COGNOS®

Cognos Impromptu (R)

Erstellen von benutzerdefinierten Funktionen (UDFs)

Produktinformationen

Dieses Dokument bezieht sich auf Impromptu (R) Version 7.1 und möglicherweise auch auf zukünftige Versionen. Jüngere Versionen dieses Dokuments finden Sie auf der Website des Cognos Support (http://support.cognos.com).

Copyright

Inhaltsverzeichnis

Herzlich willkommen!

Inhalt dieses Dokuments

In diesem Dokument finden Sie ausführliche Informationen, die Sie für das Erstellen von benutzerdefinierten Funktionen (UDFs = User-Defined Functions) zur Benutzung mit Ihrem Cognos-Produkt benötigen.

- Kapitel 1 - "Überblick über UDFs" - enthält allgemeine Informationen über das Erstellen von UDFs und die unterstützten Betriebssysteme. Dieses Kapitel enthält außerdem eine Kurzreferenz für die Schritte, die Sie für das Erstellen von UDFs durchführen.
- Kapitel 2 - "Modifizieren der .INI-Dateien " - erläutert die Änderungen, die Sie in den .INI-Dateien vornehmen müssen.
- Kapitel 3 - "Implementieren von Datenbank-UDFs " - erklärt, wie UDFs für Ihr Datenbanksystem bzw. Ihre Datenbanksysteme eingerichtet werden.
- Kapitel 4 - "Implementieren von externen UDFs " - beschreibt, wie eine UDF in einer DLL (für Windows-Umgebungen) oder in einer gemeinsam benutzten Bibliothek (für UNIX-Umgebungen) eingerichtet wird.
- Der Anhang enthält zusätzliche Informationen über Dateinamen-Konventionen, die Dateien im UDF Software-Entwickler-Kit (SDK) sowie über das Dienstprogramm COGUDF.EXE zum Überprüfen der Syntax oder der .SQL-Dateien.

Voraussetzungen für die Arbeit mit diesem Dokument

Für das Erstellen von benutzerdefinierten Funktionen (UDFs) müssen Sie sich mit Folgendem auskennen:

- Benutzen eines Texteditors zum Modifizieren von Textdateien (.INI und .SQL).
- Hinzufügen von Funktionen zu Ihrer Datenbank (für Datenbank-UDFs)
- Schreiben, Kompilieren und Verbinden von Programmen in C-Sprache (für externe UDFs)

Software-Entwickler-Kit

Für die benutzerdefinierten Funktionen (UDFs) müssen Sie auf den Software-Entwickler-Kit (SDK) zugreifen. Mit Hilfe des SDK können Sie Datenbank-UDFs und externe UDFs für Windows und UNIX erstellen.

Der SDK befindet sich standardmäßig im Ordner *Installationsverzeichnis*\cer*n*\bin\Benutzerdefinierte Funktionen.

Der SDK enthält die folgenden Dateien:

- dieses Dokument
- die C-Quelldateien, die für die Implementierung externer UDFs erforderlich sind
- ein Dienstprogramm, mit dem Sie schnell die UDF-Definitions-Syntax in .SQL-Dateien überprüfen können
- Beispieldateien und die Readme-Textdateien, die das Benutzen der Beispieldateien erläutern

Informationen über die Dateien im SDK finden Sie im Anhang auf (S. 35).

Die SDK-Dateien und Beispiele wurden mit Hilfe von Microsoft Developer Studio 97 mit Visual C++ 4.2 erstellt.

Konventionen in diesem Dokument

- Code-Beispiele werden im Schriftstil `Courier` angezeigt.
- Um Verwechslungen zwischen Windows- und UNIX-Umgebungen zu vermeiden, werden alle Dateinamen in diesem Dokument in Großbuchstaben abgebildet, außer in solchen Fällen, wo die Groß- und Kleinschreibung beachtet werden muss.

Weitere Informationen

Diese Dokumentation enthält Benutzerhandbücher, Lernprogramme, Referenzhandbücher und andere Unterlagen zum Nachschlagen für unsere Kunden.

Alle Informationen sind als Online-Hilfe verfügbar. Die Online-Hilfe wird in Windows-Produkten über die Hilfe-Schaltfläche aufgerufen.

Die Informationen der Online-Hilfesysteme stehen auch im Online-Buchformat (PDF) zur Verfügung. Die Informationen in einem bestimmten Hilfesystem können aber auf mehrere Online-Bücher aufgeteilt sein. Verwenden Sie Online-Bücher, wenn Sie eine gedruckte Version eines Dokuments benötigen oder das gesamte Dokument durchsuchen möchten. Sie können ausgewählte Seiten, einen Abschnitt oder das ganze Buch ausdrucken. Cognos erteilt Ihnen eine nicht exklusive, nicht übertragbare Lizenz zum Verwenden, Kopieren und Vervielfältigen der Copyright-Unterlagen in gedrucktem oder elektronischem Format zum ausschließlichen Zweck der internen Schulung, Verwendung und Wartung der Cognos-Software.

In Windows-Produkten stehen die Online-Bücher im Windows-Startmenü (Cognos) sowie im Hilfe-Menü des jeweiligen Produkts zur Verfügung (Bücher zum Drucken). Die komplette Online-Dokumentation steht auf der Cognos-Dokumentations-CD zur Verfügung. Dort finden Sie auch die Readme-Dateien und die Installationsanleitungen zu den einzelnen Produkten.

Nur die Installationshandbücher sind als Druckversion verfügbar.

Über das Windows-Startmenü oder das Hilfe-Menü von Impromptu ist die *Dokumentationsübersicht*, eine kommentierte Liste anderer Dokumentationen, verfügbar.

Fragen oder Anmerkungen?

Umgehende Antworten auf Ihre Fragen zu Impromptu erhalten Sie vom Kundendienst.

Informationen über die Adressen und Programme des Customer Supports finden Sie unter *Cognos im Web* im Hilfe-Menü oder auf der Website des Cognos-Supports (http://support.cognos.com).

Kapitel 1: UDFs auf einen Blick

In diesem Kapitel finden Sie Folgendes:

- allgemeine Hinweise zu UDFs (benutzerdefinierten Funktionen)
- eine Beschreibung der zwei Typen von UDFs und wie Sie diese in Ihrer Berichtsumgebung nutzen können
- Informationen über unterstützte Betriebssysteme
- Schritte zum Erstellen von UDFs

Überblick

Sie können UDFs für folgende Zwecke erstellen:

- Bereitstellen einer angepassten oder geschäftsbezogenen Funktion für Ihre Berichtsumgebung, die nicht in Ihrem Cognos-Produktpaket enthalten ist
- Implementieren einer Funktion als zentralen Speicher in einer Datenbank
- Zugriff auf Bit-Masken oder andere Anwendungen, die Daten verschlüsseln und verpacken
- Arbeiten mit benutzerdefinierten Datentypen, die in Datenbanken wie zum Beispiel Informix, DB2 oder Oracle auftreten

Typen von UDFs

Es gibt zwei Arten von UDFs:

- Datenbankfunktionen

  Datenbank-UDFs sind Funktionen, die Sie zur Verwendung mit einem oder mehreren Datenbanksystemen definieren können. Weitere Informationen über das Benutzen von UDFs in Ihrer Datenbankumgebung finden Sie in Ihrer RDBMS-Dokumentation.

  Hinweis: In Cognos Architect werden Datenbankfunktionen als gespeicherte Prozeduren bezeichnet.
- Externe Funktionen

  Externe UDFs werden in C oder einer anderen Sprache geschrieben (dafür muss die C-Aufrufkonvention unterstützt werden) und zur Verwendung mit Windows NT in DLL-Dateien kompiliert. Für die Verwendung in UNIX-Umgebungen werden externe UDFs in gemeinsam benutzte Bibliotheken kompiliert.

Cognos-Produkte unterstützen Scalar-UDFs - Funktionen, die bei jedem Aufrufen einen einzigen Wert zurückgeben.

Wie Cognos-Produkte Funktionsnamen auflösen

Um einen Funktionsnamen in einer SQL-Anweisung aufzulösen, suchen Cognos-Produkte in der folgenden Reihenfolge:

1. Datenbankfunktionen (systemeigene und UDFs), die sich auf eine bestimmte Instanz eines Datenbanktyps beziehen
2. Datenbankfunktionen, die sich nur auf den Datenbanktyp beziehen
3. externe UDFs
4. Funktionen, die in Ihrem Cognos-Paket enthalten sind

Wichtig!

- Sie sollten von allen .INI und .SQL-Dateien für UDFs, die Sie modifizieren möchten, Sicherungskopien herstellen, wenn Sie mit den Beispieldateien arbeiten oder eigene UDFs herstellen. Diese Dateien werden überschrieben, wenn Sie Ihr Cognos-Produkt erneut installieren, und jegliche Änderungen, die Sie eventuell beibehalten möchten, gehen verloren.
- Um Ihre UDFs einfach in zukünftige Ausgaben von Cognos-Produkten zu übertragen, sollten Sie den UDFs solche Namen geben, die bestimmt nicht in zukünftigen Ausgaben des Produkts vorkommen werden.

Unterstützte Betriebssysteme

Informationen über unterstützte Betriebssysteme und Versionen finden Sie auf der Website des Cognos-Supports (http://support.cognos.com).

UDF-Verfahren

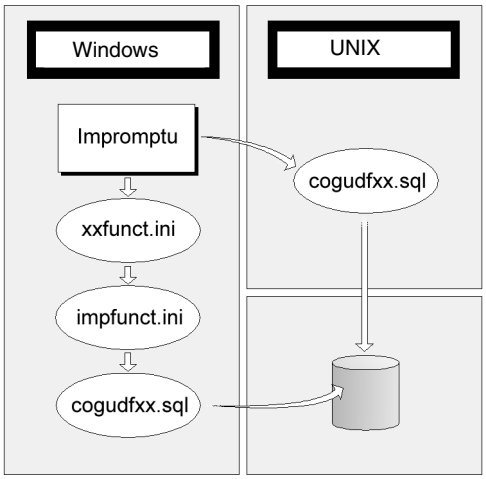
Implementieren Sie UDFs mit den folgenden Verfahren:

- Vorgehensweise zum Implementieren einer Datenbank-UDF
- Vorgehensweise zur Verwendung einer Datenbank-UDF mit mehreren Datenbanken
- Vorgehensweise zum Implementieren einer externen UDF

Der Anhang enthält eine Liste von zweistelligen Datenbankabkürzungen, mit denen die Namen der .SQL und .INI-Dateien gebildet werden, die Sie in diesen Verfahren modifizieren müssen.

Vorgehensweise zum Implementieren einer Datenbank-UDF

Das untenstehende Diagramm zeigt die Dateien, mit denen Cognos-Produkte auf eine Datenbank-UDF zugreifen:



1. Fügen Sie die Funktion zur Datenbank hinzu, sofern sie noch nicht vorhanden ist. (Weitere Informationen hierzu finden Sie in Ihrer RDBMS-Dokumentation.)
2. Fügen Sie den Funktionsnamen zur .INI-Datei für die Datenbank hinzu. Die Datei befindet sich standardmäßig im Ordner *Installationsverzeichnis*\cer*n*\bin.

   Das Format des Dateinamens lautet XXFUNCT.INI und stellt eine Kombination aus einer zweistelligen Kennung für die Datenbank und den Buchstaben „FUNCT" dar. Beispiel: Die .INI-Datei für Oracle lautet ORFUNCT.INI.

   Fügen Sie den Funktionsnamen in den Abschnitt [Database-specific Function List] der .INI-Datei ein.

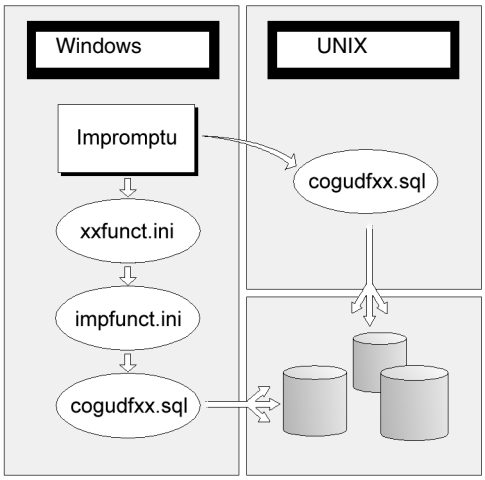
3. Fügen Sie den Funktionsnamen als neuen Abschnittsnamen am Ende der .INI-Datei ein. Der Name muss sich in eckigen Klammern befinden. Beispiel: [MeineFunktion].
4. Fügen Sie im neuen Abschnitt die Definition Ihrer Funktion ein.
5. Fügen Sie die Funktionsdefinition zur entsprechenden .SQL-Datei hinzu. Die Datei befindet sich standardmäßig im Ordner *Installationsverzeichnis*\cer*n*\bin. Das Format des Dateinamens lautet COGUDFXX.SQL und stellt eine Kombination aus den Buchstaben „COGUDF" und der zweistelligen Kennung für die Datenbank dar. Gibt es im Ordner *BIN* keine .SQL-Datei für die Datenbank, erstellen Sie die Datei.

   Mit Hilfe des Windows Syntaxprüfungs-Dienstprogrammes COGUDF.EXE (im SDK enthalten) können Sie überprüfen, was Sie zur .SQL-Datei hinzugefügt haben. Weitere Informationen finden Sie im Anhang auf (S. 35).

Wenn die UDF richtig zu allen erforderlichen Dateien hinzugefügt wurde, erscheint sie im Ausdruckseditor, wenn Sie Ihr Cognos-Produkt starten und zu einer Datenbank des Typs verbinden, die Ihre UDF benutzt.

Vorgehensweise zur Verwendung einer Datenbank-UDF mit mehreren Datenbanken

Das untenstehende Diagramm zeigt die Dateien, mit denen Cognos-Produkte auf eine Datenbank-UDF für mehrere Datenbanken zugreifen:



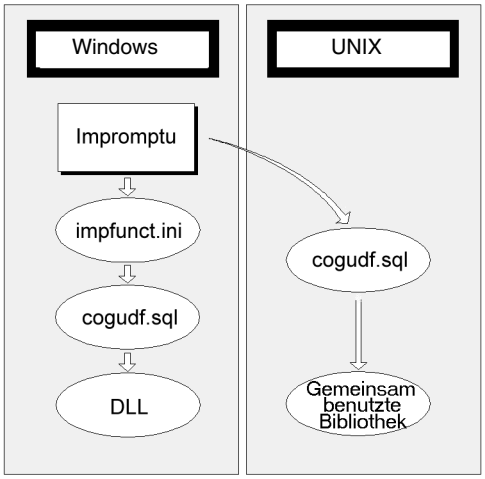
1. Fügen Sie die Funktion zur Datei IMPFUNCT.INI im Abschnitt [Common Database Function List] hinzu.
2. Fügen Sie den Funktionsnamen als neuen Abschnittsnamen am Ende der Datei IMPFUNCT.INI ein. Der Name muss sich in eckigen Klammern befinden. Beispiel: [Meine-Funktion].
3. Fügen Sie in der Datei IMPFUNCT.INI die vollständige Funktionsdefinition unter dem neuen Abschnittsnamen hinzu.
4. Fügen Sie die Funktionsnamen aller Datenbanken, für die Sie die Funktion verwenden wollen, zum Abschnitt [Common Database Function List] der .INI-Datei der entsprechenden Datenbank hinzu (XXFUNCT.INI).
5. Fügen Sie in allen Datenbanken, für die Sie die Funktion vewenden wollen, in der entsprechenden .SQL-Datei (COGUDFXX.SQL) des Bin-Ordners die Funktionsdefinition ein. Enthält eine Datenbank, die Sie mit der UDF verwenden wollen, keine .SQL-Datei, erstellen Sie die Datei.

   Mit Hilfe des Windows Syntaxprüfungs-Dienstprogrammes COGUDF.EXE (im SDK enthalten) können Sie überprüfen, was Sie zur .SQL-Datei hinzugefügt haben. Weitere Informationen finden Sie im Anhang auf (S. 35).

Wenn die UDF richtig zu allen erforderlichen Dateien hinzugefügt wurde, erscheint sie im Ausdruckseditor, wenn Sie Ihr Cognos-Produkt starten und zu den Datenbanken des Typs verbinden, die die UDF benutzen.

Vorgehensweise zum Implementieren einer externen UDF

Das untenstehende Diagramm zeigt die Dateien, mit denen Cognos-Produkte auf eine externe UDF zugreifen:



1. Erstellen Sie die DLL oder die gemeinsam benutzte Bibliothek.
   Die DLL sollte in einen der folgenden Ordner kopiert werden:
   - in den Ordner BIN
   - in eins der Verzeichnisse, die von der Umgebungsvariablen PATH festgelegt werden
   - in den Windows-Ordner
   - in den Windows-Systemordner

   In UNIX müssen Sie mit einer der folgenden Umgebungsvariablen auf die gemeinsam benutzte Bibliothek verweisen:
   - LD_LIBRARY_PATH (Solaris)
   - LIBPATH (AIX)
   - SHLIB_PATH (HP-UX)

   Weitere Informationen finden Sie in den UNIX-Man-Seiten für „ld".
2. Fügen Sie den Namen der externen UDF zum Abschnitt [Built-in Function List] der Datei IMPFUNCT.INI hinzu.
3. Fügen Sie den Funktionsnamen als neuen Abschnittsnamen am Ende der Datei IMPFUNCT.INI ein. Der Name muss sich in eckigen Klammern befinden. Beispiel: [Meine-Funktion].
4. Fügen Sie unterhalb des neuen Abschnittnamens die vollständige Definition Ihrer Funktion hinzu.
5. Fügen Sie die Funktionsdefinition zur Datei COGUDF.SQL hinzu.
   Mit Hilfe des Windows Syntaxprüfungs-Dienstprogrammes COGUDF.EXE (im SDK enthalten) können Sie überprüfen, was Sie zur .SQL-Datei hinzugefügt haben. Mit diesem Dienstprogramm können Sie die .SQL-Dateien auf Ihrem PC überprüfen, bevor Sie sie in Ihre UNIX-Umgebung kopieren. Weitere Informationen finden Sie im Anhang auf (S. 35).

Wenn die UDF richtig zu allen erforderlichen Dateien hinzugefügt wurde, erscheint sie im Ausdrucksеditor, wenn Sie Ihr Cognos-Produkt starten.

Kapitel 2: Modifizieren der .INI-Dateien

In diesem Kapitel werden Sie lernen,

- was die Datei IMPFUNCT.INI ist und wie man sie modifizieren kann
- wie die Struktur der Datenbank-spezifischen .INI-Dateien aussieht

Überblick

Wie im vorigen Kapitel bereits erwähnt, müssen Sie die .INI-Dateien modifizieren, um Ihre UDFs Ihrem Cognos-Produkt zur Verfügung zu stellen. Die .INI-Dateien befinden sich standardmäßig im Ordner *Installationsverzeichnis*\cer*n*\bin. Die Dateien, die Sie modifizieren müssen, sind folgende:

- IMPFUNCT.INI (für Datenbank-UDFs, die mit mehreren Datenbanksystemen verwendet werden sollen, sowie für externe UDFs)
- eine oder mehr Datenbank-spezifische .INI-Dateien (nur für Datenbank-UDFs)

Informationen über die Liste von Datenbank-spezifischen .INI-Dateien finden Sie im Anhang auf (S. 35).

Inhalt der Datei IMPFUNCT.INI

Die Datei IMPFUNCT.INI enthält Definitionen für

- allgemeine Datenbankfunktionen
- Datenbank-UDFs
- externe UDFs

Die Datei ist in mehrere Abschnitte unterteilt. Für das Erstellen von UDFs, die für mehrere Datenbanken verfügbar sind, modifizieren Sie den Abschnitt [Common Database Function List].

Hinweis

Wenn Sie eine Datenbank-UDF erstellen, dürfen Sie die existierenden Einträge im Abschnitt [Built-in Function List] nicht modifizieren. Wenn Sie jedoch eine externe UDF erstellen, müssen Sie den neuen Eintrag zu diesem Abschnitt hinzufügen.

Jeder Funktionsname wird in einer Einzelzeile gezeigt. Dem Namen folgt ein Gleichheitszeichen (=). Zum Beispiel:

```
A_Funktion=
B_Funktion=
```

Bei den Funktionsnamen wird nicht zwischen Groß- und Kleinschreibung unterschieden: A_Funktion ist dasselbe wie a_Funktion.

Weiter unten in der .INI-Datei befinden sich Abschnitte, die nach diesen Funktionen benannt wurden. Diese Abschnitte definieren, was die Funktionen ausführen und wie sie im Ausdruckseditor angezeigt werden.

Funktionsdefinitions-Eigenschaften

Jede Funktionsdefinition enthält eine Reihe von Eigenschaften. Jede Eigenschaft nimmt eine einzelne Zeile ein. Im folgenden Beispiel für eine absolute Funktion wurde die Zeile, die mit "tip=" beginnt, aus Gründen der Darstellung in diesem Dokument umgebrochen, sollte sich aber auf einer einzelnen Zeile in der .INI-Datei befinden.

```
[absolute]
label=Absolut
param=1
return=NM
1=NM;numeric_exp
exp=absolute ( ^1 )
tip=Syntax: ABSOLUTER-WERT (numerischer_Ausdr) \nGibt den
absoluten Wert von numerisch_Ausdr zurück. Das Vorzeichen für negative
Werte wird in positive geändert. Beispiele: ABSOLUTER-WERT (5)
gibt 5 zurück; ABSOLUTER-WERT (-5) gibt 5 zurück.
tip1=Numerischer Ausdruck
```

Die Beschriftungs-Eigenschaft ist optional, die anderen Eigenschaften sind jedoch erforderlich. Jeder Eigenschaftseintrag ist auf 4096 Zeichen begrenzt (außer dem Parameter *tip*). Das Limit dieses Parameters ist 512 Zeichen.

Im folgenden Text werden die Funktionseigenschaften beschrieben:

Funktionseigenschaften

Beschriftung

Optional. Der Name der Funktion, wie er in der Funktionsliste im Ausdruckseditor angezeigt wird. Ist keine Beschriftung angegeben, wird der Name der Funktion verwendet.

param

Gibt die Anzahl der Parameter für die Funktion an. Muss kleiner als 32767 sein.

return

Gibt den Typ des Wertes an, der von der Funktion zurückgegeben wird. Dieser Wert ist einer dieser zweibuchstabigen Codes:

NM Numerisch

CH Zeichen

DA Datum

DT Datum und Uhrzeit

IN Intervall

TM Uhrzeit

In komplexen Ausdrücken (zum Beispiel in Ausdrücken mit unterschiedlichen Rückgabewerten wie die Oracle Entschlüsselungsfunktion) ist der Wert ein Einzelzeichen im Bereich von „A" bis „Z" und gibt eine Parameterklasse an (siehe unten). Der Rückgabetyp ist derselbe wie der Typ der Parameterklasse. Beispiel:

return=**B**

1=DA,DT,NM,CH,IN,TM:A;Ausdruck

2=DA,DT,NM,CH,IN,TM:A;Suche1

3=DA,DT,NM,CH,IN,TM:**B**;Resultat1

4=DA,DT,NM,CH,IN,TM:A;Suche2

5=DA,DT,NM,CH,IN,TM:**B**;Resultat2

6=DA,DT,NM,CH,IN,TM:**B**;Standard

Parameterklasse A (Parameter 1, 2 und 4) hat denselben Typ, und Parameterklasse B (Parameter 3, 5 und 6) hat denselben Typ. Der Typ der Rückgabefunktion in diesem Beispiel ist der Typ der Parameterklasse B. Die Oracle Entschlüsselungsfunktion (aus der dieser Auszug stammt) kann wie in den folgenden Beispielen beschrieben aufgerufen werden (wobei NM? einen numerischen Wert, CH? einen Zeichenwert und DA? einen Datumswert bezeichnet):

decode (NM1, NM2, CH3, NM4, CH5, NM6, CH7, CH8) gibt ein Zeichen zurück

decode (DA1, DA2, NM3) gibt einen numerischen Wert zurück

<Parameterbeschreibung>

Jeder Parameter wird mit einer durch Kommata abgetrennten Liste möglicher Typen beschrieben, denen eine beschreibende Zeichenkette folgt, die mit einem Semikolon (;) getrennt ist.

Für jeden Parameter gibt es eine Beschreibungseigenschaft, die mit 1, 2, 3 usw. beschriftet ist.

Es gibt folgende Typen:

NM Numerisch

CH Zeichen

DA Datum

DT Datum und Uhrzeit

IN Intervall

TM Uhrzeit

Zu mehreren dieser Funktionen können Parameter verschiedener Typen weitergegeben werden. Um dies durchzuführen, listen Sie die verschiedenen Typen durch Kommata getrennt in der nummerierten Parameterbeschreibung auf. So akzeptiert die Funktion *add-days()* zum Beispiel sowohl einen Datums- als auch Datum-Uhrzeit-Wert als ersten Parameter.

1=DA,DT;date_exp

In komplexen Ausdrücken folgt der Liste von Typen für jede Parameterbeschreibung ein Doppelpunkt (:) und ein einzelnes Zeichen, das eine Parameterklasse angibt. Alle Parameter in einer Parameterklasse müssen dieselbe Liste von Typen in ihren Parameter-Beschreibungsattributen haben.

Alle aktuellen Parameter in einer Parameterklasse müssen vom selben Typ sein.

exp

Dies ist der Ausdruck. Die SQL für das Attribut *exp* muss gültig sein. Jedes Token wird durch eine oder mehr Leertasten oder Tabulatorzeichen getrennt.

In der generierten SQL-Anweisung werden Parameter zugewiesen, indem die Parametermarkierungen mit der für jeden Parameter generierten SQL ersetzt werden. Eine Parametermarkierung ist ein Caret-Zeichen (^), auf das eine Zahl folgt. Beispiel: ^1.

Es können komplexe Ausdrücken definiert werden (zum Beispiel das Übertragen des Eintrages *date-to-string* in die Datei IMPFUNCT.INI). Die meisten Definitionen jedoch bestehen aus dem Funktionsnamen, auf den eine durch Kommata getrennte Liste von Parametermarkierungen folgt.

Um auf eine externe UDF hinzuweisen, muss der Funktionsname in eckige Klammern ([...]) eingeschlossen sein:

```
exp=[meinefunkt ( ^1 , ^2 )
```

Mit eckigen Klammern werden darüber hinaus optionale Sequenzen gekennzeichnet. Ein Paar eckiger Klammern, auf die ein Sternchen (*) folgt, bezeichnet eine wiederholende Sequenz. So enthält die Entschlüsselungs-Funktionsdefinition die folgende Ausdrucksdefinition mit wiederholten und optionalen Sequenzen:

```
exp=decode ( ^1 , ^2 , ^3 [ , ^4 , ^5 ]* [ , ^6 ] )
```

[, ^4, ^5]* ist wiederholend, [, ^6] ist nicht wiederholend. In einer Funktionsdefinition kann eine wiederholte und eine optionale Sequenz gemeinsam auftreten. Es kann jedoch nur je eine dieser Funktionsdefinitionen vorliegen. Die wiederholende Sequenz (falls vorhanden) muss der optionalen Sequenz (falls vorhanden) vorausgehen.

In wiederholenden Sequenzen verarbeiten Cognos-Produkte alle von Ihnen angegebene Parameter, bis keiner mehr übrig ist.

Hier ist ein weiteres Beispiel für die Funktion einer wiederholenden Sequenz. Wenn Sie Folgendes definieren:

```
exp=FooFunc( ^1 [, ^2 , ^3]*)
```

wird die Funktion wie folgt aufgerufen:

```
FooFunc(A, B, C)
FooFunc(A, B, C, D, E)
FooFunc(A, B, C, D, E, F, G)
```

tip

Gibt den Anzeigetext an, der im Ausdrucksеditor für die Funktion angezeigt wird. Diese Zeichenkette darf nicht mehr als 512 Zeichen haben.

parameter tip

Gibt den Anzeigetext für die Parameter an. Für jeden Parameter muss ein "parameter tip" vorliegen. Parameter tips haben die Beschriftungen tip1, tip2, tip3 und so weiter.

type

Gibt an, dass die Funktionsdefinition einen wiederholten Parameter oder einen dynamischen Rückgabetyp oder beide enthält. Der Wert dieser Funktionseigenschaft muss eine Zeichenkette sein, die die Werte D oder R oder DR enthält:

- D — Rückgabetyp ist dynamisch (kann abhängig von den Parametertypen variieren)
- R — Parameter können wiederholend sein.
- DR — Parameter können wiederholend sein, und der Rückgabetyp ist dynamisch.

Datenbank-spezifische .INI-Dateien

Das Format der Dateinamen für die Datenbank-spezifischen .INI-Dateien ist XXFUNCT.INI und stellt eine Kombination aus der zweistelligen Kennung für die Datenbank und der Buchstabenfolge FUNCT dar. Eine Liste der zweistelligen Identifizierungen finden Sie im Anhang.

Datenbank-spezifische .INI-Dateien werden verarbeitet, wenn eine Verbindung zur Datenbank hergestellt wird.

[Built-in Function List]

Dieser Abschnitt hat dieselbe Syntax wie die Datei IMPFUNCT.INI. Eine hier aufgelistete Funktion wird in der Datenbank als native SQL implementiert. Beachten Sie, dass Sie externe UDF-Definitionen zum Abschnitt [Built-in Function List] in der Datei IMPFUNCT.INI hinzufügen und nicht zu einer Datenbank-spezifischen .INI-Datei.

[Common Database Function List]

Dieser Abschnitt listet gemeinsam benutzte Datenbankfunktionen auf, die von der Datenbank implementiert werden. Eine Funktion, die nicht im Abschnitt [Common Database Function List] in der Datei IMPFUNCT.INI aufgelistet ist, wird nicht von der Datenbank implementiert und erscheint nicht in der Funktionsliste im Ausdrucksеditor.

[Database-specific Function List]

Dieser Abschnitt listet Funktionen auf, die in der entsprechenden Datenbank implementiert werden.

Individuelle Funktionsdefinitionen

Das Format dieses Abschnitts ist dasselbe wie in der Datei IMPFUNCT.INI. Sie können die Beschreibungsattribute "exp" und "parameter description" in der Datei IMPFUNCT.INI überschreiben, indem Sie diese hier neu definieren. So können Sie diese Attribute zum Beispiel überschreiben, um die Funktion "exp" in eine Datenbank-spezifische Syntax zu ändern oder die Parametertypen zu modifizieren, die diese Funktion akzeptiert. Sie können keine anderen Attribute überschreiben.

Für die Funktionen, die im Abschnitt [Database-specific Function List] aufgelistet werden, müssen in diesem Abschnitt alle Attribute definiert sein.

Wird in diesem Abschnitt eine Funktion zweimal definiert, dann wird nur die letzte Definition verwendet.

Kapitel 3: Implementieren von Datenbank-UDFs

In diesem Kapitel werden Sie lernen, wie

- die entsprechenden .SQL-Dateien modifiziert werden
- die Syntax für das Definieren der Datenbank-UDFs benutzt wird
- überladene Datenbank-UDFs definiert werden

Datenbank-SQL-Dateien (COGUDFXX.SQL)

Um eine Datenbank-UDF (benutzerdefinierte Definition) einzurichten, müssen Sie folgende Schritte ausführen:

- Hinzufügen der Funktion zur Datenbank (falls erforderlich)
  Weitere Informationen finden Sie in Ihrer RDBMS-Dokumentation.
- Modifizieren der .INI-Datei für den Datenbanktyp
- Modifizieren der IMPFUNCT.INI-Datei, wenn die UDF mit mehreren Datenbanksystemen benutzt werden soll
- Hinzufügen der Funktionsdefinition zur .SQL-Datei für den Datenbanktyp.

Eine SQL-Datei für eine Datenbank-UDF enthält UDF-Definitionen und kann darüber hinaus Kommentare enthalten. Sie können Kommentare mit Hilfe von zwei Bindestrichen (- -) erstellen. Jeglicher Text, der diesen Bindestrichen in derselben Zeile folgt, wird ignoriert.

Hinweis: Die COGUDFXX.SQL-Dateien dürfen keine Anweisungen in der Cognos SQL Data Manipulation Language (DML)-Sprache enthalten (wie zum Beispiel SELECT oder INSERT).

Syntax für das Definieren einer Datenbank-UDF

Die fettgedruckten Einträge sind obligatorisch. In eckigen Klammern [ ] gezeigte Einträge sind optional. In geschweifte Klammern { } gesetzte Einträge können vorhanden sein. Das Format sowie die Groß- und Kleinschreibung der von Ihnen benutzten Datenbanknamen müssen mit den Namen übereinstimmen, mit denen Sie die UDF in der Datenbank definiert haben.

```
<Deklaration für die Datenbankfunktion> ::=
  DECLARE [ DATABASE] [ <Funktionstyp> ] FUNCTION
  <Funktionsname> [<Formale Parameterliste>]
  RETURNS <Datentyp>
  FUNCTION NAME <Datenbank-Funktionsname> ;

<Funktionstyp> ::=
      SCALAR

<Funktionsname> ::=
      [<Logischer Datenbankname>.]<Identifizierung>

<Formale Parameterliste> ::=
      ([<Datenyp>] [{ , <Datenyp>} … ] )

<Zurückgegebener Datentyp> ::=
      <Datentyp>

<Datenbankfunktions-Name> ::=
    : [<Katalog>.][<Schema>.]<Funktionsname>

< Logischer Datenbankname> ::=
  : <Identifizierung>

<Katalog> ::=
  : <Identifizierung>

<Schema> ::=
  : <Identifizierung>

<Funktionsname> ::=
  : <Identifizierung>

<Identifizierung> ::=
  : Text
  | "<Text>"

<Datentyp> ::=
     STRING
   | BOOLEAN
   | NUMBER
   | BINARY
   | DATE
   | TIME
   | TIMESTAMP
   | INTERVAL
   | BLOB
   | TEXT
   | <Ausdrücklicher Wert>

<Ausdrücklicher Wert> ::=
  : '<Text>'
```

Syntax-Beschreibung

- Ein Parameter kann vom Typ BINARY sein, der zurückgegebene Wert einer Funktion kann jedoch nicht BINARY sein.
- Der ursprüngliche <Funktionsname> ist der Name, mit dem das Cognos-Produkt die UDF identifiziert.
- Der Typ STRING repräsentiert Zeichenketten mit fester oder variierender Länge.
- Der Typ NUMBER repräsentiert die folgenden Typen numerischer Werte:
  - SMALLINT
  - INTEGER
  - DECIMAL
  - NUMERIC
  - REAL
  - FLOAT
  - DOUBLE PRECISION
- Sie können angeben, dass eine Datenbankfunktion keine Parameter hat und keine Klammern benötigt, indem Sie in der Funktionsdefinition keine Klammern für die Parameterliste einschließen.
- Der <Datenbankfunktionsname> ist der Name der zugrundeliegenden Datenbank, mit der die Funktion identifiziert wird. Damit kann auf Datenbankfunktionen, die in einem anderen Schema oder Katalog definiert sind, in der SQL mit einem einfachen, benutzerdefinierten Namen verwiesen werden. Der Name wird genauso benutzt, wie Sie ihn eingeben - wenn Sonderzeichen oder getrennte Namen erforderlich sind, sollte der gesamte Funktionsname mit doppelten Anführungszeichen getrennt werden. Eingebettete doppelte Anführungszeichen werden durch zwei aufeinanderfolgende doppelte Anführungszeichen angegeben.
- Ein Parameter kann einen festen Wert haben, der durch eine mit einzelnen Anführungszeichen getrennte Textzeichenkette angegeben wird. Eingebettete einzelne Anführungszeichen werden durch zwei aufeinanderfolgende einzelne Anführungszeichen angegeben. Ein Parameter mit einem angegebenen, festen Wert kann mit keinem anderen Parameter-Datentyp kombiniert werden.
- Der zurückgegebene Wert einer Datenbankfunktion kann kein ausdrücklicher Wert sein.

Beispiele für Datenbank-UDF-Definitionen

Die .SQL-Dateien enthalten Definitionen ähnlich der folgenden:

```
DECLARE FUNCTION substitute( STRING, STRING )
RÜCKGABE-ZEICHENKETTE
FUNCTION NAME löwe.tiger.subst;

DECLARE DATABASE FUNCTION localDB.timeOfDay()
RETURNS TIME
FUNKTIONSNAME sysfunc.time_of_day;

Declare Database Function "Meine benutzerdefinierte Funktion"
( Zahl, 'Fester Wert hier!' )
Gibt Zahl zurück
Funktionsname "Mein Katalog"."Mein Schema"."Benutzerdefinierte Funktion";
```

Überladene Datenbank-Funktionen

Cognos-Produkte können auf Datenbank-UDFs zugreifen, ohne beachten zu müssen, wie diese in der zugrundeliegenden Datenbank implementiert sind. Es muss jedoch möglich sein, auf die Funktionen mit einer SQL-Anweisung zuzugreifen.

Einige Datenbanken, wie zum Beispiel DB2 und Informix, unterstützen die Funktionsüberladung. Überladen bedeutet, dass zwei oder mehr Funktionen mit demselben Namen vorliegen können. Überladene Funktionen werden durch die Anzahl und Typen der Parameter, die jede Funktion enthält, unterschieden.

Cognos-Produkte unterstützen momentan die Funktionsüberladung nicht. Jede Funktion muss einen eindeutigen Namen haben. Um auf die verschiedenen Versionen einer überladenen Funktion zugreifen zu können, muss für Cognos-Produkte jede Instanz der Funktion in der entsprechenden .SQL-Datei für die Datenbank eindeutig identifiziert sein.

Die Syntax für das Identifizieren der Funktion ist dieselbe - unabhängig davon, ob die Datenbank-UDF in C, Java oder in einer systemgebundenen Datenbank-Prozedursprache wie zum Beispiel PL/SQL geschrieben wurde.

Beispiel

Die Funktion MEINEFUNK ist in DB2 zweifach definiert. Die erste Definition ist:

```
CREATE FUNCTION meinefunk (SMALLINT)
    RETURNS SMALLINT
    EXTERNAL NAME mylib!meinefunk
    LANGUAGE C...
```

Die zweite Definition ist:

```
CREATE FUNCTION meinefunk (FLOAT, FLOAT)
    RETURNS FLOAT
    EXTERNAL NAME meinelib!meinefloatfunc
    LANGUAGE C...
```

Um diese UDFs in einer Cognos-SQL-Anweisung verwenden zu können, müssen Sie die zwei Funktionen in der Datei COGUDFD2.SQL wie folgt definieren:

```
DECLARE DATABASE FUNCTION meineIntFunc (NUMBER)
RETURNS NUMBER
FUNCTION NAME meinefunk;

DECLARE DATABASE FUNCTION meineFloatFunc (NUMBER, NUMBER)
RETURNS NUMBER
FUNCTION NAME meinefunk;
```

Oracle-Funktionen

Das untenstehende Beispiel ist in C-Sprache. Sie können eine UDF in einer Oracle-Datenbank jedoch auch mit PL/SQL definieren. Weitere Informationen finden Sie in Ihrer Oracle-Dokumentation.

Beispiel

```
CREATE FUNCTION Stadt_finden
( längengrad IN FLOAT,
  breitengrad IN FLOAT )
RETURN FLOAT AS EXTERNAL
LIBRARY meinelibrary
NAME "Stadt"
LANGUAGE C;
```

Die Definition dieser Funktion in der Date COGUDFOR.SQL ist folgende:

```
DECLARE DATABASE FUNCTION Stadtsuchen (NUMBER, NUMBER)
RÜCKGABE-ZEICHENKETTE
FUNCTION NAME Stadt_suchen;
```

DB2-Funktionen

Wie oben bereits beschrieben, unterstützt DB2 Funktions-Überladung. In Ihrem Cognos-Produkt müssen eindeutige Namen benutzt werden, um jede Version der überladenen Funktion einzeln zu identifizieren.

Beispiel

```
CREATE FUNCTION stadt_suchen (FLOAT, FLOAT)
RETURNS VARCHAR(50)
EXTERNAL NAME "meinelibrary!Stadt"
LANGUAGE C
PARAMETER STYLE DB2SQL
DETERMINISTIC
NO SQL
NO EXTERNAL ACTION
```

Die Funktions-Deklaration in der Datei COGUDFD2.SQL ist dieselbe wie für das Oracle-Beispiel.

Informix-Funktionen

Cognos-Produkte unterstützen keine Informix Statement Local Variables (SLVs). Eine SLV ist eine Variable in einer gespeicherten Prozedur, auf die Sie in der aufrufenden SQL-Anweisung verweisen können. Weitere Informationen über SLVs finden Sie in Ihrer Informix-Dokumentation.

Informix unterstützt die Funktionsüberladung. Auch hier müssen in Ihrem Cognos-Produkt eindeutige Namen benutzt werden, um jede Version der überladenen Funktion einzeln zu identifizieren.

UDFs für Informix Universal Server können außerhalb der Datenbank geschrieben werden. UDFs für Informix Online werden als Prozeduren geschrieben. Das folgende Beispiel ist für Informix Universal Server 9.12.

Beispiel

```
CREATE FUNCTION Stadt_suchen (längengrad FLOAT, breitengrad FLOAT)
RETURNING VARCHAR(50)
EXTERNAL NAME "/usr/lib/local/stadt.so"
LANGUAGE C
END FUNCTION;
```

Die Funktions-Deklaration in der Datei COGUDFIF.SQL ist dieselbe wie für das Oracle-Beispiel.

Kapitel 4: Implementieren externer UDFs

In diesem Kapitel werden Sie lernen,

- wie Sie Ihre externen UDFs (benutzerdefinierten Funktionen) zur Datei COGUDF.SQL hinzufügen
- wie Sie externe UDFs kodieren
- welche Anforderungen und Aspekte beim Entwerfen einer externen UDF-Bibliothek berücksichtigt werden müssen

Überblick

Um eine externe UDF zu implementieren, müssen Sie eine gemeinsam benutzte Bibliothek oder DLL erstellen und installieren und anschließend

- die Definitionen von externen UDFs zur Datei COGUDF.SQL hinzufügen
- Namen und Definition der Funktion zur Datei IMPFUNCT.INI hinzufügen

In diesem Kapitel wird erläutert, wie ein Eintrag zur Datei COGUDF.SQL hinzugefügt und eine externe UDF-Bibliothek entworfen wird.

Anforderungen und Beschränkungen

Für externe UDFs treffen folgende Anforderungen und Beschränkungen zu:

- Sie können externe UDFs mit jeder Programmiersprache außer C entwerfen. Die UDF muss jedoch die C-Aufrufkonvention unterstützen.
- Der Name, den Sie einer externen UDF geben, muss eindeutig sein.
- Eine externe UDF kann maximal 16 Parameter aufnehmen.
- Parameter können nicht mehr als einen Datentyp annehmen.
- Binäre Werte, Text und BLOBS werden weder als Parameter in UDFs noch als von externen UDFs zurückgegebene Werte unterstützt.
- Alle Dateneinträge, die an externe UDFs übergeben oder daraus zurückgegeben werden, werden in Abhängigkeit vom Datentyp ausgerichtet.
- Alle externen UDFs müssen einen leeren („void") Wert zurückgeben.

Die externe UDF SQL-Datei (COGUDF.SQL)

Die Datei COGUDF.SQL befindet sich standardmäßig im Ordner *Installationsverzeichnis*\cer*n*\bin. Cognos-Produkte lesen diese Datei, wenn sie eine Funktion antreffen, die nicht als Datenbankfunktion erkannt wird. Die Datei COGUDF.SQL teilt Ihrem Cognos-Produkt folgende Informationen mit:

- wo die UDF gefunden werden kann
- den Namen der UDF in C
- den Typ für jeden Parameter
- den Rückgabetyp der UDF

Suchpfade für .SQL-Dateien

In einer Windows-Umgebung werden die .SQL-Dateien in folgenden Pfaden gesucht:

1. im aktuellen Arbeitsordner
2. im Ordner, der im Abschnitt`servicesL` in der COGNOS.INI-Datei angegeben wird
3. in den Ordner BIN

In UNIX-Umgebungen werden die .SQL-Dateien in folgenden Pfaden gesucht:

1. im aktuellen Arbeitsverzeichnis
2. in dem Verzeichnis, auf das mit der Umgebungsvariablen `COGUDFSQL` verwiesen wird
3. in dem Verzeichnis, auf das mit der Umgebungsvariablen `DMDBIMI` erwiesen wird

Inhalt

Die Datei COGUDF.SQL enthält Anweisungen in Cognos SQL Data Definition Language (DDL)-Sprache. Sie darf keine Anweisungen in Cognos SQL Data Manipulation Language (DML)-Sprache (wie zum Beispiel SELECT oder INSERT) oder Datenbank-UDF-Definitionen enthalten.

Wie andere .SQL-Dateien kann auch die Datei COGUDF.SQL Kommentare enthalten. Sie können Kommentare mit Hilfe von zwei aufeinanderfolgenden Bindestrichen (- -) erstellen.

Syntax für eine externe UDF

In der untenstehenden Syntax sind in eckigen Klammern [ ] gezeigte Einträge optional. In geschweiften Klammern { } gezeigte Einträge können mehrmals oder gar nicht vorkommen.

```
<Deklaration der externen Funktion> ::=
  DECLARE EXTERNAL [ <Funktionstyp> ]
  FUNCTION <Funktionsname> ( <formale Parameterliste> )
  RETURNS <Datentyp> [ <Resultatausgabe> ]
  FUNCTION NAME <Name der externen Funktion> ;

<Funktionstyp> ::=
      SCALAR

<Funktionsname> ::=
      <Identifizierung>

<Formale Parameterliste> ::=
      [<Datentyp>] [ { , <Datentyp> } … ]

<Resultatausgabe> ::=
      CAST AS <Datentyp>

<Name der externen Funktion> ::=
      [ <Modulname> . ] <Name der externen Funktion>

<Modulname> ::=
      <Text>

<Name der externen Funktion> ::=
      <Text>

<Identifizierung> ::=
  : <Text>
  | „Text"
<Datentyp> ::=
      CHARACTER <optionale Länge>
    | CHAR <optionale Länge>
    | BINARY <optionale Länge>
    | SMALLINT <optionale Skalierung>
    | INTEGER <optionale Skalierung>
    | QUADWORD <optionale Skalierung>
    | DECIMAL <Präzision und Skalierung>
    | NUMERIC <Präzision und Skalierung>
    | REAL
    | DOUBLE PRECISION
    | DATE
    | TIME
    | TIMESTAMP
    | INTERVAL

<optionale Länge> ::=
   ( <Länge> )

<optionale Skalierung> ::=
   ( <Skalierung> )

<Präzision und Skalierung> ::=
   ( <Präzision> [, <Skalierung> ] )

<Länge> ::=
   <Ganzzahl ohne Vorzeichen>

<Präzision> ::=
   <Ganzzahl ohne Vorzeichen>

<Skalierung> ::=
   <Ganzzahl mit Vorzeichen>
```

Syntax-Beschreibung

- <Funktionsname> ist der Name, mit dem das Cognos-Produkt die externe Funktion identifiziert. Wie bereits erwähnt müssen die Namen der Funktionen eindeutig sein. Wenn ein doppelter Name angetroffen wird, dann wird nur die erste Funktion dieses Namens anerkannt.

- CHARACTER und CHAR sind Synonyme.
- Für einen Parameter des Typs CHARACTER darf keine Größe angegeben sein. Die Größe aller Zeichenparameter wird durch den Wert der Eingabe in die Funktion vor der Ausführung bestimmt. Wenn der Rückgabetyp einer Funktion CHARACTER ist, muss eine Länge angegeben werden. Dies ist die maximale Größe, die von der externen UDF gehandhabt werden kann.
- Ein Parameter kann vom Typ BINARY sein, aber der zurückgegebene oder ausgegebene Wert einer Funktion kann nicht BINARY sein.
- Der <Modulname> des <Namens der externen Funktion> ist der Name der Dynamic Link Library (DLL) oder der gemeinsam benutzten Bibliothek, die diese Funktion enthält. Der <Funktionsname> des <Namens der externen Funktion> ist der Name der Funktion in der Bibliothek. Beispiel: meineDLL.meineFunktion.
- Die Standardskalierung der Werte SMALLINT, INTEGER und QUADWORD ist Null. Eine Skalierung, die größer als Null ist, gibt die Anzahl der Stellen an, die nach dem Dezimalzeichen angezeigt werden müssen. Eine Skalierung, die kleiner als Null ist, gibt die Anzahl zusätzlicher Stellen an, die vor dem Dezimalzeichen angezeigt werden müssen. Wenn zum Beispiel ein INTEGER-Wert von 123 die Skalierung 2 hat, ist sein eigentlicher Wert 1,23. Bei einer Skalierung von -2 ist der eigentliche Wert 12300.
- Die Standardskalierung der Werte DECIMAL und NUMERIC ist Null. DECIMAL- und NUMERIC-Skalenwerte sind entweder Null oder größer als Null.
- Bei Cognos-Funktionsnamen muss die Groß- und Kleinschreibung nicht beachtet werden, für externe Funktionsnamen hingegen schon.

Beispiele für externe-UDF-Definitionen

Die .SQL-Datei enthält Definitionen wie folgende:

```
DECLARE EXTERNAL FUNCTION substitute( CHAR, CHAR )
SCALAR RETURNS CHAR(10) CAST AS CHAR(50)
FUNCTION NAME rxxfunc.subst;

DECLARE EXTERNAL FUNCTION timeOfDay()
RETURNS TIME
FUNKTIONSNAME sysfunc.time_of_day;
```

Diese zwei Funktionen werden in C wie nachfolgend gezeigt deklariert:

```
void subst( CogChar *, CogChar *, CogChar ** )
{    /* Code goes here. */
    return;

} /* subst */

void time_of_day( CogTime ** )
{    /* Code goes here. */
    return;

} /* time_of_day */
```

Im obigen Beispiel wird angenommen, dass die Bibliothek vom dynamischen Lader ohne die vollständige Pfadbeschreibung gefunden werden kann. Der Name der DLL oder gemeinsam benutzten Bibliothek kann ein einfacher Name sein oder kann einen Pfad und Dateierweiterungen haben. Wenn der Name einen Pfad oder Dateierweiterungen enthält, muss der Name der Bibliothek in doppelte Anführungszeichen eingeschlossen sein. Das folgende Beispiel zeigt eine gültige DECLARE FUNCTION Anweisung:

```
DECLARE EXTERNAL FUNCTION meineFunktion( INTEGER )
RETURNS INTEGER
FUNCTION NAME "/usr/local/lib/mylib.so.1".meine_Funktion;
```

Erstellen einer externen UDF-Bibliothek

Überlegungen für Windows-Umgebungen

Alle UDFs müssen aus der DLL exportiert werden, in der sie sich befinden, damit Ihr Cognos-Produkt darauf Zugriff hat. Ihr Cognos-Produkt kann auf mehrere DLLs, die UDFs enthalten, gleichzeitig zugreifen.

DLLs, die UDFs enthalten, müssen sich in folgenden Pfaden befinden:

- im aktuellen Verzeichnis der Anwendung, die ausgeführt werden soll
- in einem der Verzeichnisse, die von der Umgebungsvariablen PATH festgelegt werden
- im Windows-Verzeichnis
- im Windows-Systemverzeichnis

Alle DLLs müssen 32-Bit und mit einer 8-Byte Ausrichtung kompiliert sein.

Überlegungen für UNIX-Umgebungen

UDFs können in einer beliebigen Anzahl von gemeinsam benutzten Bibliotheken vorliegen. Die gemeinsam benutzte Bibliothek für die UDFs muss sich in einem Verzeichnis befinden, auf das der Laufzeit-Linker Zugriff hat. Sie können mit den folgenden UNIX-Umgebungsvariablen auf eine gemeinsam benutzte Bibliothek für UDFs verweisen:

- LD_LIBRARY_PATH (Solaris)
- LIBPATH (AIX)
- SHLIB_PATH (HP-UX)

Die Datei COGUDF.H

Die Datei COGUDF.H sollte in jede Datei mit eingeschlossen werden, die UDFs enthält. Diese Datei ermöglicht den Zugriff auf alle erforderlichen Makros, Typdefinitionen und Funktionsdefinitionen. Die Datei COGUDF.H enthält die Datei COGUDFTY.H, so dass Sie letztere nicht ausdrücklich mit einschließen müssen.

Wenn eine Bibliothek mit UDFs eine der in COGUDF.H definierten Funktionen verwendet, muss die Bibliothek mit der UDF-Bibliothek (UDFLIB.LIB in Windows, UDFLIB.A in UNIX) verbunden sein.

Eine externe UDF kann Funktionen aus verschiedenen Bibliotheken (statische oder gemeinsam benutzte) oder aus DLLs verwenden. Eine UDF muss jedoch vor dem Abschließen ihrer Aufgabe allen zugewiesenen Speicherplatz freigeben.

Überprüfen des NULL-Parameters

Alle Parameter werden durch Verweisung an eine UDF weitergegeben. Der Wert NULL für einen bestimmten Parameter gibt an, dass der zugeordnete Wert aus der teilweise fertiggestellten SQL-Abfrage selbst NULL ist.

Da die Verweisung auf einen NULL-Parameter zum unnormalen Abbruch Ihres Cognos-Produkts führt, muss eine UDF vor der weiteren Verarbeitung prüfen, ob einer ihrer Parameter NULL ist.

Wenn die UDF einen NULL-Parameter findet, sollte ein Wert von NULL zurückgegeben werden.

Der folgende Code zeigt, wie auf NULL-Werte geprüft wird:

```
Void meineNeueFunktion( CogInt32 *  param1,
                    CogInt32 *  param2,
                    CogInt32 ** result );
{
    if( param1 == NULL || param2 == NULL )
    {
        *result = NULL;
    }
    else
    {
        **result = *param1 - *param2;
    }

    return;

} /* meineNeueFunktion */
```

Der Rückgabewert aller UDFs ist der letzte Parameter der Funktion. Dieser Wert wird stets durch einfach-indirekten Zugriff auf NULL gesetzt, und durch doppelt-indirekten Zugriff wird ihm ein Wert zugewiesen (wie oben gezeigt).

Eine UDF kann bis zu 16 Eingabeparameter plus die zusätzlichen Parameter für den Rückgabewert aufnehmen.

Fehlerbehandlung

Für UDFs sind momentan keine Fehlerbehandlungen verfügbar. Wenn ein Fehler auftritt, muss eine UDF in der Lage sein, einen entsprechenden Wert wiederherzustellen und zurückzugeben.

Datentypen

Für UDFs steht der folgende feste Satz von Datentypen zur Verfügung:

Datentyp	Beschreibung
CogChar *	Ein Zeiger zu einer mit Null beendeten Zeichenkette.
CogInt16	16-Bit Ganzzahl mit einem Vorzeichen.
CogInt32	32-Bit Ganzzahl mit einem Vorzeichen.
CogInt64	64-Bit Ganzzahl mit einem Vorzeichen.
CogFloat32	32-Bit Gleitkomma-Wert.
CogFloat64	64-Bit Gleitkomma-Wert.
Packed Decimal (CogUInt8)	Exakter numerischer Wert mit fester Länge (unten beschrieben).
CogDate	Binäre Darstellung eines Jahr/Monat/Tag-Wertes.
CogTime	Binäre Darstellung eines Stunde:Minute:Sekunde:Millisekunde-Wertes.
CogTimestamp	Binäre Darstellung eines Tag/Uhrzeit-Wertes.
CogInterval	Binäre Darstellung eines Tagesstunde:Minute:Sekunde:Millisekunde-Wertes.
CogBinary	Binäre Daten mit fester Länge.

Diese Datentypen (außer Packed Decimal) werden in der C-Header-Datei COGUDFTY.H definiert.

Die Datentypen CogChar, CogInt16, CogInt32, CogFloat32 and CogFloat64 werden durch zugrundeliegende C-Datentypen repräsentiert. Außer dem Datentyp CogInt64 werden die Datentypen als Eigenschaftswerte repräsentiert. Der Datentyp CogInt64 wird als nativer 64-Bit Ganzzahl-Datentyp in allen von UDFs unterstützten Betriebssystemen unterstützt.

CogInt64 Native Datentypen	
Windows	LONGLONG
Solaris	long long
HP-UX	long long
AIX	long long

Jede UNIX-Plattform erfordert eine bestimmte Compiler-Option, um die Unterstützung für 64-Bit-Ganzzahlen zu aktivieren. Weitere Informationen finden Sie in Ihrer UNIX-Dokumentation.

Packed decimal-Datentyp

Packed decimal-Werte werden mit einem einzelnen Nibble (4 Bits) für jede Ziffer in einer Zahl repräsentiert. Alle Werte werden in führende und nachfolgende Nullen eingebettet, um sicherzustellen, dass eine feste Anzahl von Ziffern stets durch einen Packed decimal-Wert repräsentiert wird. Um anzugeben ob das End-Nibble (das Halbbyte am weitesten rechts) eines Packed decimal-Wertes positiv oder negativ ist, wird ihm einer der folgenden Werte zugewiesen:

- `COG_DECIMAL_POSITIVE_1`
- `COG_DECIMAL_POSITIVE_2`
- `COG_DECIMAL_NEGATIVE`

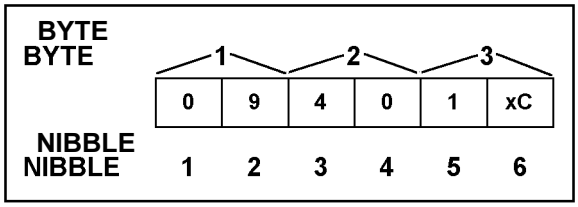
Diese Werte sind in COGUDFTY.H definiert. Beachten Sie, dass für Ihre UDF die Überprüfung auf negative Indikatoren einfacher ist als die Überprüfung auf positive Indikatoren.

Die maximale Präzision eines Packed decimal-Wertes beträgt 77. Die Skalierung eines Packed decimal-Wertes muss größer als oder gleich Null und kleiner als die oder gleich der Präzision sein.

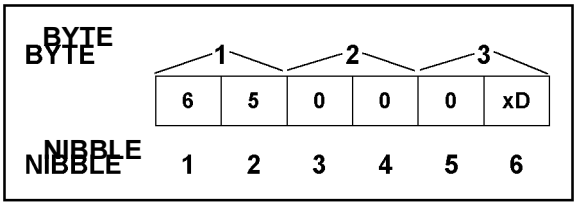
Die Anzahl der zum Repräsentieren eines numerischen Wertes erforderlichen Bytes hängt von der Präzision des Datentyps und davon ab, ob die Präzision gerade oder ungerade ist. Die Anzahl von Bytes, die zum Repräsentieren eines Packed decimal-Wertes erforderlich sind, wird wie folgt berechnet:

```
Anzahl von Bytes- (Anzahl der Ziffern / 2 ) + 1
```

Wenn ein Parameter zum Beispiel mit Präzision 4 und Skalierung 2 definiert ist, wird der Wert 94,01 wie folgt repräsentiert:



Wenn ein Parameter zum Beispiel mit Präzision 5 und Skalierung 3 definiert ist, wird der Wert -65 wie folgt repräsentiert:



Beachten Sie, dass im obigen Beispiel der Wert -650 in einem Dezimalparameter nicht mit Präzision 5 und Skalierung 3 dargestellt werden kann. Wenn Sie UDFs für das Akzeptieren oder Zurückgeben dezimaler (numerischer) Werte definieren, müssen Sie sicherstellen, dass nur akzeptable Werte an die Funktionen weitergegeben werden oder der Bereich akzeptierter Werte umfangreich ist (hohe Präzision). Wenn Ihre Anwendung keine höhere Genauigkeit als 19 Stellen erfordert, können Sie stattdessen CogInt64-Werte verwenden.

UDFs bearbeiten Packed decimal-Werte als eine Reihe von vorzeichenlosen Bytes, deren Anzahl von der Anzahl der Stellen im Packed decimal-Wert bestimmt wird (wie oben beschrieben).

Die folgenden zwei Funktionen werden mit dem SDK für die UDF geliefert, um Packed decimal-Werte in Zeichenketten und aus Zeichenketten zu konvertieren. Die Zeichenketten müssen vor dem Aufrufen der Funktion *cogDecimalToString* lange genug zugewiesen werden, um den resultierenden Wert zu halten.

```
CogChar * cogDecimalToString(
    CogUInt8 * pDecimal,
    CogChar  * pString
    CogInt32   precision,
    CogInt32   scale,
);

CogUInt8 * cogStringToDecimal(
    CogChar  * pString,
    CogUInt8 * pDecimal
    CogInt32   precision,
    CogInt32   scale
```

);

Es folgt die Definition einer UDF, die Packed decimal-Werte als Eingabeparameter akzeptiert und einen 32-Bit-Gleitkommawert zurückgibt:

```
void meinePDFunktion( CogUInt8 * param1, CogFloat32 ** result )
{
    /*
     * Präzision und Skalierung sind für eine UDF vordefiniert.  Für
     * dieses Beispiel wird angenommen, dass die Präzision fünf und die Skalierung
     * zwei ist.
     */

     if( param1 == NULL )
     {
        *result = NULL;
     }
     else
     {
        CogInt16   i;

        for( i = 0, **result = 0.0; i < 5; i++ )
        {
            **result *= 10;

            if( i & 1 )
            {
                **result += ( param1[ i / 2 ] ) & 0xF;
            }
            else
            {
                **result += ( param1[ i / 2 ] >> 4 ) & 0xF;
            }
        }

        /*
         * Berechnung für die Skalierung.
         */

        **result /= 100;

        if( ( param1[ 2 ] & 0x0F ) == COG_DECIMAL_NEGATIVE ){
            **result = -**result;
        }
     }

     return;

} /* meinePDFunktion */
```

Datums-Datentyp

Datumswerte werden durch die CogDate-Struktur repräsentiert. Die folgenden Funktionen werden als Teil des UDF SDK für die Bearbeitung von Datenwerten geliefert:

```
CogDate * cogConstructDate(
    CogDate * pCogDate,
    Int32     year,
    Int32     month,
    Int32     day
);
CogInt32 cogGetDatePart(
    CogDate * pCogDate,
    UInt32    datePart
);
```

Mögliche Werte für den Parameter *datePart* sind folgende:

- COG_DATE_YEAR
- COG_DATE_MONTH
- COG_DATE_DAY

Im folgenden Beispiel wird eine UDF gezeigt, die zum bereitgestellten Datumswert ein Jahr hinzufügt:

```
void myDateFunction( CogDate * param1, CogDate ** result )
{
    CogInt32 year;
    CogInt32 month;
    CogInt32 day;

    if( param1 == NULL )
    {
        *result = NULL;
    }
    else
    {
        year  = cogGetDatePart( param1, COG_DATE_YEAR );
        month = cogGetDatePart( param1, COG_DATE_MONTH );
        Tag   = cogGetDatePart( param1, COG_DATE_DAY );

        year++;

        cogConstructDate( *result, year, month, day );
    }

    return;

} /* meineDatumsFunktion */
```

Uhrzeit-Datentyp

Uhrzeitwerte werden durch die CogUdfTime-Struktur repräsentiert. Die folgenden Funktionen werden als Teil des UDF SDK für die Bearbeitung von Uhrzeitwerten geliefert:

```
CogTime * cogConstructTime(
    CogTime * pCogTime,
    Int32         hour,
    Int32         minutes,
    Int32         seconds,
    In32          milliseconds
);
CogInt32 cogGetTimePart(
    CogTime * pCogTime,
    UInt32    timePart
);
```

Mögliche Werte für den Parameter *timePart* sind folgende:

- COG_TIME_HOUR
- COG_TIME_MINUTE
- COG_TIME_SECOND
- COG_TIME_MILLISECOND

Im folgenden Beispiel wird eine UDF gezeigt, die einen Zeitwert auf die nächste Minute aufrundet:

```
void meineUhrzeitFunktion( CogTime * param1, CogTime ** result )
{
    CogInt32 hour;
    CogInt32 minute;
    CogInt32 seconds;
    CogInt32 fseconds;

    if( param1 == NULL )
    {
        *result = NULL;
    }
    else
    {
        hour     = cogGetTimePart( param1, COG_TIME_HOUR );
        minute   = cogGetTimePart( param1, COG_TIME_MINUTE );
        seconds  = cogGetTimePart( param1, COG_TIME_SECOND );
        fseconds = cogGetTimePart( param1,
                                   COG_TIME_MILLISECOND );

        /*
         * Auf die nächstgelegene Minute auf- bzw. abrunden.
         */

        if( fseconds >= 500 )
        {
            seconds++;
        }

        if( seconds >= 30 )
        {
            minute++;
        }

        if( minute == 60 )
        {
            minute = 0;
            hour++;
        }

        if( hour == 24 )
        {
            hour = 0;
        }

        fseconds = 0;
        seconds  = 0;

        cogConstructTime( *result,
                          hour,
                          minute,
                          seconds,
                          fseconds );
    }

    return;

} /* meineUhrzeitFunktion */
```

Zeitstempel-Datentyp

Zeitstempelwerte werden durch die CogTimestamp-Struktur repräsentiert.

```
typedef struct {
    CogDate date;
    CogTime time;
} CogTimestamp;
```

Die verschiedenen CogDate- und CogTime-Funktionen können dazu verwendet werden, in einer CogTimestamp-Struktur die Felder zu konstruieren oder die relevanten Teile daraus zu extrahieren.

Intervall-Datentyp

Intervallwerte werden durch die CogInterval-Struktur repräsentiert. Die folgenden Funktionen werden als Teil des UDF SDK für die Bearbeitung von Intervallwerten geliefert:

```
CogInterval * cogConstructInterval(
    CogInterval * pCogInterval,
    Int32         days,
    Int32         hour,
    Int32         minutes,
    Int32         seconds,
    In32          milliseconds
);
CogInt32 cogGetIntervalPart(
    CogInterval * pCogInterval,
    UInt32        intervalPart
);
```

Mögliche Werte für den Parameter intervalPart sind folgende:

- COG_INTERVAL_DAYS
- COG_INTERVAL_HOUR
- COG_INTERVAL_MINUTE
- COG_INTERVAL_SECOND
- COG_INTERVAL_MILLISECOND

Ein Intervall gibt die Anzahl von Tagen und einen Uhrzeitwert an. Ein Intervall gibt kein bestimmtes Datum bzw. keine bestimmte Uhrzeit, sondern ein Zeitintervall an. Ein Intervall kann zum Beispiel die Zeitspanne zwischen zwei bestimmten Zeitstempelwerten sein.

Binär-Datentyp

Binäre Werte werden in C-Sprache mit einem Descriptor repräsentiert:

```
typedef struct
{
    Int32      length;
    CogUInt8 * data;
}
CogBinary;
```

Die Feldlänge repräsentiert die Anzahl von Bytes binärer Daten, auf die durch das Datenfeld verwiesen wird.

Eine Funktion, die einen binären Wert zurückgibt, hat eine vordefinierte maximale Größe, die in der UDF-Definitionsdatei angegeben ist. Eine UDF muss dem Längenfeld die Anzahl von Bytes zuweisen, die gültige Werte enthalten (wenn der Wert nicht NULL ist).

Wichtige Überlegungen zu Datentypen

Wenn eine UDF entworfen wird, müssen Sie die Umgebung berücksichtigen, in der die UDF benutzt werden soll. Sie müssen die folgenden wichtigen Fragen stellen:

- Wenn die UDF eine Zeichenkette zurückgibt, wird welche maximale Größe unterstützt?
- Welchen Wertebereich sollte die UDF unterstützen?
- Sollte die UDF eine breite Auswahl von Datentypen unterstützen (die implizite oder explizite Typenausgaben erfordern?)

Wenn Sie diese Fragen nicht beachten, kann Ihre UDF fehlschlagen bzw. das System destabilisieren oder zwar ausgeführt werden, jedoch ungültige oder falsche Daten wiedergeben.

Zeichen-Datentyp

Alle Zeichenketten-Parameter werden als null-terminierte Zeichenketten an die UDF übergeben. Für die Länge eines Eingabeparameters in eine UDF bestehen keine Längenbegrenzungen. Dies trifft jedoch nicht auf den zurückgegebenen Wert zu. Die maximale Länge (außer dem Null-Schließer) muss in der Datei COGUDF.SQL vordefiniert sein. Cognos-Produkte statten die UDF mit einem Puffer in der definierten Größe (plus ein Byte für den Null-Schließer) aus. Das Ergebnis darf nicht größer sein als die definierte Größe.

Die maximale Länge einer Zeichenkette ist 2^{31} - 1 Bytes.

Beachten Sie außerdem, dass in einem nicht-englischen Gebietsschema die Anzahl der Zeichen in einer Zeichenkette nicht unbedingt mit der Anzahl von Bytes identisch ist. Ein Zeichen kann 1 Byte bis 4 Bytes erfordern. Die in der Datei COGUDF.SQL definierte Größe repräsentiert die maximale Anzahl von Bytes und nicht von Zeichen.

Kleine Ganzzahlen-, Ganzzahlen- und 64-Bit-Ganzzahl-Datentypen

Diese exakten numerischen Datentypen repräsentieren einen ansteigenden Wertebereich. Wenn der Wert, der von einem Parameter oder einem Rückgabewert bearbeitet werden soll, in einen bestimmten Bereich fällt, ist es am besten, den kleinsten passenden Typ zu wählen.

Ganzzahl-Typen eignen sich für das Durchführen arithmetischer Berechnungen und das Angeben exakter Werte. Ihr Wertebereich kann jedoch beschränkt sein. Obwohl 64-Bit-Ganzzahlen 19 Stellen Präzision liefern, werden sie nicht nativ von jeder Hardware unterstützt.

Beachten Sie außerdem, dass in der Datei COGUDF.SQL eine optionale Skalierung für einen Ganzzahl-Parameter definiert werden kann, die die Anzahl der Stellen nach der Dezimalstelle definiert. Dies kann für finanzielle Berechnungen hilfreich sein.

Packed decimal-Datentyp

Wie bereits erwähnt, müssen alle Packed decimal-Werte - sowohl Parameter als auch Rückgabewerte - eine vordefinierte Präzision und Skalierung haben. Dies kann ihre Verwendbarkeit in allgemeinen UDFs einschränken. Eine mögliche Lösung ist, in allen Packed decimal-Parametern und Rückgabewerten die maximale Präzision zu definieren.

Ein weiteres Problem mit Packed decimal-Werten ist, dass in C keine native Unterstützung für sie vorliegt. Solche Werte müssen zuerst in eine andere Datenrepräsentation konvertiert werden, bevor sie bearbeitet werden können.

Gleit- und doppelte Präzisions-Datentypen

Gleitwerte und numerische Werte mit doppelter Präzision unterstützen einen breiten Bereich numerischer Datentypen, liefern jedoch nur 7 und 15 Stellen von Präzision (entsprechend für Gleitwerte und doppelte Präzisionswerte). Wenn Sie feststellen, dass keine höhere Präzision erforderlich ist, sind diese ungefähren numerischen Datentypen vielleicht die beste Option für numerische Parameter und Rückgabewerte.

Beachten Sie, dass Gleitkomma-Berechnungen längere Bearbeitungszeiten beanspruchen als gleichwertige Vorgänge mit exakten numerischen Typen, selbst wenn der betreffende Computer über einen Gleitkomma-Prozessor verfügt.

Binär-Datentyp

Beim Handhaben von binären Daten wird angenommen, dass die UDF alle binärische Daten, die als Parameter geliefert werden, kennt, und dass die Anwendungen Kenntnis vom Inhalt der binären Daten hat, die von einer UDF zurückgegeben werden.

Implizite Ausgaben

Typenausgaben erlauben das Aufrufen einer Funktion mit Werten, die nicht genau mit den Datentypen der Funktionsparameter übereinstimmen. Es sind verschiedene implizite Typenausgaben verfügbar. Wenn eine Ausgabe nicht ausgeführt werden kann, tritt bei der Vorbereitung einer SQL-Abfrage ein Fehler auf.

Obwohl die Konvertierung von exakten numerischen Datentypen zu ungefähren Datentypen unterstützt wird, ist es möglich, dass bei der Konvertierung wichtige Stellen verloren gehen. In Konvertierungen von und zu Packed decimal-Werten kann, abhängig von der Präzision der Packed decimal-Werte, ein Überlauf auftreten.

In der folgenden Tabelle werden die Datentypen für eine einfache Darstellung in der zweiten Tabelle kodifiziert. Die letzte der drei Tabellen enthält eine Legende für die zweite Tabelle.

Datentyp	Code	Datentyp	Code
CogUInt16	UI16	CogChar	CH
CogInt16	I16	CogDate	DT
CogUInt32	UI32	CogTime	TM
CogInt32	I32	CogTimestamp	TS
CogInt64	I64	CogInterval	IN
CogFloat32	F32	CogBinary	BN
CogFloat64	F64	Packed Decimal	PD

Bis	I16	UI16	I32	UI32	I64	PD	F32	F64	CH	DT	TM	TS	IN	BN
Von														
I16	+	+	+	+	+	+	+	+	+					
UI16	●	+	+	+	+	+	+	+	+					
I32	●	●	+	+	+	+	+	+	+					
UI32	●	●	●	+	+	+	+	+	+					
I64	●	●	●	●	+	+	+	+	+					
PD	●	●	●	●	●	+	+	+	+					
F32	●	●	●	●	●	●	+	+	+					
F64	●	●	●	●	●	●	●	+	+					
CH	+	+	+	+	+	+	+	+	+	+	+	+	+	
DT									+	+		+		
TM									+		+	+		
TS									+	+	+	+		
IN									+				+	
BN														+

Symbol	Legende
+	Die Ausgabe wird unterstützt.
Kein Eintrag	Die Ausgabe wird nicht unterstützt.
●	Obwohl die Ausgabe unterstützt wird, kann ein Überlauf auftreten.

Anhang

In diesem Anhang finden Sie Folgendes:

- eine Tabelle mit Cognos-Identifizierungen für unterstützte Datenbanksysteme und die Datenbank-spezifischen Dateinamen .INI und .SQL.
- Tipps für die Fehlerbehebung
- Informationen über das Syntax-Prüfungsdienstprogramm cogudf.exe
- eine Liste der Dateien, aus denen der Software-Entwickler-Kit für die UDFs besteht

Datenbank-Identifizierungen

Datenbank	Identifi-zierung	.INI-Datei	.SQL-Datei
DB2	D2	d2funct.ini	cogudfd2.sql
Informix	IF	iffunct.ini	cogudfif.sql
Microsoft SQL Server	MS	msfunct.ini	cogudfms.sql
ODBC	OD	odfunct.ini	cogudfod.sql
Oracle	OR	orfunct.ini	cogudfor.sql
Sybase CT-Library	CT	ctfunct.ini	cogudfct.sql

Fehlerbehebung

In diesem Abschnitt finden Sie Korrekturschritte, die Sie ausführen können, wenn Sie auf die hier aufgeführten Fehlermeldungen oder Probleme treffen.

Fehler

```
Die Funktion <Name der Funktion> ist nicht als externe, Datenbank- oder integrierte
Funktion verfügbar.
```

Korrekturschritt

Stellen Sie sicher, dass sich die .SQL-Datei in einem der folgenden Pfade befindet:

- im aktuellen Arbeitsverzeichnis
- in Windows: im Ordner, der im Abschnitt `ServicesL` der Datei Cognos.ini oder im Ordner /bin angegeben ist
- in UNIX: im Verzeichnis, das entweder durch die Umgebungsvariable `COGUDFSQL` oder `DMDBINI` festgelegt wird

Wenn sich die Datei im richtigen Pfad befindet, sollten Sie sicherstellen, dass die Funktionsdefinition keine Syntaxfehler aufweist. Überprüfen Sie die Syntax mit dem Dienstprogramm COGUDF.

Fehler

```
Die externe Funktion <Name der Funktion> erfordert 2 Parameter, es sind jedoch 3
angegeben.
```

Korrekturschritt

Für die Funktion wurde eine falsche Anzahl von Parametern angegeben. Stellen Sie sicher, dass

- Ihre Funktionsdefinition korrekt ist
- die Funktion richtig in der SQL-Anweisung aufgerufen wird
- in der entsprechenden .SQL-Datei nur eine einzige Definition der Funktion vorliegt - es wird nur die erste Definition der Funktion benutzt.

Fehler

```
Auf die externe Funktion <Name der Funktion> kann nicht zugegriffen werden.
```

Korrekturschritt

Die Funktion kann nicht gefunden werden. Mögliche Erklärungen:

- Der in der .SQL-Datei angegebene Name der Bibliothek stimmt nicht mit dem Namen der gemeinsam benutzten Bibliothek überein, die die externe Funktion enthält.
- Der Name der Funktion im C-Modul und der Name der C-Funktion, der in der .SQL-Datei definiert ist, sind verschieden
- in UNIX wurde für die Plattform-spezifische Bibliothekspfad-Umgebungsvariable kein Wert zugewiesen
- in UNIX befindet sich die gemeinsam benutzte Bibliothek nicht in einem der Verzeichnisse, die in der Bibliothekspfad-Umgebungsvariablen angegeben werden
- in Windows befindet sich die gemeinsam benutzte Bibliothek nicht in einem der Verzeichnisse, die in der PFAD-Umgebungsvariablen angegeben werden
- die Funktion wurde von der gemeinsam benutzten Bibliothek oder DLL nicht importiert
- der in der Exportliste angegebene Name der Funktion stimmt nicht mit dem Namen der Funktion im C-Modul überein

Unlesbare Daten oder Unerwartetes Beenden

Wenn Sie in Windows in einer Funktion unlesbare Daten erhalten oder die Funktion die Anwendung abbricht, sollten Sie sicherstellen, dass Sie Ihre DLL mit 8-Byte-Ausrichtung kompiliert haben.

Datenbankfehler

Wenn Sie beim Aufrufen einer Datenbank-UDF in einer SQL-Anweisung eine Datenbank-Fehlermeldung erhalten, stellen Sie sicher, dass

- der Name der UDF in der entsprechenden .SQL-Datei richtig geschrieben wurde
- die Datenbank-UDF über die erforderliche Qualifikation verfügt

  Wenn die UDF zum Beispiel in einem anderen als dem Standard-Schema installiert wurde, erfordert die UDF sehr wahrscheinlich eine Schema-Qualifikation in der Funktionsdefinition der .SQL-Datei. Im Allgemeinen ist es am besten, Datenbank-UDF-Funktionsnamen vollständig zu qualifizieren, um Zweideutigkeiten auszuschließen und Datenbankfehler zu vermeiden, die entstehen können, wenn die Datenbank die UDF nicht finden kann.

Das Syntaxprüfungs-Dienstprogramm COGUDF

Mit Hilfe des Windows-Dienstprogrammes COGUDF.EXE können Sie Ihre .SQL-Dateien auf Syntaxfehler überprüfen.

Wenn Sie UDFs für eine UNIX-Plattform entwickeln, können Sie Ihre .SQL-Dateien mit Hilfe von COGUDF in einer Windows-Umgebung debuggen, bevor Sie die UDFs in Ihrem UNIX-System installieren.

Um COGUDF zu benutzen, führen Sie die Datei einfach aus. Sie akzeptiert keine Befehlszeilen-Optionen. Das Dienstprogramm COGUDF durchsucht zuerst den aktuellen Ordner und danach die Ordner, die Sie im Abschnitt `ServicesL` in der Datei Cognos.ini angegeben haben.

Das Dienstprogramm COGUDF meldet sämtliche Fehler, die es in den .SQL-Dateien findet. In Windows-Umgebungen werden die Ausgabedaten in die Protokolldatei geschrieben, die durch den Eintrag `UDF Log File` im Abschnitt `[ServicesL]` in der Datei Cer*n*.ini festgelegt ist:

```
[ServicesL]
UDF Log File=d:\temp\udf.log
```

Wenn dieser Eintrag nicht existiert oder keinen Wert hat, werden keine Informationen in die Protokolldatei geschrieben. Wenn der Eintrag existiert, wird die Datei entweder erstellt oder an eine vorhandene Datei, die eventuell gefunden wird, angefügt.

Ihr Cognos-Produkt schreibt ebenfalls in die Protokolldatei, wenn eine UDF in einem SQL-Ausdruck gefunden wird.

In einer UNIX-Umgebung verweist die Umgebungsvariable `COGUDFLOG` auf die Protokolldatei. Wenn Sie diese Variable angeben, wird nur in die Protokolldatei geschrieben, wenn Request Server eine UDF antrifft.

Dateien im UDF Software-Entwickler-Kit

Die SDK-Dateien und Beispiele wurden mit Hilfe von Microsoft Developer Studio 97 mit Visual C++ 4.2 erstellt.

Datei	Beschreibung
cogudfty.h	Für UNIX und Windows. Eine Header-Datei, die die von den UDFs unterstützten Datentypen definiert. Enthalten in cogudf.h.
cogudf.h	Für UNIX und Windows. Eine Header-Datei, die alle erforderlichen Makros, Typen und Funktionen für die Funktionalität der UDFs definiert.
udfLib.a	Für UNIX. Eine Bibliothek, die die in cogudf.h definierten Funktionen enthält. Sie muss mit allen DLLs verbunden sein, die UDFs enthalten, wenn eine der Cognos-Unterstützungsfunktionen für UDFs benutzt wird.
udflib.lib	Für Windows. Eine Bibliothek, die die in cogudf.h definierten Funktionen enthält. Sie muss mit allen DLLs verbunden sein, die UDFs enthalten, wenn eine der Cognos-Unterstützungsfunktionen für UDFs benutzt wird.
cogudf.exe	Für Windows. Ein Dienstprogramm, das alle verfügbaren .SQL-Dateien, die Datenbanken und externe Funktionsdefinitionen enthalten, auf Syntax- oder Semantik-Fehler überprüft.
cogudf.sql	Für UNIX und Windows. Die .SQL-Datei, die die UDF-Funktionen enthält.

Datei	Beschreibung
benutzerdefinierte funktion db - readme.txt	Readme-Datei für die Beispieldatenbank-UDF. Diese Datei erklärt die Benutzung folgender Dateien: benutzerdefinierte funktion oracle.txt orfunct aktualisierung.txt benutzerdefinierte funktion informix.txt Cogudfxx.sql
encrypt - readme.txt	Readme-Datei für die externen Verschlüsselungs-Beispiel-UDFs: Diese Datei erklärt die Benutzung folgender Dateien: UDFs. impfunct aktualisierung.txt Encrypt.c Encrypt.def Encrypt.dll cogudf encrypt aktualisierung.sql impfunct encrypt aktualisierung.txt
html - readme.TXT	Dateien für die externen Beispiel-HTML-UDFs. Pubhtml.c Pubhtml.c Pubhtml.def Pubhtml.dll cogudf html aktualisierung.sql impfunct html aktualisierung.txt
Udf.mdp Udf.mak Udf.ncb	Die Microsoft Developer Studio-Projektdateien für die Beispiele.

Index