Klammern auch die Zeichenverweise '{' und '}' verwenden. Eine einzelne linke geschweifte Klammer ({) wird als Anfangsbegrenzer für einen eingeschlossenen Ausdruck interpretiert. Eine einzelne rechte geschweifte Klammer (}) ohne eine entsprechende linke geschweifte Klammer führt zu einem Fehler.

## Direkte Elementkonstruktoren

Direkte Elemementkonstruktoren verwenden eine XML-artige Notation zum Erstellen von Elementknoten. Der erstellte Knoten kann sowohl ein einfaches Element sein oder ein komplexes Element, das Attribute, Textinhalt und verschachtelte Elemente enthält.

Das Ergebnis eines direkten Elementkonstruktors ist ein neuer Elementknoten, der über eine eigene Knotenidentität verfügt. Sämtliche Attributknoten und untergeordnete Knoten des neuen Elementknotens sind ebenfalls neue Knoten mit einer eigenen Knotenidentität.

### Syntax




#### Deklarationsattribut für einen Namensbereich:

*Elementname*
Ein qualifizierter Name (QName), der den Namen des zu erstellenden Elements angibt. Der für *Elementname* verwendete Name im Endbefehl muss mit dem im entsprechenden Startbefehl verwendeten Namen identisch sein (einschließlich Präfix, sofern vorhanden). Enthält der Wert für *Elementname* ein Namensbereichspräfix, wird das Präfix unter Verwendung der statisch bekann-

ten Namensbereiche in eine Namensbereichs-URI aufgelöst. Hat *Elementname* kein Namensbereichspräfix, wird der Name implizit durch den Standardnamensbereich für Elemente/Typen qualifiziert. Der aus der Auswertung von *Elementname* resultierende erweiterte QName wird als Name des erstellten Elementknotens verwendet.

*Attributname*
Ein QName, der den Namen des zu erstellenden Attributs angibt. Enthält der Wert für *Attributname* ein Namensbereichspräfix, wird das Präfix unter Verwendung der statisch bekannten Namensbereiche in eine Namensbereichs-URI aufgelöst. Enthält *Attributname* kein Namensbereichspräfix, befindet sich das Attribut in keinem Namensbereich. Der aus der Auswertung von *Attributname* resultierende erweiterte QName wird als Name des erstellten Attributknotens verwendet. Der erweiterte QName jedes Attributs muss eindeutig sein, da der Ausdruck ansonsten einen Fehler ergibt.

Jedes Attribut in einem direkten Elementkonstruktor erstellt einen neuen Attributknoten mit eigener Knotenidentität. Das übergeordnete Element (Elter) des neuen Attributknotens ist der erstellte Elementknoten. Dem neuen Attributknoten wird die Typenannotation 'xdt:untypedAtomic' zugeordnet.

*Attributwert*
Eine Zeichenfolge, die einen Wert für das Attribut angibt. Der Attributwert kann eingeschlossene Ausdrücke (Ausdrücke in geschweiften Klammern) enthalten, die bei der Verarbeitung des Elementkonstruktors ausgewertet und durch die ihnen zugeordneten Werte ersetzt werden. Vordefinierte Entitäts- und Zeichenverweise sind ebenfalls gültig und werden durch die von ihnen dargestellten Zeichen ersetzt. Die folgende Tabelle enthält eine Liste der innerhalb von *Attributwert* gültigen Sonderzeichen. Diese Sonderzeichen müssen jedoch durch Doppelzeichen oder einen Entitätsverweis dargestellt werden.

*Tabelle 29. Darstellung von Sonderzeichen in Attributwerten*

| Zeichen | Erforderliche Darstellung in Attributwerten |
|---|---|
| { | Zwei offene geschweifte Klammern ({{) |
| } | Zwei geschlossene geschweifte Klammern (}}) |
| < | < |
| & | & |
| " | " oder zwei doppelte Anführungszeichen ("") |
| ' | ' oder zwei einfache Anführungszeichen ('') |

**xmlns**
Das Wort, mit dem ein Deklarationsattribut für einen Namensbereich beginnt. Bei Angabe als Präfix in einem qualifizierten Namen gibt **xmlns** an, dass der Wert von *zu_bindendes_Präfix* an die von *URI-Literal* angegebene URI gebunden wird. Diese Namensbereichsbindung wird den statisch bekannten Namensbereichen für diesen Konstruktorausdruck und für alle innerhalb des Ausdrucks verschachtelten Ausdrücke hinzugefügt (sofern die Bindung nicht durch ein verschachteltes Deklarationsattribut für einen Namensbereich überschrieben wird). In folgendem Beispiel bindet das Deklarationsattribut für einen Namensbereich `xmlns:metric = "http://example.org/metric/units"` das Präfix `metric` an den Namensbereich `http://example.org/metric/units`.

Bei Angabe als vollständiger QName ohne Präfix gibt **xmlns** an, dass der Standardnamensbereich für Elemente/Typen auf den Wert von *URI-Literal* gesetzt wird. Dieser Standardnamensbereich für Elemente/Typen gilt für diesen Konstruktorausdruck und für alle innerhalb des Konstruktorausdrucks ver-

schachtelten Ausdrücke (sofern die Deklaration nicht durch ein verschachteltes Deklarationsattribut für einen Namensbereich überschrieben wird). In folgendem Beispiel setzt das Deklarationsattribut für einen Namensbereich `xmlns = "http://example.org/animals"` den Standardnamensbereich für Elemente/Typen auf `http://example.org/animals`.

*zu_bindendes_Präfix*
Das Präfix, das an die für *URI-Literal* angegebene URI gebunden werden soll. Der Wert von *zu_bindendes_Präfix* darf nicht `xml` oder `xmlns` sein. Die Angabe eines dieser Werte verursacht einen Fehler.

*URI-Literal*
Ein Zeichenfolgeliteral (eine Sequenz von null oder mehr Zeichen in einfachen oder doppelten Anführungszeichen) zur Darstellung einer URI. Der Wert des Zeichenfolgeliterals muss eine gültige URI sein. Der Wert von *URI-Literal* kann nur dann eine Zeichenfolge mit Nulllänge sein, wenn das Deklarationsattribut für einen Namensbereich verwendet wird, um den Standardnamensbereich für Elemente/Typen zu setzen. Andernfalls verursacht die Angabe einer Zeichenfolge mit Nulllänge für *URI-Literal* einen Fehler.

*Elementinhalt*
Der Inhalt des direkten Elementkonstruktors. Der Inhalt besteht aus allem, was zwischen dem Start- und Endbefehl des Konstruktors angegeben ist. Die Art der Verarbeitung von Begrenzungsbereichen innerhalb von Elementkonstruktoren wird von der Deklaration 'boundary-space' im Prolog gesteuert. Die resultierende Inhaltssequenz ist eine Verknüpfung der Inhaltselemente. Alle resultierenden benachbarten Textzeichen (einschließlich Text aus eingeschlossenen Ausdrücken) werden zu einem einzigen Textknoten zusammengeführt. Alle resultierenden Attributknoten müssen in der resultierenden Inhaltssequenz vor allen anderen Inhaltselementen stehen.

*Elementinhalt* kann aus folgenden Inhaltselementen bestehen:

- **Textzeichen.** Aus Textzeichen werden Textknoten erstellt, und benachbarte Textknoten werden zu einem einzigen Textknoten zusammengeführt. Zeilenenden innerhalb von Zeichenfolgen werden gemäß den für XML 1.0 angegebenen Regeln für die Verarbeitung von Zeilenenden normalisiert. Die folgende Tabelle enthält eine Liste der innerhalb von *Elementinhalt* gültigen Sonderzeichen. Diese Sonderzeichen müssen jedoch durch Doppelzeichen oder einen Entitätsverweis dargestellt werden.

*Tabelle 30. Darstellung von Sonderzeichen im Elementinhalt*

| Zeichen | Erforderliche Darstellung im Elementinhalt |
|---|---|
| { | Zwei offene geschweifte Klammern ({{) |
| } | Zwei geschlossene geschweifte Klammern (}}) |
| < | < |
| & | & |

- **Verschachtelte direkte Konstruktoren.**
- **CDATA-Abschnitte.** CDATA-Abschnitte (CDataSections) werden mithilfe der folgenden Syntax angegeben: `<![CDATA[Inhalt]]>`. Hierbei besteht *Inhalt* aus einer Reihe von Zeichen. Die für *Inhalt* angegebenen Zeichen (einschließlich Sonderzeichen wie beispielsweise < und &) werden als Literalzeichen und nicht als Begrenzer behandelt. Die Sequenz ]]> beendet den CDATA-Abschnitt und ist daher innerhalb von *Inhalt* unzulässig.

- **Zeichenverweise und vordefinierte Entitätsverweise.** Während der Verarbeitung werden vordefinierte Entitätsverweise und Zeichenverweise in die jeweils zugeordneten Referenzzeichenfolgen aufgelöst.
- **Eingeschlossene Ausdrücke.** Bei einem eingeschlossenen Ausdruck handelt es sich um einen XQuery-Ausdruck in geschweiften Klammern. Der Ausdruck {5 + 7} beispielsweise ist ein eingeschlossener Ausdruck. Der Wert eines eingeschlossenen Ausdrucks kann eine beliebige Sequenz aus Knoten und atomaren Werten sein. Eingeschlossene Ausdrücke können innerhalb des Inhalts eines direkten Elementkonstruktors verwendet werden, um sowohl den Inhalt als auch die Attribute des erstellten Knotens zu berechnen. Für jeden von einem eingeschlossenen Ausdruck zurückgegebenen Knoten wird eine neue Kopie des Knotens und sämtlicher seiner untergeordneten Elemente erstellt, die ihre ursprünglichen Typenannotationen beibehalten. Sämtliche von *Elementinhalt* zurückgegebenen Attributknoten müssen am Anfang der resultierenden Inhaltssequenz stehen. Diese Attributknoten werden als Attribute des erstellten Elements verwendet. Alle von *Elementinhalt* zurückgegebenen Element-, Inhalts- und Verarbeitungsanweisungsknoten werden als untergeordnete Elemente (Kinder) des neu erstellten Knotens verwendet. Alle von *Elementinhalt* zurückgegebenen atomaren Werte werden in Zeichenfolgen umgesetzt und in Textknoten gespeichert, die als Kinder des erstellten Knotens verwendet werden. Benachbarte Textknoten werden zu einem einzigen Textknoten zusammengeführt.

## Beispiele

- Der folgende direkte Elementkonstruktor erstellt ein Element vom Typ 'book' (Buch). Das Element 'book' weist einen komplexen Inhalt auf, der einen Attributknoten, einige verschachtelte Elementknoten und einige Textknoten umfasst:

```
<book isbn="isbn-0060229357">
    <title>Harold and the Purple Crayon</title>
    <author>
        <first>Crockett</first>
        <last>Johnson</last>
    </author>
</book>
```

- Die nachstehenden Beispiele veranschaulichen, wie Elementinhalt in direkten Elementkonstruktoren verarbeitet wird:
  - Der folgende Ausdruck erstellt einen Elementknoten mit einem Kind, d. h. einem Textknoten mit dem Wert ″1″:

    ```
    <a>{1}</a>
    ```

  - Der folgende Ausdruck erstellt einen Elementknoten mit einem Kind, d. h. einem Textknoten mit dem Wert ″1 2 3″:

    ```
    <a>{1, 2, 3}</a>
    ```

  - Der folgende Ausdruck erstellt einen Elementknoten mit einem Kind, d. h. einem Textknoten mit dem Wert ″123″:

    ```
    <c>{1}{2}{3}</c>
    ```

  - Der folgende Ausdruck erstellt einen Elementknoten mit einem Kind, d. h. einem Textknoten mit dem Wert ″1 2 3″:

    ```
    <b>{1, "2", "3"}</b>
    ```

  - Der folgende Ausdruck erstellt einen Elementknoten mit einem Kind, d. h. einem Textknoten mit dem Wert ″I saw 8 cats″:

    ```
    <fact>I saw 8 cats.</fact>
    ```

  - Der folgende Ausdruck erstellt einen Elementknoten mit einem Kind, d. h. einem Textknoten mit dem Wert ″I saw 8 cats″:

    ```
    <fact>I saw {5 + 3} cats.</fact>
    ```

– Der folgende Ausdruck erstellt einen Elementknoten mit drei Kindern: einem Textknoten mit dem Inhalt ″I saw ″, einem Kindelementknoten namens `howmany` und einem Textknoten mit dem Inhalt `"cats"`. Der Kindelementknoten hat ein einziges Kind, d. h. einen Textknoten mit dem Wert ″8″.

```
<fact>I saw <howmany>{5 + 3}</howmany> cats.</fact>
```

## Deklarationsattribute für Namensbereiche

Deklarationsattribute für Namensbereiche werden im Startbefehl eines direkten Elementkonstruktors angegeben. Deklarationsattribute für Namensbereiche werden für folgende beiden Zwecke verwendet: zur Bindnung eines Namensbereichspräfixes an eine URI und zum Festlegen des Standardnamensbereichs für Elemente/Typen für den erstellten Elementknoten und dessen Attribute und untergeordneten Elemente (Nachkommen).

Die Syntax eines Deklarationsattributs für einen Namensbereich entspricht dem Format eines Attributs in einem direkten Elementkonstruktor: Das Attribut wird durch einen Namen und einen Wert angegeben. Der Attributname ist ein konstanter qualifizierter Name (QName). Der Attributwert ist ein Zeichenfolgeliteral, das eine gültie URI darstellt.

Ein Deklarationsattribut für einen Namensbereich führt nicht zur Erstellung eines Attributknotens.

**Wichtig:** Der Name jedes einzelnen Deklarationsattributs für einen Namensbereich in einem direkten Elementkonstruktor muss eindeutig sein. Andernfalls führt der Ausdruck zu einem Fehler.

### Binden eines Namensbereichspräfixes an eine URI

Wenn der QName mit dem Präfix `xmlns` beginnt, auf das ein lokaler Name folgt, wird die Deklaration verwendet, um das Namensbereichspräfix (das als lokaler Name angegeben ist) an eine URI (die als Attributwert angegeben ist) zu binden. Das Deklarationsattribut `xmlns:metric = "http://example.org/metric/units"` beispielsweise bindet das Präfix `metric` an den Namensbereich `http://example.org/metric/units`.

Bei der Verarbeitung des Deklarationsattributs für den Namensbereich werden das Präfix und die URI den statisch bekannten Namensbereichen des Konstruktorausdrucks hinzugefügt, und die neue Bindung überschreibt jede vorhandene Bindung für das betreffende Präfix. Das Präfix und die URI werden darüber hinaus auch als Namensbereichsbindung den gültigen Namensbereichen des erstellten Elements hinzugefügt.

Im folgenden Elementkonstruktor beispielsweise werden Deklarationsattribute für Namensbereiche verwendet, um die Namensbereichspräfixe `metric` und `english` zu binden:

```
<box xmlns:metric = "http://example.org/metric/units"
xmlns:english = "http://example.org/english/units">
  <height> <metric:meters>3</metric:meters> </height>
  <width> <english:feet>6</english:feet> </width>
  <depth> <english:inches>18</english:inches> </depth>
</box>
```

### Festlegen des Standardnamensbereichs für Elemente/Typen

Wenn der QName den Wert `xmlns` ohne Präfix hat, wird die Deklaration verwendet, um den Standardnamensbereich für Elemente/Typen festzulegen. Das

Deklarationsattribut `xmlns = "http://example.org/animals"` beispielsweise legt den Standardnamensbereich für Elemente/Typen auf `http://example.org/animals` fest.

Bei der Verarbeitung des Deklarationsattributs wird der Wert als eine Namensbereichs-URI interpretiert. Diese URI gibt den Standardnamensbereich für Elemente/Typen des Konstruktorausdrucks an, und die neue Spezifikation überschreibt alle bisher vorhandenen Standardwerte. Darüber hinaus wird die URI (ohne Präfix) den gültigen Namensbereichen des erstellten Elements hinzugefügt, und die neue Spezifikation überschreibt alle bisher vorhandenen Namensbereichsbindungen, die kein Präfix aufweisen. Ist die Namensbereichs-URI eine Zeichenfolge mit Nulllänge, wird der Standardnamensbereich für Elemente/Typen des Konstruktorausdrucks auf ″none″ (nicht vorhanden) gesetzt.

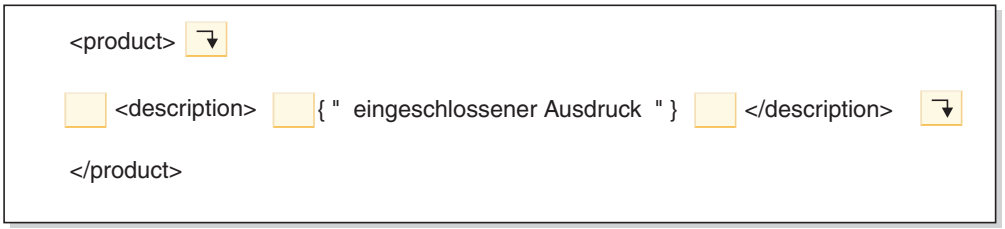
Im folgenden direkten Elementkonstruktor beispielsweise legt ein Deklarationsattribut für Namensbereiche den Standardnamensberich für Elemente/Typen auf `http://example.org/animals` fest:

```
<cat xmlns = "http://example.org/animals">
  <breed>Persian</breed>
</cat>
```

## Begrenzungsleerzeichen in direkten Elementkonstruktoren

In einem direkten Elementkonstruktor sind *Begrenzungsleerzeichen* eine Sequenz aus aufeinanderfolgenden einzelnen Leerzeichen, die rechts und links entweder vom Beginn bzw. Ende des Inhalts, von einem direkten Konstruktor oder von einem eingeschlossenen Ausdruck begrenzt ist. So können Begrenzungsleerzeichen beispielsweise im Inhalt des Konstruktors verwendet werden, um den Endbefehl eines direkten Konstruktors vom Startbefehl eines verschachtelten Elements zu trennen.

Das folgende Diagramm zeigt ein Beispiel eines direkten Elementkonstruktors mit hervorgehobenen Begrenzungsleerzeichen:




Die Begrenzungsleerzeichen in diesem Beispiel umfassen die folgenden Zeichen: ein Zeilenvorschubzeichen und vier einzelne Leerzeichen zwischen den Startbefehlen der Elemente `product` (Produkt) und `description` (Beschreibung), vier einzelne Leerzeichen zwischen dem Startbefehl des Elements `description` und dem eingeschlossenen Ausdruck, vier einzelne Leerzeichen zwischen dem eingeschlossenen Ausdruck und dem Endbefehl des Elements `description`, sowie ein Zeilenvorschubzeichen nach dem Endbefehl des Elements `description`.

Die folgenden Leerzeichentypen gehören nicht zu den Begrenzungsleerzeichen:

- Leerzeichen, die von einem eingeschlossenen Ausdruck generiert werden.
- Zeichen, die von Zeichenverweisen (beispielsweise ` `) oder einem CDATA-Abschnitt generiert werden.
- Leerzeichen, die neben einem Zeichenverweis oder einem CDATA-Abschnitt stehen.

Mithilfe der Richtlinie für Begrenzungsleerzeichen (boundary-space) wird gesteuert, ob Begrenzungsleerzeichen von Elementkonstruktoren beibehalten werden oder nicht. Diese Richtlinie wird von einer Deklaration vom Typ 'boundary-space' im Abfrageprolog angegeben. Wenn in der Deklaration 'boundary-space' der Wert **strip** angegeben wird, werden Begrenzungsleerzeichen verworfen. Wenn in der Deklaration 'boundary-space' der Wert **preserve** angegeben wird, werden Begrenzungsleerzeichen beibehalten. Wird keine Deklaration 'boundary-space' angegeben, werden Begrenzungsleerzeichen während der Elementerstellung standardmäßig verworfen.

### Beispiele

- Im folgenden Beispiel hat der erstellte Elementknoten `cat` (Katze) zwei Kindelementknoten namens `breed` (Rasse) und `color` (Farbe):

  ```
  <cat>
    <breed>{$b}</breed>
    <color>{$c}</color>
  </cat>
  ```

  Da die Richtlinie für Begrenzungsleerzeichen standardmäßig **strip** lautet, werden die Leerzeichen um die Kindelemente herum vom Elementkonstruktor verworfen.
- Im nachstehenden Beispiel lautet die Richtlinie für Begrenzungsleerzeichen **strip**. Dieses Beispiel ist äquivalent zu `<a>abc</a>`:

  ```
  declare boundary-space strip;
  <a> {"abc"} </a>
  ```
- Im nachstehenden Beispiel lautet die Richtlinie für Begrenzungsleerzeichen jedoch **preserve**. Dieses Beispiel ist äquivalent zu `<a>abc</a>`:

  ```
  declare boundary-space preserve;
  <a> {"abc"} </a>
  ```

  Da die Richtlinie für Begrenzungsleerzeichen **preserve** lautet, werden die Leerzeichen vor und nach dem eingeschlossenen Ausdruck vom Elementkonstruktor beibehalten.
- Im nachstehenden Beispiel sind die Leerzeichen um den Buchstaben `z` herum keine Begrenzungsleerzeichen. Die Leerzeichen werden stets beibehalten, und dieses Beispiel ist äquivalent zu `<a> z abc</a>`:

  ```
  <a> z {"abc"}</a>
  ```
- Im nachstehenden Beispiel werden die vom Zeichenverweis generierten Leerzeichen und die benachbarten Leerzeichen beibehalten, unabhängig von der Richtlinie für Begrenzungsleerzeichen. Dieses Beispiel ist äquivalent zu `<a>  abc</a>`:

  ```
  <a>  &#x20;{"abc"}</a>
  ```
- Im nachstehenden Beispiel werden die Leerzeichen im eingeschlossenen Ausdruck unabhängig von der Richtlinie für Begrenzungsleerzeichen beibehalten, da die von einem eingeschlossenen Ausdruck generierten Leerzeichen nie als Begrenzungsleerzeichen gelten. Dieses Beispiel ist äquivalent zu `<a> </a>`:

  ```
  <a>{" "}</a>
  ```

  Die beiden Leerzeichen im eingeschlossenen Ausdruck werden vom Elementkonstruktor beibehalten und zwischen dem Startbefehl und dem Endbefehl im Ergebnis angezeigt.

## Gültige Namensbereiche eines erstellten Elements

Ein erstellter Elementknoten hat ein Merkmal für gültige Namensbereiche, das aus einer Gruppe von Namensbereichsbindungen besteht. Jede Namensbereichsbindung ordnet ein Namensbereichspräfix einer URI zu. Die Namensbereichsbindungen definieren die Gruppe der Namensbereichspräfixe, die für das Interpretieren qualifizierter Namen (QNames) innerhalb des Geltungsbereichs eines Elements verfügbar sind.

**Wichtig:** Zum Verständnis dieses Abschnitts muss der Unterschied zwischen den folgenden Konzepten bekannt sein:

**Statisch bekannte Namensbereiche**
: *Statisch bekannte Namensbereiche* sind ein Merkmal eines Ausdrucks. Dieses Merkmal gibt die Gruppe der Namensbereichsbindungen an, die von XQuery verwendet werden, um Namensbereichspräfixe während der Verarbeitung des Ausdrucks aufzulösen. Diese Bindungen gehören nicht zum Abfrageergebnis.

**Gültige Namensbereiche**
: *Gültige Namensbereiche* sind ein Merkmal eines Elementknotens. Dieses Merkmal gibt die Gruppe der Namensbereichsbindungen an, die Anwendungen außerhalb von XQuery zur Verfügung stehen, wenn das Element samt Inhalt verarbeitet wird. Diese Bindungen werden als Teil des Abfrageergebnisses serialisiert, sodass sie auch externen Anwendungen zur Verfügung stehen.

Die gültigen Namensbereiche eines erstellten Elements umfassen alle Namensbereichsbindungen, die wie folgt erstellt werden:

- **Explizit mithilfe von Deklarationsattributen für Namensbereiche.** Eine Namensbereichsbindung wird für alle diese Deklarationsattribute erstellt, die in den folgenden Konstruktoren deklariert werden:
  - Im aktuellen Elementkonstruktor.
  - In einem übergeordneten direkten Elementkonstruktor (sofern das Deklarationsattribut für einen Namensbereich nicht vom aktuellen Elementkonstruktor oder einem Zwischenkonstruktor überschrieben wird).
- **Automatisch durch das System.** Eine Namensbereichsbindung wird in folgenden Situationen erstellt:
  - Um das Präfix `xml` an die Namensbereichs-URI `http://www.w3.org/XML/1998/namespace` zu binden. Diese Bindung wird für jedes erstellte Element generiert.
  - Für jeden Namensbereich, der im Namen eines erstellten Elements oder in den Namen von dessen Attributen verwendet wird (sofern die Namensbereichsbindung nicht bereits in den gültigen Namensbereichen des Elements vorhanden ist). Enthält der Name des Knotens ein Präfix, wird dieses Präfix beim Binden des Namensbereichs verwendet. Hat der Name kein Präfix, wird eine Bindung für das leere Präfix erstellt. Tritt ein Konflikt auf, der zwei unterschiedliche Bindungen desselben Präfix erforderlich machen würde, wird das Präfix, das im Knotennamen verwendet wird, in ein beliebiges Präfix geändert, und die Namensbereichsbindung wird für das beliebige Präfix erstellt.

    **Hinweis:** Ein Präfix, das in einem qualifizierten Namen (QName) verwendet wird, muss sich in eine gültige URI auflösen lassen. Andernfalls kann den

gültigen Namensbereichen des Elements keine Bindung für das betreffende Präfix hinzugefügt werden. Kann der QName nicht aufgelöst werden, führt der Ausdruck zu einem Fehler.

### Beispiel

Die folgende Abfrage umfasst einen Prolog mit Namensbereichsdeklarationen und einen Hauptteil mit einem direkten Elementkonstruktor:

```
declare namespace p="http://example.com/ns/p";
declare namespace q="http://example.com/ns/q";
declare namespace f="http://example.com/ns/f";
<p:newElement q:b="{f:func(2)}" xmlns:r="http://example.com/ns/r"/>
```

Die Namensbereichsdeklarationen im Prolog fügen die Namensbereichsbindungen den statisch bekannten Namensbereichen des Ausdrucks hinzu. Die Namensbereichsbindungen werden den gültigen Namensbereichen des erstellten Elements jedoch nur dann hinzugefügt, wenn die qualifizierten Namen (QNames) im Konstruktor diese Namensbereiche verwenden. Daher bestehen die gültigen Namensbereiche von `p:newElement` aus den folgenden Namensbereichsbindungen:

- `p = "http://example.com/ns/p"` - Diese Namensbereichsbindung wird den gültigen Namensbereichen hinzugefügt, weil das Präfix `p` im qualifizierten Namen `p:newElement` vorkommt.
- `q = "http://example.com/ns/q"` - Diese Namensbereichsbindung wird den gültigen Namensbereichen hinzugefügt, weil das Präfix `q` im qualifizierten Attributnamen `q:b` vorkommt.
- `r = "http://example.com/ns/r"` - Diese Namensbereichsbindung wird den gültigen Namensbereichen hinzugefügt, weil sie von einem Deklarationsattribut für einen Namensbereich definiert ist.
- `xml = "http://www.w3.org/XML/1998/namespace"` - Diese Namensbereichsbindung wird den gültigen Namensbereiche hinzugefügt, weil sie für jeden erstellten Elementknoten definiert ist.

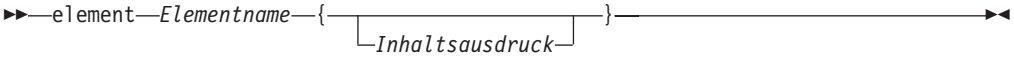
Hierbei ist zu beachten, dass den gültigen Namensbereichen keine Bindung für den Namensbereich `f="http://example.com/ns/f"` hinzugefügt wird. Dies liegt daran, dass der Elementkonstruktor keine Element- und Attributnamen umfasst, die das Präfix `f` verwenden (obwohl `f:func(2)` im Inhalt des Attributs `q:b` vorkommt). Daher kommt diese Namensbereichsbindung im Abfragebereich nicht vor, obwohl sie in den statisch bekannten Namensbereichen auftritt und während der Verarbeitung der Abfrage verwendet werden kann.

## Berechnete Elementkonstruktoren

Ein berechneter Elementkonstruktor erstellt einen Elementknoten, für den der Knoteninhalt auf Basis eines eingeschlossenen Ausdrucks berechnet wird.

Das Ergebnis eines berechneten Elementkonstruktors ist ein neuer Elementknoten, der über eine eigene Knotenidentität verfügt. Sämtliche Attributknoten und untergeordneten Knoten des neuen Elementknotens sind ebenfalls neue Knoten mit einer eigenen Knotenidentität, selbst wenn es sich um Kopien bereits vorhandener Knoten handeln sollte.

### Syntax



**element**
Ein Schlüsselwort, das angibt, dass ein Elementknoten erstellt wird.

*Elementname*
Der qualifizierte Name (QName) des zu erstellenden Elements. Enthält der Wert für *Elementname* ein Namensbereichspräfix, wird das Präfix unter Verwendung der statisch bekannten Namensbereiche in eine Namensbereichs-URI aufgelöst. Hat *Elementname* kein Namensbereichspräfix, wird der Name implizit durch den Standardnamensbereich des Eements/Typs qualifiziert. Der aus der Auswertung von *Elementname* resultierende erweiterte QName wird als Name des erstellten Elementknotens verwendet.

*Inhaltsausdruck*
Ein Ausdruck, der den Inhalt des erstellten Elementknotens generiert. Der Wert von *Inhaltsausdruck* kann eine beliebige Sequenz aus Knoten und atomaren Werten sein. Der Wert von *Inhaltsausdruck* kann verwendet werden, um sowohl den Inhalt als auch die Attribute des erstellten Knotens zu berechnen. Für jeden von *Inhaltsausdruck* zurückgegebenen Knoten wird eine neue Kopie des Knotens und sämtlicher seiner untergeordneten Elemente (Nachkommen) erstellt, wobei deren ursprünglichen Typenannotationen beibehalten werden. Sämtliche von *Inhaltsausdruck* zurückgegebenen Attributknoten müssen am Anfang der Knotensequenz stehen (vor allen anderen Knoten). Diese Attributknoten werden als Attribute des erstellten Elements verwendet. Alle von *Inhaltsausdruck* zurückgegebenen Element-, Inhalts- oder Verarbeitungsanweisungsknoten werden als direkt untergeordnete Elemente (Kinder) des neu erstellten Knotens verwendet. Alle von *Inhaltsausdruck* zurückgegebenen atomaren Werte werden in Zeichenfolgen umgesetzt und in Textknoten gespeichert, die als Kinder des erstellten Knotens verwendet werden. Benachbarte Textknoten werden zu einem einzigen Textknoten zusammengeführt.

### Beispiel

Mit dem nachstehenden Ausdruck erstellt ein berechneter Elementkonstruktor eine geänderte Kopie eines vorhandenen Elements. Angenommen, die Variable `$e` wird an ein Element mit numerischem Inhalt gebunden. Dieser Konstruktor erstellt ein neues Element namens `length`, das dieselben Attribute wie `$e` aufweist und dessen numerischer Inhalt dem doppelten Inhalt von `$e` entspricht:

```
element length {$e/@*, 2 * fn:data($e)}
```

In diesem Beispiel gilt: Wenn die Variable `$e` an den Ausdruck `let $e := <length units="inches">{5}</length>` gebunden wird, ist das Ergebnis des Beispielausdrucks das Element `<length units="inches">10</length>`.

## Berechnete Attributkonstruktoren

Ein berechneter Attributkonstruktor erstellt einen Attributknoten, für den der Attributwert auf Basis eines eingeschlossenen Ausdrucks berechnet wird.

Das Ergebnis eines berechneten Attributkonstruktors ist ein neuer Attributknoten, der über eine eigene Knotenidentität verfügt.

**Anmerkung:** Um einen Attributknoten direkt zu erstellen, müssen Sie das Attribut in einem direkten Elementkonstruktor deklarieren.

### Syntax

**attribute**
Ein Schlüsselwort, das angibt, dass ein Attributknoten erstellt wird.

*Attributname*
Der qualifizierte Name (QName) des zu erstellenden Attributs. Enthält der Wert für *Attributname* ein Namensbereichspräfix, wird das Präfix unter Verwendung der statisch bekannten Namensbereiche in eine Namensbereichs-URI aufgelöst. Enthält *Attributname* kein Namensbereichspräfix, befindet sich das Attribut in keinem Namensbereich. Der aus der Auswertung von *Attributname* resultierende erweiterte QName wird als Name des erstellten Attributknotens verwendet. Der erweiterte QName jedes Attributs in einem Element muss eindeutig sein, da der Ausdruck ansonsten einen Fehler ergibt.

*Attributwertausdruck*
Ein Ausdruck, der den Wert des Attributknotens generiert. Während der Verarbeitung wird das Ergebnis von *Attributwertausdruck* atomisiert, und jeder atomare Werte in der daraus resultierenden Sequenz wird in eine Zeichenfolge umgesetzt. Die einzelnen Zeichenfolgen, die sich aus der Umsetzung ergeben, werden mit einem Zwischenleerschritt verknüpft. Die verknüpfte Zeichenfolge wird als Wert des erstellten Attributknotens verwendet.

### Beispiel

Der folgende berechnete Attributkonstruktor erstellt ein Attribut namens 'size' mit dem Wert "7".

```
attribute size {4 + 3}
```

## Dokumentknotenkonstruktoren

Alle Dokumentknotenkonstruktoren sind berechnete Konstruktoren. Ein Dokumentknotenkonstruktor erstellt einen Dokumentknoten, für den der Knoteninhalt auf Basis eines eingeschlossenen Ausdrucks berechnet wird. Ein Dokumentknotenkonstruktor ist dann von Nutzen, wenn das Ergebnis einer Abfrage ein vollständiges Dokument ist.

Das Ergebnis eines Dokumentknotenkonstruktors ist ein neuer Dokumentknoten, der über eine eigene Knotenidentität verfügt.

**Wichtig:** Für den erstellten Dokumentknoten wird keine Gültigkeitsprüfung ausgeführt. Der XQuery-Dokumentknotenkonstruktor erzwingt nicht die Regeln von XML 1.0, denen die Struktur von XML-Dokumenten unterliegen. So muss ein Dokumentknoten beispielsweise nicht unbedingt genau ein untergeordnetes Element (Kind) haben, das ein Elementknoten ist.

### Syntax

**document**
Ein Schlüsselwort, das angibt, dass ein Dokumentknoten erstellt wird.

*Inhaltsausdruck*
Ein Ausdruck, der den Inhalt des erstellten Dokumentknotens generiert. Der Wert von *Inhaltsausdruck* kann eine beliebige Sequenz aus Knoten und atomaren Werten sein, mit Ausnahme eines Attributknotens. Attributknoten in der Inhaltssequenz führen zu einem Fehler. Dokumentknoten in der Inhaltssequenz werden durch ihre jeweiligen Kinder ersetzt. Für jeden von *Inhaltsausdruck* zurückgegebenen Knoten wird eine neue Kopie des Knotens und sämtlicher seiner untergeordneten Elemente erstellt, die ihre ursprünglichen Typenannotationen beibehalten. Alle vom Inhaltsausdruck zurückgegebenen atomaren Werte werden in Zeichenfolgen umgesetzt und in Textknoten gespeichert, die als Kinder des erstellten Dokumentknotens verwendet werden. Benachbarte Textknoten werden zu einem einzigen Textknoten zusammengeführt.

### Beispiel

Der folgende Dokumentknotenkonstruktor umfasst einen Inhaltsausdruck, der ein XML-Dokument zurückgibt, das ein Stammelement namens `customer-list` enthält:

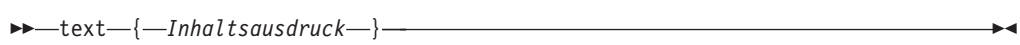
```
document
{
<customer-list>
   {db2-fn:xmlcolumn('MYSCHEMA.CUSTOMER.INFO')/ns1:customerinfo/name}
</customer-list>
}
```

## Textknotenkonstruktoren

Alle Textknotenkonstruktoren sind berechnete Konstruktoren. Ein Textknotenkonstruktor erstellt einen Textknoten, für den der Knoteninhalt auf Basis eines eingeschlossenen Ausdrucks berechnet wird.

Das Ergebnis eines Textknotenkonstruktors ist ein neuer Textknoten, der über eine eigene Knotenidentität verfügt.

### Syntax




**text**
Ein Schlüsselwort, das angibt, dass ein Textknoten erstellt wird.

*Inhaltsausdruck*
Ein Ausdruck, der den Inhalt des erstellten Textknotens generiert. Während der Verarbeitung wird das Ergebnis von *Inhaltsausdruck* atomisiert, und jeder atomare Wert in der daraus resultierenden Sequenz wird in eine Zeichenfolge umgesetzt. Die einzelnen Zeichenfolgen, die sich aus der Umsetzung ergeben, werden mit einem Zwischenleerschritt verknüpft. Die verknüpfte Zeichenfolge wird als Wert des erstellten Textknotens verwendet. Ergibt die Atomisierung eine leere Sequenz, wird kein Textknoten erstellt.

**Anmerkung:** Ein Textknotenkonstruktor kann verwendet werden, um einen Textknoten zu erstellen, der eine Zeichenfolge mit Nulllänge enthält. Wird dieser Textknoten jedoch im Inhalt eines erstellten Element- oder Dokumentknotens verwendet, wird der Textknoten gelöscht oder mit einem anderen Textknoten gemischt.

### Beispiel

Der folgende Konstruktor erstellt einen Textknoten mit der Zeichenfolge ″Hello″ als Inhalt:

```
text {"Hello"}
```

## Verarbeitungsanweisungskonstruktoren

Verarbeitungsanweisungskonstruktoren erstellen Verarbeitungsanweisungsknoten. XQuery stellt für die Erstellung von Verarbeitungsanweisungsknoten sowohl direkte als auch berechnete Konstruktoren zur Verfügung.

Der erstellte Knoten weist die folgenden Knotenmerkmale auf:

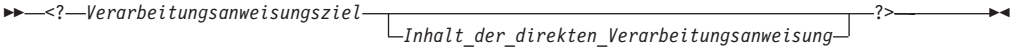
**Ein Zielmerkmal (target)**
Identifiziert die Anwendung, an die die Verarbeitungsanweisung übertragen wird.

**Ein Inhaltsmerkmal (content)**
Gibt den Inhalt der Verarbeitungsanweisung an.

### Direkte Verarbeitungsanweisungskonstruktoren

Direkte Verarbeitungsanweisungskonstruktoren verwenden eine XML-artige Notation zum Erstellen von Verarbeitungsanweisungsknoten.

#### Syntax




*Verarbeitungsanweisungsziel*
Ein Name ohne Doppelpunkte (NCName), der den Namen der Verarbeitungsanwendung darstellt, an die die Verarbeitungsanweisung übertragen wird. Das Ziel einer Verarbeitungsanweisung darf nicht aus den Zeichen ″XML″ bestehen, und zwar in keinerlei Kombination aus Groß- und Kleinbuchstaben.

*Inhalt_der_direkten_Verarbeitungsanweisung*
Eine Reihe von Zeichen, die den Inhalt der Verarbeitungsanweisung angeben. Der Inhalt einer Verarbeitungsanweisung darf nicht die Zeichenfolge ?> enthalten.

#### Beispiel

Der folgende Konstruktor erstellt einen Verarbeitungsanweisungsknoten:

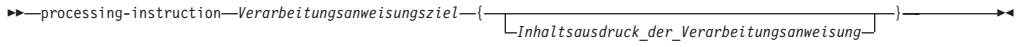
```
<?format role="output" ?>
```

### Berechnete Verarbeitungsanweisungskonstruktoren

Ein berechneter Verarbeitungsanweisungskonstruktor erstellt einen Verarbeitungsanweisungsknoten, für den der Knoteninhalt auf Basis eines eingeschlossenen Ausdrucks berechnet wird.

Das Ergebnis eines berechneten Verarbeitungsanweisungskonstruktors ist ein neuer Verarbeitungsanweisungsknoten, der über eine eigene Knotenidentität verfügt.

### Syntax




**processing-instruction**
Ein Schlüsselwort, das angibt, dass ein Verarbeitungsanweisungsknoten erstellt wird.

*Verarbeitungsanweisungsziel*
Ein Name ohne Doppelpunkte (NCName), der den Namen der Verarbeitungsanwendung darstellt, an die die Verarbeitungsanweisung übertragen wird. Dieser Name muss dem von *Namespaces in XML* (Namensbereiche in XML) definierten Format für NCNames entsprechen.

*Inhaltsausdruck_der_Verarbeitungsanweisung*
Ein Ausdruck, der den Inhalt des Verarbeitungsanweisungsknotens generiert. Während der Verarbeitung wird das Ergebnis von *Inhaltsausdruck_der_Verarbeitungsanweisung* atomisiert, und jeder atomare Werte in der daraus resultierenden Sequenz wird in eine Zeichenfolge umgesetzt. Die einzelnen Zeichenfolgen, die sich aus der Umsetzung ergeben, werden mit einem Zwischenleerschritt verknüpft. Führende Leerzeichen werden entfernt, und die verknüpfte Zeichenfolge wird als Inhalt des Verarbeitungsanweisungsknotens verwendet. Ergibt die Atomisierung eine leere Sequenz, wird die Sequenz durch eine Zeichenfolge mit Nulllänge ersetzt. Die Inhaltssequenz darf nicht die Zeichenfolge `"?>"` enthalten.

### Beispiel

Der folgende berechnete Konstruktor erstellt die Verarbeitungsanweisung `<?audio-output beep?>`:

```
processing-instruction audio-output {"beep"}
```

## Kommentarkonstruktoren

Mithilfe von Kommentarkonstruktoren werden Kommentarknoten erstellt. XQuery stellt für die Erstellung von Kommentarknoten sowohl direkte als auch berechnete Konstruktoren zur Verfügung.

### Direkte Kommentarkonstruktoren

Direkte Kommentarkonstruktoren verwenden eine XML-artige Notation zum Erstellen von Kommentarknoten.

### Syntax

*Direkter_Kommentarinhalt*
Eine Reihe von Zeichen, die den Inhalt des Kommentars angeben. Der Inhalt eines Kommentars darf nicht zwei aufeinander folgende Bindestriche enthalten und nicht mit einem Bindestrich enden.

### Beispiel

Der folgende Konstruktor erstellt einen Kommentarknoten:

```
<!-- Dies ist ein XML-Kommentar. -->
```

### Berechnete Kommentarkonstruktoren

Ein berechneter Kommentarkonstruktor erstellt einen Kommentarknoten, für den der Knoteninhalt auf Basis eines eingeschlossenen Ausdrucks berechnet wird.

Das Ergebnis eines berechneten Kommentarkonstruktors ist ein neuer Kommentarknoten, der über eine eigene Knotenidentität verfügt.

### Syntax

**comment**
Ein Schlüsselwort, das angibt, dass ein Kommentarknoten erstellt wird.

*Kommentarinhalt*
Ein Ausdruck, der den Inhalt des Kommentars generiert. Während der Verarbeitung wird das Ergebnis von *Kommentarinhalt* atomisiert, und jeder atomare Werte in der atomisierten Sequenz wird in eine Zeichenfolge umgesetzt. Die einzelnen Zeichenfolgen, die sich aus der Umsetzung ergeben, werden mit einem Zwischenleerschritt verknüpft. Ergibt die Atomisierung eine leere Sequenz, wird die Sequenz durch eine Zeichenfolge mit Nulllänge ersetzt. Die Inhaltssequenz darf nicht zwei aufeinander folgende Bindestriche enthalten und nicht mit einem Bindestrich enden.

### Beispiel

Der folgende berechnete Konstruktor erstellt den Kommentar `<!--Houston, we have a problem.-->`:

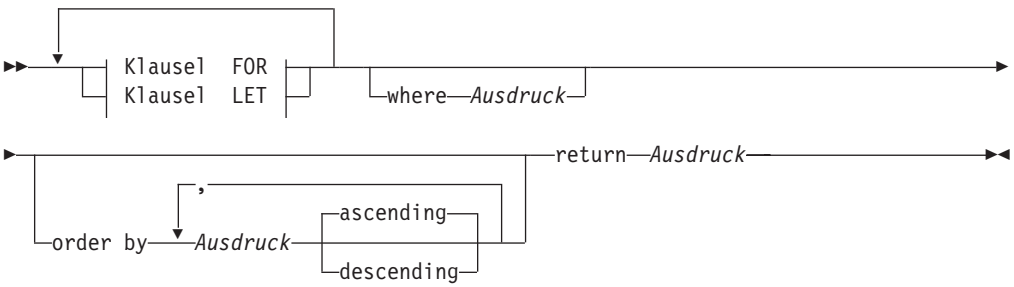
```
let $homebase := "Houston"
return comment {fn:concat($homebase, ", we have a problem.")}
```

## FLWOR-Ausdrücke

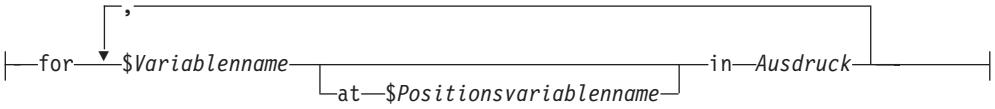
FLWOR-Ausdrücke durchlaufen über Sequenzen und binden Variablen an Zwischenergebnisse. FLWOR-Ausdrücke sind nützlich, um Verknüpfungen (Joins) zwischen mindestens zwei Dokumenten zu erstellen, Daten zu restrukturieren und das Ergebnis zu sortieren.

## Syntax von FLWOR-Ausdrücken

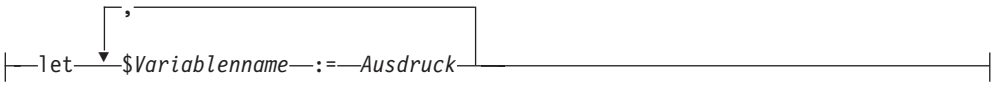
Ein FLWOR-Ausdruck besteht aus den folgenden Klauseln, von denen einige optional sind: **FOR**, **LET**, **WHERE**, **ORDER BY** und **RETURN**.



**Klausel FOR:**




**Klausel LET:**




**for**
Das Schlüsselwort, das eine Klausel vom Typ **FOR** einleitet. Eine Klausel **FOR** durchläuft das Ergebnis von *Ausdruck* und bindet *Variablenname* an jedes Element, das von *Ausdruck* zurückgegeben wird.

**let** Das Schlüsselwort, das eine Klausel vom Typ **LET** einleitet. Eine Klausel **LET** bindet *Variablenname* an das gesamte Ergebnis von *Ausdruck*.

*Variablenname*
Der Name der Variablen, die an das Ergebnis von *Ausdruck* gebunden werden soll.

*Positionsvariablenname*
Der Name einer optionalen Variablen, die innerhalb des Eingabedatenstroms an die Position des Elements gebunden wird, das von jeder Iteration der Klausel **FOR** gebunden wird.

*Ausdruck*
Ein beliebiger XQuery-Ausdruck. Wenn der Ausdruck einen Kommaoperator auf oberster Ebene enthält, muss der Ausdruck in runde Klammern eingeschlossen werden.

**where**
Das Schlüsselwort, das eine Klausel vom Typ **WHERE** einleitet. Eine Klausel **WHERE** filtert die Tupeln von Variablenbindungen, die von Klauseln vom Typ **FOR** und **LET** generiert werden.

**order by**
Die Schlüsselwörter, die eine Klausel vom Typ **ORDER BY** einleiten. Eine Klausel **ORDER BY** gibt die Reihenfolge an, in der Werte von der Klausel **RETURN** verarbeitet werden sollen.

**ascending**
Gibt an, dass Sortierschlüssel in aufsteigender Reihenfolge verarbeitet werden.

**descending**
Gibt an, dass Sortierschlüssel in absteigender Reihenfolge verarbeitet werden.

**return**
Das Schlüsselwort, das eine Klausel vom Typ **RETURN** einleitet. Der Ausdruck in der Klausel **RETURN** wird für jeden Tupel von gebundenen Variablen, der von den Klauseln **FOR**, **LET**, **WHERE** und **ORDER BY** generiert wird, einmal ausgewertet. Enthält die Klausel **RETURN** einen Nichtaktualisierungsausdruck, handelt es sich beim FLWOR-Ausdruck um einen Nichtaktualisierungs-

ausdruck. Die Ergebnisse aller Auswertungen der Klausel RETURN werden zu einer einzigen Sequenz verknüpft, die das Ergebnis des FLWOR-Ausdrucks darstellt.

Enthält die Klausel **RETURN** einen Aktualisierungsausdruck, handelt es sich beim FLWOR-Ausdruck um einen Aktualisierungsausdruck. Ein FLWOR-Aktualisierungsausdruck muss innerhalb der Klausel **MODIFY** eines Umsetzungsausdrucks angegeben werden. Das Ergebnis des FLWOR-Aktualisierungsausdrucks ist eine Liste von Aktualisierungen. Der übergeordnete Umsetzungsausdruck führt die Aktualisierungen aus, nachdem sie mit anderen Aktualisierungen gemischt wurden, die von anderen Aktualisierungsausdrücken innerhalb der Klausel **MODIFY** des Umsetzungsausdrucks zurückgegeben worden sind.

## Klauseln FOR und LET

Die Klausel **FOR** oder **LET** in einem FLWOR-Ausdruck bindet mindestens eine Variable an Werte, die in anderen Klauseln des betreffenden FLWOR-Ausdrucks verwendet werden.

### Klauseln FOR

Eine Klausel **FOR** durchläuft das Ergebnis eines Ausdrucks und bindet eine Variable an jedes Element in der Sequenz.

Der einfachste Typ einer Klausel **FOR** enthält eine Variable und einen zugeordneten Ausdruck. In dem folgenden Beispiel enthält die Klausel **FOR** eine Variable namens `$i` sowie einen Ausdruck, der die Zeichenfolge `(1, 2, 3)`erstellt:

```
for $i in (1, 2, 3)
return <output>{$i}</output>
```

Bei der Auswertung der Klausel **FOR** werden drei Variablenbindungen erstellt (eine Bindung für jedes Element in der Sequenz):

```
$i = 1
$i = 2
$i = 3
```

Die Klausel **RETURN** in dem Beispiel wird für jede Bindung einmal ausgeführt. Der Ausdruck resultiert in folgender Ausgabe:

```
<output>1</output>
<output>2</output>
<output>3</output>
```

Eine Klausel **FOR** kann auch mehrere Variablen enthalten, von denen jede an das Ergebnis eines Ausdrucks gebunden wird. In dem nachstehenden Beispiel enthält die Klausel **FOR** zwei Variablen (`$a` und `$b`), sowie Ausdrücke, die die Sequenzen `1 2` und `4 5` erstellen:

```
for $a in (1, 2), $b in (4, 5)
return <output>{$a, $b}</output>
```

Bei der Auswertung der Klausel **FOR** wird ein Tupel von Variablenbindungen für jede Wertekombination erstellt. Daraus ergeben sich vier Tupeln von Variablenbindungen:

```
($a = 1, $b = 4)
($a = 2, $b = 4)
($a = 1, $b = 5)
($a = 2, $b = 5)
```

Die Klausel **RETURN** in dem Beispiel wird für jeden Tupel von Bindungen einmal ausgeführt. Der Ausdruck resultiert in folgender Ausgabe:

```
<output>1 4</output>
<output>2 4</output>
<output>1 5</output>
<output>2 5</output>
```

Wenn die Auswertung des Bindeausdrucks eine leere Sequenz ergibt, wird keine Bindung für **FOR** generiert, und es erfolgt kein Durchlauf (Iteration) des Ergebnisses. In dem nachstehenden Beispiel ergibt die Auswertung der Bindesequenz eine leere Sequenz, und es wird keine Iteration ausgeführt. Die Knotensequenz in der Klausel **RETURN** wird nicht zurückgegeben:

```
for $node in (<a test = "b" />, <a test = "c" />, <a test = "d" /> )[@test = "1"]
return
  <test>
    Sample return response
  </test>
```

## Positionsgebundene Variablen in Klauseln vom Typ FOR

Jeder in einer Klausel **FOR** gebundenen Variablen kann eine positionsgebundene Variable zugeordnet sein, die gleichzeitig gebunden wird. Vor dem Namen der positionsgebundenen Variablen steht das Schlüsselwort **at**. Wenn eine Variable die Elemente in einer Sequenz durchläuft, durchläuft die positionsgebundene Variable die ganzen Zahlen, die die Positionen der betreffenden Elemente in ihrer Reihenfolge (beginnend mit 1) darstellen.

In dem nachstehenden Beispiel enthält die Klausel **FOR** eine Variable namens `$cat` und einen Ausdruck, der die Sequenz `("Persian", "Calico", "Siamese")` erstellt. Außerdem enthält die Klausel die positionsgebundene Variable `$i`, auf die in einem Attributkonstruktor verwiesen wird, um den Wert des Attributs `order` zu berechnen:

```
for $cat at $i in ("Persian", "Calico", "Siamese")
return <cat order = "{$i}"> { $cat } </cat>
```

Bei der Auswertung der Klausel **FOR** werden drei Tupeln von Variablenbindungen erstellt, von denen jeder Tupel eine Bindung für die positionsgebundene Variable enthält:

```
($i = 1, $cat = "Persian")
($i = 2, $cat = "Calico")
($i = 3, $cat = "Siamese")
```

Die Klausel **RETURN** in dem Beispiel wird für jeden Tupel von Bindungen einmal ausgeführt. Der Ausdruck resultiert in folgender Ausgabe:

```
<cat order = "1">Persian</cat>
<cat order = "2">Calico</cat>
<cat order = "3">Siamese</cat>
```

Obwohl jedes Ausgabeelement ein Attribut vom Typ 'order' enthält, kann die tatsächliche Reihenfolge der Elemente im Ausgabedatenstrom nur dann garantiert werden, wenn der FLWOR-Ausdruck eine Klausel vom Typ **ORDER BY** wie beispielsweise `order by $i` enthält. Die positionsgebundene Variable stellt die Ordinalposition eines Wertes in der Eingabesequenz und nicht in der Ausgabesequenz dar.

## Klauseln LET

Eine Klausel **LET** bindet eine Variable an das gesamte Ergebnis eines Ausdrucks. Eine Klausel LET führt keine Iteration durch.

Der einfachste Typ einer Klausel **LET** enthält eine Variable und einen zugeordneten Ausdruck. In dem folgenden Beispiel enthält die Klausel **LET** eine Variable namens `$j` sowie einen Ausdruck, der die Zeichenfolge `(1, 2, 3)`erstellt.

```
let $j := (1, 2, 3)
return <output>{$j}</output>
```

Bei Auswertung der Klausel **LET** wird eine einzige Bindung für die gesamte Sequenz erstellt, die sich aus der Auswertung des Ausdrucks ergibt:

```
$j = 1 2 3
```

Die Klausel **RETURN** in dem Beispiel wird einmal ausgeführt. Der Ausdruck resultiert in folgender Ausgabe:

```
<output>1 2 3</output>
```

Eine Klausel **LET** kann mehrere Variablen enthalten. Im Gegensatz zu einer Klausel **FOR** bindet eine Klausel **LET** jede Variable an das Ergebnis ihres zugeordneten Ausdrucks - ohne Iteration. In dem nachstehenden Beispiel enthält die Klausel **LET** zwei Variablen, `$a` und `$b`, sowie Ausdrücke, die die Sequenzen `1 2` und `4 5` erstellen:

```
let $a := (1, 2), $b := (4, 5)
return <output>{$a, $b}</output>
```

Bei Auswertung der Klausel **LET** wird ein Tupel von Variablenbindungen erstellt:

```
($a = 1 2, $b = 4 5)
```

Die Klausel **RETURN** in dem Beispiel wird einmal für den Tupel ausgeführt. Der Ausdruck resultiert in folgender Ausgabe:

```
<output>1 2 4 5</output>
```

Wenn die Auswertung des Bindeausdrucks eine leere Sequenz ergibt, wird eine LET-Bindung erstellt, die eine leere Zeichenfolge enthält.

## Klauseln FOR und LET in demselben Ausdruck

Wenn ein FLWOR-Ausdruck sowohl Klauseln vom Typ **FOR** als auch Klauseln vom Typ **LET** enthält, werden die von den Klauseln vom Typ **LET** generierten Variablenbindungen den von den Klauseln vom Typ **FOR** generierten Variablenbindungen hinzugefügt.

In dem nachstehenden Beispiel enthält die Klausel **FOR** eine Variable namens `$a` sowie einen Ausdruck, der die Zeichenfolge `(1, 2, 3)`erstellt. Die Klausel **LET** umfasst eine Variable namens `$b` sowie einen Ausdruck, der die Zeichenfolge `(4, 5, 6)` erstellt:

```
for $a in (1, 2, 3)
let $b := (4, 5, 6)
return <output>{$a, $b}</output>
```

Die Klauseln **FOR** und **LET** in diesem Beispiel resultieren in drei Tupeln von Bindungen. Die Anzahl der Tupel wird von der Klausel **FOR** festgelegt:

```
($a = 1, $b = 4 5 6)
($a = 2, $b = 4 5 6)
($a = 3, $b = 4 5 6)
```

Die Klausel **RETURN** in dem Beispiel wird für jeden Tupel von Bindungen einmal ausgeführt. Der Ausdruck resultiert in folgender Ausgabe:

```
<output>1 4 5 6</output>
<output>2 4 5 6</output>
<output>3 4 5 6</output>
```

## Klauseln FOR und LET im Vergleich

Die Klauseln **FOR** und **LET** binden zwar beide Variablen, doch erfolgt die Variablenbindung auf unterschiedliche Art und Weise.

Die folgende Tabelle enthält Beispiele zum Vergleich der Ergebnisse, die von FLWOR-Ausdrücken zurückgegeben werden, die ähnliche Klauseln vom Typ **FOR** und **LET** enthalten.

*Tabelle 31. Vergleich der Klauseln* **FOR** *und* **LET** *in FLWOR-Ausdrücken*

| **Beschreibung der Abfrage** | **FLWOR-Ausdruck** | **Ergebnis** |
|---|---|---|
| Bindung einer einzelnen Variablen | `for $i in ("a", "b", "c")`<br>`return <output>{$i}</output>` | `<output>a</output>`<br>`<output>b</output>`<br>`<output>c</output>` |
| | `let $i := ("a", "b", "c")`<br>`return <output>{$i}</output>` | `<output>a b c</output>` |
| Bindung mehrerer Variablen | `for $i in ("a", "b"), $j in ("c", "d")`<br>`return <output>{$i, $j}</output>` | `<output>a c</output>`<br>`<output>b c</output>`<br>`<output>a d</output>`<br>`<output>b d</output>` |
| | `let $i := ("a", "b"), $j := ("c", "d")`<br>`return <output>{$i, $j}</output>` | `<output>a b c d</output>` |

**Anmerkung:** Da die Ausdrücke in dieser Tabelle keine Klausel vom Typ **ORDER BY** enthalten, ist die Reihenfolge der Ausgabeelemente nicht deterministisch.

## Geltungsbereich von Variablen in Klauseln vom Typ FOR und LET

Eine Variable, die in einer Klausel vom Typ **FOR** oder **LET** gebunden ist, gilt für alle Unterausdrücke des FLWOR-Ausdrucks, die nach der Variablenbindung vorkommen. Dies bedeutet, dass eine Klausel **FOR** oder **LET** auf Variablen verweisen kann, die in früheren Klauseln oder in früheren Bindungen in derselben Klausel gebunden sind.

Im nachstehenden Beispiel umfasst ein FLWOR-Ausdruck die folgenden Klauseln:

- Eine Klausel vom Typ **LET**, die die Variable `$orders` bindet.
- Eine Klausel vom Typ **FOR**, die auf die Variable `$orders` verweist und die Variable `$i` bindet.
- Eine weitere Klausel vom Typ **LET**, die sowohl auf die Variable `$orders` als auch die Variable `$i` verweist und die Variable `$c` bindet.

In diesem Beispiel werden alle unterschiedlichen Artikelnummern (itemno) in einer Gruppe von Bestellungen (orders) gefunden, und es wird die Anzahl der Bestellungen für jede unterschiedliche Artikelnummer zurückgegeben.

```
let $orders := db2-fn:xmlcolumn("ORDERS.XMLORDER")
for $i in fn:distinct-values($orders/order/itemno)
let $c := fn:count($orders/order[itemno = $i])
return
```

```
<ordercount>
  <itemno> {$i} </itemno>
  <count> {$c} </count>
</ordercount>
```

**Wichtig:** Die Klauseln vom Typ **FOR** und **LET** in einem FLWOR-Ausdruck können einen Variablennamen jeweils immer nur einmal binden.

## Klauseln WHERE

Eine Klausel vom Typ **WHERE** in einem FLWOR-Ausdruck filtert die Tupel von Variablenbindungen, die von Klauseln vom Typ **FOR** und **LET** generiert werden.

Die Klausel **WHERE** gibt eine Bedingung an, die auf jeden Tupel von Variablenbindungen angewandt wird. Tritt die Bedingung als wahr ein (d. h. ergibt der Ausdruck den effektiven Booleschen Wert 'true'), wird der Tupel beibehalten und seine Bindungen werden bei Ausführung der Klausel **RETURN** verwendet. Andernfalls wird der Tupel verworfen.

Im folgenden Beispiel bindet die Klausel **FOR** die Variablen `$x` und `$y` an Sequenzen aus numerischen Werten:

```
for $x in (1.5, 2.6, 1.9), $y in (.5, 1.6, 1.7)
where ((fn:floor($x) eq 1) and (fn:floor($y) eq 1))
return <output>{$x, $y}</output>
```

Bei der Auswertung der Klausel **FOR** werden folgende neun Tupel von Variablenbindungen erstellt:

```
($x = 1.5, $y = .5)
($x = 2.6, $y = .5)
($x = 1.9, $y = .5)
($x = 1.5, $y = 1.6)
($x = 2.6, $y = 1.6)
($x = 1.9, $y = 1.6)
($x = 1.5, $y = 1.7)
($x = 2.6, $y = 1.7)
($x = 1.9, $y = 1.7)
```

Die Klausel **WHERE** filtert diese Tupel, wobei die folgenden Tupel beibehalten werden:

```
($x = 1.5, $y = 1.6)
($x = 1.9, $y = 1.6)
($x = 1.5, $y = 1.7)
($x = 1.9, $y = 1.7)
```

Die Klausel **RETURN** wird einmal für jeden noch vorhandenen Tupel ausgeführt, und der Ausdruck resultiert in der folgenden Ausgabe:

```
<output>1.5 1.6</output>
<output>1.9 1.6</output>
<output>1.5 1.7</output>
<output>1.9 1.7</output>
```

Da der Ausdruck in diesem Beispiel keine Klausel vom Typ **ORDER BY** enthält, ist die Reihenfolge der Ausgabeelemente nicht deterministisch.

## Klauseln ORDER BY

Eine Klausel vom Typ **ORDER BY** in einem FLWOR-Ausdruck gibt die Reihenfolge an, in der Werte von der Klausel **RETURN** verarbeitet werden. Ist keine Klausel **ORDER BY** vorhanden, werden die Ergebnisse eines FLWOR-Ausdrucks in einer nicht deterministischen Reihenfolge zurückgegeben.

Eine Klausel vom Typ **ORDER BY** enthält mindestens eine Sortierspezifikation. Sortierspezifikationen werden verwendet, um die Tupeln von Variablenbindungen, die nach der Filterung durch die Klausel **WHERE** verbleiben, erneut zu sortieren. Die resultierende Reihenfolge legt die Reihenfolge fest, in der die Klausel **RETURN** ausgewertet wird.

Jede Sortierspezifikation besteht aus einem Ausdruck, der zwecks Erstellung eines Sortierschlüssels ausgewertet wird, und aus einem Sortierwert, der die Sortierreihenfolge (aufsteigend oder absteigend) für die Sortierschlüssel angibt. Die relative Reihenfolge von zwei Tupeln wird ermittelt, indem die Werte ihrer Sortierschlüssel von links nach rechts verglichen werden.

Im folgenden Beispiel umfasst ein FLWOR-Ausdruck eine Klausel **ORDER BY**, die Produkte in absteigender Reihenfolge nach Preis sortiert:

```
<price_list>{
    for $prod in db2-fn:xmlcolumn('PRODUCT.DESCRIPTION')/product/description
    order by xs:decimal($prod/price) descending
    return
    <product>{$prod/name, $prod/price}</product>}
</price_list>
```

Während der Verarbeitung der Klausel **ORDER BY** wird der Ausdruck in der Sortierspezifikation für jeden Tupel ausgewertet, der von der Klausel **FOR** erstellt wird. Für den ersten Tupel wird vom Ausdruck `xs:decimal($prod/price)` der Wert 9.99 zurückgegeben. Anschließend wird der Ausdruck für den nächsten Tupel ausgewertet und der Wert 19.99 zurückgegeben. Da die Sortierspezifikation angibt, dass Artikel in absteigender Reihenfolge sortiert werden sollen, wird das Produkt mit dem Preis 19.99 in der Reihenfolge vor das Produkt mit dem Preis 9.99 gestellt. Dieser Sortiervorgang wird so lange fortgesetzt, bis alle Tupel erneut sortiert worden sind. Anschließend wird die Klausel **RETURN** für jeden Tupel im erneut sortierten Tupeldatenstrom jeweils einmal ausgeführt.

Bei Ausführung der Beispielabfrage für die Tabelle PRODUCT.DESCRIPTION der Beispieldatenbank SAMPLE wird das folgende Ergebnis zurückgegeben:

```
<price_list>
    <product>
        <name>Snow Shovel, Super Deluxe 26"</name>
        <price>49.99</price>
    </product>
    <product>
        <name>Snow Shovel, Deluxe 24"</name>
        <price>19.99</price></product>
    <product>
        <name>Snow Shovel, Basic 22"</name>
        <price>9.99</price>
    </product>
    <product>
        <name>Ice Scraper, Windshield 4" Wide</name>
        <price>3.99</price>
    </product>
</price_list>
```

In diesem Beispiel erstellt der Ausdruck in der Sortierspezifikation einen Wert vom Typ 'xs:decimal' aus dem Wert des Elements `price`. Diese Typumsetzung ist notwendig, weil die Typenannotation des Elements `price` im XML-Schema 'xs:string' lautet. Ohne diese Umsetzung würde für das Ergebnis keine numerische Sortierung, sondern eine Zeichenfolgesortierung verwendet werden.

Eine explizite Typumsetzung ist auch dann erforderlich, wenn der dynamische Typ des Sortierschlüsselwerts 'xdt:untypedAtomic' lautet, da die Regeln für den Vergleich von Sortierschlüsseln vorschreiben, dass nicht typisierte Datenatome als Zeichenfolge zu behandeln sind.

**Tipp:** Eine Klausel **ORDER BY** kann in einem FLWOR-Ausdruck verwendet werden, um eine Wertesortierung in einer Abfrage anzugeben, für die ansonsten keine Iteration erforderlich wäre. Beispiel: Der folgende Pfadausdruck gibt eine Liste von `customerinfo`-Elementen mit einer Kunden-ID (Cid) größer als 1000 zurück:

```
db2-fn:xmlcolumn('CUSTOMER.INFO')/customerinfo[@Cid > "1000"]
```

Um diese Elemente in aufsteigender Reihenfolge nach Kundenname zurückzugeben, müsste jedoch ein FLWOR-Ausdruck angegeben werden, der eine Klausel vom Typ **ORDER BY** enthält:

```
for $custinfo in db2-fn:xmlcolumn('CUSTOMER.INFO')/customerinfo
where ($custinfo/@Cid > "1000")
order by $custinfo/name ascending
return $custinfo
```

Der Sortierschlüssel muss kein Teil der Ausgabe sein. Die folgende Abfrage erstellt eine Liste von Produktnamen in absteigender Reihenfolge nach Preis, wobei der Preis selbst nicht in der Ausgabe enthalten ist:

```
for $prod in db2-fn:xmlcolumn('PRODUCT.DESCRIPTION')/product
order by $prod/description/price descending
return $prod/name
```

## Regeln für den Vergleich von Sortierspezifikationen

Das Auswerten und Vergleichen von Sortierspezifikationen unterliegt den folgenden Regeln:

- Der Ausdruck in der Sortierspezifikation wird ausgewertet, und das Ergebnis wird anschließend atomisiert. Das Ergebnis der Atomisierung muss entweder ein einzelner atomarer Wert oder eine leere Sequenz sein. Andernfalls wird ein Fehler zurückgegeben. Das Ergebnis der Auswertung einer Sortierspezifikation wird als Sortierschlüssel bezeichnet.
- Wenn der Typ eines Sortierschlüssels 'xdt:untypedAtomic' lautet, muss der betreffende Schlüssel in den Typ 'xs:string' umgesetzt werden. Durch die konsistente Behandlung nicht typisierter Werte als Zeichenfolgen kann mit der Sortierung begonnen werden, ohne sämtliche Typen aller zu sortierenden Werte kennen zu müssen.
- Wenn die von einer Sortierspezifikation generierten Werte nicht alle denselben Typ aufweisen, werden diese Werte (Schlüssel) mithilfe einer Subtypsubstitution oder Typumstufung in einen gemeinsamen Typ umgesetzt. Schlüssel werden verglichen, indem sie in den kleinsten gemeinsamen Typ umgesetzt werden, der den Operator **gt** unterstützt. Beispiel: Wenn eine Sortierspezifikation eine Liste von Schlüsseln generiert, die sowohl Werte vom Typ 'xs:anyURI' als auch vom Typ 'xs:string' enthält, werden die Schlüssel verglichen, indem der Operator **gt** des Datentyps 'xs:string' verwendet wird. Wenn die Sortierschlüssel, die von einer bestimmten Sortierspezifikation generiert werden, keinen gemeinsamen Typ aufweisen, der den Operator **gt** unterstützt, wird ein Fehler zurückgegeben.
- Die Werte der Sortierschlüssel werden verwendet, um die Reihenfolge zu ermitteln, in der Tupel von gebundenen Variablen an die Klausel RETURN zur Verarbeitung übergeben werden. Die Sortierreihenfolge von Tupeln wird ermittelt, indem ihre Sortierschlüssel von links nach rechts aufgrundlage der folgenden Regeln verglichen werden:

– Bei aufsteigender Sortierreihenfolge werden die Tupel mit Sortierschlüsseln, die größer sind als die anderer Tupel, nach diesen Tupeln sortiert.
– Bei absteigender Sortierreihenfolge werden die Tupel mit Sortierschlüsseln, die größer sind als die anderer Tupel, vor diesen Tupeln sortiert.

Die Größer-als-Beziehungen für Sortierschlüssel sind wie folgt definiert:

– Eine leere Sequenz ist größer als alle anderen Werte.
– Eine Nichtzahl (NaN) wird als ein Wert interpretiert, der größer als alle anderen Werte ist, mit Ausnahme einer leeren Sequenz.
– Ein Wert ist größer als ein anderer Wert, wenn der Parameter **gt** beim Vergleich der betreffenden Werte den Wert 'true' (wahr) zurückgibt.
– Keiner der Gleitkommasonderwerte 'positive Null' oder 'negative Null' ist größer als der andere, da sowohl beim Vergleich `+0.0 gt -0.0` als auch beim Vergleich `-0.0 gt +0.0` der Wert 'false' (falsch) zurückgegeben wird.

**Anmerkung:** Tupel mit leerem Sortierschlüssel werden am Ende des Ausgabedatenstroms aufgeführt, wenn die Standardoption **ascending** (aufsteigend) angegeben ist, bzw. am Anfang des Ausgabedatenstroms, wenn die Option **descending** (absteigend) angegeben ist.

## Klauseln RETURN

Eine Klausel vom Typ **RETURN** generiert das Ergebnis eines FLWOR-Ausdrucks.

Die Klausel **RETURN** wird für jeden Tupel von Variablenbindungen, der von den anderen Klauseln des FLWOR-Ausdrucks erstellt wird, jeweils einmal ausgewertet. Die Reihenfolge, in der Tupel von gebundenen Variablen von der Klausel **RETURN** verarbeitet werden, ist nicht deterministisch, sofern im FLWOR-Ausdruck nicht eine Klausel vom Typ **ORDER BY** verwendet wird.

Wenn es sich bei dem Ausdruck in der Klausel **RETURN** um einen Nichtaktualisierungsausdruck handelt, werden die Ergebnisse aller Auswertungen der Klausel **RETURN** verknüpft und bilden zusammen das Ergebnis des FLWOR-Nichtaktualisierungsausdrucks.

Wenn es sich bei dem Ausdruck in der Klausel **RETURN** um einen Aktualisierungsausdruck handelt, ist das Ergebnis aller Auswertungen der Klausel **RETURN** eine Liste von Aktualisierungen. Der übergeordnete Umsetzungsausdruck, der den FLWOR-Ausdruck enthält, führt die Aktualisierungen aus, nachdem sie mit den Aktualisierungen gemischt wurden, die von anderen Aktualisierungsausdrücken innerhalb der Klausel **MODIFY** des Umsetzungsausdrucks zurückgegeben worden sind.

**Tipp:** Innerhalb einer Klausel vom Typ **RETURN** müssen Ausdrücke, die Kommaoperatoren auf oberster Ebene enthalten, in runde Klammern eingeschlossen werden. Da FLWOR-Ausdrücke eine höhere Vorrangstellung besitzen als der Kommaoperator, können Ausdrücke, die Kommaoperatoren auf oberster Ebene enthalten, zu Fehlern oder unerwarteten Ergebnissen führen, wenn keine runden Klammern verwendet werden.

## FLWOR-Beispiele

Diese Beispiele zeigen, wie FLWOR-Ausdrücke in vollständigen Abfragen verwendet werden, um Verknüpfungen (Joins), Gruppierungen und Spaltenberechnungen durchzuführen.

## FLWOR-Ausdruck zum Verknüpfen von XML-Daten

Die folgende Abfrage verknüpft XML-Daten aus den Tabellen PRODUCT und PURCHASEORDER in der Datenbank SAMPLE, um die Namen von Produkten aufzulisten, die 2005 bestellt wurden.

Da sich die Elemente sowohl in den Produktdokumenten (product) als auch in den Bestelldokumenten (PurchaseOrder) in demselben Namensbereich befinden, beginnt die Abfrage mit der Deklaration eines Standardnamensbereichs, damit für die Elementnamen in der Abfrage keine Präfixe erforderlich sind. Die Klausel **FOR** durchläuft die Spalte PURCHASEORDER.PORDER, um insbesondere die Bestellungen zu ermitteln, bei denen der Attributwert für das Bestelldatum (OrderDate) mit ″2005″ beginnt. Für jede Bestellung ordnet die Klausel **LET** der Variable *$parts* die entsprechenden Teilekennungen (partid) zu. Die Klausel **RETURN** listet anschließend die Namen der Produkte auf, die in der Bestellung enthalten sind.

```
declare default element namespace 'http://posample.org';
for $po in db2-fn:xmlcolumn('PURCHASEORDER.PORDER')
  /PurchaseOrder[fn:starts-with(@OrderDate, "2005")]
let $parts := $po/item/partid
return
<ProductList PoNum = "{$po/@PoNum}">
    { db2-fn:xmlcolumn('PRODUCT.DESCRIPTION')
    /product[@pid = $parts]/description/name }
</ProductList>
```

Die Abfrage gibt das folgende Ergebnis zurück:

```
<ProductList xmlns="http://posample.org" PoNum="5001">
    <name>Snow Shovel, Deluxe 24 inch</name>
    <name>Snow Shovel, Super Deluxe 26 inch</name>
    <name>Ice Scraper, Windshield 4 inch</name>
</ProductList>
<ProductList xmlns="http://posample.org" PoNum="5003">
    <name>Snow Shovel, Basic 22 inch</name>
</ProductList>
<ProductList xmlns="http://posample.org" PoNum="5004">
    <name>Snow Shovel, Basic 22 inch</name>
    <name>Snow Shovel, Super Deluxe 26 inch</name>
</ProductList>
```

## FLWOR-Ausdruck zum Gruppieren von Elementen

Die folgende Abfrage gruppiert die Kundennamen in der Tabelle CUSTOMER der Datenbank SAMPLE nach Städten. Die Klausel **FOR** durchläuft die Dokumente der Kundeninformationen (customerinfo) und bindet jedes Stadtelement (city) an die Variable *$city*. Für jede Stadt bindet die Klausel **LET** die Variable *$cust-names* an eine ungeordnete Liste aller Kundennamen in der betreffenden Stadt. Die Abfrage gibt Stadtelemente zurück, die jeweils den Namen einer Stadt und die verschachtelten Namenselemente aller in der betreffenden Stadt lebenden Kunden enthalten.

```
declare default element namespace 'http://posample.org';
for $city in fn:distinct-values(db2-fn:xmlcolumn('CUSTOMER.INFO')
    /customerinfo/addr/city)
let $cust-names := db2-fn:xmlcolumn('CUSTOMER.INFO')
    /customerinfo/name[../addr/city = $city]
order by $city
return <city>{$city, $cust-names} </city>
```

Die Abfrage gibt das folgende Ergebnis zurück:

```
<city xmlns="http://posample.org">Aurora
      <name>Robert Shoemaker</name>
</city>
```

```
<city xmlns="http://posample.org">Markham
      <name>Kathy Smith</name>
                                          <name>Jim Noodle</name>
</city>
<city xmlns="http://posample.org">Toronto
      <name>Kathy Smith</name>
    <name>Matt Foreman</name>
    <name>Larry Menard</name>
</city>
```

## FLWOR-Ausdruck zur Spaltenberechnung von Daten

Die folgende Abfrage gibt den Gesamtumsatz pro Bestellung im Jahr 2005 zurück und erstellt einen HTML-Bericht.

Die Abfrage durchläuft jedes Bestellelement (PurchaseOrder) mit einem Bestelldatum im Jahr 2005 und bindet das Element an die Variable *$po* in der Klausel **FOR**. Der Pfadausdruck `$po/item/` versetzt anschließend die Kontextposition auf jedes Artikelelement (item) innerhalb eines Bestellelements (PurchaseOrder). Der verschachtelte Ausdruck `(price * quantity)` ermittelt den Gesamtumsatz für den betreffenden Artikel. Die Funktion fn:sum addiert die resultierende Sequenz aus den Gesamtumsätzen pro Artikel. Die Klausel **LET** bindet das Ergebnis der Funktion fn:sum an die Variable *$revenue*. Die Klausel **ORDER BY** sortiert die Ergebnisse nach Gesamtumsatz für jede Bestellung. Abschließend erstellt die Klausel **RETURN** eine Zeile in der Berichtstabelle für jede Bestellung.

```
declare default element namespace 'http://posample.org';
<html>
<body>
<h1>PO totals</h1>
<table>
<thead>
    <tr>
        <th>PO Number</th>
        <th>Status</th>
        <th>Revenue</th>
    </tr>
</thead>
<tbody>{
    for $po in db2-fn:xmlcolumn('PURCHASEORDER.PORDER')/
      PurchaseOrder[fn:starts-with(@OrderDate, "2005")]
    let $revenue := sum($po/item/(price * quantity))
    order by $revenue descending
    return
      <tr>
          <td>{string($po/@PoNum)}</td>
          <td>{string($po/@Status)}</td>
          <td>{$revenue}</td>
      </tr>
}
</tbody>
</table>
</body>
</html>
```

Die Abfrage gibt das folgende Ergebnis zurück:

```
<html xmlns="http://posample.org">
<body>
<h1>PO totals</h1>
<table>
<thead>
    <tr>
        <th>PO Number</th>
        <th>Status</th>
```

```
        <th>Revenue</th>
    </tr>
</thead>
<tbody>
    <tr>
        <td>5004</td>
        <td>Shipped</td>
        <td>139.94</td>
    </tr>
    <tr>
        <td>5001</td>
        <td>Shipped</td>
        <td>123.96</td>
    </tr>
    <tr>
        <td>5003</td>
        <td>UnShipped</td>
        <td>9.99</td>
    </tr>
</tbody>
</table>
</body>
</html>
```

Bei Anzeige in einem Browser würde die Ausgabe der Abfrage ähnlich wie in der folgenden Tabelle aussehen:

**PO totals**

| PO Number | Status | Revenue |
|---|---|---|
| 5004 | Shipped | 139.94 |
| 5001 | Shipped | 123.96 |
| 5003 | Unshipped | 9.99 |

## FLWOR-Ausdruck zum Aktualisieren von XML-Daten

In dem folgenden Beispiel wird die Tabelle CUSTOMER aus der DB2-Datenbank SAMPLE verwendet. In der Tabelle CUSTOMER enthält die XML-Spalte INFO die Adresse und Telefonnummer von Kunden.

Der Umsetzungsausdruck erstellt eine Kopie eines XML-Dokuments, das Kundeninformationen enthält. In der Klausel **MODIFY** ändern der FLWOR-Ausdruck und der Ausdruck 'rename' alle Instanzen des Knotennamens 'phone' in den Namen 'phonenumber':

```
  xquery
declare default element namespace 'http://posample.org';
transform
  copy $mycust := db2-fn:sqlquery('select info from customer where cid = 1003')
  modify
    for $phone in $mycust/customerinfo/phone
    return
      do rename $phone as "phonenumber"
  return $mycust
```

Bei Ausführung für die Datenbank SAMPLE ändert der Ausdruck den Knotennamen 'phone' in 'phonenumber' und gibt das folgende Ergebnis zurück:

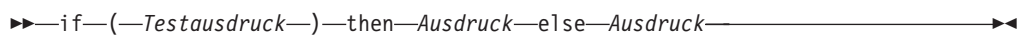
```
<customerinfo xmlns="http://posample.org" Cid="1003">
    <name>Robert Shoemaker</name>
                                        <addr country="Canada">
        <street>1596 Baseline</street>
```

```
        <city>Aurora</city>
                                              <prov-state>Ontario</prov-state>
        <pcode-zip>N8X 7F8</pcode-zip>
                                           </addr>
  <phonenumber type="work">905-555-7258</phonenumber>
  <phonenumber type="home">416-555-2937</phonenumber>
  <phonenumber type="cell">905-555-8743</phonenumber>
  <phonenumber type="cottage">613-555-3278</phonenumber>
</customerinfo>
```

# Bedingungsausdrücke

In Bedingungsausdrücken werden die Schlüsselwörter **if**, **then** und **else** verwendet, um einen von zwei Ausdrücken entsprechend auszuwerten, je nachdem, ob der Wert eines Testausdrucks wahr ('true') oder falsch ('false') ist.

## Syntax




**if** Das Schlüsselwort, das direkt vor dem Testausdruck angegeben wird.

*Testausdruck*
Ein XQuery-Ausdruck, der festlegt, welcher Teil des Bedingungsausdrucks ausgewertet werden soll.

**then**
Wenn der effektive Boolesche Wert des *Testausdrucks* wahr ('true') ist, dann wird der auf dieses Schlüsselwort folgende Ausdruck ausgewertet. Der Ausdruck wird nicht ausgewertet und nicht auf Fehler überprüft, wenn der effektive Boolesche Wert des Testausdrucks falsch ('false') ist.

**else**
Wenn der effektive Boolesche Wert des *Testausdrucks* falsch ('false') ist, dann wird der auf dieses Schlüsselwort folgende Ausdruck ausgewertet. Der Ausdruck wird nicht ausgewertet und nicht auf Fehler überprüft, wenn der effektive Boolesche Wert des Testausdrucks wahr ('true') ist.

*Ausdruck*
Ein XQuery-Ausdruck. Wenn der Ausdruck einen Kommaoperator auf oberster Ebene enthält, muss der Ausdruck in runde Klammern eingeschlossen werden.

Wenn entweder die Verzweigung **then** oder **else** der Bedingung einen Aktualisierungsausdruck enthält, dann ist der Bedingungsausdruck ein Aktualisierungsausdruck. Ein Aktualisierungsausdruck muss sich innerhalb der Klausel **MODIFY** eines Umsetzungsausdrucks befinden.

Bei einem Bedingungsausdruck für Aktualisierung muss jede Verzweigung entweder einen Aktualisierungsausdruck oder eine leere Sequenz enthalten. In Abhängigkeit vom Wert des Testausdrucks wird entweder die Klausel **then** oder **else** ausgewählt und ausgewertet. Das Ergebnis des bedingten Aktualisierungsausdrucks ist eine Liste der von der ausgewählten Verzweigung zurückgegebenen Aktualisierungen. Der übergeordnete Umsetzungsausdruck führt die Aktualisierungen aus, nachdem sie mit den Aktualisierungen gemischt wurden, die von anderen Aktualisierungsausdrücken innerhalb der Klausel **MODIFY** des Umsetzungsausdrucks zurückgegeben worden sind.

### Beispiel

In nachstehendem Beispiel erstellt die Abfrage eine Liste der Elemente vom Typ `product` (Produkt), die ein Attribut namens `basic` einschließen. Der Wert des Attributs `basic` wird bedingt angegeben, je nachdem, ob der Wert des Elements `price` (Preis) kleiner als 10 ist:

```
declare namespace ns1= "http://posample.org";
for $prod in db2-fn:xmlcolumn('PRODUCT.DESCRIPTION')/ns1:product/ns1:description
return (
if (xs:decimal($prod/ns1:price) < 10)
  then <product basic = "true">{fn:data($prod/ns1:name)}</product>
  else <product basic = "false">{fn:data($prod/ns1:name)}</product>)
```

Die Abfrage gibt das folgende Ergebnis zurück:

```
<product basic="true">Snow Shovel, Basic 22"</product>
<product basic="false">Snow Shovel, Deluxe 24"</product>
<product basic="false">Snow Shovel, Super Deluxe 26"</product>
<product basic="true">Ice Scraper, Windshield 4" Wide</product>
```

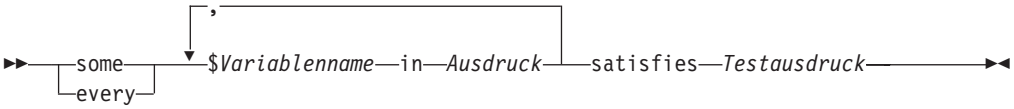
In diesem Beispiel erstellt der Testausdruck einen Wert des Typs xs:decimal aus dem Wert des Elements `price`. Die Funktion 'xs:decimal' wird verwendet, um einen Dezimalvergleich zu erzwingen.

## Quantifizierte Ausdrücke

Quantifizierte Ausdrücke geben den Wert 'true' (wahr) zurück, wenn einige oder alle Elemente in mindestens einer Sequenz eine bestimmte Bedingung erfüllen. Der Wert eines quantifizierten Ausdrucks ist stets 'true' (wahr) oder 'false' (falsch).

Ein quantifizierter Ausdruck beginnt mit einem Quantor (**some** oder **every** für 'einige' bzw. 'alle'), der angibt, ob der Ausdruck eine existenzielle oder eine universelle Quantifizierung durchführt. Auf den Quantor folgt mindestens eine Klausel, die Variablen an Elemente bindet, die von Ausdrücken zurückgegeben werden. Die gebundenen Variablen werden dann in einem Testausdruck referenziert, um zu ermitteln, ob einige oder alle der gebundenen Werte eine bestimmte Bedingung erfüllen.

### Syntax




**some**
Bei Angabe dieses Schlüsselwortes gibt der quantifizierte Ausdruck den Wert 'true' (wahr) zurück, wenn der effektive Boolesche Wert von *Testausdruck* für *mindestens ein* von *Ausdruck* zurückgegebenes Element 'true' ist. Andernfalls gibt der quantifizierte Ausdruck den Wert 'false' (falsch) zurück.

**every**
Bei Angabe dieses Schlüsselwortes gibt der quantifizierte Ausdruck den Wert 'true' (wahr) zurück, wenn der effektive Boolesche Wert von *Testausdruck* für *jedes* von *Ausdruck* zurückgegebene Element 'true' ist. Andernfalls gibt der quantifizierte Ausdruck den Wert 'false' (falsch) zurück.

*Variablenname*
Der Name der Variablen, die an jedes Element im Ergebnis von *Ausdruck*

gebunden werden soll. Variablen, die in einem quantifizierten Ausdruck gebunden werden, gelten für alle Unterausdrücke, die nach der Variablenbindung in dem betreffenden quantifizierten Ausdruck vorkommen.

*Ausdruck*
Ein beliebiger XQuery-Ausdruck. Wenn der Ausdruck einen Kommaoperator auf oberster Ebene enthält, muss der Ausdruck in runde Klammern eingeschlossen werden.

**satisfies**
Das Schlüsselwort, das direkt vor dem Testausdruck angegeben wird.

*Testausdruck*
Ein XQuery-Ausdruck, der die Bedingung angibt, die von einigen oder allen Elementen in der von *Ausdruck* zurückgegebenen Sequenz erfüllt werden muss.

**Anmerkung:** Beim Auftreten von Fehlern kann das Ergebnis eines quantifizierten Vergleichs entweder ein Boolescher Wert oder ein Fehler (error) sein.

### Beispiele

- Der quantifizierte Ausdruck in folgendem Beispiel gibt den Wert 'true' (wahr) zurück, wenn jeder Kunde in der Spalte CUSTOMER.INFO der Beispieldatenbank SAMPLE eine Adresse in Kanada hat:

```
every $cust in db2-fn:xmlcolumn('CUSTOMER.INFO')/customerinfo
satisfies $cust/addr/@country = "Canada"
```

- In den nachstehenden Beispielen wertet jeder quantifizierte Ausdruck seinen Testausdruck für jede Kombination von Werten aus, die an die Variablen `a` und `b` gebunden sind (insgesamt neun Kombinationen).

  Das Ergebnis des folgenden Ausdrucks ist 'true' (wahr):

```
some $a in (3, 5, 9), $b in (1, 3, 5)
satisfies $a * $b = 27
```

  Das Ergebnis des folgenden Ausdrucks ist 'false' (falsch):

```
every $a in (3, 5, 9), $b in (1, 3, 5)
satisfies $a * $b = 27
```

- Das nachstehende Beispiel zeigt, dass das Ergebnis eines quantifizierten Ausdrucks beim Auftreten von Fehlern nicht deterministisch ist. Der Ausdruck kann entweder den Wert 'true' oder einen Fehler (error) zurückgeben, da der Testausdruck für eine Variablenbindung den Wert 'true' und für eine andere Variablenbindung einen Fehler (error) zurückgibt:

```
some $a in (3, 5, "six") satisfies $a * 3 = 9
```

  Ebenso kann der folgende Ausdruck entweder den Wert 'false' oder einen Fehler (error) zurückgeben:

```
every $a in (3, 5, "six") satisfies $a * 3 = 9
```

## Umsetzungsausdrücke

Ein Umsetzungsausdruck erstellt aufgrundlage eines vorhandenen Wertes einen neuen Wert eines bestimmten Typs.

Ein Umsetzungsausdruck verwendet zwei Operanden: einen Eingabeausdruck und einen Zieltyp. Bei der Auswertung des Umsetzungsausdrucks wird das Ergebnis des Eingabeausdrucks mithilfe von Atomisierung in einen atomaren Wert oder eine leere Sequenz umgesetzt. Ergibt die Atomisierung eine Sequenz mit mehr als einem atomaren Wert, wird ein Fehler zurückgegeben. Werden keine Fehler zurückgegeben, versucht der Umsetzungsausdruck aufgrundlage des Eingabe-

wertes einen neuen Wert mit dem Zieltyp zu erstellen. Bestimmte Kombinationen aus Eingabe- und Zieltypen werden für die Umsetzung nicht unterstützt. Der Abschnitt „Typumsetzung" auf Seite 27 enthält Informationen dazu, welche Typen jeweils in welche anderen Typen umgesetzt werden können. Beim Umsetzen eines Wertes in einen Datentyp können Sie den Ausdruck 'castable' verwenden, um zu testen, ob der Wert tatsächlich in den betreffenden Datentyp umgesetzt werden kann.

Eine leere Sequenz ist nur dann als Eingabewert gültig, wenn nach dem Zieltyp ein Fragezeichen (?) folgt.

Wenn der Zieltyp in einem Ausdruck 'cast' den Wert 'xs:QName' oder einen von 'xs:QName' abgeleiteten Typ, oder 'xs:NOTATION' aufweist und der Eingabeausdruck den Typ 'xs:string' hat, aber keine Literalzeichenfolge ist, wird ein Fehler zurückgegeben.

### Syntax

*Ausdruck*
Ein beliebiger XQuery-Ausdruck, der einen einzelnen atomaren Wert oder eine leere Sequenz zurückgibt. Eine leere Sequenz ist zulässig, sofern auf den *Zieltyp* ein Fragezeichen (?) folgt.

*Zieltyp*
Der Typ, in den der Wert von *Ausdruck* umgesetzt wird. Bei *Zieltyp* muss es sich um einen atomaren Typ handeln, der sich unter den vordefinierten atomaren Typen des XML-Schemas befindet. Die Datentypen 'xs:NOTATION', 'xdt:anyAtomicType' und 'xs:anySimpleType' sind für *Zieltyp* nicht gültig.

**?** Dieses Zeichen gibt an, dass das Ergebnis von *Ausdruck* eine leere Sequenz sein kann.

### Beispiel

In folgendem Beispiel wird der Wert des Elements `price`, das den Datentyp 'xs:string' aufweist, mithilfe eines Umsetzungsausdrucks in den Typ 'xs:decimal' umgesetzt:

```
for $price in db2-fn:xmlcolumn('PRODUCT.DESCRIPTION')/product/description/price
return $price cast as xs:decimal
```

Bei Ausführung der Beispielabfrage für die Tabelle PRODUCT.DESCRIPTION der Beispieldatenbank SAMPLE wird das folgende Ergebnis zurückgegeben:

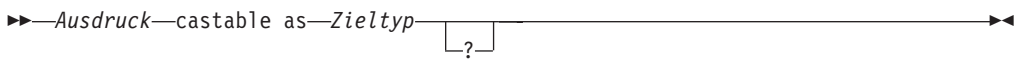
```
9.99
19.99
49.99
3.99
```

## Ausdruck 'castable'

Anhand von Ausdrücken vom Typ 'castable' wird getestet, ob ein Wert in einen bestimmten Datentyp umgesetzt werden kann. Ist es möglich, den Wert in den gewünschten Datentyp umzusetzen, gibt der Ausdruck 'castable' den Wert 'true' (wahr) zurück. Ansonsten gibt der Ausdruck den Wert 'false' (falsch) zurück.

Ausdrücke vom Typ 'castable' können als Vergleichselemente verwendet werden, um Umsetzungsfehler bei der Auswertung zu vermeiden. Sie können auch dazu verwendet werden, einen geeigneten Typ für die Verarbeitung eines Wertes auszuwählen. Der Abschnitt „Typumsetzung" auf Seite 27 enthält Informationen dazu, welche Typen jeweils in welche anderen Typen umgesetzt werden können.

## Syntax




*Ausdruck*
Ein XQuery-Ausdruck, der einen einzelnen atomaren Wert oder eine leere Sequenz zurückgibt.

*Zieltyp*
Der Typ, anhand dessen getestet wird, ob der Wert von *Ausdruck* entsprechend umgesetzt werden kann. Bei *Zieltyp* muss es sich um einen atomaren Typ handeln, der sich unter den vordefinierten XML-Schematypen befindet. Die Datentypen 'xs:NOTATION', 'xdt:anyAtomicType' und 'xs:anySimpleType' sind für *Zieltyp* nicht gültig.

**?** Gibt an, dass eine leere Sequenz als gültige Instanz des Zieltyps gilt. Wenn *Ausdruck* bei der Auswertung eine leere Sequenz ergibt und ? nicht angegeben ist, gibt der Ausdruck 'castable' den Wert 'false' (falsch) zurück.

## Zurückgegebener Wert

Ist es möglich, den Wert von *Ausdruck* in den *Zieltyp* umzusetzen, gibt der Ausdruck 'castable' den Wert 'true' (wahr) zurück. Ansonsten gibt der Ausdruck den Wert 'false' (falsch) zurück.

Ist das Ergebnis von *Ausdruck* eine leere Sequenz und folgt auf *Zieltyp* das Fragezeichen, gibt der Ausdruck 'castable' den Wert 'true' (wahr) zurück. Im folgenden Beispiel folgt das Fragezeichen auf den Zieltyp 'xs:integer':

```
$prod/revision castable as xs:integer?
```

Wenn *Zieltyp* in einem Ausdruck 'castable' den Wert 'xs:QName' oder einen von 'xs:QName' abgeleiteten Typ oder 'xs:NOTATION' aufweist und *Ausdruck* den Typ 'xs:string' hat, aber keine Literalzeichenfolge ist, gibt der Ausdruck 'castable' den Wert 'false' (falsch) zurück.

Ist das Ergebnis von *Ausdruck* eine Sequenz aus mehr als einem atomaren Wert, wird ein Fehler zurückgegeben.

## Beispiele

In nachstehendem Beispiel wird der Ausdruck 'castable' als Vergleichselement verwendet, um Fehler bei der Auswertung zu vermeiden. In diesem Beispiel wird ein dynamischer Fehler vermieden, falls `@OrderDate` kein gültiges Datum aufweist:

```
$po/orderID[if ( $po/@OrderDate castable as xs:date)
   then xs:date($po/@OrderDate) gt xs:date("2000-01-01")
   else false()]
```

Das Vergleichselement ist nur dann wahr und gibt nur dann einen Wert für `orderID` zurück, wenn das Datumsattribut 'date' ein gültiges Datum nach dem 1. Januar 2000 enthält. Ansonsten ist das Vergleichselement falsch und gibt eine leere Sequenz zurück.

In nachstehendem Beispiel wird der Ausdruck 'castable' verwendet, um einen geeigneten Typ für die Verarbeitung eines bestimmten Wertes auszuwählen. In diesem Beispiel wird 'castable' verwendet, um eine Postleitzahl (postalcode) entweder als ganze Zahl (integer) oder als Zeichenfolge (string) umzusetzen:

```
if ($postalcode castable as xs:integer)
   then $postalcode cast as xs:integer
   else $postalcode cast as xs:string
```

Im nachstehenden Beispiel wird der Ausdruck 'castable' in der FLWOR-Klausel **LET** verwendet, um den Wert von `$prod/mfgdate` zu testen und einen Wert an `$currdate` zu binden. Der Ausdruck 'castable' und der Ausdruck 'cast' unterstützen die Verarbeitung von leeren Sequenzen bei Verwendung des Fragezeichens als Bezugswert:

```
let $currdate := if ($prod/mfgdate castable as xs:date?)
   then $prod/mfgdate cast as xs:date?
   else "1000-01-01" cast as xs:date
```

Wenn der Wert von `$prod/mfgdate` in 'xs:date' umgesetzt werden kann, wird er in diesen Datentyp umgesetzt und an `$currdate` gebunden. Ist `$prod/mfgdate` eine leere Sequenz, wird eine leere Sequenz an `$currdate` gebunden. Kann `$prod/mfgdate` nicht in 'xs:date' umgesetzt werden, wird der Wert '1000-01-01' vom Typ 'xs:date' an `$currdate` gebunden.

In nachstehendem Beispiel wird der Ausdruck 'castable' verwendet, um den Wert der Produktkategorie (prod/category) vor der Durchführung eines Vergleichs zu testen. In der XML-Spalte FEATURES.INFO enthalten die Dokumente das Element `/prod/category`. Der Wert ist entweder ein numerischer Code oder ein Zeichenfolgecode. Der Ausdruck 'castable' im Vergleichselement XMLEXISTS testet den Wert von `/prod/category`, bevor ein Vergleich durchgeführt wird, um Fehler bei der Auswertung zu vermeiden:

```
SELECT F.PRODID FROM F FEATURES
WHERE xmlexists('$test/prod/category[ (( . castable as xs:double) and . > 100 ) or
   (( . castable as xs:string) and . > "A100" )]'
   passing F.INFO as "test")
```

Die zurückgegebenen Werte sind Produkt-IDs, bei denen die Kategoriecodes entweder größer als 100 oder größer als die Zeichenfolge "A100" sind.

## Umsetzungsausdruck und Aktualisierungsausdrücke

Um vorhandene XML-Daten mithilfe von DB2 XQuery zu aktualisieren, werden Aktualisierungsausdrücke innerhalb der Klausel **MODIFY** eines Umsetzungsausdrucks verwendet.

### Aktualisierungsausdrücke in einem Umsetzungsausdruck verwenden

In DB2 XQuery müssen Aktualisierungsausdrücke innerhalb der Klausel **MODIFY** eines Umsetzungsausdrucks verwendet werden. Die Aktualisierungsausdrücke wirken sich auf die kopierten Knoten aus, die von der Klausel **COPY** des Umsetzungsausdrucks erstellt werden.

Bei den folgenden Ausdrücken handelt es sich um Aktualisierungsausdrücke:

- Löschausdrücke
- Einfügeausdrücke
- Umbenennungsausdrücke
- Ersetzungsausdrücke
- FLWOR-Ausdrücke, die einen Aktualisierungsausdruck in ihrer Klausel **RETURN** enthalten
- Bedingungsausdrücke, die einen Aktualisierungsausdruck in ihrer Klausel **THEN** oder **ELSE** enthalten
- Zwei oder mehr durch Kommata getrennte Aktualisierungsausdrücke, bei denen alle Operanden entweder Aktualisierungsausdrücke oder eine leere Sequenz sind

Bei ungültigen Aktualisierungsausdrücke gibt DB2 XQuery einen Fehler zurück. Beispiel: DB2 XQuery gibt einen Fehler zurück, wenn eine Verzweigung eines Bedingungsausdrucks einen Aktualisierungsausdruck enthält und die andere Verzweigung einen Nichtaktualisierungsausdruck enthält, der keine leere Sequenz ist.

Ein Umsetzungsausdruck ist kein Aktualisierungsausdruck, da er keine vorhandenen Knoten ändert. Ein Umsetzungsausdruck erstellt geänderte Kopien von vorhandenen Knoten. Das Ergebnis eines Umsetzungsausdrucks kann Knoten umfassen, die von Aktualisierungsausdrücken in der Klausel **MODIFY** des Umsetzungsausdrucks erstellt wurden, sowie Kopien bereits vorhandener Knoten.

## XQuery-Aktualisierungsoperationen verarbeiten

In einem Umsetzungsausdruck kann die Klausel **MODIFY** mehrere Aktualisierungen angeben. Die Klausel **MODIFY** kann beispielsweise zwei Aktualisierungsausdrücke enthalten - einen zum Ersetzen eines vorhandenen Wertes und einen anderen zum Einfügen eines neuen Elements. Wenn die Klausel **MODIFY** mehrere Aktualisierungsausdrücke enthält, wird jeder Aktualisierungsausdruck unabhängig von den anderen ausgewertet und resultiert in einer Liste von Änderungsoperationen mit bestimmten Knoten, die von der Klausel **COPY** des Umsetzungsausdrucks erstellt wurden.

In einer Klausel **MODIFY** können Aktualisierungsausdrücke keine neuen Knoten ändern, die von anderen Aktualisierungsausdrücken hinzugefügt werden. Wenn ein Aktualisierungsausdruck beispielsweise einen neuen Elementknoten hinzufügt, kann ein anderer Aktualisierungsausdruck den Knotennamen des neu erstellten Knotens nicht ändern.

Alle in der Klausel **MODIFY** des Umsetzungsausdrucks angegebenen Änderungsoperationen werden erfasst und effektiv in der folgenden Reihenfolge angewandt:

1. Die folgenden Aktualisierungsoperationen werden in einer nicht deterministischen Reihenfolge ausgeführt:
   - Einfügeoperationen ohne Schlüsselwörter für die Sortierung wie beispielsweise **before**, **after**, **as first** oder **as last**.
   - Alle Umbenennungsoperationen.
   - Ersetzungsoperationen, bei denen die Schlüsselwörter **value of** angegeben sind und bei denen der Zielknoten ein Attribut-, Text-, Kommentar- oder Verarbeitungsanweisungsknoten ist.
2. Einfügeoperationen mit Schlüsselwörtern für die Sortierung wie beispielsweise **before**, **after**, **as first** oder **as last**.

3. Ersetzungsoperationen, bei denen die Schlüsselwörter **value of** nicht angegeben sind.
4. Ersetzungsoperationen, bei denen die Schlüsselwörter **value of** angegeben sind und bei denen der Zielknoten ein Elementknoten ist.
5. Alle Löschoperationen.

Die Reihenfolge, in der Änderungsoperationen angewandt werden, stellt sicher, dass eine Reihe mehrerer Änderungen zu einem deterministischen Ergebnis führt. Der letzte XQuery-Ausdruck unter „Beispiele" zeigt anhand eines Beispiels, wie die Reihenfolge der Aktualisierungsoperationen dafür sorgt, dass eine Reihe von mehreren Änderungen ein deterministisches Ergebnis hat.

## Ungültige XQuery-Aktualisierungsoperationen

Bei der Verarbeitung eines Umsetzungsausdrucks gibt DB2 XQuery einen Fehler zurück, wenn eine der folgenden Bedingungen eintritt:

- Zwei oder mehr Umbenennungsoperationen werden auf denselben Knoten angewandt.
- Zwei oder mehr Ersetzungsoperationen, die die Schlüsselwörter **value of** verwenden, werden auf denselben Knoten angewandt.
- Zwei oder mehr Ersetzungsoperationen, die die Schlüsselwörter **value of** nicht verwenden, werden auf denselben Knoten angewandt.
- Das Ergebnis des Umsetzungsausdrucks ist keine gültige XMD-Instanz.

  Eine XDM-Instanz, die ein Element mit zwei Attributen desselben Namens enthält, ist beispielsweise ungültig.
- Die XDM-Instanz enthält inkonsistente Namensbereichsbindungen.

  Die folgenden Beispiele stehen für inkonsistente Namensbereichsbindungen:
  - Eine Namensbereichsbindung im QName eines Attributknotens ist mit den Namensbereichsbindungen ihres übergeordneten Elementknotens nicht kompatibel.
  - Die Namensbereichsbindungen in zwei Attributknoten mit demselben übergeordneten Element sind miteinander nicht kompatibel.

## Beispiele

Im folgenden Beispiel bindet die Klausel **COPY** eines Umsetzungsausdrucks die Variable `$product` an eine Kopie eines Elementknotens, und die Klausel **MODIFY** des Umsetzungsausdrucks verwendet zwei Aktualisierungsausdrücke zum Ändern des kopierten Knotens:

```
  xquery
declare default element namespace "http://posample.org";
transform
copy $product := db2-fn:sqlquery(
  "select description from product where pid='100-100-01'")/product
modify(
  do replace value of $product/description/price with 349.95,
  do insert <status>Available</status> as last into $product )
return $product
```

Das nachstehende Beispiel verwendet einen XQuery-Umsetzungsausdruck innerhalb einer SQL-Anweisung UPDATE zum Ändern von XML-Daten in der Tabelle CUSTOMER. Die SQL-Anweisung UPDATE wird für eine Zeile in der Tabelle CUSTOMER ausgeführt. Der Umsetzungsausdruck erstellt eine Kopie des XML-Dokuments anhand der Spalte INFO der Zeile und fügt der Kopie des Dokuments

ein Element status hinzu. Die Anweisung UPDATE ersetzt das Dokument in der Spalte INFO der Zeile durch die Kopie des durch den Umsetzungsausdruck geänderten Dokuments:

```
UPDATE customer
SET info = xmlquery( 'declare default element namespace "http://posample.org";
   transform
   copy $newinfo := $info
   modify do insert <status>Current</status> as last into $newinfo/customerinfo
   return $newinfo' passing info as "info")
WHERE cid = 1003
```

In nachstehendem Beispiel wird die Tabelle CUSTOMER aus der DB2-Beispieldatenbank SAMPLE verwendet. In der Tabelle CUSTOMER enthält die XML-Spalte INFO die Adresse und Telefonnummer von Kunden.

In nachstehendem Beispiel wird die SQL-Anweisung SELECT für eine Zeile in der Tabelle CUSTOMER ausgeführt. Die Klausel **COPY** des Umsetzungsausdrucks erstellt eine Kopie des XML-Dokuments anhand der Spalte INFO. Der Löschausdruck löscht Adressinformationen und Telefonnummern (phone) aus der Kopie des Dokuments. Vom Löschen ausgenommen sind die Telefonnummern vom Typ 'work' (Geschäftsnummer). Die Klausel **RETURN** verwendet das Attribut für die Kunden-ID (cid) und das Attribut für das Land (country) aus dem ursprünglichen Dokument der Tabelle CUSTOMER:

```
SELECT XMLQUERY ('declare default element namespace "http://posample.org";
  transform
  copy $mycust := $d
  modify
    do delete ( $mycust/customerinfo/addr,
       $mycust/customerinfo/phone[@type != "work"] )
  return
  <custinfo>
    <Cid>{data($d/customerinfo/@Cid)}</Cid>
    {$mycust/customerinfo/*}
    <country>{data($d/customerinfo/addr/@country)}</country>
  </custinfo>'
  passing INFO as "d")
FROM CUSTOMER
WHERE CID = 1003
```

Bei Ausführung für die Datenbank SAMPLE gibt die Anweisung das folgende Ergebnis zurück:

```
<custinfo xmlns="http://posample.org">
  <Cid>1003</Cid>
    <name>Robert Shoemaker</name>
    <phone type="work">905-555-7258</phone>
  <country>Canada</country>
</custinfo>
```

In nachstehendem Beispiel zeigt der XQuery-Ausdruck, wie die Reihenfolge der Aktualisierungsoperation dafür sorgt, dass eine Reihe von mehreren Änderungen ein deterministisches Ergebnis hat. Der Einfügeausdruck fügt ein Element vom Typ status nach dem Element phone ein, und der Ersetzungsausdruck ersetzt das Element phone durch das Element email:

```
  xquery
declare default element namespace "http://posample.org";
let $email  := <email>jnoodle@my-email.com</email>
let $status := <status>current</status>
return
  transform
  copy $mycust := db2-fn:sqlquery('select info from customer where cid = 1002')
```

```
  modify (
    do replace $mycust/customerinfo/phone with $email,
    do insert $status after $mycust/customerinfo/phone[@type = "work"] )
  return $mycust
```

In der Klausel **MODIFY** steht der Ersetzungsausdruck vor dem Einfügeausdruck. Beim Aktualisieren der kopierten Knotensequenz `$mycust` hingegen wird die Einfügeoperation vor der Ersetzungsoperation ausgeführt, um ein deterministisches Ergebnis zu gewährleisten. Bei Ausführung für die Datenbank SAMPLE gibt der Ausdruck das folgende Ergebnis zurück:

```
<customerinfo xmlns="http://posample.org" Cid="1002">
                                        <name>Jim Noodle</name>
    <addr country="Canada">
                                             <street>25 EastCreek</street>
        <city>Markham</city>
                                             <prov-state>Ontario</prov-state>
        <pcode-zip>N9C 3T6</pcode-zip>
                                        </addr>
  <email>jnoodle@my-email.com</email>
  <status>current</status>
</customerinfo>
```

Wenn die Ersetzungsoperation zuerst ausgeführt würde, befände sich das Element `phone` nicht in der Knotensequenz, und die Operation zum Einfügen des Elements `status` nach dem Element `phone` wäre sinnlos.

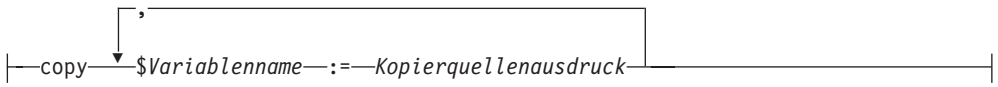
Der Abschnitt „XQuery-Aktualisierungsoperationen verarbeiten” auf Seite 125 enthält Informationen zur Reihenfolge von Aktualisierungsoperationen.

## Umsetzungsausdruck

Ein Umsetzungsausdruck erstellt Kopien von mindestens einem Knoten. Aktualisierungsausdrücke in der Klausel **MODIFY** des Umsetzungsausdrucks ändern die kopierten Knoten. Der Ausdruck in der Klausel **RETURN** gibt das Ergebnis des Umsetzungsausdrucks an.

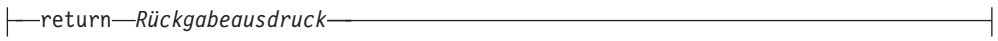
### Syntax

**Klausel COPY:**




**Klausel MODIFY:**

**Klausel RETURN:**



## Parameter

**transform**
Optionales Schlüsselwort, das zum Starten eines Umsetzungsausdrucks verwendet werden kann.

**copy**
Schlüsselwort, das die Klausel **COPY** eines Umsetzungsausdrucks einleiten kann. Jeder *Variablenname* in der Klausel **COPY** ist an eine Kopie der Knotenbaumstruktur gebunden, die vom entsprechenden *Kopierquellenausdruck* zurückgegeben wird.

*Variablenname*
Der Name der Variablen, die an eine Kopie der Knotenbaumstruktur im Ergebnis von *Kopierquellenausdruck* gebunden werden soll.

*Kopierquellenausdruck*
Ein XQuery-Ausdruck, der kein Aktualisierungsausdruck ist. Der Ausdruck muss einen einzelnen Knoten zusammen mit dessen untergeordneten Elementen (sofern vorhanden ) zurückgeben, was als *Knotenbaumstruktur* bezeichnet wird.

Wenn der Ausdruck einen Kommaoperator auf oberster Ebene enthält, muss der Ausdruck in runde Klammern eingeschlossen werden. *Kopierquellenausdruck* wird so ausgewertet wie ein in einem Elementkonstruktor eingeschlossener Ausdruck.
Die von der Klausel **COPY** erstellten Knoten haben neue Knotenidentitäten und sind nicht typisiert. Wenn beispielsweise eine Kopie eines Elementknotens vom Typ 'xs:decimal' erstellt wird, hat der kopierte Knoten eine eigene Knotenidentität und den Typ 'xs:anyType'.

**modify**
Schlüsselwort, das die Klausel **MODIFY** eines Umsetzungsausdrucks einleiten kann.

*Änderungsausdruck*
Ein Aktualisierungsausdruck oder eine leere Sequenz. Wenn der Ausdruck einen Kommaoperator auf oberster Ebene enthält, muss der Ausdruck in runde Klammern eingeschlossen werden. Der Aktualisierungsausdruck wird ausgewertet, und die resultierenden Aktualisierungen werden auf die von der Klausel **COPY** erstellten Knoten angewandt.

DB2 XQuery gibt einen Fehler zurück, wenn der Zielknoten eines Aktualisierungsausdrucks kein Knoten ist, der von der Klausel **COPY** im übergeordneten Umsetzungsausdruck erstellt wurde. So gibt DB2 XQuery beispielsweise einen Fehler zurück, wenn ein Umbenennungsausdruck versucht, einen Knoten umzubenennen, der nicht von der Klausel **COPY** erstellt wurde.

Die in einer Klausel vom Typ **MODIFY** angegebenen Aktualisierungen können als Ergebnis einen Knoten haben, dessen untergeordnete Elemente mehrere benachbarte Textknoten aufweisen. Verfügt ein Knoten unter seinen untergeordneten Elementen über mehrere benachbarte Textknoten, werden diese zu einem einzigen Textknoten zusammengeführt. Der Zeichenfolgewert des resultierenden Textknotens besteht aus den verknüpften Zeichenfolgewerten der benachbarten Textknoten ohne Zwischenleerzeichen. Wird ein untergeordneter Knoten erstellt, bei dem es sich um einen Textknoten mit einen Zeichenfolgewert mit Nulllänge handelt, wird der Textknoten gelöscht.

**return**
Schlüsselwort, das die Klausel **RETURN** eines Umsetzungsausdrucks einleiten kann.

*Rückgabeausdruck*
Ein XQuery-Ausdruck, der kein Aktualisierungsausdruck ist. Wenn der Ausdruck einen Kommaoperator auf oberster Ebene enthält, muss der Ausdruck in runde Klammern eingeschlossen werden.

Der Ausdruck in der Klausel **RETURN** wird ausgewertet und als Ergebnis des Umsetzungsausdrucks zurückgegeben. Ausdrücke in der Klausel **RETURN** können auf die Knoten zugreifen, die von Aktualisierungsausdrücken in der Klausel **MODIFY** geändert oder erstellt wurden.

Die Klausel **RETURN** eines Umsetzungsausdrucks ist nicht auf die Rückgabe von Knoten beschränkt, die von der Klausel **COPY** erstellt wurden. Der *Rückgabeausdruck* kann eine beliebige Kombination aus kopierten Knoten, ursprünglichen Knoten und erstellten Knoten zurückgeben.

Wenn das Ergebnis von *Rückgabeausdruck* ein XML-Dokument ist, das nicht typisierte Knoten enthält (wie beispielsweise neue Knoten oder kopierte Knoten), können Sie die SQL/XML-Funktion XMLVALIDATE verwenden, um den Knoten im XML-Dokument bestimmte Typinformationen zuzuordnen.

## Beispiele

In nachstehendem Beispiel wird die Tabelle CUSTOMER aus der DB2-Beispieldatenbank SAMPLE verwendet. In der Tabelle CUSTOMER enthält die XML-Spalte INFO die Adresse und Telefonnummer von Kunden.

In nachstehendem Beispiel erstellt die Klausel **COPY** des Umsetzungsausdrucks eine Kopie des XML-Dokuments aus der Spalte INFO. In der Klausel **MODIFY** löscht der Löschausdruck aus dem XML-Dokument alle Telefonnummern, bei denen das Typattribut des Elements 'phone' nicht 'home' (Privatnummer) ist:

```
  xquery
declare default element namespace 'http://posample.org';
transform
   copy $mycust := db2-fn:sqlquery('select INFO from CUSTOMER where Cid = 1003')
   modify
     do delete $mycust/customerinfo/phone[@type!="home"]
   return $mycust;
```

Bei Ausführung für die Datenbank SAMPLE gibt der Ausdruck das folgende Ergebnis zurück:

```
<customerinfo xmlns="http://posample.org" Cid="1003">
    <name>Robert Shoemaker</name>
    <addr country="Canada">
        <street>1596 Baseline</street>
        <city>Aurora</city>
                                          <prov-state>Ontario</prov-state>
        <pcode-zip>N8X 7F8</pcode-zip>
                                       </addr>
    <phone type="home">416-555-2937</phone>
</customerinfo>
```

Im folgenden Ausdruck wird das optionale Schlüsselwort **transform** nicht verwendet. Der Umsetzungsausdruck beginnt mit der Klausel **COPY** und ist äquivalent zum vorherigen Ausdruck.

```
  xquery
declare default element namespace 'http://posample.org';
copy $mycust := db2-fn:sqlquery('select INFO from CUSTOMER where Cid = 1003')
modify
   do delete $mycust/customerinfo/phone[@type!="home"]
return $mycust;
```

In nachstehendem Beispiel modifiziert und prüft die SQL-Anweisung UPDATE das XML-Dokument anhand einer Zeile der Tabelle CUSTOMER. Die Klausel **COPY** des Umsetzungsausdrucks erstellt eine Kopie des XML-Dokuments anhand der Spalte INFO. Der Ersetzungsausdruck ändert den Wert des Elements `name` in der Kopie des XML-Dokuments. Die Kopie des Dokuments wird nicht geprüft, und die Knoten der Kopie des Dokuments weisen keine Typinformationen auf. Die Funktion XMLVALIDATE prüft die Kopie des Dokuments und fügt den Knoten die Typinformationen hinzu:

```
UPDATE customer set info = XMLVALIDATE(
  XMLQUERY(' declare default element namespace "http://posample.org";
  transform
  copy $mycust := $cust
  modify
    do replace value of $mycust/customerinfo/name with "Larry Menard, Jr."
  return $mycust'
  passing info as "cust" )
ACCORDING TO XMLSCHEMA ID customer)
where cid = 1005
```

## Grundlegende Aktualisierungsausdrücke

Mithilfe der vier grundlegenden XQuery-Aktualisierungsausdrücke können Sie komplexe Aktualisierungsausdrücke zum Aktualisieren bestehender XML-Daten erstellen. In DB2 XQuery werden Aktualisierungsausdrücke innerhalb der Klausel **MODIFY** eines Umsetzungsausdrucks verwendet.

### Löschausdruck

Mit einem Löschausdruck werden null oder mehr Knoten aus einer Knotensequenz gelöscht.

#### Syntax

**do delete**
Die Schlüsselwörter, die einen Löschausdruck einleiten.

*Zielausdruck*
Ein XQuery-Ausdruck, der kein Aktualisierungsausdruck ist. Wenn der Ausdruck einen Kommaoperator auf oberster Ebene enthält, muss der Ausdruck in runde Klammern eingeschlossen werden. Das Ergebnis von *Zielausdruck* muss eine Sequenz aus null oder mehr Knoten sein. Das Merkmal des übergeordneten Elements (Elter) darf bei keinem Knoten leer sein.

Der Umsetzungsausdruck wertet den Löschausdruck aus und generiert eine Liste mit Aktualisierungen, die aus den zu löschenden Knoten besteht. Jeder Knoten, der mit *Zielausdruck* übereinstimmt, wird für Löschen markiert. Beim Löschen der mit *Zielausdruck* übereinstimmenden Knoten wird deren jeweilige Zuordnung zum entsprechenden übergeordneten Knoten aufgehoben. Die Knoten und die untergeordneten Elemente (Kinder) der Knoten sind nicht mehr Teil der Knotensequenz.

#### Beispiele

In nachstehendem Beispiel wird die Tabelle CUSTOMER aus der DB2-Beispieldatenbank SAMPLE verwendet. In der Tabelle CUSTOMER enthält die XML-Spalte INFO die Adresse und Telefonnummer von Kunden.

Der folgende Ausdruck löscht das Adresselement (addr) und alle seine untergeordneten Knoten sowie alle Telefonnummern aus dem XML-Dokument, in dem der Attributtyp für die Telefonnummer (phone) nicht 'home' (für die Privatnummer) ist:

```
  xquery
declare default element namespace 'http://posample.org';
transform
copy $mycust := db2-fn:sqlquery('select INFO from CUSTOMER where Cid =1003')
modify
  do delete ($mycust/customerinfo/addr, $mycust/customerinfo/phone[@type!="home"])
return $mycust
```

Bei Ausführung für die Datenbank SAMPLE gibt der Ausdruck das folgende Ergebnis zurück:

```
<customerinfo xmlns="http://posample.org" Cid="1003">
    <name>Robert Shoemaker</name>
    <phone type="home">416-555-2937</phone>
</customerinfo>
```

In folgendem Beispiel wird das Typattribut (type) aus jedem Elementknoten 'phone' gelöscht, wenn dessen Attributwert 'home' lautet:

```
  xquery
declare default element namespace "http://posample.org";
transform
copy $mycust := db2-fn:sqlquery('select info from customer where cid = 1004')
modify (
  for $phone in $mycust/customerinfo//phone[@type="home"]
  return
    do delete $phone/@type )
return $mycust
```

Bei Ausführung für die Datenbank SAMPLE gibt der Ausdruck das folgende Ergebnis zurück:

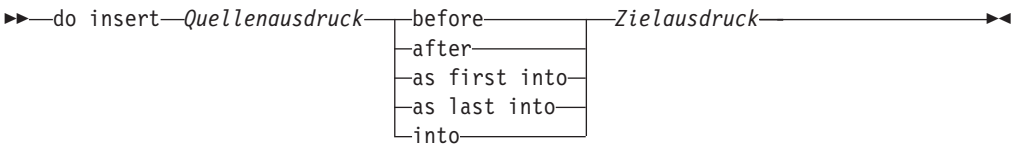
```
<customerinfo xmlns="http://posample.org" Cid="1004">
  <name>Matt Foreman</name>
    <addr country="Canada">
        <street>1596 Baseline</street>
        <city>Toronto</city>
                                          <prov-state>Ontario</prov-state>
    <pcode-zip>M3Z 5H9</pcode-zip>
                                      </addr>
  <phone type="work">905-555-4789</phone>
  <phone>416-555-3376</phone>
  <assistant><name>Gopher Runner</name>
    <phone>416-555-3426</phone>
  </assistant>
</customerinfo>
```

Der Ausdruck löscht das Typattribut sowohl aus der Telefonnummer des Kunden (customer) als auch des Assistenten (assistant).

## Einfügeausdruck

Ein Einfügeausdruck fügt Kopien von mindestens einem Knoten an einer angegebenen Position in eine Knotensequenz ein.

### Syntax




**do insert**
Die Schlüsselwörter, die einen Einfügeausdruck einleiten.

*Quellenausdruck*
Ein XQuery-Ausdruck, der kein Aktualisierungsausdruck ist. Wenn der Ausdruck einen Kommaoperator auf oberster Ebene enthält, muss der Ausdruck in runde Klammern eingeschlossen werden. Der Ausdruck wird so ausgewertet wie ein eingeschlossener Ausdruck in einem Elementkonstruktor. Das Ergebnis von *Quellenausdruck* ist eine Sequenz aus null oder mehr einzufügenden Knoten. Diese Sequenz wird als *Einfügesequenz* bezeichnet. Wenn die Einfügesequenz einen Dokumentknoten enthält, wird dieser Dokumentknoten in der Einfügesequenz durch seine untergeordneten Elemente ersetzt.

Wenn die Einfügesequenz Attributknoten enthält, die am Anfang der Sequenz stehen, werden die Attribute - je nach angegebenem Schlüsselwort - entweder dem Knoten *Zielausdruck* oder dessen übergeordnetem Element hinzugefügt. Wenn die Einfügesequenz einen Attributknoten enthält, der auf einen Knoten folgt, der kein Attributknoten ist, gibt DB2 XQuery einen Fehler zurück.

**before**
Dieses Schlüsselwort gibt an, dass die Knoten von *Quellenausdruck* zu den führenden gleichgeordneten Elementen des Knotens von *Zielausdruck* werden.

Die Knoten von *Quellenausdruck* werden unmittelbar vor dem Knoten von *Zielausdruck* eingefügt. Werden mehrere Knoten vor *Zielausdruck* eingefügt, werden sie in nicht deterministischer Reihenfolge eingefügt. Die Gruppe der eingefügten Knoten wird jedoch unmittelbar vor *Zielausdruck* angezeigt. Wenn die Einfügesequenz Attributknoten enthält, die am Anfang der Sequenz stehen, werden die Attribute zu Attributen des übergeordneten Elements des Zielknotens.

**after**
Dieses Schlüsselwort gibt an, dass die Knoten von *Quellenausdruck* zu den nachfolgenden gleichgeordneten Elementen des Knotens von *Zielausdruck* werden.

Die Knoten von *Quellenausdruck* werden unmittelbar nach dem Knoten von *Zielausdruck* eingefügt. Werden mehrere Knoten nach *Zielausdruck* eingefügt, werden sie in nicht deterministischer Reihenfolge eingefügt. Die Gruppe der eingefügten Knoten wird jedoch unmittelbar nach *Zielausdruck* angezeigt. Wenn die Einfügesequenz Attributknoten enthält, die am Anfang der Sequenz stehen, werden die Attribute zu Attributen des übergeordneten Elements des Zielknotens.

**as first into**
Dieses Schlüsselwort gibt an, dass die Knoten von *Quellenausdruck* zu den ersten untergeordneten Elementen des Knotens von *Zielausdruck* werden.

Werden mehrere Knoten als erste untergeordnete Elemente des Knotens von *Zielausdruck* eingefügt, werden sie in nicht deterministischer Reihenfolge eingefügt. Die Gruppe der eingefügten Knoten wird jedoch als die ersten untergeordneten Kinder von *Zielausdruck* angezeigt. Wenn die Einfügesequenz Attributknoten enthält, die am Anfang der Sequenz stehen, werden die Attribute zu Attributen des Zielknotens.

**as last into**
Dieses Schlüsselwort gibt an, dass die Knoten von *Quellenausdruck* zu den letzten untergeordneten Elementen des Knotens von *Zielausdruck* werden.

Werden mehrere Knoten als letzte untergeordnete Elemente des Knotens von *Zielausdruck* eingefügt, werden sie in nicht deterministischer Reihenfolge eingefügt. Die Gruppe der eingefügten Knoten wird jedoch als die letzten untergeordneten Kinder von *Zielausdruck* angezeigt. Wenn die Einfügesequenz Attributknoten enthält, die am Anfang der Sequenz stehen, werden die Attribute zu Attributen des Zielknotens.

**into**
Dieses Schlüsselwort gibt an, dass die Knoten von *Quellenausdruck* zu untergeordneten Elementen des Knotens von *Zielausdruck* in nicht deterministischer Reihenfolge werden.

Die Knoten von *Quellenausdruck* werden als untergeordnete Elemente des Knotens von *Zielausdruck* an nicht deterministischen Positionen eingefügt. Wenn die Einfügesequenz Attributknoten enthält, die am Anfang der Sequenz stehen, werden die Attribute zu Attributen des Zielknotens.

*Zielausdruck*
Ein XQuery-Ausdruck, der kein Aktualisierungsausdruck ist. Wenn der Ausdruck einen Kommaoperator auf oberster Ebene enthält, muss der Ausdruck in runde Klammern eingeschlossen werden. In Abhängigkeit von den Schlüsselwörtern, die vor *Zielausdruck* angegeben werden, gelten die folgenden Regeln:

- Bei Angabe von **before** oder **after** muss das Ergebnis von *Zielausdruck* ein Element-, Text-, Verarbeitungsanweisungs- oder Kommentarknoten sein, dessen Elternmerkmal nicht leer ist. Ist das übergeordnete Element (Elter) von *Zielausdruck* ein Dokumentknoten und wird **before** oder **after** angegeben, darf die Einfügesequenz keinen Attributknoten enthalten.
- Bei Angabe von **into**, **as first into** oder **as last into** muss das Ergebnis von *Zielausdruck* ein einzelner Elementknoten oder ein einzelner Dokumentknoten sein.
- Wird **into** angegeben und ist *Zielausdruck* ein Dokumentknoten, darf die Einfügesequenz keinen Attributknoten enthalten.

## Beispiele

In nachstehendem Beispiel wird die Tabelle CUSTOMER aus der DB2-Beispieldatenbank SAMPLE verwendet. In der Tabelle CUSTOMER enthält die XML-Spalte INFO die Adresse und Telefonnummer von Kunden.

In nachstehendem Beispiel erstellt die Klausel **COPY** des Umsetzungsausdrucks eine Kopie des XML-Dokuments aus der Spalte INFO. Der Einfügeausdruck fügt das Element `billto` (Rechnungsadresse) und alle dessen untergeordnete Elemente nach dem letzten Element `phone` (Telefonnummer) ein:

```
  xquery
declare default element namespace 'http://posample.org';
   transform
   copy $mycust := db2-fn:sqlquery('select info from customer where cid = 1004')
   modify
     do insert
       <billto country="Canada">
          <street>4441 Wagner</street>
              <city>Aurora</city>
                                                  <prov-state>Ontario</prov-state>
```

```
            <pcode-zip>N8X 7F8</pcode-zip>
       </billto>
    after $mycust/customerinfo/phone[last()]
   return $mycust
```

Bei Ausführung für die Datenbank SAMPLE gibt der Ausdruck das folgende Ergebnis zurück:

```
<customerinfo xmlns="http://posample.org" Cid="1004">
  <name>Matt Foreman</name>
    <addr country="Canada">
        <street>1596 Baseline</street>
        <city>Toronto</city>
                                            <prov-state>Ontario</prov-state>
    <pcode-zip>M3Z 5H9</pcode-zip>
                                       </addr>
  <phone type="work">905-555-4789</phone>
  <phone type="home">416-555-3376</phone>
  <billto country="Canada">
    <street>4441 Wagner</street>
        <city>Aurora</city>
                                            <prov-state>Ontario</prov-state>
        <pcode-zip>N8X 7F8</pcode-zip>
  </billto>
  <assistant>
    <name>Gopher Runner</name>
    <phone type="home">416-555-3426</phone>
  </assistant>
</customerinfo>
```

In folgendem Beispiel wird das Attribut `extension` (Durchwahl) mit dem Wert `x2334` in das Element `phone` eingefügt, sofern der Attributtyp von 'phone' `work` (Geschäftsnummer) lautet:

```
  xquery
let $phoneext :=  attribute extension { "x2334" }
return
  transform
  copy $mycust := db2-fn:sqlquery('select info from customer where cid = 1004')
  modify
    do insert  $phoneext into $mycust/*:customerinfo/*:phone[@type="work"]
  return $mycust
```

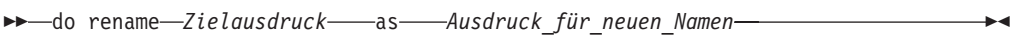
Bei Ausführung für die Datenbank SAMPLE gibt der Ausdruck das folgende Ergebnis zurück:

```
<customerinfo xmlns="http://posample.org" Cid="1004">
  <name>Matt Foreman</name>
      <addr country="Canada">
          <street>1596 Baseline</street>
          <city>Toronto</city>
                                              <prov-state>Ontario</prov-state>
      <pcode-zip>M3Z 5H9</pcode-zip>
                                         </addr>
  <phone extension="x2334" type="work">905-555-4789</phone>
  <phone type="home">416-555-3376</phone>
  <assistant>
    <name>Gopher Runner</name>
    <phone type="home">416-555-3426</phone>
  </assistant>
</customerinfo>
```

## Umbenennungsausdruck

Ein Umbenennungsausdruck ersetzt das Namensmerkmal eines Datenmodellknotens durch einen neuen qualifizierten Namen (QName).

## Syntax




**do rename**
Die Schlüsselwörter, die einen Umbenennungsausdruck einleiten.

*Zielausdruck*
Ein XQuery-Ausdruck, der kein Aktualisierungsausdruck ist. Das Ergebnis von *Zielausdruck* muss ein einzelner Element-, Attribut- oder Verarbeitungsanweisungsknoten sein. Wenn der Ausdruck einen Kommaoperator auf oberster Ebene enthält, muss der Ausdruck in runde Klammern eingeschlossen werden.

Der Umbenennungsausdruck wirkt sich nur auf den Knoten von *Zielausdruck* aus. Wenn der Knoten von *Zielausdruck* ein Elementknoten ist, wirkt sich der Ausdruck nicht auf die Attribute oder untergeordneten Elemente des Zielknotens aus. Wenn der Knoten von *Zielausdruck* ein Attributknoten ist, wirkt sich der Ausdruck nicht auf andere Attribute oder untergeordnete Elemente des übergeordneten Knotens des Knotens von *Zielausdruck* aus.

**as** Das Schlüsselwort, das den Ausdruck für den neuen Namen einleitet.

*Ausdruck_für_neuen_Namen*
Ein XQuery-Ausdruck, der kein Aktualisierungsausdruck ist. Das Ergebnis von *Ausdruck_für_neuen_Namen* muss ein Wert vom Typ 'xs:string', 'xs:QName' oder 'xs:untypedAtomic' sein oder ein Knoten, aus dem ein solcher Wert durch Atomisierung extrahiert werden kann. Wenn der Ausdruck einen Kommaoperator auf oberster Ebene enthält, muss der Ausdruck in runde Klammern eingeschlossen werden.

Der Ergebniswert wird in einen qualifizierten Namen (QName) umgewandelt, wobei das betreffende Namensbereichspräfix (sofern vorhanden) entsprechend den statisch bekannten Namensbereichen aufgelöst wird. Das Ergebnis ist entweder ein Fehler oder ein erweiterter qualifizierter Name. Der erweiterte qualifizierte Name ersetzt den Namen des Knotens von *Zielausdruck*.

Enthält der neue qualifizierte Name dasselbe Präfix, aber eine andere URI als ein gültiger Namensbereich des Knotens von *Zielausdruck*, gibt DB2 XQuery einen Fehler zurück.

## Beispiele

In nachstehendem Beispiel wird die Tabelle CUSTOMER aus der DB2-Beispieldatenbank SAMPLE verwendet. In der Tabelle CUSTOMER enthält die XML-Spalte INFO die Adresse und Telefonnummer von Kunden.

In nachstehendem Beispiel erstellt die Klausel **COPY** des Umsetzungsausdrucks eine Kopie des XML-Dokuments aus der Spalte INFO. Der Umbenennungsausdruck ändert das Namensmerkmal des Elements `addr` (Adresse) in `shipto` (Lieferadresse):

```
  xquery
declare default element namespace 'http://posample.org';
transform
copy $mycust := db2-fn:sqlquery('select info from customer where cid = 1000')
modify
  do rename $mycust/customerinfo/addr as "shipto"
return $mycust
```

Bei Ausführung für die Datenbank SAMPLE gibt der Ausdruck das folgende Ergebnis zurück:

```
<customerinfo xmlns="http://posample.org" Cid="1000">
    <name>Kathy Smith</name>
  <shipto country="Canada">
        <street>5 Rosewood</street>
        <city>Toronto</city>
                                          <prov-state>Ontario</prov-state>
        <pcode-zip>M6W 1E6</pcode-zip>
  </shipto>
    <phone type="work">416-555-1358</phone>
</customerinfo>
```

Im folgenden Beispiel enthält die Klausel **MODIFY** des Umsetzungsausdrucks einen FLWOR-Ausdruck und einen Umbenennungsausdruck, der das Namensmerkmal aller Instanzen des Elements `phone` in `phonenumber` ändert:

```
  xquery
declare default element namespace 'http://posample.org';
transform
copy $mycust :=  db2-fn:sqlquery('select info from customer where cid = 1003')
modify
  for $phone in $mycust/customerinfo/phone
  return
    do rename $phone as "phonenumber"
return $mycust
```

Bei Ausführung für die Datenbank SAMPLE gibt der Ausdruck das folgende Ergebnis zurück:

```
<customerinfo xmlns="http://posample.org" Cid="1003">
    <name>Robert Shoemaker</name>
    <addr country="Canada">
        <street>1596 Baseline</street>
        <city>Aurora</city>
                                          <prov-state>Ontario</prov-state>
        <pcode-zip>N8X 7F8</pcode-zip>
                                       </addr>
  <phonenumber type="work">905-555-7258</phonenumber>
  <phonenumber type="home">416-555-2937</phonenumber>
  <phonenumber type="cell">905-555-8743</phonenumber>
  <phonenumber type="cottage">613-555-3278</phonenumber>
</customerinfo>
```

Im folgenden Beispiel ändert der Umbenennungsausdruck den Namen des Attributs des Elements `addr` von `country` (Land) in `geography` (Region):

```
  xquery
declare default element namespace 'http://posample.org';
transform
copy $mycust := db2-fn:sqlquery('select info from customer where cid = 1000')
modify
  do rename $mycust/customerinfo/addr/@country as "geography"
return $mycust
```

Bei Ausführung für die Datenbank SAMPLE gibt der Ausdruck das folgende Ergebnis zurück:

```
<customerinfo xmlns="http://posample.org" Cid="1000">
    <name>Kathy Smith</name>
  <addr geography="Canada">
        <street>5 Rosewood</street>
        <city>Toronto</city>
                                          <prov-state>Ontario</prov-state>
```

```
        <pcode-zip>M6W 1E6</pcode-zip>
                                        </addr>
    <phone type="work">416-555-1358</phone>
</customerinfo>
```

In nachstehendem Beispiel werden der Umbenennungsausdruck und die Funktion 'fn:QName' verwendet, um das Namensbereichspräfix `other` den Namen der Elemente und Attribute von 'phone' (außer vom Typ 'work') hinzuzufügen. Das Präfix `other` ist an die URI `http://otherphone.com` gebunden:

```
  xquery
declare default element namespace 'http://posample.org';
transform
copy $mycust := db2-fn:sqlquery('select info from customer where cid = 1004')
modify
  for $elem in $mycust/customerinfo/phone[@type != "work"]
  let $elemLocalName := fn:local-name($elem)
  let $newElemQName := fn:QName("http://otherphone.com", fn:concat("other:",
     $elemLocalName))
  return
    ( do rename $elem as $newElemQName,
    for $a in $elem/@* let $attrlocalname := fn:local-name($a)
    let $newAttrName := fn:QName("http://otherphone.com", fn:concat("other:",
       $attrlocalname))
    return
      do rename $a as $newAttrName )
return $mycust
```

Bei Ausführung für die Datenbank SAMPLE gibt der Ausdruck das folgende Ergebnis zurück:

```
<customerinfo xmlns="http://posample.org" Cid="1004">
  <name>Matt Foreman</name>
    <addr country="Canada">
        <street>1596 Baseline</street>
        <city>Toronto</city>
                                          <prov-state>Ontario</prov-state>
    <pcode-zip>M3Z 5H9</pcode-zip>
                                      </addr>
  <phone type="work">905-555-4789</phone>
  <other:phone xmlns:other="http://otherphone.com" other:type="home">
    416-555-3376</other:phone>
  <assistant>
    <name>Gopher Runner</name>
    <phone type="home">416-555-3426</phone>
  </assistant>
</customerinfo>
```

Bei Verwendung des folgenden Ausdrucks in der Klausel **RETURN** des Umsetzungsausdrucks werden die Knoten des Elements `phone`, die den Standardnamensbereich des Elements verwenden, als untergeordnete Knoten des Primärknotens (`primary`) angezeigt, und der Elementknoten `other:phone` wird als untergeordneter Knoten des Sekundärknotens (`secondary`) angezeigt:

```
<phonenumbers xmlns:other="http://otherphone.com">
  <primary>
    { $mycust//phone }
  </primary>
  <secondary>
    { $mycust//other:phone }
  </secondary>
</phonenumbers>
```

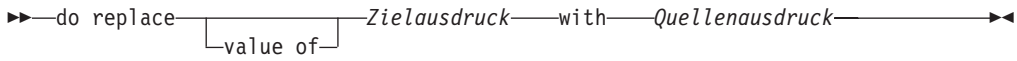
Bei Ausführung für die Datenbank SAMPLE gibt der Umsetzungsausdruck das folgende Ergebnis zurück:

```
<phonenumbers xmlns:other="http://otherphone.com" xmlns="http://posample.org">
  <primary>
    <phone type="work">905-555-4789</phone>
    <phone type="home">416-555-3426</phone>
  </primary>
  <secondary>
    <other:phone other:type="home">416-555-3376</other:phone>
  </secondary>
</phonenumbers>
```

## Ersetzungsausdruck

Ein Ersetzungsausdruck ersetzt entweder einen vorhandenen Knoten durch eine neue Sequenz aus null oder mehr Knoten oder den Wert eines Knotens bei gleichzeitiger Beibehaltung der Identität des betreffenden Knotens.

### Syntax




**do replace**
Die Schlüsselwörter, die einen Ersetzungsausdruck einleiten.

*Zielausdruck*
Ein XQuery-Ausdruck, der kein Aktualisierungsausdruck ist. Wenn der Ausdruck einen Kommaoperator auf oberster Ebene enthält, muss der Ausdruck in runde Klammern eingeschlossen werden. Das Ergebnis von *Zielausdruck* muss ein einzelner Knoten sein, der kein Dokumentknoten ist. Ist das Ergebnis von *Zielausdruck* ein Dokumentknoten, gibt DB2 XQuery einen Fehler zurück.

Werden die Schlüsselwörter **value of** nicht angegeben, muss das Ergebnis von *Zielausdruck* ein einzelner Knoten sein, dessen Elternmerkmal nicht leer ist.

**value of**
Die Schlüsselwörter, die angeben, dass der Wert des Knotens von *Zielausdruck* bei gleichzeitiger Beibehaltung der Identität des Knotens ersetzt werden soll.

**with**
Das Schlüsselwort, das den Quellenausdruck einleitet.

*Quellenausdruck*
Ein XQuery-Ausdruck, der kein Aktualisierungsausdruck ist. Wenn der Ausdruck einen Kommaoperator auf oberster Ebene enthält, muss der Ausdruck in runde Klammern eingeschlossen werden.

Werden die Schlüsselwörter **value of** angegeben, wird *Quellenausdruck* so ausgewertet wie der Inhaltsausdruck eines Textknotenkonstruktors. Das Ergebnis von *Quellenausdruck* ist ein einzelner Textknoten oder eine leere Sequenz.

Werden die Schlüsselwörter **value of** nicht angegeben, muss das Ergebnis von *Quellenausdruck* eine Knotensequenz sein. *Quellenausdruck* wird so ausgewertet wie ein in einem Elementkonstruktor eingeschlossener Ausdruck. Enthält die Sequenz von *Quellenausdruck* einen Dokumentknoten, wird dieser Dokumentknoten durch seine untergeordneten Elemente (Kinder) ersetzt. Die Sequenz von *Quellenausdruck* muss aus den folgenden Knotentypen bestehen:

- Wenn der Knoten von *Zielausdruck* ein Attributknoten ist, muss die Ersatzsequenz aus null oder mehr Knoten bestehen.

- Ist der Knoten von *Zielausdruck* ein Element-, Text-, Kommentar- oder Verarbeitungsanweisungsknoten, muss die Ersatzsequenz aus einer Kombination aus null oder mehr Element-, Text-, Kommentar- oder Verarbeitungsanweisungsknoten bestehen.

Die folgenden Aktualisierungen werden generiert, wenn die Schlüsselwörter **value of** angegeben sind:

- Wenn der Knoten von *Zielausdruck* ein Elementknoten ist, werden die vorhandenen untergeordneten Elemente des Knotens von *Zielausdruck* durch den von *Quellenausdruck* zurückgegebenen Textknoten ersetzt. Wenn *Quellenausdruck* eine leere Sequenz zurückgibt, wird das Kindermerkmal des Knotens von *Zielausdruck* leer. Enthält der Knoten von *Zielausdruck* Attributknoten, werden diese nicht beeinflusst.
- Wenn der Knoten von *Zielausdruck* kein Elementknoten ist, wird der Zeichenfolgewert des Knotens von *Zielausdruck* durch den Zeichenfolgewert des von *Quellenausdruck* zurückgegebenen Textknotens ersetzt. Wenn der *Quellenausdruck* keinen Textknoten zurückgibt, wird der Zeichenfolgewert des Knotens von *Zielausdruck* durch eine Zeichenfolge mit Nulllänge ersetzt.

Die folgenden Aktualisierungen werden generiert, wenn die Schlüsselwörter **value of** nicht angegeben sind:

- Die Knoten von *Quellenausdruck* ersetzen den Knoten von *Zielausdruck*. Der übergeordnete Knoten des Knotens von *Zielausdruck* wird der übergeordnete Knoten jedes einzelnen Knotens von *Quellenausdruck*. Die Knoten von *Quellenausdruck* nehmen die Position in der Knotenhierarchie ein, die zuvor vom Knoten von *Zielausdruck* besetzt war.
- Die Zuordnung des Knotens von *Zielausdruck* sowie aller seiner Attribute und untergeordneten Elemente zur Knotensequenz wird aufgehoben.

## Beispiele

In nachstehendem Beispiel wird die Tabelle CUSTOMER aus der DB2-Beispieldatenbank SAMPLE verwendet. In der Tabelle CUSTOMER enthält die XML-Spalte INFO die Adresse und Telefonnummer von Kunden.

In nachstehendem Beispiel erstellt die Klausel **COPY** des Umsetzungsausdrucks eine Kopie des XML-Dokuments aus der Spalte INFO. Der Ersetzungsausdruck ersetzt das Element `addr` (Adresse) und dessen untergeordneten Elemente (Kinder):

```
  xquery
declare default element namespace 'http://posample.org';
transform
copy $mycust := db2-fn:sqlquery('select info from customer where cid = 1000')
modify
  do replace $mycust/customerinfo/addr
    with
        <addr country="Canada">
        <street>1596 14th Avenue NW</street>
        <city>Calgary</city>
        <prov-state>Alberta</prov-state>
        <pcode-zip>T2N 1M7</pcode-zip>
                                          </addr>
return $mycust
```

Bei Ausführung für die Datenbank SAMPLE gibt der Ausdruck das folgende Ergebnis mit den ersetzten Adressinformationen zurück:

```
<customerinfo xmlns="http://posample.org" Cid="1000">
    <name>Kathy Smith</name>
    <addr country="Canada">
    <street>1596 14th Avenue NW</street>
    <city>Calgary</city>
    <prov-state>Alberta</prov-state>
    <pcode-zip>T2N 1M7</pcode-zip>
                                        </addr>
    <phone type="work">416-555-1358</phone>
</customerinfo>
```

Der folgende Ausdruck ersetzt den Wert home (Privatnummer) des Attributs type des Elements für die Kundentelefonnummer (customer phone) durch den Wert personal (persönlich):

```
  xquery
declare default element namespace 'http://posample.org';
transform
copy $mycust := db2-fn:sqlquery('select info from customer where cid = 1004')
modify
  do replace value of $mycust/customerinfo/phone[@type="home"]/@type with "personal"
return $mycust
```

Bei Ausführung für die Datenbank SAMPLE gibt der Ausdruck das folgende Ergebnis mit dem ersetzten Attributwert zurück:

```
<customerinfo xmlns="http://posample.org" Cid="1004">
  <name>Matt Foreman</name>
    <addr country="Canada">
        <street>1596 Baseline</street>
        <city>Toronto</city>
                                          <prov-state>Ontario</prov-state>
    <pcode-zip>M3Z 5H9</pcode-zip>
                                       </addr>
  <phone type="work">905-555-4789</phone>
  <phone type="personal">416-555-3376</phone>
  <assistant>
    <name>Gopher Runner</name>
    <phone type="home">416-555-3426</phone>
  </assistant>
</customerinfo>
```

Der Wert des Attributs 'phone' des Assistenten (assistant) wurde nicht geändert.

# Kapitel 5. Integrierte Funktionen

DB2 XQuery stellt eine Bibliothek integrierter Funktionen für die Arbeit mit XML-Daten bereit. Diese integrierten Funktionen umfassen auch XQuery-definierte Funktionen und DB2-definierte Funktionen.

**XQuery-definierte Funktionen**
XQuery-definierte Funktionen befinden sich in dem Namensbereich, der an das Präfix `fn` gebunden ist. Dieser Namensbereich ist der Standardfunktionsnamensbereich, sodass die XQuery-definierten Funktionen ohne Angabe eines Namensbereichspräfixes aufgerufen werden können. Wenn Sie diesen Standardfunktionsnamensbereich mit einer Deklaration für Standardfunktionsnamensbereiche im Abfrageprolog überschreiben, müssen Sie das Präfix `fn` verwenden, um XQuery-definierte Funktionen aufzurufen.

**DB2-definierte Funktionen**
Die in DB2 definierten Funktionen lauten db2-fn:xmlcolumn und db2-fn:sqlquery. Diese Funktionen werden verwendet, um über eine DB2-Datenbank auf XML-Werte zuzugreifen. Das Präfix `db2-fn` ist nicht der Standardfunktionsnamensbereich, sodass beim Aufrufen dieser Funktionen das Namensbereichspräfix verwendet werden muss, sofern der Standardnamensbereich nicht mit einer Deklaration für Standardfunktionsnamensbereiche im Abfrageprolog überschrieben wird.

## Funktionen nach Kategorie

Die folgenden Funktionskategorien stehen zur Verfügung: Zeichenfolgefunktionen (string), Boolesche Funktionen (Boolean), Zahlfunktionen (number), Datums- und Zeitfunktionen (date/time), Sequenzfunktionen (sequence), QName-Funktionen und sonstige Funktionen.

### Zeichenfolgefunktionen

| Funktion | Beschreibung |
|---|---|
| „Funktion 'codepoints-to-string'" auf Seite 159 | Die Funktion fn:codepoints-to-string gibt das Zeichenfolgeäquivalent einer Sequenz aus Unicode-Codepunkten zurück. |
| „Funktion 'compare'" auf Seite 160 | Die Funktion fn:compare vergleicht zwei Zeichenfolgen miteinander. |
| „Funktion 'concat'" auf Seite 161 | Die Funktion fn:concat gibt eine Zeichenfolge zurück, bei der es sich um die Verknüpfung von mindestens zwei atomaren Werten handelt. |
| „Funktion 'contains'" auf Seite 161 | Die Funktion fn:contains ermittelt, ob eine Zeichenfolge eine bestimmte Unterzeichenfolge enthält. Die Unterzeichenfolge wird unter Verwendung der Standardsortierfolge abgeglichen. |

| Funktion | Beschreibung |
| --- | --- |
| „Funktion 'ends-with'" auf Seite 171 | Die Funktion fn:ends-with ermittelt, ob eine Zeichenfolge mit einer bestimmten Zeichenfolge endet. Die Unterzeichenfolge wird unter Verwendung der Standardsortierfolge abgeglichen. |
| „Funktion 'lower-case'" auf Seite 180 | Die Funktion fn:lower-case setzt eine Zeichenfolge in Kleinbuchstaben um. |
| „Funktion 'matches'" auf Seite 181 | Die Funktion fn:matches ermittelt, ob eine Zeichenfolge mit einem bestimmten Muster übereinstimmt. |
| „Funktion 'normalize-space'" auf Seite 193 | Die Funktion fn:normalize-space entfernt führende und abschließende Leerzeichen aus einer Zeichenfolge und ersetzt jede interne Sequenz aus Leerzeichen durch ein einzelnes Leerzeichen. |
| „Funktion 'normalize-unicode'" auf Seite 194 | Die Funktion fn:normalize-unicode führt eine Unicode-Normalisierung für eine Zeichenfolge aus. |
| „Funktion 'replace'" auf Seite 199 | Die Funktion fn:replace vergleicht jede Gruppe von Zeichen innerhalb einer Zeichenfolge mit einem bestimmten Muster und ersetzt anschließend die Zeichen, die dem Muster entsprechen, mit einer anderen Gruppe von Zeichen. |
| „Funktion 'starts-with'" auf Seite 211 | Die Funktion fn:starts-with ermittelt, ob eine Zeichenfolge mit einer bestimmten Zeichenfolge beginnt. Die Unterzeichenfolge wird unter Verwendung der Standardsortierfolge abgeglichen. |
| „Funktion 'string'" auf Seite 212 | Die Funktion fn:string gibt die Zeichenfolgedarstellung eines Wertes zurück. |
| „Funktion 'string-join'" auf Seite 213 | Die Funktion fn:string-join gibt eine Zeichenfolge zurück, die durch Verknüpfung von Elementen erstellt wird, die durch ein Trennzeichen voneinander getrennt sind. |
| „Funktion 'string-length'" auf Seite 213 | Die Funktion fn:string-length gibt die Länge einer Zeichenfolge zurück. |
| „Funktion 'string-to-codepoints'" auf Seite 214 | Die Funktion fn:string-to-codepoints gibt eine Sequenz aus Unicode-Codepunkten zurück, die einem Zeichenfolgewert entsprechen. |
| „Funktion 'substring'" auf Seite 215 | Die Funktion fn:substring gibt eine Unterzeichenfolge einer Zeichenfolge zurück. |
| „Funktion 'substring-after'" auf Seite 216 | Die Funktion fn:substring-after gibt eine Unterzeichenfolge zurück, die in einer Zeichenfolge nach dem Ende des ersten Vorkommens eines bestimmten Suchbegriffs auftritt. Der Suchbegriff wird unter Verwendung der Standardsortierfolge abgeglichen. |

| Funktion | Beschreibung |
| --- | --- |
| „Funktion 'substring-before'" auf Seite 217 | Die Funktion fn:substring-before gibt eine Unterzeichenfolge zurück, die in einer Zeichenfolge vor dem ersten Auftreten eines bestimmten Suchbegriffs auftritt. Der Suchbegriff wird unter Verwendung der Standardsortierfolge abgeglichen. |
| „Funktion 'tokenize'" auf Seite 221 | Die Funktion fn:tokenize unterteilt eine Zeichenfolge in eine Sequenz aus Unterzeichenfolgen. |
| „Funktion 'translate'" auf Seite 223 | Die Funktion fn:translate ersetzt ausgewählte Zeichen in einer Zeichenfolge durch Ersetzungszeichen. |
| „Funktion 'upper-case'" auf Seite 225 | Die Funktion fn:upper-case setzt eine Zeichenfolge in Großbuchstaben um. |

## Boolesche Funktionen

| Funktion | Beschreibung |
| --- | --- |
| „Funktion 'boolean'" auf Seite 157 | Die Funktion fn:boolean gibt den effektiven Booleschen Wert einer Sequenz zurück. |
| „Funktion 'false'" auf Seite 172 | Die Funktion fn:false gibt den Wert 'false' (falsch) vom Typ 'xs:boolean' zurück. |
| „Funktion 'not'" auf Seite 195 | Die Funktion fn:not function gibt den Wert 'false' (falsch) zurück, wenn der effektive Boolesche Wert einer Sequenz 'true' (wahr) ist, bzw. 'true' (wahr), wenn der effektive Boolesche Wert einer Sequenz 'false' (falsch) ist. |
| „Funktion 'true'" auf Seite 224 | Die Funktion fn:true gibt den Wert 'true' (wahr) vom Typ 'xs:boolean' zurück. |

## Zahlfunktionen

| Funktion | Beschreibung |
| --- | --- |
| „Funktion 'abs'" auf Seite 155 | Die Funktion fn:abs gibt den absoluten Wert eines numerischen Werts zurück. |
| „Funktion 'avg'" auf Seite 156 | Die Funktion fn:avg gibt den Durchschnitt der Werte in einer Sequenz zurück. |
| „Funktion 'ceiling'" auf Seite 158 | Die Funktion fn:ceiling gibt die kleinste ganze Zahl zurück, die größer-gleich einem bestimmten numerischen Wert ist. |
| „Funktion 'floor'" auf Seite 173 | Die Funktion fn:floor gibt die größte ganze Zahl zurück, die kleiner-gleich einem bestimmten numerischen Wert ist. |
| „Funktion 'max'" auf Seite 183 | Die Funktion fn:max gibt das Maximum der Werte in einer Sequenz zurück. |
| „Funktion 'min'" auf Seite 184 | Die Funktion fn:min gibt das Minimum der Werte in einer Sequenz zurück. |
| „Funktion 'number'" auf Seite 196 | Die Funktion fn:number setzt einen Wert in den Datentyp 'xs:double' um. |

| Funktion | Beschreibung |
| --- | --- |
| „Funktion 'round'" auf Seite 203 | Die Funktion fn:round gibt diejenige ganze Zahl zurück, die einem bestimmten numerischen Wert am nächsten kommt. |
| „Funktion 'round-half-to-even'" auf Seite 204 | Die Funktion fn:round-half-to-even gibt denjenigen numerischen Wert mit einer bestimmten Genauigkeit zurück, der einem bestimmten numerischen Wert am nächsten kommt. |
| „Funktion 'sum'" auf Seite 218 | Die Funktion fn:sum gibt die Summe der Werte in einer Sequenz zurück. |

## Funktionen für Datumsangaben, Uhrzeit und Dauer

| Funktion | Beschreibung |
| --- | --- |
| „Funktion 'adjust-date-to-timezone'" auf Seite 150 | Die Funktion fn:adjust-date-to-timezone passt einen Wert vom Typ 'xs:date' an eine bestimmte Zeitzone an oder entfernt die Zeitzonenkomponente aus dem Wert. |
| „Funktion 'adjust-dateTime-to-timezone'" auf Seite 152 | Die Funktion fn:adjust-dateTime-to-timezone passt einen Wert vom Typ 'xs:dateTime' an eine bestimmte Zeitzone an oder entfernt die Zeitzonenkomponente aus dem Wert. |
| „Funktion 'adjust-time-to-timezone'" auf Seite 154 | Die Funktion fn:adjust-time-to-timezone passt einen Wert vom Typ 'xs:time' an eine bestimmte Zeitzone an oder entfernt die Zeitzonenkomponente aus dem Wert. |
| „Funktion 'current-date'" auf Seite 162 | Die Funktion fn:current-date gibt das aktuelle Datum in der impliziten Zeitzone UTC zurück. |
| „Funktion 'current-dateTime'" auf Seite 163 | Die Funktion fn:current-dateTime gibt das aktuelle Datum und die aktuelle Zeit in der impliziten Zeitzone UTC zurück. |
| „Funktion 'current-time'" auf Seite 163 | Die Funktion fn:current-time gibt die aktuelle Uhrzeit in der impliziten Zeitzone UTC zurück. |
| „Funktion 'dateTime'" auf Seite 164 | Die Funktion fn:dateTime erstellt einen Wert vom Typ 'xs:dateTime' auf der Grundlage eines Wertes vom Typ 'xs:date' und eines Wertes vom Typ 'xs:time'. |
| „Funktion 'day-from-date'" auf Seite 165 | Die Funktion fn:day-from-date gibt die Tageskomponente eines Wertes vom Typ 'xs:date' zurück. |
| „Funktion 'day-from-dateTime'" auf Seite 166 | Die Funktion fn:day-from-dateTime gibt die Tageskomponente eines Wertes vom Typ 'xs:dateTime' zurück. |
| „Funktion 'days-from-duration'" auf Seite 166 | Die Funktion fn:days-from-duration gibt die Tageskomponente einer Zeitdauer zurück. |
| „Funktion 'hours-from-dateTime'" auf Seite 174 | Die Funktion fn:hours-from-dateTime gibt die Stundenkomponente eines Wertes vom Typ 'xs:dateTime' zurück. |

| Funktion | Beschreibung |
| --- | --- |
| „Funktion 'hours-from-duration'" auf Seite 174 | Die Funktion fn:hours-from-duration gibt die Stundenkomponente eines Wertes für Dauer zurück. |
| „Funktion 'hours-from-time'" auf Seite 175 | Die Funktion fn:hours-from-time gibt die Stundenkomponente eines Wertes vom Typ 'xs:time' zurück. |
| „Funktion 'implicit-timezone'" auf Seite 176 | Die Funktion fn:implicit-timezone gibt den impliziten Zeitzonenwert PT0S zurück, der den Datentyp 'xs:dayTimeDuration' aufweist. Der Wert PT0S gibt an, dass es sich bei der impliziten Zeitzone um die Zeitzone UTC handelt. |
| „Funktion 'minutes-from-dateTime'" auf Seite 186 | Die Funktion fn:minutes-from-dateTime gibt die Minutenkomponente eines Wertes vom Typ 'xs:dateTime' zurück. |
| „Funktion 'minutes-from-duration'" auf Seite 186 | Die Funktion fn:minutes-from-duration gibt die Minutenkomponente einer Zeitdauer zurück. |
| „Funktion 'minutes-from-time'" auf Seite 187 | Die Funktion fn:minutes-from-time gibt die Minutenkomponente eines Wertes vom Typ 'xs:time' zurück. |
| „Funktion 'month-from-date'" auf Seite 188 | Die Funktion fn:month-from-date gibt die Monatskomponente eines Wertes vom Typ 'xs:date' zurück. |
| „Funktion 'month-from-dateTime'" auf Seite 188 | Die Funktion fn:month-from-dateTime gibt die Monatskomponente eines Wertes vom Typ 'xs:dateTime' zurück. |
| „Funktion 'months-from-duration'" auf Seite 189 | Die Funktion fn:months-from-duration gibt die Monatskomponente eines Wertes für Dauer zurück. |
| „Funktion 'seconds-from-dateTime'" auf Seite 206 | Die Funktion fn:seconds-from-dateTime gibt die Sekundenkomponente eines Wertes vom Typ 'xs:dateTime' zurück. |
| „Funktion 'seconds-from-duration'" auf Seite 206 | Die Funktion fn:seconds-from-duration gibt die Sekundenkomponente einer Zeitdauer zurück. |
| „Funktion 'seconds-from-time'" auf Seite 207 | Die Funktion fn:seconds-from-time gibt die Sekundenkomponente eines Wertes vom Typ 'xs:time' zurück. |
| „Funktion 'timezone-from-date'" auf Seite 219 | Die Funktion fn:timezone-from-date gibt die Zeitzonenkomponente eines Wertes vom Typ 'xs:date' zurück. |
| „Funktion 'timezone-from-dateTime'" auf Seite 220 | Die Funktion fn:timezone-from-dateTime gibt die Zeitzonenkomponente eines Wertes vom Typ 'xs:dateTime' zurück. |
| „Funktion 'timezone-from-time'" auf Seite 220 | Die Funktion fn:timezone-from-time gibt die Zeitzonenkomponente eines Wertes vom Typ 'xs:time' zurück. |
| „Funktion 'year-from-date'" auf Seite 227 | Die Funktion fn:year-from-date gibt die Jahreskomponente eines Wertes vom Typ 'xs:date' zurück. |

| Funktion | Beschreibung |
| --- | --- |
| „Funktion 'year-from-dateTime'" auf Seite 228 | Die Funktion fn:year-from-dateTime gibt die Jahreskomponente eines Wertes vom Typ 'xs:dateTime' zurück. |
| „Funktion 'years-from-duration'" auf Seite 228 | Die Funktion fn:years-from-duration gibt die Jahreskomponente einer Zeitdauer zurück. |

## Sequenzfunktionen

| Funktion | Beschreibung |
| --- | --- |
| „Funktion 'count'" auf Seite 162 | Die Funktion fn:count gibt die Anzahl der Werte in einer Sequenz zurück. |
| „Funktion 'data'" auf Seite 164 | Die Funktion fn:data gibt die Eingabesequenz zurück, nachdem alle Knoten in der Eingabesequenz durch ihre typisierten Werte ersetzt worden sind. |
| „Funktion 'deep-equal'" auf Seite 167 | Die Funktion fn:deep-equal vergleicht zwei Sequenzen miteinander, um zu ermitteln, ob sie die Anforderungen für tiefe Gleichheit erfüllen. |
| „Funktion 'distinct-values'" auf Seite 170 | Die Funktion fn:distinct-values gibt die unterschiedlichen Werte in einer Sequenz zurück. |
| „Funktion 'empty'" auf Seite 170 | Die Funktion fn:empty gibt an, ob das Argument eine leere Sequenz ist. |
| „Funktion 'exactly-one'" auf Seite 172 | Die Funktion fn:exactly-one gibt ihr Argument zurück, wenn es genau ein Element enthält. |
| „Funktion 'exists'" auf Seite 172 | Die Funktion fn:exists gibt an, ob eine Sequenz vorhanden und daher keine leere Sequenz ist. |
| „Funktion 'last'" auf Seite 179 | Die Funktion fn:last gibt die Anzahl der Werte in der Sequenz zurück, die zum jeweiligen Zeitpunkt verarbeitet wird. |
| „Funktion 'index-of'" auf Seite 177 | Die Funktion fn:index-of gibt die Positionen zurück, an denen ein Element in einer Sequenz vorkommt. |
| „Funktion 'insert-before'" auf Seite 178 | Die Funktion fn:insert-before fügt eine Sequenz vor einer bestimmten Position in eine andere Sequenz ein. |
| „Funktion 'one-or-more'" auf Seite 196 | Die Funktion fn:one-or-more gibt ihr Argument zurück, wenn es mindestens ein Element enthält. |
| „Funktion 'position'" auf Seite 197 | Die Funktion fn:position gibt die Position des Kontextelements in der Sequenz zurück, die zum jeweiligen Zeitpunkt verarbeitet wird. |
| „Funktion 'remove'" auf Seite 198 | Die Funktion fn:remove entfernt ein Element aus einer Sequenz. |
| „Funktion 'reverse'" auf Seite 202 | Die Funktion fn:reverse kehrt die Reihenfolge der Elemente in einer Sequenz um. |

| Funktion | Beschreibung |
|---|---|
| „Funktion 'subsequence'" auf Seite 214 | Die Funktion fn:subsequence gibt eine Teilsequenz einer Sequenz zurück. |
| „Funktion 'unordered'" auf Seite 224 | Die Funktion fn:unordered gibt die Elemente in einer Sequenz in nicht deterministischer Reihenfolge zurück. |
| „Funktion 'zero-or-one'" auf Seite 229 | Die Funktion fn:zero-or-one gibt ihr Argument zurück, wenn es entweder ein (1) Element enthält oder eine leere Sequenz ist. |

## QName-Funktionen

| Funktion | Beschreibung |
|---|---|
| „Funktion 'in-scope-prefixes'" auf Seite 176 | Die Funktion fn:in-scope-prefixes gibt eine Liste von Präfixen für alle gültigen Namensbereiche eines Elements zurück. |
| „Funktion 'local-name-from-QName'" auf Seite 180 | Die Funktion fn:local-name-from-QName gibt den lokalen Teil eines Wertes vom Datentyp 'xs:QName' zurück. |
| „Funktion 'namespace-uri-for-prefix'" auf Seite 192 | Die Funktion fn:namespace-uri-for-prefix gibt die Namensbereichs-URI zurück, die einem Präfix in den gültigen Namensbereichen eines Elements zugeordnet ist. |
| „Funktion 'namespace-uri-from-QName'" auf Seite 192 | Die Funktion fn:namespace-uri-from-QName gibt den Teil mit der Namensbereichs-URI eines Wertes vom Typ 'xs:QName' zurück. |
| „Funktion 'QName'" auf Seite 197 | Die Funktion fn:QName erstellt einen erweiterten Namen aus einer Namensbereichs-URI und einer Zeichenfolge, die einen lexikalischen qualifizierten Namen (mit einem optionalen Präfix) enthält. |
| „Funktion 'resolve-QName'" auf Seite 201 | Die Funktion fn:resolve-QName setzt eine Zeichenfolge, die einen lexikalischen qualifizierten Namen enthält, in einen erweiterten qualifizierten Namen um, indem sie die gültigen Namensbereiche eines Elements verwendet, um das Namensbereichspräfix in eine Namensbereichs-URI aufzulösen. |

## Knotenfunktionen

| Funktion | Beschreibung |
|---|---|
| „Funktion 'local-name'" auf Seite 179 | Die Funktion fn:local-name gibt den lokalen Namen eines Knotens zurück. |
| „Funktion 'name'" auf Seite 190 | Die Funktion fn:name gibt das Präfix und den lokalen Namensteil eines Knotennamens zurück. |
| „Funktion 'namespace-uri'" auf Seite 191 | Die Funktion fn:namespace-uri gibt die Namensbereichs-URI des qualifizierten Namens eines Knotens zurück. |

| Funktion | Beschreibung |
|---|---|
| „Funktion 'node-name'" auf Seite 193 | Die Funktion fn:node-name gibt den erweiterten qualifizierten Namen (QName) eines Knotens zurück. |
| „Funktion 'root'" auf Seite 203 | Die Funktion fn:root gibt den Rootknoten einer Baumstruktur zurück, zu der ein Knoten gehört. |

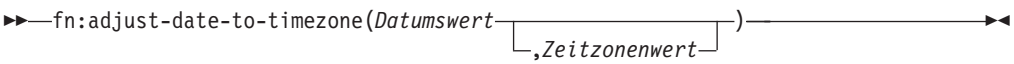
### Sonstige Funktionen

| Funktion | Beschreibung |
|---|---|
| „Funktion 'default-collation'" auf Seite 169 | Die Funktion fn:default-collation gibt eine URI zurück, die die Standardsortierfolge darstellt, die für die Datenbank definiert ist. |
| „Funktion 'sqlquery'" auf Seite 208 | Die Funktion db2-fn:sqlquery ruft eine Sequenz ab, die das Ergebnis einer SQL-Fullselect-Anweisung in der zum jeweiligen Zeitpunkt verbundenen DB2-Datenbank ist. |
| „Funktion 'xmlcolumn'" auf Seite 226 | Die Funktion db2-fn:xmlcolumn ruft eine Sequenz aus einer Spalte in der zum jeweiligen Zeitpunkt verbundenen DB2-Datenbank ab. |

## Funktion 'adjust-date-to-timezone'

Die Funktion fn:adjust-date-to-timezone passt einen Wert vom Typ 'xs:date' an eine bestimmte Zeitzone an oder entfernt die Zeitzonenkomponente aus dem Wert.

### Syntax




*Datumswert*
Der Datumswert, der angepasst werden soll.

*Datumswert* hat den Datentyp 'xs:date' oder ist eine leere Sequenz.

*Zeitzonenwert*
Eine Dauer, die die Zeitzone darstellt, an die *Datumswert* angepasst werden soll.

*Zeitzonenwert* kann eine leere Sequenz oder ein Einzelwert vom Typ 'xdt-:dayTimeDuration' zwischen -PT14H und PT14H einschließlich sein. Der Wert kann eine ganzzahlige Anzahl an Minuten aufweisen und darf keine Sekundenkomponente haben. Ist kein *Zeitzonenwert* angegeben, wird der Standardwert PT0H verwendet, der die Zeitzone UTC darstellt.

### Zurückgegebener Wert

In Abhängigkeit von den angegebenen Parametern ist der zurückgegebene Wert entweder ein Wert vom Typ 'xs:date' oder eine leere Sequenz. Ist *Datumswert* keine leere Sequenz, hat der zurückgegebene Wert den Typ 'xs:date'. Die folgende Tabelle enthält die möglichen Rückgabewerte:

*Tabelle 32. Typen von Eingabe- und Rückgabewerten für die Funktion fn:adjust-date-to-timezone*

| *Datumswert* | *Zeitzonenwert* | **Zurückgegebener Wert** |
|---|---|---|
| *Datumswert* mit einer Zeitzonenkomponente | Ein expliziter Wert oder kein Wert (Dauer von PT0H) | Der *Datumswert*, der an die von *Zeitzonenwert* dargestellte Zeitzone angepasst ist. |
| *Datumswert* mit einer Zeitzonenkomponente | Eine leere Sequenz | Der *Datumswert* ohne Zeitzonenkomponente. |
| *Datumswert* ohne Zeitzonenkomponente. | Ein expliziter Wert oder kein Wert (Dauer von PT0H) | Der *Datumswert* mit Zeitzonenkomponente. Die Zeitzonenkomponente ist die durch *Zeitzonenwert* dargestellte Zeitzone. Die Datumskomponente wird nicht an die Zeitzone angepasst. |
| *Datumswert* ohne Zeitzonenkomponente. | Eine leere Sequenz | Der *Datumswert*. |
| Eine leere Sequenz | Ein expliziter Wert, eine leere Sequenz oder kein Wert | Eine leere Sequenz. |

Beim Anpassen von *Datumswert* an eine andere Zeitzone, wird *Datumswert* als Wert für Datum und Uhrzeit mit der Zeitkomponente 00:00:00 behandelt. Der zurückgegebene Wert enthält die von *Zeitzonenwert* dargestellte Zeitzonenkomponente. Die folgende Funktion berechnet den angepassten Datumswert:

```
xs:date(fn:adjust-dateTime-to-timezone(xs:dateTime(Datumswert),Zeitzonenwert))
```

## Beispiele

In den nachstehenden Beispielen steht die Variable `$tz` für eine Zeitdauer von -10 Stunden, die als `xdt:dayTimeDuration("-PT10H")` definiert ist.

Die nachstehende Funktion passt den folgenden Datumswert an: 07. Mai 2002 in Zeitzone UTC+1. Die Funktion gibt den *Zeitzonenwert* -PT10H an:

```
fn:adjust-date-to-timezone(xs:date("2002-05-07+01:00"), $tz)
```

Der zurückgegebene Wert ist '2002-05-06-10:00'. Das Datum wird an die Zeitzone UTC-10 angepasst.

Die nachstehende Funktion fügt dem folgenden Wert eine Zeitzonenkomponente hinzu: 07. März 2002, ohne Zeitzonenkomponente. Die Funktion gibt den *Zeitzonenwert* -PT10H an:

```
fn:adjust-date-to-timezone(xs:date("2002-03-07"), $tz)
```

Der zurückgegebene Wert ist '2002-03-07-10:00'. Die Zeitzonenkomponente wird dem Datumswert hinzugefügt.

Die nachstehende Funktion passt den folgenden Datumswert an: 09. Februar 2002 in Zeitzone UTC-7. Da kein *Zeitzonenwert* angegeben ist, verwendet die Funktion den standardmäßigen *Zeitzonenwert* PT0H:

```
fn:adjust-date-to-timezone(xs:date("2002-02-09-07:00"))
```

Der zurückgegebene Datumswert lautet '2002-02-09Z'. Dieses Datum ist an die Zeitzone UTC angepasst.

Die nachstehende Funktion entfernt die Zeitzonenkomponente aus dem folgenden Datumswert: 07. Mai 2002, 10:00 Uhr in Zeitzone UTC-7. Der *Zeitzonenwert* ist eine leere Sequenz:

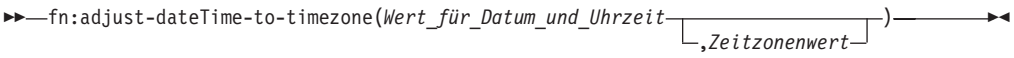
```
fn:adjust-date-to-timezone(xs:date("2002-05-07-07:00"), ())
```

Der zurückgegebene Wert lautet '2002-05-07'.

## Funktion 'adjust-dateTime-to-timezone'

Die Funktion fn:adjust-dateTime-to-timezone passt einen Wert vom Typ 'xs:dateTime' an eine bestimmte Zeitzone an oder entfernt die Zeitzonenkomponente aus dem Wert.

### Syntax




*Wert_für_Datum_und_Uhrzeit*
Der Wert für das Datum und die Uhrzeit, der angepasst werden soll.

*Wert_für_Datum_und_Uhrzeit* hat den Datentyp 'xs:dateTime' oder ist eine leere Sequenz.

*Zeitzonenwert*
Eine Dauer, die die Zeitzone darstellt, an die *Wert_für_Datum_und_Uhrzeit* angepasst werden soll.

*Zeitzonenwert* kann eine leere Sequenz oder ein Einzelwert vom Typ 'xdt:dayTimeDuration' zwischen -PT14H und PT14H einschließlich sein. Der Wert kann eine ganzzahlige Anzahl an Minuten aufweisen und darf keine Sekundenkomponente haben. Ist kein *Zeitzonenwert* angegeben, wird der Standardwert PT0H verwendet, der die Zeitzone UTC darstellt.

### Zurückgegebener Wert

In Abhängigkeit vom Typ der Eingabewerte ist der zurückgegebene Wert entweder ein Wert vom Typ 'xs:dateTime' oder eine leere Sequenz. Ist *Wert_für_Datum_und_Uhrzeit* keine leere Sequenz, hat der zurückgegebene Wert den Typ 'xs:dateTime'. Die folgende Tabelle enthält die möglichen Rückgabewerte:

*Tabelle 33. Typen von Eingabe- und Rückgabewerten für die Funktion fn:adjust-dateTime-to-timezone*

| *Wert_für_Datum_und_Uhrzeit* | *Zeitzonenwert* | **Zurückgegebener Wert** |
|---|---|---|
| *Wert_für_Datum_und_Uhrzeit* mit einer Zeitzonenkomponente | Ein expliziter Wert oder kein Wert (Dauer von PT0H) | Der *Wert_für_Datum_und_Uhrzeit*, der an die von *Zeitzonenwert* dargestellte Zeitzone angepasst ist. Der zurückgegebene Wert enthält die von *Zeitzonenwert* dargestellte Zeitzonenkomponente. |
| *Wert_für_Datum_und_Uhrzeit* mit einer Zeitzonenkomponente | Eine leere Sequenz | Der *Wert_für_Datum_und_Uhrzeit* ohne Zeitzonenkomponente. |

*Tabelle 33. Typen von Eingabe- und Rückgabewerten für die Funktion fn:adjust-dateTime-to-timezone (Forts.)*

| *Wert_für_Datum_und_Uhrzeit* | *Zeitzonenwert* | **Zurückgegebener Wert** |
|---|---|---|
| *Wert_für_Datum_und_Uhrzeit* ohne Zeitzonenkomponente. | Ein expliziter Wert oder kein Wert (Dauer von PT0H) | Der *Wert_für_Datum_und_Uhrzeit* mit einer Zeitzonenkomponente. Die Zeitzonenkomponente ist die durch *Zeitzonenwert* dargestellte Zeitzone. Die Datums- und Zeitkomponenten werden nicht an die Zeitzone angepasst. |
| *Wert_für_Datum_und_Uhrzeit* ohne Zeitzonenkomponente. | Eine leere Sequenz | Der *Wert_für_Datum_und_Uhrzeit*. |
| Eine leere Sequenz | Ein expliziter Wert, eine leere Sequenz oder kein Wert | Eine leere Sequenz. |

## Beispiele

In den nachstehenden Beispielen steht die Variable `$tz` für eine Zeitdauer von -10 Stunden, die als `xdt:dayTimeDuration("-PT10H")` definiert ist.

Die folgende Funktion passt das Datum und die Uhrzeit (hier den 07. März 2002, 10:00 Uhr in Zeitzone UTC-7) an die mit *Zeitzonenwert* angegebene Zeitzone -PT10H an:

```
fn:adjust-dateTime-to-timezone(xs:dateTime("2002-03-07T10:00:00-07:00"), $tz)
```

Der zurückgegebene Wert für Datum und Uhrzeit lautet '2002-03-07T07:00:00-10:00'.

Die nachstehende Funktion passt den folgenden Wert für Datum und Uhrzeit an: 07. März 2002, 10:00 Uhr. Der *Wert_für_Datum_und_Uhrzeit* hat keine Zeitzonenkomponente, und die Funktion gibt den *Zeitzonenwert* -PT10H an:

```
fn:adjust-dateTime-to-timezone(xs:dateTime("2002-03-07T10:00:00"), $tz)
```

Der zurückgegebene Wert für Datum und Uhrzeit lautet '2002-03-07T10:00:00-10:00'.

Die nachstehende Funktion passt den folgenden Wert für Datum und Uhrzeit an: 04. Juni 2006, 10:00 Uhr in Zeitzone UTC-7. Da kein *Zeitzonenwert* angegeben ist, verwendet die Funktion den Standardzeitzonenwert PT0H:

```
fn:adjust-dateTime-to-timezone(xs:dateTime("2006-06-04T10:00:00-07:00"))
```

Der zurückgegebene Wert lautet '2006-06-04T17:00:00Z'. Dies ist der an die Zeitzone UTC angepasste Wert für Datum und Uhrzeit.

Die nachstehende Funktion entfernt die Zeitzonenkomponente aus dem folgenden Wert für Datum und Uhrzeit: 07. März 2002, 10:00 Uhr in Zeitzone UTC-7. Der *Zeitzonenwert* ist eine leere Sequenz:

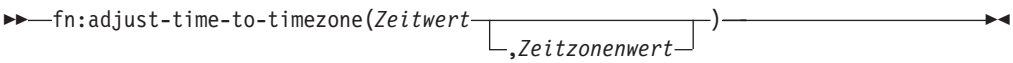
```
fn:adjust-dateTime-to-timezone(xs:dateTime("2002-03-07T10:00:00-07:00"), ())
```

Der zurückgegebene Wert für Datum und Uhrzeit lautet '2002-03-07T10:00:00'.

# Funktion 'adjust-time-to-timezone'

Die Funktion fn:adjust-time-to-timezone passt einen Wert vom Typ 'xs:time' an eine bestimmte Zeitzone an oder entfernt die Zeitzonenkomponente aus dem Wert.

## Syntax




*Zeitwert*
Der Zeitwert, der angepasst werden soll.

*Zeitwert* hat den Datentyp 'xs:time' oder ist eine leere Sequenz.

*Zeitzonenwert*
Eine Dauer, die die Zeitzone darstellt, an die *Zeitwert* angepasst werden soll.

*Zeitzonenwert* kann eine leere Sequenz oder ein Einzelwert vom Typ 'xdt-:dayTimeDuration' zwischen -PT14H und PT14H einschließlich sein. Der Wert kann eine ganzzahlige Anzahl an Minuten aufweisen und darf keine Sekundenkomponente haben. Ist kein *Zeitzonenwert* angegeben, wird der Standardwert PT0H verwendet, der die Zeitzone UTC darstellt.

## Zurückgegebener Wert

In Abhängigkeit von den angegebenen Parametern ist der zurückgegebene Wert entweder ein Wert vom Typ 'xs:time' oder eine leere Sequenz. Ist *Zeitwert* keine leere Sequenz, hat der zurückgegebene Wert den Typ 'xs:time'. Die folgende Tabelle enthält die möglichen Rückgabewerte:

*Tabelle 34. Typen von Eingabe- und Rückgabewerten für die Funktion fn:adjust-time-to-timezone*

| *Datumswert* | *Zeitzonenwert* | **Zurückgegebener Wert** |
|---|---|---|
| *Zeitwert* mit einer Zeitzonenkomponente | Ein expliziter Wert oder kein Wert (Dauer von PT0H) | Der *Zeitwert*, der an die von *Zeitzonenwert* dargestellte Zeitzone angepasst ist. Der zurückgegebene Wert enthält die von *Zeitzonenwert* dargestellte Zeitzonenkomponente. Wird bei der Zeitzonenanpassung Mitternacht überschritten, wird die Datumsänderung ignoriert. |
| *Zeitwert* mit einer Zeitzonenkomponente | Eine leere Sequenz | Der *Zeitwert* ohne Zeitzonenkomponente. |
| *Zeitwert* ohne Zeitzonenkomponente. | Ein expliziter Wert oder kein Wert (Dauer von PT0H) | Der *Zeitwert* mit Zeitzonenkomponente. Die Zeitzonenkomponente ist die durch *Zeitzonenwert* dargestellte Zeitzone. Die Zeitkomponente wird nicht an die Zeitzone angepasst. |
| *Zeitwert* ohne Zeitzonenkomponente. | Eine leere Sequenz | Der *Zeitwert*. |

*Tabelle 34. Typen von Eingabe- und Rückgabewerten für die Funktion fn:adjust-time-to-timezone (Forts.)*

| *Datumswert* | *Zeitzonenwert* | **Zurückgegebener Wert** |
|---|---|---|
| Eine leere Sequenz | Ein expliziter Wert, eine leere Sequenz oder kein Wert | Eine leere Sequenz. |

### Beispiele

In den nachstehenden Beispielen steht die Variable `$tz` für eine Zeitdauer von -10 Stunden, die als `xdt:dayTimeDuration("-PT10H")` definiert ist.

Die folgende Funktion passt den Zeitwert (hier 10:00 Uhr in Zeitzone UTC-7) an die mit *Zeitzonenwert* angegebene Zeitzone -PT10H an:

```
fn:adjust-time-to-timezone(xs:time("10:00:00-07:00"), $tz)
```

Der zurückgegebene Wert ist '7:00:00-10:00'. Die Uhrzeit ist an die von der Dauer -PT10H dargestellte Zeitzone angepasst.

Die nachstehende Funktion passt den Zeitwert 13:00 Uhr an. Der Zeitwert hat keine Zeitzonenkomponente:

```
fn:adjust-time-to-timezone(xs:time("13:00:00"), $tz)
```

Der zurückgegebene Wert ist '13:00:00-10:00'. Die Uhrzeit enthält eine Zeitzonenkomponente, die durch die Dauer von -PT10H dargestellt wird.

Die nachstehende Funktion passt den folgenden Zeitwert an: 10:00 Uhr in Zeitzone UTC-7. Die Funktion enthält keinen *Zeitzonenwert* und verwendet daher den Standardwert PT0H:

```
fn:adjust-time-to-timezone(xs:time("10:00:00-07:00"))
```

Der zurückgegebene Wert lautet '17:00:00Z'. Dies ist die an die Zeitzone UTC angepasste Uhrzeit.

Die nachstehende Funktion entfernt die Zeitzonenkomponente aus dem folgenden Zeitwert: 08:00 Uhr in Zeitzone UTC-7. Der *Zeitzonenwert* ist eine leere Sequenz:

```
fn:adjust-time-to-timezone(xs:time("08:00:00-07:00"), ())
```

Der zurückgegebene Wert ist '8:00:00'.

In nachstehendem Beispiel werden zwei Uhrzeiten miteinander verglichen. Die Zeitzonenanpassung überschreitet Mitternacht und führt daher zu einer Datumsänderung. Die Funktion fn:adjust-time-to-timezone ignoriert jedoch Datumsänderungen:

```
fn:adjust-time-to-timezone(xs:time("01:00:00+14:00"), $tz) eq xs:time("01:00:00-10:00")
```

Der zurückgegebene Wert ist 'true' (wahr).

## Funktion 'abs'

Die Funktion fn:abs gibt den absoluten Wert eines numerischen Werts zurück.

### Syntax

*numerischer_Wert*
Ein atomarer Wert oder eine leere Sequenz.

Ist *numerischer_Wert* ein atomarer Wert, weist er einen der folgenden Typen auf:
- xs:float
- xs:double
- xs:decimal
- xs:integer
- Einen Typ, der von einem der vorstehend aufgeführten Typen abgeleitet ist
- xdt:untypedAtomic

Hat *numerischer_Wert* den Typ 'xdt:untypedAtomic', wird er in einen Wert vom Typ 'xs:double' umgesetzt.

### Zurückgegebener Wert

Wenn *numerischer_Wert* keine leere Sequenz ist, ist der zurückgegebene Wert der absolute Wert von *numerischer_Wert*.

Ist *numerischer_Wert* eine leere Sequenz, gibt die Funktion 'fn:abs' die leere Sequenz zurück.

Der Datentyp des zurückgegebenen Wertes hängt vom Datentyp von *numerischer_Wert* ab:
- Hat *numerischer_Wert* den Datentyp 'xs:float', 'xs:double', 'xs:decimal' oder 'xs:integer', weist der zurückgegebene Wert denselben Typ wie *numerischer_Wert* auf.
- Hat *numerischer_Wert* einen Datentyp, der von 'xs:float', 'xs:double', 'xs:decimal' oder 'xs:integer' abgeleitet ist, weist der zurückgegebene Wert den direkten übergeordneten Datentyp von *numerischer_Wert* auf.
- Hat *numerischer_Wert* den Datentyp 'xdt:untypedAtomic', weist der zurückgegebene Wert den Datentyp 'xs:double' auf.

### Beispiel

Die folgende Funktion gibt den absoluten Wert von –10.5 zurück.

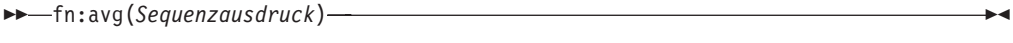
```
fn:abs(-10.5)
```

Der Wert 10.5 wird zurückgegeben.

## Funktion 'avg'

Die Funktion fn:avg gibt den Durchschnitt der Werte in einer Sequenz zurück.

### Syntax



*Sequenzausdruck*
Eine Sequenz mit Elementen, die einen der folgenden atomaren Typen aufweisen, oder eine leere Sequenz.
- xs:float
- xs:double
- xs:decimal
- xs:integer
- xdt:untypedAtomic
- xdt:dayTimeDuration
- xdt:yearMonthDuration
- Einen Typ, der von einem der vorstehend aufgeführten Typen abgeleitet ist

Eingabeelemente vom Typ 'xdt:untypedAtomic' werden in den Typ 'xs:double' umgesetzt. Nach dieser expliziten Typumsetzung müssen alle Elemente in der Eingabesequenz durch Umstufung oder Subtypsubstitution in einen gemeinsamen Typ konvertierbar sein. Der Durchschnittswert wird in diesem gemeinsamen Typ berechnet. Beispiel: Enthält die Eingabesequenz Elemente vom Typ 'money' (Geld), der von 'xs:decimal' abgeleitet ist, und vom Typ 'stockprice' (Aktienkurs), der von 'xs:float' abgeleitet ist, wird der Durchschnittswert im Typ 'xs:float' berechnet.

### Zurückgegebener Wert

Wenn *Sequenzausdruck* keine leere Sequenz ist, entspricht der zurückgegebene Wert dem Durchschnitt der Werte in *Sequenzausdruck*. Der Datentyp des zurückgegebenen Werts entspricht dem Datentyp der Elemente in *Sequenzausdruck* oder dem Datentyp, in den die Elemente in *Sequenzausdruck* umgestuft werden.

Ist *Sequenzausdruck* eine leere Sequenz, wird die leere Sequenz zurückgegeben.

### Beispiel

Die folgende Funktion gibt den Durchschnitt der Sequenz (5, 1.0E2, 40.5) zurück:

```
fn:avg((5, 1.0E2, 40.5))
```

Die Werte werden in den Datentyp 'xs:double' umgestuft. Die Funktion gibt den Wert 4.85E1 vom Typ 'xs:double' zurück, der als ″48.5″ serialisiert wird.

## Funktion 'boolean'

Die Funktion fn:boolean gibt den effektiven Booleschen Wert einer Sequenz zurück.

### Syntax

*Sequenzausdruck*
Eine beliebige Sequenz mit Elementen beliebigen Typs oder eine leere Sequenz.

### Zurückgegebener Wert

Der zurückgegebene effektive Boolesche Wert (EBW) hängt vom Wert von *Sequenzausdruck* ab:

*Tabelle 35. Für bestimmte Wertetypen in XQuery zurückgegebene effektive Boolesche Werte*

| Beschreibung des Wertes | Zurückgegebener EBW |
|---|---|
| Eine leere Sequenz | false (falsch) |
| Eine Sequenz mit einem Knoten als erstem Element | true (wahr) |
| Ein Einzelwert vom Typ 'xs:boolean' (oder von 'xs:boolean' abgeleitet) | false (falsch) - sofern der Wert von 'xs:boolean' 'false' ist |
| | true (wahr) - sofern der Wert von 'xs:boolean' 'true' ist |
| Ein Einzelwert vom Typ 'xs:string' oder 'xdt:untypedAtomic' (bzw. von einem von diesen Typen abgeleiteten Typ) | false (falsch) - sofern der Wert eine Nulllänge aufweist |
| | true (wahr) - sofern die Länge des Wertes größer als null ist |
| Ein Einzelwert eines numerischen Typs (oder von einem von einem numerischen Typ abgeleiteten Typ) | false (falsch) - sofern der Wert eine Nichtzahl (NaN) ist oder numerisch gleich null ist |
| | true (wahr) - sofern der Wert numerisch ungleich null ist |
| Alle anderen Werte | error (Fehler) |

**Anmerkung:** Der effektive Boolesche Wert einer Sequenz, die mindestens einen Knoten und mindestens einen atomaren Wert enthält, ist in einer Abfrage mit unvorhersehbarer Reihenfolge nicht deterministisch.

### Beispiele

**Beispiel mit einem Argument aus einem einzelnen numerischen Wert:** Die folgende Funktion gibt den effektiven Booleschen Wert des Wertes 0 zurück:

```
fn:boolean(0)
```

Der zurückgegebene Wert ist 'false' (falsch).

**Beispiel mit einem Argument aus einer Sequenz mit mehreren Elementen:** Die folgende Funktion gibt den effektiven Booleschen Wert der Sequenz (<a/>, 0, <b/>) zurück:

```
fn:boolean((<a/>, 0, <b/>))
```

Der zurückgegebene Wert ist 'true' (wahr).

## Funktion 'ceiling'

Die Funktion fn:ceiling gibt die kleinste ganze Zahl zurück, die größer-gleich einem bestimmten numerischen Wert ist.

### Syntax

*numerischer_Wert*
Ein atomarer Wert oder eine leere Sequenz.

Ist *numerischer_Wert* ein atomarer Wert, weist er einen der folgenden Typen auf:
- xs:float

- xs:double
- xs:decimal
- xs:integer
- xdt:untypedAtomic
- Einen Typ, der von einem der vorstehend aufgeführten Typen abgeleitet ist

Hat *numerischer_Wert* den Typ 'xdt:untypedAtomic', wird er in einen Wert vom Typ 'xs:double' umgesetzt.

### Zurückgegebener Wert

Wenn *numerischer_Wert* keine leere Sequenz ist, ist der zurückgegebene Wert die kleinste ganze Zahl, die größer-gleich *numerischer_Wert* ist. Der Datentyp des zurückgegebenen Wertes hängt vom Datentyp von *numerischer_Wert* ab:

- Hat *numerischer_Wert* den Datentyp 'xs:float', 'xs:double', 'xs:decimal' oder 'xs:integer', weist der zurückgegebene Wert denselben Typ wie *numerischer_Wert* auf.
- Hat *numerischer_Wert* einen Datentyp, der von 'xs:float', 'xs:double', 'xs:decimal' oder 'xs:integer' abgeleitet ist, weist der zurückgegebene Wert den direkten übergeordneten Datentyp von *numerischer_Wert* auf.

Ist *numerischer_Wert* eine leere Sequenz, ist der zurückgegebene Wert die leere Sequenz.

### Beispiele

**Beispiel mit einem positiven Argument:** Die folgende Funktion gibt den ceiling-Wert von 0.5 zurück:

```
fn:ceiling(0.5)
```

Der zurückgegebene Wert ist 1.

**Beispiel mit einem negativen Argument:** Die folgende Funktion gibt den ceiling-Wert von (-1.2) zurück:

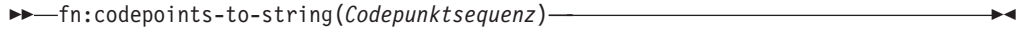
```
fn:ceiling(-1.2)
```

Der zurückgegebene Wert ist -1.

## Funktion 'codepoints-to-string'

Die Funktion fn:codepoints-to-string gibt das Zeichenfolgeäquivalent einer Sequenz aus Unicode-Codepunkten zurück.

### Syntax




*Codepunktsequenz*
Eine Sequenz aus ganzen Zahlen, die Unicode-Codepunkten entsprechen, oder eine leere Sequenz.

### Zurückgegebener Wert

Wenn *Codepunktsequenz* keine leere Sequenz ist, ist der zurückgegebene Wert eine Zeichenfolge, bei der es sich um die Verknüpfung der Zeichenäquivalente der in

*Codepunktsequenz* enthaltenen Elemente handelt. Ist eines der Elemente in *Codepunktsequenz* kein gültiger Unicode-Codepunkt, wird ein Fehler zurückgegeben.

Ist *Codepunktsequenz* eine leere Sequenz, ist der zurückgegebene Wert eine Zeichenfolge mit Nulllänge.

### Beispiel

Die folgende Funktion gibt das Zeichenäquivalent der Sequenz aus UTF-8-Codepunkten (88,81,117,101,114,121) zurück:

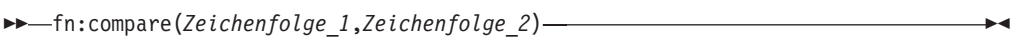
```
fn:codepoints-to-string((88,81,117,101,114,121))
```

Der zurückgegebene Wert ist 'XQuery'.

## Funktion 'compare'

Die Funktion fn:compare vergleicht zwei Zeichenfolgen miteinander.

### Syntax




*Zeichenfolge_1* , *Zeichenfolge_2*
Die Werte vom Typ 'xs:string', die verglichen werden sollen.

### Zurückgegebener Wert

Wenn *Zeichenfolge_1* und *Zeichenfolge_2* keine leere Sequenzen sind, wird einer der folgenden Werte zurückgegeben:

**-1** Wenn *Zeichenfolge_1* kleiner als *Zeichenfolge_2* ist.

**0** Wenn *Zeichenfolge_1* gleich *Zeichenfolge_2* ist.

**1** Wenn *Zeichenfolge_1* größer als *Zeichenfolge_2* ist.

*Zeichenfolge_1* und *Zeichenfolge_2* sind gleich, wenn sie dieselbe Länge haben, einschließlich einer Nulllänge, und alle entsprechenden Zeichen gemäß der Standardsortierfolge gleich sind.

Wenn *Zeichenfolge_1* und *Zeichenfolge_2* ungleich sind, wird ihre Beziehung (d. h. der größere Wert der beiden) ermittelt, indem das erste Paar ungleicher Zeichen vom linken Ende der Zeichenfolgen her verglichen wird. Dieser Vergleich erfolgt gemäß der Standardsortierfolge.

Wenn *Zeichenfolge_1* länger als *Zeichenfolge_2* ist und alle Zeichen von *Zeichenfolge_2* gleich den führenden Zeichen von *Zeichenfolge_1* sind, ist *Zeichenfolge_1* größer als *Zeichenfolge_2*.

Wenn *Zeichenfolge_1* oder *Zeichenfolge_2* eine leere Sequenz ist, wird die leere Sequenz zurückgegeben.

### Beispiel

Die folgende Funktion vergleicht die Zeichenfolge 'ABC' mit der Zeichenfolge 'ABD' anhand der Standardsortierfolge:

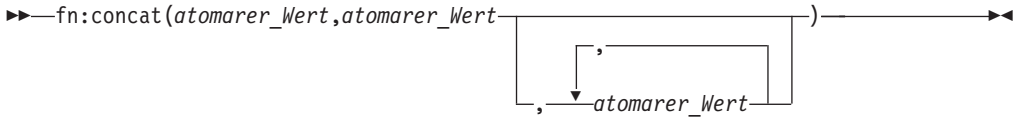
```
fn:compare('ABC', 'ABD')
```

'ABC' ist kleiner als 'ABD'. Der zurückgegebene Wert ist -1.

## Funktion 'concat'

Die Funktion fn:concat gibt eine Zeichenfolge zurück, bei der es sich um die Verknüpfung von mindestens zwei atomaren Werten handelt.

### Syntax




*atomarer_Wert*
Ein atomarer Wert oder eine leere Sequenz. Wenn ein Argument eine leere Sequenz ist, wird es als Zeichenfolge mit Nulllänge behandelt. Wenn *atomarer_Wert* kein Wert vom Typ 'xs:string' ist, wird er vor dem Verknüpfen der Werte in 'xs:string' umgesetzt.

### Zurückgegebener Wert

Wenn alle Argumente vom Typ *atomarer_Wert* eine leere Sequenz sind, ist der zurückgegebene Wert eine Zeichenfolge mit Nulllänge. Ansonsten ist der zurückgegebene Wert die Verknüpfung der Werte vom Typ 'xs:string', die aus der Umsetzung der Argumente vom Typ *atomarer_Wert* in Zeichenfolgen resultieren.

### Beispiel

Die folgende Funktion verknüpft die Zeichenfolgen 'ABC', 'ABD', eine leere Sequenz und 'ABE':

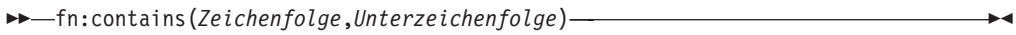
```
fn:concat('ABC', 'ABD', (), 'ABE')
```

Der zurückgegebene Wert ist 'ABCABDABE'.

## Funktion 'contains'

Die Funktion fn:contains ermittelt, ob eine Zeichenfolge eine bestimmte Unterzeichenfolge enthält. Die Unterzeichenfolge wird unter Verwendung der Standardsortierfolge abgeglichen.

### Syntax




*Zeichenfolge*
Die Zeichenfolge, in der nach *Unterzeichenfolge* gesucht werden soll.

*Zeichenfolge* hat den Datentyp 'xs:string' oder ist eine leere Sequenz. Ist *Zeichenfolge* eine leere Sequenz, wird *Zeichenfolge* auf eine Zeichenfolge mit Nulllänge gesetzt.

*Unterzeichenfolge*
Die Unterzeichenfolge, nach der in *Zeichenfolge* gesucht werden soll.

*Unterzeichenfolge* hat den Datentyp 'xs:string' oder ist eine leere Sequenz.

**Längenbeschränkung**

Die Länge von *Unterzeichenfolge* ist auf 32.000 Byte beschränkt.

### Zurückgegebener Wert

Der zurückgegebene Wert ist der Wert 'true' (wahr) vom Typ 'xs:boolean', wenn eine der folgenden Bedingungen erfüllt ist:

- *Unterzeichenfolge* kommt an einer beliebigen Stelle in *Zeichenfolge* vor.
- *Unterzeichenfolge* ist eine leere Sequenz oder eine Zeichenfolge mit Nulllänge.

Ansonsten ist der zurückgegebene Wert 'false' (falsch).

### Beispiel

Die folgende Funktion ermittelt, ob die Zeichenfolge 'Test literal' die Unterzeichenfolge 'lite' enthält:

```
fn:contains('Test literal','lite')
```

Der zurückgegebene Wert ist 'true' (wahr).

## Funktion 'count'

Die Funktion fn:count gibt die Anzahl der Werte in einer Sequenz zurück.

### Syntax

*Sequenzausdruck*
Eine Sequenz mit Elementen beliebigen Typs oder eine leere Sequenz.

### Zurückgegebener Wert

Wenn *Sequenzausdruck* keine leere Sequenz ist, wird die Anzahl der Werte in *Sequenzausdruck* zurückgegeben. Ist *Sequenzausdruck* eine leere Sequenz, wird 0 zurückgegeben.

### Beispiel

Die folgende Funktion gibt die Anzahl der Elemente in der Sequenz (5, 1.0E2, 40.5) zurück:

```
fn:count((5, 1.0E2, 40.5))
```

Der zurückgegebene Wert ist 3.

## Funktion 'current-date'

Die Funktion fn:current-date gibt das aktuelle Datum in der impliziten Zeitzone UTC zurück.

### Syntax

### Zurückgegebener Wert

Der zurückgegebene Wert hat den Typ 'xs:date' und ist das aktuelle Datum.

### Beispiel

Die folgende Funktion gibt das aktuelle Datum zurück:

```
fn:current-date()
```

Wenn diese Funktion am 2. Dezember 2005 aufgerufen worden wäre, hätte der zurückgegebene Wert 2005-12-02Z gelautet.

## Funktion 'current-dateTime'

Die Funktion fn:current-dateTime gibt das aktuelle Datum und die aktuelle Zeit in der impliziten Zeitzone UTC zurück.

### Syntax

### Zurückgegebener Wert

Der zurückgegebene Wert hat den Typ 'xs:dateTime' und ist das aktuelle Datum und die aktuelle Uhrzeit.

### Beispiel

Die folgende Funktion gibt das aktuelle Datum und die aktuelle Uhrzeit zurück:

```
fn:current-dateTime()
```

Wenn diese Funktion am 2. Dezember 2005 um 6:25 Uhr in Toronto (Zeitzone -PT5H) aufgerufen worden wäre, hätte der zurückgegebene Wert 2005-12-02T011:25:30.864001Z gelautet.

## Funktion 'current-time'

Die Funktion fn:current-time gibt die aktuelle Uhrzeit in der impliziten Zeitzone UTC zurück.

### Syntax

### Zurückgegebener Wert

Der zurückgegebene Wert hat den Typ 'xs:time' und ist die aktuelle Uhrzeit.

### Beispiel

Die folgende Funktion gibt die aktuelle Uhrzeit zurück:

```
fn:current-time()
```

Würde diese Funktion um 6:31 Westeuropäischer Zeit aufgerufen, könnte der zurückgegebene Wert beispielsweise '06:31:35.519001Z' lauten.

## Funktion 'data'

Die Funktion fn:data gibt die Eingabesequenz zurück, nachdem alle Knoten in der Eingabesequenz durch ihre typisierten Werte ersetzt worden sind.

### Syntax

*Sequenzausdruck*
Eine beliebige Sequenz, einschließlich einer leeren Sequenz.

### Zurückgegebener Wert

Ist *Sequenzausdruck* eine leere Sequenz, ist der zurückgegebene Wert eine leere Sequenz.

Ist *Sequenzausdruck* ein einzelner atomarer Wert, ist der zurückgegebene Wert *Sequenzausdruck*.

Ist *Sequenzausdruck* ein einzelner Knoten, ist der zurückgegebene Wert der typisierte Wert von *Sequenzausdruck*.

Ist *Sequenzausdruck* eine Sequenz aus mehr als einem Element, wird eine Sequenz aus atomaren Werten für die Elemente in *Sequenzausdruck* zurückgegeben. Jeder atomare Wert in *Sequenzausdruck* bleibt unverändert. Jeder Knoten in *Sequenzausdruck* wird durch seinen jeweiligen typisierten Wert ersetzt, bei dem es sich um eine Sequenz aus null oder mehr atomaren Werten handelt.

### Beispiel

Die folgende Funktion gibt eine Sequenz mit den atomaren Werten aus der Sequenz (<x xsi:type="string">ABC</x>,<y xsi:type="decimal">1.23</y>) zurück:

```
fn:data((<x xsi:type="string">ABC</x>,<y xsi:type="decimal">1.23</y>))
```

Der zurückgegebene Wert ist ("ABC",1.23).

## Funktion 'dateTime'

Die Funktion fn:dateTime erstellt einen Wert vom Typ 'xs:dateTime' auf der Grundlage eines Wertes vom Typ 'xs:date' und eines Wertes vom Typ 'xs:time'.

### Syntax

*Datumswert*
Ein Wert vom Typ 'xs:date'.

*Zeitwert*
Ein Wert vom Typ 'xs:time'.

### Zurückgegebener Wert

Der zurückgegebene Wert ist ein Wert vom Typ 'xs:dateTime' mit einer Datumskomponente entsprechend *Datumswert* und einer Zeitkomponente entsprechend *Zeitwert*. Die Zeitzone des Ergebnisses wird wie folgt berechnet:

- Wenn keines der Argumente eine Zeitzone aufweist, hat das Ergebnis ebenfalls keine Zeitzone.
- Wenn nur eines der Argumente eine Zeitzone aufweist oder wenn beide Argumente dieselbe Zeitzone aufweisen, hat das Ergebnis genau diese Zeitzone.
- Wenn beide Argumente jeweils eine unterschiedliche Zeitzone aufweisen, wird ein Fehler zurückgegeben.

### Beispiel

Die folgende Funktion gibt einen Wert vom Typ 'xs:dateTime' auf der Grundlage eines Wertes vom Typ 'xs:date' und eines Wertes vom Typ 'xs:time' zurück.

```
fn:dateTime((xs:date("2005-04-16")), (xs:time("12:30:59")))
```

Der zurückgegebene Wert ist der Wert '2005-04-16T12:30:59' vom Typ 'xs:dateTime'.

## Funktion 'day-from-date'

Die Funktion fn:day-from-date gibt die Tageskomponente eines Wertes vom Typ 'xs:date' zurück.

### Syntax

*Datumswert*
Der Datumswert, aus dem die Tageskomponente extrahiert werden soll.

*Datumswert* hat den Datentyp 'xs:date' oder ist eine leere Sequenz.

### Zurückgegebener Wert

Hat *Datumswert* den Typ 'xs:date', weist der zurückgegebene Wert den Typ 'xs:integer' auf und liegt im Bereich von 1 bis 31 einschließlich. Der Wert ist die Tageskomponente in *Datumswert*.

Ist *Datumswert* eine leere Sequenz, ist der zurückgegebene Wert eine leere Sequenz.

### Beispiel

Die folgende Funktion gibt die Tageskomponente des folgenden Datumswerts zurück: 01. Juni 2005:

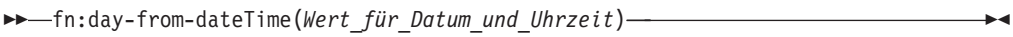
```
fn:day-from-date(xs:date("2005-06-01"))
```

Der zurückgegebene Wert ist 1.

## Funktion 'day-from-dateTime'

Die Funktion fn:day-from-dateTime gibt die Tageskomponente eines Wertes vom Typ 'xs:dateTime' zurück.

### Syntax




*Wert_für_Datum_und_Uhrzeit*
Der Wert für Datum und Uhrzeit, aus dem die Tageskomponente extrahiert werden soll.

*Wert_für_Datum_und_Uhrzeit* hat den Datentyp 'xs:dateTime' oder ist eine leere Sequenz.

### Zurückgegebener Wert

Hat *Wert_für_Datum_und_Uhrzeit* den Typ 'xs:dateTime', weist der zurückgegebene Wert den Typ 'xs:integer' auf und liegt im Bereich von 1 bis 31 einschließlich. Der Wert ist die Tageskomponente von *Wert für Datum und Uhrzeit*.

Ist *Wert_für_Datum_und_Uhrzeit* eine leere Sequenz, ist der zurückgegebene Wert eine leere Sequenz.

### Beispiel

Die folgende Funktion gibt die Tageskomponente des folgenden Werts für Datum und Uhrzeit zurück: 31. Januar 2005, 20:00 Uhr in Zeitzone UTC+4:

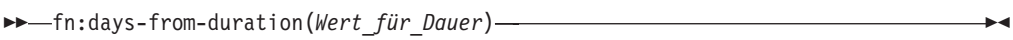
```
fn:day-from-dateTime(xs:dateTime("2005-01-31T20:00:00+04:00"))
```

Der zurückgegebene Wert lautet '31'.

## Funktion 'days-from-duration'

Die Funktion fn:days-from-duration gibt die Tageskomponente einer Zeitdauer zurück.

### Syntax




*Wert_für_Dauer*
Der Wert für die Zeitdauer, aus dem die Tageskomponente extrahiert werden soll.

*Wert_für_Dauer* ist eine leere Sequenz oder ein Wert, der einen der folgenden Datentypen aufweist: xdt:dayTimeDuration, xs:duration oder xdt:yearMonthDuration.

### Zurückgegebener Wert

Der zurückgegebene Wert hängt vom Datentyp von *Wert_für_Dauer* ab:

- Hat *Wert_für_Dauer* den Typ 'xdt:dayTimeDuration' oder 'xs:duration', weist der zurückgegebene Wert den Typ 'xs:integer' auf und die Komponente der Tage aus

*Wert_für_Dauer* wird als 'xdt:dayTimeDuration' umgesetzt. Der zurückgegebene Wert ist negativ, wenn *Wert_für_Dauer* negativ ist.

- Hat *Wert_für_Dauer* den Datentyp 'xdt:yearMonthDuration', ist der zurückgegebene Wert 0.
- Ist *Wert_für_Dauer* eine leere Sequenz, ist der zurückgegebene Wert eine leere Sequenz.

Die als 'xdt:dayTimeDuration' umgesetzte Tageskomponente aus *Wert_für_Dauer* ist die ganzzahlige Anzahl der Tage, die als (*S* `idiv 86400`) berechnet ist. Der Wert *S* ist die Gesamtzahl der Sekunden von *Wert_für_Dauer*, die als 'xdt:dayTimeDuration' umgesetzt wird, um die Komponenten der Jahre und Monate zu entfernen.

### Beispiele

Die folgende Funktion gibt die Komponente der Tage in der Dauer als -10 Tage und 0 Stunden zurück:

```
fn:days-from-duration(xdt:dayTimeDuration("-P10DT00H"))
```

Der zurückgegebene Wert ist -10.

Die folgende Funktion gibt die Komponente der Tage in der Dauer als 3 Tage und 55 Stunden zurück:

```
fn:days-from-duration(xdt:dayTimeDuration("P3DT55H"))
```

Der zurückgegebene Wert ist 5. Bei der Berechnung der Gesamtzahl der Tage in der Dauer werden die 55 Stunden in 2 Tage und 7 Stunden umgewandelt. Die Dauer entspricht P5D7H. Die Tageskomponente gibt hierbei 5 Tage an.

## Funktion 'deep-equal'

Die Funktion fn:deep-equal vergleicht zwei Sequenzen miteinander, um zu ermitteln, ob sie die Anforderungen für tiefe Gleichheit erfüllen.

### Syntax

*Sequenz_1*, *Sequenz_2*
: Die Sequenzen, die verglichen werden sollen. Die Elemente in den einzelnen Sequenzen können atomare Werte eines beliebigen Typs oder Knoten sein.

### Zurückgegebener Wert

Der zurückgegebene Wert ist der Wert 'true' (wahr) vom Typ 'xs:boolean', wenn *Sequenz_1* und *Sequenz_2* eine tiefe Gleichheit aufweisen. Ansonsten wird der Wert 'false' (falsch) zurückgegeben.

Wenn *Sequenz_1* und *Sequenz_2* eine leere Sequenz sind, weisen Sie eine tiefe Gleichheit auf.

Wenn die beiden Sequenzen nicht leer sind, weisen sie eine tiefe Gleichheit auf, wenn sie die beiden folgenden Bedingungen erfüllen:

- Die Anzahl der Elemente in *Sequenz_1* entspricht der Anzahl der Elemente in *Sequenz_2*.
- Jedes Element in *Sequenz_1* (*Element_1*) erfüllt die Bedingungen für tiefe Gleichheit mit dem entsprechenden Eintrag in *Sequenz_2* (*Element_2*). *Element_1* und *Element_2* weisen eine tiefe Gleichheit auf, wenn sie eine der folgenden Bedingungen erfüllen:
  – *Element_1* und *Element_2* sind beide atomare Werte und erfüllen eine der folgenden Bedingungen:
    - Der Ausdruck *Element_1* `eq` *Element_2* gibt 'true' (wahr) zurück.
    - Sowohl *Element_1* als auch *Element_2* sind vom Typ 'type xs:float' oder 'xs:double' und haben den Wert 'NaN'.
  – *Element_1* und *Element_2* sind beide Knoten derselben Sorte und erfüllen die Bedingungen für tiefe Gleichheit in der nachstehenden Tabelle.

*Tabelle 36. Tiefe Gleichheit für Knoten in einer Sequenz*

| **Knotensorte von Element_1 und Element_2** | **Bedingungen für tiefe Gleichheit** |
|---|---|
| Dokument | Die Sequenz aus Text- und Elementkindern von Element_1 weist eine tiefe Gleichheit mit der Sequenz aus den Text- und Elementkindern von Element_2 auf. |
| Element | Sämtliche der folgenden Bedingungen müssen erfüllt sein:<br>• *Element_1* und *Element_2* haben denselben Namen, d. h. sowohl ihre Namensbereichs-URIs als auch ihre lokalen Namen stimmen überein. Namensbereichspräfixe werden ignoriert. Der Namensabgleich erfolgt über einen binären Vergleich.<br>• *Element_1* und *Element_2* verfügen über dieselbe Anzahl an Attributen, und jedes Attribut von *Element_1* weist eine tiefe Gleichheit mit einem Attribut von *Element_2* auf.<br>• Eine der folgenden Bedingungen ist erfüllt:<br>– Beide Knoten wurden entweder keiner Gültigkeitsprüfung unterzogen oder beiden Knoten wurde bei der Gültigkeitsprüfung ein Typ zugeordnet, bei dem ein gemischter Inhalt (sowohl Textelemente als auch untergeordnete Elemente) zulässig ist, und die Sequenz aus den Text- und Elementkindern von *Element_1* weist eine tiefe Gleichheit mit der Sequenz aus den Text- und Elementkindern von *Element_2* auf.<br>– Beiden Knoten wurde bei der Gültigkeitsprüfung ein einfacher Typ (beispielsweise 'as xs:decimal') oder ein Typ mit einfachem Inhalt (beispielsweise der Typ "temperature", dessen Inhalt 'xs:decimal' ist) zugeordnet, und der typisierte Wert von *Element_1* weist eine tiefe Gleichheit mit dem typisierten Wert von *Element_2* auf.<br>– Beiden Knoten wurde bei der Gültigkeitsprüfung ein Typ zugeordnet, für den kein Inhalt zulässig ist (weder Text- noch Kindelemente).<br>– Beiden Knoten wurde bei der Gültigkeitsprüfung ein Typ zugeordnet, für den ausschließlich Kindelemente (kein Text) zulässig ist, und jedes Kindelement von *Element_1* weist eine tiefe Gleichheit mit dem entsprechenden Kindelement von *Element_2* auf. |

*Tabelle 36. Tiefe Gleichheit für Knoten in einer Sequenz (Forts.)*

| **Knotensorte von Element_1 und Element_2** | **Bedingungen für tiefe Gleichheit** |
|---|---|
| Attribut | Sämtliche der folgenden Bedingungen müssen erfüllt sein:<br>• *Element_1* und *Element_2* haben denselben Namen, d. h. sowohl ihre Namensbereichs-URIs als auch ihre lokalen Namen stimmen überein. Namensbereichspräfixe werden ignoriert. Der Namensabgleich erfolgt über einen binären Vergleich.<br>• Der typisierte Wert von *Element_1* weist eine tiefe Gleichheit mit dem typisierten Wert von *Element_2* auf. |
| Text | Die Merkmalwerte des Inhalts sind identisch, wenn sie als Zeichenfolgen mithilfe des Operators eq unter Verwendung der Standardsortierfolge verglichen werden. |
| Kommentar | Die Merkmalwerte des Inhalts sind identisch, wenn sie als Zeichenfolgen mithilfe des Operators eq unter Verwendung der Standardsortierfolge verglichen werden. |
| Verarbeitungsanweisung | Sämtliche der folgenden Bedingungen müssen erfüllt sein:<br>• *Element_1* und *Element_2* haben denselben Namen.<br>• Die Merkmalwerte des Inhalts sind identisch, wenn sie als Zeichenfolgen mithilfe des Operators eq unter Verwendung der Standardsortierfolge verglichen werden. |

### Beispiel

Die folgende Funktion vergleicht die Sequenzen (1,'ABC') und (1,'ABCD') auf tiefe Gleichheit. Bei Zeichenfolgevergleichen wird die Standardkorrelation verwendet.

```
fn:deep-equal((1,'ABC'), (1,'ABCD'))
```

Der zurückgegebene Wert ist 'false' (falsch).

## Funktion 'default-collation'

Die Funktion fn:default-collation gibt eine URI zurück, die die Standardsortierfolge darstellt, die für die Datenbank definiert ist.

### Syntax

### Zurückgegebener Wert

Der zurückgegebene Wert hat den Typ 'xs:anyURI' und gibt die Sortierfolge der Datenbank an.

### Beispiel

Eine DB2-Datenbank wird unter Angabe der Sortierfolge 'UCA400_NO' erstellt. Bei Abfrage dieser Datenbank gibt die folgende Funktion den Wert `http://www.ibm.com/xmlns/prod/db2/sql/collations?name=UCA400_NO` zurück:

```
fn:default-collation()
```

## Funktion 'distinct-values'

Die Funktion fn:distinct-values gibt die unterschiedlichen Werte in einer Sequenz zurück.

### Syntax

*Sequenzausdruck*
Eine Sequenz aus atomaren Werten oder eine leere Sequenz.

### Zurückgegebener Wert

Wenn *Sequenzausdruck* keine leere Sequenz ist, entspricht der zurückgegebene Wert einer Sequenz, die die unterschiedlichen Werte in *Sequenzausdruck* enthält. Zwei Werte (*Wert_1* und *Wert_2*) sind unterschiedlich, wenn der Ausdruck '*Wert_1* eq *Wert_2*' unter Verwendung der Standardsortierfolge den Wert 'false' (falsch) ergibt. Ist für zwei Werte nicht der Operator 'eq' definiert, gelten diese Werte als unterschiedlich.

Werte vom Typ 'xdt:untypedAtomic' werden vor dem Vergleich in Werte vom Typ 'xs:string' umgesetzt.

Für Werte vom Typ 'xs:float' und 'xs:double' wird eine einzelne Nichtzahl (NaN) zurückgegeben, wenn *Sequenzausdruck* mehrere NaN-Werte enthält.

Werte vom Typ 'xs:dateTime', 'xs:date' oder 'xs:time' werden vor dem Vergleich an Zeitzonenunterschiede angepasst. Weist ein Wert keine Zeitzone auf, wird die implizite Zeitzone (UCT) verwendet.

Ist *Sequenzausdruck* eine leere Sequenz, wird die leere Sequenz zurückgegeben.

Wenn zwei Werte in der Eingabesequenz gemäß dem Operator *eq* zwar identisch sind, aber unterschiedliche Typen aufweisen, kann die Ergebnissequenz einen der Werte, nicht jedoch beide enthalten. Die Ergebnissequenz behält unter Umständen nicht die Reihenfolge der Eingabesequenz bei.

### Beispiel

Die folgende Funktion gibt die unterschiedlichen Werte in einer Sequenz nach Atomisierung der in dieser Sequenz enthaltenen Knoten zurück:

```
fn:distinct-values((1, 'a', 1.0, 'A', <greeting>Hello</greeting>))
```

Der zurückgegebene Wert kann entweder (1, 'a', 'A', 'Hello') oder (1.0, 'A', 'a', 'Hello') lauten.

## Funktion 'empty'

Die Funktion fn:empty gibt an, ob das Argument eine leere Sequenz ist.

### Syntax

*Element*
Ein Ausdruck eines beliebigen Datentyps oder eine leere Sequenz.

### Zurückgegebener Wert

Der zurückgegebene Wert ist 'True' (wahr), wenn *Element* eine leere Sequenz ist. Ansonsten ist der zurückgegebene Wert 'false' (falsch).

### Beispiel

In dem folgenden Beispiel wird die Funktion 'empty' verwendet, um zu ermitteln, ob die Sequenz in der Variablen '$seq' eine leere Sequenz ist.

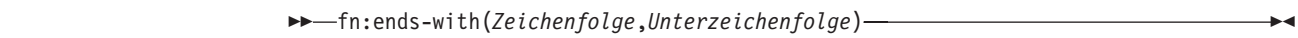
```
let $seq := (5, 10)
return fn:empty($seq)
```

Der zurückgegebene Wert ist 'false' (falsch).

## Funktion 'ends-with'

Die Funktion fn:ends-with ermittelt, ob eine Zeichenfolge mit einer bestimmten Zeichenfolge endet. Die Unterzeichenfolge wird unter Verwendung der Standardsortierfolge abgeglichen.

### Syntax




*Zeichenfolge*
Die Zeichenfolge, in der nach *Unterzeichenfolge* gesucht werden soll. *Zeichenfolge* hat den Datentyp 'xs:string' oder ist eine leere Sequenz. Ist *Zeichenfolge* eine leere Sequenz, wird *Zeichenfolge* auf eine Zeichenfolge mit Nulllänge gesetzt.

*Unterzeichenfolge*
Die Unterzeichenfolge, nach der am Ende von *Zeichenfolge* gesucht werden soll. *Unterzeichenfolge* hat den Datentyp 'xs:string' oder ist eine leere Sequenz.

**Längenbeschränkung**

Die Länge von *Unterzeichenfolge* ist auf 32.000 Byte beschränkt.

### Zurückgegebener Wert

Der zurückgegebene Wert ist der Wert 'true' (wahr) vom Typ 'xs:boolean', wenn eine der folgenden Bedingungen erfüllt ist:

- *Unterzeichenfolge* kommt am Ende von *Zeichenfolge* vor.
- *Unterzeichenfolge* ist eine leere Sequenz oder eine Zeichenfolge mit Nulllänge.

Ansonsten ist der zurückgegebene Wert 'false' (falsch).

### Beispiel

Die folgende Funktion ermittelt, ob die Zeichenfolge 'Test literal' mit der Zeichenfolge 'literal' endet:

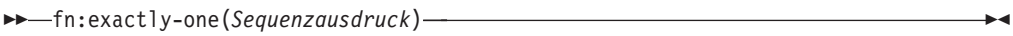
```
fn:ends-with('Test literal','literal')
```

Der zurückgegebene Wert ist 'true' (wahr).

## Funktion 'exactly-one'

Die Funktion fn:exactly-one gibt ihr Argument zurück, wenn es genau ein Element enthält.

### Syntax




*Sequenzausdruck*
Eine beliebige Sequenz, einschließlich einer leeren Sequenz.

### Zurückgegebener Wert

Wenn *Sequenzausdruck* genau ein Element enthält, wird *Sequenzausdruck* zurückgegeben. Andernfalls wird ein Fehler zurückgegeben.

### Beispiel

In dem folgenden Beispiel wird die Funktion 'exactly-one' verwendet, um zu ermitteln, ob die Sequenz in der Variablen '$seq' genau ein Element enthält.

```
let $seq := 5
return fn:exactly-one($seq)
```

Der Wert 5 wird zurückgegeben.

## Funktion 'exists'

Die Funktion fn:exists gibt an, ob eine Sequenz vorhanden und daher keine leere Sequenz ist.

### Syntax

*Sequenzausdruck*
Eine Sequenz eines beliebigen Datentyps oder eine leere Sequenz.

### Zurückgegebener Wert

Der zurückgegebene Wert ist 'true' (wahr), wenn *Sequenzausdruck* keine leere Sequenz ist. Ansonsten ist der zurückgegebene Wert 'false' (falsch).

### Beispiel

In dem folgenden Beispiel wird die Funktion 'exists' verwendet, um zu ermitteln, ob die Sequenz in der Variablen '$seq' eine leere Sequenz ist oder nicht.

```
let $seq := (5, 10)
return fn:exists($seq)
```

Der Wert 'true' (wahr) wird zurückgegeben.

## Funktion 'false'

Die Funktion fn:false gibt den Wert 'false' (falsch) vom Typ 'xs:boolean' zurück.

### Syntax

### Zurückgegebener Wert

Der zurückgegebene Wert der Wert 'false' (falsch) vom Typ xs:boolean.

### Beispiel

Verwenden Sie die Funktion 'false', um den Wert 'false' zurückzugeben.

```
fn:false()
```

Der Wert 'false' wird zurückgegeben.

## Funktion 'floor'

Die Funktion fn:floor gibt die größte ganze Zahl zurück, die kleiner-gleich einem bestimmten numerischen Wert ist.

### Syntax

*numerischer_Wert*
Ein atomarer Wert oder eine leere Sequenz.

Ist *numerischer_Wert* ein atomarer Wert, weist er einen der folgenden Typen auf:
- xs:float
- xs:double
- xs:decimal
- xs:integer
- xdt:untypedAtomic
- Einen Typ, der von einem der vorstehend aufgeführten Typen abgeleitet ist

Hat 'numerischer_Wert' den Typ 'xdt:untypedAtomic', wird er in einen Wert vom Typ 'xs:double' umgesetzt.

### Zurückgegebener Wert

Wenn *numerischer_Wert* keine leere Sequenz ist, ist der zurückgegebene Wert die größte ganze Zahl, die kleiner als *numerischer_Wert* ist. Der Datentyp des zurückgegebenen Wertes hängt vom Datentyp von *numerischer_Wert* ab:
- Hat *numerischer_Wert* den Datentyp 'xs:float', 'xs:double', 'xs:decimal' oder 'xs:integer', weist der zurückgegebene Wert denselben Typ wie *numerischer_Wert* auf.
- Hat *numerischer_Wert* einen Datentyp, der von 'xs:float', 'xs:double', 'xs:decimal' oder 'xs:integer' abgeleitet ist, weist der zurückgegebene Wert den direkten übergeordneten Datentyp von *numerischer_Wert* auf.

Ist *numerischer_Wert* eine leere Sequenz, ist der zurückgegebene Wert die leere Sequenz.

### Beispiele

**Beispiel mit einem positiven Argument:** Die folgende Funktion gibt den floor-Wert von 0.5 zurück:

```
fn:floor(0.5)
```

Der zurückgegebene Wert ist 0.

**Beispiel mit einem negativen Argument:** Die folgende Funktion gibt den floor-Wert von (-1.2) zurück:

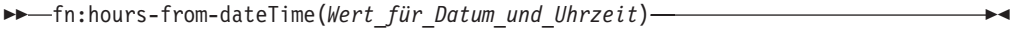
```
fn:floor(-1.2)
```

Der zurückgegebene Wert ist -2.

## Funktion 'hours-from-dateTime'

Die Funktion fn:hours-from-dateTime gibt die Stundenkomponente eines Wertes vom Typ 'xs:dateTime' zurück.

### Syntax




*Wert_für_Datum_und_Uhrzeit*
Der Wert für Datum und Uhrzeit, aus dem die Stundenkomponente extrahiert werden soll.

*Wert_für_Datum_und_Uhrzeit* hat den Datentyp 'xs:dateTime' oder ist eine leere Sequenz.

### Zurückgegebener Wert

Hat *Wert_für_Datum_und_Uhrzeit* den Typ 'xs:dateTime', weist der zurückgegebene Wert den Typ 'xs:integer' auf und liegt im Bereich von 0 bis 23 einschließlich. Der Wert ist die Stundenkomponente von *Wert für Datum und Uhrzeit*.

Ist *Wert_für_Datum_und_Uhrzeit* eine leere Sequenz, ist der zurückgegebene Wert eine leere Sequenz.

### Beispiel

Die folgende Funktion gibt die Stundenkomponente des folgenden Werts für Datum und Uhrzeit zurück: 31. Januar 2005, 14:00 Uhr in Zeitzone UTC-8:

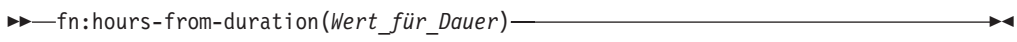
```
fn:hours-from-dateTime(xs:dateTime("2005-01-31T14:00:00-08:00"))
```

Der zurückgegebene Wert lautet '14'.

## Funktion 'hours-from-duration'

Die Funktion fn:hours-from-duration gibt die Stundenkomponente eines Wertes für Dauer zurück.

### Syntax




*Wert_für_Dauer*
Der Wert für Dauer, aus dem die Stundenkomponente extrahiert werden soll.

*Wert_für_Dauer* ist eine leere Sequenz oder ein Wert, der einen der folgenden Datentypen aufweist: xdt:dayTimeDuration, xs:duration oder xdt:yearMonthDuration.

### Zurückgegebener Wert

Der zurückgegebene Wert hängt vom Datentyp von *Wert_für_Dauer* ab:

- Hat *Wert_für_Dauer* den Typ 'xdt:dayTimeDuration' oder 'xs:duration', weist der zurückgegebene Wert den Typ 'xs:integer' auf und liegt im Bereich von -23 bis 23 einschließlich. Der Wert ist die Stundenkomponente von *Wert_für_Dauer*, die als 'xdt:dayTimeDuration' umgesetzt ist. Der Wert ist negativ, wenn *Wert_für_Dauer* negativ ist.
- Hat *Wert_für_Dauer* den Typ 'xdt:yearMonthDuration', weist der zurückgegebene Wert den Typ 'xs:integer' auf und ist 0.
- Ist *Wert_für_Dauer* eine leere Sequenz, ist der zurückgegebene Wert eine leere Sequenz.

Die als 'xdt:dayTimeDuration' umgesetzte Stundenkomponente aus *Wert_für_Dauer* ist die ganzzahlige Anzahl der Stunden, die als `((S mod 86400) idiv 3600)` berechnet ist. Der Wert *S* ist die Gesamtzahl der Sekunden von *Wert_für_Dauer*, die als 'xdt:dayTimeDuration' umgesetzt wird, um die Komponenten der Tage und Monate zu entfernen. Der Wert 86400 ist die Anzahl der Sekunden eines Tages, und 3600 ist die Anzahl der Sekunden einer Stunde.

### Beispiel

Die folgende Funktion gibt die Stundenkomponente der Dauer von 126 Stunden zurück:

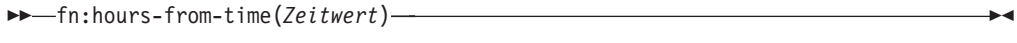
```
fn:hours-from-duration(xdt:dayTimeDuration("PT126H"))
```

Der zurückgegebene Wert ist 6, da bei der Berechnung der Gesamtzahl der Stunden in der Dauer die 126 Stunden in 5 Tage und 6 Stunden umgewandelt werden. Die Dauer entspricht P5DT6H. Hierbei gibt die Stundenkomponente 6 Stunden an.

## Funktion 'hours-from-time'

Die Funktion fn:hours-from-time gibt die Stundenkomponente eines Wertes vom Typ 'xs:time' zurück.

### Syntax




*Zeitwert*
Der Zeitwert, aus dem die Stundenkomponente extrahiert werden soll.

*Zeitwert* hat den Datentyp 'xs:time' oder ist eine leere Sequenz.

### Zurückgegebener Wert

Ist *Zeitwert* keine leere Sequenz, weist der zurückgegebene Wert den Typ 'xs:integer' auf und liegt im Bereich von 0 bis 23 einschließlich. Der Wert ist die Stundenkomponente von *Zeitwert*.

Ist *Zeitwert* eine leere Sequenz, ist der zurückgegebene Wert eine leere Sequenz.

### Beispiel

Die folgende Funktion gibt die Stundenkomponente des folgenden Zeitwerts zurück: 09:30 Uhr in Zeitzone UTC-8:

```
fn:hours-from-time(xs:time("09:30:00-08:00"))
```

Der zurückgegebene Wert ist 9.

## Funktion 'implicit-timezone'

Die Funktion fn:implicit-timezone gibt den impliziten Zeitzonenwert PT0S zurück, der den Datentyp 'xs:dayTimeDuration' aufweist. Der Wert PT0S gibt an, dass es sich bei der impliziten Zeitzone um die Zeitzone UTC handelt.

### Syntax




### Zurückgegebener Wert

Der zurückgegebene Wert ist PT0S, wobei es sich um UTC, dargestellt durch den Datentyp 'xs:dayTimeDuration', handelt.

### Beispiel

Die folgende Funktion gibt 'xdt:dayTimeDuration("PT0S")' zurück:

```
fn:implicit-timezone()
```

## Funktion 'in-scope-prefixes'

Die Funktion fn:in-scope-prefixes gibt eine Liste von Präfixen für alle gültigen Namensbereiche eines Elements zurück.

### Syntax




*Element*
: Das Element, für das die Präfixe für gültige Namensbereiche abgerufen werden sollen.

### Zurückgegebener Wert

Der zurückgegebene Wert ist eine Sequenz aus Werten vom Typ 'xs:NCName', bei denen es sich um die Präfixe für alle gültigen Namensbereiche für *Element* handelt. Wenn für *Element* ein Standardnamensbereich gültig ist, dann ist das Sequenz-

element für das Präfix des Standardnamensbereichs eine Zeichenfolge mit Nulllänge. Der Namensbereich ″xml″ gehört stets zu den gültigen Namensbereichen eines Elements.

### Beispiel

Die folgende Abfrage gibt eine Sequenz aus Präfixen (als NCNames) für gültige Namensbereiche für das Element emp zurück:

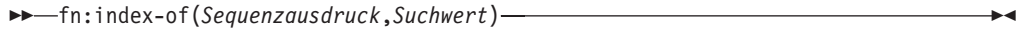
```
declare namespace d="http://www.mycompany.com";
let $department := document {
    <comp:dept xmlns:comp="http://www.mycompany.com" id="A07">
        <comp:emp id="31201" />
    </comp:dept> }
return fn:in-scope-prefixes($department/d:dept/d:emp)
```

Der zurückgegebene Wert lautet (″xml″, ″comp″), jedoch nicht unbedingt in dieser Reihenfolge.

## Funktion 'index-of'

Die Funktion fn:index-of gibt die Positionen zurück, an denen ein Element in einer Sequenz vorkommt.

### Syntax




*Sequenzausdruck*
Eine beliebige Sequenz aus atomaren Typen oder eine leere Sequenz.

*Suchwert*
Der Wert, nach dem in *Sequenzausdruck* gesucht werden soll.

### Zurückgegebener Wert

Der zurückgegebene Wert ist eine Sequenz aus Werten vom Typ 'xs:integer', die die Positionen von Elementen in *Sequenzausdruck* darstellen, die mit *Suchwert* übereinstimmen, wenn der Vergleich unter Verwendung der Regeln des Operators **eq** mit der Standardsortierfolge durchgeführt wird. Elemente, die nicht verglichen werden können, weil der Operator **eq** für ihre jeweiligen Typen nicht definiert ist, gelten als nicht mit *Suchwert* übereinstimmend, sodass die entsprechenden Positionen nicht zurückgegeben werden. Das erste Element in einer Sequenz hat die Position 1.

Die Funktion gibt eine leere Sequenz zurück, wenn *Suchwert* mit keinem Element in *Sequenzausdruck* übereinstimmt oder wenn *Sequenzausdruck* eine leere Sequenz ist.

### Beispiel

Die folgende Funktion gibt die Positionen zurück, an denen 'ABC' in einer Sequenz vorkommt.

```
fn:index-of(('ABC','DEF','ABC','123'), 'ABC')
```

Der zurückgegebene Wert ist die Sequenz (1,3).

# Funktion 'insert-before'

Die Funktion fn:insert-before fügt eine Sequenz vor einer bestimmten Position in eine andere Sequenz ein.

## Syntax

*Quellensequenz*
Die Sequenz, in die eine Sequenz eingefügt werden soll.

*Quellensequenz* ist eine Sequenz aus Elementen eines beliebigen Datentyps oder eine leere Sequenz.

*Einfügeposition*
Die Position in *Quellensequenz*, vor der eine Sequenz eingefügt werden soll. *Einfügeposition* hat den Datentyp 'xs:integer'. Wenn *Einfügeposition* <= 0 ist, wird *Einfügeposition* auf 1 gesetzt. Wenn *Einfügeposition* größer als die Anzahl der Elemente in *Quellensequenz* ist, wird *Einfügeposition* auf einen Wert gesetzt, der um 1 größer ist als die Anzahl der Elemente in *Quellensequenz*.

*Einfügesequenz*
Die Sequenz, die in *Quellensequenz* eingefügt werden soll.

*Einfügesequenz* ist eine Sequenz aus Elementen eines beliebigen Datentyps oder eine leere Sequenz.

## Zurückgegebener Wert

Wenn *Quellensequenz* keine leere Sequenz ist:

- Wenn *Einfügesequenz* keine leere Sequenz ist, ist der zurückgegebene Wert eine Sequenz mit den folgenden Elementen in der nachfolgend angegeben Reihenfolge:
  - Die Elemente in *Quellensequenz* vor dem Element *Einfügeposition*
  - Die Elemente in *Einfügesequenz*
  - Das Element in *Quellensequenz* beim Element *Einfügeposition*
  - Die Elemente in *Quellensequenz* nach dem Element *Einfügeposition*
- Ist *Einfügesequenz* eine leere Sequenz, ist der zurückgegebene Wert *Quellensequenz*.

Wenn *Quellensequenz* eine leere Sequenz ist:

- Ist *Einfügesequenz* keine leere Sequenz, ist der zurückgegebene Wert *Einfügesequenz*.
- Ist *Einfügesequenz* eine leere Sequenz, ist der zurückgegebene Wert die leere Sequenz.

## Beispiel

Die folgende Funktion gibt die Sequenz zurück, die sich ergibt, wenn die Sequenz (4,5,6) vor Position 4 in der Sequenz (1,2,3,7)eingefügt wird:

```
fn:insert-before((1,2,3,7),4,(4,5,6))
```

Der zurückgegebene Wert ist (1,2,3,4,5,6,7).

## Funktion 'last'

Die Funktion fn:last gibt die Anzahl der Werte in der Sequenz zurück, die zum jeweiligen Zeitpunkt verarbeitet wird.

### Syntax

### Zurückgegebener Wert

Wenn die zum jeweiligen Zeitpunkt verarbeitete Sequenz keine leere Sequenz ist, handelt es sich bei dem zurückgegebenen Wert um die Anzahl der Werte in der Sequenz. Wenn die zum jeweiligen Zeitpunkt verarbeitete Sequenz eine leere Sequenz ist, wird die leere Sequenz als Wert zurückgegeben.

### Beispiel

In dem folgenden Beispiel wird die Funktion als Vergleichselementausdruck verwendet, um das letzte Element in der aktuellen Sequenz zurückzugeben:

```
(<a/>, <b/>, <c/>)[fn:last()]
```

Der zurückgegebene Wert lautet <c/>.

## Funktion 'local-name'

Die Funktion fn:local-name gibt den lokalen Namen eines Knotens zurück.

### Syntax

*Knoten* Der Knoten, für den der lokale Name abgerufen werden soll. Wenn *Knoten* nicht angegeben ist, wird die Funktion 'fn:local-name' für den aktuellen Kontextknoten ausgewertet.

### Zurückgegebener Wert

Der zurückgegebene Wert hängt davon ab, ob *Knoten* angegeben ist, und wenn ja, welcher Wert für *Knoten* angegeben ist:

- Ist *Knoten* nicht angegeben, wird der lokale Name des Kontextknotens zurückgegeben.
- Wenn der Wert für *Knoten* eine der folgenden Bedingungen erfüllt, wird eine Zeichenfolge mit Nulllänge zurückgegeben:
  - *Knoten* ist eine leere Sequenz.
  - *Knoten* ist kein Elementknoten, kein Attributknoten und kein Verarbeitungsanweisungsknoten.
- Wenn der Wert für *Knoten* eine der folgenden Bedingungen erfüllt, wird ein Fehler zurückgegeben:
  - *Knoten* ist nicht definiert.
  - *Knoten* ist kein Knoten.

- Andernfalls wird ein Wert vom Typ 'xs:string' zurückgegeben, der den Teil mit dem lokalen Namen des erweiterten Namens für *Knoten* enthält.

### Beispiel

Die folgende Funktion gibt den lokalen Namen für den Knoten 'emp' zurück:

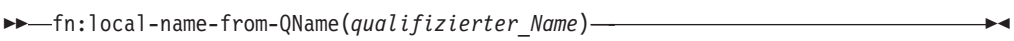
```
declare namespace a="http://posample.org";
fn:local-name(<a:b/>)
```

Der zurückgegebene Wert ist `b`.

## Funktion 'local-name-from-QName'

Die Funktion fn:local-name-from-QName gibt den lokalen Teil eines Wertes vom Datentyp 'xs:QName' zurück.

### Syntax




*qualifizierter_Name*
Der qualifizierte Name, aus dem der lokale Namensteil abgerufen werden soll.

Das Argument *qualifizierter_Name* hat den Datentyp 'xs:QName' oder ist eine leere Sequenz.

### Zurückgegebener Wert

Wenn *qualifizierter_Name* keine leere Zeichenfolge ist, wird ein Wert mit dem Datentyp 'xs:NCName' zurückgegeben, der den lokalen Teil von *qualifizierter_Name* darstellt. Ist *qualifizierter_Name* eine leere Zeichenfolge, wird die leere Zeichenfolge zurückgegeben.

### Beispiel

Die folgende Funktion gibt den lokalen Teil eines qualifizierten Namens zurück:

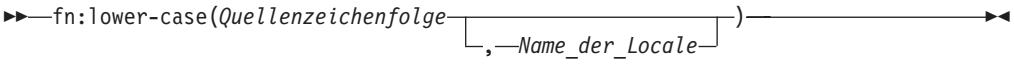
```
fn:local-name-from-QName(fn:QName("http://www.mycompany.com/", "ns:employee"))
```

Der zurückgegebene Wert lautet "employee".

## Funktion 'lower-case'

Die Funktion fn:lower-case setzt eine Zeichenfolge in Kleinbuchstaben um.

### Syntax




*Quellenzeichenfolge*
Die Zeichenfolge, die in Kleinbuchstaben konvertiert werden soll.

*Quellenzeichenfolge* hat den Datentyp 'xs:string' oder ist eine leere Sequenz.

*Name_der_Locale*
Eine Zeichenfolge mit den Ländereinstellungen (Locale), die für die Operation in Kleinbuchstaben verwendet werden soll.

*Name_der_Locale* hat den Datentyp 'xs:string' oder ist eine leere Sequenz. Ist *Name_der_Locale* keine leere Sequenz, ist der Wert von *Name_der_Locale* nicht von der Groß-/Kleinschreibung abhängig und muss eine gültige Locale oder eine Zeichenfolge mit Nulllänge sein.

### Zurückgegebener Wert

Ist *Quellenzeichenfolge* keine leere Zeichenfolge, dann ist der zurückgegebene Wert *Quellenzeichenfolge*, wobei jedes Zeichen in den ihm entsprechenden Kleinbuchstaben konvertiert ist. Wird für *Name_der_Locale* kein Wert angegeben oder ist der Wert eine leere Sequenz oder eine Zeichenfolge mit Nulllänge, dann werden die Regeln für Kleinschreibung gemäß Unicode-Standard verwendet. Ansonsten werden die Regeln für Kleinschreibung für die angegebene Locale verwendet. Jedes Zeichen, das über keine Entsprechung in Kleinschreibung verfügt, wird in seiner ursprünglichen Form in den zurückgegebenen Wert eingeschlossen.

Ist *Quellenzeichenfolge* eine leere Sequenz, ist der zurückgegebene Wert eine Zeichenfolge mit Nulllänge.

### Beispiele

Die folgende Funktion konvertiert die Zeichenfolge ″Wireless Router TB2561″ in Kleinschreibung:

```
fn:lower-case("Wireless Router TB2561")
```

Der zurückgegebene Wert lautet ″wireless router tb2561″.

In der folgenden Funktion wird die türkische Locale (tr_TR) angegeben, und der Buchstabe ″I″ und der numerische Zeichenverweis İ (der Zeichenverweis für das lateinische große I mit Punkt) werden konvertiert:

```
fn:lower-case("I&amp;#x130;", "tr_TR")
```

Der zurückgegebene Wert besteht aus zwei Zeichen - dem von ı dargestellten Zeichen (lateinisches kleines i ohne Punkt) und dem Buchstaben ″i″. Bei der türkischen Locale wird der Buchstabe ″I″ in das von ı dargestellte Zeichen (lateinisches kleines i ohne Punkt) konvertiert, und İ (lateinisches großes I mit Punkt) wird in ″i″ konvertiert.

In der folgenden Funktion wird keine Locale angegeben, und der Buchstabe ″I″ wird anhand der im Unicode-Standard definierten Regeln in die Kleinschreibung konvertiert:

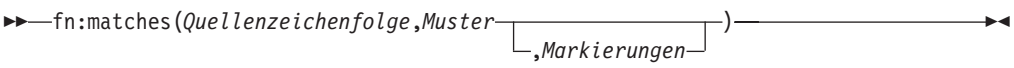
```
fn:lower-case("I")
```

Der zurückgegebene Wert ist der Buchstabe ″i″.

## Funktion 'matches'

Die Funktion fn:matches ermittelt, ob eine Zeichenfolge mit einem bestimmten Muster übereinstimmt.

## Syntax




*Quellenzeichenfolge*
Eine Zeichenfolge, die mit einem Muster verglichen wird.

Der Wert von *Quellenzeichenfolge* hat den Datentyp 'xs:string' oder ist eine leere Sequenz.

*Muster* Ein regulärer Ausdruck, der mit *Quellenzeichenfolge* verglichen wird. Ein regulärer Ausdruck ist eine Gruppe von Zeichen, Platzhalterzeichen und Operatoren, die eine Zeichenfolge oder Gruppe von Zeichenfolgen in einem Suchmuster definieren.

*Muster* ist ein Wert vom Datentyp 'xs:string'.

*Markierungen*
Ein Wert vom Typ 'xs:string', der einen der folgenden Werte enthalten kann, die den Abgleich von *Muster* mit *Quellenzeichenfolge* steuern:

**s** Gibt an, dass der Punkt (.) einem beliebigen Zeichen entspricht.

Wenn die Markierung 's' nicht angegeben ist, entspricht der Punkt jedem Zeichen mit Ausnahme des Zeilenvorschubzeichens (X'0A').

**m** Gibt an, dass das Winkelzeichen (^) dem Beginn einer Zeile entspricht (der Position nach einem Zeilenvorschubzeichen), und dass das Dollarzeichen ($) dem Ende einer Zeile entspricht (der Position vor einem neuen Zeilenvorschubzeichen).

Wenn die Markierung 'm' nicht angegeben ist, entspricht das Winkelzeichen (^) dem Beginn einer Zeichenfolge, und das Dollarzeichen ($) entspricht dem Ende der Zeichenfolge.

**i** Gibt an, dass der Abgleich nicht von der Groß-/Kleinschreibung abhängt.

Wenn die Markierung 'i' nicht angegeben ist, erfolgt der Abgleich unter Beachtung der Groß-/Kleinschreibung.

**x** Gibt an, dass Leerzeichen innerhalb von *Muster* ignoriert werden.

Wenn die Markierung 'x' nicht angegeben ist, werden Leerzeichen beim Abgleich verwendet.

## Zurückgegebener Wert

Wenn *Quellenzeichenfolge* keine leere Zeichenfolge ist, ist der zurückgegebene Wert 'true' (wahr), wenn *Quellenzeichenfolge* mit *Muster* übereinstimmt. Der zurückgegebene Wert ist 'false' (falsch), wenn *Quellenzeichenfolge* nicht mit *Muster* übereinstimmt.

Wenn *Muster* kein Winkelzeichen (^) als Startzeichen einer Zeichenfolge oder Zeile bzw. kein Dollarzeichen ($) als Endezeichen einer Zeichenfolge oder Zeile enthält, gilt *Quellenzeichenfolge* dann als mit *Muster* übereinstimmend, wenn eine beliebige Unterzeichenfolge von *Quellenzeichenfolge* mit *Muster* übereinstimmt. Wenn *Muster* das Winkelzeichen (^) als Startzeichen einer Zeichenfolge oder Zeile enthält, gilt *Quellenzeichenfolge* nur dann als mit *Muster* übereinstimmend, wenn *Quellenzeichenfolge* ab dem Anfang von *Quellenzeichenfolge* bzw. ab dem Anfang einer Zeile in

*Quellenzeichenfolge* mit *Muster* übereinstimmt. Wenn *Muster* das Dollarzeichen ($) als Endezeichen einer Zeichenfolge oder Zeile enthält, gilt *Quellenzeichenfolge* nur dann als mit *Muster* übereinstimmend, wenn *Quellenzeichenfolge* am Ende von *Quellenzeichenfolge* bzw. am Ende einer Zeile in *Quellenzeichenfolge* mit *Muster* übereinstimmt. Die Markierung 'm' bestimmt, ob der Abgleich ab dem Anfang der Zeichenfolge oder dem Anfang einer Zeile erfolgt.

Wenn *Quellenzeichenfolge* eine leere Sequenz ist, ist der zurückgegebene Wert 'false' (falsch).

### Beispiele

**Beispiel des Abgleichs eines Musters mit einer beliebigen Unterzeichenfolge innerhalb einer Zeichenfolge:** Die folgende Funktion ermittelt, ob die Zeichen ″ac″ oder ″bd″ an einer beliebigen Stelle innerhalb der Zeichenfolge ″abbcacadbdcd″ vorkommen:

```
fn:matches("abbcacadbdcd","(ac)|(bd)")
```

Der zurückgegebene Wert ist 'true' (wahr).

**Beispiel des Abgleichs eines Musters mit der gesamten Zeichenfolge:** Die folgende Funktion bestimmt, ob die Zeichen ″ac″ oder ″bd″ mit der Zeichenfolge ″bd″ übereinstimmen:

```
fn:matches("bd","^(ac)|(bd)$")
```

Der zurückgegebene Wert ist 'true' (wahr).

**Beispiel eines Musterabgleichs, bei dem Leerzeichen und Großschreibung ignoriert werden:** In der folgenden Funktion werden die Markierungen i und x verwendet, damit die Großschreibung und Leerzeichen beim Ermitteln, ob die Zeichenfolge ″abc1234″ mit dem Muster ″ABC 1234″ übereinstimmt, ignoriert werden:

```
fn:matches("abc1234","ABC 1234", "ix")
```

Der zurückgegebene Wert ist 'true' (wahr).

## Funktion 'max'

Die Funktion fn:max gibt das Maximum der Werte in einer Sequenz zurück.

### Syntax

*Sequenzausdruck*
Eine Sequenz mit Elementen, die einen der folgenden atomaren Typen aufweisen, oder eine leere Sequenz.
- xs:float
- xs:double
- xs:decimal
- xs:integer
- xs:string
- xs:date
- xs:time
- xs:dateTime

- xdt:untypedAtomic
- xdt:dayTimeDuration
- xdt:yearMonthDuration
- Einen Typ, der von einem der vorstehend aufgeführten Typen abgeleitet ist

Eingabeelemente vom Typ 'xdt:untypedAtomic' werden in den Typ 'xs:double' umgesetzt. Nach dieser expliziten Typumsetzung müssen alle Elemente in der Eingabesequenz durch Umstufung oder Subtypsubstitution in einen gemeinsamen Typ konvertierbar sein, der den Operator **ge** unterstützt. Der Maximalwert wird in diesem gemeinsamen Typ berechnet. Beispiel: Enthält die Eingabesequenz Elemente vom Typ 'money' (Geld), der von 'xs:decimal' abgeleitet ist, und vom Typ 'stockprice' (Aktienkurs), der von 'xs:float' abgeleitet ist, wird das Maximum im Typ 'xs:float' berechnet.

Vor dem Vergleich von Werten für Datum (date), Uhrzeit (time) oder Datum und Uhrzeit (dateTime) werden diese an eine einheitliche Zeitzone angepasst. Bei Werten für Datum und Uhrzeit ohne ein explizites Zeitzonenelement wird die implizite Zeitzone (UTC) verwendet.

Zeichenfolgewerte werden unter Verwendung der Standardsortierfolge verglichen.

### Zurückgegebener Wert

Wenn *Sequenzausdruck* keine leere Sequenz ist, entspricht der zurückgegebene Wert dem Maximum der Werte in *Sequenzausdruck*. Der Datentyp des zurückgegebenen Werts entspricht dem Datentyp der Elemente in *Sequenzausdruck* oder dem gemeinsamen Datentyp, in den die Elemente in *Sequenzausdruck* umgestuft werden.

Ist *Sequenzausdruck* eine leere Sequenz, wird die leere Sequenz zurückgegeben. Wenn die Sequenz eine Nichtzahl (NaN) enthält, wird NaN zurückgegeben.

### Beispiel

Die folgende Funktion gibt das Maximum der Sequenz (500, 1.0E2, 40.5) zurück:

```
fn:max((500, 1.0E2, 40.5))
```

Die Werte werden in den Datentyp 'xs:double' umgestuft. Die Funktion gibt den Wert 5.0E2 vom Typ 'xs:double' zurück, der als ″500″ serialisiert wird.

## Funktion 'min'

Die Funktion fn:min gibt das Minimum der Werte in einer Sequenz zurück.

### Syntax

*Sequenzausdruck*
Eine Sequenz mit Elementen, die einen der folgenden atomaren Typen aufweisen, oder eine leere Sequenz.
- xs:float
- xs:double
- xs:decimal
- xs:integer

- xs:string
- xs:date
- xs:time
- xs:dateTime
- xdt:untypedAtomic
- xdt:dayTimeDuration
- xdt:yearMonthDuration
- Einen Typ, der von einem der vorstehend aufgeführten Typen abgeleitet ist

Eingabeelemente vom Typ 'xdt:untypedAtomic' werden in den Typ 'xs:double' umgesetzt. Nach dieser expliziten Typumsetzung müssen alle Elemente in der Eingabesequenz durch Umstufung oder Subtypsubstitution in einen gemeinsamen Typ konvertierbar sein, der den Operator **le** unterstützt. Der Mindestwert wird in diesem gemeinsamen Typ berechnet. Beispiel: Enthält die Eingabesequenz Elemente vom Typ 'money' (Geld), der von 'xs:decimal' abgeleitet ist, und vom Typ 'stockprice' (Aktienkurs), der von 'xs:float' abgeleitet ist, wird das Minimum im Typ 'xs:float' berechnet.

Vor dem Vergleich von Werten für Datum (date), Uhrzeit (time) oder Datum und Uhrzeit (dateTime) werden diese an eine einheitliche Zeitzone angepasst. Bei Werten für Datum und Uhrzeit ohne ein explizites Zeitzonenelement wird die implizite Zeitzone (UTC) verwendet.

Zeichenfolgewerte werden unter Verwendung der Standardsortierfolge verglichen.

## Zurückgegebener Wert

Wenn *Sequenzausdruck* keine leere Sequenz ist, entspricht der zurückgegebene Wert dem Minimum der Werte in *Sequenzausdruck*. Der Datentyp des zurückgegebenen Werts entspricht dem Datentyp der Elemente in *Sequenzausdruck* oder dem gemeinsamen Datentyp, in den die Elemente in *Sequenzausdruck* umgestuft werden.

Ist *Sequenzausdruck* eine leere Sequenz, wird die leere Sequenz zurückgegeben. Wenn die Sequenz eine Nichtzahl (NaN) enthält, wird NaN zurückgegeben.

## Beispiele

**Beispiel mit numerischen Argumenten:** Die folgende Funktion gibt das Minimum der Sequenz (500, 1.0E2, 40.5) zurück:

```
fn:min((500, 1.0E2, 40.5))
```

Die Werte werden in den Datentyp 'xs:double' umgestuft. Die Funktion gibt den Wert 4.05E1 vom Typ 'xs:double' zurück, der als ″40.5″ serialisiert wird.

**Beispiel mit Zeichenfolgeargumenten:** Die nachstehende Funktion gibt das Minimum der Sequenz (″x″, ″y″, ″Z″) zurück, wobei die Standardsortierfolge verwendet wird. Hierbei gilt die Annahme, dass die Standardsortierfolge alphabetische Zeichen in Kleinschreibung vor alphabetischen Zeichen in Großschreibung sortiert:

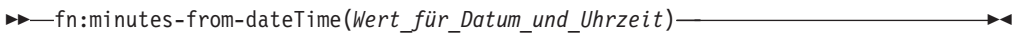
```
fn:min(("x", "y", "Z"))
```

Der zurückgegebene Wert lautet ″x″.

## Funktion 'minutes-from-dateTime'

Die Funktion fn:minutes-from-dateTime gibt die Minutenkomponente eines Wertes vom Typ 'xs:dateTime' zurück.

### Syntax




*Wert_für_Datum_und_Uhrzeit*
Der Wert für Datum und Uhrzeit, aus dem die Minutenkomponente extrahiert werden soll.

*Wert_für_Datum_und_Uhrzeit* hat den Datentyp 'xs:dateTime' oder ist eine leere Sequenz.

### Zurückgegebener Wert

Hat *Wert_für_Datum_und_Uhrzeit* den Typ 'xs:dateTime', weist der zurückgegebene Wert den Typ 'xs:integer' auf und liegt im Bereich von 0 bis 59 einschließlich. Der Wert ist die Minutenkomponente von *Wert für Datum und Uhrzeit*.

Ist *Wert_für_Datum_und_Uhrzeit* eine leere Sequenz, ist der zurückgegebene Wert eine leere Sequenz.

### Beispiel

Die folgende Funktion gibt die Minutenkomponente des folgenden Werts für Datum und Uhrzeit zurück: 23. Januar 2005, 09:42 Uhr in Zeitzone UTC-8:

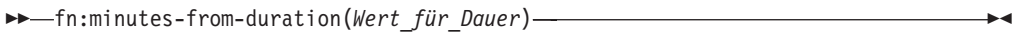
```
fn:minutes-from-dateTime(xs:dateTime("2005-01-23T09:42:00-08:00"))
```

Der zurückgegebene Wert lautet '42'.

## Funktion 'minutes-from-duration'

Die Funktion fn:minutes-from-duration gibt die Minutenkomponente einer Zeitdauer zurück.

### Syntax




*Wert_für_Dauer*
Der Wert für Dauer, aus dem die Minutenkomponente extrahiert werden soll.

*Wert_für_Dauer* ist eine leere Sequenz oder ein Wert, der einen der folgenden Datentypen aufweist: xdt:dayTimeDuration, xs:duration oder xdt:yearMonthDuration.

### Zurückgegebener Wert

Der zurückgegebene Wert hängt vom Datentyp von *Wert_für_Dauer* ab:

- Hat *Wert_für_Dauer* den Typ 'xdt:dayTimeDuration' oder 'xs:duration', weist der zurückgegebene Wert den Typ 'xs:integer' auf und liegt im Bereich von -59 bis

59 einschließlich. Der Wert ist die Minutenkomponente von *Wert_für_Dauer*, die als 'xdt:dayTimeDuration' umgesetzt ist. Der Wert ist negativ, wenn *Wert_für_Dauer* negativ ist.

- Hat *Wert_für_Dauer* den Datentyp 'xdt:yearMonthDuration', ist der zurückgegebene Wert 0.
- Ist *Wert_für_Dauer* eine leere Sequenz, ist der zurückgegebene Wert eine leere Sequenz.

Die als 'xdt:dayTimeDuration' umgesetzte Minutenkomponente aus *Wert_für_Dauer* ist die ganzzahlige Anzahl der Minuten, die als `((S mod 3600) idiv 60)` berechnet ist. Der Wert *S* ist die Gesamtzahl der Sekunden von *Wert_für_Dauer*, die als 'xdt:dayTimeDuration' umgesetzt wird, um die Komponenten der Jahre und Monate zu entfernen.

### Beispiel

Die folgende Funktion gibt die Minutenkomponente der Dauer von 2 Tagen, 16 Stunden und 93 Minuten zurück:

```
fn:minutes-from-duration(xdt:dayTimeDuration("P2DT16H93M"))
```

Der zurückgegebene Wert ist 33. Bei der Berechnung der Gesamtzahl der Minuten in der Dauer werden die 93 Minuten in 1 Stunde und 33 Minuten umgewandelt. Die Dauer entspricht P2DT17H33M. Hierbei gibt die Minutenkomponente 33 Minuten an.

## Funktion 'minutes-from-time'

Die Funktion fn:minutes-from-time gibt die Minutenkomponente eines Wertes vom Typ 'xs:time' zurück.

### Syntax

*Zeitwert*
Der Zeitwert, aus dem die Minutenkomponente extrahiert werden soll.

*Zeitwert* hat den Datentyp 'xs:time' oder ist eine leere Sequenz.

### Zurückgegebener Wert

Hat *Zeitwert* den Typ 'xs:time', weist der zurückgegebene Wert den Typ 'xs:integer' auf und liegt im Bereich von 0 bis 59 einschließlich. Der Wert ist die Minutenkomponente von *Zeitwert*.

Ist *Zeitwert* eine leere Sequenz, ist der zurückgegebene Wert eine leere Sequenz.

### Beispiel

Die folgende Funktion gibt die Minutenkomponente des folgenden Zeitwerts zurück: 08:59 Uhr in Zeitzone UTC-8:

```
fn:minutes-from-time(xs:time("08:59:00-08:00"))
```

Der zurückgegebene Wert ist 59.

## Funktion 'month-from-date'

Die Funktion fn:month-from-date gibt die Monatskomponente eines Wertes vom Typ 'xs:date' zurück.

### Syntax

*Datumswert*
Der Datumswert, aus dem die Monatskomponente extrahiert werden soll.

*Datumswert* hat den Datentyp 'xs:date' oder ist eine leere Sequenz.

### Zurückgegebener Wert

Hat *Datumswert* den Typ 'xs:date', weist der zurückgegebene Wert den Typ 'xs:integer' auf und liegt im Bereich von 1 bis 12 einschließlich. Der Wert ist die Monatskomponente in *Datumswert*.

Ist *Datumswert* eine leere Sequenz, ist der zurückgegebene Wert eine leere Sequenz.

### Beispiel

Die folgende Funktion gibt die Monatskomponente des folgenden Datumswerts zurück: 01. Dezember 2005:

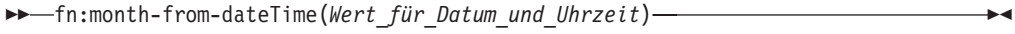
```
fn:month-from-date(xs:date("2005-12-01"))
```

Der zurückgegebene Wert lautet 12.

## Funktion 'month-from-dateTime'

Die Funktion fn:month-from-dateTime gibt die Monatskomponente eines Wertes vom Typ 'xs:dateTime' zurück.

### Syntax




*Wert_für_Datum_und_Uhrzeit*
Der Wert für Datum und Uhrzeit, aus dem die Monatskomponente extrahiert werden soll.

*Wert_für_Datum_und_Uhrzeit* hat den Datentyp 'xs:dateTime' oder ist eine leere Sequenz.

### Zurückgegebener Wert

Hat *Wert_für_Datum_und_Uhrzeit* den Typ 'xs:dateTime', weist der zurückgegebene Wert den Typ 'xs:integer' auf und liegt im Bereich von 1 bis 12 einschließlich. Der Wert ist die Monatskomponente aus *Wert für Datum und Uhrzeit*.

Ist *Wert_für_Datum_und_Uhrzeit* eine leere Sequenz, ist der zurückgegebene Wert eine leere Sequenz.

### Beispiel

Die folgende Funktion gibt die Monatskomponente des folgenden Werts für Datum und Uhrzeit zurück: 31. Oktober 2005, 08:15 Uhr in Zeitzone UTC-8:

```
fn:month-from-dateTime(xs:dateTime("2005-10-31T08:15:00-08:00"))
```

Der zurückgegebene Wert ist 10.

## Funktion 'months-from-duration'

Die Funktion fn:months-from-duration gibt die Monatskomponente eines Wertes für Dauer zurück.

### Syntax

*Wert_für_Dauer*
Der Wert für Dauer, aus dem die Monatskomponente extrahiert werden soll.

*Wert_für_Dauer* ist eine leere Sequenz oder ein Wert, der einen der folgenden Datentypen aufweist: xdt:dayTimeDuration, xs:duration oder xdt:yearMonthDuration.

### Zurückgegebener Wert

Der zurückgegebene Wert hängt vom Datentyp von *Wert_für_Dauer* ab:

- Hat *Wert_für_Dauer* den Typ 'xs:duration' oder 'xdt:yearMonthDuration', weist der zurückgegebene Wert den Typ 'xs:integer' auf und liegt im Bereich von -11 bis 11 einschließlich. Der Wert ist die Monatskomponente aus *Wert_für_Dauer*, die als 'xdt:yearMonthDuration' umgesetzt ist. Der Wert ist negativ, wenn *Wert_für_Dauer* negativ ist.
- Hat *Wert_für_Dauer* den Datentyp 'xdt:dayTimeDuration', ist der zurückgegebene Wert 0.
- Ist *Wert_für_Dauer* eine leere Sequenz, ist der zurückgegebene Wert eine leere Sequenz.

Die als 'xdt:yearMonthDuration' umgesetzte Monatskomponente aus *Wert_für_Dauer* ist die ganzzahlige Anzahl der Restmonate, berechnet aus der Gesamtzahl der Monate von *Wert_für_Dauer* dividiert durch 12.

### Beispiele

Die folgende Funktion gibt die Monatskomponente der Dauer von 20 Jahren und 5 Monaten (Typ 'xs:duration') zurück:

```
fn:months-from-duration(xs:duration("P20Y5M"))
```

Der zurückgegebene Wert ist 5.

Die folgende Funktion gibt die Monatskomponente der Dauer von -9 Jahren und -13 Monaten (Typ 'xdt:yearMonthDuration') zurück:

```
fn:months-from-duration(xdt:yearMonthDuration("-P9Y13M"))
```

Der zurückgegebene Wert ist -1. Bei der Berechnung der Gesamtzahl der Monate in der Dauer werden die -13 Monate in -1 Jahr und -1 Monat umgewandelt. Die Dauer entspricht -P10Y1M. Hierbei gibt die Monatskomponente -1 Monat an.

Die folgende Funktion gibt die Monatskomponente der Dauer von 14 Jahren, 11 Monaten, 40 Tagen und 13 Stunden (Typ 'xs:duration') zurück:

```
xquery fn:months-from-duration(xs:duration("P14Y11M40DT13H"))
```

Der zurückgegebene Wert ist 11.

# Funktion 'name'

Die Funktion fn:name gibt das Präfix und den lokalen Namensteil eines Knotennamens zurück.

## Syntax

*Knoten* Der qualifizierte Name eines Knotens, für den der Name abgerufen werden soll. Wenn *Knoten* nicht angegeben ist, wird die Funktion 'fn:name' für den aktuellen Kontextknoten ausgewertet.

## Zurückgegebener Wert

Der zurückgegebene Wert hängt vom Wert für *Knoten* ab:

- Wenn der Wert für *Knoten* eine der folgenden Bedingungen erfüllt, wird eine Zeichenfolge mit Nulllänge zurückgegeben:
  - *Knoten* ist eine leere Sequenz.
  - *Knoten* ist kein Elementknoten, kein Attributknoten und kein Verarbeitungsanweisungsknoten.
- Wenn der Wert für *Knoten* eine der folgenden Bedingungen erfüllt, wird ein Fehler zurückgegeben:
  - *Knoten* ist nicht definiert.
  - *Knoten* ist kein Knoten.
- Andernfalls wird ein Wert vom Typ 'xs:string' zurückgegeben, der das Präfix (sofern vorhanden) und den lokalen Namen für *Knoten* enthält.

## Beispiele

Die folgende Abfrage gibt den Wert ″comp:emp″ zurück:

```
declare namespace d="http://www.mycompany.com";
let $department := document {
    <comp:dept xmlns:comp="http://www.mycompany.com" id="A07">
        <comp:emp id="31201" />
    </comp:dept> }
return fn:name($department/d:dept/d:emp)
```

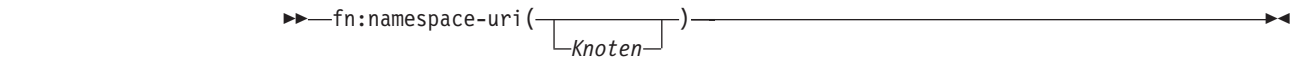
Die folgende Abfrage gibt ebenfalls den Wert ″comp:emp″ zurück:

```
declare namespace d="http://www.mycompany.com";
let $department := document {
    <comp:dept xmlns:comp="http://www.mycompany.com" id="A07">
        <comp:emp id="31201" />
    </comp:dept> }
return $department/d:dept/d:emp/fn:name()
```

## Funktion 'namespace-uri'

Die Funktion fn:namespace-uri gibt die Namensbereichs-URI des qualifizierten Namens eines Knotens zurück.

### Syntax




*Knoten* Der qualifizierte Name eines Knotens, für den die Namensbereichs-URI abgerufen werden soll. Wenn *Knoten* nicht angegeben ist, wird die Funktion 'fn:namespace-uri' für den aktuellen Kontextknoten ausgewertet.

### Zurückgegebener Wert

Der zurückgegebene Wert hängt vom Wert für *Knoten* ab:

- Wenn der Wert für *Knoten* eine der folgenden Bedingungen erfüllt, wird eine Zeichenfolge mit Nulllänge zurückgegeben:
  - *Knoten* ist eine leere Sequenz.
  - *Knoten* ist kein Elementknoten und kein Attributknoten.
  - *Knoten* ist ein Elementknoten oder ein Attributknoten, aber der erweiterte qualifizierte Name für *Knoten* befindet sich nicht in einem Namensbereich.
- Wenn der Wert für *Knoten* eine der folgenden Bedingungen erfüllt, wird ein Fehler zurückgegeben:
  - *Knoten* ist nicht definiert.
  - *Knoten* ist kein Knoten.
- Andernfalls wird ein Wert vom Typ 'xs:string' zurückgegeben, der die Namensbereichs-URI des erweiterten Namens für *Knoten* enthält.

### Beispiele

Die folgende Abfrage gibt den Wert "http://www.mycompany.com" zurück:

```
declare namespace d="http://www.mycompany.com";
let $department := document {
    <comp:dept xmlns:comp="http://www.mycompany.com" id="A07">
        <comp:emp id="31201" />
    </comp:dept> }
return fn:namespace-uri($department/d:dept/d:emp)
```

Die folgende Abfrage gibt ebenfalls den Wert "http://www.mycompany.com" zurück:

```
declare namespace d="http://www.mycompany.com";
let $department := document {
    <comp:dept xmlns:comp="http://www.mycompany.com" id="A07">
        <comp:emp id="31201" />
    </comp:dept> }
return $department/d:dept/d:emp/fn:namespace-uri()
```

## Funktion 'namespace-uri-for-prefix'

Die Funktion fn:namespace-uri-for-prefix gibt die Namensbereichs-URI zurück, die einem Präfix in den gültigen Namensbereichen eines Elements zugeordnet ist.

### Syntax

*Präfix* Das Präfix, für das der Namensbereich zurückgegeben wird.

*Präfix* hat den Datentyp 'xs:string' mit möglicher Nulllänge oder ist eine leere Sequenz.

*Element*
Ein Element mit einem gültigen Namensbereich, der an *Präfix* gebunden ist.

### Zurückgegebener Wert

Der zurückgegebene Wert hängt vom Wert für *Präfix* ab:

- Wenn *Element* einen gültigen Namensbereich aufweist, dessen Präfixwert mit dem Wert von *Präfix* übereinstimmt, wird die Namensbereichs-URI für den betreffenden Namensbereich zurückgegeben.
- Wenn *Element* keinen gültigen Namensbereich aufweist, dessen Präfixwert mit dem Wert von *Präfix* übereinstimmt, wird eine leere Sequenz zurückgegeben.
- Wenn *Präfix* eine Zeichenfolge mit Nulllänge oder eine leere Sequenz ist, wird die Namensbereichs-URI für den Standardnamensbereich zurückgegeben.

### Beispiel

Die folgende Abfrage gibt den Wert "http://www.mycompany.com" zurück:

```
declare namespace d="http://www.mycompany.com";
let $department := document {
    <comp:dept xmlns:comp="http://www.mycompany.com" id="A07">
        <comp:emp id="31201" />
    </comp:dept> }
return fn:namespace-uri-for-prefix("comp", $department/d:dept/d:emp)
```

## Funktion 'namespace-uri-from-QName'

Die Funktion fn:namespace-uri-from-QName gibt den Teil mit der Namensbereichs-URI eines Wertes vom Typ 'xs:QName' zurück.

### Syntax

*qualifizierter_Name*
Der qualifizierte Name, aus dem der Teil mit der Namensbereichs-URI abgerufen werden soll.

Das Argument *qualifizierter_Name* hat den Datentyp 'xs:QName' oder ist eine leere Sequenz.

### Zurückgegebener Wert

Wenn *qualifizierter_Name* keine leere Zeichenfolge ist, wird ein Wert mit dem Datentyp 'xs:string' zurückgegeben, der den Teil mit der Namensbereichs-URI von *qualifizierter_Name* darstellt. Wenn *qualifizierter_Name* sich in keinem Namensbereich befindet, wird eine Zeichenfolge mit Nulllänge zurückgegeben. Ist *qualifizierter-_Name* eine leere Zeichenfolge, wird die leere Zeichenfolge zurückgegeben.

### Beispiel

Die folgende Funktion gibt den Zeichenfolgewert "http://www.mycompany.com" zurück:

```
fn:namespace-uri-from-QName(fn:QName("http://www.mycompany.com", "comp:employee"))
```

## Funktion 'node-name'

Die Funktion fn:node-name gibt den erweiterten qualifizierten Namen (QName) eines Knotens zurück.

### Syntax

*Knoten* Der Knoten, für den der erweiterte Name abgerufen werden soll.

### Zurückgegebener Wert

Der zurückgegebene Wert hat den Typ 'xs:QName' und enthält den erweiterten qualifizierten Namen für *Knoten*. Ist *Knoten* eine leere Zeichenfolge, wird die leere Zeichenfolge zurückgegeben.

### Beispiel

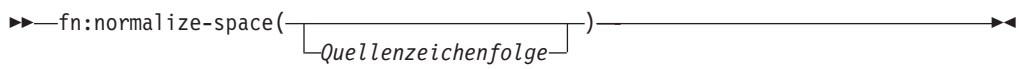
Die folgende Abfrage gibt den erweiterten qualifizierten Namen, der mit der URI 'http://www.mycompany.com' übereinstimmt, sowie den lexikalischen qualifizierten Namen 'comp:emp' zurück:

```
declare namespace d="http://www.mycompany.com";
let $department := document {
    <comp:dept xmlns:comp="http://www.mycompany.com" id="A07">
        <comp:emp id="31201" />
    </comp:dept> }
return fn:node-name($department/d:dept/d:emp)
```

## Funktion 'normalize-space'

Die Funktion fn:normalize-space entfernt führende und abschließende Leerzeichen aus einer Zeichenfolge und ersetzt jede interne Sequenz aus Leerzeichen durch ein einzelnes Leerzeichen.

### Syntax



*Quellenzeichenfolge*
Eine Zeichenfolge, in der Leerzeichen normalisiert werden sollen.

*Quellenzeichenfolge* ist ein Wert vom Typ 'xs:string' oder eine leere Sequenz.

Wird *Quellenzeichenfolge* nicht angegeben, wird als Argument von 'fn:normalize-space' das aktuelle Kontextelement verwendet, das mithilfe der Funktion 'fn:string' in einen Wert vom Typ 'xs:string' konvertiert wird.

### Zurückgegebener Wert

Der zurückgegebene Wert ist der Wert vom Typ 'xs:string', der sich ergibt, wenn die folgenden Operationen für *Quellenzeichenfolge* ausgeführt werden:

- Führende und abschließende Leerzeichen werden entfernt.
- Jede interne Sequenz aus einem Zeichen oder mehreren benachbarten Zeichen der Leerzeichenklasse wird durch ein einzelnes Leerzeichen (X'20') ersetzt.

Die Leerzeichenklasse umfasst Leerzeichen (X'20'), Tabulatorzeichen (X'09'), Zeilenvorschubzeichen (X'0A') und Rücklaufzeichen (X'0D').

Wenn *Quellenzeichenfolge* eine leere Sequenz ist, wird eine Zeichenfolge mit Nulllänge zurückgegeben.

### Beispiel

Die folgende Funktion entfernt überflüssige Leerzeichen aus der Zeichenfolge `"a b c d "`.

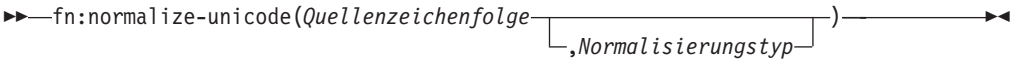
```
fn:normalize-space(" a b c d " )
```

Der zurückgegebene Wert lautet "a b c d".

## Funktion 'normalize-unicode'

Die Funktion fn:normalize-unicode führt eine Unicode-Normalisierung für eine Zeichenfolge aus.

### Syntax




*Quellenzeichenfolge*
Ein Wert, für den die Unicode-Normalisierung ausgeführt werden soll.

*Quellenzeichenfolge* ist ein Wert vom Typ 'xs:string' oder eine leere Sequenz.

*Normalisierungstyp*
Ein Wert vom Typ 'xs:string', der den Typ der Unicode-Normalisierung angibt, die ausgeführt werden soll. Mögliche Werte:

**NFC** Unicode Normalization Form C. Wird für *Normalisierungstyp* kein Wert angegeben, wird die NFC-Normalisierung ausgeführt.

**NFD** Unicode Normalization Form D.

**NFKC** Unicode Normalization Form KC.

**NFKD** Unicode Normalization Form KD.

Wird eine Zeichenfolge mit Nulllänge angegeben, wird keine Normalisierung ausgeführt.

### Zurückgegebener Wert

Wenn *Quellenzeichenfolge* keine leere Sequenz ist, ist der zurückgegebene Wert der Wert vom Typ 'xs:string', der sich ergibt, wenn die von *Normalisierungstyp* angegebene Unicode-Normalisierung für *Quellenzeichenfolge* ausgeführt wird. Wird *Normalisierungstyp* nicht angegeben, wird Unicode Normalization Form C (NFC) für *Quellenzeichenfolge* ausgeführt. Die Unicode-Normalisierung wird in *Character Model for the World Wide Web 1.0* beschrieben.

Wenn *Quellenzeichenfolge* eine leere Sequenz ist, wird eine Zeichenfolge mit Nulllänge zurückgegeben.

### Beispiele

Die folgende Funktion führt Unicode Normalization Form C für die Zeichenfolge "ṃ" (ein lateinischer Kleinbuchstabe m mit einem Punkt darunter) aus:

```
fn:normalize-unicode("&#x6d;&#x323;","NFC")
```

Der zurückgegebene Wert ist das durch die numerische Zeichenreferenz &x1e43; dargestellte UTF-8-Zeichen eines lateinischen Kleinbuchstabens m mit einem Punkt darunter.

In dem folgenden Beispiel wird der normalisierte Unicode in den dezimalen Codepunkt konvertiert:

```
fn:string-to-codepoints(fn:normalize-unicode("&#x6d;&#x323;", "NFC"))
```

Der zurückgegebene Wert lautet '7747'.

## Funktion 'not'

Die Funktion fn:not function gibt den Wert 'false' (falsch) zurück, wenn der effektive Boolesche Wert einer Sequenz 'true' (wahr) ist, bzw. 'true' (wahr), wenn der effektive Boolesche Wert einer Sequenz 'false' (falsch) ist.

### Syntax

*Sequenzausdruck*
Eine beliebige Sequenz mit Elementen beliebigen Typs oder eine leere Sequenz.

### Zurückgegebener Wert

Ist *Sequenzausdruck* keine leere Sequenz, ist der zurückgegebene Wert 'true' (wahr), wenn der effektive Boolesche Wert der Sequenz 'false' (falsch) ist. Der zurückgegebene Wert ist 'false' (falsch), wenn der effektive Boolesche Wert der Sequenz 'true' (wahr) ist.

Ist *Sequenzausdruck* eine leere Sequenz, ist der zurückgegebene Wert 'true' (wahr).

### Beispiel

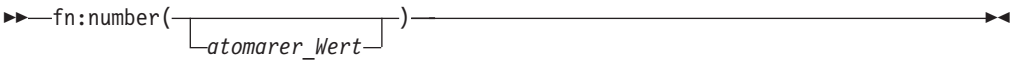
Die folgende Funktion gibt den Wert 'false' (falsch) zurück, da der effektive Boolesche Wert eines Knotens 'true' (wahr) ist.

```
fn:not(<employee />)
```

## Funktion 'number'

Die Funktion fn:number setzt einen Wert in den Datentyp 'xs:double' um.

### Syntax




*atomarer_Wert*
Ein atomarer Wert oder eine leere Sequenz. Wenn *atomarer_Wert* nicht angegeben ist, wird die Funktion 'fn:number' für das aktuelle Kontextelement ausgewertet.

### Zurückgegebener Wert

Wenn *atomarer_Wert* keine leere Sequenz ist, ist der zurückgegebene Wert das Ergebnis der Umsetzung von *atomarer_Wert* in 'xs:double'. Kann *atomarer_Wert* nicht in den Datentyp 'xs:double' umgesetzt werden, wird eine Nichtzahl (NaN) zurückgegeben.

Ist *atomarer_Wert* eine leere Sequenz, wird eine Nichtzahl (NaN) zurückgegeben.

### Beispiele

**Beispiel der Umsetzung eines Wertes vom Typ 'xs:decimal' in den Typ 'xs:double':** Die folgende Funktion konvertiert den Wert 2.75 vom Typ 'xs:decimal' in den Typ 'xs:double':

```
fn:number(2.75)
```

Der zurückgegebene Wert lautet '2.75E0'.

**Beispiel der Umsetzung eines Wertes vom Typ 'xs:boolean' in den Typ 'xs:double':** Die folgende Funktion konvertiert den Booleschen Wert 'false()' in den Typ 'xs:double':

```
fn:number(false())
```

Der zurückgegebene Wert lautet '0.0E0'.

## Funktion 'one-or-more'

Die Funktion fn:one-or-more gibt ihr Argument zurück, wenn es mindestens ein Element enthält.

### Syntax

*Sequenzausdruck*
Eine beliebige Sequenz, einschließlich einer leeren Sequenz.

### Zurückgegebener Wert

Wenn *Sequenzausdruck* mindestens ein Element enthält, wird *Sequenzausdruck* zurückgegeben. Andernfalls wird ein Fehler zurückgegeben.

### Beispiel

In dem folgenden Beispiel wird die Funktion 'fn:one-or-more' verwendet, um zu ermitteln, ob die Sequenz in der Variablen '$seq' mindestens ein Element enthält.

```
let $seq := (5,10)
return fn:one-or-more($seq)
```

Als Ergebnis wird (5,10) zurückgegeben.

## Funktion 'position'

Die Funktion fn:position gibt die Position des Kontextelements in der Sequenz zurück, die zum jeweiligen Zeitpunkt verarbeitet wird.

### Syntax

### Zurückgegebener Wert

Der zurückgegebene Wert hat den Datentyp 'xs:integer' und gibt die Position des Kontextelements in der zum jeweiligen Zeitpunkt verarbeiteten Sequenz an. Ist das Kontextelement nicht definiert, wird ein Fehler zurückgegeben. Die Funktion 'position' gibt nur dann ein deterministisches Ergebnis zurück, wenn die Sequenz, die das Kontextelement enthält, eine deterministische Reihenfolge aufweist. Die Funktion 'position' wird normalerweise in einem Vergleichselement verwendet.

### Beispiel

Im nachstehenden Ausdruck wird die Funktion 'position' für jedes Element in einer aus zehn Elementen bestehenden Sequenz aufgerufen. Für jedes Element gibt die Funktion 'position' die Position des betreffenden Elements in der Sequenz zurück. Das Vergleichselement `position() eq 5` ist nur für das fünfte Element in der Sequenz wahr.

```
(11 to 20)[position() eq 5]
```

Der vom Ausdruck zurückgegebene Wert lautet daher 15.

## Funktion 'QName'

Die Funktion fn:QName erstellt einen erweiterten Namen aus einer Namensbereichs-URI und einer Zeichenfolge, die einen lexikalischen qualifizierten Namen (mit einem optionalen Präfix) enthält.

### Syntax

*URI* Der Teil eines erweiterten Namens mit dem Namensbereich.

*URI* hat den Datentyp 'xs:string' oder ist eine leere Zeichenfolge bzw. leere Sequenz.

*QName*

Ein Wert im korrekten lexikalischen Format des Datentyps 'xs:QName'.

*QName* hat den Datentyp 'xs:string'.

### Zurückgegebener Wert

Der zurückgegebene Wert ist ein Wert vom Typ 'xs:QName'. Hierbei handelt es sich um einen erweiterten Namen mit einer Namensbereichs-URI, die von *URI* angegeben wird, und dem Präfix und lokalen Namen, die von *QName* angegeben werden.

Die Funktion 'fn:QName' ordnet das Namensbereichspräfix von *QName* dem Wert von *URI* zu. Verfügt *QName* über ein Namensbereichspräfix, darf *URI* keine Zeichenfolge mit Nulllänge und keine leere Sequenz sein. Wenn *QName* lediglich einen lokalen Namen und kein Präfix aufweist, kann *URI* eine Zeichenfolge mit Nulllänge oder eine leere Sequenz sein.

### Beispiel

Der folgenden Funktion wird eine Namensbereichs-URI und eine Zeichenfolge mit einem lexikalischen QName zugeordnet, und sie gibt einen Wert vom Typ 'xs:QName' zurück.

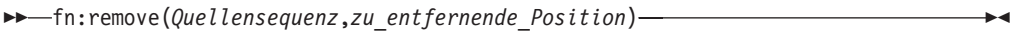
```
fn:QName("http://www.mycompany.com", "comp:employee")
```

Der zurückgegebene Wert ist ein Wert vom Typ 'xs:QName' mit der Namensbereichs-URI "http://www.mycompany.com", dem Präfix "comp" und dem lokalen Namen "employee".

## Funktion 'remove'

Die Funktion fn:remove entfernt ein Element aus einer Sequenz.

### Syntax




*Quellensequenz*

Die Sequenz, aus der ein Element entfernt werden soll.

*Quellensequenz* ist eine Sequenz aus Elementen eines beliebigen Datentyps oder eine leere Sequenz.

*zu_entfernende_Position*

Die Position des zu entfernenden Elements in *Quellensequenz*. Das Argument *zu_entfernende_Position* hat den Datentyp 'xs:integer'.

### Zurückgegebener Wert

Wenn *Quellensequenz* keine leere Sequenz ist:

- Wenn der Wert für *zu_entfernende_Position* kleiner als 1 oder größer als die Länge von *Quellensequenz* ist, dann entspricht der zurückgegebene Wert dem Wert von *Quellensequenz*.
- Wenn der Wert für *zu_entfernende_Position* größer-gleich 1 und kleiner-gleich der Länge von *Quellensequenz* ist, dann ist der zurückgegebene Wert eine Sequenz mit den folgenden Elementen in der folgenden Reihenfolge:
  - Den Elementen in *Quellensequenz* vor dem Element *zu_entfernende_Position*
  - Den Elementen in *Quellensequenz* nach dem Element *zu_entfernende_Position*
- Wenn *Quellensequenz* eine leere Sequenz ist, dann ist der zurückgegebene Wert die leere Sequenz.

### Beispiel

Die folgende Funktion gibt die Sequenz zurück, die sich ergibt, wenn das Element an Position drei aus der Sequenz (1,2,4,7) entfernt wird:

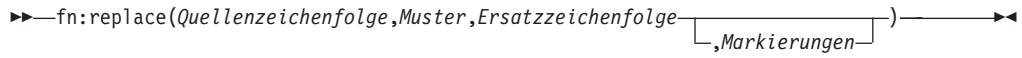
```
fn:remove((1,2,4,7),3)
```

Der zurückgegebene Wert lautet (1,2,7).

## Funktion 'replace'

Die Funktion fn:replace vergleicht jede Gruppe von Zeichen innerhalb einer Zeichenfolge mit einem bestimmten Muster und ersetzt anschließend die Zeichen, die dem Muster entsprechen, mit einer anderen Gruppe von Zeichen.

### Syntax




*Quellenzeichenfolge*
Eine Zeichenfolge mit Zeichen, die ersetzt werden sollen.

*Quellenzeichenfolge* ist ein Wert vom Typ 'xs:string' oder eine leere Sequenz.

*Muster* Ein regulärer Ausdruck, der mit *Quellenzeichenfolge* verglichen wird. Ein regulärer Ausdruck ist eine Gruppe von Zeichen, Platzhalterzeichen und Operatoren, die eine Zeichenfolge oder Gruppe von Zeichenfolgen in einem Suchmuster definieren.

*Muster* ist ein Wert vom Datentyp 'xs:string'.

*Ersatzzeichenfolge*
Eine Zeichenfolge mit Zeichen, die diejenigen Zeichen ersetzen, die mit *Muster* in *Quellenzeichenfolge* übereinstimmen.

*Ersatzzeichenfolge* ist ein Wert vom Typ 'xs:string'.

*Ersatzzeichenfolge* kann die Variablen $0 bis $9 enthalten. Die Variable $0 stellt die gesamte Zeichenfolge in *Muster* dar. Die Variablen $1 bis $9 einschließlich stellen jeweils einen von neun möglichen, in runde Klammern eingeschlossenen Unterausdrücken in *Muster* dar. (Die Variable $1 stellt den ersten Unterausdruck dar, die Variable $2 den zweiten Unterausdruck usw.)

Um das Dollarzeichen ($) als Literal in *Ersatzzeichenfolge* verwenden zu können, muss die Zeichenfolge "\$" verwendet werden. Um den Backslash (\) als Literal in *Ersatzzeichenfolge* verwenden zu können, muss die Zeichenfolge "\\" verwendet werden.

*Markierungen*
Ein Wert vom Typ 'xs:string', der einen der folgenden Werte enthalten kann, die den Abgleich von *Muster* mit *Quellenzeichenfolge* steuern:

**s** Gibt an, dass der Punkt (.) ein beliebiges Zeichen ersetzt.

Wenn die Markierung 's' nicht angegeben ist, ersetzt der Punkt jedes Zeichen mit Ausnahme des Zeilenvorschubzeichens (X'0A').

**m** Gibt an, dass das Winkelzeichen (^) den Beginn einer Zeile ersetzt (die Position nach einem Zeilenvorschubzeichen), und dass das Dollarzeichen ($) das Ende einer Zeile ersetzt (die Position vor einem neuen Zeilenvorschubzeichen).

Wenn die Markierung 'm' nicht angegeben ist, ersetzt das Winkelzeichen (^) den Beginn einer Zeichenfolge, und das Dollarzeichen ($) ersetzt das Ende der Zeichenfolge.

**i** Gibt an, dass der Abgleich nicht von der Groß-/Kleinschreibung abhängt.

Wenn die Markierung 'i' nicht angegeben ist, erfolgt der Abgleich unter Beachtung der Groß-/Kleinschreibung.

**x** Gibt an, dass Leerzeichen innerhalb von *Muster* ignoriert werden.

Wenn die Markierung 'x' nicht angegeben ist, werden Leerzeichen beim Abgleich verwendet.

## Zurückgegebener Wert

Wenn *Quellenzeichenfolge* keine leere Sequenz ist, entspricht der zurückgegebene Wert der Zeichenfolge, die sich ergibt, wenn die folgenden Operationen für *Quellenzeichenfolge* ausgeführt werden:

- *Quellenzeichenfolge* wird auf Zeichen durchsucht, die mit *Muster* übereinstimmen. Wenn *Muster* zwei oder mehr alternative Zeichengruppen enthält, gilt die erste Zeichengruppe in *Muster*, die mit den Zeichen in *Quellenzeichenfolge* übereinstimmt, als das übereinstimmende Muster.
- Jede Zeichengruppe in *Quellenzeichenfolge*, die mit *Muster* übereinstimmt, wird durch *Ersatzzeichenfolge* ersetzt. Wenn *Ersatzzeichenfolge* eine der Variablen $0 bis $9 einschließlich enthält, ersetzt die Unterzeichenfolge von *Quellenzeichenfolge*, die mit dem der Variablen entsprechenden Unterausdruck in *Muster* übereinstimmt, die betreffende Variable in *Ersatzzeichenfolge*. Anschließend wird der geänderte Wert von *Ersatzzeichenfolge* in *Quellenzeichenfolge* eingefügt. Wenn eine Variable über keinen entsprechenden Unterausdruck in *Muster* verfügt, weil mehr Variablen als Unterausdrücke vorhanden sind oder weil ein Unterausdruck keine Übereinstimmung in *Quellenzeichenfolge* besitzt, wird die Variable in *Ersatzzeichenfolge* durch eine Zeichenfolge mit Nulllänge ersetzt.

Wird *Muster* in *Quellenzeichenfolge* nicht gefunden, wird ein Fehler zurückgegeben.

Ist *Quellenzeichenfolge* eine leere Sequenz, wird eine Zeichenfolge mit Nulllänge zurückgegeben.

### Beispiele

**Beispiel für das Ersetzen einer Unterzeichenfolge durch eine andere Unterzeichenfolge:** Die folgende Funktion ersetzt alle Vorkommen von ″a″ in der Zeichenfolge ″abbcacadbdcd″ durch den Wert ″ba″.

```
fn:replace("abbcacadbdcd","a","ba")
```

Der zurückgegebene Wert lautet ″babbcbacbadbdcd″.

**Beispiel für das Ersetzen einer Unterzeichenfolge durch eine Ersatzzeichenfolge mit Variablen:** Die folgende Funktion ersetzt in der Zeichenfolge ″abbcacadbdcd″ den Buchstaben ″a″ durch eine Zeichenfolge aus zweimal dem Buchstaben, der auf ″a″ folgt:

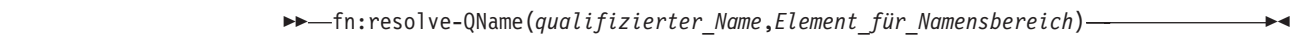
```
fn:replace("abbcacadbdcd","a(.)","$1$1")
```

Der zurückgegebene Wert lautet ″bbbcccddbdcd″.

## Funktion ’resolve-QName’

Die Funktion fn:resolve-QName setzt eine Zeichenfolge, die einen lexikalischen qualifizierten Namen enthält, in einen erweiterten qualifizierten Namen um, indem sie die gültigen Namensbereiche eines Elements verwendet, um das Namensbereichspräfix in eine Namensbereichs-URI aufzulösen.

### Syntax




*qualifizierter_Name*
Eine Zeichenfolge im Format eines qualifizierten Namens.

Das Argument *qualifizierter_Name* hat den Datentyp ’xs:string’ oder ist eine leere Sequenz.

*Element_für_Namensbereich*
Ein Element, das die gültigen Namensbereiche für *qualifizierter_Name* bereitstellt.

*Element_für_Namensbereich* ist ein Elementknoten.

### Zurückgegebener Wert

Wenn *qualifizierter_Name* keine leere Sequenz ist, ist der zurückgegebene Wert ein erweiterter Name, der wie folgt aufgebaut ist:

- Das Präfix und der lokale Name des erweiterten qualifizierten Namens (QName) stammen aus *qualifizierter_Name*.
- Wenn *qualifizierter_Name* über ein Präfix verfügt und dieses Präfix mit einem Präfix in den gültigen Namensbereichen von *Element_für_Namensbereich* übereinstimmt, dann ist die Namensbereichs-URI, an die dieses Präfix gebunden ist, die Namensbereichs-URI für den zurückgegebenen Wert.
- Wenn *qualifizierter_Name* kein Präfix hat und eine Standardnamensbereichs-URI in den gültigen Namensbereichen von *Element_für_Namensbereich* definiert ist, dann ist diese Standardnamensbereichs-URI die Namensbereichs-URI für den zurückgegebenen Wert.

- Wenn *qualifizierter_Name* kein Präfix hat und keine Standardnamensbereichs-URI in den gültigen Namensbereichen von *Element_für_Namensbereich* definiert ist, dann hat auch der zurückgegebene Wert keine Namensbereichs-URI.
- Stimmt das Präfix für *qualifizierter_Name* mit keinem Namensbereichspräfix in den gültigen Namensbereichen von *Element_für_Namensbereich* überein oder liegt *qualifizierter_Name* nicht im Format eines gültigen qualifizierten Namens vor, wird ein Fehler zurückgegeben.

Ist *qualifizierter_Name* eine leere Zeichenfolge, wird die leere Zeichenfolge zurückgegeben.

### Beispiel

Die folgende Abfrage gibt den erweiterten qualifizierten Namen (QName), der mit der URI 'http://www.mycompany.com' übereinstimmt, sowie den lexikalischen qualifzierten Namen 'comp:dept' zurück:

```
declare namespace d="http://www.mycompany.com";
let $department := document {
<comp:dept xmlns:comp="http://www.mycompany.com" id="A07">
  <comp:emp id="31201" />
</comp:dept> }
return fn:resolve-QName("comp:dept", $department/d:dept/d:emp)
```

## Funktion 'reverse'

Die Funktion fn:reverse kehrt die Reihenfolge der Elemente in einer Sequenz um.

### Syntax

*Quellensequenz*
Die Sequenz, die umgekehrt werden soll.

*Quellensequenz* ist eine Sequenz aus Elementen eines beliebigen Datentyps oder eine leere Sequenz.

### Zurückgegebener Wert

Wenn *Quellensequenz* keine leere Sequenz ist, entspricht der zurückgegebene Wert einer Sequenz, die die Elemente von *Quellensequenz* in umgekehrter Reihenfolge enthält.

Ist *Quellensequenz* eine leere Sequenz, wird die leere Sequenz zurückgegeben.

### Beispiel

Die folgende Funktion gibt die Elemente der Sequenz (1,2,3,7) in umgekehrter Reihenfolge zurück:

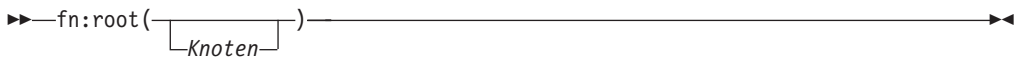
```
fn:reverse((1,2,3,7))
```

Der zurückgegebene Wert lautet (7,3,2,1).

## Funktion 'root'

Die Funktion fn:root gibt den Rootknoten einer Baumstruktur zurück, zu der ein Knoten gehört.

### Syntax




*Knoten* Ein Knoten oder eine leere Sequenz. Der Standardwert für *Knoten* ist ein Kontextknoten.

### Zurückgegebener Wert

Wenn *Knoten* keine leere Sequenz ist, entspricht der zurückgegebene Wert dem Rootknoten der Baumstruktur, zu der *Knoten* gehört. Ist *Knoten* der Rootknoten der Baumstruktur, entspricht der zurückgegebene Wert dem Wert von *Knoten*.

Ist *Knoten* eine leere Sequenz, ist der zurückgegebene Wert die leere Sequenz.

### Beispiel

Angenommen, einige XQuery-Variablen sind wie folgt definiert:

```
let $f := <first>Laura</first>
let $e := <emp> {$f} <last>Brown</last> </emp>
let $doc := document {<emps>{$e}</emps>}
```

**Beispiel mit zurückgegebenem Rootknoten eines Elements:** Die folgende Funktion gibt den Rootknoten des Elements 'last' zurück:

```
fn:root($e/last)
```

Der zurückgegebene Wert lautet
<emp><first>Laura</first><last>Brown</last></emp>.

**Beispiel mit zurückgegebenem Knoten eines Dokuments:** Die folgende Funktion gibt den Rootknoten des Dokuments zurück, das an die Variable `$doc` gebunden ist:

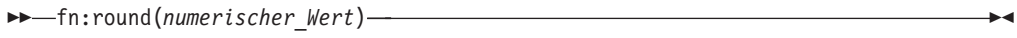
```
fn:root($doc)
```

Der zurückgegebene Wert ist ein Dokumentknoten.

## Funktion 'round'

Die Funktion fn:round gibt diejenige ganze Zahl zurück, die einem bestimmten numerischen Wert am nächsten kommt.

### Syntax




*numerischer_Wert*
Ein atomarer Wert oder eine leere Sequenz.

Ist *numerischer_Wert* ein atomarer Wert, weist er einen der folgenden Typen auf:
- xs:float
- xs:double
- xs:decimal
- xs:integer
- xdt:untypedAtomic
- Einen Typ, der von einem der vorstehend aufgeführten Typen abgeleitet ist

Hat *numerischer_Wert* den Typ 'xdt:untypedAtomic', wird er in einen Wert vom Typ 'xs:double' umgesetzt.

### Zurückgegebener Wert

Wenn *numerischer_Wert* keine leere Sequenz ist, dann ist der zurückgegebene Wert die ganze Zahl, die dem Wert von *numerischer_Wert* am nächsten ist. Dies bedeutet, dass die Funktion 'fn:round(*numerischer_Wert*) äquivalent zur Funktion 'fn:floor-(*numerischer_Wert*+0.5) ist. Der Datentyp des zurückgegebenen Wertes hängt vom Datentyp von *numerischer_Wert* ab:
- Hat *numerischer_Wert* den Datentyp 'xs:float', 'xs:double', 'xs:decimal' oder 'xs:integer', weist der zurückgegebene Wert denselben Typ wie *numerischer_Wert* auf.
- Hat *numerischer_Wert* einen Datentyp, der von 'xs:float', 'xs:double', 'xs:decimal' oder 'xs:integer' abgeleitet ist, weist der zurückgegebene Wert den direkten übergeordneten Datentyp von *numerischer_Wert* auf.

Ist *numerischer_Wert* eine leere Sequenz, ist der zurückgegebene Wert die leere Sequenz.

### Beispiele

**Beispiel mit einem positiven Argument:** Die folgende Funktion gibt den gerundeten Wert für 0.5 zurück:

```
fn:round(0.5)
```

Der zurückgegebene Wert ist 1.

**Beispiel mit einem negativen Argument:** Die folgende Funktion gibt den gerundeten Wert für (-1.5) zurück:

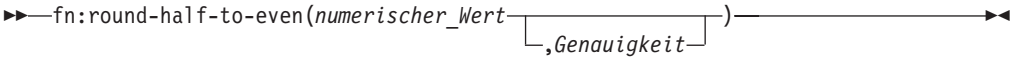
```
fn:round(-1.5)
```

Der zurückgegebene Wert ist -1.

## Funktion 'round-half-to-even'

Die Funktion fn:round-half-to-even gibt denjenigen numerischen Wert mit einer bestimmten Genauigkeit zurück, der einem bestimmten numerischen Wert am nächsten kommt.

### Syntax



*numerischer_Wert*
Ein atomarer Wert oder eine leere Sequenz.

Ist *numerischer_Wert* ein atomarer Wert, weist er einen der folgenden Typen auf:
- xs:float
- xs:double
- xs:decimal
- xs:integer
- xdt:untypedAtomic
- Einen Typ, der von einem der vorstehend aufgeführten Typen abgeleitet ist

Hat *numerischer_Wert* den Typ 'xdt:untypedAtomic', wird er in einen Wert vom Typ 'xs:double' umgesetzt.

*Genauigkeit*
Die Anzahl der Stellen nach dem Dezimalzeichen, auf die *numerischer_Wert* gerundet werden soll. *Genauigkeit* ist ein Wert vom Typ 'xs:integer'. Der Standardwert für *Genauigkeit* ist 0.

## Zurückgegebener Wert

Wenn *numerischer_Wert* keine leere Sequenz ist und *Genauigkeit* den Wert 0 aufweist oder nicht angegeben ist, dann ist der zurückgegebene Wert diejenige ganze Zahl, die dem Wert von *numerischen_Wert* am nächsten kommt. Ist *numerischer_Wert* zwei ganzen Zahlen gleich nah, dann ist der zurückgegebene Wert die gerade ganze Zahl.

Ist *numerischer_Wert* keine leere Sequenz und *Genauigkeit* ist nicht 0, dann ist der zurückgegebene Wert ein numerischer Wert, der die mit *Genauigkeit* angegebene Anzahl anstellen aufweist und dem Wert von *numerischer_Wert* am nächsten ist. Ist *numerischer_Wert* zwei Werten gleich nah, dann ist der zurückgegebene Wert derjenige Wert, dessen niedrigstwertige Stelle eine gerade Zahl ist.

Der Datentyp des zurückgegebenen Wertes hängt vom Datentyp von *numerischer_Wert* ab:
- Hat *numerischer_Wert* den Datentyp 'xs:float', 'xs:double', 'xs:decimal' oder 'xs:integer', weist der zurückgegebene Wert denselben Typ wie *numerischer_Wert* auf.
- Hat *numerischer_Wert* einen Datentyp, der von 'xs:float', 'xs:double', 'xs:decimal' oder 'xs:integer' abgeleitet ist, weist der zurückgegebene Wert den direkten übergeordneten Datentyp von *numerischer_Wert* auf.

Ist *numerischer_Wert* eine leere Sequenz, ist der zurückgegebene Wert die leere Sequenz.

## Beispiele

**Beispiel ohne Argument für Genauigkeit:** Die folgende Funktion gibt den gerundeten Wert von 0.5 zurück.

```
fn:round-half-to-even(0.5)
```

Der zurückgegebene Wert ist 0.

**Beispiel mit einem Argument für Genauigkeit mit einem Wert von ungleich Null:** Die folgende Funktion gibt den auf zwei Dezimalstellen gerundeten Wert von 1.5432 zurück:

```
fn:round-half-to-even(1.5432,2)
```

Der zurückgegebene Wert ist 1.54.

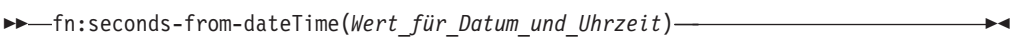
**Beispiel mit negativer Genauigkeit:** Die folgende Funktion gibt den Wert 35600 zurück.

```
fn:round-half-to-even(35612.25, -2)
```

## Funktion 'seconds-from-dateTime'

Die Funktion fn:seconds-from-dateTime gibt die Sekundenkomponente eines Wertes vom Typ 'xs:dateTime' zurück.

### Syntax




*Wert_für_Datum_und_Uhrzeit*
: Der Wert für Datum und Uhrzeit, aus dem die Sekundenkomponente extrahiert werden soll.

  *Wert_für_Datum_und_Uhrzeit* hat den Datentyp 'xs:dateTime' oder ist eine leere Sequenz.

### Zurückgegebener Wert

Hat *Wert_für_Datum_und_Uhrzeit* den Typ 'xs:dateTime', weist der zurückgegebene Wert den Typ 'xs:decimal' auf und liegt im Bereich von 0 bis kleiner als 60. Der Wert ist die Sekundenkomponente (Sekunden und Sekundenbruchteile) von *Wert für Datum und Uhrzeit*.

Ist *Wert_für_Datum_und_Uhrzeit* eine leere Sequenz, ist der zurückgegebene Wert eine leere Sequenz.

### Beispiele

Die folgende Funktion gibt die Sekundenkomponente des folgenden Werts für Datum und Uhrzeit zurück: 08. Februar 2005, 14:16:23 Uhr in Zeitzone UTC-8.

```
fn:seconds-from-dateTime(xs:dateTime("2005-02-08T14:16:23-08:00"))
```

Der zurückgegebene Wert lautet '23'.

Die folgende Funktion gibt die Sekundenkomponente des folgenden Werts für Datum und Uhrzeit zurück: 23. Juni 2005, 09:16:20.43 in Zeitzone UTC:

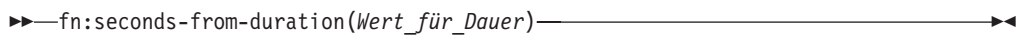
```
fn:seconds-from-dateTime(xs:dateTime("2005-06-23T09:16:23.43Z"))
```

Der zurückgegebene Wert ist 20.43.

## Funktion 'seconds-from-duration'

Die Funktion fn:seconds-from-duration gibt die Sekundenkomponente einer Zeitdauer zurück.

### Syntax




*Wert_für_Dauer*
Der Wert für Dauer, aus dem die Sekundenkomponente extrahiert werden soll.

*Wert_für_Dauer* ist eine leere Sequenz oder ein Wert, der einen der folgenden Datentypen aufweist: xdt:dayTimeDuration, xs:duration oder xdt:yearMonthDuration.

### Zurückgegebener Wert

Der zurückgegebene Wert hängt vom Datentyp von *Wert_für_Dauer* ab:

- Hat *Wert_für_Dauer* den Typ 'xdt:dayTimeDuration' oder 'xs:duration', weist der zurückgegebene Wert den Typ 'xs:decimal' auf und liegt im Bereich von größer als -60 bis kleiner als 60. Der Wert ist die Sekundenkomponente (Sekunden und Sekundenbruchteile) von *Wert_für_Dauer*, die als 'xdt:dayTimeDuration' umgesetzt ist. Der Wert ist negativ, wenn *Wert_für_Dauer* negativ ist.
- Hat *Wert_für_Dauer* den Typ 'xdt:yearMonthDuration', weist der zurückgegebene Wert den Typ 'xs:integer' auf und ist 0.
- Ist *Wert_für_Dauer* eine leere Sequenz, ist der zurückgegebene Wert eine leere Sequenz.

Die als 'xdt:dayTimeDuration' umgesetzte Sekundenkomponente aus *Wert_für_Dauer* wird als `(S mod 60)` berechnet. Der Wert *S* ist die Gesamtzahl der Sekunden und Sekundenbruchteile von *Wert_für_Dauer*, die als 'xdt:dayTimeDuration' umgesetzt wird, um die Komponenten der Jahre und Monate zu entfernen.

### Beispiel

Die folgende Funktion gibt die Sekundenkomponente der Dauer von 150.5 Sekunden zurück:

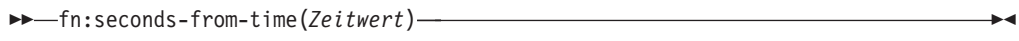
```
fn:seconds-from-duration(xdt:dayTimeDuration("PT150.5S"))
```

Der zurückgegebene Wert lautet 30.5. Bei der Berechnung der Gesamtzahl der Sekunden in der Dauer werden die 150.5 Sekunden in 2 Minuten und 30.5 Sekunden umgewandelt. Die Dauer entspricht PT2M30.5S. Hierbei gibt die Sekundenkomponente 30.5 Sekunden an.

## Funktion 'seconds-from-time'

Die Funktion fn:seconds-from-time gibt die Sekundenkomponente eines Wertes vom Typ 'xs:time' zurück.

### Syntax




*Zeitwert*
Der Zeitwert, aus dem die Sekundenkomponente extrahiert werden soll.

*Zeitwert* hat den Datentyp 'xs:time' oder ist eine leere Sequenz.

### Zurückgegebener Wert

Hat *Zeitwert* den Typ 'xs:time', weist der zurückgegebene Wert den Typ 'xs:decimal' auf und liegt im Bereich von null bis weniger als 60. Der Wert ist die Sekundenkomponente (Sekunden und Sekundenbruchteile) von *Zeitwert*.

Ist *Zeitwert* eine leere Sequenz, ist der zurückgegebene Wert eine leere Sequenz.

### Beispiel

Die folgende Funktion gibt die Sekundenkomponente des folgenden Zeitwerts zurück: 08:59:59 Uhr in Zeitzone UTC-8:

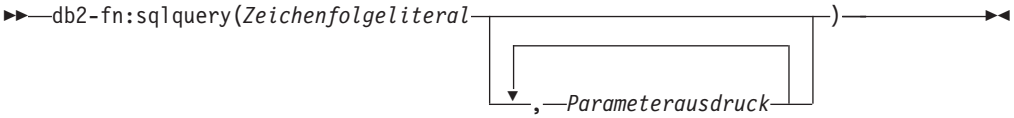
```
fn:seconds-from-time(xs:time("08:59:59-08:00"))
```

Der zurückgegebene Wert ist 59.

## Funktion 'sqlquery'

Die Funktion db2-fn:sqlquery ruft eine Sequenz ab, die das Ergebnis einer SQL-Fullselect-Anweisung in der zum jeweiligen Zeitpunkt verbundenen DB2-Datenbank ist.

### Syntax




*Zeichenfolgeliteral*

Enthält einen Fullselect. Die Fullselect-Operation muss eine aus einer Spalte bestehende Ergebnismenge angeben, wobei diese Spalte den Datentyp XML aufweisen muss. Der Bereich der Fullselect-Operation ist ein neuer SQL-Abfragebereich, keine verschachtelte SQL-Abfrage.

Wenn der Fullselect einfache Anführungszeichen enthält (beispielsweise zum Einschließen einer Zeichenfolgekonstanten), muss das Funktionsargument in doppelte Anführungszeichen eingeschlossen werden. Beispiel:

```
"select c1 from t1 where c2 = 'Hello'"
```

Wenn der Fullselect doppelte Anführungszeichen enthält (beispielsweise zum Einschließen eines begrenzten Bezeichners), muss das Funktionsargument in einfache Anführungszeichen eingeschlossen werden. Beispiel:

```
'select c1 from "t1" where c2 = 47'
```

Wenn der Fullselect sowohl einfache als auch doppelte Anführungszeichen enthält, muss das Funktionsargument in einfache Anführungszeichen eingeschlossen werden, und jedes interne einfache Anführungszeichen muss durch zwei benachbarte einfache Anführungszeichen dargestellt werden. Beispiel:

```
'select c1 from "t1" where c2 = ''Hello'''
```

Der Fullselect kann Aufrufe an die Funktion PARAMETER enthalten, um auf den Ergebniswert von jedem im Funktionsaufruf 'db2-fn:sqlquery' angegebenen *Parameterausdruck* zu verweisen. Die Aufrufe der Funktion

PARAMETER werden bei Ausführung der Fullselect-Operation durch den Ergebniswert des entsprechenden *Parameterausdrucks* ersetzt.

*Parameterausdruck*
Ein XQuery-Ausdruck, der einen Wert zurückgibt. Durch Angeben einer SQL-Funktion PARAMETER mit einem Argument für ganzzahlige Werte in der SQL-Fullselect-Operation kann auf den Ergebniswert jedes *Parameterausdrucks* verwiesen werden. Der ganzzahlige Wert ist ein Index für *Parameterausdruck* anhand von dessen Position im Funktionsaufruf 'db2-fn:sqlquery'. Die gültigen ganzzahligen Werte liegen zwischen 1 und der Gesamtzahl der Argumente vom Typ *Parameterausdruck* im Funktionsaufruf. Beispiel: Wenn das Argument *Zeichenfolgeliteral* die Elemente `PARAMETER(1)` und `PARAMETER(2)` im SQL-Fullselect enthält, muss der Funktionsaufruf zwei XQuery-Argumente vom Typ *Parameterausdruck* angeben. `PARAMETER(1)` verweist auf das Ergebnis des ersten Arguments *Parameterausdruck*, und `PARAMETER(2)` verweist auf das Ergebnis des zweiten Arguments *Parameterausdruck*.

Während der Verarbeitung der SQL-Fullselect-Operation wird jeder Funktionsaufruf vom Typ PARAMETER durch den Ergebniswert des entsprechenden *Parameterausdrucks* im Funktionsaufruf 'db2-fn:sqlquery' ersetzt. Jeder *Parameterausdruck* wird genau einmal ausgewertet, unabhängig davon, wie oft auf ihn in der SQL-Fullselect-Operation verwiesen wird.

Der Ergebnisdatentyp des entsprechenden *Parameterausdrucks* muss gemäß den Regeln von XMLCAST in den Ergebnistyp der Funktion PARAMETER umsetzbar sein. Andernfalls wird ein Fehler zurückgegeben.

Der Ergebnistyp der Funktion PARAMETER wird ermittelt, als handelte es sich um eine Parametermarke im SQL-Fullselect. Eine Parametermarke wird in anderen Kontexten beispielsweise durch ein Fragezeichen (?) angezeigt. Kann der Ergebnistyp für eine Funktion vom Typ PARAMETER nicht ermittelt werden, wird ein Fehler zurückgegeben.

**Tipp:** Wenn eine nicht mit einem Datentyp versehene Parametermarke für eine Operation nicht zulässig ist, können Sie die Spezifikation CAST oder XMLCAST verwenden, um einen Typ anzugeben. Um beispielsweise `PARAMETER(1)` in den Typ DOUBLE umzusetzen, verwenden Sie die folgende CAST-Spezifikation: `CAST(PARAMETER(1) as double)`.

## Zurückgegebener Wert

Der zurückgegebene Wert ist eine Sequenz, die das Ergebnis der Fullselect-Operation in *Zeichenfolgeliteral* darstellt. Der Fullselect wird als SQL-Anweisung verarbeitet, entsprechend der üblichen dynamischen SQL-Regeln für Berechtigung und Namensauflösung. Wenn der Fullselect Aufrufe an die Funktion PARAMETER enthält, werden diese Aufrufe bei der Auswertung des Fullselects durch den Ergebniswert des XQuery-Ausdrucks des entsprechenden Arguments *Parameterausdruck* ersetzt. Die vom Fullselect zurückgegebenen XML-Werte werden verknüpft, und die verknüpften Werte stellen das Ergebnis der Funktion dar. Zeilen mit Nullwerten haben keine Auswirkung auf die Ergebnissequenz. Wenn der Fullselect keine Zeilen oder nur Nullwerte zurückgibt, ist das Ergebnis der Funktion eine leere Sequenz.

Die Anzahl der Elemente in der Sequenz, die von der Funktion 'db2-fn:sqlquery' zurückgegeben wird, kann sich von der Anzahl der vom Fullselect zurückgegebenen Zeilen unterscheiden, da einige dieser Zeilen unter Umständen Nullwerte oder Sequenzen mit mehreren Elementen enthalten.

## Beispiele

**Beispiel einer Fullselect-Operation, die eine Sequenz aus XML-Dokumenten zurückgibt:** Das nachstehende Beispiel zeigt mehrere Funktionsaufrufe, die dieselbe Sequenz aus Dokumenten aus der Tabelle PRODUCT zurückgeben. Die Dokumente befinden sich in der Spalte DESCRIPTION.

Jede der folgenden Funktionen generiert dasselbe Ergebnis:

```
db2-fn:sqlquery('select description from product')
db2-fn:sqlquery('SELECT DESCRIPTION FROM PRODUCT')
db2-fn:sqlquery('select "DESCRIPTION" from "PRODUCT"')
```

**Beispiel von Fullselect-Operationen, die ein einziges XML-Dokument zurückgeben:** Das nachstehende Beispiel gibt eine Sequenz zurück, bei der es sich um ein einzelnes Dokument in der Tabelle PRODUCT handelt. Das Dokument befindet sich in der Spalte DESCRIPTION und wird durch den Wert '100-103-01' für die Spalte PID identifiziert.

Jede der folgenden Funktionen generiert dasselbe Ergebnis:

```
db2-fn:sqlquery('select Description from Product where pID=''100-103-01''')
db2-fn:sqlquery("select description from product where pid='100-103-01'")
db2-fn:sqlquery("select ""DESCRIPTION"" from product where pid='100-103-01'")
```

**Beispiel einer Fullselect-Operation mit zwei Aufrufen an die Funktion PARAMETER und einem Ausdruck:** Das folgende Beispiel gibt die Bestellkennung (purchase ID), die Teilekennung (part ID) und das Bestelldatum (purchase date) für alle während der Werbeaktion (promotion dates) verkauften Teile zurück:

```
  xquery
declare default element namespace "http://posample.org";
for $po in db2-fn:xmlcolumn('PURCHASEORDER.PORDER')/PurchaseOrder,
  $item in $po/item/partid
for $p in db2-fn:sqlquery(
  "select description
  from product
  where promostart < parameter(1)
  and  promoend  >  parameter(1)",
  $po/@OrderDate )
where $p//@pid = $item
return
<RESULT>
  <PoNum>{data($po/@PoNum)}</PoNum>
  <PartID>{data($item)} </PartID>
  <PoDate>{data($po/@OrderDate)}</PoDate>
</RESULT>
```

Während der Verarbeitung der Funktion 'db2-fn:sqlquery' geben beide Verweise auf `parameter(1)` den Wert des Attributs für das Bestelldatum `$po/@OrderDate` zurück.

**Beispiel einer Fullselect-Operation mit zwei Aufrufen an die Funktion PARAMETER und zwei Ausdrücken:** Das nachstehende Beispiel verwendet die Tabelle PURCHASEORDER aus der DB2-Beispieldatenbank SAMPLE. Der XQuery-Ausdruck ruft nicht versandte ("unshipped") Bestellungen mit einem Bestelldatum vor dem 4. April 2006 ab und listet die unterschiedlichen Teilenummern aus jeder Bestellung auf:

```
  xquery
declare default element namespace 'http://posample.org' ;
let $status := ( "Unshipped" ), $date := ( "2006-04-04" )
for $myorders in db2-fn:sqlquery(
```

```
   "select porder from purchaseorder
   where status = parameter(1)
   and orderdate < parameter(2)",
   $status, $date )
return
<LateOrder>
  <PoNum>
  {data($myorders/PurchaseOrder/@PoNum)}
  </PoNum>
  <PoDate>
  {data($myorders/PurchaseOrder/@OrderDate)}
  </PoDate>
  <Items>
     {for $itemID in distinct-values( $myorders/PurchaseOrder/item/partid )
     return
     <PartID>
     {$itemID}
     </PartID>}
   </Items>
</LateOrder>
```

Während der Verarbeitung der Funktion 'db2-fn:sqlquery' gibt der Verweis auf `parameter(1)` den Ergebniswert des Ausdrucks `$status` zurück, und der Verweis auf `parameter(2)` gibt den Ergebniswert des Ausdrucks `$date` zurück.

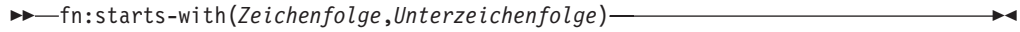
Bei Ausführung für die Datenbank SAMPLE gibt der Ausdruck das folgende Ergebnis zurück:

```
<LateOrder xmlns="http://posample.org">
  <PoNum>5000</PoNum>
  <PoDate>2006-02-18</PoDate>
  <Items>
    <PartID>100-100-01</PartID>
    <PartID>100-103-01</PartID>
  </Items>
</LateOrder>
```

## Funktion 'starts-with'

Die Funktion fn:starts-with ermittelt, ob eine Zeichenfolge mit einer bestimmten Zeichenfolge beginnt. Die Unterzeichenfolge wird unter Verwendung der Standardsortierfolge abgeglichen.

### Syntax




*Zeichenfolge*
: Die Zeichenfolge, in der nach *Unterzeichenfolge* gesucht werden soll.

  *Zeichenfolge* hat den Datentyp 'xs:string' oder ist eine leere Sequenz. Ist *Zeichenfolge* eine leere Sequenz, wird das Argument auf eine Zeichenfolge mit Nulllänge gesetzt.

*Unterzeichenfolge*
: Die Unterzeichenfolge, nach der am Anfang von *Zeichenfolge* gesucht werden soll.

  *Unterzeichenfolge* hat den Datentyp 'xs:string' oder ist eine leere Sequenz.

**Längenbeschränkung**

: Die Länge von *Unterzeichenfolge* ist auf 32.000 Byte beschränkt.

### Zurückgegebener Wert

Der zurückgegebene Wert ist der Wert 'true' (wahr) vom Typ 'xs:boolean', wenn eine der folgenden Bedingungen erfüllt ist:

- *Unterzeichenfolge* kommt am Anfang von *Zeichenfolge* vor.
- *Unterzeichenfolge* ist eine leere Sequenz oder eine Zeichenfolge mit Nulllänge.

Ansonsten ist der zurückgegebene Wert 'false' (falsch).

### Beispiel

Die folgende Funktion ermittelt, ob die Zeichenfolge 'Test literal' mit der Zeichenfolge 'lite' beginnt:

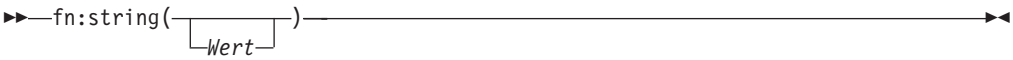
```
fn:starts-with('Test literal','lite')
```

Der zurückgegebene Wert ist 'false' (falsch).

## Funktion 'string'

Die Funktion fn:string gibt die Zeichenfolgedarstellung eines Wertes zurück.

### Syntax




*Wert* Der Wert, der als Zeichenfolge dargestellt werden soll.

*Wert* ist ein Knoten, ein atomarer Wert oder eine leere Sequenz.

Wenn *Wert* nicht angegeben ist, wird die Funktion 'fn:string' für das aktuelle Kontextelement ausgewertet. Ist das aktuelle Kontextelement nicht definiert, wird ein Fehler zurückgegeben.

### Zurückgegebener Wert

Wenn *Wert* keine leere Sequenz ist:

- Ist *Wert* ein Knoten, ist der zurückgegebene Wert der Zeichenfolgewert des Knotens.
- Ist *Wert* ein atomarer Wert, ist der zurückgegebene Wert das Ergebnis der Umsetzung von *Wert* in den Datentyp 'xs:string'.

Ist *Wert* eine leere Sequenz, ist das Ergebnis eine Zeichenfolge mit Nulllänge.

### Beispiel

Die folgende Funktion gibt die Zeichenfolgedarstellung von 123 zurück:

```
fn:string(xs:integer(123))
```

Der zurückgegebene Wert ist '123'.

## Funktion 'string-join'

Die Funktion fn:string-join gibt eine Zeichenfolge zurück, die durch Verknüpfung von Elementen erstellt wird, die durch ein Trennzeichen voneinander getrennt sind.

### Syntax

*Sequenz*
Die Sequenz aus Elementen, die zu einer Zeichenfolge verknüpft werden sollen.

*Sequenz* ist eine beliebige Sequenz aus Werten vom Typ 'xs:string' oder eine leere Sequenz.

*Trennzeichen*
Ein Begrenzer, der zwischen die Elemente aus *Sequenz* in die Ergebniszeichenfolge eingefügt wird.

*Trennzeichen* hat den Datentyp 'xs:string'.

### Zurückgegebener Wert

Der zurückgegebene Wert ist eine Zeichenfolge, bei der es sich um die Verknüpfung der Elemente in *Sequenz* handelt, die durch *Trennzeichen* getrennt sind. Ist *Trennzeichen* eine Zeichenfolge mit Nulllänge, werden die Elemente in *Sequenz* ohne Trennzeichen verknüpft. Ist *Sequenz* eine leere Sequenz, wird eine Zeichenfolge mit Nulllänge zurückgegeben.

### Beispiel

Die folgende Funktion gibt die Zeichenfolge zurück, die sich aus der Verknüpfung der Elemente in der Sequenz (″I″ , ″made″, ″a″, ″sentence!″) ergibt. Als Trennzeichen wird hierbei das Leerzeichen verwendet.

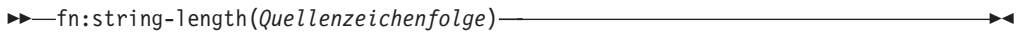
```
fn:string-join(("I" , "made", "a", "sentence!"), " ")
```

Der zurückgegebene Wert ist die Zeichenfolge ″I made a sentence!″.

## Funktion 'string-length'

Die Funktion fn:string-length gibt die Länge einer Zeichenfolge zurück.

### Syntax




*Quellenzeichenfolge*
Die Zeichenfolge, deren Länge zurückgegeben werden soll.

*Quellenzeichenfolge* hat den Datentyp 'xs:string' oder ist eine leere Sequenz.

### Zurückgegebener Wert

Wenn *Quellenzeichenfolge* keine leere Sequenz ist, ist der zurückgegebene Wert die Länge von *Quellenzeichenfolge* in Zeichen. Codepunkte oberhalb von xFFFF, die zwei als Ersatzzeichenpaare bezeichnete 16-Bit-Werte verwenden, werden bei der Länge der Zeichenfolge als ein Zeichen gezählt. *Quellenzeichenfolge* ist ein Wert vom Typ 'xs:integer'.

Wenn *Quellenzeichenfolge* eine leere Sequenz ist, ist der zurückgegebene Wert '0'.

### Beispiel

Die folgende Funktion gibt die Länge der Zeichenfolge 'Test literal' zurück.

```
fn:string-length('Test literal')
```

Der zurückgegebene Wert lautet 12.

## Funktion 'string-to-codepoints'

Die Funktion fn:string-to-codepoints gibt eine Sequenz aus Unicode-Codepunkten zurück, die einem Zeichenfolgewert entsprechen.

### Syntax

*Quellenzeichenfolge*
Ein Zeichenfolgewert, für den der Unicode-Codepunkt jedes einzelnen Zeichens zurückgegeben werden soll, oder eine leere Sequenz.

### Zurückgegebener Wert

Wenn *Quellenzeichenfolge* keine leere Sequenz ist und keine Nulllänge aufweist, ist der zurückgegebene Wert eine Sequenz aus Werten vom Typ 'xs:integer', die die Codepunkte für die Zeichen in *Quellenzeichenfolge* darstellen.

Wenn *Quellenzeichenfolge* eine leere Sequenz ist oder eine Nulllänge aufweist, ist der zurückgegebene Wert die leere Sequenz.

### Beispiel

Die folgende Funktion gibt eine Sequenz aus Codepunkten zurück, die die Zeichen in der Zeichenfolge 'XQuery' darstellen.

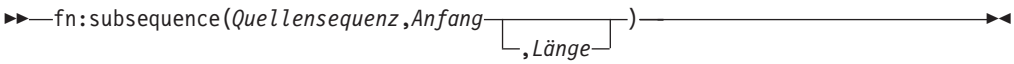
```
fn:string-to-codepoints("XQuery")
```

Der zurückgegebene Wert ist (88,81,117,101,114,121).

## Funktion 'subsequence'

Die Funktion fn:subsequence gibt eine Teilsequenz einer Sequenz zurück.

### Syntax




*Quellensequenz*
Die Sequenz aus der die Teilsequenz abgerufen wird.

*Quellensequenz* ist eine beliebige Sequenz, einschließlich einer leeren Sequenz.

*Anfang*
Die Anfangsposition der Teilsequenz in *Quellensequenz*. Die erste Position von *Quellensequenz* ist 1. Wenn *Anfang*<=0, wird *Anfang* auf 1 gesetzt.

*Anfang* hat den Datentyp 'xs:double'.

*Länge* Die Anzahl der Elemente in der Teilsequenz. Der Standardwert für *Länge* ist die Anzahl der Elemente in *Quellensequenz*. Wenn '*Anfang*+*Länge*-1' größer als die Länge von *Quellensequenz* ist, wird *Länge* auf 'Länge von *Quellensequenz*-*Anfang*+1' gesetzt.

*Länge* hat den Datentyp 'xs:double'.

### Zurückgegebener Wert

Wenn *Quellensequenz* keine leere Sequenz ist, dann ist der zurückgegebene Wert eine Teilsequenz von *Quellensequenz*, die bei Position *Anfang* beginnt und eine dem Wert von *Länge* entsprechende Anzahl an Elementen enthält.

Ist *Quellensequenz* eine leere Sequenz, wird die leere Sequenz zurückgegeben.

### Beispiel

Die folgende Funktion gibt drei Elemente aus der Sequenz ('T','e','s','t',' ','s','e','q','u','e','n','c','e') zurück, wobei der Anfang beim sechsten Element liegt:

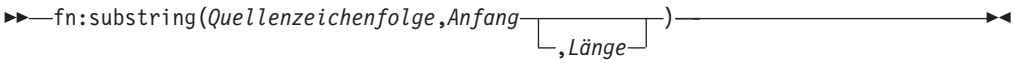
```
fn:subsequence(('T','e','s','t',' ','s','e','q','u','e','n','c','e'),6,3)
```

Der zurückgegebene Wert lautet ('s','e','q').

## Funktion 'substring'

Die Funktion fn:substring gibt eine Unterzeichenfolge einer Zeichenfolge zurück.

### Syntax




*Quellenzeichenfolge*
Die Zeichenfolge, aus der die Unterzeichenfolge abgerufen wird.

*Quellenzeichenfolge* hat den Datentyp 'xs:string' oder ist eine leere Sequenz.

*Anfang*
Die Anfangszeichenposition der Unterzeichenfolge in *Quellenzeichenfolge*. Die erste Position von *Quellenzeichenfolge* ist 1. Wenn *Anfang*<=0, wird *Anfang* auf 1 gesetzt. Codepunkte oberhalb von xFFFF, die zwei als Ersatzzeichenpaare bezeichnete 16-Bit-Werte verwenden, werden als ein Zeichen gezählt.

*Anfang* hat den Datentyp 'xs:double'.

*Länge* Die Länge der Unterzeichenfolge in Zeichen. Der Standardwert für *Länge* ist die Länge von *Quellenzeichenfolge*. Wenn '*Anfang+Länge*-1' größer als die Länge von *Quellenzeichenfolge* ist, wird *Länge* auf 'Länge von *Quellenzeichenfolge-Anfang*+1' gesetzt. Codepunkte oberhalb von xFFFF, die zwei als Ersatzzeichenpaare bezeichnete 16-Bit-Werte verwenden, werden bei der Länge der Zeichenfolge als ein Zeichen gezählt.

*Länge* hat den Datentyp 'xs:double'.

### Zurückgegebener Wert

Wenn *Quellenzeichenfolge* keine leere Sequenz ist, dann ist der zurückgegebene Wert eine Unterzeichenfolge von *Quellenzeichenfolge*, die bei Zeichenposition *Anfang* beginnt und eine dem Wert von *Länge* entsprechende Anzahl an Zeichen enthält. Wenn *Quellenzeichenfolge* eine leere Sequenz ist, dann ist das Ergebnis eine Zeichenfolge mit Nulllänge.

### Beispiel

Die folgende Funktion gibt sieben Zeichen zurück, wobei der Anfang beim sechsten Zeichen der Zeichenfolge 'Test literal' liegt.

```
fn:substring('Test literal',6,7)
```

Der zurückgegebene Wert lautet 'literal'.

## Funktion 'substring-after'

Die Funktion fn:substring-after gibt eine Unterzeichenfolge zurück, die in einer Zeichenfolge nach dem Ende des ersten Vorkommens eines bestimmten Suchbegriffs auftritt. Der Suchbegriff wird unter Verwendung der Standardsortierfolge abgeglichen.

### Syntax

*Quellenzeichenfolge*
Die Zeichenfolge, aus der die Unterzeichenfolge abgerufen wird.

*Quellenzeichenfolge* hat den Datentyp 'xs:string' oder ist eine leere Sequenz. Ist *Quellenzeichenfolge* eine leere Sequenz, wird das Argument auf eine Zeichenfolge mit Nulllänge gesetzt.

*Suchbegriff*
Die Zeichenfolge, nach deren erstem Vorkommen in *Quellenzeichenfolge* gesucht werden soll.

*Suchbegriff* hat den Datentyp 'xs:string' oder ist eine leere Sequenz.

**Längenbeschränkung**

Die Länge von *Suchbegriff* ist auf 32.000 Byte beschränkt.

### Zurückgegebener Wert

Wenn *Suchbegriff* keine leere Sequenz und keine Zeichenfolge mit Nulllänge ist:

- Angenommen, die Länge von *Quellenzeichenfolge* ist *n* und es gilt: *m*<*n*. Wird *Suchbegriff* in *Quellenzeichenfolge* gefunden, und das Ende des ersten Vorkommens von *Suchbegriff* in *Quellenzeichenfolge* liegt bei Position *m*, dann ist der zurückgegebene Wert die Unterzeichenfolge, die bei Position *m*+1 beginnt und bei Position *n* von *Quellenzeichenfolge* endet.
- Angenommen, die Länge von *Quellenzeichenfolge* ist *n*. Wird *Suchbegriff* in *Quellenzeichenfolge* gefunden und liegt das Ende des ersten Vorkommens von *Suchbegriff* in *Quellenzeichenfolge* bei Position *n*, dann ist der zurückgegebene Wert eine Zeichenfolge mit Nulllänge.
- Ist *Suchbegriff* eine leere Zeichenfolge oder eine Zeichenfolge mit Nulllänge, wird *Quellenzeichenfolge* als Wert zurückgegeben.
- Wird *Suchbegriff* in *Quellenzeichenfolge* nicht gefunden, ist der zurückgegebene Wert eine Zeichenfolge mit Nulllänge.

Ist *Quellenzeichenfolge* eine leere Sequenz oder eine Zeichenfolge mit Nulllänge, ist der zurückgegebene Wert eine Zeichenfolge mit Nulllänge.

### Beispiel

Die folgende Funktion sucht die Zeichen nach 'ABC' in der Zeichenfolge 'DEFABCD' unter Verwendung der Standardsortierfolge.

```
fn:substring-after('DEFABCD', 'ABC')
```

Der zurückgegebene Wert lautet 'D'.

## Funktion 'substring-before'

Die Funktion fn:substring-before gibt eine Unterzeichenfolge zurück, die in einer Zeichenfolge vor dem ersten Auftreten eines bestimmten Suchbegriffs auftritt. Der Suchbegriff wird unter Verwendung der Standardsortierfolge abgeglichen.

### Syntax

*Quellenzeichenfolge*
Die Zeichenfolge, aus der die Unterzeichenfolge abgerufen wird.

*Quellenzeichenfolge* hat den Datentyp 'xs:string' oder ist eine leere Sequenz. Ist *Quellenzeichenfolge* eine leere Sequenz, wird das Argument auf eine Zeichenfolge mit Nulllänge gesetzt.

*Suchbegriff*
Die Zeichenfolge, nach deren erstem Vorkommen in *Quellenzeichenfolge* gesucht werden soll.

*Suchbegriff* hat den Datentyp 'xs:string' oder ist eine leere Sequenz.

**Längenbeschränkung**

Die Länge von *Suchbegriff* ist auf 32.000 Byte beschränkt.

### Zurückgegebener Wert

Wenn *Suchbegriff* keine leere Sequenz und keine Zeichenfolge mit Nulllänge ist:

- Wird *Suchbegriff* bei Position *m* in *Quellenzeichenfolge* gefunden und ist *m*>1, dann ist der zurückgegebene Wert die Unterzeichenfolge, die in *Quellenzeichenfolge* bei Position 1 beginnt und bei Position *m* endet.
- Wird *Suchbegriff* bei Position 1 in *Quellenzeichenfolge* gefunden, ist der zurückgegebene Wert eine Zeichenfolge mit Nulllänge.
- Ist *Suchzeichenfolge* eine leere Sequenz oder eine Zeichenfolge mit Nulllänge, ist der zurückgegebene Wert eine Zeichenfolge mit Nulllänge.
- Wird *Suchbegriff* in *Quellenzeichenfolge* nicht gefunden, ist der zurückgegebene Wert eine Zeichenfolge mit Nulllänge.

Ist *Quellenzeichenfolge* eine leere Sequenz oder eine Zeichenfolge mit Nulllänge, ist der zurückgegebene Wert eine Zeichenfolge mit Nulllänge.

### Beispiel

Die folgende Funktion sucht die Zeichen vor 'ABC' in der Zeichenfolge 'DEFABCD' unter Verwendung der Standardsortierfolge.

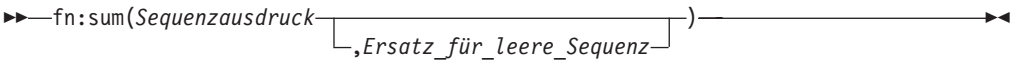
```
fn:substring-before('DEFABCD', 'ABC')
```

Der zurückgegebene Wert ist 'DEF'.

## Funktion 'sum'

Die Funktion fn:sum gibt die Summe der Werte in einer Sequenz zurück.

### Syntax




*Sequenzausdruck*

Eine Sequenz mit Elementen, die einen der folgenden atomaren Typen aufweisen, oder eine leere Sequenz.

- xs:float
- xs:double
- xs:decimal
- xs:integer
- xdt:untypedAtomic
- xdt:dayTimeDuration
- xdt:yearMonthDuration
- Einen Typ, der von einem der vorstehend aufgeführten Typen abgeleitet ist

Eingabeelemente vom Typ 'xdt:untypedAtomic' werden in den Typ 'xs:double' umgesetzt. Nach dieser expliziten Typumsetzung müssen alle Elemente in der Eingabesequenz durch Umstufung oder Subtypsubstitution in einen gemeinsamen Typ konvertierbar sein. Die Summe wird in diesem gemeinsamen Datentyp berechnet. Beispiel: Enthält die Eingabesequenz Elemente vom Typ 'money' (Geld), der von 'xs:decimal' abgeleitet ist, und vom Typ 'stockprice' (Aktienkurs), der von 'xs:float' abgeleitet ist, wird die Summe im Typ 'xs:float' berechnet.

*Ersatz_für_leere_Sequenz*
Der Wert, der zurückgegeben wird, wenn *Sequenzausdruck* eine leere Sequenz ist. *Ersatz_für_leere_Sequenz* kann einen der für *Sequenzausdruck* aufgeführten Datentypen aufweisen.

### Zurückgegebener Wert

Wenn *Sequenzausdruck* keine leere Sequenz ist, entspricht der zurückgegebene Wert der Summe der Werte in *Sequenzausdruck*. Der Datentyp des zurückgegebenen Werts entspricht dem Datentyp der Elemente in *Sequenzausdruck* oder dem Datentyp, in den die Elemente in *Sequenzausdruck* umgestuft werden.

Ist *Sequenzausdruck* eine leere Sequenz und ist *Ersatz_für_leere_Sequenz* nicht angegeben, gibt die Funktion 'fn:sum' den Wert 0.0E0 zurück. Ist *Sequenzausdruck* eine leere Sequenz und ist *Ersatz_für_leere_Sequenz* angegeben, gibt die Funktion 'fn:sum' den Wert für *Ersatz_für_leere_Sequenz* zurück.

### Beispiel

Die folgende Funktion gibt die Summe aus der Sequenz (500, 1.0E2, 40.5) zurück:

```
fn:sum((500, 1.0E2, 40.5))
```

Die Werte werden in den Datentyp 'xs:double' umgestuft. Die Funktion gibt den Wert 6.405E2 vom Typ 'xs:double' zurück, der als "640.5" serialisiert wird.

## Funktion 'timezone-from-date'

Die Funktion fn:timezone-from-date gibt die Zeitzonenkomponente eines Wertes vom Typ 'xs:date' zurück.

### Syntax

*Datumswert*
Der Datumswert, aus dem die Zeitzonenkomponente extrahiert werden soll.

*Datumswert* hat den Datentyp 'xs:date' oder ist eine leere Sequenz.

### Zurückgegebener Wert

Hat *Datumswert* den Typ 'xs:date' und eine explizite Zeitzonenkomponente, weist der zurückgegebene Wert den Typ 'xdt:dayTimeDuration' auf und liegt im Bereich von -PT14H Stunden bis PT14H einschließlich. Der Wert ist die Abweichung der Zeitzonenkomponete aus *Datumswert* von der Zeitzone UTC.

Verfügt *Datumswert* nicht über eine explizite Zeitzonenkomponente oder ist *Datumswert* eine leere Sequenz, ist der zurückgegebene Wert eine leere Sequenz.

### Beispiel

Die folgende Funktion gibt die Zeitzonenkomponente des folgenden Datumswerts zurück: 13. März 2007 in Zeitzone UTC-8:

```
fn:timezone-from-date(xs:date("2007-03-13-08:00"))
```

Der zurückgegebene Wert ist -PT8H.

## Funktion 'timezone-from-dateTime'

Die Funktion fn:timezone-from-dateTime gibt die Zeitzonenkomponente eines Wertes vom Typ 'xs:dateTime' zurück.

### Syntax

*Wert_für_Datum_und_Uhrzeit*
Der Wert für Datum und Uhrzeit, aus dem die Zeitzonenkomponente extrahiert werden soll.

*Wert_für_Datum_und_Uhrzeit* hat den Datentyp 'xs:dateTime' oder ist eine leere Sequenz.

### Zurückgegebener Wert

Hat *Wert_für_Datum_und_Uhrzeit* den Typ 'xs:dateTime' und eine explizite Zeitzonenkomponente, weist der zurückgegebene Wert den Typ 'xdt:dayTimeDuration' auf und liegt im Bereich von -PT14H bis PT14H einschließlich. Der Wert ist die Abweichung der Zeitzonenkomponente aus *Wert_für_Datum_und_Uhrzeit* von der Zeitzone UTC.

Verfügt *Wert_für_Datum_und_Uhrzeit* nicht über eine explizite Zeitzonenkomponente oder ist *Wert_für_Datum_und_Uhrzeit* eine leere Sequenz, ist der zurückgegebene Wert eine leere Sequenz.

### Beispiele

Die folgende Funktion gibt die Zeitzonenkomponente des folgenden Werts für Datum und Uhrzeit zurück: 30. Oktober 2005, 07:30 Uhr in Zeitzone UTC-8:

```
fn:timezone-from-dateTime(xs:dateTime("2005-10-30T07:30:00-08:00"))
```

Der zurückgegebene Wert ist -PT8H.

Die folgende Funktion gibt die Zeitzonenkomponente des folgenden Werts für Datum und Uhrzeit zurück: 1. Januar 2005, 14:30 Uhr in Zeitzone UTC+10:30:

```
fn:timezone-from-dateTime(xs:dateTime("2005-01-01T14:30:00+10:30"))
```

Der zurückgegebene Wert lautet PT10H30M.

## Funktion 'timezone-from-time'

Die Funktion fn:timezone-from-time gibt die Zeitzonenkomponente eines Wertes vom Typ 'xs:time' zurück.

### Syntax

*Zeitwert*
Der Zeitwert, aus dem die Zeitzonenkomponente extrahiert werden soll.

*Zeitwert* hat den Datentyp 'xs:time' oder ist eine leere Sequenz.

### Zurückgegebener Wert

Hat *Zeitwert* den Typ 'xs:time' und eine explizite Zeitzonenkomponente, weist der zurückgegebene Wert den Typ 'xdt:dayTimeDuration' auf und liegt im Bereich von -PT14H Stunden bis PT14H einschließlich. Der Wert ist die Abweichung der Zeitzonenkomponente aus *Zeitwert* von der Zeitzone UTC.

Verfügt *Zeitwert* nicht über eine explizite Zeitzonenkomponente oder ist *Zeitwert* eine leere Sequenz, ist der zurückgegebene Wert eine leere Sequenz.

### Beispiele

Die folgende Funktion gibt die Zeitzonenkomponente des folgenden Zeitwerts zurück: 12:00 Uhr in Zeitzone UTC-5:

```
fn:timezone-from-time(xs:time("12:00:00-05:00"))
```

Der zurückgegebene Wert ist -PT5H.

In der folgenden Funktion hat der Zeitwert 13:00 Uhr keine Zeitzonenkomponente:

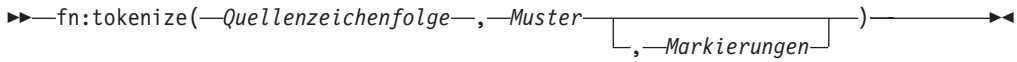
```
fn:timezone-from-time(xs:time("13:00:00"))
```

Der zurückgegebene Wert ist eine leere Sequenz.

## Funktion 'tokenize'

Die Funktion fn:tokenize unterteilt eine Zeichenfolge in eine Sequenz aus Unterzeichenfolgen.

### Syntax




*Quellenzeichenfolge*
Eine Zeichenfolge, die in eine Sequenz aus Unterzeichenfolgen unterteilt werden soll.

*Quellenzeichenfolge* ist ein Wert vom Typ 'xs:string' oder eine leere Sequenz.

*Muster* Der Begrenzer zwischen den Unterzeichenfolgen in *Quellenzeichenfolge*.

*Muster* ist ein Wert vom Datentyp 'xs:string', der einen regulären Ausdruck enthält. Ein regulärer Ausdruck ist eine Gruppe von Zeichen, Platzhalterzeichen und Operatoren, die eine Zeichenfolge oder Gruppe von Zeichenfolgen in einem Suchmuster definieren.

*Markierungen*
Ein Wert vom Typ 'xs:string', der einen der folgenden Werte enthalten kann, die den Abgleich von *Muster* mit den Zeichen in *Quellenzeichenfolge* steuern:

**s** Gibt an, dass der Punkt (.) im regulären Ausdruck einem beliebigen Zeichen entspricht, einschließlich des Zeilenvorschubzeichens (X'0A').

Wenn die Markierung 's' nicht angegeben ist, entspricht der Punkt jedem Zeichen mit Ausnahme des Zeilenvorschubzeichens (X'0A').

**m** Gibt an, dass das Winkelzeichen (^) dem Beginn einer Zeile entspricht (der Position nach einem Zeilenvorschubzeichen), und dass das Dollarzeichen ($) dem Ende einer Zeile entspricht (der Position vor einem neuen Zeilenvorschubzeichen).

Wenn die Markierung 'm' nicht angegeben ist, entspricht das Winkelzeichen (^) dem Beginn einer Zeichenfolge, und das Dollarzeichen ($) entspricht dem Ende der Zeichenfolge.

**i** Gibt an, dass der Abgleich nicht von der Groß-/Kleinschreibung abhängt.

Wenn die Markierung 'i' nicht angegeben ist, erfolgt der Abgleich unter Beachtung der Groß-/Kleinschreibung.

**x** Gibt an, dass Leerzeichen innerhalb von *Muster* ignoriert werden.

Wenn die Markierung 'x' nicht angegeben ist, werden Leerzeichen beim Abgleich verwendet.

## Zurückgegebener Wert

Wenn *Quellenzeichenfolge* keine leere Sequenz und keine Zeichenfolge mit Nulllänge ist, entspricht der zurückgegebene Wert der Sequenz, die sich ergibt, wenn die folgenden Operationen für *Quellenzeichenfolge* ausgeführt werden:

- *Quellenzeichenfolge* wird auf Zeichen durchsucht, die mit *Muster* übereinstimmen.
- Wenn *Muster* zwei oder mehr alternative Zeichengruppen enthält, gilt die erste Zeichengruppe in *Muster*, die mit den Zeichen in *Quellenzeichenfolge* übereinstimmt, als das übereinstimmende Muster.
- Jede Gruppe von Zeichen, die nicht mit *Muster* übereinstimmt, wird als Element in der Ergebnissequenz verwendet.
- Wenn *Muster* mit Zeichen am Anfang von *Quellenzeichenfolge* übereinstimmt, dann ist das erste Element in der zurückgegebenen Sequenz eine Zeichenfolge mit Nulllänge.
- Wenn in *Quellenzeichenfolge* zwei aufeinanderfolgende Übereinstimmungen mit *Muster* gefunden werden, wird der Sequenz eine Zeichenfolge mit Nulllänge hinzugefügt.
- Wenn *Muster* mit Zeichen am Ende von *Quellenzeichenfolge* übereinstimmt, dann ist das letzte Element in der zurückgegebenen Sequenz eine Zeichenfolge mit Nulllänge.

Wird *Muster* in *Quellenzeichenfolge* nicht gefunden, wird ein Fehler zurückgegeben.

Ist *Quellenzeichenfolge* eine leere Sequenz oder eine Zeichenfolge mit Nulllänge, ist das Ergebnis die leere Sequenz.

## Beispiel

Die nachstehende Funktion erstellt eine Sequenz aus der Zeichenfolge "Tokenize this sentence, please.". "\s+" ist ein regulärer Ausdruck, der für mindestens ein Leerzeichen steht.

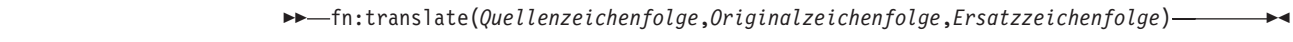
```
fn:tokenize("Tokenize this sentence, please.", "\s+")
```

Der zurückgegebene Wert ist die Zeichenfolge ("Tokenize", "this", "sentence,", "please.").

## Funktion 'translate'

Die Funktion fn:translate ersetzt ausgewählte Zeichen in einer Zeichenfolge durch Ersetzungszeichen.

### Syntax




*Quellenzeichenfolge*
Die Zeichenfolge, in der Zeichen konvertiert werden sollen.

*Quellenzeichenfolge* hat den Datentyp 'xs:string' oder ist eine leere Sequenz.

*Originalzeichenfolge*
Eine Zeichenfolge mit den Zeichen, die konvertiert werden können.

*Originalzeichenfolge* hat den Datentyp 'xs:string'.

*Ersatzzeichenfolge*
Eine Zeichenfolge mit den Zeichen, die die Zeichen in *Originalzeichenfolge* ersetzen sollen.

*Ersatzzeichenfolge* hat den Datentyp 'xs:string'.

Wenn die Länge von *Ersatzzeichenfolge* größer ist als die Länge von *Originalzeichenfolge*, werden die zusätzlichen Zeichen in *Ersatzzeichenfolge* ignoriert.

**Längenbeschränkung**

Die Länge von *Originalzeichenfolge* und *Ersatzzeichenfolge* ist auf 32.000 Byte beschränkt.

### Zurückgegebener Wert

Wenn *Quellenzeichenfolge* keine leere Sequenz ist, entspricht der zurückgegebene Wert dem Wert vom Typ 'xs:string', der sich ergibt, wenn die folgenden Operationen ausgeführt werden:

- Für jedes Zeichen in *Quellenzeichenfolge*, das in *Originalzeichenfolge* vorkommt, wird das entsprechende Zeichen in *Quellenzeichenfolge* durch das Zeichen in *Ersatzzeichenfolge* ersetzt, das an der gleichen Position wie das Zeichen in *Originalzeichenfolge* auftritt. Der Zeichenabgleich erfolgt über einen binären Vergleich.

  Wenn die Länge von *Originalzeichenfolge* größer ist als die Länge von *Ersatzzeichenfolge*, wird jedes Zeichen in *Quellenzeichenfolge*, das zwar in *Originalzeichenfolge* vorkommt, dessen Position in *Originalzeichenfolge* jedoch keine entsprechende Position in *Ersatzzeichenfolge* aufweist, gelöscht.

  Wenn ein Zeichen mehr als einmal in *Originalzeichenfolge* vorkommt, legt die Position des ersten Auftretens des Zeichens in *Originalzeichenfolge* das zu verwendende Zeichen in *Ersatzzeichenfolge* fest.
- Für jedes Zeichen in *Quellenzeichenfolge*, das nicht in *Originalzeichenfolge* auftritt, wird das entsprechende Zeichen unverändert gelassen.

Ist *Quellenzeichenfolge* eine leere Sequenz, wird eine Zeichenfolge mit Nulllänge zurückgegeben.

### Beispiele

Die folgende Funktion gibt die Zeichenfolge zurück, die sich ergibt, wenn in der Zeichenfolge 'Test literal' 'e' durch 'o' und 'l' durch 'm' ersetzt wird.

```
fn:translate('Test literal','el','om')
```

Der zurückgegebene Wert lautet 'Tost mitoram'.

Die folgende Funktion gibt die Zeichenfolge zurück, die sich ergibt, wenn im Zeichenfolgeliteral 'Another test literal' 'A' durch B, 't' durch 'f', 'e' durch 'i' und 'r' durch 'm' ersetzt wird.

```
fn:translate('Another test literal', 'Ater', 'Bfim')
```

Der zurückgegebene Wert lautet 'Bnofhim fisf lifimal'.

## Funktion 'true'

Die Funktion fn:true gibt den Wert 'true' (wahr) vom Typ 'xs:boolean' zurück.

### Syntax

### Zurückgegebener Wert

Der zurückgegebene Wert ist der Wert 'true' (wahr) vom Typ xs:boolean.

### Beispiel

Verwenden Sie die Funktion 'true', um den Wert 'true' zurückzugeben.

```
fn:true()
```

Der Wert 'true' (wahr) wird zurückgegeben.

## Funktion 'unordered'

Die Funktion fn:unordered gibt die Elemente in einer Sequenz in nicht deterministischer Reihenfolge zurück.

### Syntax

*Sequenzausdruck*
Eine beliebige Sequenz, einschließlich einer leeren Sequenz.

### Zurückgegebener Wert

Der zurückgegebene Wert entspricht den Elementen in *Sequenzausdruck* in nicht deterministischer Reihenfolge. Diese Funktion hilft dem Abfrageoptimierungs-

programm bei der Auswahl von Zugriffspfaden, die nicht von der Reihenfolge der Elemente in der Sequenz abhängen.

### Beispiel

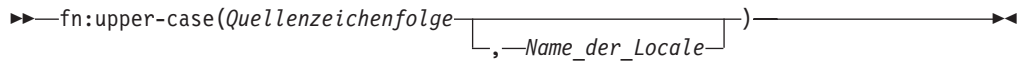
Die folgende Funktion gibt die Elemente in der Sequenz (1,2,3) in nicht deterministischer Reihenfolge zurück.

```
fn:unordered((1,2,3))
```

## Funktion 'upper-case'

Die Funktion fn:upper-case setzt eine Zeichenfolge in Großbuchstaben um.

### Syntax




*Quellenzeichenfolge*
Die Zeichenfolge, die in Großbuchstaben konvertiert werden soll.

*Quellenzeichenfolge* hat den Datentyp 'xs:string' oder ist eine leere Sequenz.

*Name_der_Locale*
Eine Zeichenfolge mit den Ländereinstellungen (Locale), die für die Operation in Großbuchstaben verwendet werden soll.

*Name_der_Locale* hat den Datentyp 'xs:string' oder ist eine leere Sequenz. Ist *Name_der_Locale* keine leere Sequenz, ist der Wert von *Name_der_Locale* nicht von der Groß-/Kleinschreibung abhängig und muss eine gültige Locale oder eine Zeichenfolge mit Nulllänge sein.

### Zurückgegebener Wert

Ist *Quellenzeichenfolge* keine leere Zeichenfolge, dann ist der zurückgegebene Wert *Quellenzeichenfolge*, wobei jedes Zeichen in den ihm entsprechenden Großbuchstaben konvertiert ist. Wird für *Name_der_Locale* kein Wert angegeben oder ist der Wert eine leere Sequenz oder eine Zeichenfolge mit Nulllänge, dann werden die Regeln für Großschreibung gemäß Unicode-Standard verwendet. Ansonsten werden die Regeln für Großschreibung für die angegebene Locale verwendet. Jedes Zeichen, das über keine Entsprechung in Großschreibung verfügt, wird in seiner ursprünglichen Form in den zurückgegebenen Wert eingeschlossen.

Ist *Quellenzeichenfolge* eine leere Sequenz, ist der zurückgegebene Wert eine Zeichenfolge mit Nulllänge.

### Beispiele

Die folgende Funktion konvertiert die Zeichenfolge 'Test literal 1' in Großbuchstaben.

```
fn:upper-case('Test literal 1')
```

Der zurückgegebene Wert lautet 'TEST LITERAL 1'.

In der folgenden Funktion wird die türkische Locale (tr_TR) angegeben, und der Buchstabe ″i″ und der numerische Zeichenverweis ı (der Zeichenverweis für das lateinische kleine i ohne Punkt) werden konvertiert:

```
fn:upper-case("i&#x131;", "tr_TR")
```

Der zurückgegebene Wert besteht aus zwei Zeichen - dem von İ dargestellten Zeichen (lateinisches großes I mit Punkt) und dem Buchstaben ″I″. Bei der türkischen Locale wird der Buchstabe ″i″ in das von İ dargestellte Zeichen (lateinisches großes I mit Punkt) konvertiert, und das von ı dargestellte Zeichen (lateinisches kleines i ohne Punkt) wird in ″I″ konvertiert.

In der folgenden Funktion wird keine Locale angegeben, und es werden zwei Zeichen anhand der im Unicode-Standard definierten Regeln in die Großschreibung konvertiert:

```
fn:upper-case("&#x131;i")
```

Die Funktion gibt die Zeichen ″II″ zurück. Die Funktion fn:upper-case konvertiert sowohl den von ı dargestellten Kleinbuchstaben als auch den Buchstaben ″i″ in den Großbuchstaben ″I″.

## Funktion 'xmlcolumn'

Die Funktion db2-fn:xmlcolumn ruft eine Sequenz aus einer Spalte in der zum jeweiligen Zeitpunkt verbundenen DB2-Datenbank ab.

### Syntax

*Zeichenfolgeliteral*
Gibt den Namen der Spalte an, aus der die Sequenz abgerufen wird. Der Spaltenname muss durch den Namen einer Tabelle oder Sicht oder einen Aliasnamen qualifiziert sein und auf eine Spalte mit dem Datentyp XML verweisen. Der SQL-Schemaname ist optional. Wird kein Schemaname angegeben, wird das Sonderregister CURRENT SCHEMA als implizites Qualifikationsmerkmal für die Tabelle oder Sicht verwendet. Das *Zeichenfolgeliteral* ist von der Groß-/Kleinschreibung abhängig. Für *Zeichenfolgeliteral* müssen genau die Zeichen verwendet werden, die den Spaltennamen in der Datenbank identifizieren.

### Zurückgegebener Wert

Der zurückgegebene Wert ist eine Zeichenfolge, bei der es sich um die Verknüpfung der XML-Werte, die ungleich null sind, aus der Spalte handelt, die durch *Zeichenfolgeliteral* angegeben wird. Enthält die Tabelle oder Sicht keine Zeilen, gibt die Funktion 'db2-fn:xmlcolumn' eine leere Sequenz zurück.

Die Anzahl der Elemente in der Sequenz, die von der Funktion 'db2-fn:xmlcolumn' zurückgegeben wird, kann sich von der Anzahl der Zeilen in der angegebenen Tabelle oder Sicht unterscheiden, da einige dieser Zeilen unter Umständen Nullwerte oder Sequenzen mit mehreren Elementen enthalten.

Die Funktion 'db2-fn:xmlcolumn' ist mit der Funktion 'db2-fn:sqlquery' verwandt, und beide Funktionen können das gleiche Ergebnis generieren. Die Argumente der beiden Funktionen unterschieden sich jedoch im Hinblick auf ihre jeweilige Abhängigkeit von der Groß-/Kleinschreibung. Das Argument in der Funktion 'db2-fn:xmlcolumn' wird von XQuery verarbeitet, sodass bei diesem Argument die Groß-/Kleinschreibung beachtet werden muss. Da Tabellennamen und Spalten-

namen in einer DB2-Datenbank standardmäßig in Großbuchstaben vorliegen, werden für das Argument von 'db2-fn:xmlcolumn' normalerweise auch Großbuchstaben verwendet. Das Argument der Funktion 'db2-fn:sqlquery' wird von SQL verarbeitet, sodass Kennungen automatisch in Großbuchstaben konvertiert werden.

Die folgenden Funktionsaufrufe sind äquivalent und geben dasselbe Ergebnis zurück:

```
db2-fn:xmlcolumn('SQL-SCHEMA.TABELLENNAME.SPALTENNAME')
db2-fn:sqlquery('select spaltenname from sql-schema.tabellename')
```

### Beispiele

**Beispiel für die Rückgabe einer Sequenz aus Dokumenten:** Die folgende Funktion gibt eine Sequenz aus XML-Dokumenten zurück, die in der XML-Spalte DESCRIPTION in der Tabelle PRODUCT gespeichert sind. Die Tabelle befindet sich in diesem Beispiel im SQL-Schema SAMPLE.

```
db2-fn:xmlcolumn('SAMPLE.PRODUCT.DESCRIPTION')
```

**Beispiel für die Verwendung eines impliziten SQL-Schemas:** Im folgenden Beispiel ist das Sonderregister CURRENT SCHEMA in einer DB2-Datenbank auf SAMPLE gesetzt, sodass die Funktion dieselben Ergebnisse wie im vorherigen Beispiel zurückgibt:

```
db2-fn:xmlcolumn('PRODUCT.DESCRIPTION')
```

**Beispiel für die Verwendung eines begrenzten SQL-Bezeichners:** Die nachstehende Funktion gibt eine Sequenz aus Dokumenten zurück, die in der Spalte ″Thesis″ der Tabelle ″Student″ gespeichert sind, wobei davon ausgegangen wird, dass sich die Tabelle in dem Schema befindet, das momentan dem Sonderregister CURRENT SCHEMA zugeordnet ist. Da der Tabellenname und der Spaltenname Kleinbuchstaben enthalten, gibt es zwei Möglichkeiten, diese Namen im Argument des Zeichenfolgeliterals der Funktion 'db2-fn:xmlcolumn' anzugeben:

- Angabe als begrenzte SQL-Bezeichner (in doppelte Anführungszeichen eingeschlossen):

  ```
  db2-fn:xmlcolumn('"Student"."Thesis"')
  ```

- Angabe als Zeichenfolge ohne Angabe, dass es sich um begrenzte SQL-Bezeichner handelt:

  ```
  db2-fn:xmlcolumn('Student.Thesis')
  ```

Im Gegensatz dazu müssen zur Angabe derselben Tabellen- und Spalteninformationen in der Funktion 'db2-fn:sqlquery' die begrenzten SQL-Bezeichner wie folgt verwendet werden:

```
db2-fn:sqlquery('select "Thesis" from "Student"')
```

## Funktion 'year-from-date'

Die Funktion fn:year-from-date gibt die Jahreskomponente eines Wertes vom Typ 'xs:date' zurück.

### Syntax

*Datumswert*
Der Datumswert, aus dem die Jahreskomponente extrahiert werden soll.

*Datumswert* hat den Datentyp 'xs:date' oder ist eine leere Sequenz.

### Zurückgegebener Wert

Hat *Datumswert* den Typ 'xs:date', weist der zurückgegebene Wert den Typ 'xs:integer' auf. Der Wert ist die Jahreskomponente von *Datumswert* und stets ein positiver Wert.

Ist *Datumswert* eine leere Sequenz, ist der zurückgegebene Wert eine leere Sequenz.

### Beispiel

Die folgende Funktion gibt die Jahreskomponente des folgenden Datumswerts zurück: 29. Oktober 2005:

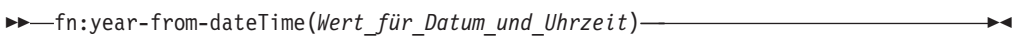
```
fn:year-from-date(xs:date("2005-10-29"))
```

Der zurückgegebene Wert lautet '2005'.

## Funktion 'year-from-dateTime'

Die Funktion fn:year-from-dateTime gibt die Jahreskomponente eines Wertes vom Typ 'xs:dateTime' zurück.

### Syntax




*Wert_für_Datum_und_Uhrzeit*
Der Wert für Datum und Uhrzeit, aus dem die Jahreskomponente extrahiert werden soll.

*Wert_für_Datum_und_Uhrzeit* hat den Datentyp 'xs:dateTime' oder ist eine leere Sequenz.

### Zurückgegebener Wert

Hat *Wert_für_Datum_und_Uhrzeit* den Typ 'xs:dateTime', weist der zurückgegebene Wert den Typ 'xs:integer' auf. Der Wert ist die Jahreskomponente von *Wert_für_Datum_und_Uhrzeit* und stets ein positiver Wert.

Ist *Wert_für_Datum_und_Uhrzeit* eine leere Sequenz, ist der zurückgegebene Wert eine leere Sequenz.

### Beispiel

Die folgende Funktion gibt die Jahreskomponente des folgenden Werts für Datum und Uhrzeit zurück: 29. Oktober 2005, 08:00 Uhr in Zeitzone UTC-8:

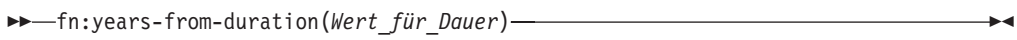
```
fn:year-from-dateTime(xs:dateTime("2005-10-29T08:00:00-08:00"))
```

Der zurückgegebene Wert lautet '2005'.

## Funktion 'years-from-duration'

Die Funktion fn:years-from-duration gibt die Jahreskomponente einer Zeitdauer zurück.

### Syntax




*Wert_für_Dauer*
Der Wert für Dauer, aus dem die Jahreskomponente extrahiert werden soll.

*Wert_für_Dauer* ist eine leere Sequenz oder ein Wert, der einen der folgenden Datentypen aufweist: xdt:dayTimeDuration, xs:duration oder xdt:yearMonthDuration.

### Zurückgegebener Wert

Der zurückgegebene Wert hängt vom Datentyp von *Wert_für_Dauer* ab:

- Hat *Wert_für_Dauer* den Typ 'xdt:yearMonthDuration' oder 'xs:duration', weist der zurückgegebene Wert den Typ 'xs:integer' auf. Der Wert ist die Jahreskomponente von *Wert_für_Dauer*, die als 'xdt:yearMonthDuration' umgesetzt ist. Der Wert ist negativ, wenn *Wert_für_Dauer* negativ ist.
- Hat *Wert_für_Dauer* den Typ 'xs:dayTimeDuration', weist der zurückgegebene Wert den Typ 'xs:integer' auf und ist 0.
- Ist *Wert_für_Dauer* eine leere Sequenz, ist der zurückgegebene Wert eine leere Sequenz.

Die als 'xdt:yearMonthDuration' umgesetzte Jahreskomponente aus *Wert_für_Dauer* ist die ganzzahlige Anzahl der Jahre, berechnet aus der Gesamtzahl der Monate von *Wert_für_Dauer*, die als 'xdt:yearMonthDuration' umgesetzt ist, dividiert durch 12.

### Beispiele

Die folgende Funktion gibt die Jahreskomponente der Dauer von -4 Jahren, -11 Monaten und -320 Tagen zurück:

```
fn:years-from-duration(xs:duration("-P4Y11M320D"))
```

Der zurückgegebene Wert ist -4.

Die folgende Funktion gibt die Jahreskomponente der Dauer von 9 Jahren und 13 Monaten zurück:

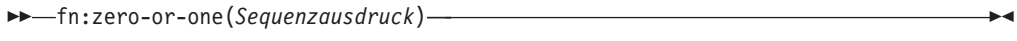
```
fn:years-from-duration(xdt:yearMonthDuration("P9Y13M"))
```

Der zurückgegebene Wert ist 10. Bei der Berechnung der Gesamtzahl der Jahre in der Dauer werden die 13 Monate in 1 Jahr und 1 Monat umgewandelt. Die Dauer entspricht P10Y1M. Hierbei gibt die Jahreskomponente 10 Jahre an.

## Funktion 'zero-or-one'

Die Funktion fn:zero-or-one gibt ihr Argument zurück, wenn es entweder ein (1) Element enthält oder eine leere Sequenz ist.

### Syntax



*Sequenzausdruck*
Eine beliebige Sequenz, einschließlich einer leeren Sequenz.

## Zurückgegebener Wert

Wenn *Sequenzausdruck* ein einzelnes Element enthält oder eine leere Sequenz ist, wird *Sequenzausdruck* zurückgegeben. Andernfalls wird ein Fehler zurückgegeben.

## Beispiel

In dem folgenden Beispiel wird die Funktion 'fn:zero-or-one' verwendet, um zu ermitteln, ob die Sequenz in der Variablen '$seq' ein einzelnes Element enthält oder leer ist.

```
let $seq := (5,10)
return fn:zero-or-one($seq)
```

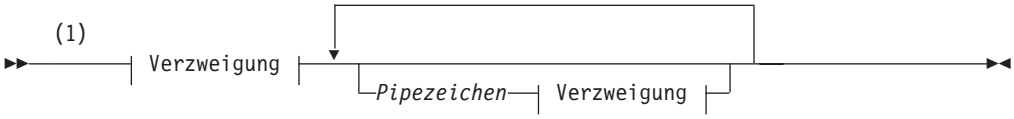
Es wird ein Fehler zurückgegeben, da die Sequenz zwei Elemente enthält.

# Kapitel 6. Reguläre Ausdrücke

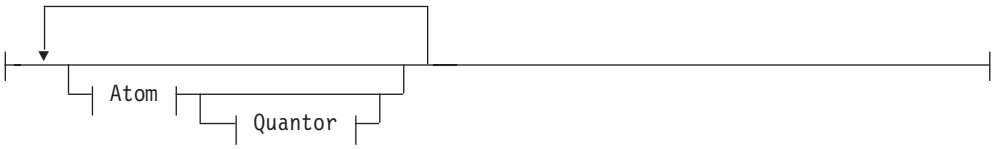
Ein regulärer Ausdruck ist eine Sequenz aus Zeichen, die als Muster für das Abgleichen und Bearbeiten von Zeichenfolgen fungieren. Reguläre Ausdrücke werden in den folgenden XQuery-Funktionen verwendet: 'fn:matches', 'fn:replace' und 'fn:tokenize'. Die Unterstützung regulärer Ausdrücke durch DB2 XQuery basiert auf der Unterstützung regulärer Ausdrücke durch das XML-Schema gemäß Definition in W3C Recommendation *XML Schema Part 2: Datatypes Second Edition* mit Erweiterungen gemäß Definition von W3C Recommendation *XQuery 1.0 and XPath 2.0 Functions and Operators*.

## Syntax

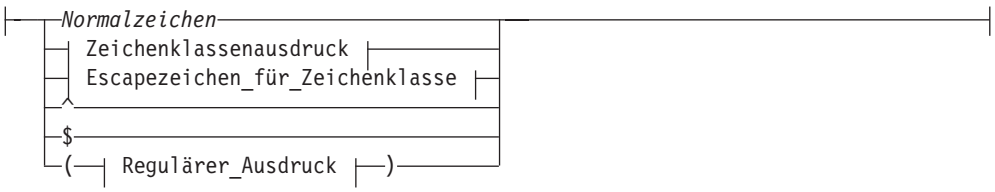
### Regulärer Ausdruck



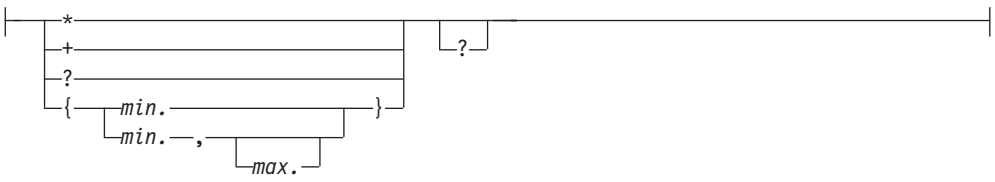

### Verzweigung:




### Atom:

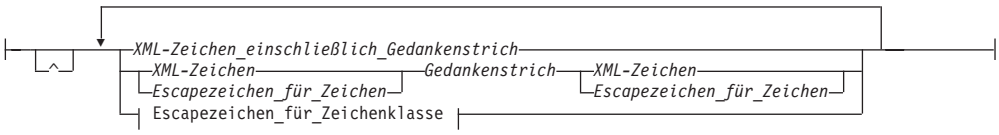



### Quantor:

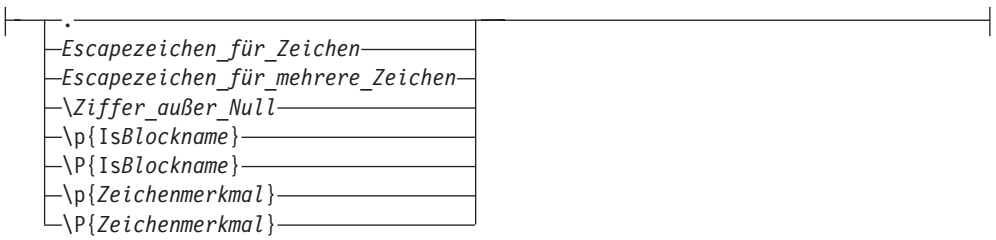



### Zeichenklassenausdruck:

**Zeichengruppe:**




**Escapezeichen_für_Zeichenklasse:**




**Anmerkungen:**

1 Die Syntax für reguläre Ausdrücke stellt den Inhalt eines Zeichenfolgeliterals dar, das keine Leerzeichen enthalten darf, mit Ausnahme von Leerzeichen, die in ihrer spezifischen Bedeutung als Zeichen in einem Muster verwendet werden. Leerschritte oder Abschnitte zwischen Syntaxelementen in diesem Diagramm sind nicht als Leerzeichen zu verstehen.

## Regulärer_Ausdruck

Ein regulärer Ausdruck enthält mindestens eine Verzweigung. Verzweigungen werden durch Pipes (|) getrennt, um anzugeben, dass jede Verzweigung ein anderes Muster darstellt.

*Pipezeichen*
Ein Pipezeichen (|) trennt unterschiedliche Verzweigungen in einem regulären Ausdruck.

**Verzweigung**
Eine Verzweigung besteht aus null oder mehr Atomen, wobei für jedes Atom ein optionaler Quantor zulässig ist.

## Atom

Ein Atom ist entweder ein Normalzeichen, ein Zeichenklassenausdruck, ein Escapezeichen für eine Zeichenklasse oder ein regulärer Ausdruck in runden Klammern.

*Normalzeichen*
Ein gültiges XML-Zeichen, das nicht zu den in Tabelle 37 auf Seite 235 aufgeführten Metazeichen gehört.

**^** Die Verwendung des Winkelzeichens (^) am Anfang einer Verzweigung gibt an, dass das Muster vom Anfang der Zeichenfolge an übereinstimmen muss.

**$** Die Verwendung des Dollarzeichens ($) am Ende einer Verzweigung gibt an, dass das Muster vom Ende der Zeichenfolge an übereinstimmen muss.

## Quantor

Der Quantor gibt die Wiederholung eines Atoms in einem regulären Ausdruck an. Standardmäßig stimmt ein Quantor weitestgehend mit einer Zielzeichenfolge überein, wobei ein so genannter *Greedy-Algorithmus* verwendet wird. Beispiel: Der reguläre Ausdruck `'A.*A'` entspricht der gesamten Zeichenfolge 'ABACADA', da die Unterzeichenfolge zwischen den erforderlichen äußeren Zeichen 'A' die Anforderung für ein beliebiges Zeichen in beliebiger Anzahl erfüllt. Der standardmäßige Greedy-Algorithmus kann durch Angabe eines Fragezeichens ( ? ) nach dem Quantor geändert werden. Das Fragezeichen gibt an, dass bei der Mustererkennung ein *Reluctant-Algorithmus* verwendet wird, der mit der nächsten kürzesten Unterzeichenfolge von links nach rechts in der Zielzeichenfolge übereinstimmt, die den regulären Ausdruck erfüllt. Beispiel: Der reguläre Ausdruck `'A.*?A'` entspricht den Unterzeichenfolgen 'ABA' und 'ADA' und nicht der gesamten Zeichenfolge 'ABACADA'. Zeichen einer Unterzeichenfolge, die bei Verwendung des Reluctant-Algorithmus mit einem regulären Ausdruck übereinstimmen, werden für weitere Abgleiche nicht mehr herangezogen. Aus diesem Grund gilt 'ACA' in vorstehendem Beispiel nicht als Übereinstimmung. Der Reluctant-Algorithmus hat zusammen mit der Funktion 'fn:replace' den größten Nutzen, da diese Funktion Übereinstimmungen und Ersetzungen von links nach rechts verarbeitet.

Beispiel: Wird der Greedy-Algorithmus in der Funktion `fn:replace("nonsensical","n(.*)s","mus")` verwendet, um die Zeichenfolge, die mit ″n″ beginnt und mit ″s″ endet, durch die Zeichenfolge ″mus″ zu ersetzen, wird der Wert 'musical' zurückgegeben. Die ursprüngliche Zeichenfolge enthielt die Unterzeichenfolgen ″nons″ und ″ns″, die dem Muster bei einer Suche nach der nächsten Übereinstimmung von links nach rechts ebenfalls entsprachen. Der Greedy-Algorithmus funktionierte bei diesen Übereinstimmungen jedoch nicht, da er längere einschließende Übereinstimmungen fand.

Wird der Reluctant-Algorithmus für dieselbe Zeichenfolge in der Funktion `fn:replace("nonsensical","n(.*?)s","mus")` verwendet, sieht das Ergebnis anders aus. Der zurückgegebene Wert lautet ″musemusical″. In diesem Fall erfolgten innerhalb der Zeichenfolge zwei Ersetzungen. Bei der ersten Übereinstimmung wurde ″nons″ durch ″mus″ ersetzt, und bei der zweiten Übereinstimmung wurde ″ns″ durch ″mus″ ersetzt.

Ein weiteres Beispiel: Wird der Greedy-Algorithmus verwendet, um in der Funktion `fn:replace("AbrAcAdAbrA","A(.*)A","X$1X")` das Zeichen 'A', das eine beliebige Anzahl an Zeichen einschließt, durch das Zeichen X zu ersetzen, das dieselben Zeichen einschließt, lautet der zurückgegebene Wert ″XbrAcAdAbrX″. Die ursprüngliche Zeichenfolge enthielt die Unterzeichenfolgen ″AbrA″ und ″AdA″, die dem Muster bei einer Suche nach der nächsten Übereinstimmung von links nach rechts ebenfalls entsprachen. Der Greedy-Algorithmus funktionierte bei diesen Übereinstimmungen jedoch nicht, da er längere einschließende Übereinstimmungen fand.

Wird der Reluctant-Algorithmus für dieselbe Zeichenfolge in der Funktion `fn:replace("AbrAcAdAbrA","A(.*?)A","X$1X")` verwendet, sieht das Ergebnis anders aus. Der zurückgegebene Wert lautet ″XbrXcXdXbrA″. In diesem Fall erfolgte innerhalb der Zeichenfolge zwei Ersetzungen: die erste für ″AbrA″ und die zweite für ″AdA″. Das abschließende ″A″ in der Zeichenfolge wurde nicht ersetzt, da der Reluctant-Algorithmus sämtliche davor stehenden Zeichen ″A″ für andere Übereinstimmungen innerhalb der Zeichenfolge verwendete. Andere Unterzeichenfolgen, die innerhalb der ursprünglichen Zeichenfolge mit dem Zeichen ″A″ beginnen und enden, wie beispielsweise ″AcA″, ″AcAdA″, ″AdAbrA″ und ″AbrA″, wer-

den nicht berücksichtigt, da der Reluctant-Algorithmus diese Zeichen als bereits verwendet betrachtet, nachdem sie zu einer Übereinstimmung mit dem Muster gehörten.

* Stimmt mit dem Atom überhaupt nicht oder mehrmals überein. Entspricht dem Quantor `{0, }`.

\+ Stimmt mindestens einmal mit dem Atom überein. Entspricht dem Quantor `{1, }`.

**?** Stimmt mit dem Atom überhaupt nicht oder einmal überein. Entspricht dem Quantor `{0, 1}`. Folgt dieser Quantor auf einen anderen, gibt er an, dass der Reluctant- und nicht der Greedy-Algorithmus verwendet wird.

*min.*
Stimmt mindestens *min.* Male mit dem Atom überein. Der Wert für *min.* muss eine positive ganze Zahl sein.

- {*min.*} stimmt genau *min.* Male mit dem Atom überein.
- {*min.*, } stimmt mindestens *min.* Male mit dem Atom überein.

*max.*
Stimmt höchstens *max.* Male mit dem Atom überein. Der Wert für *max.* muss eine positive ganze Zahl größer-gleich *min.* sein.

- {0, *max.*} stimmt höchstens *min.* Male mit dem Atom überein.
- {0, 0} stimmt lediglich mit einer leeren Zeichenfolge überein.

## Zeichengruppe

^ Gibt das Komplement der Menge der Zeichen an, die vom Rest der Zeichengruppe definiert sind.

*Gedankenstrich*
Ein Gedankenstrich (-) trennt zwei Zeichen, die die äußeren Zeichen in einem Zeichenbereich definieren. Ein Zeichenbereich im Format `s-e` ist die Gruppe von UCS2-Codepunkten, die größer-gleich `s` und kleiner-gleich `e` sind, sodass gilt:

- `s` ist kein Backslash-Zeichen (\).
- Ist `s` das erste Zeichen in einer Zeichengruppe, dann ist es kein Winkelzeichen (^).
- `e` ist kein Backslash-Zeichen (\) und keine linke eckige Klammer ([).
- Der Codepunkt von 'e' ist größer als der Codepunkt von `s`.

*XML-Zeichen_einschließlich_Gedankenstrich*
Ein einzelnes Zeichen aus der Gruppe der gültigen XML-Zeichen mit Ausnahme des Backslash-Zeichens (\) und der eckigen Klammern ([]), aber einschließlich des Gedankenstrichs (-). Der Gedankenstrich ist nur am Anfang oder am Ende einer Zeichengruppe als Zeichen gültig. Das Winkelzeichen (^) am Anfang einer Zeichengruppe gibt das Komplement der Gruppe an. An allen anderen Position in der Gruppe entspricht das Winkelzeichen lediglich einem Winkelzeichen. *XML-Zeichen_einschließlich_Gedankenstrich* kann alle Zeichen enthalten, die mit dem regulären Ausdruck `[^\#5B#5D]` übereinstimmen.

*XML-Zeichen*
Ein einzelnes Zeichen aus der Gruppe der gültigen XML-Zeichen mit Ausnahme des Backslash-Zeichens (\), der eckigen Klammern ([]) und des Gedankenstrichs (-). Der Gedankenstrich ist nur am Anfang oder am Ende einer Zeichengruppe als Zeichen gültig. Das Winkelzeichen (^) am Anfang einer Zeichengruppe gibt das Komplement der Gruppe an. An allen anderen Posi-

tion in der Gruppe entspricht das Winkelzeichen lediglich einem Winkelzeichen. *XML-Zeichen* kann alle Zeichen enthalten, die mit dem regulären Ausdruck `[^\#2D#5B#5D]` übereinstimmen.

*Escapezeichen_für_Zeichen*
Ein Backslash-Zeichen, gefolgt von einem einzelnen Metazeichen, einem Zeilenvorschubzeichen, einem Rückführzeichen oder einem Tabulatorzeichen. Für den Abgleich muss den in Tabelle 37 aufgeführten Zeichen in einem regulären Ausdruck ein Escapezeichen vorangestellt werden.

*Tabelle 37. Gültige Escapezeichen für Metazeichen*

| Escapezeichen für Zeichen | Dargestellte Zeichen | Beschreibung |
|---|---|---|
| \n | #x0A | Zeilenvorschubzeichen |
| \r | #x0D | Rückführzeichen |
| \t | #x09 | Tabulatorzeichen |
| \\ | \ | Backslash-Zeichen |
| \\| | \| | Pipe |
| \. | . | Punkt |
| \- | - | Gedankenstrich |
| \^ | ^ | Winkelzeichen |
| \? | ? | Fragezeichen |
| \$ | $ | Dollarzeichen |
| * | * | Stern |
| \+ | + | Pluszeichen |
| \{ | { | Linke geschweifte Klammer |
| \} | } | Rechte geschweifte Klammer |
| \( | ( | Linke runde Klammer |
| \) | ) | Rechte runde Klammer |
| \[ | [ | Linke eckige Klammer |
| \] | ] | Rechte eckige Klammer |

## Escapezeichen_für_Zeichenklasse

**.** Der Punkt ( . ) stimmt mit allen Zeichen außer dem Zeilenvorschub- und dem Rückführzeichen überein. Der Punkt entspricht dem Ausdruck `[^\n\r]`.

*\Ziffer_außer_Null*
Gibt einen Rückverweis an, der mit der Zeichenfolge übereinstimmt, mit der ein Unterausdruck (in runden Klammern) an der Position *Ziffer_außer_Null* im regulären Ausdruck übereinstimmte. Der Wert für *Ziffer_außer_Null* muss im Bereich von 1 bis 9 liegen. Es kann auf die ersten neun Unterausdrücke verwiesen werden.

**Anmerkung:** Für zukünftige Aufwärtskompatibilität sollte ein Rückverweis, auf den eine Ziffer folgt, in runde Klammern eingeschlossen werden. Beispiel: Ein Rückverweis auf den ersten Unterausdruck, auf den die Ziffer 3 folgt, sollte als `(/1)3` und nicht als `/13` angegeben werden, auch wenn beide Formate derzeit noch dasselbe Ergebnis liefern.

**P**{**Is***Blockname*}
Gibt das Komplement eines Bereichs von Unicode-Codepunkten an. Der Bereich wird durch *Blockname* identifiziert (siehe Liste in *XML Schema Part 2: Datatypes Second Edition*).

**p**{**Is***Blockname*}
Gibt ein Zeichen in einem bestimmten Bereich von Unicode-Codepunkten an. Der Bereich wird durch *Blockname* identifiziert (siehe Liste in *XML Schema Part 2: Datatypes Second Edition*).

*Escapezeichen_für_Zeichen*
Ein Backslash-Zeichen, gefolgt von einem einzelnen Metazeichen, einem Zeilenvorschubzeichen, einem Rückführzeichen oder einem Tabulatorzeichen. Für den Abgleich muss den in Tabelle 37 auf Seite 235 aufgeführten Zeichen in einem regulären Ausdruck ein Escapezeichen vorangestellt werden.

*Escapezeichen_für_mehrere_Zeichen*
Ein Backslash-Zeichen, gefolgt von einem Zeichen, das für den Abgleich jeweils eine häufig verwendete Zeichengruppe in einem regulären Ausdruck identifiziert (siehe Tabelle 38).

*Tabelle 38. Escapezeichen_für_mehrere_Zeichen*

| Escapezeichen für mehrere Zeichen | Entsprechender regulärer Ausdruck | Beschreibung |
|---|---|---|
| `\s` | `[#x20\t\n\r]` | Leerzeichen, Tabulatorzeichen, Zeilenvorschubzeichen oder Rückführzeichen. |
| `\S` | `[^\s]` | Alle Zeichen außer einem Leerzeichen, Tabulatorzeichen, Zeilenvorschubzeichen oder Rückführzeichen. |
| `\i` | Keiner | Die Gruppe der Zeichen, die als erstes Zeichen in einem XML-Namen zulässig sind. |
| `\I` | `[^\i]` | Nicht in der Gruppe der Zeichen, die als erstes Zeichen in einem XML-Namen zulässig sind. |
| `\c` | Keiner | Die Gruppe der Zeichen, die in einem XML-Namen zulässig sind. |
| `\C` | `[^\c]` | Nicht in der Gruppe der Zeichen, die in einem XML-Namen zulässig sind. |
| `\d` | `\p{Nd}` | Eine Dezimalziffer. |
| `\D` | `[^\d]` | Keine Dezimalziffer. |
| `\w` | `[#x0000-#x10FFFF]-[\p{P}\p{Z}\p{C}]` | Ein Wortzeichen, das die folgenden Kategorien für *Zeichenmerkmal* umfasst: Buchstaben, Markierungen, Symbole und Zahlen. |
| `\W` | `[^\w]` | Ein Nicht-Wortzeichen, das die folgenden Kategorien für *Zeichenmerkmal* umfasst: Interpunktion, Trennzeichen und sonstige. |

**p**{*Zeichenmerkmal*}
Gibt ein Zeichen in einer Kategorie an. Die Kategorien sind in Tabelle 39 auf Seite 237 aufgeführt.

**P**{*Zeichenmerkmal*}
Gibt das Komplement einer Zeichenkategorie an. Die Kategorien sind in Tabelle 39 auf Seite 237 aufgeführt.

*Tabelle 39. Unterstützte Werte für Zeichenmerkmal*

| **Kategorie** | *Zeichenmerkmal* | **Beschreibung** |
|---|---|---|
| Buchstaben | L | Alle Buchstaben |
| | Lu | Großbuchstaben |
| | Ll | Kleinschreibung |
| | Lt | Zeichen für Schreibung mit großem Anfangsbuchstaben |
| | Lm | Änderungswert |
| | Lo | Sonstiges |
| Markierungen | M | Alle Markierungen |
| | Mn | Zeichen ohne Vorschub |
| | Mc | Kombinationszeichen mit Vorschub |
| | Me | Einschließendes Zeichen |
| Zahlen | N | Alle Zahlen |
| | Nd | Dezimalziffern |
| | Nl | Buchstaben |
| | No | Sonstiges |
| Interpunktion | P | Alle Interpunktionszeichen |
| | Pc | Verbindungslinien |
| | Pd | Gedankenstrich |
| | Ps | Öffnendes Satzzeichen (eines Satzzeichenpaares) |
| | Pe | Schließendes Satzzeichen (eines Satzzeichenpaares) |
| | Pi | Linke Anführungszeichen (können sich - je nach Gebrauch - wie 'Ps' oder 'Pe' verhalten) |
| | Pf | Rechte Anführungszeichen (können sich - je nach Gebrauch - wie 'Ps' oder 'Pe' verhalten) |
| | Po | Sonstiges |
| Trennzeichen | Z | Alle Trennzeichen |
| | Zs | Leerzeichen |
| | Zl | Zeile |
| | Zp | Absatz |
| Symbole | S | Alle Symbole |
| | Sm | Mathematisches Symbol |
| | Sc | Währungssymbol |
| | Sk | Änderungswert |
| | So | Sonstiges |
| Sonstiges | C | Alle sonstigen Zeichen |
| | Cc | Steuerzeichen |
| | Cf | Format |
| | Co | Persönliche Verwendung |
| | Cn | Nicht zugeordnet |

**Anmerkung:** Der Abgleich von regulären Ausdrücken erfolgt über einen binären Vergleich. Die Standardsortierfolge wird nicht verwendet.

# Kapitel 7. Grenzwerte

Für DB2 XQuery gelten bestimmte Größenbegrenzungen und Begrenzungen für Datentypen.

## Begrenzungen für XQuery-Datentypen

Dieser Abschnitt enthält den Bereich der Werte, die für bestimmte Datentypen von DB2 XQuery zulässig sind.

*Tabelle 40. Begrenzungen für numerische XQuery-Datentypen*

| Datentyp | Beschreibung | Grenzwert |
|---|---|---|
| xs:float | Kleinster Wert | -3.4028234663852886e+38 |
| | Größter Wert | +3.4028234663852886e+38 |
| | Kleinster positiver Wert | +1.1754943508222875e-38 |
| | Größter negativer Wert | -1.1754943508222875e-38 |
| xs:double | Kleinster Wert | -1.7976931348623158e+308 |
| | Größter Wert | +1.7976931348623158e+308 |
| | Kleinster positiver Wert | +2.2250738585072014e-308 |
| | Größter negativer Wert | +2.2250738585072014e-308 |
| xs:decimal | Größte Dezimal-genauigkeit | 31 Stellen |
| xs:integer | Kleinster Wert | -9 223 372 036 854 775 808 |
| | Größter Wert | +9 223 372 036 854 775 807 |
| xs:nonPositiveInteger | Kleinster Wert | -9 223 372 036 854 775 808 |
| | Größter Wert | 0 |
| xs:negativeInteger | Kleinster Wert | -9 223 372 036 854 775 808 |
| | Größter Wert | -1 |
| xs:long | Kleinster Wert | -9 223 372 036 854 775 808 |
| | Größter Wert | 9 223 372 036 854 775 807 |
| xs:int | Kleinster Wert | -2 147 483 648 |
| | Größter Wert | +2 147 483 647 |
| xs:short | Kleinster Wert | -32 768 |
| | Größter Wert | +32 767 |
| xs:byte | Kleinster Wert | -128 |
| | Größter Wert | +127 |
| xs:nonNegativeInteger | Kleinster Wert | 0 |
| | Größter Wert | +9 223 372 036 854 775 807 |
| xs:unsignedLong | Kleinster Wert | 0 |
| | Größter Wert | +9 223 372 036 854 775 807 |
| xs:unsignedInt | Kleinster Wert | 0 |
| | Größter Wert | 4 294 967 295 |

*Tabelle 40. Begrenzungen für numerische XQuery-Datentypen (Forts.)*

| Datentyp | Beschreibung | Grenzwert |
|---|---|---|
| xs:unsignedShort | Kleinster Wert | 0 |
| | Größter Wert | +65 535 |
| xs:unsignedByte | Kleinster Wert | 0 |
| | Größter Wert | +255 |
| xs:positiveInteger | Kleinster Wert | +1 |
| | Größter Wert | +9 223 372 036 854 775 807 |

*Tabelle 41. Begrenzungen für XQuery-Datentypen für Datumsangaben (date), Uhrzeit (time) und Dauer (duration)*

| Datentyp | Beschreibung | Grenzwert |
|---|---|---|
| xs:duration | Kleinster Wert | -P83333333333333Y3M11574074074DT1H46M39.999999S |
| | Größter Wert | P83333333333333Y3M11574074074DT1H46M39.999999S |
| xdt:yearMonthDuration | Kleinster Wert | -P83333333333333Y3M |
| | Größter Wert | P83333333333333Y3M |
| xdt:dayTimeDuration | Kleinster Wert | -P11574074074DT1H46M39.999999S |
| | Größter Wert | P11574074074DT1H46M39.999999S |
| xs:dateTime[1, 2] | Kleinster Wert | 0001-01-01T00:00:00.000000Z |
| | Größter Wert | 9999-12-31T23:59:59.999999Z |
| xs:date[1] | Kleinster Wert | 0001-01-01Z |
| | Größter Wert | 9999-12-31Z |
| xs:time[2] | Kleinster Wert | 00:00:00Z |
| | Größter Wert | 23:59:59Z |
| xs:gDay[1] | Kleinster Wert | 01Z |
| | Größter Wert | 31Z |
| xs:gMonth[1] | Kleinster Wert | 01Z |
| | Größter Wert | 12Z |
| xs:gYear[1] | Kleinster Wert | 0001Z |
| | Größter Wert | 9999Z |
| xs:gYearMonth[1] | Kleinster Wert | 0001-01Z |
| | Größter Wert | 9999-12Z |
| xs:gMonthDay[1] | Kleinster Wert | 01-01Z |
| | Größter Wert | 12-31Z |

**Anmerkung:** DB2 XQuery bietet keine Unterstützung für negative Datumsangaben.

## Größenbegrenzungen

In DB2 XQuery gibt es Größenbegrenzungen für Zeichenfolgeliterale und Abfragen.

Die Größenbegrenzung für Zeichenfolgeliterale beträgt 32.672 Byte.

Die Größenbegrenzung für die Länge einer Abfrage beträgt 2.097.152 Byte.

# Anhang A. Übersicht über die technischen Informationen zu DB2

Die technischen Informationen zu DB2 stehen über die folgenden Tools und Methoden zur Verfügung:

- *DB2-Informationszentrale*
  - Themen (zu Tasks, Konzepten und Referenzinformationen)
  - Hilfe für DB2-Tools
  - Beispielprogramme
  - Lernprogramme
- DB2-Bücher
  - PDF-Dateien (für den Download verfügbar)
  - PDF-Dateien (auf der DB2-PDF-DVD)
  - Gedruckte Bücher
- Befehlszeilenhilfe
  - Hilfe für Befehle
  - Hilfe für Nachrichten

**Anmerkung:** Die Themen der *DB2-Informationszentrale* werden häufiger aktualisiert als die PDF- und Hardcopybücher. Um stets die neuesten Informationen zur Verfügung zu haben, sollten Sie die Dokumentationsaktualisierungen installieren, sobald diese verfügbar sind, oder die *DB2-Informationszentrale* unter ibm.com aufrufen.

Darüber hinaus können Sie auf zusätzliche technische Informationen zu DB2, wie beispielsweise technische Hinweise (Technotes), White Papers und IBM Redbooks, online über ibm.com zugreifen. Rufen Sie die Website 'DB2 Information Management - Software - Library' unter http://www.ibm.com/software/data/sw-library/ auf.

## Feedback zur Dokumentation

Senden Sie uns Ihr Feedback zur DB2-Dokumentation! Wenn Sie Anregungen zur Verbesserung der DB2-Dokumentation haben, senden Sie eine E-Mail an db2docs@ca.ibm.com. Das DB2-Dokumentationsteam bearbeitet das gesamte Feedback, kann jedoch nicht im Einzelnen auf Ihre E-Mails antworten. Nennen Sie uns, wenn möglich, konkrete Beispiele, sodass wir die Problemstellung besser beurteilen können. Wenn Sie uns Feedback zu einem bestimmten Thema oder einer bestimmten Hilfedatei senden, geben Sie den entsprechenden Titel sowie die URL an.

Verwenden Sie diese E-Mail-Adresse nicht, wenn Sie sich an die DB2-Kundenunterstützung wenden möchten. Wenn ein technisches Problem bei DB2 vorliegt, das Sie mithilfe der Dokumentation nicht beheben können, fordern Sie beim zuständigen IBM Service-Center Unterstützung an.

Wenn Sie IBM dabei unterstützen möchten, die Benutzerfreundlichkeit der IBM Information Management-Produkte weiter zu verbessern, können Sie uns Ihr Feedback mithilfe der Umfrage zur Verbraucherfreundlichkeit von Software geben: http://www.ibm.com/software/data/info/consumability-survey/ (landessprachliche Version unter https://www-950.ibm.com/survey/oid/wsb.dll/studies/consumabilitywebform.htm?renderlang=de).

# Bibliothek mit technischen Informationen zu DB2 im Hardcopy- oder PDF-Format

Die folgenden Tabellen enthalten eine Beschreibung der DB2-Bibliothek, die im IBM Publications Center unter www.ibm.com/shop/publications/order zur Verfügung steht. Über die folgende Adresse können Sie englische Handbücher im PDF-Format sowie übersetzte Versionen zu DB2 Version 9.5 herunterladen: www.ibm.com/support/docview.wss?rs=71&uid=swg2700947.

In den Tabellen sind die Bücher, die in gedruckter Form zur Verfügung stehen, gekennzeichnet; möglicherweise sind diese in Ihrem Land oder Ihrer Region jedoch nicht verfügbar.

Die Formnummer wird bei jeder Aktualisierung eines Handbuchs erhöht. Anhand der nachfolgenden Liste können Sie sicherstellen, dass Sie die jeweils neueste Version des Handbuchs lesen.

**Anmerkung:** Die *DB2-Informationszentrale* wird häufiger aktualisiert als die PDF- und Hardcopybücher.

*Tabelle 42. Technische Informationen zu DB2*

| Name | IBM Form | In gedruckter Form verfügbar | Letzte Aktualisierung |
|---|---|---|---|
| *Administrative API Reference* | SC23-5842-02 | Ja | April 2009 |
| *Administrative Routines and Views* | SC23-5843-02 | Nein | April 2009 |
| *Call Level Interface Guide and Reference, Volume 1* | SC23-5844-02 | Ja | April 2009 |
| *Call Level Interface Guide and Reference, Volume 2* | SC23-5845-02 | Ja | April 2009 |
| *Command Reference* | SC23-5846-02 | Ja | April 2009 |
| *Dienstprogramme für das Versetzen von Daten - Handbuch und Referenz* | SC12-3917-02 | Ja | April 2009 |
| *Datenrecovery und hohe Verfügbarkeit - Handbuch und Referenz* | SC12-3919-02 | Ja | April 2009 |
| *Datenserver, Datenbanken und Datenbankobjekte* | SC12-3912-02 | Ja | April 2009 |
| *Datenbanksicherheit* | SC12-3914-02 | Ja | April 2009 |
| *Developing ADO.NET and OLE DB Applications* | SC23-5851-02 | Ja | April 2009 |
| *Developing Embedded SQL Applications* | SC23-5852-02 | Ja | April 2009 |
| *Developing Java Applications* | SC23-5853-02 | Ja | April 2009 |

*Tabelle 42. Technische Informationen zu DB2 (Forts.)*

| Name | IBM Form | In gedruckter Form verfügbar | Letzte Aktualisierung |
|---|---|---|---|
| *Developing Perl and PHP Applications* | SC23-5854-02 | Nein | April 2009 |
| *Developing User-defined Routines (SQL and External)* | SC23-5855-02 | Ja | April 2009 |
| *Getting Started with Database Application Development* | GC23-5856-02 | Ja | April 2009 |
| *Installation und Verwaltung von DB2 unter Linux und Windows - Erste Schritte* | GC12-3922-02 | Ja | April 2009 |
| *Internationalisierung* | SC12-3916-02 | Ja | April 2009 |
| *Fehlernachrichten, Band 1* | GI11-3098-01 | Nein | April 2009 |
| *Fehlernachrichten, Band 2* | GI11-3099-01 | Nein | April 2009 |
| *Migration* | GC12-3921-02 | Ja | April 2009 |
| *Net Search Extender - Verwaltung und Benutzerhandbuch* | SC12-3979-02 | Ja | April 2009 |
| *Partitionierung und Clustering* | SC12-3915-02 | Ja | April 2009 |
| *Query Patroller - Verwaltung und Benutzerhandbuch* | SC12-3977-01 | Ja | April 2009 |
| *IBM Data Server-Clients - Einstieg* | GC12-3924-02 | Nein | April 2009 |
| *DB2-Server - Einstieg* | GC12-3923-02 | Ja | April 2009 |
| *Spatial Extender und Geodetic Data Management Feature - Benutzer- und Referenzhandbuch* | SC12-3978-02 | Ja | April 2009 |
| *SQL Reference, Volume 1* | SC23-5861-02 | Ja | April 2009 |
| *SQL Reference, Volume 2* | SC23-5862-02 | Ja | April 2009 |
| *Systemmonitor - Handbuch und Referenz* | SC12-3918-02 | Ja | April 2009 |
| *Text Search* | SC12-3920-01 | Ja | April 2009 |
| *Fehlerbehebung* | GI11-3097-02 | Nein | April 2009 |
| *Optimieren der Datenbankleistung* | SC12-3913-02 | Ja | April 2009 |
| *Lernprogramm für Visual Explain* | SC12-3932-00 | Nein | |
| *Neue Funktionen* | SC12-3928-02 | Ja | April 2009 |
| *Workload-Manager - Handbuch und Referenz* | SC12-3929-02 | Ja | April 2009 |

Tabelle 42. Technische Informationen zu DB2 (Forts.)

| Name | IBM Form | In gedruckter Form verfügbar | Letzte Aktualisierung |
|---|---|---|---|
| *pureXML - Handbuch* | SC12-3930-02 | Ja | April 2009 |
| *XQuery - Referenz* | SC12-3931-02 | Nein | April 2009 |

Tabelle 43. Technische Informationen zu DB2 Connect

| Name | IBM Form | In gedruckter Form verfügbar | Letzte Aktualisierung |
|---|---|---|---|
| *DB2 Connect Personal Edition - Einstieg* | GC12-3926-02 | Ja | April 2009 |
| *DB2 Connect-Server - Einstieg* | GC12-3927-02 | Ja | April 2009 |
| *DB2 Connect - Benutzerhandbuch* | SC12-3925-02 | Ja | April 2009 |

Tabelle 44. Technische Informationen zu Information Integration

| Name | IBM Form | In gedruckter Form verfügbar | Letzte Aktualisierung |
|---|---|---|---|
| *Information Integration: Föderierte Systeme - Verwaltung* | SC12-3759-01 | Ja | März 2008 |
| *Information Integration: ASNCLP Program Reference for Replication and Event Publishing* | SC19-1018-02 | Ja | März 2008 |
| *Information Integration: Konfiguration föderierter Datenquellen* | SC12-3777-01 | Nein | |
| *Information Integration: SQL Replication - Handbuch und Referenz* | SC12-3782-01 | Ja | März 2008 |
| *Information Integration: Replikation und Event-Publishing - Einführung* | GC12-3779-01 | Ja | März 2008 |

## Bestellen gedruckter DB2-Bücher

Gedruckte DB2-Bücher können Sie in den meisten Ländern oder Regionen online bestellen. Das Bestellen gedruckter DB2-Bücher ist stets über den zuständigen IBM Ansprechpartner möglich. Beachten Sie hierbei bitte, dass einige Softcopybücher auf der DVD mit der *DB2-PDF-Dokumentation* nicht in gedruckter Form verfügbar sind. So sind beispielsweise die beiden Bände des Handbuchs *DB2 Fehlernachrichten* nicht in gedruckter Form erhältlich.

Gedruckte Versionen vieler DB2-Bücher, die auf der DVD mit der DB2-PDF-Dokumentation verfügbar sind, können gegen eine Gebühr bei IBM bestellt werden. Abhängig vom jeweiligen Land bzw. der jeweiligen Region können Sie Bücher möglicherweise online über das IBM Publications Center bestellen. Ist im jeweiligen Land bzw. der jeweiligen Region keine Onlinebestellung möglich, können Sie

gedruckte DB2-Bücher stets über den zuständigen IBM Ansprechpartner bestellen. Nicht alle Bücher, die auf der DVD mit der DB2-PDF-Dokumentation verfügbar sind, können in gedruckter Form bestellt werden.

**Anmerkung:** Über http://publib.boulder.ibm.com/infocenter/db2luw/v9r5 haben Sie Zugriff auf die DB2-Informationszentrale, wo Sie die neueste und umfassendste DB2-Dokumentation finden.

Gehen Sie wie folgt vor, um gedruckte DB2-Bücher zu bestellen:

- Informationen dazu, ob in Ihrem Land oder Ihrer Region die Bestellung von gedruckten DB2-Büchern möglich ist, finden Sie auf der Website mit dem IBM Publications Center unter http://www.ibm.com/shop/publications/order. Wählen Sie ein Land, eine Region oder eine Sprache aus, um die Bestellinformationen für Veröffentlichungen aufzurufen, und führen Sie dann die entsprechenden Schritte des Bestellverfahrens für Ihr Land bzw. Ihre Region aus.
- Gehen Sie wie folgt vor, um gedruckte DB2-Bücher beim zuständigen IBM Ansprechpartner zu bestellen:
  1. Kontaktinformationen zum zuständigen Ansprechpartner finden Sie auf einer der folgenden Websites:
     - IBM Verzeichnis weltweiter Kontakte unter www.ibm.com/planetwide.
     - Website mit IBM Veröffentlichungen unter http://www.ibm.com/shop/publications/order. Wählen Sie das gewünschte Land, die gewünschte Region oder die gewünschte Sprache aus, um auf die entsprechende Homepage mit Veröffentlichungen Ihres Landes bzw. Ihrer Region zuzugreifen. Folgen Sie auf dieser Seite dem Link für Informationen zu dieser Site ("About this Site").
  2. Geben Sie bei Ihrem Anruf an, dass Sie eine DB2-Veröffentlichung bestellen möchten.
  3. Teilen Sie dem zuständigen Ansprechpartner die Titel und Formularnummern der Bücher mit, die Sie bestellen möchten. Titel und Formularnummern finden Sie unter „Bibliothek mit technischen Informationen zu DB2 im Hardcopy- oder PDF-Format" auf Seite 242.

## Aufrufen der Hilfe für den SQL-Status über den Befehlszeilenprozessor

DB2 gibt für Bedingungen, die aufgrund einer SQL-Anweisung generiert werden können, einen SQLSTATE-Wert zurück. Die SQLSTATE-Hilfe erläutert die Bedeutung der SQL-Statuswerte und der SQL-Statusklassencodes.

Zum Aufrufen der Hilfe für SQL-Statuswerte müssen Sie den Befehlszeilenprozessor öffnen und Folgendes eingeben:

```
? sqlstate oder ? klassencode
```

Hierbei steht *sqlstate* für einen gültigen fünfstelligen SQL-Statuswert und *klassencode* für die ersten beiden Ziffern dieses Statuswertes.
So kann beispielsweise durch die Eingabe von `? 08003` Hilfe für den SQL-Statuswert 08003 angezeigt werden, durch die Eingabe von `? 08` Hilfe für den Klassencode 08.

## Zugriff auf verschiedene Versionen der DB2-Informationszentrale

Für Themen aus DB2 Version 9.5 lautet die URL der DB2-Informationszentrale http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/.

Für Themen aus DB2 Version 9 lautet die URL der DB2-Informationszentrale http://publib.boulder.ibm.com/infocenter/db2luw/v9/.

Für Themen aus DB2 Version 8 lautet die URL der Informationszentrale (Version 8, 'Information - Unterstützung') http://publib.boulder.ibm.com/infocenter/db2luw/v8/.

## Anzeigen von Themen in der gewünschten Sprache in der DB2-Informationszentrale

In der DB2-Informationszentrale werden Themen, wenn möglich, in der Sprache angezeigt, die in den Vorgaben Ihres Browsers angegeben ist. Falls ein Thema nicht in die gewünschte Sprache übersetzt wurde, wird es in der DB2-Informationszentrale in Englisch angezeigt.

- Um Themen in der gewünschten Sprache im Browser 'Internet Explorer' anzuzeigen, gehen Sie wie folgt vor:
  1. Klicken Sie im Internet Explorer **Extras** —> **Internetoptionen...** —> **Sprachen...** an. Das Fenster **Spracheinstellung** wird geöffnet.
  2. Stellen Sie sicher, dass die gewünschte Sprache als erster Eintrag in der Liste angegeben ist.
     - Klicken Sie den Knopf **Hinzufügen...** an, um eine neue Sprache zur Liste hinzuzufügen.

       **Anmerkung:** Das Hinzufügen einer Sprache bedeutet nicht zwangsläufig, dass der Computer über die erforderlichen Schriftarten verfügt, um die Themen in der gewünschten Sprache anzuzeigen.
     - Um eine Sprache an den Anfang der Liste zu verschieben, wählen Sie zunächst die gewünschte Sprache und anschließend den Knopf **Nach oben** aus, bis die Sprache an erster Stelle in der Liste steht.
  3. Löschen Sie den Inhalt des Browser-Cache, und aktualisieren Sie anschließend die Seite, um die DB2-Informationszentrale in der gewünschten Sprache anzuzeigen.
- Um Themen in der gewünschten Sprache in einem Firefox- oder Mozilla-Browser anzuzeigen, gehen Sie wie folgt vor:
  1. Wählen Sie den Knopf im Bereich **Languages** des Dialogfensters **Tools** —> **Options** —> **Advanced** aus. Die Anzeige für die Auswahl der Sprache wird im Fenster mit den Einstellungen aufgerufen.
  2. Stellen Sie sicher, dass die gewünschte Sprache als erster Eintrag in der Liste angegeben ist.
     - Wenn Sie eine neue Sprache zur Liste hinzufügen möchten, klicken Sie den Knopf **Add...** an, um eine Sprache im entsprechenden Fenster auszuwählen.
     - Um eine Sprache an den Anfang der Liste zu verschieben, wählen Sie zunächst die gewünschte Sprache und anschließend den Knopf **Move Up** aus, bis die Sprache an erster Stelle in der Liste steht.
  3. Löschen Sie den Inhalt des Browser-Cache, und aktualisieren Sie anschließend die Seite, um die DB2-Informationszentrale in der gewünschten Sprache anzuzeigen.

Bei einigen Kombinationen aus Browser und Betriebssystem müssen Sie möglicherweise auch die Ländereinstellungen des Betriebssystems in die gewünschte Locale und Sprache ändern.

## Aktualisieren der auf Ihrem Computer oder Intranet-Server installierten DB2-Informationszentrale

Wenn Sie die DB2-Informationszentrale lokal installiert haben, können Sie Dokumentationsaktualisierungen von IBM abrufen und installieren.

Zur Aktualisierung der lokal installierten *DB2-Informationszentrale* sind die folgenden Schritte erforderlich:

1. Stoppen Sie die *DB2-Informationszentrale* auf Ihrem Computer und starten Sie die Informationszentrale im Standalone-Modus erneut. Die Ausführung der Informationszentrale im Standalone-Modus verhindert, dass andere Benutzer in Ihrem Netz auf die Informationszentrale zugreifen, und ermöglicht das Anwenden von Aktualisierungen. *DB2-Informationszentralen*, deren Installation nicht als Administrator oder Root ausgeführt wurde, werden stets im Standalone-Modus ausgeführt.
2. Verwenden Sie die Aktualisierungsfunktion, um zu prüfen, welche Aktualisierungen verfügbar sind. Falls Aktualisierungen verfügbar sind, die Sie installieren möchten, können Sie die Aktualisierungsfunktion verwenden, um diese abzurufen und zu installieren.

   **Anmerkung:** Wenn es in der verwendeten Umgebung erforderlich ist, die Aktualisierungen für die *DB2-Informationszentrale* auf einer Maschine zu installieren, die nicht über ein Verbindung zum Internet verfügt, müssen Sie die Aktualisierungssite auf ein lokales Dateisystem spiegeln und dabei eine Maschine verwenden, die mit dem Internet verbunden ist und auf der die *DB2-Informationszentrale* installiert ist. Wenn viele Benutzer Ihres Netzes die Dokumentationsaktualisierungen installieren sollen, können Sie die Zeit, die jeder einzelne Benutzer für die Aktualisierungen benötigt, reduzieren, indem Sie die Aktualisierungssite lokal spiegeln und ein Proxy dafür erstellen. Ist dies der Fall, verwenden Sie die Aktualisierungsfunktion, um die Pakete abzurufen. Die Aktualisierungsfunktion ist jedoch nur im Standalone-Modus verfügbar.
3. Stoppen Sie die im Standalone-Modus gestartete Informationszentrale, und starten Sie die *DB2-Informationszentrale* auf Ihrem Computer erneut.

**Anmerkung:** Unter Windows Vista müssen Sie zur Ausführung der nachfolgend aufgeführten Befehle über Administratorberechtigung verfügen. Zum Starten einer Eingabeaufforderung oder eines Grafiktools mit vollen Administratorberechtigungen klicken Sie mit der rechten Maustaste auf die Verknüpfung, und wählen Sie **Als Administrator ausführen** aus.

Gehen Sie wie folgt vor, um die auf Ihrem Computer bzw. Intranet-Server installierte *DB2-Informationszentrale* zu aktualisieren:

1. Stoppen Sie die *DB2-Informationszentrale*.
   - Unter Windows klicken Sie **Start** → **Einstellungen** → **Systemsteuerung** → **Verwaltung** → **Dienste** an. Klicken Sie mit der rechten Maustaste die **DB2-Informationszentrale** an, und wählen Sie **Stoppen** aus.
   - Unter Linux: Geben Sie den folgenden Befehl ein:
     ```
     /etc/init.d/db2icdv95 stop
     ```
2. Starten Sie die Informationszentrale im Standalone-Modus.
   - Unter Windows:
     a. Öffnen Sie ein Befehlsfenster.

b. Navigieren Sie zu dem Pfad, in dem die Informationszentrale installiert ist. Standardmäßig ist die *DB2-Informationszentrale* im Verzeichnis *Programme*\IBM\DB2 Information Center\Version 9.5 installiert, wobei *Programme* die Position des Programmdateiverzeichnisses ist.
c. Navigieren Sie vom Installationsverzeichnis in das Verzeichnis doc\bin.
d. Führen Sie die Datei help_start.bat aus:

```
help_start.bat
```

- Unter Linux:
  a. Navigieren Sie zu dem Pfad, in dem die Informationszentrale installiert ist. Standardmäßig ist die *DB2-Informationszentrale* im Verzeichnis /opt/ibm/db2ic/V9.5 installiert.
  b. Navigieren Sie vom Installationsverzeichnis in das Verzeichnis doc/bin.
  c. Führen Sie das Script help_start aus:

```
help_start
```

Der standardmäßig auf dem System verwendete Web-Browser wird aufgerufen und zeigt die Standalone-Informationszentrale an.

3. Klicken Sie den Aktualisierungsknopf ( ) an. Klicken Sie im rechten Fenster der Informationszentrale den Knopf für die Suche nach Aktualisierungen an. Eine Liste der Aktualisierungen für die vorhandene Dokumentation wird angezeigt.
4. Wählen Sie zum Initiieren des Installationsprozesses die gewünschten Aktualisierungen aus, und klicken Sie anschließend den Knopf für die Installation der Aktualisierungen an.
5. Klicken Sie nach Abschluss des Installationsprozesses **Fertig stellen** an.
6. Stoppen Sie die im Standalone-Modus gestartete Informationszentrale:
   - Unter Windows: Navigieren Sie in das Verzeichnis doc\bin des Installationsverzeichnisses, und führen Sie die Datei help_end.bat aus:

     ```
     help_end.bat
     ```

     **Anmerkung:** Die Stapeldatei help_end enthält die Befehle, die erforderlich sind, um die Prozesse, die mit der Stapeldatei help_start gestartet wurden, ordnungsgemäß zu beenden. Verwenden Sie nicht die Tastenkombination `Strg+C` oder eine andere Methode, um help_start.bat zu beenden.
   - Unter Linux: Navigieren Sie in das Verzeichnis doc/bin des Installationsverzeichnisses, und führen Sie das Script help_end aus:

     ```
     help_end
     ```

     **Anmerkung:** Das Script help_end enthält die Befehle, die erforderlich sind, um die Prozesse, die mit dem Script help_start gestartet wurden, ordnungsgemäß zu beenden. Verwenden Sie keine andere Methode, um das Script help_start zu beenden.
7. Starten Sie die *DB2-Informationszentrale* erneut.
   - Unter Windows klicken Sie **Start** → **Einstellungen** → **Systemsteuerung** → **Verwaltung** → **Dienste** an. Klicken Sie mit der rechten Maustaste die **DB2-Informationszentrale** an, und wählen Sie **Start** aus.
   - Unter Linux: Geben Sie den folgenden Befehl ein:

     ```
     /etc/init.d/db2icdv95 start
     ```

In der aktualisierten *DB2-Informationszentrale* werden die neuen und aktualisierten Themen angezeigt.

## DB2-Lernprogramme

Die DB2-Lernprogramme unterstützen Sie dabei, sich mit den unterschiedlichen Aspekten der DB2-Produkte vertraut zu machen. Die Lerneinheiten bieten eine in einzelne Schritte unterteilte Anleitung.

### Vorbereitungen

Die XHTML-Version des Lernprogramms kann über die Informationszentrale unter http://publib.boulder.ibm.com/infocenter/db2help/ angezeigt werden.

In einigen der Lerneinheiten werden Beispieldaten und Codebeispiele verwendet. Informationen zu bestimmten Voraussetzungen für die Ausführung der Tasks finden Sie in der Beschreibung des Lernprogramms.

### DB2-Lernprogramme

Klicken Sie zum Anzeigen des Lernprogramms den Titel an.

**„pureXML" in** *pureXML - Handbuch*
Einrichten einer DB2-Datenbank, um XML-Daten zu speichern und Basisoperationen mit dem nativen XML-Datenspeicher auszuführen.

**„Visual Explain" in** *Lernprogramm für Visual Explain*
Analysieren, Optimieren und Anpassen von SQL-Anweisungen zur Leistungsverbesserung mithilfe von Visual Explain.

## Informationen zur Fehlerbehebung in DB2

Eine breite Palette verschiedener Informationen zur Fehlerbestimmung und Fehlerbehebung steht zur Verfügung, um Sie bei der Verwendung von DB2-Datenbankprodukten zu unterstützen.

**DB2-Dokumentation**
Informationen zur Fehlerbehebung stehen im Handbuch DB2-Fehlerbehebung oder im Abschnitt mit grundlegenden Informationen zu Datenbanken in der DB2-Informationszentrale zur Verfügung. Dort finden Sie Informationen dazu, wie Sie Probleme mithilfe der DB2-Diagnosetools und -Dienstprogramme eingrenzen und identifizieren können, Lösungen für einige der häufigsten Probleme sowie weitere Hinweise zur Behebung von Fehlern und Problemen, die bei der Verwendung der DB2-Datenbankprodukte auftreten können.

**DB2-Website mit technischer Unterstützung**
Auf der DB2-Website mit technischer Unterstützung finden Sie Informationen zu Problemen und den möglichen Ursachen und Fehlerbehebungsmaßnahmen. Die Website mit technischer Unterstützung enthält Links zu den neuesten DB2-Veröffentlichungen, technischen Hinweisen (TechNotes), APARs (Authorized Program Analysis Reports) und Fehlerkorrekturen, Fixpacks sowie weiteren Ressourcen. Sie können diese Wissensbasis nach möglichen Lösungen für aufgetretene Probleme durchsuchen.

Rufen Sie die DB2-Website mit technischer Unterstützung unter http://www.ibm.com/software/data/db2/support/db2_9/ auf.

## Bedingungen

Die Berechtigungen zur Nutzung dieser Veröffentlichungen werden Ihnen auf der Basis der folgenden Bedingungen gewährt.

**Persönliche Nutzung:** Sie dürfen diese Veröffentlichungen für Ihre persönliche, nicht kommerzielle Nutzung unter der Voraussetzung vervielfältigen, dass alle Eigentumsvermerke erhalten bleiben. Sie dürfen diese Veröffentlichungen oder Teile der Veröffentlichungen ohne ausdrückliche Genehmigung von IBM nicht weitergeben, anzeigen oder abgeleitete Werke davon erstellen.

**Kommerzielle Nutzung:** Sie dürfen diese Veröffentlichungen nur innerhalb Ihres Unternehmens und unter der Voraussetzung, dass alle Eigentumsvermerke erhalten bleiben, vervielfältigen, weitergeben und anzeigen. Sie dürfen diese Veröffentlichungen oder Teile der Veröffentlichungen ohne ausdrückliche Genehmigung von IBM außerhalb Ihres Unternehmens nicht vervielfältigen, weitergeben, anzeigen oder abgeleitete Werke davon erstellen.

Abgesehen von den hier gewährten Berechtigungen erhalten Sie keine weiteren Berechtigungen, Lizenzen oder Rechte (veröffentlicht oder stillschweigend) in Bezug auf die Veröffentlichungen oder darin enthaltene Informationen, Daten, Software oder geistiges Eigentum.

IBM behält sich das Recht vor, die in diesem Dokument gewährten Berechtigungen nach eigenem Ermessen zurückzuziehen, wenn sich die Nutzung der Veröffentlichungen für IBM als nachteilig erweist oder wenn die obigen Nutzungsbestimmungen nicht genau befolgt werden.

Sie dürfen diese Informationen nur in Übereinstimmung mit allen anwendbaren Gesetzen und Vorschriften, einschließlich aller US-amerikanischen Exportgesetze und Verordnungen, herunterladen und exportieren.

IBM übernimmt keine Gewährleistung für den Inhalt dieser Informationen. Diese Veröffentlichungen werden auf der Grundlage des gegenwärtigen Zustands (auf "as-is"-Basis) und ohne eine ausdrückliche oder stillschweigende Gewährleistung für die Handelsüblichkeit, die Verwendungsfähigkeit oder die Freiheit der Rechte Dritter zur Verfügung gestellt.

# Anhang B. Bemerkungen

Die vorliegenden Informationen wurden für Produkte und Services entwickelt, die auf dem deutschen Markt angeboten werden.

Möglicherweise bietet IBM die in dieser Dokumentation beschriebenen Produkte, Services oder Funktionen in anderen Ländern nicht an. Informationen über die gegenwärtig im jeweiligen Land verfügbaren Produkte und Services sind beim zuständigen IBM Ansprechpartner erhältlich. Hinweise auf IBM Lizenzprogramme oder andere IBM Produkte bedeuten nicht, dass nur Programme, Produkte oder Services von IBM verwendet werden können. Anstelle der Produkte, Programme oder Services können auch andere ihnen äquivalente Produkte, Programme oder Services verwendet werden, solange diese keine gewerblichen oder andere Schutzrechte der IBM verletzen. Die Verantwortung für den Betrieb der Produkte, Programme oder Dienstleistungen in Verbindung mit Fremdprodukten und Fremddienstleistungen liegt beim Kunden, soweit nicht ausdrücklich solche Verbindungen erwähnt sind.

Für in diesem Handbuch beschriebene Erzeugnisse und Verfahren kann es IBM Patente oder Patentanmeldungen geben. Mit der Auslieferung dieses Handbuchs ist keine Lizenzierung dieser Patente verbunden. Lizenzanforderungen sind schriftlich an folgende Adresse zu richten (Anfragen an diese Adresse müssen auf Englisch formuliert werden):

IBM Director of Licensing
IBM Europe, Middle East & Africa
Tour Descartes
2, avenue Gambetta
92066 Paris La Defense
France

Trotz sorgfältiger Bearbeitung können technische Ungenauigkeiten oder Druckfehler in dieser Veröffentlichung nicht ausgeschlossen werden. Die Angaben in diesem Handbuch werden in regelmäßigen Zeitabständen aktualisiert. Die Änderungen werden in Überarbeitungen oder in Technical News Letters (TNLs) bekannt gegeben. IBM kann ohne weitere Mitteilung jederzeit Verbesserungen und/oder Änderungen an den in dieser Veröffentlichung beschriebenen Produkten und/oder Programmen vornehmen.

Dieses Dokument enthält möglicherweise Links oder Verweise auf Websites und Ressourcen anderer Anbieter. Es bestehen keine Zusicherungen, Gewährleistungen oder Verpflichtungen von IBM hinsichtlich der Websites oder Ressourcen anderer Anbieter, auf die im vorliegenden Dokument verwiesen wird, Zugriff besteht oder Links vorhanden sind. Ein Link auf eine Website eines anderen Anbieters bedeutet nicht, dass IBM den Inhalt und die Verwendung dieser Website billigt oder deren Eigentümer anerkennt. Darüber hinaus ist IBM nicht an Transaktionen beteiligt und übernimmt keine Verantwortung für Transaktionen zwischen Ihnen und anderen Anbietern, auch wenn die Informationen (oder Links) zu diesen Anbietern auf einer IBM Website zur Verfügung stehen. IBM ist nicht für die Verfügbarkeit solcher externen Sites oder Ressourcen verantwortlich und übernimmt keine Verantwortung oder Haftung für Inhalte, Services, Produkte oder sonstiges Material, die bzw. das auf diesen oder über diese Sites oder Ressourcen verfügbar sind. Die Software anderer Anbieter unterliegt den Lizenzbedingungen der jeweiligen Software.

Werden an IBM Informationen eingesandt, können diese beliebig verwendet werden, ohne dass eine Verpflichtung gegenüber dem Einsender entsteht.

Lizenznehmer des Programms, die Informationen zu diesem Produkt wünschen mit der Zielsetzung: (i) den Austausch von Informationen zwischen unabhängigen, erstellten Programmen und anderen Programmen (einschließlich des vorliegenden Programms) sowie (ii) die gemeinsame Nutzung der ausgetauschten Informationen zu ermöglichen, wenden sich an folgende Adresse:

IBM Canada Limited
Office of the Lab Director
8200 Warden Avenue
Markham, Ontario
L6G 1C7
CANADA

Die Bereitstellung dieser Informationen kann unter Umständen von bestimmten Bedingungen - in einigen Fällen auch von der Zahlung einer Gebühr - abhängig sein.

Die Lieferung des im Dokument aufgeführten Lizenzprogramms sowie des zugehörigen Lizenzmaterials erfolgt auf der Basis der IBM Rahmenvereinbarung sowie der Allgemeinen Geschäftsbedingungen von IBM, der Internationalen Nutzungsbedingungen der IBM für Programmpakete oder einer äquivalenten Vereinbarung.

Alle in diesem Dokument enthaltenen Leistungsdaten stammen aus einer kontrollierten Umgebung. Die Ergebnisse, die in anderen Betriebsumgebungen erzielt werden, können daher erheblich von den hier erzielten Ergebnissen abweichen. Einige Daten stammen möglicherweise von Systemen, deren Entwicklung noch nicht abgeschlossen ist. Eine Garantie, dass diese Daten auch in allgemein verfügbaren Systemen erzielt werden, kann nicht gegeben werden. Darüber hinaus wurden einige Daten unter Umständen durch Extrapolation berechnet. Die tatsächlichen Ergebnisse können abweichen. Benutzer dieses Dokuments sollten die entsprechenden Daten in ihrer spezifischen Umgebung prüfen.

Alle Informationen zu Produkten anderer Anbieter stammen von den Anbietern der aufgeführten Produkte, deren veröffentlichen Ankündigungen oder anderen allgemein verfügbaren Quellen. IBM hat diese Produkte nicht getestet und kann daher keine Aussagen zu Leistung, Kompatibilität oder anderen Merkmalen machen. Fragen zu den Leistungsmerkmalen von Produkten anderer Anbieter sind an den jeweiligen Anbieter zu richten.

Aussagen über Pläne und Absichten von IBM unterliegen Änderungen oder können zurückgenommen werden und repräsentieren nur die Ziele von IBM.

Diese Veröffentlichung enthält Beispiele für Daten und Berichte des alltäglichen Geschäftsablaufes. Sie sollen nur die Funktionen des Lizenzprogrammes illustrieren; sie können Namen von Personen, Firmen, Marken oder Produkten enthalten. Alle diese Namen sind frei erfunden, Ähnlichkeiten mit tatsächlichen Namen und Adressen sind rein zufällig.

COPYRIGHTLIZENZ:

Diese Veröffentlichung enthält Musteranwendungsprogramme, die in Quellensprache geschrieben sind. Sie dürfen diese Musterprogramme kostenlos kopieren, ändern und verteilen, wenn dies zu dem Zweck geschieht, Anwendungsprogramme zu entwickeln, verwenden, vermarkten oder zu verteilen, die mit der Anwendungsprogrammierschnittstelle konform sind, für die diese Musterprogramme geschrieben werden. Diese Beispiele wurden nicht unter allen denkbaren Bedingungen getestet. Daher kann IBM die Zuverlässigkeit, Wartungsfreundlichkeit oder Funktion dieser Programme weder zusagen noch gewährleisten.

Kopien oder Teile der Musterprogramme bzw. daraus abgeleiteter Code müssen folgenden Copyrightvermerk beinhalten:

© (*Name Ihrer Firma*) (*Jahr*). Teile des vorliegenden Codes wurden aus Musterprogrammen der IBM Corp. abgeleitet. © Copyright IBM Corp. *_Jahr/Jahre angeben_*. Alle Rechte vorbehalten.

## Marken

IBM, das IBM Logo und ibm.com sind Marken oder eingetragene Marken der IBM Corporation in den USA und/oder anderen Ländern. Weitere Produkt- oder Servicenamen können Marken von IBM oder anderen Herstellern sein. Eine aktuelle Liste der IBM Marken finden Sie auf der Webseite Copyright and trademark information unter www.ibm.com/legal/copytrade.shtml.

Folgende Namen sind Marken oder eingetragene Marken anderer Unternehmen.

- Linux ist eine eingetragene Marke von Linus Torvalds in den USA und/oder anderen Ländern.
- Java und alle auf Java basierenden Marken und Logos sind Marken von Sun Microsystems, Inc. in den USA und/oder anderen Ländern.
- UNIX ist eine eingetragene Marke von The Open Group in den USA und anderen Ländern.
- Intel, das Intel-Logo, Intel Inside, das Intel Inside-Logo, Intel Centrino, das Intel Centrino-Logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium und Pentium sind Marken oder eingetragene Marken der Intel Corporation oder ihrer Tochtergesellschaften in den USA und/oder anderen Ländern.
- Microsoft, Windows, Windows NT und das Windows-Logo sind Marken der Microsoft Corporation in den USA und/oder anderen Ländern.

Weitere Unternehmens-, Produkt- oder Servicenamen können Marken anderer Hersteller sein.

# Index

## A


Abfragen
  Struktur 1
Abfragesprachen
  Kommentare 17
  von Groß-/Kleinschreibung abhängig 16
  XML-Daten 2
Abgekürzte Syntax 74
abs, Funktion 156
Achsen
  abgekürzte Syntax 74
  in Pfadausdrücken 71
Achsenschritte
  in Pfadausdrücken 71
  Knotentests 72
adjust-date-to-timezone, Funktion 150
adjust-dateTime-to-timezone, Funktion 152
adjust-time-to-timezone, Funktion 154
Aktualisieren von XML-Daten mit XQuery 125
Aktualisierungen
  DB2-Informationszentrale 247
Aktualisierungsausdrücke 125
  kombinieren 125
Allgemeine Vergleiche 86
AND, Operator 89
Ändern von Knotennamen 136
anyAtomicType, Datentyp 30
anySimpleType, Datentyp 30
anyType, Datentyp 30
anyURI, Datentyp 30
Arithmetische Ausdrücke 81
Atomare Typen 19
Atomare Werte 6
Atomisierung 61
Attribute
  berechnete Konstruktoren
    Beschreibung 101
  erstellen 101
  Namensbereichsdeklaration 96
attribute, Achse 71
Attributknoten 10
Ausdrücke
  arithmetische 81
  Atomisierung 61
  bedingte 119
  Bereich 79
  dynamischer Kontext 57
  effektiver Boolescher Wert 63
  eingeschlossene, in Konstruktoren 91
  Fehler beim Aktualisieren von XML-Daten 126
  Filter 79
  FLWOR
    Beispiel 116
    gemeinsame Verwendung der Klauseln FOR und LET 110
    Klauseln FOR 108
    Klauseln FOR und LET, Geltungsbereich von Variablen 111
    Klauseln FOR und LET, Übersicht 108
    Klauseln FOR und LET, Vergleich 111
    Klauseln LET 110
Ausdrücke *(Forts.)*
  FLWOR *(Forts.)*
    Klauseln ORDER BY 113
    Klauseln RETURN 115
    Klauseln WHERE 112
    Syntax 106
    Übersicht 106
  Fokus 57
  für Löschen 131
  für Umbenennung 136
  Knotensequenzen kombinieren 80
  Konstruktoren
    berechnete, für Elemente 100
    berechnete Verarbeitungsanweisung 105
    berechneter Kommentar 106
    berechnetes Attribut 101
    Beschreibung 90
    Deklarationsattribute für Namensbereiche 96
    direkte, für Elemente 92
    direkte Verarbeitungsanweisung 104
    direkter Kommentar 105
    Dokumentknoten 102
    gültige Namensbereiche 99
    Textknoten 103
    Verarbeitungsanweisung 104
  logische 89
  Pfad
    abgekürzte Syntax 74
    Beschreibung 68
    Syntax 69
  primäre
    Entitätsverweise 65
    Funktionsaufrufe 67
    in Klammern 67
    Kontextelement 67
    Literale 64
    Übersicht 64
    Variablenverweise 66
    Zeichenverweise 66
  quantifizierte 120
  Reihenfolge von Ergebnissen 58
  Sequenz 78
  Sequenzen erstellen 78
  Subtypsubstitution 62
  Typumstufung 62
  umsetzbare (castable) 123
  Umsetzung 121
  verarbeiten 57
  Vergleich
    allgemein 86
    für Werte 84
    Knoten 88
    Übersicht 84
  Vergleichselemente 77
  Vorrangstellung 58
  XML-Daten aktualisieren 125
  zum Einfügen 132
  zum Ersetzen 139
  zum Umsetzen
    Klausel COPY 129
    Klausel RETURN 129

Ausdrücke *(Forts.)*
zum Umsetzen *(Forts.)*
Klauseln MODIFY 129
Syntax 128
Auswerten von Ausdrücken 57
avg, Funktion 157

# B

base64Binary, Datentyp 31
Bedingungen
Verwendung der Veröffentlichungen 250
Bedingungsausdrücke 119
Begrenzungen
Größe 240
XQuery-Datentypen 239
Begrenzungsleerzeichen
Deklaration 50
in direkten Elementkonstruktoren 97
Bemerkungen 251
Berechnete Attributkonstruktoren 101
Berechnete Konstruktoren
Attribut
Beschreibung 101
Beschreibung 90
Element 100
Kommentar 106
Verarbeitungsanweisung 105
Bereichsausdrücke 79
Bestellen von DB2-Büchern 244
boolean, Datentyp 31
boolean, Funktion 157
Boolesche Funktionen, Liste 145
boundary-space, Deklaration 50
Bücher
gedruckt
bestellen 244
byte, Datentyp 31

# C

castable, Ausdruck 123
ceiling, Funktion 158
child, Achse 71
codepoints-to-string, Funktion 159
compare, Funktion 160
concat, Funktion 161
construction, Deklaration 51
contains, Funktion 161
COPY, Klausel 129
copy-namespace, Deklaration 51
count, Funktion 162
current-date, Funktion 163
current-dateTime, Funktion 163
current-time, Funktion 163

# D

data, Funktion 164
date, Datentyp 31
Datenmodell
XQuery und XPath 5
Datentypen
Begrenzungen 239
explizite Umsetzung (Casting) 27
für Datumsangaben, Uhrzeit und Dauer, Liste 24
Datentypen *(Forts.)*
generische, Liste 21
integrierte, Konstruktoren 26
Kategorien 21
Liste 21
nicht typisierte, Liste 21
numerische, Liste 22
sonstige, Liste 25
Substitution 62
Übersicht 19
Umsetzung von Werten testen (DB2 XQuery) 123
Umstufung 62
xdt: 46
xdt:anyAtomicType 30
xdt:dayTimeDuration 33
xdt:untyped 46
xdt:untypedAtomic 46
xs:anySimpleType 30
xs:anyType 30
xs:anyURI 30
xs:base64Binary 31
xs:boolean 31
xs:byte 31
xs:date 31
xs:dateTime 32
xs:decimal 35
xs:double 35
xs:duration 36
xs:ENTITY 37
xs:float 37
xs:gDay 38
xs:gMonth 38
xs:gMonthDay 39
xs:gYear 39
xs:gYearMonth 40
xs:hexBinary 40
xs:ID 40
xs:IDREF 40
xs:int 41
xs:integer 41
xs:language 41
xs:long 41
xs:Name 41
xs:NCName 42
xs:negativeInteger 42
xs:NMTOKEN 42
xs:nonNegativeInteger 42
xs:nonPositiveInteger 42
xs:normalizedString 43
xs:NOTATION 43
xs:positiveInteger 43
xs:QName 43
xs:short 44
xs:string 44
xs:time 44
xs:token 45
xs:unsignedByte 45
xs:unsignedInt 45
xs:unsignedLong 45
xs:unsignedShort 45
Zeichenfolge, Liste 21
Datentypen für Datumsangaben, Liste 24
Datentypen für Dauer, Liste 24
Datentypen für Uhrzeit, Liste 24
Datentyphierarchie 19
dateTime, Datentyp 32
dateTime, Funktion 165

Datums- und Uhrzeitfunktionen, Liste 146
day-from-date, Funktion 165
day-from-dateTime, Funktion 166
days-from-duration, Funktion 166
dayTimeDuration, Datentyp 33
  normalisiertes Format 33
DB2-definierte Funktionen 143
DB2-Informationszentrale
  Aktualisierung 247
  in verschiedenen Sprachen anzeigen 246
  Sprachen 246
  Versionen 245
DB2 XQuery, Übersicht 1
DB2 XQuery, XML-Daten aktualisieren 125
DB2 XQuery-Funktionen
  abs 156
  adjust-date-to-timezone 150
  adjust-dateTime-to-timezone 152
  adjust-time-to-timezone 154
  avg 157
  boolean 157
  Boolesche Funktionen, Liste 145
  ceiling 158
  codepoints-to-string 159
  compare 160
  concat 161
  contains 161
  count 162
  current-date 163
  current-dateTime 163
  current-time 163
  data 164
  dateTime 165
  day-from-date 165
  day-from-dateTime 166
  days-from-duration 166
  deep-equal 167
  default-collation 169
  distinct-values 170
  empty 171
  ends-with 171
  exactly-one 172
  exists 172
  false 173
  floor 173
  Funktionen für Datumsangaben, Uhrzeit und Dauer, Liste 146
  Funktionsliste 146
  hours-from-dateTime 174
  hours-from-duration 175
  hours-from-time 175
  implicit-timezone 176
  in-scope-prefixes 176
  index-of 177
  insert-before 178
  Knotenfunktionen, Liste 149
  last 179
  local-name 179
  local-name-from-QName 180
  lower-case 180
  matches 182
  max 183
  min 184
  minutes-from-dateTime 186
  minutes-from-duration 186
  minutes-from-time 187
  month-from-date 188
DB2 XQuery-Funktionen *(Forts.)*
  month-from-dateTime 188
  months-from-duration 189
  name 190
  namespace-uri 191
  namespace-uri-for-prefix 192
  namespace-uri-from-QName 192
  node-name 193
  normalize-space 194
  normalize-unicode 194
  not 195
  number 196
  one-or-more 197
  position 197
  QName 198
  QName-Funktionen, Liste 149
  remove 198
  replace 199
  resolve-QName 201
  reverse 202
  root 203
  round 203
  round-half-to-even 205
  seconds-from-dateTime 206
  seconds-from-duration 207
  seconds-from-time 207
  Sequenzfunktionen, Liste 148
  sonstige Funktionen, Liste 150
  sqlquery 4, 208
  starts-with 211
  string 212
  string-join 213
  string-length 213
  string-to-codepoints 214
  subsequence 215
  substring 215
  substring-after 216
  substring-before 217
  sum 218
  timezone-from-date 219
  timezone-from-dateTime 220
  timezone-from-time 220
  tokenize 221
  translate 223
  true 224
  unordered 224
  upper-case 225
  xmlcolumn 3, 226
  year-from-date 227
  year-from-dateTime 228
  years-from-duration 229
  Zahlfunktionen, Liste 145
  Zeichenfolgefunktionen, Liste 143
  zero-or-one 230
decimal, Datentyp 35
deep-equal, Funktion 167
default-collation, Funktion 169
default element/type namespace, Deklaration 52
default function namespace, Deklaration 53
Deklarationen
  boundary-space 50
  construction 51
  copy-namespaces 51
  default element/type namespace 52
  default function namespace 53
  empty order 54
  namespace 55

Deklarationen *(Forts.)*
ordering 54
Prolog 49
version 50
Deklarationsattribute für Namensbereiche 96
descendant, Achse 71
descendant-or-self, Achse 72
Direkte Konstruktoren
Beschreibung 90
Element
Beschreibung 92
Kommentar 105
Leerzeichen im Element 97
Verarbeitungsanweisung 104
distinct-values, Funktion 170
Dokumentation
gedruckt 242
Nutzungsbedingungen 250
PDF 242
Übersicht 241
Dokumentknoten
Beschreibung 9
erstellen 102
Dokumentreihenfolge 11
double, Datentyp 35
duration, Datentyp 36
normalisiertes Format 36
Dynamischer Kontext, Ausdrücke 57

## E

Effektiver Boolescher Wert 63
Einfügeausdruck 132
Einfügen von Knoten 132
Eingeschlossene Ausdrücke
in Konstruktoren 91
Elemente
berechnete Konstruktoren 100
direkte Konstruktoren 92
gültige Namensbereiche 99
Elemente in Sequenzen 5
Elementknoten 9
empty, Funktion 171
empty order, Deklaration 54
ends-with, Funktion 171
Entfernen von Knoten 131
Entitätsverweise 65
ENTITY, Datentyp 37
Ergebnisse
Reihenfolge für Ausdrücke 58
Ersetzen von Knoten und Knotenwerten 139
Ersetzungsausdruck 139
Erweiterte qualifizierte Namen (QNames)
Beschreibung 14
konvertieren 201
exactly-one, Funktion 172
exists, Funktion 172

## F

false, Funktion 173
Fehler bei XQuery-Aktualisierungen 126
Fehlerbehebung
Lernprogramme 249
Onlineinformationen 249
Fehlerbestimmung
Lernprogramme 249
verfügbare Informationen 249
Filterausdrücke 79
float, Datentyp 37
floor, Funktion 173
FLWOR-Ausdrücke
Beispiel 116
Klauseln FOR 108
Klauseln FOR und LET
Geltungsbereich von Variablen 111
in demselben Ausdruck 110
Übersicht 108
Vergleich 111
Klauseln LET
Beschreibung 110
Klauseln ORDER BY 113
Klauseln RETURN 115
Klauseln WHERE 112
Syntax 106
Übersicht 106
Fokus von Ausdrücken 57
FOR, Klauseln
Beschreibung 108
Funktionen
DB2 XQuery
abs 156
adjust-date-to-timezone 150
adjust-dateTime-to-timezone 152
adjust-time-to-timezone 154
avg 157
boolean 157
Boolesche Funktionen, Liste 145
ceiling 158
codepoints-to-string 159
compare 160
concat 161
contains 161
count 162
current-date 163
current-dateTime 163
current-time 163
data 164
dateTime 165
day-from-date 165
day-from-dateTime 166
days-from-duration 166
deep-equal 167
default-collation 169
distinct-values 170
empty 171
ends-with 171
exactly-one 172
exists 172
false 173
floor 173
Funktionen für Datumsangaben, Uhrzeit und Dauer, Liste 146
hours-from-dateTime 174
hours-from-duration 175
hours-from-time 175
implicit-timezone 176
index-of 177
ins-scope-prefixes 176
insert-before 178
Kategorien 143
Knotenfunktionen, Liste 149

Funktionen *(Forts.)*
DB2 XQuery *(Forts.)*
last 179
Listen 143
local-name 179
local-name-from-QName 180
lower-case 180
matches 182
max 183
min 184
minutes-from-dateTime 186
minutes-from-duration 186
minutes-from-time 187
month-from-date 188
month-from-dateTime 188
months-from-duration 189
name 190
namespace-uri 191
namespace-uri-for-prefix 192
namespace-uri-from-QName 192
node-name 193
normalize-space 194
normalize-unicode 194
not 195
number 196
one-or-more 197
position 197
QName 198
QName-Funktionen, Liste 149
remove 198
replace 199
resolve-QName 201
reverse 202
root 203
round 203
round-half-to-even 205
seconds-from-dateTime 206
seconds-from-duration 207
seconds-from-time 207
Sequenzfunktionen, Liste 148
sonstige Funktionen, Liste 150
sqlquery 208
starts-with 211
string 212
string-join 213
string-length 213
string-to-codepoints 214
subsequence 215
substring 215
substring-after 216
substring-before 217
sum 218
timezone-from-date 219
timezone-from-dateTime 220
timezone-from-time 220
tokenize 221
translate 223
true 224
Uhrzeitfunktionen, Liste 146
unordered 224
upper-case 225
xmlcolumn 226
year-from-date 227
year-from-dateTime 228
years-from-duration 229
Zahlfunktionen, Liste 145
Zeichenfolgefunktionen, Liste 143
Funktionen *(Forts.)*
DB2 XQuery *(Forts.)*
zero-or-one 230
Funktionsaufrufe 67

# G

gDay, Datentyp 38
Generische Datentypen, Liste 21
gMonth, Datentyp 38
gMonthDay, Datentyp 39
Groß-/Kleinschreibung, abhängige Abfragesprache 16
Gültige Namensbereiche 99
gYear, Datentyp 39
gYearMonth, Datentyp 40

# H

hexBinary, Datentyp 40
Hierarchie von Datentypen 19
Hierarchie von Knoten 11
Hilfe
Konfiguration der Sprache 246
SQL-Anweisungen 245
Hinzufügen von Knoten 132
hours-from-dateTime, Funktion 174
hours-from-duration, Funktion 175
hours-from-time, Funktion 175

# I

ID, Datentyp 40
Identität von Knoten 11
IDREF, Datentyp 40
if-then-else-Ausdrücke
Beschreibung 119
implicit-timezone, Funktion 176
in-scope-prefixes, Funktion 176
index-of, Funktion 177
insert-before, Funktion 178
int, Datentyp 41
integer, Datentyp 41
Integrierte Datentypen, Konstruktoren 26
Integrierte Funktionen 143

# K

Klammerausdrücke 67
Knoten
Attribut 10
Dokument
Beschreibung 9
erstellen 102
doppelte 11
Element 9
Hierarchie 11
Identität 11
Kommentar
berechnete Konstruktoren 106
Beschreibung 11
direkte Konstruktoren 105
erstellen, Übersicht 105
Merkmale 8
Sequenzen kombinieren 80

Knoten *(Forts.)*
Text
Beschreibung 10
erstellen 103
typisierte Werte 12
Übersicht 6, 9
Verarbeitungsanweisung
Beschreibung 10
erstellen 104
vergleichen 88
Zeichenfolgewerte 12
Knoten hinzufügen 132
Knoten löschen 131
Knoten und Knotenwerte ersetzen 139
Knotennamen ändern 136
Knotentests 72
Kombinieren von Aktualisierungsausdrücken 125
Kommentare
Abfragesprache, Beschreibung 17
berechnete Konstruktoren 106
direkte Konstruktoren 105
erstellen, Übersicht 105
Kommentarknoten 11
Konstruktoren
Attribut
Beschreibung 101
berechnete, für Elemente 100
berechnete Verarbeitungsanweisung 105
berechneter Kommentar 106
berechnetes Attribut 101
Deklarationsattribute für Namensbereiche 96
direkte, für Elemente
Beschreibung 92
direkte Verarbeitungsanweisung 104
direkter Kommentar 105
Dokumentknoten 102
eingeschlossene Ausdrücke 91
gültige Namensbereiche 99
integrierte Typen 26
Textknoten 103
Verarbeitungsanweisung 104
XML 90
Kontext von Ausdrücken 57
Kontextelementausdrücke 67

# L

language, Datentyp 41
last, Funktion 179
Leere Sequenzen sortieren 54
Leerzeichen
Begrenzung 50
Beschreibung 16
in direkten Elementkonstruktoren 97
Lernprogramme
Fehlerbehebung 249
Fehlerbestimmung 249
Visual Explain 249
LET, Klauseln
Beschreibung 110
Literale 64
local-name, Funktion 179
local-name-from-QName, Funktion 180
Logische Ausdrücke 89
long, Datentyp 41
Löschausdruck 131
Löschen von Knoten 131
lower-case, Funktion 180

# M

matches, Funktion 182
max, Funktion 183
min, Funktion 184
minutes-from-dateTime, Funktion 186
minutes-from-duration, Funktion 186
minutes-from-time, Funktion 187
MODIFY, Klauseln 129
month-from-date, Funktion 188
month-from-dateTime, Funktion 188
months-from-duration, Funktion 189

# N

Name, Datentyp 41
name, Funktion 190
Namensbereiche 14
deklarieren 55
Funktion, Standard 53
gültige 99
Präfix binden 96
Standard für Element-/Typnamen 52, 96
Standardwert festlegen 96
Namensbereichsdeklarationen 55
Namenstests 72
namespace-uri, Funktion 191
namespace-uri-for-prefix, Funktion 192
namespace-uri-from-QName, Funktion 192
NCName, Datentyp 42
negativeInteger, Datentyp 42
Nicht typisierte Datentypen, Liste 21
NMTOKEN, Datentyp 42
node-name, Funktion 193
nonNegativeInteger, Datentyp 42
nonPositiveInteger, Datentyp 42
Normalisiertes Format für Dauer
Datentyp 'dayTimeDuration' 33
Datentyp 'duration' 36
Datentyp 'yearMonthDuration' 46
normalize-space, Funktion 194
normalize-unicode, Funktion 194
normalizedString, Datentyp 43
not, Funktion 195
NOTATION, Datentyp 43
number, Funktion 196
Numerische Datentypen, Liste 22
Numerische Vergleichselemente 77
Numerisches Literal 64

# O

one-or-more, Funktion 197
Operatoren
Vorrangstellung 58
OR, Operator 89
ORDER BY, Klauseln 113

# P

parent, Achse 72
Pfadausdrücke
abgekürzte und nicht abgekürzte Syntax 74

Pfadausdrücke *(Forts.)*
Achsenschritte 71
Beschreibung 68
Syntax 69
position, Funktion 197
Positionsgebundene Vergleichswerte 77
positiveInteger, Datentyp 43
Primärausdrücke 64
Primitive Typen, explizite Typumsetzung (Casting) 27
Prolog
Deklaration 'boundary-space' 50
Deklaration 'construction' 51
Deklaration 'copy-namespace' 51
Deklaration 'default element/type namespace' 52
Deklaration 'default function namespace' 53
Deklaration 'empty order' 54
Namensbereichsdeklarationen 55
Sortiermodusdeklaration 54
Syntax 49
Versionsdeklarationen 50

# Q

QName, Datentyp 43
QName, Funktion 198
QNames (qualifizierte Namen) 14
erweiterte, konvertieren 201
Übersicht 14
Qualifizierte Namen (QNames)
erweiterte, konvertieren 201
Übersicht 14
Quantifizierte Ausdrücke 120

# R

Reguläre Ausdrücke
Beschreibung 231
Syntax 231
Reihenfolge der Verarbeitung 113
Reihenfolge von Ergebnissen 58
remove, Funktion 198
replace, Funktion 199
resolve-QName, Funktion 201
Ressourcen
XQuery 17
RETURN, Klauseln 115, 129
reverse, Funktion 202
root, Funktion 203
round, Funktion 203
round-half-to-even, Funktion 205
Rückwärtsachse 71
Runde Klammern, Vorrangstellung von Operationen 58

# S

seconds-from-dateTime, Funktion 206
seconds-from-duration, Funktion 207
seconds-from-time, Funktion 207
self, Achse 72
Sequenzausdrücke 78
Sequenzen
Atomisierung 61
Beschreibung 5
effektiver Boolescher Wert 63
erstellen 78
Sequenzen *(Forts.)*
Knoten
kombinieren 80
leere 54
Sequenzfunktionen, Liste 148
Serialisierung
XML-Daten 13
Setter, Prolog 49
short, Datentyp 44
Sortentests 73
Sortiermodusdeklaration 54
Spezifikationen
XQuery 17
SQL-Anweisungen
Hilfe anzeigen 245
sqlquery, Funktion 4, 208
starts-with, Funktion 211
Statisch bekannte Namensbereiche 99
string, Datentyp 44
string, Funktion 212
string-join, Funktion 213
string-length, Funktion 213
string-to-codepoints, Funktion 214
subsequence, Funktion 215
substring, Funktion 215
substring-after, Funktion 216
substring-before, Funktion 217
Subtypsubstitution 62
sum, Funktion 218
Syntax
abgekürzte 74
FLWOR-Ausdrücke 106
Umsetzungsausdruck 128

# T

Testen der Umsetzung von Werten (DB2 XQuery) 123
Textknoten
Beschreibung 10
erstellen 103
time, Datentyp 44
timezone-from-date, Funktion 219
timezone-from-dateTime, Funktion 220
timezone-from-time, Funktion 220
token, Datentyp 45
tokenize, Funktion 221
translate, Funktion 223
true, Funktion 224
Typen
siehe Datentypen 21
Typhierarchie 19
Typisierte Werte von Knoten 12
Typumsetzung 27
Typumstufung 62

# U

Überprüfung der Umsetzung von Werten (DB2 XQuery) 123
Uhrzeitfunktionen, Liste 146
Umbenennungsausdruck 136
Umsetzung von Datentypen 27
Umsetzungsausdruck
Klausel COPY 129
Klausel RETURN 129
Klauseln MODIFY 129
Syntax 128

Umsetzungsausdrücke 121
Unbenennen von Knoten 136
Unicode-Zeichen 66
unordered, Funktion 224
unsignedByte, Datentyp 45
unsignedInt, Datentyp 45
unsignedLong, Datentyp 45
unsignedShort, Datentyp 45
untyped, Datentyp 46
untypedAtomic, Datentyp 46
upper-case, Funktion 225
URI
  Namensbereichspräfix binden 96

## V

Variablen
  Geltungsbereich für Klauseln FOR und LET 111
  positionsgebundene, in Klauseln FOR 109
  Verweise 66
Verarbeitungsanweisungsknoten
  Beschreibung 10
  erstellen 104
Verarbeitungsreihenfolge 113
Vergleichsausdrücke
  allgemeine 86
  für Knoten 88
  für Werte 84
  Übersicht 84
Vergleichselemente
  in Ausdrücken 77
Versionsdeklarationen 50
Visual Explain
  Lernprogramm 249
Vorrangstellung
  Operatoren und Ausdrücke 58
Vorwärtsachse 71

## W

Werte, atomare 6
Wertevergleiche 84
WHERE, Klauseln
  Beschreibung 112

## X

XDM, siehe XQuery- und XPath-Datenmodell 5
XML-Daten
  in DB2-Datenbank abfragen 2
  serialisieren 13
xmlcolumn, Funktion 3, 226
XMLEXISTS, Funktion 3
XMLQUERY, Funktion 3
XMLTABLE, Funktion 3
XQuery
  Aktualisierungsausdrücke 124, 125, 131
  Aktualisierungsausdrücke kombinieren 125
  Aufruf über SQL 3
  Ausdrücke 57
  Basiskonzepte 1
  Größenbegrenzungen und Begrenzungen für Datentypen 239
  Ressourcen 17
  Sprachkonventionen 16
  statisch bekannte Namensbereiche 15
XQuery *(Forts.)*
  Übersicht 1
XQuery - Referenz, Übersicht i
XQuery- und XPath-Datenmodell 5
XQuery-Aktualisierungen
  Fehler 126
XQuery-definierte Funktionen 143

## Y

year-from-date, Funktion 227
year-from-dateTime, Funktion 228
yearMonthDuration, Datentyp 46
  normalisiertes Format 46
years-from-duration, Funktion 229

## Z

Zahlfunktionen, Liste 145
Zeichenfolgedatentypen, Liste 21
Zeichenfolgefunktionen, Liste 143
Zeichenfolgeliteral 64
Zeichenfolgewerte von Knoten 12
Zeichenverweise 66
Zeitzone, implizite 176
zero-or-one, Funktion 230

SC12-3931-02

Spine information:

DB2 Version 9.5 für Linux, UNIX und Windows

**XQuery - Referenz**