Kapitel 12

PHP und SQL Server

In diesem Kapitel:

Arno Hollosi: PHP programmieren unter Windows, Microsoft Press 2010 (ISBN 978-3-86645-447-7)

In diesem Kapitel wird gezeigt, wie Sie auf Daten in SQL Server von PHP aus zugreifen können. Neben dem prinzipiellen Ablauf und Basisfunktionen der SQL Server-PHP-Erweiterung, wird beschrieben, wie sich die Wahl des Benutzers, der sich zur Datenbank verbindet, auf das Verbindungspooling auswirkt, wie Sie mit parametrisierten Anweisungen SQL Injection vorbeugen können und wie die Konvertierung verschiedener Datentypen zwischen PHP und SQL Server funktioniert.

Vorgehen und Ablauf

Die Ansteuerung von SQL Server von PHP aus erfolgt immer nach demselben Schema: Datenbankverbindung öffnen, T-SQL-Anweisung senden, Ergebnisse auslesen und am Ende Datenbankverbindung wieder schließen. Dieser Ablauf und die dafür notwendigen PHP-Funktionen werden im Folgenden anhand eines Beispielprogramms beschrieben.

Vorbereitungen

Damit Sie das Beispielprogramm erfolgreich ausführen können, müssen die SQL Server-PHP-Erweiterung und die Beispieldatenbank *AdventureWorksLT2008* installiert sein (siehe Kapitel 9).

Zudem benötigen Sie eine gültige Anmeldung für die Beispieldatenbank. Im Beispiel wird die Anmeldung über den aktuellen Windows-Benutzer der PHP-Anwendung durchgeführt. Die verschiedenen Authentifizierungsmethoden werden im Abschnitt »Authentifizierung«, ab Seite 272, beschrieben. Wie Sie Benutzer und Anmeldungen in SQL Server anlegen und berechtigen, wird in Kapitel 14 beschrieben.

Wenn sich die Datenbank nicht am selben Computer befindet, auf dem das PHP-Skript ausgeführt wird, müssen Sie zudem sicherstellen, dass der Remotezugriff auf SQL Server eingerichtet ist und funktioniert (siehe Kapitel 9).

Das Beispielprogramm

Das Beispielprogramm *product_list.php* ist in Listing 12.1 zu sehen. Es enthält alle wesentlichen Schritte der Verwendung von SQL Server von PHP aus:

- Öffnen der Datenbankverbindung
- Senden eines T-SQL-Befehls
- Auslesen der Antwort
- Schließen der Datenbankverbindung

Im Abschnitt »Die Schritte im Einzelnen« ab Seite 268 wird die Funktionsweise der einzelnen Programmschritte genauer beschrieben. Im Abschnitt »Unterstützendes Skript« ab Seite 271 finden Sie das hier verwendete und inkludierte PHP-Skript *utils.php*.

Arno Hollosi: PHP programmieren unter Windows, Microsoft Press 2010 (ISBN 978-3-86645-447-7)

```
<!DOCTYPE html>
<html>
<head>
    <title>Adventure Works : Produkte</title>
    <style type="text/css">
        th { font-size: 110%; border-bottom: 2px solid black; }
        td { padding: 3px; border-bottom: 1px solid #aaa }
    </style>
</head>
<body>
<h1>Adventure Works : Produkte</h1>
<?php
require './utils.php';
// Verbinden über Windows-Authentifizierung
$server = '(local)';
$connectionInfo = array('Database' => 'AdventureWorksLT2008', 'CharacterSet' => 'UTF-8');
$db = sqlsrv_connect($server, $connectionInfo);
if ($db === false) {
    exitWithSQLError('Datenbankverbindung fehlgeschlagen');
}
// Produkte mit Namen, Listenpreis und Kategorie auswählen
$query = "SELECT p.ProductID, p.Name AS ProductName, p.ListPrice, pc.Name as CategoryName
          FROM SalesLT.Product AS p
          JOIN SalesLT.ProductCategory AS pc ON p.ProductCategoryID = pc.ProductCategoryID
          ORDER BY p.Name";
// Abfrage ausführen
$qresult = sqlsrv_query($db, $query);
if ($qresult === false) {
    exitWithSQLError('Abfrage der Produktdaten fehlgeschlagen.');
}
echo '<tr><th>ID</th><th>Produkt</th><th>Kategorie</th><th>Listenpreis</th></tr>';
// Einzelne Zeilen des Ergebnisses auslesen
while ($row = sqlsrv_fetch_array($qresult)) {
    echo '<tr><td>', htmlspecialchars($row['ProductID']),
         '</td><td>', htmlspecialchars($row['ProductName']),
         '</td><td>', htmlspecialchars($row['CategoryName']),
         '</td><td>', htmlspecialchars($row['ListPrice']),
         "</td></tr>\n";
}
// null == keine weiteren Zeilen mehr, false == Fehler
if ($row === false) {
    exitWithSQLError('Abfruf von Produktdateneintrag fehlgeschlagen.');
}
// Ergebnisliste freigeben und Verbindung schließen
sqlsrv_free_stmt($qresult);
sqlsrv_close($db);
?>
</body>
</html>
```

Listing 12.1 *product_list.php* – Auflisten der Produkte von Adventure Works

Die Ausgabe von *product_list.php* sehen Sie in Abbildung 12.1. Die Produkte, ihre Kennnummern, Namen, Kategorien und Listenpreise, werden als alphabetisch sortierte Liste ausgegeben.

HINWEIS Wenn Sie eine Fehlermeldung von SQL Server erhalten, überprüfen Sie, ob Sie die Voraussetzungen für die PHP-SQL Server-Anbindung geschaffen haben. Insbesondere fehlende Berechtigungen sind häufig Ursache von Fehlern.

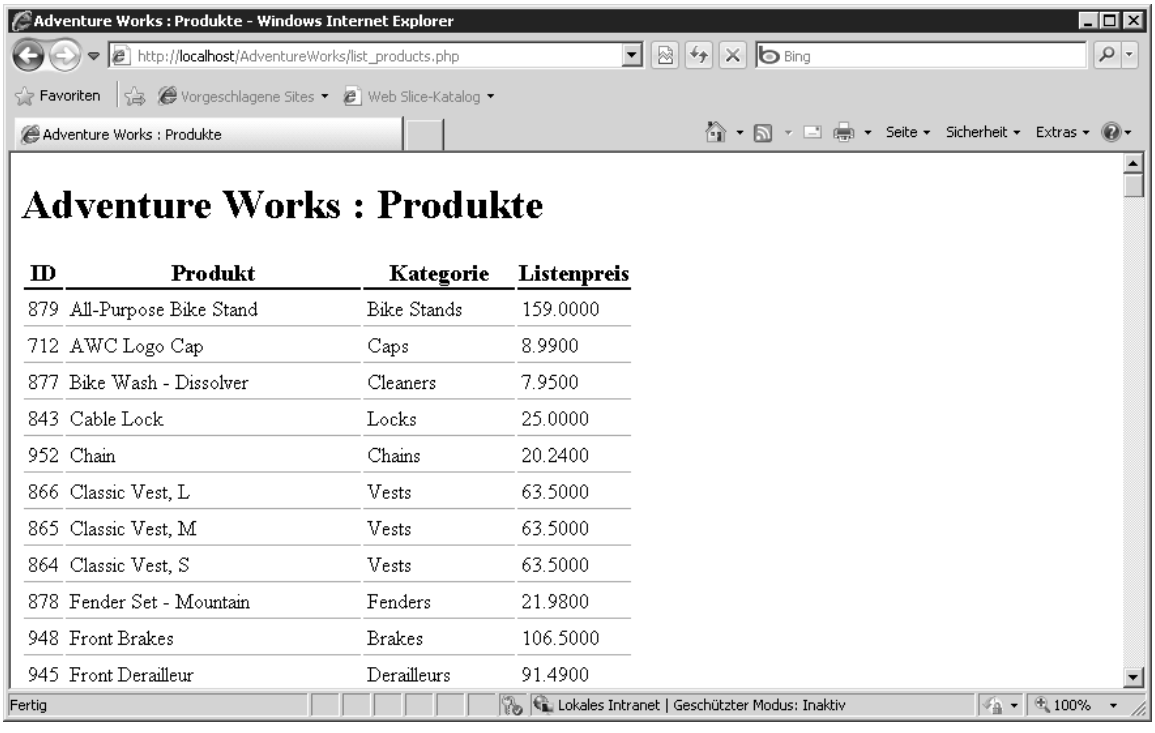

Abbildung 12.1 Ausgabe der Produkte von Adventure Works mit *product_list.php*

Die Schritte im Einzelnen

Im Folgenden werden die einzelnen Schritte des Beispielprogramms aus Listing 12.1 näher beschrieben. Jedes PHP-Programm, das Daten von SQL Server liest oder schreibt, durchläuft dabei prinzipiell die gleichen Schritte, die in Abbildung 12.2 zu sehen sind. Als Erstes wird die Datenbankverbindung geöffnet, dann ein oder mehrere Befehle an SQL Server über diese Verbindung gesandt und das Ergebnis ausgewertet und schließlich freigegeben. Zum Schluss wird die Datenbankverbindung wieder geschlossen.

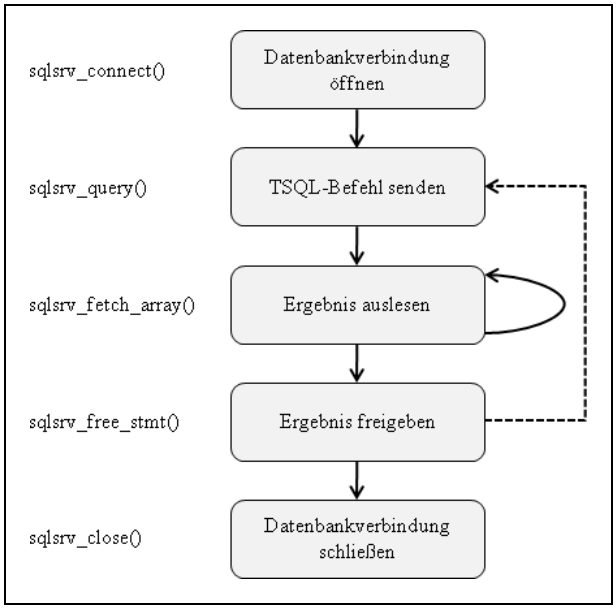

Abbildung 12.2 Schritte eines typischen Ablaufs einer Datenbankabfrage

Öffnen der Datenbankverbindung

Der erste Schritt ist, eine Verbindung zur SQL Server-Datenbank aufzubauen. Eine Verbindung wird mit der Funktion *sqlsrv_connect()* hergestellt. Die Funktion hat zwei Parameter: Den Namen des Servers, zu dem verbunden werden soll, und Verbindungsparameter, vor allem die auszuwählende Datenbank für den Datenbankkontext der Verbindung.

Im Beispielprogramm aus Listing 12.1 wird eine Verbindung mit dem lokal laufenden SQL Server hergestellt und die Datenbank *AdventureWorksLT2008* ausgewählt:

```
// Verbinden über Windows-Authentifizierung
$server = '(local)';
$connectionInfo = array('Database' => 'AdventureWorksLT2008', 'CharacterSet' => 'UTF-8');
$db = sqlsrv_connect($server, $connectionInfo);
```

Wenn, wie in diesem Fall, kein Benutzername und Passwort in den Verbindungsparametern übergeben wird, authentifiziert sich PHP mit dem aktuellen Windows-Benutzer des PHP-Anwendungspools (beispielsweise *IIS AppPool\DefaultAppPool*). Ein Vorteil dieser Variante ist, dass keine Passwörter in den PHP-Skripts gespeichert werden müssen.

Absenden der Abfrage

Im nächsten Schritt wird die Abfrage an die Datenbank geschickt. Als Funktion wird *sqlsrv_query()* verwendet, die im verwendeten Fall zwei Parameter hat: die zu verwendende Datenbankverbindung (*$db*) und den T-SQL-Befehl (*$query*), der an die Datenbank geschickt wird:

```
// Produkte mit Namen, Listenpreis und Kategorie auswählen
$query = "SELECT p.ProductID, p.Name AS ProductName, p.ListPrice, pc.Name as CategoryName
          FROM SalesLT.Product AS p
          JOIN SalesLT.ProductCategory AS pc ON p.ProductCategoryID = pc.ProductCategoryID
          ORDER BY p.Name";
// Abfrage ausführen
$qresult = sqlsrv_query($db, $query);
```

Die *SELECT*-Abfrage holt sich die auszugebende Information über einen *JOIN* aus den Tabellen *SalesLT.Product* und *SalesLT.ProductCategory*.

Als Ergebnis des *sqlsrv_query()*-Aufrufs wird eine Anweisungsstruktur (*T-SQL statement*) retourniert (gespeichert in *$qresult*). Diese kann in Folge verwendet werden, um die Ergebnisse der *SELECT*-Abfrage auszulesen.

Auslesen des Ergebnisses

Nachdem der T-SQL-Befehl an die Datenbank gesendet worden ist, können über die retournierte Anweisungsstruktur (*$qresult*) mit *sqlsrv_fetch_array()* die Zeilen des *SELECT*-Resultats ausgelesen werden. *sqlsrv_fetch_array()* erzeugt ein assoziatives Array, bei dem die Namen den Spaltennamen der *SELECT*-Anweisung entsprechen:

```
// Einzelne Zeilen des Ergebnisses auslesen
while ($row = sqlsrv_fetch_array($qresult)) {
    echo '<tr><td>', htmlspecialchars($row['ProductID']),
         '</td><td>', htmlspecialchars($row['ProductName']),
         '</td><td>', htmlspecialchars($row['CategoryName']),
         '</td><td>', htmlspecialchars($row['ListPrice']),
         "</td></tr>\n";
}
```

Wenn keine weiteren Zeilen mehr zur Verfügung stehen, liefert *sqlsrv_fetch_array()* als Ergebnis *null* zurück und die *while*-Schleife wird beendet.

ACHTUNG In einem Fehlerfall liefert *sqlsrv_fetch_array()* den Boolean-Wert *false* zurück. Um also zwischen Fehler und Ende der Ergebnisliste unterscheiden zu können, ist mit *$row === false* auf Identität mit *false* zu prüfen. Andere Tests, insbesondere *!$row* oder *empty($row)*, erlauben keine Unterscheidung der beiden Fälle.

Schließen der Datenbankverbindung

Nachdem die Daten ausgelesen worden sind, sollte die zur Abfrage zugehörige Anweisungsstruktur freigegeben werden und die Datenbankverbindung geschlossen werden. Bei SQL Server werden für die Freigabe der Anweisungsstruktur *sqlsrv_free_stmt()* und für das Schließen der Datenbankverbindung *sqlsrv_close()* verwendet:

```
// Ergebnisliste freigeben und Verbindung schließen
sqlsrv_free_stmt($qresult);
sqlsrv_close($db);
```

Unterstützendes Skript

Das Beispielprogramm bindet noch das Hilfsskript *utils.php* aus Listing 12.2 ein, welches die Hilfsfunktion *exitWithSQLError()* zur Ausgabe von Fehlermeldungen enthält. Die Funktion wird aufgerufen, wenn bei der Kommunikation mit SQL Server Fehler auftreten.

```
<?php
/**
 * Programm beenden, Fehlermeldung von SQL Server ausgeben
 * @param string $txt Beschreibung des Fehlerkontexts
 */
function exitWithSQLError($txt)
{
    $errors = sqlsrv_errors();
    echo '<h1>Datenbankfehler</h1>', "<p>Fehler: $txt</p>";
    foreach ($errors as $error) {
        echo '<p><b>SQL-Status:</b> ', htmlspecialchars($error['SQLSTATE']), '<br />',
             '<b>Code:</b> ', htmlspecialchars($error['code']), '<br />',
             '<b>Meldung</b>: ',
             // Fehlermeldungen werden im ISO-8859-1-Format übertragen
             htmlspecialchars(iconv('ISO-8859-1', 'UTF-8', $error['message'])),
             '</p>';
    }
    echo '<p>Programm mit Fehlern beendet.</p>';
    exit;
}
?>
```

Listing 12.2 *utils.php* – Hilfsfunktionen für Fehlerbehandlung und Datenausgabe

Wie in Listing 12.2 zu sehen, können Fehlerinformationen mit der Funktion *sqlsrv_errors()* abgefragt werden, welche ein Array mit Status, Code und Textbeschreibung der aufgetretenen Fehler zurückliefert.

Datenbankverbindungen

Eine Verbindung zu SQL Server herzustellen, ist der erste Schritt, um Abfragen und Anweisungen an die Datenbank zu senden. Verbindungen werden mit der Funktion *sqlsrv_connect()* hergestellt. Die Funktion hat zwei Parameter: den Name des Servers bzw. der Instanz, zu der verbunden werden soll, und Angaben zu den Verbindungseigenschaften.

Servernamen

SQL Server erlaubt die Verbindung auf unterschiedliche Arten, die wichtigsten drei davon sind:

- **Shared memory** Die Standardverbindung, wenn SQL Server und PHP am selben Server betrieben werden. Shared memory ist im Regelfall die effizienteste Art der Verbindung.
- **TCP/IP** Es wird eine Verbindung über das Netzwerk zu einem bestimmten Port aufgebaut
- **Named pipe** Eine Form der Interprozesskommunikation, über Pipes mit definiertem Namen

Beim Verbindungsaufbau können über den ersten Parameter nicht nur der Server und die Instanz bestimmt werden, sondern auch die Art der Verbindung. Die allgemeine Form des Servernamens ist:

```
[ <Protokoll>: ] <Servername> [ ,<Port> ]
```

Als Bezeichner für Protokolle sind folgende Kürzel vorgesehen:

- *lpc* für Shared memory-Verbindungen
- *tcp* für TCP/IP-Verbindungen
- *np* für Named pipes-Verbindungen

Tabelle 12.1 führt einige Beispiele für Servernamen für unterschiedliche Verbindungsarten auf.

WICHTIG Verbindungen zu SQL Server sind nur möglich, wenn entsprechende Endpunkte konfiguriert sind (siehe dazu auch Kapitel 9). Externe Zugriffe sind zudem nur möglich, wenn diese nicht von der Windows-Firewall oder externen Firewalls blockiert werden.

Verbindungsart	Beispiel	Beschreibung
Named pipes	np:\\.\pipe\sql\query	Verbindet mit Standardinstanz auf lokalem Rechner
	np:\\db.xmp.site\pipe\MSSQL$AdvWorks\sql\query	Verbindet mit der Instanz *AdvWorks* am Server *db.xmp.site* unter der Verwendung der Standardpipe
	np:\\db.xmp.site\pipe\php\app	Verbindet mit der Standardinstanz am Server *db.xmp.site* über die Pipe *php/app*
Shared memory	lpc:localhost	Verbindet mit der Standardinstanz am lokalen Server
	lpc:webhost.local\AdvWorks	Verbindet mit der Instanz *AdvWorks* am Server *webhost.local*, der gleich dem eigenen (lokalen) Server sein muss
	(local)	Verbindet mit der Standardinstanz am lokalen Server
TCP/IP	tcp:localhost	Verbindet mit Standardinstanz am lokalen Server
	tcp:db.xmp.site,2566	Verbindet mit der Standardinstanz am Server *db.xmp.site* auf Port 2566
	tcp:10.12.24.48	Verbindet mit Standardinstanz auf Server mit IP-Adresse 10.12.24.48 über Standardport (1443)

Tabelle 12.1 Beispiele für Servernamen je nach Verbindungsart

Authentifizierung

SQL Server erlaubt (bei entsprechender Konfiguration) zwei unterschiedliche Arten der Authentifizierung von Verbindungen:

- **Windows-Authentifizierung** Das Windows-Benutzerkonto, welches das PHP-Programm ausführt, wird auch zum Login bei SQL Server verwendet. Die Anmeldeinformationen des Prozesses werden in diesem Fall wiederverwendet.
- **SQL Server-Authentifizierung** Ein (ausschließlich) in SQL Server angelegter Benutzer wird für die Anmeldung verwendet. Bei der Verbindung müssen Benutzername und Passwort angegeben werden.

Beide Methoden haben Vor- und Nachteile. Für die Windows-Authentifizierung spricht, dass damit die gesamte Benutzerverwaltung zentralisiert werden kann, mit entsprechenden Vorteilen bei der Administration und Durchsetzung von Richtlinien. Zudem müssen in diesem Fall keine Passwörter im PHP-Skript stehen. Der Windows-Authentifizierung sollte insbesondere bei Intranet-Anwendungen der Vorzug gegeben werden.

Für die SQL Server-Benutzerkonten spricht, dass sie besonders bei öffentlich zugänglichen Webanwendungen, die vornehmlich anonyme Zugriffe bzw. selbstregistrierte und in der PHP-Anwendung verwaltete Konten haben, ihre Stärken ausspielen können: Einige wenige SQL Server-Konten dienen als Rollen für verschiedene Typen von Webnutzern bzw. zur Trennung verschiedener Bereiche der PHP-Anwendung.

Windows-Authentifizierung

Die Windows-Authentifizierung ist die Standardvariante beim Verbinden mit SQL Server. Zusätzliche Parameter müssen nicht angegeben werden. Um eine Verbindung zur Standardinstanz des lokal laufenden SQL Server mit Windows-Authentifizierung herzustellen, rufen Sie *sqlsrv_connect()* wie folgt auf:

```
$db = sqlsrv_connect('');
```

Der leere String steht für die Standardinstanz am lokalen Rechner. Alternativen sind *(local)*, ».« (Punkt), *(localhost)* oder *localhost*. Da keine weiteren Parameter angegeben sind, wird der Datenbankkontext auf die Standarddatenbank des Benutzers gesetzt, oder auf die Systemdatenbank *master*, falls der Benutzer keine Standarddatenbank gesetzt hat. Die zu verwendende Datenbank kann per Parameter angegeben werden:

```
$db = sqlsrv_connect('.', array ('Database' => 'AdventureWorksLT2008'));
```

Der Windows-Benutzer, der zur Authentifizierung gegenüber SQL Server verwendet wird, hängt von der Konfiguration von IIS und PHP ab, wie in Kapitel 5 beschrieben:

- **fastcgi.impersonate=0** Wenn die Übernahme der Identität des Anwendungsbenutzers ausgeschaltet ist, wird der Benutzer des IIS-Anwendungspools, beispielsweise *IIS AppPool\DefaultAppPool* verwendet.
- **fastcgi.impersonate=1, anonymer Webbenutzer** In diesem Fall wird der IIS-eigene anonyme Benutzer *NT-AUTORITÄT\IUSR* verwendet
- **fastcgi.impersonate=1, authentifizierter Webbenutzer** In diesem Fall wird die Identität des Webbenutzers von PHP übernommen und mit diesem Benutzerkonto auch die Authentifizierung bei SQL Server durchgeführt
- **fastcgi.impersonate=1, Pfadanmeldeinformation** Bei IIS-Anwendungen und virtuellen Verzeichnissen mit definiertem Benutzer in der Pfadanmeldeinformation wird dieser Benutzer für die Authentifizierung gegenüber SQL Server verwendet

Je nach Anwendungsfall müssen in SQL Server die Rechte entsprechend gesetzt werden.

TIPP Um (nach erfolgreicher Authentifizierung) den aktuellen Benutzer der Datenbankverbindung und die ausgewählte Datenbank in Erfahrung zu bringen, können Sie folgende Anweisung verwenden:

```
SELECT SYSTEM_USER AS CurrentUser, DB_NAME() AS CurrentDatabase
```

SQL Server-Authentifizierung

Die SQL Server-Authentifizierung ist unabhängig vom Windows-Benutzer, unter dem PHP ausgeführt wird: Benutzername und Passwort werden direkt im Verbindungsaufbau mit *sqlsrv_connect()* angegeben, wie in Listing 12.3 zu sehen. Der Benutzername wird über den Verbindungsparameter *UID*, das Passwort über den Parameter *PWD* übergeben.

```
// Verbinden über SQL Server-Authentifizierung
$server = '(local)';
$connectionInfo = array('UID' => 'Manfred',        // SQL Server-Benutzername
                        'PWD' => 'streng-geheim',  // Passwort
                        'Database' => 'AdventureWorksLT2008',
                        'CharacterSet' => 'UTF-8');
$db = sqlsrv_connect($server, $connectionInfo);
```

Listing 12.3 Verbindungsaufbau mit der SQL Server-Authentifizierung

Der offensichtliche Nachteil ist, dass das Passwort als Klartext im PHP-Skript enthalten ist und dass die Benutzerverwaltung nicht zentral (beispielsweise über Active Directory) erfolgen kann. Die Vorteile dieser Authentifizierungsmethode liegen in der Trennung von Datenbankbenutzer und ausführendem PHP-Benutzer und der einfachen Art und Weise, wie in derselben PHP-Anwendung zwischen verschiedenen SQL Server-Benutzern gewechselt werden kann.

Die SQL Server-Authentifizierung (bzw. die Verwendung von Datenbankbenutzern statt Betriebssystembenutzern) ist für PHP-Anwendungen die verbreitetste Methode der Authentifizierung.

Verbindungspooling

Unter Verbindungspooling wird das Wiederverwenden von bestehenden Datenbankverbindungen verstanden. Eine Verbindung aufzubauen und die Authentifizierung durchzuführen, kann im Vergleich zur T-SQL-Anweisung selbst einen merkbaren Mehraufwand bedeuten. Verbindungspooling hilft hier, Datenbankabfragen effizienter durchzuführen, indem bereits authentifizierte Verbindungen wiederverwendet werden, ein erneuter Aufbau der Verbindung also entfällt.

Standardmäßig ist Verbindungspooling beim SQL Server-PHP-Treiber aktiviert: Wenn eine mit *sqlsrv_connect()* neu herzustellende Verbindung bereits im Pool existiert, wird diese übernommen und der Verbindungsstatus zurückgesetzt. *sqlsrv_close()* schließt die Verbindung nicht vollständig, sondern gibt sie an den Verbindungspool zurück.

ACHTUNG Da Verbindungen wiederverwendet werden, sollten mit T-SQL-Anweisungen oder SQL Server-Prozeduren keine Veränderungen an Grundeinstellungen der Verbindungen selbst vorgenommen werden, da für nachfolgende Verwendungen der Verbindung diese Veränderungen dann ebenfalls wirksam sind. Das könnte zu unerwünschten Konsequenzen führen. Auch die Verwendung von Anwendungsrollen (mit *sp_setapprole()*) sollte vermieden werden.

Fragmentierung

Bei Verbindungspooling wird eine Verbindung nicht unmittelbar geschlossen, sondern vorerst in einen Pool zurückgegeben und erst verzögert, nach einer definierten inaktiven Zeit, geschlossen. Ob eine Verbindung wiederverwendet werden kann oder eine neue Verbindung (möglicherweise in einem neuen Pool) aufge-

baut wird, hängt von den Parametern von *sqlsrv_connect()* ab: Wenn Benutzer oder Datenbank verschieden sind, wird eine neue Verbindung in einem neuen Pool aufgebaut. Darauf ist beim Design von Webanwendungen zu achten.

Wenn eine Webanwendung viele unterschiedliche authentifizierte Benutzer hat, und die Benutzer, beispielsweise per Windows-Authentifizierung, auch für die Kommunikation mit SQL Server verwendet werden, wird für jeden Benutzer ein eigener Verbindungspool angelegt. Steigt die Zahl der zeitgleich aktiven Benutzer, führt das zur Fragmentierung der Verbindungspools und in Folge möglicherweise zur Beeinträchtigung der Leistung.

In diesem Fall sollte das Design der PHP-Anwendung auf wenige (rollenabhängige) Datenbankbenutzer umgestellt werden, die für alle Webbenutzer verwendet werden, indem:

- auf die SQL Server-Authentifizierung umgestellt wird
- oder bei Windows-Authentifizierung PHP mit *fastcgi.impersonate=0* konfiguriert wird oder für die Anwendung mithilfe der Pfadanmeldeinformation ein fixer Benutzer definiert wird

Generell sollten Datenbankverbindungen so spät wie möglich geöffnet und so früh wie möglich geschlossen werden, um Ressourcen zu sparen.

Verbindungsoptionen

Beim Verbindungsaufbau mit *sqlsrv_connect()* kann mit dem Parameter *ConnectionPooling* angegeben werden, ob Verbindungspooling benutzt werden soll (*true*) oder nicht (*false*). Die Standardeinstellung ist, dass Verbindungspooling verwendet wird. Listing 12.4 zeigt, wie für eine zu öffnende Verbindung Verbindungspooling unterbunden wird. In diesem Fall schließt *sqlsrv_close()* die Verbindung unmittelbar, eine Rückgabe in einen Pool oder ein zeitverzögertes Schließen finden nicht statt.

```
// Verbinden über Windows-Authentifizierung ohne Verbindungspooling
$server = '(local)';
$connectionInfo = array('ConnectionPooling' => false,
                        'Database' => 'AdventureWorksLT2008',
                        'CharacterSet' => 'UTF-8');
$db = sqlsrv_connect($server, $connectionInfo);
```

Listing 12.4 Aufbau einer Verbindung ohne Verbindungspooling

Datenbankabfragen

SQL Server bietet verschiedene Möglichkeiten, Datenbankabfragen von PHP aus zu erzeugen und die Resultate auszulesen. Im Folgenden werden parametrisierte Anweisungen vorgestellt, die eine sichere Übergabe von Parametern an die T-SQL-Anweisung erlauben und vorbereitete Anweisungen, die eine besonders effiziente mehrfache Ausführung von Anweisungen ermöglichen. Zum Auslesen von Ergebnissen werden zwei weitere Methoden vorgestellt: Auslesen der Datensätze als Objekte und Auslesen einzelner Spalten der Datensätze.

HINWEIS In den folgenden Beispielen werden zwei unterstützende Skripts verwendet, die Sie in Anhang A finden:

- *DatabaseConnection.php*: Öffnen und Schließen der Datenbankverbindung und Ausgabe von SQL Server-Fehlermeldungen
- *HTMLPage.php*: Erstellen einer HTML-Seite und sichere Ausgabe von Daten als HTML

Parametrisieren von Anweisungen

Die Sicherheit von Webanwendungen beruht größtenteils auf dem richtigen Umgang mit Eingangsdaten, die vom Benutzer stammen: Das Filtern und Maskieren dieser Daten ist essentziell, wenn Sie nicht Ihre Anwendung oder Ihre Benutzer gefährden wollen. Ungefilterte Daten können zu einer *SQL Injection* führen. Neben dem Filtern bieten parametrisierte Anweisungen eine Lösung für dieses Problem.

SQL Injection

SQL Injection – das bösartige Einfügen und Ausführen von SQL-Anweisungen – ist ein weit verbreitetes Problem von Webanwendungen. Der Grund liegt in der fehlenden Validierung und Maskierung von benutzerabhängigen Eingangsdaten. Listing 12.5 zeigt den Programmausschnitt eines typischen Beispiels:

```
$productID = $_GET['id'];
$query = "SELECT Name, Color FROM SalesLT.Product WHERE ProductID=$productID";
$stmt = sqlsrv_query($db, $query)
```

Listing 12.5 Fehlerhaftes Programm, das SQL Injection erlaubt

Das Programm erwartet eine Zahl, die Produktnummer, als Eingabewert. Das Problem ist, dass *$GET['id']* jede beliebige Zeichenfolge enthalten kann, zum Beispiel

```
-1 UNION SELECT CompanyName, EmailAddress FROM SalesLT.Customer;
```

Wird das PHP-Programm ausgeführt, hat *$query* folgenden Inhalt:

```
SELECT Name, Color FROM SalesLT.Product
WHERE ProductID=-1 UNION SELECT CompanyName, EmailAddress FROM SalesLT.Customer;
```

Die zusammengesetzte Anweisung stellt eine gültige Abfrage dar, nur werden statt Produktnamen und -farbe der Firmenname und die E-Mail-Adresse der Kunden ausgegeben.

SQL Injection kann aber auch verwendet werden, um Werte einzufügen (beispielsweise Benutzer mit erhöhten Rechten), Werte zu verändern, Zeilen oder Tabellen zu löschen, Prozeduren aufzurufen und vieles mehr. SQL Injection stellt eine ernste Bedrohung für Webanwendungen dar.

Filtern und Maskieren der Daten

Der erste Ansatz ist, gefährliche Zeichen zu filtern bzw. für die Übergabe an SQL Server zu maskieren. Tabelle 12.2 gibt einen Überblick über die wichtigsten Zeichen, die typischerweise bei SQL Injection verwendet werden. Wenn möglich, sollten diese Zeichen nicht zugelassen werden.

Zeichen	Bedeutung
; (Strichpunkt)	Grenzt eine T-SQL-Anweisung von der nächsten Anweisung ab
' (Apostroph)	Begrenzung von Zeichenfolgen
--	Leitet Kommentar ein (gilt bis zum Ende der Zeile)
/* und */	Begrenzungen für Kommentare

Tabelle 12.2 Gefährliche Zeichen und Zeichenfolgen, die zur SQL Injection verwendet werden

Mögliche Filterungen für Zahlen, wie der *ProductID*, könnten wie folgt aussehen:

```
// Variante 1: cast auf Integer
$productID = (int)$_GET['id'];
// Variante 2: Alles was keine Zahl ist ausfiltern
$productID = preg_replace('/[^0-9]/', '', $_GET['id']);
// Variante 3: filter_input-Funktion von PHP verwenden
$productID = filter_input(INPUT_GET, 'id', FILTER_SANITIZE_NUMBER_INT);
```

Werden Zeichenfolgen als Eingabedaten übergeben, etwa um ein Produkt nach Namen zu suchen, ist die Filterung so eng wie möglich auszulegen. Wenn also beispielsweise nur Buchstaben und Leerzeichen zulässig sind, dann könnte folgende Funktion verwendet werden:

```
mb_regex_encoding('UTF-8');
if (!mb_check_encoding($_GET['produkt'], 'UTF-8')) {
    die('Fehlerhafte Zeichencodierung bei Eingabewert.');
}
$name = mb_ereg_replace('[^[:alpha:] ]', '', $_GET['produkt']);
```

ACHTUNG Wenn Sie mit Unicode-Eingabedaten arbeiten (wie bei den Beispielen in diesem Buch), dann sollten Sie nur Multibyte-Zeichenfolgen-Funktionen (*mb_**) verwenden. PCRE-Funktionen (*preg_**) und andere einfache Zeichenoperationen können falsche Ergebnisse liefern oder die Codierung der Zeichenfolge zerstören.

Wenn auch der Apostroph zugelassen werden soll, dann sollte dieser verdoppelt werden:

```
$name = mb_ereg_replace("[^[:alpha:]' ]", '', $_GET['produkt']);
$name = mb_ereg_replace("'", "''", $name);
```

Jetzt kann *$name* für T-SQL-Abfragen verwendet werden.

WICHTIG Sicherheit gegen SQL Injection und verwandte Sicherheitsschwächen ist nur gewährleistet, wenn Sie über die ganze Verarbeitungskette (HTML, PHP, SQL Server) dieselbe Zeichencodierung verwenden. Andernfalls kann die unterschiedliche Interpretation der Daten bei den Übergängen zu Sicherheitsproblemen führen. Im Buch wird deshalb durchgängig UTF-8 verwendet.

Parametrisieren von Anweisungen

SQL Server bietet eine einfache Alternative, um sicherzustellen, dass Daten richtig maskiert sind: parametrisierte Abfragen. *sqlsrv_query()* kann weitere Parameter beim Aufruf entgegennehmen:

```
sqlsrv_query($DBVerbindung, $T-SQL [,$parameter [,$optionen]])
```

$parameter ist ein Array von Werten, die in die T-SQL-Anweisung an Stelle von Fragezeichen eingefügt werden. *$optionen* geben zusätzliche Abfrageoptionen an, wie zum Beispiel eine Zeitgrenze für die Abarbeitung der Abfrage. Um beispielsweise die Nummer eines bestimmten Produkts abzufragen, würde der *sqlsrv_query()*-Aufruf wie folgt aussehen:

```
sqlsrv_query($db,
             'SELECT ProductID FROM SalesLT.Products WHERE Name=?',
             array($name))
```

Das Fragezeichen bei *Name=?* wird durch den Wert von *$name* ersetzt. *sqlsrv_query()* stellt dabei sicher, dass der Wert so übergeben wird, dass keine SQL Injection passieren kann.

Der Vorteil von parametrisierten Anfragen ist, dass der SQL Server-PHP-Treiber automatisch die für die Datenbankverbindung passende Codierung wählt, um den Wert in die SQL-Anweisung einzufügen und die Maskierung passend für den Datentyp vornehmen kann.

Beispielprogramm

Das Beispielprogramm zur Namenssuche nach Produkten ist in Listing 12.6 und Listing 12.7 aufgeführt. Es setzt eine parametrisierte Abfrage ein und bereinigt die Eingabedaten von unerwünschten Zeichen.

search_products.php (Listing 12.6) übernimmt dabei den Teil der Ein- und Ausgabe:

- Ein *HTMLPage*-Objekt wird angelegt und ein Formular in die Seite eingefügt
- Falls ein Suchbegriff eingegeben worden ist, werden die Eingabedaten mit *sanitizeName()* bereinigt, wie in Abschnitt »Filtern und Maskieren der Daten« ab Seite 276 beschrieben
- Nachdem *getProductsByName()* die Daten aus der Datenbank ausgelesen hat, werden die Ergebnisse als Tabelle in das HTML eingefügt und die HTML-Seite ausgegeben

HINWEIS Die PHP-Erweiterung für die Multibyte-Funktionen (*mb_**) ist in der Standardinstallation von PHP nicht aktiviert. Um die Erweiterung zu aktivieren, fügen Sie folgende Zeile im *php.ini* ein und starten Sie die zugehörigen IIS-Anwendungspools anschließend neu:

```
extension=php_mbstring.dll
```

```
<?php
namespace net\xmp\phpbuch;
require './DatabaseConnection.php';
require './HTMLPage.php';
require './search_products_db.php';

$html = new HTMLPage('AdventureWorks : Produktsuche');
$form = <<<EOF
<form action="" method="get">
Produktname: <input name="produkt" />
<input type="submit" value="Suchen" />
</form><br />
EOF;
$html->addHTML($form);
if (isset($_GET['produkt'])) {
    $name = sanitizeName($_GET['produkt']);
    $products = getProductsByName($name);
    if ($products) {
        $html->addTable($products);
    }
    else {
        $html->addElement('p', 'Keine Produkte gefunden.');
    }
}
$html->printPage();
exit;
```

```
/**
 * Zeichenfolge auf richtige Codierung prüfen und
 * verbotene Zeichen ausfiltern
 */
function sanitizeName($txt)
{
    if (!mb_check_encoding($txt, 'UTF-8')) {
        die('Fehlerhafte Zeichencodierung bei Eingabewert.');
    }
    mb_regex_encoding('UTF-8');
    return mb_ereg_replace("[^-[:alnum:]', ]", '', $txt);
}
?>
```

Listing 12.6 *search_products.php* – Suche von Produkten nach Namen

search_products_db.php (Listing 12.7) enthält die Funktion zur Datenbankabfrage:

- Zu Beginn wird ein *DatbaseConnection*-Objekt angelegt und die Verbindung zur Datenbank geöffnet
- Die SQL-Anweisung ist parametrisiert: *$query* enthält in der *WHERE*-Klausel den Platzhalter (Fragezeichen), der durch die Parameter (*$params*, nur ein Wert) ersetzt wird
- In Folge werden mit *sqlsrv_fetch_array()* die Daten ausgelesen und in ein Array zur HTML-Ausgabe geschrieben
- Zum Abschluss wird die zur Abfrage zugehörige Ressource mit *sqlsrv_free_stmt()* freigegeben und die Datenbankverbindung geschlossen

```
<?php
namespace net\xmp\phpbuch;

/**
 * Abfrage aller Produkte, deren Name mit $name anfängt.
 * @return array Daten für die HTML-Tabelle
 */
function getProductsByName($name)
{
    $db = new DatabaseConnection();
    $db->connect();
    // Produkte mit Namen und Listenpreis auswählen
    $query = 'SELECT ProductID, Name, ListPrice
              FROM SalesLT.Product WHERE Name LIKE ?
              ORDER BY Name';
    $params = array($name . '%');
    // Abfrage ausführen
    $stmt = sqlsrv_query($db->handle, $query, $params);
    if ($stmt === false) {
        $db->exitWithError('Abfrage der Produktdaten fehlgeschlagen.');
    }
    if (!sqlsrv_has_rows($stmt)) {
        return false;   // Keine Treffer in Datenbank (leeres Ergebnis)
    }
    // Einzelne Zeilen des Ergebnisses auslesen
    $table = array( array('ID', 'Produkt', 'Listenpreis') );
    while ($row = sqlsrv_fetch_array($stmt)) {        $table[] = array($row['ProductID'], $row['Name'],
                        $row['ListPrice']);
    }
    // null == keine weiteren Zeilen mehr, false == Fehler
    if ($row === false) {
```

```
        $db->exitWithError('Abfruf von Produktdateneintrag fehlgeschlagen.');
    }
    sqlsrv_free_stmt($stmt);
    $db->close();
    return $table;
}
?>
```

Listing 12.7 *search_products_db.php* – Datenbankfunktion für die Namenssuche

TIPP Mit der Funktion *sqlsrv_has_rows()* lässt sich abfragen, ob Ergebniszeilen vorhanden sind oder nicht. In Listing 12.7 wird die Funktion verwendet, um festzustellen, ob der Suchbegriff zu Treffern geführt hat.

Eine Alternative ist die Funktion *sqlsrv_num_rows()*, die die Anzahl der Zeilen im Ergebnis zurückliefert. Jedoch kann diese Funktion nicht mit dem Standardcursortyp von SQL Server-Anfragen genutzt werden.

Abbildung 12.3 zeigt das Ergebnis der Produktsuche nach »Men's«.

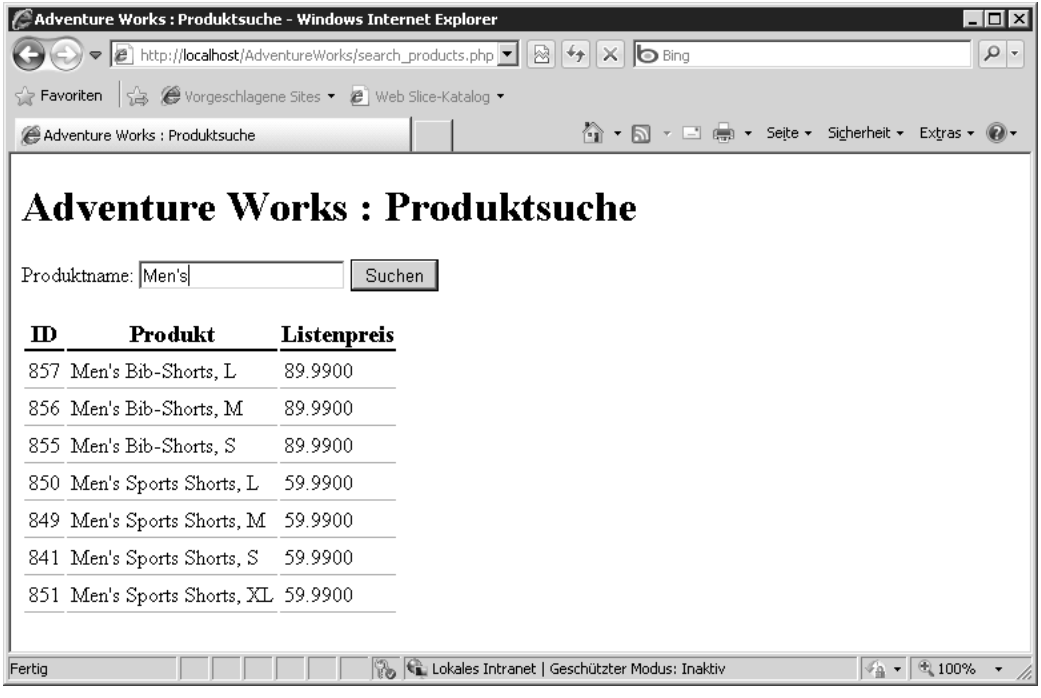

Abbildung 12.3 Ergebnis der Produktnamensuche mithilfe einer parametrisierten T-SQL-Anweisung

Auslesen von Ergebnissen

Nachdem eine Anforderung an SQL Server abgesetzt worden ist, wird eine Anweisungsressource zurück geliefert, über die Ergebnisse ausgelesen werden können. Bisher wurden Ergebnisse immer mit *sqlsrv_fetch_array()* ausgelesen. Im Folgenden sollen zwei weitere Arten vorgestellt werden: Ergebnisse als Objekte auslesen und einzelne Felder von Objekten auslesen.

Auslesen als Objekt

Um Ergebniszeilen einer T-SQL-Abfrage als Objekt abzufragen, wird die Funktion *sqlsrv_fetch_object()* verwendet. Die Funktion erlaubt zusätzlich die Angabe einer Klasse, die für das Ergebnisobjekt instanziiert werden soll.

Listing 12.8 zeigt einen Programmausschnitt (ohne Fehlerbehandlung). Die Eigenschaften des erzeugten Objekts tragen die Namen der ausgewählten Felder des *SELECT*-Ausdrucks.

```
$stmt = sqlsrv_query($db, 'SELECT DISTINCT TOP(10) FirstName, LastName, EmailAddress
                     FROM SalesLT.Customer ORDER BY LastName, FirstName');
while ($obj = sqlsrv_fetch_object($stmt)) {
    printf("<li>%s %s &lt;%s&gt;</li>\n", htmlspecialchars($obj->FirstName),
           htmlspecialchars($obj->LastName), htmlspecialchars($obj->EmailAddress));
}
sqlsrv_free_stmt($stmt);
```

Listing 12.8 Abfrage von Daten mit *sqlsrv_fetch_object()*

Analog zu *sqlsrv_fetch_array()* liefert *sqlsrv_fetch_object()* im Fehlerfall *false* zurück und wenn keine weiteren Ergebniszeilen mehr vorhanden sind, den Wert *null*.

Ergebnisspalten müssen einen Namen tragen, damit sie als Objekteigenschaft zugewiesen werden können. Namenlose Spalten erzeugen eine Warnung bzw. Fehler. Berechnete Spalten der Art *count(*)*, *max(ListPrice)*, *SubTotal+TaxAmt* müssen ein Alias zugewiesen bekommen, damit sie als Objekteigenschaft gesetzt werden können, beispielsweise *count(*) AS anzahl*.

WICHTIG Beachten Sie die Groß-/Kleinschreibung der Spaltennamen: Die Namen der Objekteigenschaften werden genau in der Schreibweise der T-SQL-Anweisung gesetzt. Wenn Sie in Folge von der so definierten Schreibweise abweichen, greifen Sie auf eine undefinierte Objekteigenschaft zu.

Um ein Ergebnisobjekt einer bestimmten Klasse zu instanziieren, wird ein weiterer Parameter angegeben, wie in Listing 12.9 zu sehen.

```
class Customer
{
    function printData()
    {
        printf("<li>%s %s &lt;%s&gt;</li>\n", htmlspecialchars($obj->FirstName),
               htmlspecialchars($obj->LastName), htmlspecialchars($obj->EmailAddress));
    }
}
$stmt = sqlsrv_query($db, 'SELECT DISTINCT TOP(10) FirstName, LastName, EmailAddress
                     FROM SalesLT.Customer ORDER BY LastName, FirstName');
while ($obj = sqlsrv_fetch_object($stmt, 'Customer')) {
    $obj->printData();
}
sqlsrv_free_stmt($stmt);
```

Listing 12.9 Instanziieren einer bestimmten Klasse mit *sqlsrv_fetch_object()*

Wenn die Klasse in einem PHP 5.3-Namespace liegt, muss der vollständige Klassenname angegeben werden. Wenn also beispielsweise die Klasse *Customer* im Namespace *net\xmp\phpbuch* liegt, würde der Aufruf so aussehen:

```
$obj = sqlsrv_fetch_object($stmt, 'net\\xmp\\phpbuch\\Customer');
```

Wenn der Konstruktor einer Klasse Parameter benötigt, können diese als Array bei *sqlsrv_fetch_object()* mitgegeben werden, zum Beispiel:

```
$obj = sqlsrv_fetch_object($stmt, 'Customer', array($param1, $param2, $param2));
```

Auslesen einzelner Felder

Die dritte Variante, Ergebniszeilen auszulesen, ist das Auslesen einzelner Felder mit einer Kombination der beiden Funktionen *sqlsrv_fetch()* und *sqlsrv_get_field()*: Eine Ergebniszeile wird mit *sqlsrv_fetch()* geholt, danach können einzelne Spalten mit *sqlsrv_get_field()* ausgelesen werden.

Listing 12.10 zeigt ein Beispiel: Nachdem die T-SQL-Anweisung abgesendet worden ist, wird mit *sqlsrv_fetch($stmt)* die nächste Zeile des Ergebnisses geholt. War das Lesen der Zeile erfolgreich, liefert die Funktion *true*, im Fehlerfall *false* und, falls keine weiteren Zeilen mehr vorhanden sind, den Wert *null* zurück. In Folge werden die einzelnen Spalten per *sqlsrv_get_field()* ausgelesen. Die Spalten werden über einen Index angesprochen, Spaltennamen können nicht verwendet werden.

```
$stmt = sqlsrv_query($db, 'SELECT DISTINCT TOP(10) FirstName, LastName, EmailAddress
                           FROM SalesLT.Customer ORDER BY LastName, FirstName');
while (sqlsrv_fetch($stmt)) {
    printf("<li>%s %s %s</li>\n", sqlsrv_get_field($stmt, 0), sqlsrv_get_field($stmt, 1),
                                  sqlsrv_get_field($stmt, 2));
}
sqlsrv_free_stmt($stmt);
```

Listing 12.10 Auslesen von Ergebnissen mit *sqlsrv_fetch()* und *sqlsrv_get_field()*

Während die Entscheidung zwischen *sqlsrv_fetch_array()* und *sqlsrv_fetch_object()* vom persönlichen Programmierstil abhängt, bietet die Kombination aus *sqlsrv_fetch()* und *sqlsrv_get_field()* zwei wesentliche Unterscheidungsmerkmale: Erstens wird nur genau jene Spalte in den PHP-Speicherraum geladen, die gerade angefordert wird, während bei *sqlsrv_fetch_array()* und *sqlsrv_fetch_object()* sofort die ganze Zeile in den PHP-Speicherraum transferiert wird. Gerade bei großen Spalten (*nvarchar(MAX)*, *varbinary(MAX)*) kann das eine Auswirkung haben. Zweitens kann der PHP-Typ der auszulesenden Spalte angegeben werden, und – ebenfalls wieder für große Spalten von Interesse – die Spalte als Stream ausgelesen werden. Details dazu finden Sie im Abschnitt »Datentypen« ab Seite 286.

Vorbereitete Anweisungen

Wenn eine T-SQL-Anweisung mehrfach im PHP-Programm ausgeführt wird, können vorbereitete Anweisungen (*prepared statements*) die Programmierung vereinfachen und zur Leistungssteigerung beitragen. Bei vorbereiteten Anwendungen wird *sqlsrv_query()* auf zwei Schritte aufgeteilt:

- **sqlsrv_prepare()** Die T-SQL-Anweisung wird analysiert und für die Ausführung vorbereitet. Parameter werden noch nicht in die Anweisung eingesetzt.
- **sqlsrv_execute()** Die Parameter werden eingesetzt und die Anweisung ausgeführt. Dieser Schritt kann (mit jeweils neuen Daten) mehrfach erfolgen. Da die Analyse und Ausführungsvorbereitung der Anweisung wegfallen, kann die Leistung erhöht bzw. der Ressourcenverbrauch verkleinert werden.

Im Folgenden soll die Programmierung mit vorbereiteten Anweisungen anhand einer kleinen Beispielanwendung gezeigt werden. Vor allem die Bindung der Parameter ist dabei von Bedeutung. Die Beispielanwendung aktualisiert mithilfe einer vorbereiteten T-SQL-Anweisung die Bestellmenge und den Rabatt für einzelne Produkte auf einer Rechnung. Die zugehörigen Daten sind in der Tabelle *SalesLT.SalesOrderDetail* zu finden.

Listing 12.11 enthält die Datenbankklasse. Neben dem Konstruktor ist auch eine Methode *getByHeader()* enthalten, die zur Auflistung der Inhalte vor und nach der Aktualisierung verwendet wird. *getByHeader()* hat den bereits bekannten Aufbau: Ausführen einer parametrisierten T-SQL-Anweisung mit *sqlsrv_query()*, dann Auslesen der Ergebniszeilen mit *sqlsrv_fetch_array()*, zum Schluss Freigeben der Anweisungsressource mit *sqlsrv_free_stmt()*.

Die Methode *prepare()* bereitet die T-SQL-Anweisung mit *sqlsrv_prepare()* vor. Auf den ersten Blick handelt es sich um eine parametrisierte Anweisung, wie an den Platzhaltern in T-SQL ersichtlich. Ein wesentlicher und wichtiger Punkt ist aber, dass die Parameter als Verweise (*&$...*) gebunden werden, nicht als Werte (*$...*). Dadurch übertragen sich alle späteren Änderungen an den Variablen an jene Stellen, bei denen die Verweise verwendet worden sind.

Die Methode *update()* führt dann *sqlsrv_execute()* aus. Die Parameter werden nicht direkt bei *sqlsrv_execute()* gesetzt, sondern es werden die Werte der über Verweise gebundenen Variablen geändert. Die neuen Werte werden automatisch für die Ausführung der T-SQL-Anweisung übernommen.

Die Methode *free()* gibt schließlich die verwendete Anweisungsressource frei.

TIPP Mit der Funktion *sqlsrv_rows_affected()* können bei *UPDATE*, *INSERT* und *DELETE* die Anzahl der betroffenen Zeilen abgefragt werden. Wurden keine Zeilen verändert, liefert die Funktion 0 zurück, wenn es keine Information gibt (beispielsweise nach einem *SELECT*), liefert die Funktion -1 zurück, *false* wird im Fehlerfall retourniert.

```
<?php
namespace net\xmp\phpbuch;
class UpdateOrders
{
    protected $db;
    protected $stmt;

    // An Anweisung gebundene Variablen
    protected $sql_qty, $sql_discount, $sql_id;

    /**
     * Konstruktor setzt Datenbankverbindung
     */
    function __construct($db)
    {
        $this->db = $db;
    }

    /**
     * Relevante Daten aus Tabelle auslesen und in HTML-Array schreiben.
```

```
     * @param int $id Nummer des SalesOrder
     */
    function getByHeader($id)
    {
        $query = 'SELECT SalesOrderDetailID, OrderQty, UnitPriceDiscount
                  FROM SalesLT.SalesOrderDetail WHERE SalesOrderID = ?';
        $params = array($id);
        $stmt = sqlsrv_query($this->db->handle, $query, $params);
        if ($stmt === false) {
            $this->db->exitWithError('Abfrage der Daten fehlgeschlagen.');
        }
        // Einzelne Zeilen des Ergebnisses auslesen
        $table = array( array('ID', 'Menge', 'Rabatt') );
        while ($row = sqlsrv_fetch_array($stmt)) {
            $table[] = array($row['SalesOrderDetailID'], $row['OrderQty'],
                             $row['UnitPriceDiscount']);
        }
        if ($row === false) {
            $this->db->exitWithError('Abruf der Daten fehlgeschlagen.');
        }
        sqlsrv_free_stmt($stmt);
        return $table;
    }

    /**
     * Vorbereiten der Aktualisierungsanweisung.
     */
    function prepare()
    {
        $query = 'UPDATE SalesLT.SalesOrderDetail
                  SET OrderQty = ?, UnitPriceDiscount = ?
                  WHERE SalesOrderDetailID = ?';
        // Parameter müssen als Verweise (&$...) verwendet werden.
        // Dadurch übertragen sich alle späteren Änderungen der Variablen
        // in die vorbereitete T-SQL-Anweisung
        $params = array(&$this->sql_qty, &$this->sql_discount, &$this->sql_id);
        // Anweisung vorbereiten
        $this->stmt = sqlsrv_prepare($this->db->handle, $query, $params);
        if ($this->stmt === false) {
            $this->db->exitWithError("Anweisungsvorbereitung fehlgeschlagen.");
        }
    }
    /**
     * Vorbereitete Anweisung ausführen.
     * Als Parameter werden die zu aktualisierenden Werte übergeben.
     */
    function update($id, $qty, $discount)
    {
        // Aktualisierung der Variablen überträgt sich auf T-SQL-Anweisung,
        // da Variablen als Verweise gebunden worden sind.
        $this->sql_id = $id;
        $this->sql_qty = $qty;
        $this->sql_discount = $discount;
```

```
        if (sqlsrv_execute($this->stmt) === false) {
            $this->db->exitWithError('Aktualisierung fehlgeschlagen.');
        }
    }

    function free()
    {
        sqlsrv_free_stmt($this->stmt);
    }
}
?>
```

Listing 12.11 *update_salesorder_db.php* – Datenbankklasse für die Aktualisierung der Bestelldaten mit vorbereiteten T-SQL-Anweisungen

Listing 12.12 enthält den notwendigen Rahmen zur Ausführung der Aktualisierung:

- Die neuen, aktuellen Werte sind fix definiert. In einer wirklichen Anwendung würden Sie über Eingabedaten (*$_GET*, *$_POST*) gesetzt werden
- Im Anschluss werden die Datenbankverbindung geöffnet, die *UpdateOrders*-Klasse instanziiert und die Werte der Tabelle vor der Aktualisierung ausgelesen (*getByHeader()*) und angezeigt (*$html->addTable()*)
- Die mehrfache Ausführung der vorbereiteten Anweisung ist unspektakulär: Die Anweisung wird mit *prepare()* vorbereitet, *update()* wird in einer Schleife für alle Datensätze aufgerufen und zum Abschluss wird die Anweisung mit *free()* freigegeben. Hier zeigt sich, wie vorbereitete Anweisungen die Programmierung vereinfachen können.
- Zu Demonstrationszwecken werden die aktualisierten Daten der Tabelle noch einmal ausgelesen (*getByHeader()*) und angezeigt

```
<?php
namespace net\xmp\phpbuch;
require './DatabaseConnection.php';
require './HTMLPage.php';
require './update_salesorder_db.php';

// Fix definiert für Demonstrationszwecke
$updateOrderHeader = 71915;
$updateOrders = array( array(113089, 6, 0.10), array(113090, 1, 0.0),
                       array(113091, 10, 0.20), array(113093, 3, 0.035) );
$html = new HTMLPage('AdventureWorks : Korrektur Bestellmengen und Rabatt');
$db = new DatabaseConnection();
$db->connect();
$orders = new UpdateOrders($db);
// Daten vor Aktualisierung auslesen und anzeigen
$before = $orders->getByHeader($updateOrderHeader);
$html->addHTML('<div style="position:absolute;top:4em">');
$html->addElement('h2', 'Vor Aktualisierung');
$html->addTable($before);

// Abfrage vorbereiten und mehrfach ausführen
$orders->prepare();
foreach ($updateOrders as $order) {
    list($id, $qty, $discount) = $order;
    $orders->update($id, $qty, $discount);
}
```

```
$orders->free();

// Daten nach Aktualisierung auslesen und anzeigen
$after = $orders->getByHeader($updateOrderHeader);
$html->addHTML('</div><div style="position:absolute;top:4em;left:15em">');
$html->addElement('h2', 'Nach Aktualisierung');
$html->addTable($after);
$html->addHTML('</div>');
$db->close();
$html->printPage();
?>
```

Listing 12.12 *update_salesorder.php* – Aktualisieren von Bestelldaten mit vorbereiteten T-SQL-Anweisungen

Abbildung 12.4 zeigt das Ergebnis eines Durchlaufes: Die Änderungen wurden entsprechend übernommen.

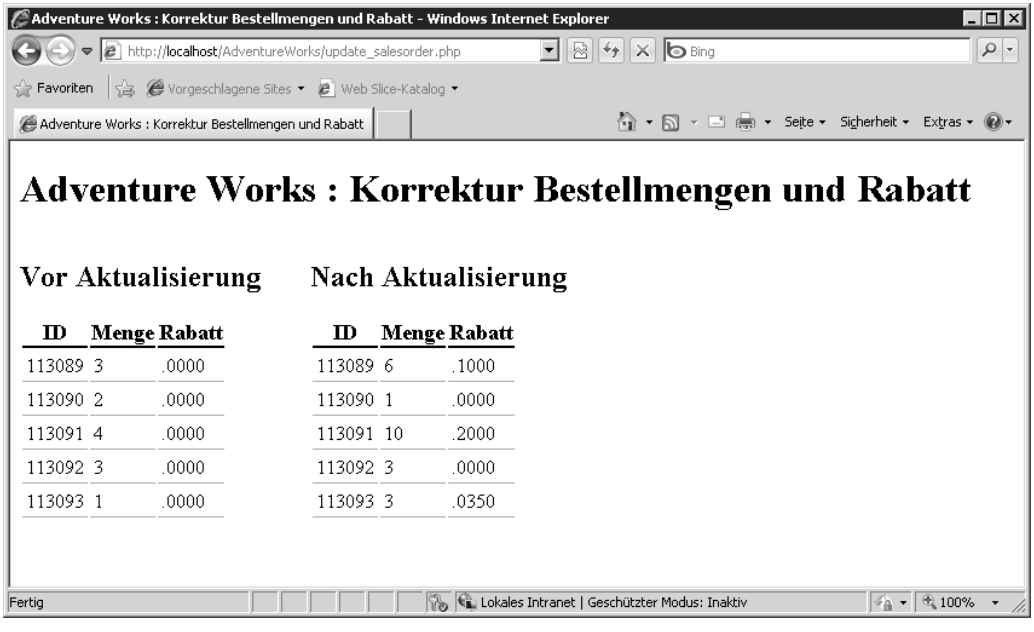

Abbildung 12.4 Beispielprogramm zur Aktualisierung der Bestellmengen und Rabatte

Datentypen

SQL Server und PHP haben jeweils eigene Datentypen. Bei der Übertragung von Daten müssen diese in den passenden Datentyp konvertiert werden. Häufig wird dabei der Weg über Zeichenfolgen bzw. Zahlen genommen: Eine PHP-Gleitkommazahl wird beispielsweise im T-SQL-Statement zur Zeichenfolge (eingeschlossen in Apostrophen) und in SQL Server zum *smallmoney*-Typ. Wenn Daten auf diese Art konvertiert werden, ist diese Konvertierung implizit, das heißt von SQL Server und PHP werden die Standardkonvertierungen angewandt.

Konvertieren von PHP nach SQL Server

Die Standardkonvertierung von Datentypen kann mithilfe von parametrisierten Anweisungen, bei denen explizite Datentypen angegeben werden, gesteuert werden. Listing 12.13 zeigt ein Beispiel: Die Parameter *$sellEnd* und *$maxPrice* werden auf die SQL Server-Datentypen *datetime* und *money* konvertiert. In *$params* wird dazu statt einzelner Variablen ein Array übergeben, dessen letzter Eintrag der gewünschte SQL Server-Datentyp ist.

```
$query = 'SELECT ProductID, Name
          FROM SalesLT.Product
          WHERE ProductCategoryID = ? AND SellEndDate >= ? AND ListPrice <= ?';
$categoryID = 6;
$sellEnd = '2003-01-01';
$maxPrice = 799.90;
$params = array($categoryID,
                array($sellEnd, null, null, SQLSRV_SQLTYPE_DATETIME),
                array($maxPrice, null, null, SQLSRV_SQLTYPE_MONEY));
$stmt = sqlsrv_query($db, $query, $params);
```

Listing 12.13 Angabe von SQL Server-Datentypen bei parametrisierter Anfrage

Die vollständige Liste der Konstanten *SQLSRV_SQLTYPE_** ist in der Dokumentation des SQL Server-PHP-Treibers zu finden. Als Faustregel gilt, dass der Name des Datentyps in Großbuchstaben angehängt wird, beispielsweise bezeichnet die Konstante *SQLSRV_SQLTYPE_DATETIMEOFFSET* den SQL Server-Typ *datetimeoffset*.

Konvertieren von SQL Server nach PHP

In der umgekehrten Richtung können die gewünschten PHP-Datentypen der Ergebniszeilen bei der Abfrage mit *sqlsrv_get_field()* angegeben werden. Listing 12.14 zeigt ein Beispiel. Der gewünschte PHP-Datentyp wird als dritter Parameter von *sqlsrv_get_field()* angegeben.

```
$query = 'SELECT ProductID, Weight, ListPrice, SellEndDate
          FROM SalesLT.Product';
$stmt = sqlsrv_query($db, $query);
while (sqlsrv_fetch($stmt)) {
    echo sqlsrv_get_field($stmt, 0, SQLSRV_PHPTYPE_INT), ' ',
         sqlsrv_get_field($stmt, 1, SQLSRV_PHPTYPE_FLOAT), ' ',
         sqlsrv_get_field($stmt, 2, SQLSRV_PHPTYPE_FLOAT), ' ',
         sqlsrv_get_field($stmt, 3, SQLSRV_PHPTYPE_STRING(SQLSRV_ENC_CHAR)), "\n";
}
```

Listing 12.14 Angabe der PHP-Datentypen bei Abfrage mit *sqlsrv_get_field()*

Mögliche PHP-Datentypen sind in Tabelle 12.3 aufgeführt. Eine direkte Konvertierung in *Boolean* oder Arrays und Objekte ist nicht möglich. (Objekte können jedoch mit *sqlsrv_fetch_object()* passend instanziiert werden.)

PHP-Datentyp	Konstante	Hinweise
DateTime	SQLSRV_PHPTYPE_DATETIME	Datumstypen werden bereits standardmäßig in PHP-*DateTime* umgewandelt
Float	SQLSRV_PHPTYPE_FLOAT	Für alle Zahlentypen mit Nachkommastellen geeignet
Integer	SQLSRV_PHPTYPE_INT	Nur ganzzahlige SQL Server-Datentypen können nach Integer konvertiert werden. Eine automatische Konvertierung von beispielsweise *money* nach Integer ist nicht möglich.
Stream	SQLSRV_PHPTYPE_STREAM()	Eine Kodierung muss als Parameter angegeben werden: *SQLSRV_ENC_BINARY:* Daten werden unbehandelt weitergegeben. *SQLSRV_ENC_CHAR:* Daten werden entsprechend dem aktuellen 8-Bit-Windows-Zeichensatz konvertiert, nicht konvertierbare Zeichen durch Fragezeichen ersetzt. *"UTF-8":* Daten werden UTF-8 kodiert übergeben.
String	SQLSRV_PHPTYPE_STRING()	

Tabelle 12.3 PHP-Datentypen, in die konvertiert werden kann

Eine – vor allem für Umsteiger von anderen Datenbanken – häufige Fehlerquelle ist, dass der SQL Server-PHP-Treiber standardmäßig die Datums- und Zeittypen als *DateTime*-PHP-Datentyp zurückliefert und nicht als Zeichenfolge. Mit *sqlsrv_get_field()* kann, wie in Listing 12.14 gezeigt, der Datentyp explizit auf *String* umgesetzt werden. Alternativ kann direkt beim Verbindungsaufbau angegeben werden, dass Datumstypen als *String* geliefert werden sollen:

```
$serverName = "(local)";
$connectionInfo = array('Database' => 'AdventureWorksLT2008', 'ReturnDatesAsStrings' => true);
$db = sqlsrv_connect($serverName, $connectionInfo);
```

Streams

Der SQL Server-PHP-Treiber bietet die Möglichkeit, Daten als Streams zu behandeln. Das ist insbesondere für binäre Datentypen (*binary*, *varbinary*) oder große Datenmengen (zum Beispiel bei *varchar(MAX)*) interessant.

Auslesen von Daten als Stream

Werden Daten mit *sqlsrv_get_field()* ausgelesen, werden die genannten Typen standardmäßig als Stream retourniert. Ein wesentlicher Vorteil ist, dass die Daten nicht vollständig im PHP-Speicherraum gehalten werden müssen, sondern stückweise über den Stream ausgelesen und ausgegeben werden können.

An Hand des Beispiels der Produktsuche aus Abschnitt »Parametrisieren von Anweisungen« ab Seite 278 soll die Verwendung verdeutlich werden. Die Produktsuche soll um Abbildungen der Produkte erweitert werden.

Listing 12.15 zeigt das zugehörige Skript zum Auslesen und Anzeigen der Bilder. Nach dem Öffnen der Datenbankverbindung wird die Spalte *ThumbNailPhoto* von *SalesLT.Products* abgefragt. Die Spalte ist vom Typ *varbinary(MAX)* und wird standardmäßig als Stream behandelt. Deshalb kann *$stream* direkt mit der Funktion *fpassthru()* ausgegeben werden.

```
<?php
namespace net\xmp\phpbuch;
require './DatabaseConnection.php';
require './HTMLPage.php';

// Datenbankverbindung herstellen
$db = new DatabaseConnection();
$db->connect();
// Abbildung für Produkt abfragen
$query = 'SELECT ThumbNailPhoto FROM SalesLT.Product WHERE ProductID = ?';
$params = array( (int)$_GET['id'] );
$stmt = sqlsrv_query($db->handle, $query, $params);
if (!sqlsrv_fetch($stmt)) {
    $db->exitWithError('Abfrage der Abbildung fehlgeschlagen.');
}
// varbinary() wird als Stream retourniert und direkt ausgegeben
$stream = sqlsrv_get_field($stmt, 0);
header('Content-type: image/gif');
fpassthru($stream);
// Statement freigeben, Datenbank schließen
sqlsrv_free_stmt($stmt);
$db->close();
```

Listing 12.15 *get_image.php* – Bild als Stream aus der Datenbank auslesen und anzeigen

HINWEIS Achten Sie darauf, dass Ihr Editor beim PHP-Skript keine UTF-8 Kodierung (*byte order mark*, BOM) am Anfang einfügt, da ansonsten *header()* nicht funktioniert bzw. die zusätzlichen Bytes das Grafikformat unlesbar machen.

Da Leerzeichen am Ende der Datei auch Probleme machen können, wurde das schließende PHP-Tag *?>* weggelassen, was gültig und gängige Praxis in solchen Fällen ist.

Wollte man die Grafik nicht als Stream retourniert bekommen, sondern als Zeichenfolge in einer Variablen speichern, wäre die Codierung direkt anzugeben:

```
// varbinary() wird als Zeichenfolge retourniert und ausgegeben
$img = sqlsrv_get_field($stmt, 0, SQLSRV_PHPTYPE_STRING(SQLSRV_ENC_BINARY));
header('Content-type: image/gif');
echo $img;
```

get_image.php aus Listing 12.15 stellt nur die Anzeige der Bilder zur Verfügung. Um die Anzeige der Bilder in das Suchresultat einzubauen, muss Listing 12.6 (ab Seite 279) abgeändert werden. Listing 12.16 zeigt das geänderte Skript *search_products_img.php*. In der Tabelle werden nun auch <*img*>-Elemente eingebunden, deren Quelle das PHP-Skript aus Listing 12.15 ist.

```
<?php
namespace net\xmp\phpbuch;
require './DatabaseConnection.php';
require './HTMLPage.php';
require './search_products_db.php';

$html = new HTMLPage('AdventureWorks : Produktsuche');
$form = <<<EOF
<form action="" method="get">
Produktname: <input name="produkt" />
<input type="submit" value="Suchen" />
</form><br />
EOF;
$html->addHTML($form);
if (isset($_GET['produkt'])) {
    $name = sanitizeName($_GET['produkt']);
    $products = getProductsByName($name);
    if ($products) {
        addImageColumn($products);
        $html->addTable($products, array(false, false, false, true));
    }
    else {
        $html->addElement('p', 'Keine Produkte gefunden.');
    }
}
$html->printPage();
exit;

function sanitizeName($txt) { ... } // gleich wie vorher

/**
 * Fügt Tabelle eine Spalte mit Produktphoto hinzu
 */
function addImageColumn(&$products)
{
    $products[0][] = 'Abbildung';
    for ($i=1; $i < count($products); $i++) {
        $products[$i][] = '<img src="get_image.php?id='
                          . $products[$i][0]
                          . '" alt="Produktabbildung" />';
    }
}
?>
```

Listing 12.16 *search_products_img.php* – Produktsuche mit Bildern

Abbildung 12.5 zeigt das Ergebnis: In der vierten Spalte werden nun die Produktabbildungen aus der Tabellenspalte *ThumbNailPhoto* angezeigt.

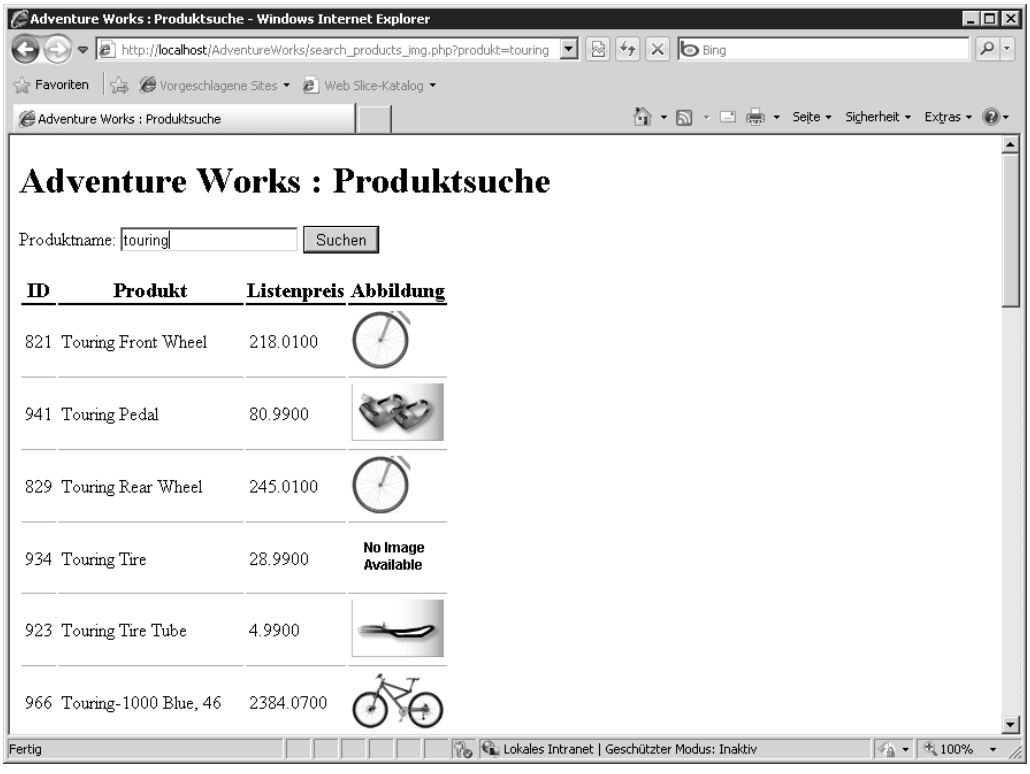

Abbildung 12.5 Produktsuche mit Abbildungen aus der Datenbank

Einfügen von Daten als Stream

Streams können nicht nur beim Auslesen von Daten aus der Datenbank verwendet werden, sondern auch beim Einfügen von Daten in die Datenbank. In diesem Fall wird bei der parametrisierten Abfrage ein geöffneter Stream übergeben.

Listing 12.17 zeigt am Beispiel des Einfügens eines Bildes in die Datenbank, wie Streams verwendet werden. In der *$params*-Variable wird das Handle für die geöffnete Datei übergeben. Da die Datenbankverbindung als UTF-8 geöffnet worden ist, empfiehlt es sich, bei *$params* auch den PHP-Datentyp (binär, keine Umcodierung notwendig) anzugeben. Damit wird eine korrekte Datenübertragung sichergestellt.

```
<?php
namespace net\xmp\phpbuch;
require './DatabaseConnection.php';
require './HTMLPage.php';

$html = new HTMLPage('AdventureWorks : Produktphoto hochladen');
$form = <<<'EOF'
<form action="" enctype="multipart/form-data" method="post">
ProduktID: <input name="productID" /><br />
Datei: <input type="file" name="productPhoto" /><br />
<input type="submit" value="Hochladen" />
</form><br />
EOF;
$html->addHTML($form);
```

```
if (isset($_POST['productID'])) {
    // Mit Datenbank verbinden
    $db = new DatabaseConnection();
    $db->connect();
    // Abfrage vorbereiten
    $query = 'UPDATE SalesLT.Product
              SET ThumbNailPhoto = ?, ThumbnailPhotoFileName = ?
              WHERE ProductID = ?';
    $id = (int) $_POST['productID'];
    $filename = filter_var($_FILES['productPhoto']['name'],
                           FILTER_SANITIZE_STRING, FILTER_FLAG_STRIP_LOW);
    // Datei öffnen
    $file = fopen($_FILES['productPhoto']['tmp_name'], 'rb');
    // Kodierung und Datentyp setzen
    $params = array(array($file, null, SQLSRV_PHPTYPE_STREAM(SQLSRV_ENC_BINARY),
                                       SQLSRV_SQLTYPE_VARBINARY('MAX')),
                    $filename, $id);
    $stmt = sqlsrv_query($db->handle, $query, $params);
    // Datei schließen
    fclose($file);
    if ($stmt === false) {
        $db->exitWithError('Hochladen von Photo fehlgeschlagen.');
    }
    if (sqlsrv_rows_affected($stmt) != 1) {
        $db->exitWithError('Falsche ID angegeben?');
    }
    $html->addElement('p', 'Hochladen erfolgreich.');
    // Datenbankverbindung schließen
    sqlsrv_free_stmt($stmt);
    $db->close();
}
$html->printPage();
?>
```

Listing 12.17 *upload_product_img.php* – Grafik per Stream in Datenbank einfügen

Zusammenfassung

Das Arbeiten mit der SQL Server-PHP-Erweiterung gestaltet sich für PHP-Programmierer einfach: Die Basisfunktionen zum Verbindungsauf- und -abbau, sowie zum Absenden von T-SQL-Anweisungen und Auslesen der Daten sind ähnlich zu anderen Datenbank-PHP-Erweiterungen gestaltet.

Die Möglichkeit, beim Auslesen von Ergebnissen gleich eigene Klassen zu instanziieren, vereinfacht das Mapping zwischen der Anwendungslogik mit PHP-Objekten und der Datenbank wesentlich. Auch die granulare Steuerung der zu verwendenden Datentypen auf PHP- und SQL Server-Seite ist ein hilfreiches Feature bei der Anwendungsentwicklung.

Die neue SQL-Server-Erweiterung *sqlsrv* ist moderner als *mssql* und wird aktiv gepflegt. Wie bei anderen datenbankspezifischen Treibern erschwert die Verwendung jedoch eine spätere Unterstützung anderer Datenbanksysteme. Mit Version 2.0 von *sqlsrv*, die derzeit (Juni 2010) als Tech Preview vorliegt, wird die Erweiterung auch PDO (*PHP Data Objects*) unterstützen, was eine transparente Ansteuerung von SQL Server und anderen Datenbanken ermöglicht.

Im folgenden Kapitel wird auf einige weiterführende Features von SQL Server eingegangen, unter anderem auf die Volltextsuche, gespeicherte Prozeduren und Trigger.