Die obengenannte Unterteilung der Systemprozeduren ist weder vollständig noch eindeutig. Viele Systemprozeduren können z.B. sowohl der ersten als auch der zweiten Gruppe zugeordnet werden.

Weil Systemprozeduren für unterschiedliche Aufgaben angewendet werden können, werden wir sie in den entsprechenden Kapiteln beschreiben. Diese Beschreibung wird nicht alle vom SQL Server angebotenen Prozeduren enthalten. Für eine vollständige Auflistung der Systemprozeduren verweisen wir den Leser auf die entsprechenden Handbücher des SQL Servers.

Jeder Benutzer kann auch eigene Systemprozeduren mit der CREATE PROCEDURE-Anweisung erstellen. Die Voraussetzung für solche Prozeduren ist, daß sie mit dem Präfix »sp_« beginnen und der *master*-Datenbank angehören. Solche Prozeduren können unabhängig von der augenblicklich aktuellen Datenbank aufgerufen werden.

8.3 Benutzerdefinierte Funktionen

Programmiersprachen unterstützen generell zwei Routine-Typen:

- Prozeduren
- Funktionen

Prozeduren enthalten keine, einen oder mehrere Parameter, liefern aber keine Rückgabewerte. Im Unterschied zu Prozeduren liefern Funktionen einen oder mehrere Rückgabewerte. (Wie aus dem folgenden Abschnitt ersichtlich wird, können benutzerdefinierte Funktionen beim SQL Server 2000 nur einen einzigen Rückgabewert liefern.)

8.3.1 Erstellung und Ausführung benutzerdefinierter Funktionen

Benutzerdefinierte Funktionen werden mit Hilfe der CREATE FUNCTION-Anweisung erstellt. Diese Anweisung hat folgende Syntax:

```
CREATE FUNCTION [benutzer.]funktion_name
     [[(]@param_1 typ_1 [=default_1]
     [{,@param_2 typ_2  [=default_2]]}...[)]]
     RETURNS {skalar_wert | {[@variable] TABLE}}
     [WITH {ENCRYPTION | SCHEMABINDING}
     AS {block | RETURN(select_anw)}
```

benutzer ist der Name des Eigentümers der Funktion, während **funktion_name** den Namen der benutzerdefinerten Funktion kennzeichnet.

@param_1, **@param_2**,... sind Parameternamen, während **typ_1**, **typ_2**,... ihren Datentypen spezifizieren. (Parameter sind Werte, die von dem Aufrufprogramm an die benutzerdefinierte Funktion geschickt werden und innerhalb der Funktion verwendet werden.) **default_1**, **default_2**,... kennzeichnen optionale Standardwerte der entsprechenden Parameter.

Die Angabe RETURNS definiert den Datentyp des Rückgabewertes. Dieser kann entweder einen Standarddatentyp haben (**skalar-wert**) oder den neuen Datentyp namens TABLE. (Die einzigen Standarddatentypen, die nicht benutzt werden können, sind TIMESTAMP, TEXT, NTEXT und IMAGE.)

Eine benutzerdefinierte Funktion beim SQL Server 2000 kann entweder einen skalaren Wert oder eine Tabelle als Rückgabewert liefern. Eine Funktion liefert einen skalaren Wert, falls in der RETURNS-Klausel ein Standarddatentyp spezifiziert wird. Jede Funktion mit der Angabe TABLE in dieser Klausel gibt eine ganze Tabelle zurück.

Abhängig davon, wie der Funktionskörper definiert ist, können Funktionen mit den Tabellen als Rückgabewerten:

- *inline* oder
- mit mehreren Anweisungen

sein. Falls die RETURNS-Klausel die Option TABLE ohne irgendwelche zusätzlichen Angaben verwendet, handelt es sich um eine *inline*-Funktion. Eine solche Funktion liefert das Ergebnis als eine Variable vom Typ TABLE. Eine Funktion mit mehreren Anweisungen in der RETURNS-Klausel enthält einen Namen, an den das Schlüsselwort TABLE angefügt ist. (Der Name ist eine interne SQL Server-Variable vom Typ TABLE.) Die Variable kann verwendet werden, um Reihen einzufügen und sie anschließend als den Rückgabewert der Funktion zurückzugeben.

Die Angabe WITH ENCRYPTION verursacht die Verschlüsselung der Anweisungen innerhalb der benutzerdefinierten Funktion. Diese Angabe ist empfehlenswert, wenn die Implementierung der Funktion nicht allgemein bekannt werden darf.

Die alternative WITH SCHEMABINDING-Klausel bindet die benutzerdefinierte Funktion zum Datenbankobjekt, das sie referenziert. (Das Ergebnis dieser Referenzierung ist, daß jeder Versuch, das Datenbankobjekt zu ändern, fehlschlägt.) Die von einer Funktion referenzierten Datenbankobjekte müssen gewisse Kriterien erfüllen, damit die SCHEMABINDING-Klausel mit ihnen verwendet werden kann. Diese sind:

- Alle Views und benutzerdefinierte Funktionen, die durch die angegebene Funktion referenziert sind, müssen selbst gebunden werden (d.h. mit der WITH SCHEMABINDING definiert werden).
- Alle referenzierten Datenbankobjekte (Tabellen, Views und benutzerdefinierte Funktionen) müssen zu derselben Datenbank wie die angegebene Funktion gehören.

block kennzeichnet einen BEGIN-Block, der den Implementierungsteil der Funktion enthält. Die letzte Anweisung im **block** muß eine RETURN-Anweisung sein. (Der Wert des Argumenten der RETURN-Anewisung entspricht dem Rückgabewert der Funktion.) Im BEGIN-Block können nur folgende Anweisungen angegeben werden:

- Zuweisungsanweisungen (z.B. die SET-Anweisung)
- Anweisungen WHILE und IF
- die DECLARE -Anweisung
- die SELECT-Anweisung, mit der den Spaltenwerten (in der Projektion), entsprechende Variablen zugewiesen werden
- die INSERT-, UPDATE-, und DELETE-Anweisung, die die Werte der Variablen vom Typ TABLE ändern.

Gleich nach der Erstellung einer Datenbank haben nur die Mitglieder der Datenbankrollen **db_owner** und **db_ddladmin** das Recht, die CREATE FUNCTION-Anweisung auszuführen. Anschließend können sie auch den anderen Benutzern mit Hilfe der GRANT CREATE FUNCTION-Anweisung die Rechte zur Erstellung von Funktionen vergeben (siehe Kapitel 12).

Das folgende Beispiel zeigt die Erstellung einer benutzerdefinierten Funktion.

Beispiel 8.8

Diese benutzerdefinierte Funktion berechnet zusätzliche Kosten, die entstehen, falls Projektmittel erhöht werden.

```
CREATE FUNCTION compute_costs (@prozent INT =10)
      RETURNS DECIMAL(16,2)
   BEGIN
    DECLARE @kosten DEC (14,2), @sum_mittel dec(16,2)
   SELECT @sum_mittel = SUM (mittel)FROM projekt
   SET @kosten = @sum_mittel * @prozent/100
   RETURN @kosten
  END
```

Die Funktion **compute_costs** berechnet die Kosten, die entstehen, falls alle Projektmittel erhöht werden. Im BEGIN-Block werden zuerst zwei lokale Variablen definiert: **kosten** und **sum_mittel**. Der zweiten Variablen wird die Summe aller Mittel mit Hilfe der SELECT-Anweisung zugewiesen. Danach berechnet die Funktion alle zusätzlichen Kosten und gibt den Wert mit Hilfe der RETURN-Anweisung zurück.

Jede benutzerdefinierte Funktion kann innerhalb einer DML-Anweisung (SELECT, INSERT, UPDATE und DELETE) aufgerufen werden. Um eine benutzerdefinierte Funktion aufzurufen, muß der Funktionsname zusammen mit einem Klammerpaar angegeben werden. Innerhalb des Klammerpaares können ein oder mehrere Argumente spezifiziert werden. (Argumente sind Werte, die den definierten Funktionsparametern zugewiesen werden. Jedes Argument wird dem entsprechenden Parameter zugewiesen.)

Das folgende Beispiel zeigt, wie die Funktion **compute_costs** (Beispiel 8.8) innerhalb einer SELECT-Anweisung verwendet werden kann.

Beispiel 8.9

```
select pr_nr, pr_name
     from projekt
     WHERE mittel < dbo.compute_costs(25)
```

Das Ergebnis ist:

```
pr_nr pr_name
----- -------------------------
p2    Gemini
```

Das Ergebnis des Beispiels 8.9 gibt alle Namen und die Nummern aller Projekte, wo die Mittel kleiner sind als zusätzliche Kosten der Mittelerhöhung aller Projekte für den angegebenen Prozentsatz.

Jeder Name einer benutzerdefinierten Funktion, der in einer Transact-SQL-Anweisung angegeben wird, muß in der Form:

```
benutzer_name.funktions_name
```

spezifiziert werden.

Das folgende Beispiel zeigt eine benutzerdefinierte Funktion, deren Rückgabewert vom Typ TABLE ist.

Beispiel 8.10

```
create function mitarbeiter_im_projekt (@pr_nummer char(4))
   returns table
   as return (select m_vorname, m_name
                     from arbeiten, mitarbeiter
                     where mitarbeiter.m_nr = arbeiten.m_nr
                     and pr_nr = @pr_nummer)
```

Die Funktion **mitarbeiter_im_projekt** wird benutzt, um die Namen aller Mitarbeiter eines Projektes zu ermitteln. Der Eingabeparameter **@pr_nummer** spezifiziert eine Projektnummer. Weil die Funktion im allgemeinen mehrere Reihen als Ergebnis liefert, muß die RETURNS-Klausel die TABLE-Angabe enthalten.

Der BEGIN-Block erscheint nicht im Beispiel 8.10, weil die RETURN-Klausel eine SELECT-Anweisung enthält.

Beispiel 8.11 gibt die Namen und Projektnummer aller Mitarbeiter aus, die im Projekt p3 arbeiten.

Beispiel 8.11

```
select *
   from mitarbeiter_im_projekt('p3')
```

Das Ergebnis ist:

```
m_vorname            m_name
-------------------- --------------------
Petra                Huber
Rainer               Meier
Brigitte             Kaufmann
```

SQL Server 2000 unterstützt zusätzlich die ALTER FUNCTION-Anweisung, die die Struktur einer benutzerdefinierten Funktion ändert. Diese Anweisung wird gewöhnlich benutzt, um die WITH SCHEMABINDING-Klausel zu entfernen. Die ALTER FUNCTION-Anweisung hat dieselben Optionen wie die CREATE FUNCTION-Anweisung.

8.4 Text/image-Datentypen

Der SQL Server unterstützt die Datentypen **text, ntext** und **image**, die bis zu 2 GB (2,147,483,637 Bytes) enthalten können. Wie wir schon in Kapitel 2 erläutert haben, beschreiben die Datentypen **text** und **ntext** Werte, die ausschließlich abdruckbare Zeichen enthalten können, während der Datentyp **image** Bitketten darstellt.

Um die Eigenschaften dieser Datentypen zu erläutern, werden wir die Tabelle **mitarbeiter** um eine weitere Spalte – **pers_akte** - erweitern. (Die Spalte **pers_akte** enthält alle Personaldaten eines Mitarbeiters.) Die so entstandene Tabelle wird **mit_akte** genannt.

Spalten vom Typ **text, ntext** und **image** werden genauso wie alle anderen Spalten definiert. Beispiel 8.12 zeigt die Definition der Tabelle **mit_akte**.

Beispiel 8.12

```
create table mit_akte
     (m_nr int not null,
     m_name char(20) not null,
     m_vorname char(20) not null,
     abt_nr char(4) null,
     pers_akte text null)
```

Daten vom Typ **text, ntext** und **image** werden getrennt von allen anderen Daten der Tabelle in einer verketteten Liste von physikalischen Seiten gespeichert. In der Tabelle wird für jedes derartige Objekt lediglich ein Zeiger, der auf den Anfang der verketteten Liste zeigt, gespeichert. Für eine Tabelle, die mehrere Spalten vom Typ **text, ntext** bzw. **image** enthält, werden alle derartigen Daten in derselben verketteten Liste gespeichert.