



PRIMEBASE TUTORIAL

July 2008

PrimeBase Systems GmbH

Max-Brauer-Allee 50 - D - 22765 Hamburg - Germany

www.primebase.com - e-mail: info@primebase.com

Einführung	5
Voraussetzungen	5
HTML, HTML-Formulare und Co.	5
TCP/IP für den Netzwerkzugriff.	5
Der Web-Browser, JavaScript und Proxy Settings.	5
Der Browser-Cache.	6
Verwendete Notationen	7
Passwörter und Eingaben	7
Screenshots und Referenzen	7
Tastaturkombinationen und Tastenbetätigungen	7
Erläuterungen und Zusammenfassungen in „Einschüben“	7
Ändern und Speichern von Dateien.	8
First Steps	8
Installation der PrimeBase-Software	8
Starten des integrierten Datenbank-Servers	8
Server als Hintergrundprozesse starten	9
Das Web-Interface des Database Servers	11
Anlegen einer neuen Datenbank	12
Tabellen und Spalten definieren und erzeugen	13
Die case-insensitive Suche oder: Wozu brauchen wir eine Domain?	17
Voreinstellung: Beachtung von Gross-/Kleinschreibung.	17
Unsere erste eigene Atomic Domain	18
Die Columns „name“, „address“, „country“ und „discount“	18
Hoppla, doch was falsch? Einfach Korrigieren!	19
Was haben wir bisher gemacht?	20
Was fehlt uns jetzt noch?	20
Jetzt aber: Die Änderungen des Schemas auf unsere Datenbank übertragen.	21
Second steps	22
Der eingebaute HTTP-Server und das Verzeichnis „docs“	22
Den Application Server starten	23
Der Default Benutzer „admin“ und der PrimeBase Datenbank Server	24
Anwendungen, Module und die „PrimeBase Enterprise Objects“.	24
Die Standardanwendung „ModuleApplication“.	25
Warum Module?	25
Unser Modul - Wie macht man das?	25
Wohin mit den HTML-Dateien, Bildern, etc.?	25
Wohin mit dem Modulverzeichnis?	26
Das Verzeichnis „originals“	26
Unser Modul anlegen	26
Deklaration unseres Moduls	26
Die „LiveTag Markup Language“ ist HTML.	27
Die Anwendungsdefinition	28
Die Anwendungsdefinition in den Application Server laden.	29
Der Systembereich des Application Servers.	29
Die erste HTML-Datei unseres Moduls.	30
Arbeitsteilung: Der Web-Designer macht das Layout.	30
Das HTML-Attribut „name“.	31
Die Einstiegsseite unseres Moduls.	31
Anmerkungen des Web-Designers in unseren HTML-Dateien.	32
Die andere HTML-Seite unseres Moduls.	32
Verknüpfung von HTML und Datenbanken	32
Objekte zur Verknüpfung von Datenbank und HTML-Dateien erstellen.	33
Die Objektdeklaration.	33
Warum haben wir ein Objekt definiert?	33
Die declareObject-Methode.	33

Der erste Parameter: Objekttyp.	34
Der zweite Parameter: Name unseres Objektes.	34
Der dritte Parameter: Vollqualifizierter Name unserer Tabelle.	34
Der vierte Parameter: Name des Primary Keys unserer Tabelle.	34
„.htm“ und „.html“ werden zu „.lml“.	35
Die weiteren Merkmale unseres „tutdemo“-Objekts: Attribute.	36
Die declareAttribute()-Methode.	36
Der erste Parameter: Objekttyp.	36
Der zweite Parameter: Name der Spalte.	36
Der letzte grosse Schritt: Anpassung der HTML-Seiten	37
Die Standard Buttons.	38
Jetzt aber: Die Anwendungsdefinition neu laden und unser Modul anschauen.	39
Was tun, wenn ein Fehler im Browser angezeigt wird?	39
Wenn wir eine Fehlermeldung bekommen ...	40
Einfach nur verschrieben.	40
HTML-Attribut vs. PrimeBase™ Enterprise Objects Attribut	41
Das HTML-Attribut „name“ vergessen.	41
Es klappt immer noch nicht?	41
Die ersten Daten eingeben.	42
Hat das Hinzufügen unseres ersten Datensatzes geklappt?	42
Ein paar Standard-Datensätze hinzufügen.	43
Die Suchfunktion nutzen.	43
Hat die Suche geklappt?	44
Warum aber werden die gefundenen Datensätze nicht angezeigt?	44
Der „Leer-Datensatz“ (empty-record) in PrimeBase Enterprise Objects.	44
Die restlichen Buttons testen.	45
Zusammenfassung: HTML, PrimeBase Enterprise Objects und Datenbanken.	46
Erstellen von Events (Ereignissen)	47
Das „Leer-Datensatz“-Problem.	47
Unser erster Event.	47
Noch ein Event: Ein Eingabefeld als Muss-Feld deklarieren.	48
Anzeigen von Muss-Feldern in der HTML-Seite.	49
Der letzte Event: Auf eine andere HTML-Seite wechseln.	50
Das <output>-LiveTag	52
Die Primary Key Column als Kunden ID-Nummer verwenden.	53
Ausgeben von Columns in der HTML-Seite.	53
Endspurt: Eine Liste der gefundenen Datensätze in „custfound.lml“	54
Die letzten Kommentare ersetzen.	54
Das <container>-LiveTag als Schleife verwenden.	56
Die mit „loopall“ Containern begonnene Schleife mit „nextrow“ Containern durchlaufen.	57
Ausgewählter Datensatz vs. aktueller Datensatz	58
Die letzte Änderung: Die HTML-Links anpassen.	59
Der grosse Bruder von „Tutorial Demo“: Das Modul „PBE Demo“.	60

1. EINFÜHRUNG

Dieses Tutorial zeigt Ihnen, wie Sie mit dem PrimeBase-Entwicklungssystem webbasierte Anwendungen mit Datenbankbindung erstellen können. Außerdem vermittelt es Ihnen, wie Sie in wenigen Schritten eine Datenbank und die dazu gehörenden Tabellen in dem Datenbank-Server erzeugen und verwalten, der im PrimeBase-Entwicklungssystem integriert ist.

Dieses Tutorial ist in zwei Teile untergliedert. Im ersten Kapitel „First Steps“ wird beschrieben, wie Sie eine Datenbank inklusive Tabellen und definierten Spalten im Datenbank-Server anlegen und verwalten.

Im zweiten Kapitel „Second Steps“ wird eine webbasierte Anwendung erstellt, die an die zuvor erstellte Datenbank angebunden wird. Nach Bearbeitung des zweiten Kapitels werden Sie in der Lage sein, eigene Webanwendungen mit Datenbankzugriff zu erstellen.

Voraussetzungen

HTML, HTML-Formulare und Co.

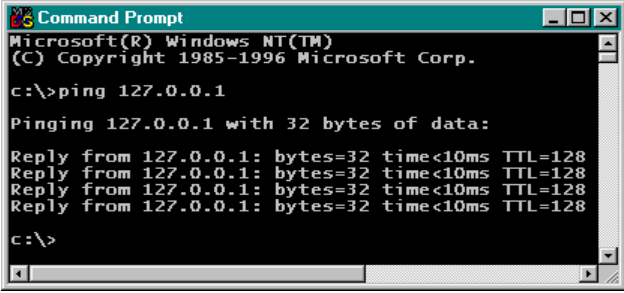
Beim Leser werden Kenntnisse von HTML, insbesondere von HTML-Formularen, vorausgesetzt.

TCP/IP für den Netzwerkzugriff.

Dieses Tutorial setzt auch voraus, dass auf dem verwendeten Computer TCP/IP korrekt installiert und funktionsfähig ist. Sie können dies unter Windows und den Unix-basierten Systemen mit dem Kommando „ping“ testen. Starten Sie hierzu die „Eingabeaufforderung“ unter Windows, bzw. öffnen Sie ein neues Shell-Fenster, geben Sie das folgende Kommando ein und bestätigen Sie die Eingabe mit der ENTER- bzw. der RETURN-Taste:

- `ping 127.0.0.1`

Ist TCP/IP korrekt installiert, sollte diese oder eine ähnliche Meldung erscheinen (ein Abbruch ist über STRG-C möglich):



```
Microsoft(R) Windows NT(TM)
(C) Copyright 1985-1996 Microsoft Corp.

c:\>ping 127.0.0.1

Pinging 127.0.0.1 with 32 bytes of data:

Reply from 127.0.0.1: bytes=32 time<10ms TTL=128
Reply from 127.0.0.1: bytes=32 time<10ms TTL=128
Reply from 127.0.0.1: bytes=32 time<10ms TTL=128
Reply from 127.0.0.1: bytes=32 time<10ms TTL=128

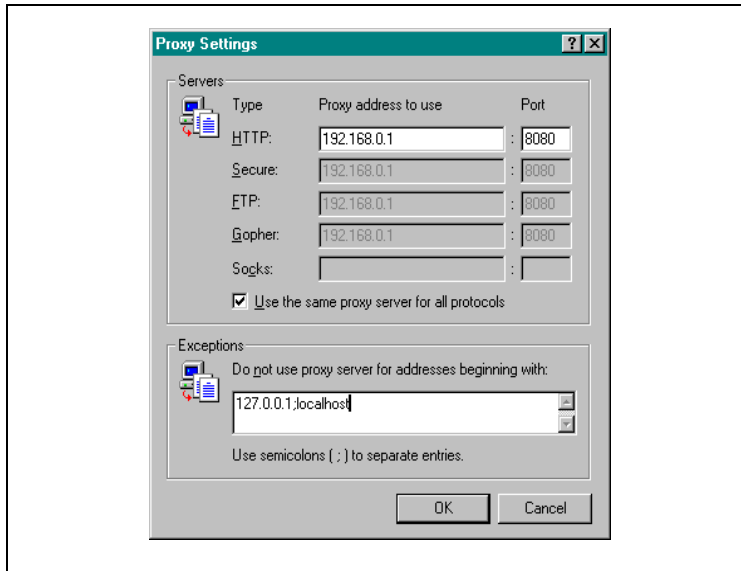
c:\>
```

SCREENSHOT 1: DAS TCP/IP PROTOKOLL IST KORREKT INSTALLIERT.

Der Web-Browser, JavaScript und Proxy Settings.

Für dieses Tutorial wird ein Web-Browser vorausgesetzt, der JavaScript unterstützt und bei dem diese Unterstützung aktiviert ist.

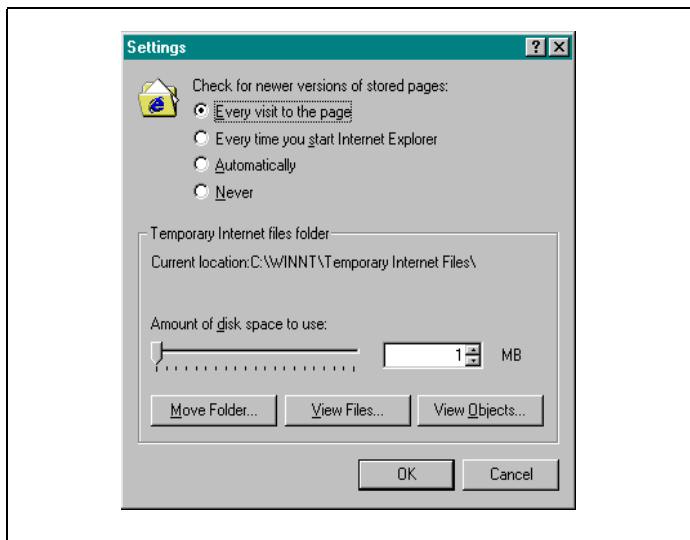
Außerdem muss der Browser den Zugriff auf lokal installierte Serveranwendungen zulassen. Deshalb muss die eventuell im Web-Browser eingestellte Nutzung eines Proxy-Servers für die Netzwerkadresse 127.0.0.1 deaktiviert werden:



SCREENSHOT 2: DIE BENUTZUNG EINES PROXY-SERVERS FÜR LOKALE SERVERANWENDUNGEN DEAKTIVIEREN.

Der Browser-Cache.

Der Browser-Cache sollte so weitgehend wie möglich deaktiviert werden, indem man den Web-Browser so konfiguriert, dass Seiteninhalte bei jedem Besuch aktualisiert werden:



SCREENSHOT 3: DAS CACHING-VERHALTEN DES BROWSERS KONFIGURIEREN, UM PROBLEME BEIM CACHING ZU VERMEIDEN.

Verwendete Notationen

Passwörter und Eingaben.

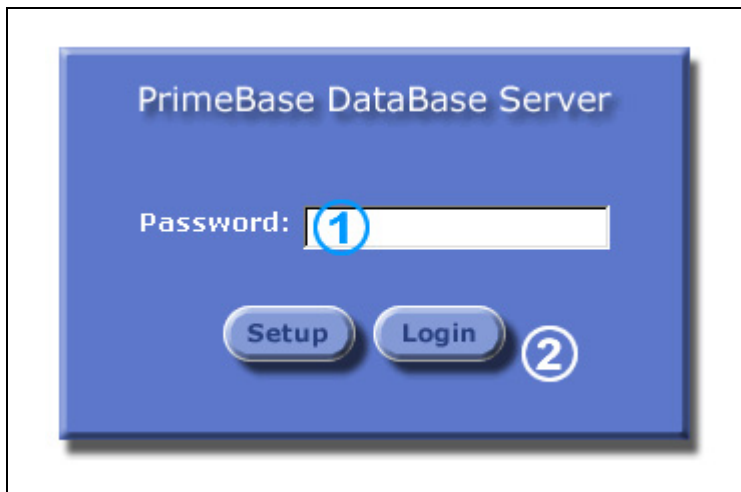
Passwörter und Eingaben sind in diesem Tutorial in Anführungszeichen („Passwort“ oder „Eingabe“) dargestellt. Diese Anführungszeichen sind bei der Eingabe wegzulassen, falls keine weiteren Angaben diesbezüglich im entsprechenden Abschnitt gemacht werden.

Screenshots und Referenzen.

Die Screenshots in diesem Tutorial sind durchnummeriert.

Einzelne Bereiche der Screenshots sind mit Zahlen beschriftet. Auf diese Bereiche wird mit einer Angabe wie „1.1“ oder „9.3“ verwiesen, wobei „9.3“ z. B. auf Screenshot 9 und den dort mit „3“ markierten Bereich verweist.

Dazu ein kleines Beispiel:



SCREENSHOT 4: LOGIN

- Geben Sie „primebase“ ein (4.1).
- Drücken Sie auf „Login ...“ (4.2).

Tastaturkombinationen und Tastenbetätigungen.

Tastaturkombinationen und die Betätigung einzelner Tasten werden durch Großbuchstagen wiedergegeben:

- STRG-R
- STRG-C
- ENTER
- RETURN

Erläuterungen und Zusammenfassungen in „Einschüben“.

In diesem Tutorial werden „Einschübe“ verwendet, wie man sie z. B. aus Zeitschriften und Lehrbüchern kennt. Entsprechend dem Vorbild enthalten diese Einschübe Zusammenfassungen und teils grundlegende teils aber auch weitergehende Erläuterungen zum jeweiligen Abschnitt.

Stoßen Sie während der Lektüre dieses Tutorials auf einen Einschub, so ist es sinnvoll, diesen direkt zusammen mit dem umgebenden Text zu lesen, um den weiteren Ausführungen folgen zu können.

Und so sieht ein Einschub in diesem Tutorial aus:

Einschub

Dies ist ein Einschub, der nur zu Demonstrationszwecken dient.

Ändern und Speichern von Dateien.

Es wird beim Ändern von Dateien nicht gesondert darauf hingewiesen, dass die Änderungen auch immer gespeichert werden müssen.

2. FIRST STEPS

Installation der PrimeBase-Software

Als erstes muss das PrimeBase-Entwicklungssystem installiert werden. Da PrimeBase plattformunabhängig ist, kann die Installation der Software je nach verwendetem Betriebssystem leicht voneinander abweichen.

- **Installieren Sie die Dateien entsprechend der Anleitung in der Datei „readme.txt“ auf der PrimeBase-CD.**

Nun befinden sich auf Ihrem Rechner die beiden Verzeichnisse „pbds“ und „pbas“ (bei älteren Versionen des PrimeBase-Entwicklungssystems können die Verzeichnisnamen leicht abweichen).

Im Verzeichnis „pbds“ befindet sich der in das PrimeBase-Entwicklungssystem integrierte Datenbank-Server „PrimeBase Database Server“ (im Folgenden „Database Server“ genannt) und im Verzeichnis „pbas“ der sogenannte „PrimeBase Application Server“ (im Folgenden „Application Server“ genannt), der für die Erstellung und Verwendung von webbasierten Anwendungen mit PrimeBase eine Schlüsselkomponente darstellt.

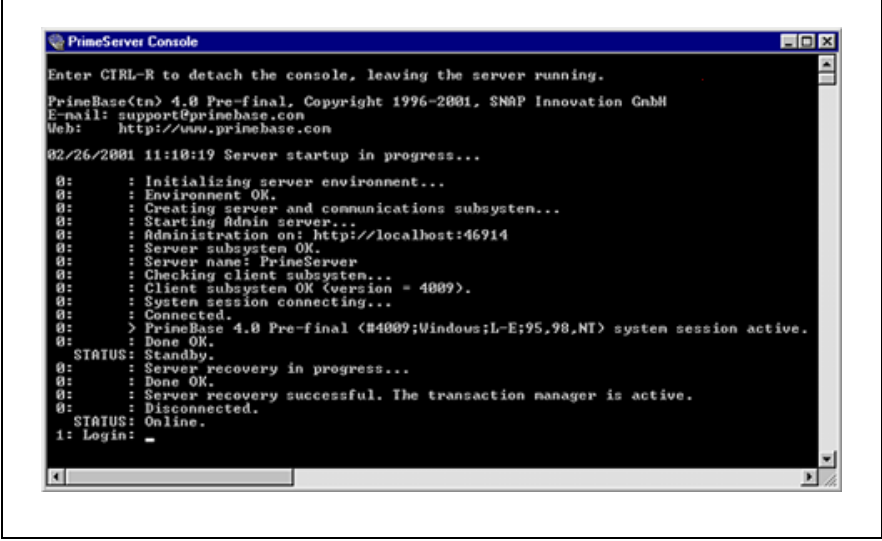
Nähere Informationen zur Installation auf anderen Plattformen (MacOS, MacOS X, Solaris, Linux, AIX) entnehmen Sie bitte der Dokumentation, die der jeweiligen Version von PrimeBase beigelegt ist.

Starten des integrierten Datenbank-Servers

Der Start des Database Servers ist denkbar einfach:

- **Führen Sie einfach die Datei „pbds.exe“ aus**

Anschließend startet der Database Server und ein Programmfenster öffnet sich:



```
PrimeServer Console
Enter CTRL-R to detach the console, leaving the server running.
PrimeBase(tm) 4.0 Pre-final. Copyright 1996-2001, SNAP Innovation GmbH
E-mail: support@primebase.com
Web: http://www.primbase.com

02/26/2001 11:10:19 Server startup in progress...

0: : Initializing server environment...
0: : Environment OK.
0: : Creating server and communications subsystem...
0: : Starting Admin server...
0: : Administration on: http://localhost:46914
0: : Server subsystem OK.
0: : Server name: PrimeServer
0: : Checking client subsystem...
0: : Client subsystem OK (version = 4009).
0: : System session connecting...
0: : Connected.
0: : PrimeBase 4.0 Pre-final (M4009;Windows;L-E;95,98,NT) system session active.
0: : Done OK.
STATUS: Standby.
0: : Server recovery in progress...
0: : Done OK.
0: : Server recovery successful. The transaction manager is active.
0: : Disconnected.
STATUS: Online.
1: Login: _
```

SCREENSHOT 5: DIE CONSOLE DES DATABASE SERVERS

Das Programmfenster (die so genannte „Console“) zeigt während des Starts des Database Servers dessen Status an, was auch für den Application Server gilt, dazu kommen wir aber erst später. Diese Console kann zur interaktiven Eingabe von Kommandos und auch zum Stoppen des Database Servers benutzt werden. In diesem Tutorial wird aber nur von letzterer Möglichkeit Gebrauch gemacht.

Der Default User „administrator“ im Database Server

Im Database Server gibt es nur einen einzigen Benutzeraccount (Useraccount), mit dessen Namen und Passwort man sich am Datenbank-Server in jedem Fall anmelden muss, um arbeiten zu können.

Der Name dieses Users lautet „administrator“, für den in der Voreinstellung kein Passwort vergeben ist.

Der User „administrator“ hat im Database Server die höchsten Rechte und ist somit vergleichbar dem User „root“ in Unix-basierten Systemen oder dem User „administrator“ bei Microsoft Windows NT.

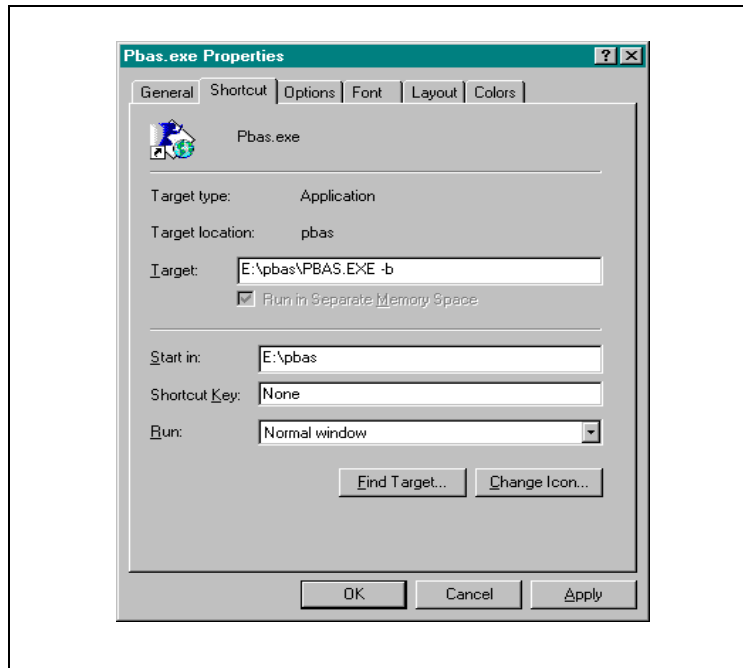
Da ein Datenbank-Server in der Regel ein Programm ist, welches seine Arbeit im Hintergrund verrichtet, wird zum Arbeiten weder ein Programmfenster noch eine Console benötigt. PrimeBase ist zwar so eingestellt, dass eine Console geöffnet wird, aber diese kann jederzeit durch die Tastenkombination STRG-R geschlossen werden.

Beachten Sie bitte, dass diese Tastenkombination nur die Console, nicht aber das eigentliche Programm schließt. Zum Beenden des eigentlichen Programms (Database Server oder Application Server) geben Sie auf der jeweiligen Console „halt“ ein und bestätigen Sie mit ENTER oder RETURN.

Server als Hintergrundprozesse starten.

Sowohl der Database Server als auch der Application Server können als so genannte Hintergrundprozesse – also ohne Programmfenster (Console) – gestartet werden. Dazu müssen Sie die beiden Programme aber explizit mit dem Parameter „-b“ starten, also mit Verweis auf das jeweilige Programmverzeichnis komplett „pbas -b“ oder

„pbds -b“ („server -b“ bei Version 3.5 oder früher) eingeben.
 Unter Windows ist es auch möglich, Shortcuts auf „pbds.exe“ und „pbas.exe“ anzulegen, z. B. zur schnellen Erreichbarkeit auf dem Desktop. In diesen Shortcuts wiederum kann man dann den Parameter in den „Eigenschaften“ (rechter Mausklick auf das jeweilige Shortcut Symbol) im Bereich „Shortcut“ in das Feld „Target“ hinter dem Programmnamen eintragen:



SCREENSHOT 6: PRIMEBASE™ ÜBER EINEN WINDOWS SHORTCUT ALS HINTERGRUNDPROZESS STARTEN

Sollten Sie den Database Server oder den Application Server ohne Fenster gestartet oder das entsprechende Fenster geschlossen haben, können Sie selbiges jederzeit wieder öffnen, indem Sie das Programm „pbcon.exe“ aus dem Verzeichnis „pbds“ bzw. dem Verzeichnis „pbas“ starten und im anschließend erscheinenden Menü die Auswahl treffen: „1“ oder „2“, abhängig davon, für welchen Server sie das Fenster wieder öffnen wollen.

Wozu braucht man die Console-Fenster des Database und des Application Servers?

Die Console des Database Servers

Erfahrenen Anwendern des PrimeBase-Entwicklungssystems dient die Console des Database Servers als Hilfe bei der Entwicklung, da über diese einzelne Programmbefehle und Datenbankabfragen ohne großen Aufwand getestet werden können.

Im Rahmen dieses Tutorials werden wir kaum Bedarf für die Console des Database Servers haben. Daher ist es in der Regel sinnvoll, diese Console mit STRG-R zu schließen, damit sie uns nicht unnötig verwirrt.

Die Console des Application Servers

Die Console des Application Servers ist hingegen während der Entwicklungsphase von großer Bedeutung, da hier diverse Informationen dargestellt werden, die bei der Fehlersuche und Problemanalyse sehr hilfreich sein können. Daher sollte diese Console – im Gegensatz zur Console des Database Servers – geöffnet bleiben.

Das Web-Interface des Database Servers

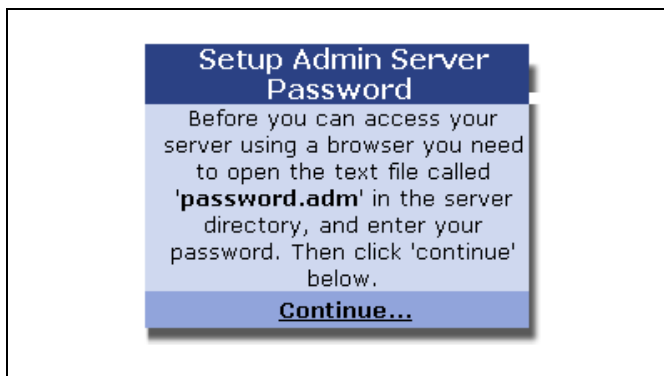
Der Database Server hat ein webbasiertes Interface, den so genannten „Administration Server“ – oder kurz „Admin-Server“. Der Admin-Server kann mit einem beliebigen Javascript-fähigen Browser benutzt werden, um die folgenden Tätigkeiten komfortabel durchzuführen:

- Konfigurieren des Database Servers.
- Anlegen, kopieren, reorganisieren und löschen von Datenbanken innerhalb des Database Servers.
- Anlegen, ändern und löschen von Tabellen („Tables“) innerhalb von Datenbanken und Spalten („Columns“) innerhalb von Tabellen.

Zum Verwenden des Admin-Servers geben Sie folgende URL in die Adresszeile eines auf dem lokalen Rechner gestarteten, Javascript-fähigen Web-Browsers ein:

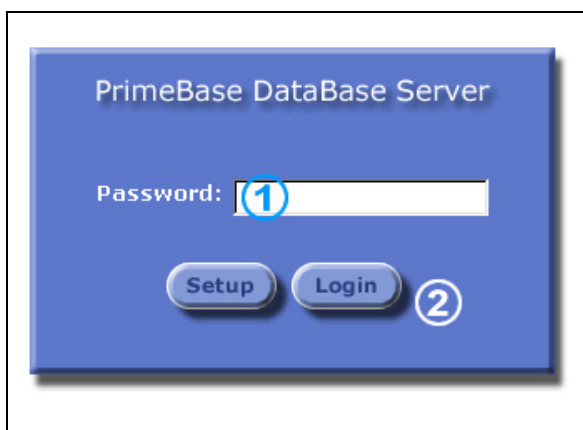
- <http://127.0.0.1:46914>

Wenn Sie sich das erste Mal einloggen, werden Sie nun aufgefordert, mit Hilfe eines Texteditors ein Passwort Ihrer Wahl in die Datei „password.adm“ (liegt im Verzeichnis ihres Database Servers) einzutragen:



SCREENSHOT 7: ERSTER LOGIN

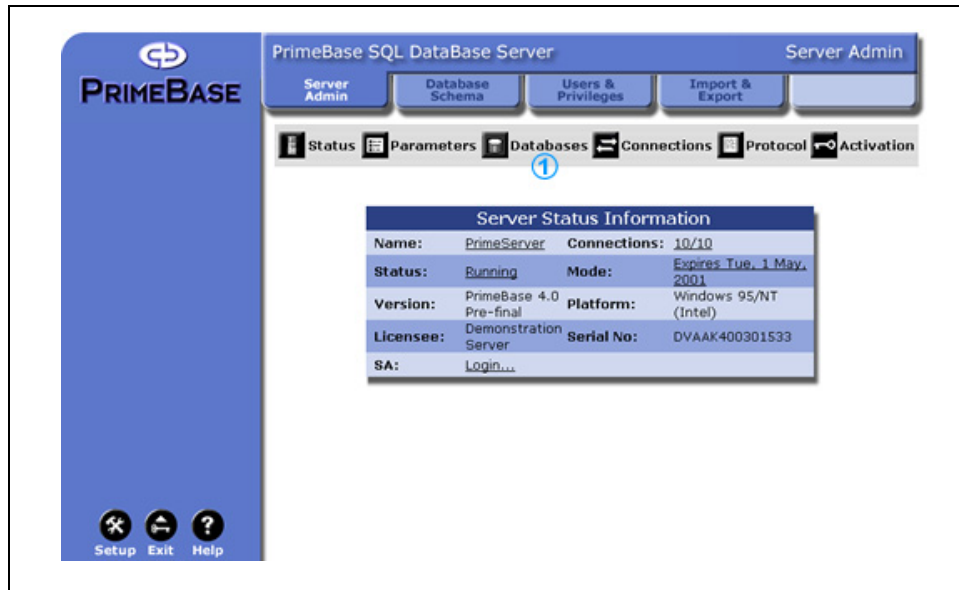
Nun können Sie sich mit ihrem Passwort einloggen:



SCREENSHOT 8: BASIC LOGIN

- Geben Sie Ihr Passwort ein (8.1).
- Drücken Sie auf „Login ...“ (8.2).

Nun erscheint das folgende Hauptmenü des Admin-Servers im Browser:



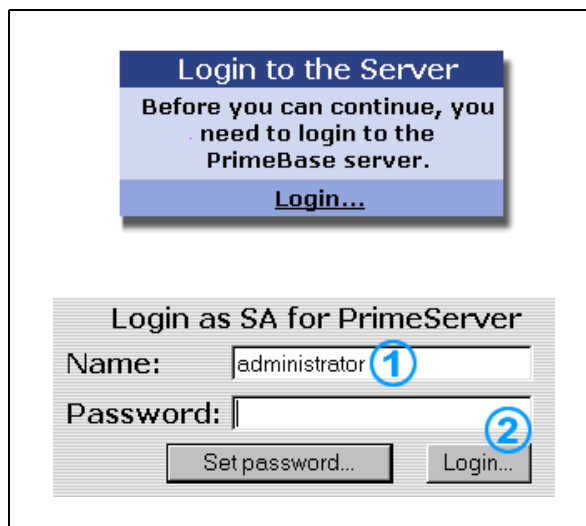
SCREENSHOT 9: DATABASE SERVER ADMINISTRATION

Anlegen einer neuen Datenbank

In diesem Schritt werden wir eine neue Datenbank anlegen, die im weiteren Verlauf des Tutorials verwendet wird, um Information über Kunden aufzunehmen. Um die Datenbank anzulegen, müssen Sie die folgenden Schritte durchführen:

- Drücken Sie auf „Databases“ (9.1).

Daraufhin erscheint der folgende Login-Dialog im Browser:



SCREENSHOT 10: USER ANMELDUNG

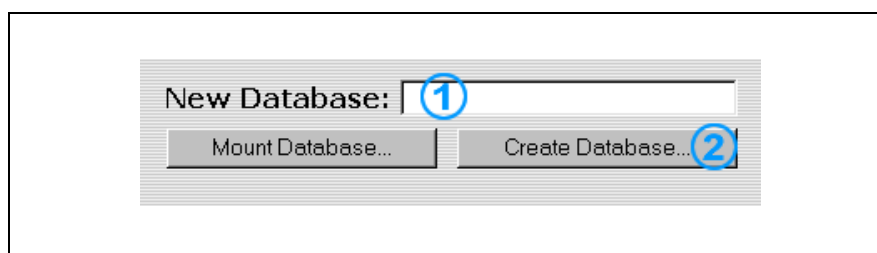
- Geben Sie „administrator“ ein (10.1).
- Drücken Sie auf „Login...“ (10.2).

Nun werden die vorhandenen Datenbanken aufgelistet, wobei die Anzeige aber von Screenshot 11 leicht abweichen kann:

ID	Name	CreatorID	CreationTime	Comments
1	Master	1	Wed, 25 Sep, 1996, 10:30 AM	The Master database all
2	Model	1	Wed, 25 Sep, 1996, 10:30 AM	The Model database is c
	New...			

SCREENSHOT 11: VORHANDENE DATENBANKEN

- Drücken Sie auf „New...“ (11.1).



SCREENSHOT 12: NEUE DATENBANK ANLEGEN

- Geben Sie „tutdemo“ ein (12.1).
- Drücken Sie auf „Create Database...“ (12.2).

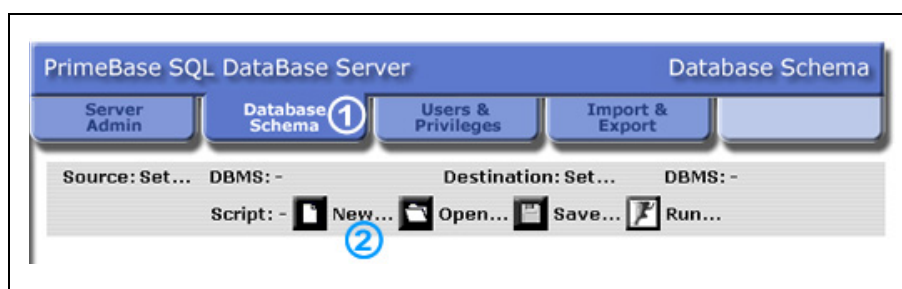
Sie haben jetzt eine Datenbank mit dem Namen „tutdemo“ angelegt. Als nächstes müssen in dieser neuen Datenbank Tabellen angelegt werden, damit überhaupt Daten in die Datenbank eingefügt werden können.

Bei den Tabellen und Spalten („Columns“) innerhalb einer Datenbank spricht man im Allgemeinen vom so genannten Datenbankschema. Und ein solches – also eben Tabellen und entsprechende Spalten – wollen wir im nächsten Schritt definieren und anschließend auf unsere neue, aber noch gänzlich leere Datenbank übertragen.

Tabellen und Spalten definieren und erzeugen

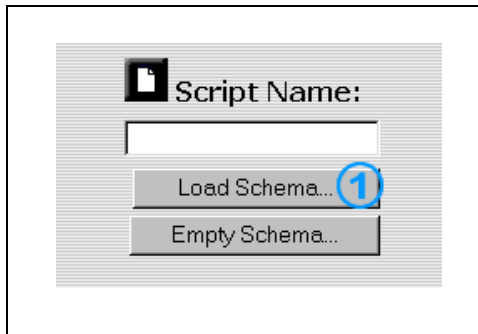
Um Tabellen und Spalten in einer bereits vorhandenen Datenbank anzulegen oder vorhandene Tabellen und Spalten zu ändern, muss als erstes das sogenannte „Database Schema“ der entsprechenden Datenbank geladen werden:

- Drücken Sie auf „Database Schema“ (13.1).
- Drücken Sie auf „New ...“ (13.2).



SCREENSHOT 13: ADMINISTRATIONSFENSTER

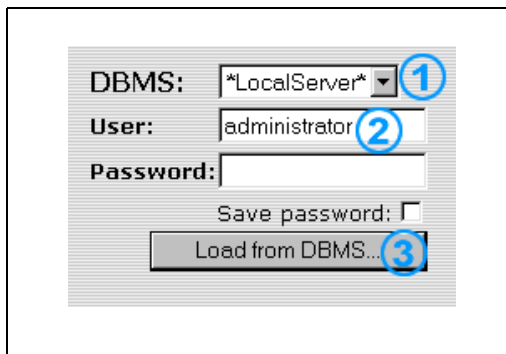
Rechts oben erscheint nun das Folgende:



SCREENSHOT 14: LADEN EINES DATENBANK SCHEMAS

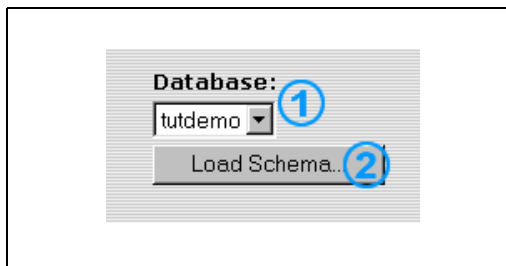
- Drücken Sie auf „Load Schema...“ (14.1).

Nun erscheint ein Dialog, der es – z. B. für die fortgeschrittene Systemadministration – erlaubt, einen anderen PrimeBase Datenbank-Server als den lokal installierten für die weiteren Aktivitäten auszuwählen. Für dieses Tutorial begnügen wir uns aber damit, die Voreinstellung „*LocalServer*“ (Screenshot 15.1) beizubehalten.



SCREENSHOT 15: LOAD FROM DBMS

- Geben Sie „administrator“ als Usernamen ein (15.2).
- Lassen Sie das Feld „Password“ frei.
- Drücken Sie auf „Load from DBMS...“ (15.3).

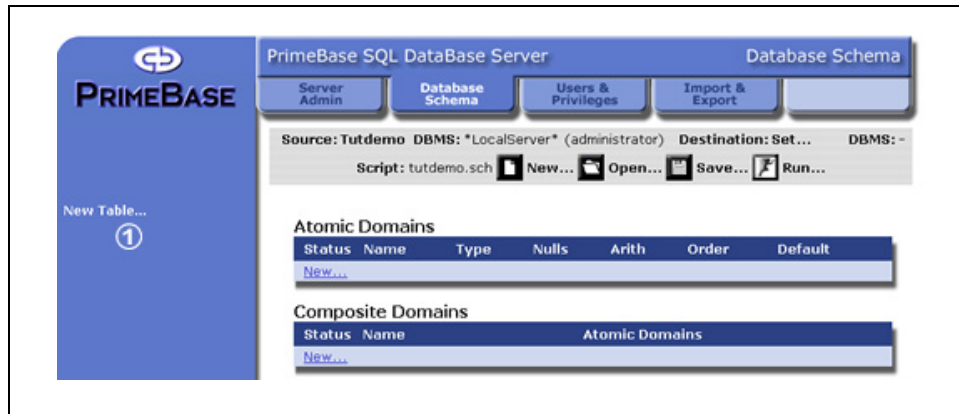


SCREENSHOT 16: LADEN DES SCHEMAS DER DATENBANK „TUT_DEMO“

- Wählen Sie „tutdemo“ aus der Dropdown-Liste aus (16.1).
- Drücken Sie auf „Load Schema...“ (16.2).

Nun haben wir das Schema der zuvor erzeugten Datenbank „tutdemo“ geladen. Wie Sie sehen können, enthält „tutdemo“ bisher weder Tabellen noch selbstdefinierte Datentypen, die ansonsten unter 17.1 oder 17.2 aufgelistet werden würden.

Für die später in „Second Steps“ folgende Beispielanwendung wird nur eine einzelne Tabelle mit dem Namen „customers“ benötigt.

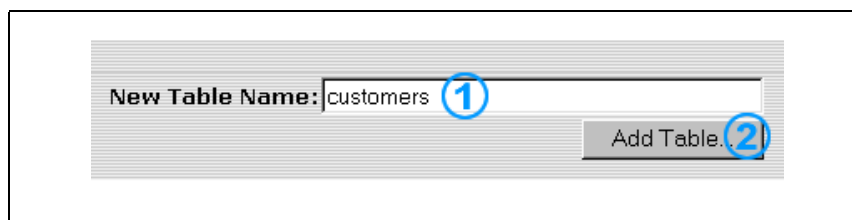


SCREENSHOT 17: INHALT DER DATENBANK „TUTDEMO“

Die Länge von Namen und Bezeichnern in PrimeBase

Die Länge von Variablen-, Tabellen-, Spaltennamen usw. ist in PrimeBase generell auf 32 Zeichen begrenzt. Bedenken Sie dies bitte bei der Wahl von Namen für Datenbanken, Tabellen und Spalten.

Zu der einen entscheidenden Ausnahme von dieser Regel werden wir in den „Second Steps“ kommen.



SCREENSHOT 18: EINGABE DES NAMENS DER NEUEN TABELLE

- Drücken Sie auf „New Table...“ (17.1).
- Geben Sie den Namen „customers“ ein (18.1).
- Drücken Sie auf „Add Table...“ (18.2).

Wir haben nun eine leere Tabelle definiert, die unter ihrem Titel „customers“ im Bereich 17.1 angezeigt wird. Zusätzlich müssen wir noch folgende Columns (Spalten) definieren:

- num (der „Primary Key“ unserer neuen Tabelle)
- name
- address
- country
- discount

Die ID-Column und: Was ist eigentlich ein Primary Key?

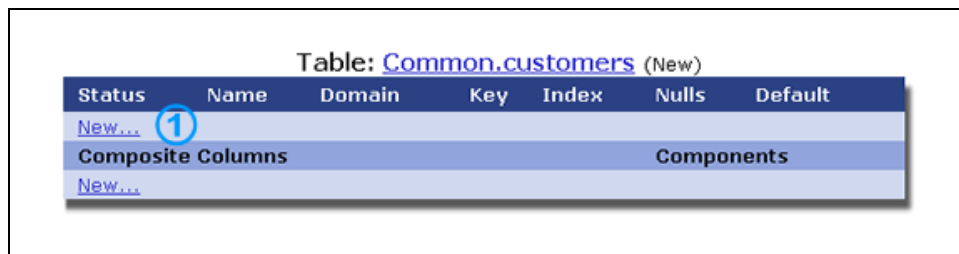
Jeder Datensatz in einer Tabelle muss ein eindeutiges Merkmal haben, welches ihn von den anderen Datensätzen in der gleichen Tabelle unterscheidet. Dieses Merkmal wird als

„Primary Key“ bezeichnet. Ohne eine als Primary Key definierte Column lässt der Database Server das Hinzufügen von Daten nicht zu.

Das bedeutet für uns, dass wir uns einfach merken, dass jede von uns angelegte Tabelle immer eine Column mit folgenden Einstellungen haben muss, wobei der Name der Column zwar frei wählbar ist, aber aus Konsistenzgründen überall gleich lauten sollte:

Name „num“
 Type „INTEGER“
 Key „Primary“
 Default „Counter“
 Index ausgewählt
 Nulls nicht ausgewählt

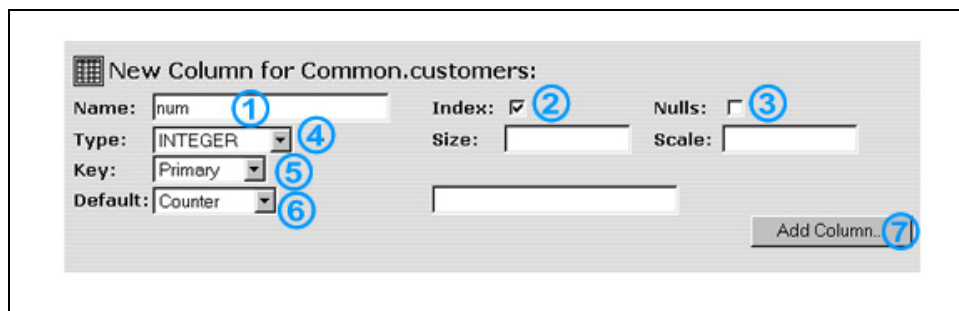
Im mittleren Bildschirmbereich werden die in der Tabelle „customers“ bisher definierten Columns angezeigt (vgl. Screenshot 19) – im Moment eben noch keine, was wir aber gleich ändern werden.



SCREENSHOT 19: ÜBERSICHT ÜBER DIE BISHERIGEN COLUMNS DER TABELLE „CUSTOMERS“

- Drücken Sie auf „New...“ (19.1).

Nun wird der folgende Dialog im unteren Bildschirmbereich angezeigt:



SCREENSHOT 20: NEUE COLUMN IN TABELLE „CUSTOMERS“ ANLEGEN

Nehmen Sie für diese erste Column nun folgende Einstellungen vor:

- Geben Sie „num“ als Namen ein (20.1).
- Aktivieren Sie die Checkbox „Index“ (20.2).
- Deaktivieren Sie die Checkbox „Nulls“ (20.3).
- Wählen Sie „INTEGER“ als Type aus (20.4).
- Wählen Sie „Primary“ als Key aus (20.5).
- Wählen Sie „Counter“ als Default aus (20.6).
- Drücken Sie auf „Add Column“ (20.7).

Die Column „num“ wurde nun in die Tabelle eingefügt.

Die case-insensitive Suche oder: Wozu brauchen wir eine Domain?

Als nächstes wäre die Column „name“ einzufügen. Diese Column soll, im Gegensatz zur Column „num“ (die wir als vom Typ „INTEGER“ – also Zahlenwerte ohne Kommastellen – definiert haben), Textdaten aufnehmen. Also würden hier die Datentypen „CHAR“ oder „VARCHAR“ zum Einsatz kommen können.

In diesem Tutorial beschränken wir uns auf die Verwendung des VARCHAR-Datentyps. Bei Textdaten gibt es allerdings doch ein paar Besonderheiten zu beachten und die wichtigste Besonderheit soll jetzt hier erläutert werden, um zeitaufwendigen Umstellungen in diesem Bereich vorzubeugen.

Voreinstellung: Beachtung von Gross-/Kleinschreibung.

Textdaten können prinzipiell auf zwei Arten durchsucht werden:

- unter Beachtung der Groß-/Kleinschreibung (case-sensitive)
- ohne Beachtung der Groß-/Kleinschreibung (case-insensitive)

Der Database Server führt eine Suche über Textdaten grundsätzlich unter Beachtung der Gross-/Kleinschreibung durch.

Das hat zur Folge, dass eine Suche nach dem Wohnort einer Person bei einem Suchwort wie „Berlin“ zwar alle Datensätze (Records) findet, deren Inhalt exakt „Berlin“ lautet, aber nicht Datensätze, in denen „berlin“, „bErLin“, „beRlin“ usw. steht. Somit kann bereits ein kleiner Tippfehler dazu führen, dass man Datensätze in der Datenbank nicht findet!

Daher ist im Regelfall eine Suche ohne Beachtung der Gross-/Kleinschreibung wünschenswert. Um dies mit PrimeBase zu realisieren, muss ein eigener Datentyp – eine so genannte Domain – angelegt werden.

Selbstdefinierte Datentypen: Domains

Der Database Server erlaubt die Definition eigener Datentypen. Diese Datentypen werden fachspezifisch „Domains“ genannt. Es gibt zwei Arten von Domains: Atomic Domains und Composite Domains.

Wirklich von Belang sind für uns in diesem Tutorial aber nur die Atomic Domains, die immer einen der vorhandenen Basisdatentypen (INTEGER, CHAR, VARCHAR, DECIMAL, MONEY usw.) als Grundlage haben.

Im Grunde sind Atomic Domains sehr einfach zu verstehen, wenn man sich denn erst einmal an den Gedanken gewöhnt hat, seinen Tabellen Spalten hinzuzufügen und für diese wiederum einen der vorhandenen Datentypen auszuwählen:

Atomic Domains sind Basisdatentypen mit vom Entwickler gewählten Namen und Parametern.

Wenn man sich zum Beispiel den Datentyp „VARCHAR“ betrachtet, so fällt auf, dass man beim Anlegen von Columns mit diesem Datentyp auch immer eine „Size“, eine Größe, mit angeben muss. Stellen Sie sich vor, Sie müssten 1000 Columns vom Typ VARCHAR mit der Größe 200 kreieren ..., das würde sehr viel Zeit für die Eingabe der Größe in Anspruch nehmen. Einfacher wäre es, einmal einen eigenen Datentyp zu definieren, der auf VARCHAR basiert, bei dem eine Größe von 200 voreingestellt ist und den man dann für alle diese Columns verwenden kann.

Oder stellen Sie sich vor, Sie müssten besagte 1000 Columns einer vorhandenen Datenbank auf eine Größe von 256 Zeichen umstellen – wenn Sie alle 1000 Columns auf einer selbstdefinierten Domain basieren lassen, ist dies sehr leicht und schnell zu erledigen, indem Sie einfach die entsprechende Atomic Domain ändern.

All dies nur am Rande, wirklich interessant ist für uns im Moment nur die Möglichkeit, bei einer Domain eine „Order“, also eine Sortierreihenfolge, anzugeben. Für unser Ziel, die case-insensitive Suche über unsere Text-Columns, müssen wir eine entsprechende Atomic Domain anlegen, für die als Order „Case Insensitive“ eingestellt ist.

Um uns vor die Qual der Wahl zu stellen, gibt es auch noch die Order-Option „Case

Ins./Ignore Dia., die nicht nur dafür sorgt, dass bei Sortierung und Suche die Groß-/Kleinschreibung ignoriert wird, sondern zusätzlich auch die „Diacritical Marks“ – die Sonderzeichen – ignoriert werden.

Unsere erste eigene Atomic Domain.

Nachdem wir durch diesen langen Einschub nun hoffentlich ein bisschen Verständnis für den Einsatzzweck von Domains gewonnen haben, werden wir jetzt zur Tat schreiten und eine eigene Domain anlegen, die wir anschließend für unsere weiteren Columns verwenden werden.

- Drücken Sie unter „Atomic Domains“ auf „New ...“ (21.1).

Im unteren Bildschirmbereich erscheint das Fenster zum Erstellen einer neuen Domain („New Domains“).

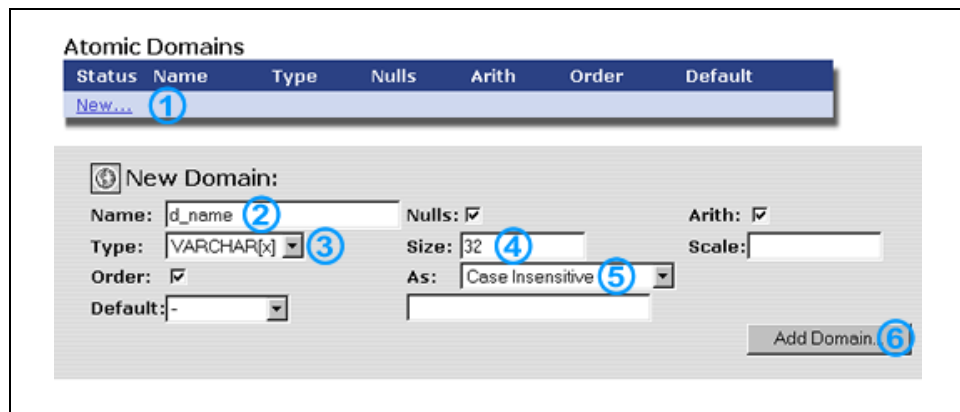
- Geben Sie als Namen „d_name“ ein (21.2).
- Wählen Sie als „Type“ „VARCHAR[x]“ aus (21.3).

Der Typ der Atomic Domain wird als „VARCHAR[x]“ festgelegt, da wir diese Domain wiederum als Typ für Columns verwenden wollen, die Textdaten aufnehmen sollen.

- Geben Sie im Feld „Size“ „32“ ein (21.4).
- Aktivieren Sie die Checkbox „Order“.
- Wählen Sie in der Drop-Down List „As:“ die Option „Case Insensitive“ aus (21.5).

Wie bereits erwähnt, aktivieren wir über diese Einstellung die case-insensitive Suche und Sortierung auf allen Columns, die diese Domain als Datentyp verwenden.

- Drücken Sie auf „Add Domain...“ (21.6).



SCREENSHOT 21: ANLEGEN EINER ATOMIC DOMAIN

Nun haben wir unsere eigene Domain angelegt und nach unseren Bedürfnissen konfiguriert. Jetzt können wir uns wieder den Columns zuwenden, die wir ja noch in unserer „customers“-Tabelle definieren wollen.

Die Columns „name“, „address“, „country“ und „discount“.

Uns fehlen also für unsere Beispielanwendung noch die Columns „name“, „address“, „country“ und „discount“. Diese Columns fügen wir prinzipiell auf die gleiche Art und Weise hinzu wie „num“, allerdings nehmen wir im Detail doch ein paar andere

Einstellungen vor, weshalb hier noch einmal der Ablauf zur Erstellung dieser weiteren Columns gezeigt wird.

Status	Name	Domain	Key	Index	Nulls	Default
New	num	INTEGER	Primary	Yes	Not Null	Counter
New...						
Composite Columns			Components			
New...						

SCREENSHOT 22: ÜBERSICHT ÜBER DIE COLUMNS

New Column for Common.customers:

Name: Index: Nulls:

Type: Size: Scale:

Key: Default:

SCREENSHOT 23: NEUE COLUMN IN TABELLE „CUSTOMERS“ ANLEGEN

- Drücken Sie auf „New...“ (22.1)
- Geben Sie „name“ als Namen ein (23.1).
- Deaktivieren Sie die Checkbox „Index“ (23.2).
- Aktivieren Sie die Checkbox „Nulls“ (23.3).
- Wählen Sie „Common.d_name“ als Type aus (23.4).
- Drücken Sie auf „Add Column“ (23.5).

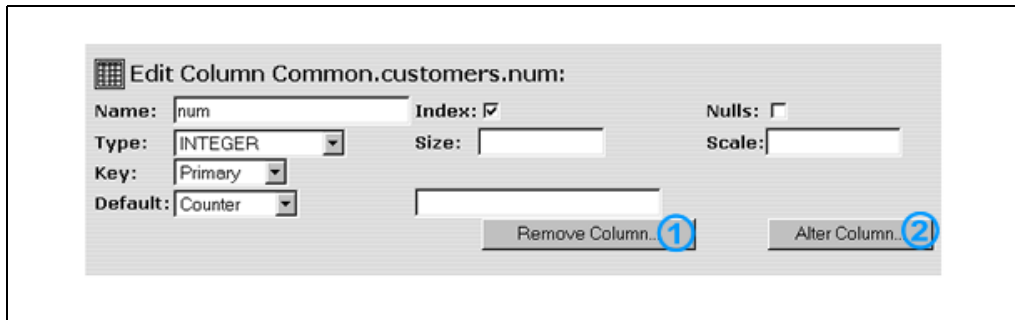
Wiederholen Sie diese Schritte für die weiteren Columns „address“, „country“ und „discount“. Sind diese Handgriffe erledigt, sollte die Anzeige dem folgenden Screenshot entsprechen:

Status	Name	Domain	Key	Index	Nulls	Default
New	num	INTEGER	Primary	Yes	Not Null	Counter
New	name	Common.d_name	-	-	Null	-
New	address	Common.d_name	-	-	Null	-
New	country	Common.d_name	-	-	Null	-
New	discount	Common.d_name	-	-	Null	-
New...						

SCREENSHOT 24: DIE VOLLSTÄNDIGE DEFINITION DER TABELLE „CUSTOMERS“

Hoppla, doch was falsch? Einfach Korrigieren!

Bei Fehleingaben können Sie jederzeit die Columns entweder löschen (25.1) oder auch nachträglich editieren (25.2), indem Sie einfach auf den Link „New“ (unter „Status“, vgl. Screenshot 24) der entsprechenden Column klicken und dann die Änderungen in dem am unteren Bildschirmbereich erscheinenden Dialog vornehmen.



SCREENSHOT 25: EDITIEREN VON COLUMNS

Was haben wir bisher gemacht?

Wir haben ...

- eine Datenbank namens „tutdemo“ angelegt,
- das Schema der Datenbank „tutdemo“ geladen,
- innerhalb des Schemas der Datenbank „tutdemo“ eine Tabelle namens „customers“ angelegt,
- innerhalb der Tabelle „customers“ die Spalten „num“, „name“, „address“, „country“ und „discount“ angelegt,
- für die Columns „name“, „address“, „country“ und „discount“ einen eigenen Datentyp, eine sogenannte Domain, angelegt, um case-insensitive Suche in diesen Columns zu ermöglichen,
- und nebenbei etwas über Datentypen, Domains, Such- und Sortierreihenfolgen und Primary Keys gelernt.

Was fehlt uns jetzt noch?

Wir müssen uns noch mit der irritierenden Tatsache auseinandersetzen, dass alle bisher von uns an der Datenbank „customers“ vorgenommenen Änderungen noch gar nicht durchgeführt wurden.

WICHTIG: Änderungen eines Datenbankschemas im PrimeBase Admin-Server

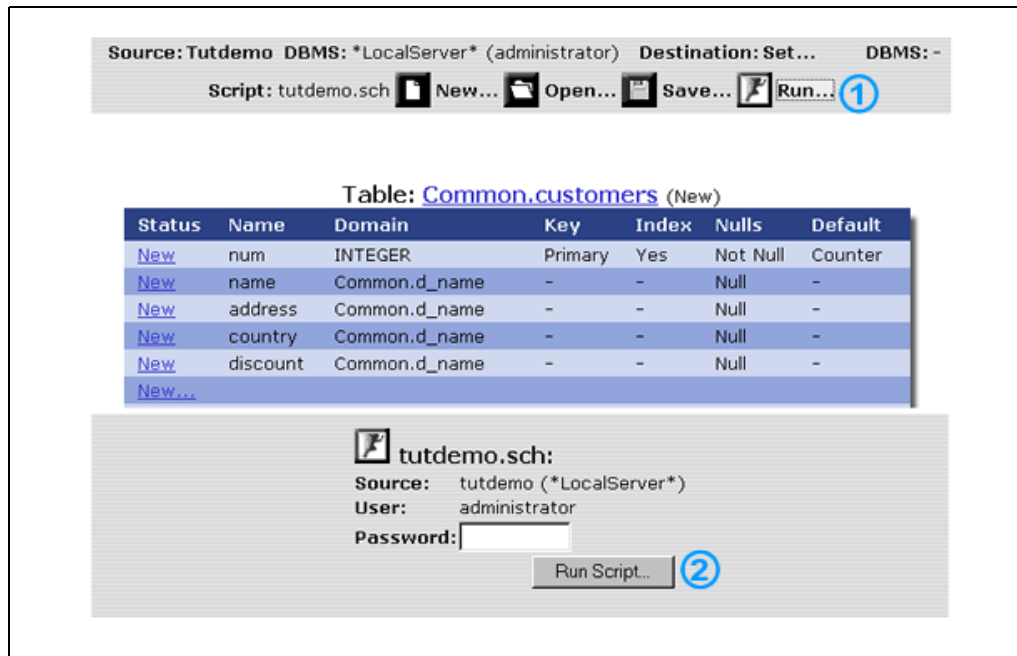
Beachten Sie unbedingt, dass bei Änderungen eines Datenbankschemas im Admin-Server nicht sofort die eigentliche Datenbank geändert wird!

Vielmehr merkt sich der Admin Server erst einmal alle von Ihnen vorgenommenen Eingriffe. Haben Sie alle gewünschten Änderungen vorgenommen und möchten diese nun auf die eigentliche Datenbank übertragen, wählen Sie „Run...“ (26.1) aus und bestätigen Sie „Run Script...“ (26.2).

Kommt es während der Übertragung der Änderungen auf die Datenbank zu Fehlern oder Problemen, werden vom Admin-Server automatisch alle bisher durchgeführten Änderungen zurückgenommen, so dass die Datenbank auf jeden Fall in einem funktionsfähigen und konsistenten Zustand verbleibt.

Jetzt aber: Die Änderungen des Schemas auf unsere Datenbank übertragen.

Wie Sie dem voranstehenden Einschub entnehmen können, sind wir wirklich nur noch einen Handgriff davon entfernt, die bisherigen Änderungen am Schema unserer Datenbank auch wirklich in unsere Datenbank „customers“ zu übertragen.



SCREENSHOT 26: DIE ÄNDERUNGEN DURCHFÜHREN.

- Drücken Sie unter „Script:“ auf „Run ...“ (26.1).

Nun erscheint oben rechts ein Bestätigungsfenster.

- Drücken Sie auf „Run Script...“ (26.2).

Die Änderungen des Schemas werden jetzt auf die eigentliche Datenbank übertragen. Während dieser Vorgang läuft, werden im unteren Bildschirmbereich Statusmeldungen angezeigt, anhand derer man den Verlauf dieses Prozesses überwachen kann.

Dies ist im Normalfall nicht weiter von Bedeutung – uns interessiert schließlich bloß, dass die Änderungen in die Datenbank aufgenommen werden. Aber im Falle eines Problems sind diese Statusmeldungen eine wichtige Hilfe bei der Suche nach möglichen Fehlerquellen.

Wie dem auch sei: Sie haben jetzt einen Einblick in den Database Server und dessen Verwaltung bekommen. Damit haben wir den ersten wichtigen Schritt hin zu der Entwicklung einer datenbankbasierten Anwendung hinter uns gebracht.

Die nächste große Schritt wird darin bestehen, uns mit dem Application Server vertraut zu machen und zu lernen, mit dessen Möglichkeiten webbasierte Datenbank-Anwendungen zu erstellen.

3. SECOND STEPS

Mit dem Application Server ist es möglich, webbasierte Datenbankanwendungen zu erstellen, welche von einem beliebigen Rechner aus per Web-Browser benutzbar sind.

Drei wesentliche Schritte sind notwendig, um eine Anwendung mit zu erstellen:

- die Definition und das Anlegen einer Datenbank mit den entsprechend benötigten Tabellen und Spalten.
- die Erstellung der HTML-Seiten, die als Grundlage für die webbasierte Anwendung dienen sollen.
- die Definition der Anwendung bzw. der Anwendungslogik.

Den ersten Schritt haben wir bereits in den „First Steps“ dieses Tutorials hinter uns gebracht, so dass wir uns jetzt auf die beiden letzteren konzentrieren können.

Die Erzeugung von HTML-Seiten für die von uns anvisierte Beispielanwendung ersparen wir Ihnen, indem wir entsprechende Dateien mitliefern. Sie finden diese Dateien im Verzeichnis des Application Servers unter „...\docs\tutdemo\originals“.

Im darüber liegenden Verzeichnis „...\docs\tutdemo“ werden wir im Verlauf dieser Übung ein paar HTML-Dateien ablegen, die wir für die Anbindung an unsere zuvor erstellte Datenbank modifizieren müssen.

Der eingebaute HTTP-Server und das Verzeichnis „docs“.

Es ist wichtig zu wissen, dass der Application Server einen eingebauten HTTP-Server – also einen Web-Server – hat.

Dies vereinfacht die Entwicklung von Anwendungen erheblich, da man sich zum Zeitpunkt der Anwendungsentwicklung überhaupt nicht mit der Integration in einen der marktüblichen HTTP-Server wie Apache, Netscape, Microsoft Internet Information Server etc. beschäftigen muss.

Dies bedeutet auch, dass keiner dieser HTTP-Server auf dem Entwicklungsrechner installiert sein muss.

Mit diesem Hintergrundwissen erklärt sich auch die vorhandene Verzeichnisstruktur, auf die wir oben schon kurz verwiesen haben („...\docs\tutdemo“).

Im Hauptverzeichnis befindet sich der Application Server, wie zuvor bereits erwähnt wurde. Das Unterverzeichnis „...\docs“ enthält alle Dateien und Verzeichnisse, auf die man über den eingebauten HTTP-Server zugreifen kann.

Üblicherweise wird dieses Verzeichnis in der Dokumentation von HTTP-Servern als „Document Root“, „Docs Root“ o. ä. bezeichnet.

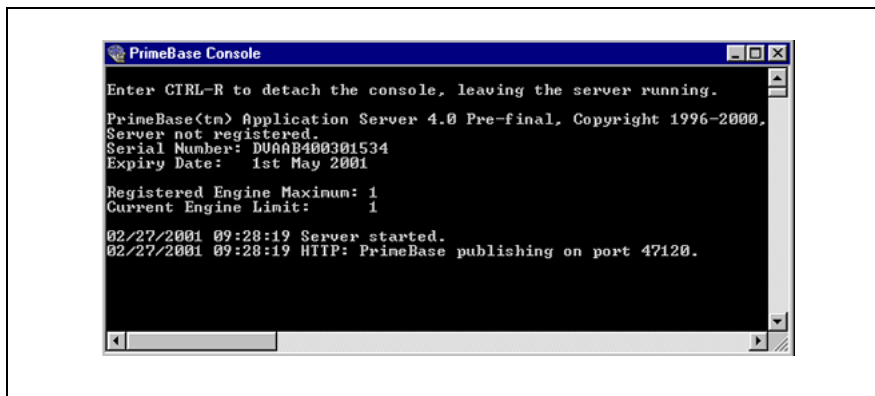
Das bedeutet, dass wir unsere Dateien, die über den eingebauten HTTP-Server geladen werden sollen, grundsätzlich in „...\docs“ oder in einem seiner Unterverzeichnisse ablegen müssen. Wir werden im Rahmen dieses Tutorials einzig im Verzeichnis „...\docs\tutdemo“ und seinen Unterverzeichnissen arbeiten.

Jetzt wollen wir aber wieder zur Tat schreiten und den Application Server in Betrieb nehmen.

Den Application Server starten

Um diese Übung durchzuführen, muss zunächst der Application Server gestartet werden:

- Führen Sie im Verzeichnis des Application Servers die Datei „pbas.exe“ aus. :



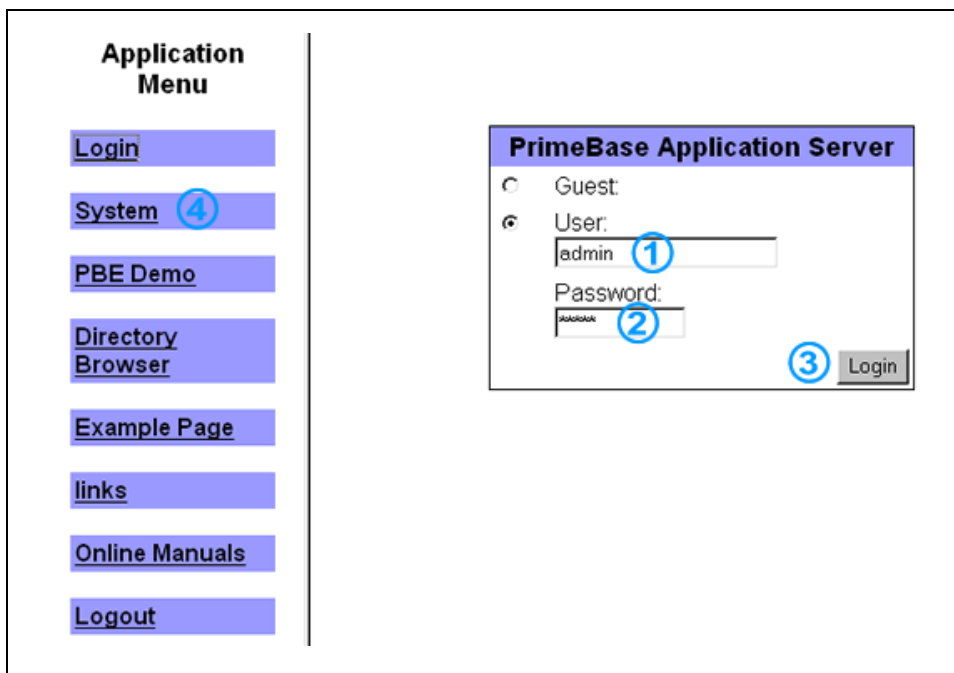
SCREENSHOT 27: DIE CONSOLE DES APPLICATION SERVERS

Von den auf der nun erscheinenden Console angezeigten Informationen ist für uns im Moment eine Angabe von besonderem Interesse und zwar die Portnummer, auf der sich der eingebaute HTTP-Server für Zugriffe von Web-Browsern publiziert.

Aus der letzten Zeile in der Console (Screenshot 27) lässt sich entnehmen, dass der eingebaute HTTP-Server auf Port 47120 publiziert wurde:

- Geben Sie in die Adresszeile des Browser „http://127.0.0.1:47120/“ ein.

Nun erscheint im Browser die Startseite des Application Servers, welche einerseits die installierten Anwendungen auflistet, auf die man ohne explizite Anmeldung, also sozusagen als „Gast“, Zugriff hat, und andererseits einen Dialog zur Systemanmeldung bietet, um Zugriff auf andere installierte Anwendungen zu bekommen, die für Gäste nicht freigeschaltet sind:



SCREENSHOT 28: DIE ANFANGSSSEITE DES APPLICATION SERVERS

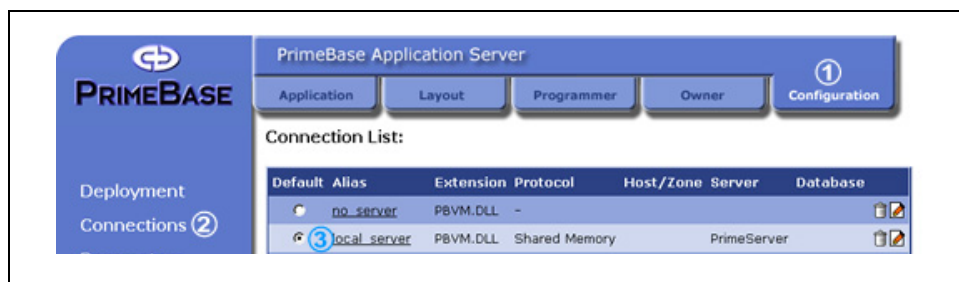
Der Default Benutzer „admin“ und der PrimeBase Datenbank Server.

Für dieses Tutorial brauchen wir uns eigentlich nicht regulär anzumelden, da wir eine Anwendung schreiben werden, die auch Gästen den Zugriff erlaubt. Allerdings müssen wir zunächst den PrimeBase Datenbank Server an den Application Server anbinden, damit letzterer auf unsere Datenbank „tutdemo“ zugreifen kann. Hierzu muß im Systembereich des Application Servers die entsprechende Einstellung vorgenommen werden. Da allerdings der Systembereich ein geschütztes Modul ist, muß man sich zuerst einloggen. Das geschieht im Auslieferungszustand des Application Server als Benutzer „admin“:

- Geben Sie den Benutzernamen „admin“ ein (28.1)
- Geben Sie das Passwort „admin“ ein (28.2)
- Drücken Sie „Login“ (28.3).
- Drücken Sie auf „System“ (28.4)

Nun können Sie innerhalb des Systemmoduls die notwendige Einstellung vornehmen:

- Wählen Sie im Systemmodul die Registrierkarte „Configuration“ an (29.1)
- Drücken Sie die Auswahl „Connections“ (29.2)
- Setzen sie die Verbindung auf „local_server“ (29.3)



SCREENSHOT 29: VERBINDUNG ZUM PRIMEBASE DATENBANK SERVER HERSTELLEN

Der Application Server übernimmt nun diese Einstellung automatisch als neue Standardverbindung. Mehr soll an dieser Stelle nicht zum Systemmodul gesagt werden, sondern zunächst nur etwas mehr zu der Unterscheidung zwischen einer Anwendung und einem Modul.

Anwendungen, Module und die „PrimeBase Enterprise Objects“.

Um hier für Klarheit zu sorgen, müssen wir ein bisschen weiter ausholen.

Zunächst ist es wichtig zu wissen, dass der Application Server mit einer umfangreichen „Bibliothek“ an Funktionalitäten ausgeliefert wird. Diese Bibliothek heißt „PrimeBase Enterprise Objects“.

Die PrimeBase Enterprise Objects (kurz „PBE“) enthalten alle wesentlichen Funktionalitäten, um HTML-Seiten mit Daten aus Datenbanken zu füllen und übliche Datenbankoperationen wie Einfügen, Ändern, Löschen usw. ohne eigene Programmierung an HTML-Seiten und HTML-Formulare zu knüpfen. Aber auch darüber hinaus gehende Funktionalität lässt sich ohne traditionelle Programmierung mit Hilfe der PrimeBase Enterprise Objects implementieren.

Zusätzlich zu den PrimeBase Enterprise Objects wird eine einzige, aber sehr wichtige Anwendung namens „ModuleApplication“ mitgeliefert. Sowohl die PrimeBase Enterprise Objects als auch die Anwendung ModuleApplication sind bereits vorinstalliert.

Die Standardanwendung „ModuleApplication“.

Bei ModuleApplication handelt es sich um eine Anwendung, die es uns ermöglicht, so genannte Module zu entwickeln.

Dies ist sehr hilfreich, da uns dies die Entwicklung einer „ausgewachsenen“ Anwendung unter PrimeBase Enterprise Objects erspart. Die Entwicklung einer PBE-Anwendung erfordert letztendlich nämlich ein bisschen mehr Übersicht über die PBE, als wir uns an dieser Stelle erarbeiten wollen (insbesondere, weil es nur in sehr seltenen Fällen notwendig sein wird, eine eigene PBE-Anwendung zu erstellen).

ModuleApplication stellt übrigens auch die oben schon kurz vorgestellte Login-Funktionalität zur Verfügung, die es uns erlaubt, unsere eigenen Module mit detaillierten Zugriffsrechten zu versehen, ohne diese Grundfunktionalität selbst implementieren zu müssen.

Warum Module?

Die Vorteile von Modularisierung sind vielleicht nicht direkt erkennbar, aber unter Zuhilfenahme von ein paar Analogien kann man sich sehr einfach verdeutlichen, warum Module Sinn machen.

Denken Sie nur an heutige Computersysteme, die in der Regel modular gestaltet und aufgebaut sind. Wer heutzutage seinem Rechner ein bisschen mehr Speicher gönnen möchte, der holt sich ein paar Speichermodule und installiert diese in den dafür vorgesehen Steckplätzen, wobei ein detailliertes Wissen vom Gesamtaufbau des Computers dafür nicht notwendig ist.

Mit der ModuleApplication verhält es sich genauso: Sie bietet uns eine Reihe definierter Schnittstellen und weitergehender Funktionalitäten an, die es uns erlaubt, Module zu schreiben, ohne dabei wissen zu müssen, wie es „innendrin“ in der ModuleApplication eigentlich aussieht.

Ein weiterer Vorteil bei der Verwendung von Modulen ist, dass sich auch größere Projekte in einem Team unabhängig voneinander entwickeln lassen.

Nicht zuletzt ist die Installation und Deinstallation von Modulen sehr leicht. Es reicht, das Verzeichnis des Moduls im Verzeichnis „...\docs“ abzulegen bzw. es dort zu entfernen und dies dem Application Server kurz mitzuteilen – schon ist ein neues Modul installiert oder deinstalliert.

Wie dem auch sei, wir behalten unsere Beispielanwendung – pardon, unser Beispielmodul – im Auge: Es soll uns zur datenbankgestützten Verwaltung von Kundeninformationen über unseren Web-Browser verhelfen.

Unser Modul - Wie macht man das?

Ein Modul zu erstellen ist sehr leicht.

Ein Modul besteht immer aus zwei Verzeichnissen:

- dem Modulverzeichnis, dessen Name keine Leerzeichen, Umlaute oder Sonderzeichen enthalten sollte,
- und dem „exec“-Verzeichnis, welches immer innerhalb des Modulverzeichnisses liegen muss.

Wohin mit den HTML-Dateien, Bildern, etc.?

Im Modulverzeichnis werden immer alle HTML-Dateien, Bilder usw. abgelegt, die zu dem jeweiligen Modul gehören und die zu der (HTML-basierten) Benutzeroberfläche des Moduls gehören.

Im Verzeichnis „exec“ werden Dateien abgelegt, die wesentliche Informationen über ein Modul enthalten – eine solche Datei werden wir in Kürze erstellen.

Wohin mit dem Modulverzeichnis?

Alle Modulverzeichnisse müssen sich in im Verzeichnis des Application Servers unter „..\docs“ befinden.

Wie Sie sehen können, befindet sich in diesem Verzeichnis – neben anderen Modulverzeichnissen – auch schon das zuvor angesprochene Verzeichnis „tutdemo“, welches aber bisher keine weiteren Dateien enthält. Allerdings befinden sich im Verzeichnis „tutdemo“ das bereits erwähnte „exec“-Verzeichnis und ein weiteres Verzeichnis namens „originals“.

Das Verzeichnis „originals“.

Im Verzeichnis „originals“ befinden sich Dateien, mit denen wir im weiteren Verlauf dieses Tutorials arbeiten werden. Es enthält außerdem bereits fertig bearbeitete Versionen der einzelnen Dateien, was uns erlaubt, bei Problemen auf die dort bereits vorbereiteten Dateien zurückzugreifen, um so das Tutorial weiter bearbeiten zu können.

Wichtig!

Arbeiten Sie bitte niemals direkt mit den im Verzeichnis „originals“ abgelegten Dateien, sondern erstellen Sie immer eine Kopie der Dateien und arbeiten Sie mit dieser Kopie. Am besten Sie erstellen diese Kopie gleich in dem Verzeichnis, in dem sie letztendlich auch abgelegt werden soll. Für dieses Tutorial ist dies das Verzeichnis „tutdemo“.

Unser Modul anlegen

Wir werden in den nächsten Schritten ein neues Modul erstellen, bei dem ein HTML-Formular mit der in „First Steps“ angelegten Kundendatenbank verknüpft wird.

Da die Verzeichnisse für unser Modul („..\docs\tutdemo“ und „..\docs\tutdemo\exec“) schon angelegt sind, müssen wir bloß noch die richtigen Dateien in diesen Verzeichnissen anlegen bzw. sie in diese kopieren.

Deklaration unseres Moduls.

- Erstellen Sie im Verzeichnis „..\docs\tutdemo\exec“ eine neue Datei mit dem Namen „tutdemo.pbt“.
- Öffnen Sie die Datei in einem beliebigen Texteditor und geben Sie die folgenden Zeilen ein:

```
static
{
    PBE:declareModule( "GuestLoginModule", "Tutorial Demo", "custform.htm", "tutdemo");
}
```

DEKLARATION UNSERES MODULS

- Speichern Sie die Datei.

Mit diesen Zeilen haben wir ein Modul vom Typ „GuestLoginModule“ definiert (erster Parameter). Das Modul trägt den Namen „Tutorial Demo“ (zweiter Parameter) und die

Einstiegsseite des Moduls ist „custform.htm“(dritter Parameter). Zuletzt wird noch die entsprechende Datenbank „tutdemo“ geöffnet (vierter Parameter).

Alles nur Methoden

Die Moduldeklaration an sich ist nichts anderes als der Aufruf einer Methode (so werden Funktionen oder Prozeduren in der Welt der objektorientierten Sprachen bezeichnet). Wir rufen mit ...

```
static
{
    PBE:declareModule( "GuestLoginModule", "Tutorial Demo", "custform.htm" , "tutdemo");
}
```

DEKLARATION UNSERES MODULS

... die Methode „declareModule()“ des bereits vorhandenen Objektes „PBE“ auf (in anderen Programmiersprachen wird übrigens meist anstelle des Doppelpunktes ein einfacher Punkt verwendet).

Später kommen wir noch zu der Deklaration von Objekten und Events und auch diese bestehen nur in dem Aufrufen von Funktionen. Insgesamt werden wir in diesem Tutorial nur fünf Methoden/Prozeduren benutzen:

- declareModule() dient zur Deklaration von Modulen
- declareObject() dient zur Deklaration von Objekten
- declareAttribute() dient zur Deklaration von Attributen
- objectEvent() dient zur Deklaration von Objekt Events/Ereignissen
- attributeEvent() dient zur Deklaration von Attribut Events/Ereignissen

DIE IN DIESEM TUTORIAL BENUTZTEN METHODEN

Das klingt recht kompliziert, aber wir brauchen uns von diesen Details nicht verwirren zu lassen, da wir diese Funktionen nur für unsere eigenen Zwecke nutzen wollen, nicht aber die dahinterliegenden Details kennen müssen.

Die „LiveTag Markup Language“ ist HTML.

Die Dateierdung „.lml“ steht für „LiveTag Markup Language“ (kurz LML). Diese ist aber keine weitere Markup Language, die Sie lernen müssen, sondern es handelt sich bei der LiveTag Markup Language um eine sinnvolle Erweiterung für HTML. Dies hat den entscheidenden Vorteil, dass beliebige HTML-Editoren verwendet werden können, um Seiten für PrimeBase Web-Anwendungen zu erstellen.

Die LiveTag Markup Language erweitert HTML lediglich um drei zusätzliche Tags:

- <container> / </container>
- <if> / </if>
- <output>

Die meisten HTML-Editoren sind in der Lage, ihnen unbekannte Tags zu „erlernen“ bzw. zu ignorieren.

Sollte die Verwendung dieser Tags dennoch ein Problem darstellen, so ist es dank der Flexibilität des Application Servers dennoch möglich, die entsprechende Funktionalität auch an andere, HTML-konforme Tags zu binden.

Die Dateitypen des Application Servers

- .dal für DAL
- .pbt für PrimeBaseTalk
- .lml für LiveTag Markup Language
- .htd für HyperText DAL

.dal: Die Data Access Language (kurz DAL) ist eine von Apple entwickelte Programmiersprache, die auf SQL basiert. SQL, die Structured Query Language, ist die am weitesten verbreitete Abfragesprache für Datenbanken.

.pbt: PrimeBaseTalk ist die in PrimeBase verwendete Programmiersprache. Sie ist objektorientiert und weitgehend kompatibel zu Java und voll kompatibel zu DAL. Sowohl .pbt- als auch .dal-Dateien enthalten immer einfachen Text, der mit einem Texteditor verändert werden kann. Dieser Quelltext wird zur Laufzeit kompiliert und anschließend ausgeführt. Damit entfällt der von anderen Programmiersprachen bekannte Kompilationsvorgang.

.lml: Die LiveTag Markup Language ist kompatibel zu HTML, definiert aber darüberhinaus drei zusätzliche Tags (<container> /</container>, <if> /</if> und <output>). Beachten Sie bitte, dass HTML-Dateien (.html und .htm) in der Grundkonfiguration des Application Servers als LML-Dateien interpretiert werden.

.htd: HyperText-DAL-Dateien können neben reinem HTML das Tag <dal> /</dal>, sogenannte DAL Kommentare (<!--DAL sourcecode -->) und DAL Attribute (<form name="!--DAL sourcecode">) enthalten. Beachten Sie, dass .htm-Dateien in der Grundkonfiguration des Application Servers auch als HyperText-DAL-Dateien interpretiert werden.

All dies nur der Vollständigkeit halber – wirklich benötigen werden wir in diesem Tutorial nur .pbt- und .lml-Dateien.

Die Anwendungsdefinition.

Die Datei „tutdemo.pbt“ enthält jetzt die Definition unseres Moduls und sie wird auch alle weiteren Definitionen aufnehmen, die wir im Verlauf dieses Tutorials noch erstellen werden.

Dementsprechend macht es Sinn, bei dieser Datei von der „Anwendungs-definitions-Datei“ oder lieber kurz: von der „Anwendungsdefinition“ zu sprechen.

Kommentare in der Anwendungsdefinition

```
static
{
    // Dies ist ein einzeliger Kommentar.
    PBE:declareModule( "GuestLoginModule", "Tutorial Demo", "custform.htm", "tutdemo");

    /* Dies ist ein mehrzeiliger Kommentar, der hier beginnt...
    // Die folgende Zeile ist irgendwie kaputt:
    PBE:delcarMod( "GuestLoginModule", "Tutorial Demo", "custform.htm", "tutdemo");
    ... und hier endet und der wirklich sehr viel Text enthalten kann*/
}
```

Die können in der Datei mit der Anwendungsdefinition natürlich auch Kommentare einfügen. Einzeilige Kommentare werden mit „//“ begonnen und enden mit dem Zeilenende. Mehrzeilige Kommentare werden mit „/*“ begonnen und mit „*/“ beendet. Mehrzeilige Kommentare können auch ineinander geschachtelt werden und dürfen einzeilige Kommentare enthalten.

Die Anwendungsdefinition in den Application Server laden.

Immer, wenn wir die Anwendungsdefinition unseres Moduls ändern, müssen wir dem Application Server diese Änderung mitteilen, indem wir ihn dazu bringen, die entsprechende Datei neu einzulesen.

Um dies zu erreichen, kann man den Application Server mit „halt“ stoppen und anschließend wieder starten, was allerdings ein recht umständlicher Vorgang ist.

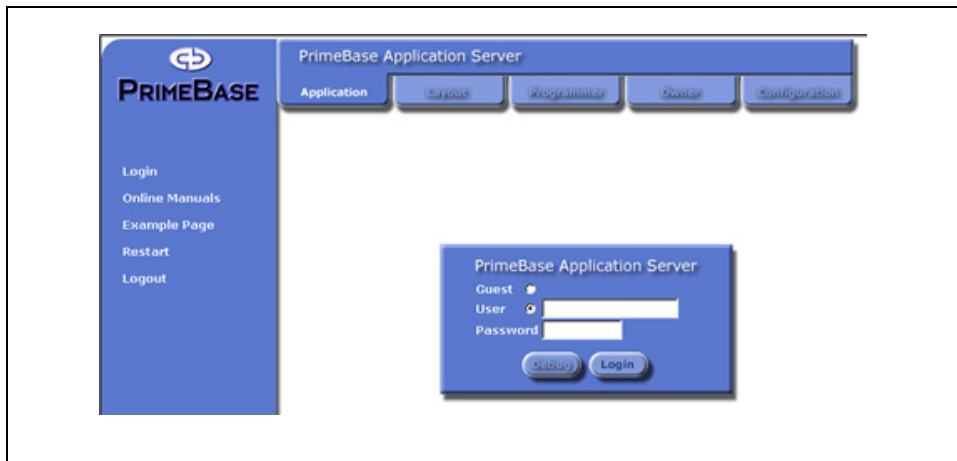
Einfacher ist das Laden der Datei „restart.exec“ über den Browser – diese Datei veranlasst den Application Server dazu, alle installierten Dateien und Module erneut einzulesen:

- **Laden Sie „<http://127.0.0.1:47120/system/restart.exec>“ im Browser.**

Der Systembereich des Application Servers.

Nun erscheint statt der Anfangsseite (Screenshot 28) erneut der Systembereich des Application Servers (Screenshot 30).

In diesem Systembereich lassen sich Konfigurationsänderungen des Application Servers, Zugriffsrechte auf Module u. a. einstellen und verwalten. Lesen Sie im „PrimeBase Application Server User's Guide“ nach, wie dies funktioniert.



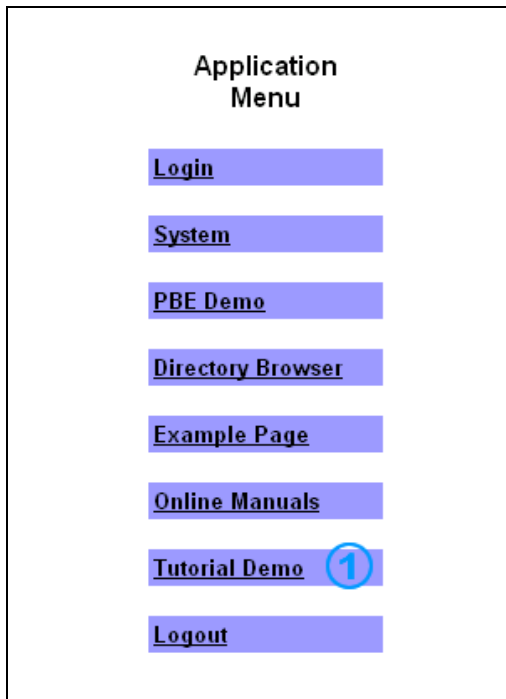
SCREENSHOT 30: DER SYSTEMBEREICH DES APPLICATION SERVERS.

Für dieses Tutorial wollen wir uns darauf beschränken, nur mit der Liste der installierten Module zu arbeiten, die wir über die URL „<http://127.0.0.1:47120/>“ zur Verfügung gestellt bekommen. Da der Application Server gerade alle installierten Dateien und Module erneut eingelesen hat, müsste in dieser Liste auch unser Modul „Tutorial Demo“ auftauchen:

- **Laden Sie „<http://127.0.0.1:47120/>“ im Browser.**

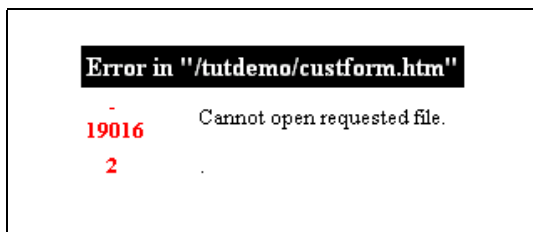
Im linken Bereich erscheint jetzt wieder die Auflistung der für den Gastzugriff installierten Module – diesmal mit dem zusätzlichen Eintrag „Tutorial Demo“.

- Klicken Sie auf den Link „Tutorial Demo“ (31.1).



SCREENSHOT 31: IM MENÜ DER INSTALLIERTEN MODULE ERSCHEINT DER EINTRAG „TUTORIAL DEMO“.

Nun wird die folgende Fehlermeldung im Browser ausgegeben:



SCREENSHOT 32: FEHLERMELDUNG, DA DIE HTML-DATEI „CUSTFORM.HTM“ NICHT GELADEN WERDEN KONNTE.

Wenn wir einen Blick in „..\docs\tutdemo“ werfen – also in das Verzeichnis, in dem die zu unserem Modul gehörigen HTML-Dateien abgelegt werden müssen – befindet sich dort noch keine Datei. Somit ist es kein Wunder, dass wir eine Fehlermeldung bekommen.

Die erste HTML-Datei unseres Moduls.

Wie uns die Fehlermeldung mitteilt, benötigen wir die Datei „custform.htm“, da wir diese selbst als Einstiegsseite unseres Moduls deklariert haben. Eine Vorlage für diese Datei finden wir im Verzeichnis „originals“.

- Kopieren Sie „custform.htm“ von „..\docs\tutdemo\originals“ in unser darüber liegendes Modulverzeichnis („..\docs\tutdemo“).

Arbeitsteilung: Der Web-Designer macht das Layout.

Die Datei „custform.htm“ liegt so vor, wie sie mit einem typischen HTML-Editor erzeugt werden würde, bzw. wie wir sie von einem Web-Designer, der das Layout für unsere Anwendung macht, bekommen würden.

Bedenken Sie, dass in der Grundkonfiguration des Application Servers Dateien mit der Endung „.htm“ anders behandelt werden als Dateien mit der Endung „.html“. Da wir aber allen HTML-Dateien die Endung „.lml“ geben werden, ist es an dieser Stelle nicht weiter wichtig, hier auf Details einzugehen. Wichtig ist nur, dass die Namen unserer unbearbeiteten HTML-Dateien mit „.htm“ enden.

Das HTML-Attribut „name“.

Eine Änderung haben wir für dieses Tutorial an der Datei „custform.htm“ gegenüber der Form, in der wir die Datei von einem Web-Designer bekommen würden, allerdings doch vorgenommen: Der Inhalt des HTML-Attributs „name“ wurde von uns in der gesamten Datei entfernt (<xyz name="" ...>).

Zu welchem Zweck? Um Fehlermeldungen vorzubeugen, da das Attribut „name“ ausgewertet wird, sobald wir Dateien mit der Endung „.lml“ über den Application Server laden (zwar heißt unsere Datei noch „custform.htm“, aber das ändern wir schon bald). Der Inhalt dieses Attributs wird als eine Referenz auf eine Definition oder ein Objekt interpretiert. Da wir bisher aber keine Objekte deklariert haben, würde ein wie auch immer gearteter Inhalt in diesem Attribut zu einer Fehlermeldung führen.

In „custform.htm“ wird auch noch ein kleines Bild mit Namen „list.gif“ referenziert, so dass wir auch diese Datei noch in unser Modulverzeichnis kopieren müssen:

- **Kopieren Sie „list.gif“ vom „originals“-Verzeichnis in das Modulverzeichnis.**

Die Einstiegsseite unseres Moduls.

Jetzt haben wir alles beisammen, um einen Blick auf die Einstiegsseite unsere Moduls zu werfen:

- **Klicken Sie auf den Link „Tutorial Demo“ (31.1) und es erscheint die Einstiegsseite unseres Moduls:**

Customer Information: [Hier soll eine ID Nummer für jeden Kunden eingeblendet werden.]

Name

Address

Country:

Discount:

Search |< < > >|

Clear Save New Delete

SCREENSHOT 33: DIE EINSTIEGSSEITE UNSERES MODULS

Drücken Sie doch jetzt versuchsweise die Knöpfe auf dieser HTML-Seite und geben Sie ruhig auch Daten in die Eingabefelder ein. Sie werden feststellen, dass dieses Formular noch nicht funktioniert.

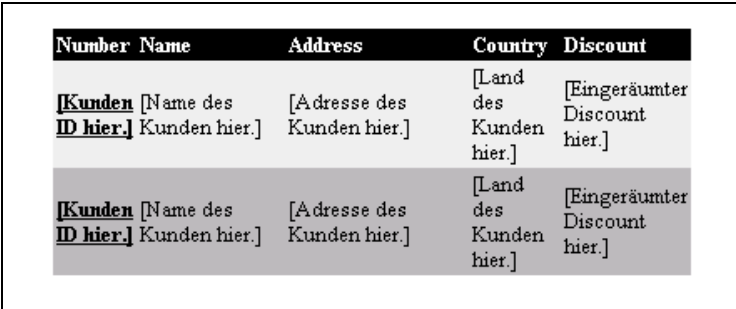
Anmerkungen des Web-Designers in unseren HTML-Dateien.

Bemerkungen des Web-Designers sind in eckigen Klammern direkt in die HTML-Dateien geschrieben. Diese Bemerkungen müssen wir – als diejenigen, die sich um Anwendungslogik und Datenbank-Anbindung kümmern – durch bestimmte Konstrukte ersetzen, um vom Application Server an den entsprechenden Stellen Daten aus der Datenbank einfügen zu lassen.

Die andere HTML-Seite unseres Moduls.

Das Bild am oberen Bildschirmrand (vgl. Screenshot 33.1) ist übrigens ein Link auf eine andere Datei, nämlich „custfound.htm“. Da diese Datei noch nicht in unserem Modulverzeichnis liegt, kopieren wir auch sie aus dem Ordner „originals“ dorthin.

- Kopieren Sie die Datei „custfound.htm“ in unser Modulverzeichnis.
- Klicken Sie im Browser auf das kleine Bild mit dem Link auf „custfound.htm“ (33.1).



Number	Name	Address	Country	Discount
[Kunden ID hier.]	[Name des Kunden hier.]	[Adresse des Kunden hier.]	[Land des Kunden hier.]	[Eingeräumter Discount hier.]
[Kunden ID hier.]	[Name des Kunden hier.]	[Adresse des Kunden hier.]	[Land des Kunden hier.]	[Eingeräumter Discount hier.]

SCREENSHOT 34: DIE DATEI „CUSTFOUND.HTM“ IM BROWSER

„custfound.htm“ soll die gefundenen Kunden tabellarisch auflisten. Davon ist natürlich bisher nicht viel zu sehen, da bisher noch keine Kundendaten in die Datenbank eingepflegt worden sind und wir uns zudem noch gar nicht um die Datenbank-Anbindung unserer HTML-Seiten gekümmert haben.

Verknüpfung von HTML und Datenbanken

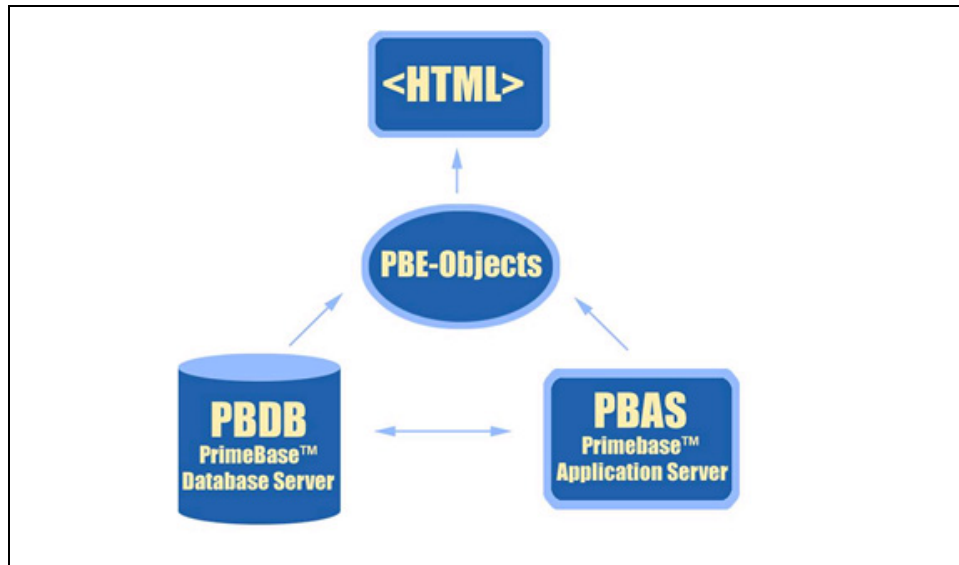
PrimeBase Web-Anwendungen und -Module bestehen aus drei Komponenten:

- eine Datenbank
- eine Anwendungsdefinition
- eine Benutzeroberfläche (HTML-Dateien)

Die Erzeugung der Datenbank haben wir für den Rahmen dieses Tutorials schon abgeschlossen. Somit können wir uns auf das Vervollständigen unserer Anwendungsdefinition und das Anpassen unserer HTML-Dateien konzentrieren.

Das HTML-Formular in „custform.htm“ soll später dazu dienen, die Kundendaten in unserer Datenbank zu verwalten (ändern, hinzufügen, löschen, blättern und suchen).

Die Verknüpfung zwischen HTML-Seiten und Datenbanken wird durch die PrimeBase Enterprise Objects hergestellt.



SCREENSHOT 35: SO ARBEITET PRIMEBASE™

Wir wollen jetzt die Objekte erstellen, die zur Anbindung der Datenbank an die HTML-Seiten benötigt werden.

Objekte zur Verknüpfung von Datenbank und HTML-Dateien erstellen.

Die Deklarationen unserer Objekte schreiben wir ebenfalls in die Datei „tutdemo.pbt“. Erweitern Sie diese Datei jetzt so, dass ihr Inhalt dem Folgenden entspricht:

```
static
{
    PBE:declareModule( "GuestLoginModule", "Tutorial Demo", "custform.lml" , "tutdemo");

    PBE:declareObject( "PBETable", "tutcust", "tutdemo!common.customers", "num" );
}
```

Die Objektdeklaration.

Was haben wir eben gemacht? Neben der Änderung des Namens unserer Einstiegsseite von „custform.htm“ auf „custform.lml“ (dazu gleich mehr), haben wir ein Objekt definiert.

Warum haben wir ein Objekt definiert?

Es muss einen Weg geben, den PrimeBase Enterprise Objects mitzuteilen, mit welcher Datenbank und mit welcher Tabelle man arbeiten möchte. Durch die Deklaration eines Objektes teilen wir den PrimeBase Enterprise Objects diese Informationen mit.

Trotz des auf den ersten Blick kompliziert erscheinenden Quelltextes erfordert die Objektdeklaration keine Programmierkenntnisse, sondern lediglich ein Grundverständnis für die Syntax, die möglichen Methoden und die dabei zu übergebenden Parameter.

Die declareObject-Methode.

Zum Erzeugen von Objekten wird die Methode „declareObject()“ verwendet. Diese Methode gehört zu dem Objekt „PBE“, daher müssen wir jedem Aufruf von

declareObject() ein „PBE:“ voranstellen. Es ist nicht wichtig, dies in seiner vollen (objektorientierten) Bedeutung zu verstehen, da wir diese Prozedur nur für unsere Zwecke nutzen wollen.

Der erste Parameter: Objekttyp.

Beim Aufruf von PBE:declareObject() übergeben wir als erstes den Typ des Objektes, welches wir benötigen („PBETable“, erster Parameter).

PBETable ist ein Objekttyp mit der Fähigkeit, auf eine Tabelle innerhalb einer Datenbank zuzugreifen. Es gibt noch weitere Objekttypen, aber wir werden uns in diesem Tutorial mit möglichst wenigen beschäftigen, um nicht unnötig Verwirrung zu stiften.

Der zweite Parameter: Name unseres Objektes.

Als zweites übergeben wir mit „tutcust“ einen eindeutigen Namen für unser Objekt. Dies ist wichtig, um bei der Verwendung mehrerer Objekte gezielt einzelne davon ansprechen zu können.

Dieser Name unterliegt erfreulicherweise nicht der Begrenzung auf 32 Zeichen, ist aber case-sensitive. Dies ist übrigens die Ausnahme von der Begrenzung von 32 Zeichen bei Namen und Bezeichnungen, auf die bereits in den „First Steps“ hingewiesen worden ist.

Der dritte Parameter: Vollqualifizierter Name unserer Tabelle.

Der dritte Parameter enthält den „vollqualifizierten“ Namen der Tabelle, mit der wir arbeiten möchten („tutdemo!common.customers“). Der vollqualifizierte Name einer Tabelle wird immer wie folgt gebildet:

Datenbankname + „!common.“ + Tabellenname

Hier ein paar Beispiele:

tutdemo!common.customers
projektdaten!common.Kunden
projekte!common.Termine

Wozu gibt es den „vollqualifizierten“ Namen einer Tabelle?

Stellen Sie sich vor, Sie müssten ein Modul entwickeln, welches auf Daten in verschiedenen Datenbanken zugreift. Jede dieser Datenbanken könnte eine eigene Tabelle mit dem Namen „customers“ enthalten.

Da die PrimeBase Enterprise Objects nicht selbst entscheiden können, welche „customers“-Tabelle in welcher Datenbank gemeint ist, muss zur Vermeidung von Mehrdeutigkeiten immer mit angegeben werden, in welcher Datenbank sich eine Tabelle befindet.

Der vierte Parameter: Name des Primary Keys unserer Tabelle.

Der vierte Parameter enthält den Namen der Primary Key Column unserer Tabelle. Wie Sie sich wahrscheinlich erinnern, lautet der Name der Primary Key Column unserer Tabelle „num“.

„.htm“ und „.html“ werden zu „.lml“.

Bei der letzten Änderung unserer Moduldeklaration haben wir den Namen der Einstiegsseite ebenfalls geändert. Dies ist notwendig, da der Application Server über die Dateiendung feststellt, um welche Art von Datei es sich handelt und wie er damit umgehen muss. Da wir unsere HTML-Seiten jetzt mit der Datenbank verknüpfen wollen, müssen unsere HTML-Dateien die Endung „.lml“ bekommen.

Auch innerhalb der HTML-Dateien müssen wir ein paar Änderungen vornehmen, da dort Verweise auf die alten Dateinamen vorhanden sind.

- **Benennen Sie „custform.htm“ nach „custform.lml“ um.**
- **Benennen Sie „custfound.htm“ nach „custfound.lml“ um.**
- **Öffnen Sie „custform.lml“.**
- **Ändern Sie Zeile 6 von ...**

```
<form name="" action="custform.htm" method="post">
```

... in ...

```
<form name="" action="custform.lml" method="post">
```

- **Ändern Sie Zeile 10 von ...**

```
<a href="custfound.htm"></a>
```

... in ...

```
<a href="custfound.lml"></a>
```

- **Speichern Sie die Änderungen und schließen Sie die Datei wieder.**
- **Öffnen Sie die Datei „custfound.lml“ in einem Texteditor.**
- **Ändern Sie Zeile 9 von ...**

```
<form name="" action="custfound.htm" method="post">
```

... in ...

```
<form name="" action="custfound.lml" method="post">
```

- **Ändern Sie Zeile 22 von ...**

```
<td><a href="custform.htm"><b>[Kunden ID hier.]</b></a></td>
```

... in ...

```
<td><a href="custform.lml"><b>[Kunden ID hier.]</b></a></td>
```

- **Ändern Sie Zeile 31 von ...**

```
<td><a href="custform.htm"><b>[Kunden ID hier.]</b></a></td>
```

... in ...

```
<td><a href="custform.lml"><b>[Kunden ID hier.]</b></a></td>
```

Die weiteren Merkmale unseres „tutdemo“-Objekts: Attribute.

Um Zugriff auf einzelne Columns in der Tabelle zu bekommen, müssen wir den PrimeBase Enterprise Objects auch mitteilen, welche Columns wir benutzen möchten. Also erweitern wir unsere Anwendungsdefinition entsprechend:

```
static
{
    PBE:declareModule( "GuestLoginModule", "Tutorial Demo", "custform.lml", "tutdemo" );

    PBE:declareObject( "PBETable", "tutcust", "tutdemo!common.customers", "num" );
    PBE:declareAttribute( "PBEColumn", "name" );
    PBE:declareAttribute( "PBEColumn", "address" );
    PBE:declareAttribute( "PBEColumn", "country" );
    PBE:declareAttribute( "PBEColumn", "discount" );
}
```

UNSER ERSTES OBJEKT - MIT ATTRIBUTEN

Die declareAttribute()-Methode.

Wie Sie sehen, haben wir für jede Column in unserer Tabelle „customers“ einmal die Methode „declareAttribute()“ aufgerufen.

Allerdings gibt es eine Ausnahme und zwar die Column „num“, die nicht extra deklariert wird, da „num“ bereits bei der Objektdeklaration angegeben wurde, wodurch sie automatisch als Attribut unseres „tutcust“ Objektes definiert wird.

Der erste Parameter: Objekttyp.

Der erste Parameter enthält den Namen des Objekttyps. Da wir auf Spalten einer Tabelle zugreifen wollen, muss das Objekt „PBEColumn“ verwendet werden, welches die dafür notwendigen Fähigkeiten besitzt.

Der zweite Parameter: Name der Spalte.

Der zweite Parameter enthält den Namen der Column in der Tabelle.

Mehrere Objekte deklarieren

Man kann ohne weiteres mehrere Objekte deklarieren, so z. B.:

```
[...]
    PBE:declareObject( "PBETable", "tutcust", "tutdemo!common.customers", "num" );
    PBE:declareAttribute( "PBEColumn", "name" );

    PBE:declareObject( "PBETable", "termine", "tutdemo!common.Termine", "num" );
    PBE:declareAttribute( "PBEColumn", "terminName" );
[...]
```

DEKLARATION MEHRERER OBJEKTE

Wie wird aber nun unterschieden, zu welchem Objekt ein Attribut gehört?

Die PrimeBase Enterprise Objects „merken“ sich die letzte Objektdeklaration und verwenden sie für alle folgenden Attributdeklarationen. Kommt eine weitere Objektdeklaration, wird diese für alle darauf folgenden Attributdeklarationen verwendet usw.

Für das obige Beispiel bedeutet dies, dass das Attribut „name“ eindeutig zu dem Objekt „tutcust“ gehört und das Attribut „terminName“ eindeutig dem Objekt „termine“ zugeordnet ist.

Der letzte große Schritt: Anpassung der HTML-Seiten.

Wir haben nun alle wesentlichen Dinge in unsere Anwendungsdefinition aufgenommen und können uns jetzt der Anpassung unsererer HTML-Seiten zuwenden.

- **Öffnen Sie die Datei „custform.lml“ mit einem Texteditor.**

Wir müssen den PrimeBase Enterprise Objects nun mitteilen, dass wir die HTML-Form, die in dieser Datei enthalten ist, mit dem Objekt „tutcust“ verknüpfen wollen:

- **Ändern Sie Zeile 6 von...**

```
<form name="" action="custform.lml" method="post">
```

... in ...

```
<form name="tutcust" action="custform.lml" method="post">
```

Dies hat zur Folge, dass alle in dieser Form definierten (HTML-) Eingabefelder mit den entsprechenden Attributen unseres Objektes „tutcust“ verbunden werden. Dies bedeutet, dass wir allen Eingabefeldern jetzt Namen von Attributen unseres Objekts „tutcust“ geben müssen.

Also zum Beispiel „num“, „name“, „address“, „country“ oder „discount“, soweit es unser „tutcust“-Objekt betrifft.

Versuchen wir, ein nicht deklariertes Attribut zu verwenden, oder wenn wir uns einfach mal verschreiben, liefert PrimeBase eine entsprechende Fehlermeldung zurück.

Jetzt haben wir im Namen der HTML-Form hinterlegt, mit welchem Objekt wir innerhalb dieser Form arbeiten wollen. Im nächsten Schritt wollen wir sämtliche Eingabefelder mit einem gültigen Namen versehen.

- **Ändern Sie Zeile 20 von ...**

```
<input name="" type="text" size="30">
```

... in ...

```
<input name="name" type="text" size="30">
```

- **Ändern Sie Zeile 25 von ...**

```
<input name="" type="text" size="30">
```

... in ...

```
<input name="address" type="text" size="30">
```

- **Ändern Sie Zeile 30 von ...**

```
<select name="" size="1">
```

... in ...

```
<select name="country" size="1">
```

- **Ändern Sie Zeile 40 von ...**

```
<input name="" type="text" size="12">
```

... in ...

```
<input name="discount" type="text" size="12">
```

Jetzt haben wir allen Eingabefeldern zu gültigen Namen verholfen.

Die Standard Buttons.

Die Buttons in „custform.lml“ möchten wir gerne auch mit gültigen Namen versehen. Aber welche gültigen Namen gibt es?

Diese Button Namen können wir ohne weiteres verwenden:

- **new** – erzeugt mit den bisher gemachten Eingaben einen neuen Datensatz.
- **delete** – löscht den dargestellten Datensatz.
- **save** – speichert den dargestellten Datensatz im Falle von gemachten Änderungen.
- **find** – sucht alle Datensätze, die mit den bisherigen Eingaben übereinstimmen.
- **first** – geht zum ersten Datensatz (eine erfolgreiche Suche vorausgesetzt).
- **last** – geht zum letzten Datensatz (eine erfolgreiche Suche vorausgesetzt).
- **next** – geht zum nächsten Datensatz (eine erfolgreiche Suche vorausgesetzt).
- **previous** – geht zum vorherigen Datensatz (eine erfolgreiche Suche vorausgesetzt).
- **clear** – geht zum „Leer-Datensatz“, effektiv werden die Inhalte aller Eingabefelder in der HTML-Seite gelöscht.

- **Ändern Sie Zeile 45 und Folgende von ...**

```
<input name="" value="Search" type="submit">
<input name="" value="|&lt;" type="submit">
<input name="" value="&lt;" type="submit">
<input name="" value="&gt;" type="submit">
<input name="" value="&gt;|" type="submit"><br>
<input name="" value="Clear" type="submit">
<input name="" value="Save" type="submit">
<input name="" value="New" type="submit">
<input name="" value="Delete" type="submit">
```

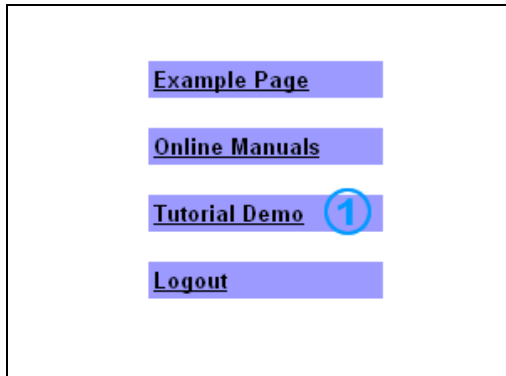
... in ...

```
<input name="find" value="Search" type="submit">
<input name="first" value="|&lt;" type="submit">
<input name="previous" value="&lt;" type="submit">
<input name="next" value="&gt;" type="submit">
<input name="last" value="&gt;|" type="submit"><br>
<input name="clear" value="Clear" type="submit">
<input name="save" value="Save" type="submit">
<input name="new" value="New" type="submit">
<input name="delete" value="Delete" type="submit">
```

Jetzt aber: Die Anwendungsdefinition neu laden und unser Modul anschauen.

Jetzt haben wir alle notwendigen Änderungen vorgenommen und können – nach dem Laden von „restart.exec“ (um dem Application Server unsere Änderungen an der Anwendungsdefinition mitzuteilen) – endlich unser Modul mit Datenbankanbindung betrachten:

- Laden Sie „http://127.0.0.1:47120/system/restart.exec“ im Browser.
- Laden Sie „http://127.0.0.1:47120/“ im Browser.
- Klicken Sie auf den Link „Tutorial Demo“ (36.1).



SCREENSHOT 36: DAS MENÜ DER INSTALLIERTEN MODULE MIT DEM EINTRAG „TUTORIAL DEMO“.

Wenn Sie bei der Eingabe sowohl der Anwendungsdefinition als auch der Namen der Buttons und Eingabefelder keine Fehler gemacht haben, dann erscheint jetzt im Browser die bereits bekannte Einstiegsseite unseres Moduls:

Customer Information: [Hier soll eine ID Nummer für jeden Kunden eingeblendet werden.]

Name

Address

Country:

Discount:

Search |< < > >|

Clear Save New Delete

SCREENSHOT 37: DIE EINSTIEGSSEITE UNSERES MODULS

Was tun, wenn ein Fehler im Browser angezeigt wird?

Haben wir aber zum Beispiel vergessen, in Zeile 20 „name“ einzugeben (`<input name="" type="text" size="30">`), erhalten wir eine Fehlermeldung:

Error in "/tutdemo/custform.lml"			
-1 "pbecontainer.pbt"@client line 166: Attribute not specified.			
No.	Function	File/String	Line
1	PBEContainer:outputValue	pbecontainer.pbt	166
2	PBEFormFieldTag:outputValue	pbetag.pbt	524
3	PBEInputTag:generateInput	pbetag.pbt	789
4	PBEInputTag:generate	pbetag.pbt	805
5	PBEApplication:liveTag	pbeapplication.pbt	296
6		/tutdemo/custform.lml	20

SCREENSHOT 38: IN ZEILE 20 WURDE DAS HTML-ATTRIBUT „NAME“ LEERGELASSEN.

Diese Fehlermeldung liefert im oberen Bereich Details zum aufgetretenen Fehler. Im unteren Teil wird ein sogenannter Stacktrace angezeigt, der Schritt für Schritt auflistet, in welcher Datei, in welcher Methode und auf welcher Zeile der Fehler aufgetreten ist.

Obwohl diese Fehlermeldung diverse Informationen enthält, die eher für Entwickler geeignet zu sein scheinen als für Einsteiger, gibt es da aber auch ein paar sehr wichtige Informationen, die auf das eigentliche Problem hindeuten und mit denen wir selbst ohne weitere Hilfe etwas anfangen können:

Während die Detailinformationen der Fehlermeldung mit ...

- "pbecontainer.pbt"@client, line 96: Attribute not specified.

... darauf hinweisen, dass wir irgendwo ein (HTML-) Attribut vergessen oder leergelassen haben, können wir anhand des Stacktraces sogar feststellen, an welcher Stelle dies aufgetreten ist – nämlich in der Datei „custform.lml“ auf Zeile 21.

Wenn wir eine Fehlermeldung bekommen ...

Dieses Vorgehen hat Allgemeingültigkeit bei der Entwicklung mit PrimeBase: Schauen Sie erst einmal, was die Detailinformationen der Fehlermeldung aussagen, und überprüfen Sie, ob eine Ihrer eigenen Dateien in dem Stacktrace aufgelistet wird.

Einfach nur verschrieben.

Schauen wir doch mal, was für eine Fehlermeldung wir bekommen, wenn wir in Zeile 20 im HTML-Attribut „name“ statt „name“ einfach nur „nam“ eingeben (<input name="nam" type="text" size="30">:

Error in "/tutdemo/custform.lml"			
-1 "pbecontainer.pbt"@client line 168: Attribute not found 'tutcust:nam'.			
No.	Function	File/String	Line
1	PBEContainer:outputValue	pbecontainer.pbt	168
2	PBEFormFieldTag:outputValue	pbetag.pbt	524
3	PBEInputTag:generateInput	pbetag.pbt	789
4	PBEInputTag:generate	pbetag.pbt	805
5	PBEApplication:liveTag	pbeapplication.pbt	296
6		/tutdemo/custform.lml	20

SCREENSHOT 39: IN ZEILE 20 WURDE EIN UNGÜLTIGER WERT EINGETRAGEN.

Auch hier bekommen wir den hilfreichen Hinweis auf den Dateinamen und die Zeile, in welcher der Fehler auftrat.

Und auch die Detailinformationen sind mit der Meldung ...

- „Attribute not found 'tutcust:nam'“

... bereits erfreulich nah dran an der Wahrheit.

HTML-Attribut vs. PrimeBase™ Enterprise Objects Attribut.

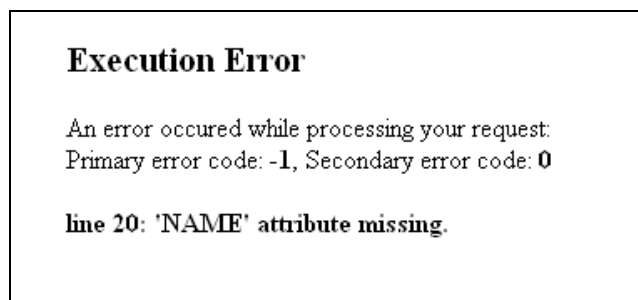
Verwirrend ist hierbei aber, dass es eben zwei Arten von Attributen gibt, von denen die Rede sein könnte:

- die Attribute eines PrimeBase Enterprise Objects, also zum Beispiel unseres eigenen Objekts „tutcust“
- die HTML-Attribute von diversen Standard HTML-Tags, also zum Beispiel „width“, „name“, „height“ usw.

Die Unterscheidung fällt bei genauerem Hinsehen aber sehr leicht: Ein HTML-Attribut hat nämlich niemals einen zweiteiligen Namen, der zudem noch einen Doppelpunkt enthält.

Das HTML-Attribut „name“ vergessen.

Was für eine Fehlermeldung erhalten wir, wenn wir das HTML-Attribut „name“ ganz vergessen (`<input type="text" size="30">`)?



SCREENSHOT 40: IN ZEILE 20 WURDE DAS
HTML-ATTRIBUT „NAME“ GANZ VERGESSEN

Auch diese Fehlermeldung hilft uns in diesem Falle weiter, da wir ja wissen, dass wir versucht haben, die Datei „custform.lml“ zu laden. Also brauchen wir bloß noch die Zeile 20 dieser Datei einer genauen Inspektion zu unterziehen, um so dem Problem auf die Schliche zu kommen.

Es klappt immer noch nicht?

Falls jetzt immer noch Probleme bestehen, schauen Sie noch einmal, ob Ihre Anwendungsdefinition exakt der Version entspricht, die wir weiter oben aufgelistet haben.

Vergessen Sie nicht, über Eingabe der URL ...

```
http://127.0.0.1:47120/system/restart.exec
```

... die Änderungen der Anwendungsdefinition beim Application Server bekannt zu machen.

Sollte dies die Probleme nicht beseitigen, wird es wahrscheinlich an der Datei „custform.lml“ liegen. Vergleichen Sie noch einmal die ursprüngliche Datei „custform.htm“ mit unserer „custform.lml“ und stellen Sie sicher, dass Sie alle oben aufgeführten Änderungen korrekt vorgenommen haben.

Findet sich das Problem trotzdem nicht, kopieren Sie einfach die Datei „custform.lml“ aus dem „originals“-Verzeichnis in unser Modulverzeichnis.

Diese Datei enthält zwar schon einige Elemente, auf die wir erst noch zu sprechen kommen, aber damit können sie wenigstens dem Tutorial weiter folgen.

Die ersten Daten eingeben.

Nach diesem Ausflug in die möglichen Fehlerquellen, die uns zu diesem Zeitpunkt das Leben schwer machen können, gehen wir jetzt davon aus, dass die Einstiegsseite unseres Moduls ohne Fehlermeldung geladen wird.

Jetzt wollen wir versuchen, ein paar Daten in unsere Datenbank einzupflegen:

- Geben Sie in die Eingabefelder die in Screenshot 41 gezeigten Daten ein.
- Drücken Sie (einmal) auf den Button „New“.

Customer Information: [Hier soll eine ID Nummer für jeden Kunden eingeblendet werden.]

Name
Name 1

Address
Adresse 1

Country:
Australia

Discount:
1

Buttons: Search, <|, <, >, >|, Clear, Save, New, Delete

SCREENSHOT 41: DIE EINSTIEGSSEITE UNSERES MODULS, DIEMAL MIT EINGABEN

Hat das Hinzufügen unseres ersten Datensatzes geklappt?

Wie Sie sehen, hat sich nicht viel getan. Um uns Klarheit zu verschaffen, ob das Einfügen des Datensatzes geklappt hat, werfen wir einen Blick auf die Console des Application Servers.

Und tatsächlich: Auf der Console des Application Servers findet sich (unter anderem) eine Zeile mit ungefähr folgendem Inhalt:

- INSERT tudemo!common.customers(address, country, discount, name) VALUES ("Adresse 1", "AUS", "1", "Name 1") [...]

INSERT ist ein Standard-SQL-Befehl, der für das Einfügen von Daten in Tabellen verwendet wird. Dass dieser Eintrag erst mit dem Drücken des Knopfes „New“ auf der Console erschienen ist und nicht von einer Fehlermeldung gefolgt wird, bedeutet, dass das Hinzufügen unseres ersten Datensatzes funktioniert hat.

Dass dieser INSERT-Eintrag auf der Console durch uns hervorgerufen wurde, lässt sich auch anhand der Werte (VALUES) in dem INSERT-Kommando nachvollziehen. Einzig „AUS“ erscheint merkwürdig, aber ein Blick in unsere HTML-Seite „custform.html“ erklärt, warum nicht der volle Name des Landes, sondern nur „AUS“ eingefügt wurde. Zeile 32:

```
<option value="AUS">Australia
```

Wollten wir stattdessen den vollen Ländernamen in der Tabelle speichern, müssten wir Zeile 33 lediglich wie folgt abändern:

```
<option value="Australia">Australia
```

Im weiteren Verlauf dieses Tutorials gehen wir aber davon aus, dass weiterhin die Kurznamen der Länder in der HTML-Datei „custform.html“ stehen.

Beachten Sie bitte unbedingt, dass bei einer solchen Änderung, die Auswirkungen auf den Inhalt der entsprechenden Tabelle bei neuen Datensätzen hat, alte Datensätze von dieser Änderung nicht betroffen sind.

Das bedeutet, dass eine Dateninkonsistenz zwischen alten und neuen Datensätzen entstehen kann und dies sollte in jedem Falle vermieden werden.

Natürlich kommt man nicht immer um solch weitgehende Änderungen herum. In solchen Fällen muss man dann aber nicht alle alten Datensätze einzeln anpassen, sondern kann dies mit ein paar gezielten SQL-Kommandos erledigen.

Das „Wie“ müssen wir Ihnen aber an dieser Stelle leider schuldig bleiben. Es sei aber auf die sehr reichhaltige Büchervielfalt zum Thema SQL verwiesen. Es ist zwar so, dass die SQL-Varianten unterschiedlicher Hersteller in vielen Details voneinander abweichen, die wesentlichen Merkmale und die grundlegenden Prinzipien unterscheiden sich aber nicht. Daher können Sie zum Lernen von für PrimeBase SQL jedes Buch über SQL verwenden.

Ein paar Standard-Datensätze hinzufügen.

Für den weiteren Verlauf des Tutorials ist es hilfreich, ein paar Standard-Datensätze in der Datenbank zu haben, anhand derer verschiedene Dinge demonstriert werden können.

- Ersetzen Sie dazu die „1“ in den entsprechenden Eingabefeldern durch eine „2“.
- Wählen Sie in „Country“ den Eintrag „Germany“ aus.
- Drücken Sie auf den Button „New“.
- Ersetzen Sie dazu die „2“ in den entsprechenden Eingabefeldern durch eine „3“.
- Wählen Sie in „Country“ den Eintrag „United States“ aus.
- Drücken Sie auf den Button „New“.

Die Suchfunktion nutzen.

Da wir jetzt drei Datensätze in unserer Datenbank (oder genauer: in unserer Tabelle „customers“ in der Datenbank „tutdemo“) haben, können wir jetzt die Suchfunktion testen.

- Drücken Sie auf den Button „Clear“.

- Geben Sie in das Feld Discount „1“ ein.
- Drücken Sie den Button „Search“.

Hat die Suche geklappt?

Was auf jeden Fall funktioniert hat, ist das Zurücksetzen aller Eingabefelder, aber das wollten wir doch gar nicht. Was ist also passiert?

Ein Blick auf die Console des Application Servers (auch während wir auf den „Search“ Button drücken) zeigt uns, dass auf unseren Knopfdruck hin tatsächlich ein Eintrag erscheint, der etwa wie folgt aussieht:

- `SELECT tutcust.address as address, tutcust.country as country, tutcust.discount as discount, tutcust.name as name, tutcust.num as num FROM tutdemo!common.customers as tutcust WHERE tutcust.discount LIKE "1%" INTO selection FOR EXTRACT;`

Was macht das SELECT-Kommando? Der Befehl SELECT ist wiederum ein Standard-SQL. Er dient zum Auswählen und Auslesen von Daten aus Tabellen.

Auffällig bei diesem Eintrag ist die sogenannte WHERE Bedingung, die in diesem Fall dafür sorgt, dass nur die Datensätze aus der Tabelle geholt werden, deren Werte in der Column „discount“ mit „1“ beginnen. Dies entspricht der Eingabe, die wir vor der Suche auf der HTML-Seite gemacht haben.

Die so genannten Jokerzeichen

Ihnen ist bei der WHERE-Bedingung auf der Console wahrscheinlich nicht entgangen, dass dort nicht nur „1“, sondern „1%“ steht. Warum ist das so?

Es gibt so genannte Jokerzeichen, die bei einer Suche typischerweise als Platzhalter für ein einzelnes Zeichen oder eine beliebige Anzahl von Zeichen dienen.

Ihnen ist dies vielleicht aus dem Umgang mit der Kommandozeile Ihres Betriebssystems geläufig. Bei DOS und Windows z. B. kann man mit folgendem Befehl in der Eingabeaufforderung alle Dateien und Verzeichnisse innerhalb des aktuellen Verzeichnisses auflisten, die mit „Test“ beginnen:

```
dir Test*
```

*Die Zeichen * und % übernehmen im Falle von SQL, bzw. des Database Servers die gleiche Funktion.*

Das „%“ in dem betrachteten SELECT-Befehl auf der Console wird von den PrimeBase Enterprise Objects dort plaziert, um bei der Suche auch automatisch alle Einträge zu berücksichtigen, die lediglich mit der eingegebenen Zeichenkette beginnen.

Warum aber werden die gefundenen Datensätze nicht angezeigt?

Wenn die Suche also durchgeführt wurde und wir davon ausgehen können, dass wirklich passende Datensätze in der Tabelle zu finden sind, warum werden diese dann nicht angezeigt?

Der „Leer-Datensatz“ (empty-record) in PrimeBase Enterprise Objects.

Unser PrimeBase Enterprise Object „tutcust“ ist intern wie eine Tabelle aufgebaut. Welche Spalten „tutcust“ besitzt, legen wir ja in der Anwendungsdefinition mit dem Aufruf der Prozedur „PBE:declareAttribute()“ fest.

Jetzt würde man erwarten, dass nach einer Suche alle gefundenen Datensätze in dieser internen Tabellenstruktur abgelegt werden – was auch der Fall ist.

Aber: Aus praktischen Gründen ist die erste Zeile der internen Tabellenstruktur unseres Objekts immer leer. Und nach einer Suche, ob erfolgreich oder nicht, wird immer diese erste, leere Zeile zur Anzeige ausgewählt.

Wenn also unsere Suche geklappt hat, würde das bedeuten, dass ein Druck auf den Knopf „>“ („next“, nächster) uns den ersten Datensatz des Suchergebnisses anzeigen müsste.

- **Drücken Sie im Browser auf den Knopf „>“.**

Customer Information: [Hier soll eine ID Nummer für jeden Kunden eingeblendet werden.]

Name
Name 1

Address
Adresse 1

Country:
Australia

Discount:
1

Search |< < > >|
Clear Save New Delete

SCREENSHOT 42: DER ERSTE „RICHTIGE“ DATENSATZ WIRD NACH EINEM DRUCK AUF DEN BUTTON „>“ ANGEZEIGT.

Das ist natürlich eine recht unintuitive Art und Weise, Suchergebnisse zu präsentieren, aber keine Sorge: Im nächsten Abschnitt („Erstellen von Events (Ereignissen)“ auf Seite 47) werden wir uns um Abhilfe für dieses ungewöhnliche Verhalten bemühen.

Die restlichen Buttons testen.

Hier zur Wiederholung noch einmal die Funktionen der einzelnen Buttons.

Diese Auflistung haben wir uns vorher, beim Anpassen der HTML-Seite „custform.html“, bereits angeschaut. Jetzt haben wir aber die internen Buttonnamen durch die Namen ersetzt, die bei der Anzeige über einen Browser angezeigt werden (beachten Sie besonders den letzten Eintrag für „Clear“):

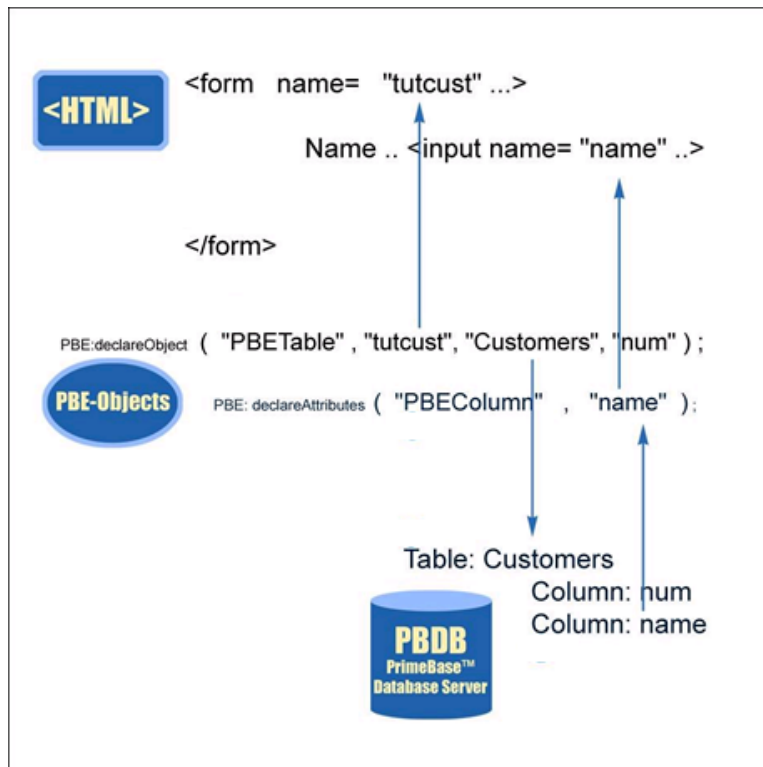
- **New** – erzeugt mit den bisher gemachten Eingaben einen neuen Datensatz.
- **Delete** – löscht den dargestellten Datensatz.
- **Save** – speichert den dargestellten Datensatz im Falle von gemachten Änderungen.
- **Search** – sucht alle Datensätze, die mit den bisherigen Eingaben übereinstimmen.
- **<** – geht zum ersten Datensatz. Dies setzt eine erfolgreiche Suche voraus.
- **>** – geht zum letzten Datensatz. Dies setzt eine erfolgreiche Suche voraus.
- **>>** – Geht zum nächsten Datensatz. Dies setzt eine erfolgreiche Suche voraus.

- < – Geht zum vorherigen Datensatz. Dies setzt eine erfolgreiche Suche voraus.
- Clear – Geht zum „Leer-Datensatz“, effektiv werden die Inhalte aller Eingabefelder in der HTML-Seite gelöscht.

Zusammenfassung: HTML, PrimeBase Enterprise Objects und Datenbanken.

Nachdem wir uns eingehend mit Detailfragen beschäftigt haben, wollen wir nun versuchen, uns wieder einen Überblick zu verschaffen.

Um ein besseres Bild über die Zusammenhänge zwischen HTML, den PrimeBase Enterprise Objects und Datenbanken zu bekommen, betrachten wir folgende Grafik:



Wie zuvor erwähnt, besteht die Anwendungsentwicklung mit PrimeBase aus drei Komponenten. Diese drei Komponenten finden sich auch in dieser Übersicht:

- Die Datenbank inklusive Tabellen und Columns.
- Die Anwendungsdefinition mit der Deklaration von Objekten und Attributen, die namentlich den Tabellen und Columns in der Datenbank entsprechen.
- Die HTML-Dateien, in denen HTML-Formulare, Input-Felder und Buttons über den Namen mit Objekten, Attributen und Datenbankfunktionen verknüpft werden.

Um die für Datenbank Anwendungen üblichen Funktionen auszulösen, verwendet man im einfachsten Fall Buttons innerhalb von HTML-Formularen. Das HTML-Attribut „name“ eines Buttons muss dabei einen der folgenden Werte enthalten:

- new, delete, save, find, first, last, next, previous, clear

Erstellen von Events (Ereignissen)

Erinnern Sie sich noch an unsere ersten Schritte mit der Suche nach Datensätzen ("Die Suchfunktion nutzen." auf Seite 43)?

Das „Leer-Datensatz“-Problem.

Wir hatten dort das Problem entdeckt, dass nach einer Suche im Browser immer erst einmal nur der so genannte „Leer-Datensatz“ angezeigt wird. Dies hatte zur Folge, dass man erst den „Richtungs“-Buttons „>“ drücken musste, um den ersten der gefundenen Datensätze anzuzeigen.

Dieser Leer-Datensatz und die Vorauswahl dieses Datensatzes nach einer Suche sind technische Notwendigkeiten, die man nicht ändern kann, ohne gravierende Nachteile an anderer Stelle hinzunehmen. Auf diesem Weg finden wir also nicht die Lösung zu diesem Problem.

Wäre es nicht praktisch, wenn man einfach folgendes sagen könnte?

Objekt „tutcust“, nachdem Du eine Suche durchgeführt hast, wähle statt des Leer-Datensatzes den ersten richtigen Datensatz zur Anzeige aus.

Und stellen Sie sich vor, fast genau so geht es mit den PrimeBase Enterprise Objects, durch die Verwendung von „Events“.

Unser erster Event.

Leider können wir den PrimeBase Enterprise Objects unser Begehren nicht ganz so umgangssprachlich vortragen wie gerade beschrieben. Aber immerhin geht es recht ähnlich:

```
PBE:objectEvent( "tutcust", "after:find:fetch( first )" );
```

„Da besteht doch gar keine Ähnlichkeit“, sagen Sie.

Schauen Sie doch vielleicht noch einmal hin, was da steht:

Objekt „tutcust“, nachdem („after“) der Button mit dem Namen „find“ („find“) gedrückt wurde (und Du die dazugehörigen Tätigkeiten durchgeführt hast), wähle („fetch“) den ersten („first“) Datensatz aus.

Wir wollen das natürlich sofort testen und bauen daher diese Event-Deklaration in unsere Anwendungsdefinition ein:

```
static
{
    PBE:declareModule( "GuestLoginModule", "Tutorial Demo", "custform.lml", "tutdemo" );

    PBE:declareObject( "PBETable", "tutcust", "tutdemo!common.customers", "num" );
    PBE:declareAttribute( "PBEColumn", "name" );
    PBE:declareAttribute( "PBEColumn", "address" );
    PBE:declareAttribute( "PBEColumn", "country" );
    PBE:declareAttribute( "PBEColumn", "discount" );

    PBE:objectEvent( "tutcust", "after:find:fetch( first )" );
}
```

UNSERE ANWENDUNGSDEFINITION MIT UNSEREM ERSTEN EVENT

- Laden Sie „<http://127.0.0.1:47120/system/restart.exec>“ im Browser.

- Laden Sie „http://127.0.0.1:47120/“ im Browser.
- Klicken Sie auf den Link „Tutorial Demo“.
- Drücken Sie den Button „Search“.

Customer Information: [Hier soll eine ID Nummer für jeden Kunden eingeblendet werden.]

Name
Name 1

Address
Adresse 1

Country:
Australia

Discount:
1

Search |< < > >|

Clear Save New Delete

SCREENSHOT 43: DIE SUCHE HAT GEKLAPPT UND DER ERSTE „RICHTIGE“ DATENSATZ WIRD ANGEZEIGT.

Der erste gefundene Datensatz wird automatisch angezeigt – also hat unser Event allem Anschein nach funktioniert.

Genau diesen Objekt-Event (das Auswählen des ersten Datensatzes nach einer Suche) wird man in der Praxis übrigens recht häufig verwenden.

Noch ein Event: Ein Eingabefeld als Muss-Feld deklarieren.

Was ist, wenn wir die Eingabe von Daten in eine unserer Columns unbedingt benötigen, also ein sogenanntes „Muss-Feld“ haben?

Gewieft HTML-Entwickler werden nun „JavaScript“ benutzen. Mit JavaScript lassen sich die HTML-Eingabefelder vor dem Abschicken des HTML-Formulares auf Inhalt und Korrektheit prüfen.

Das funktioniert aber nicht mit Browsern, bei denen die Ausführung von JavaScript aus Sicherheitsgründen deaktiviert wurde.

Wünschenswert ist in jedem Fall, dass auch die PrimeBase Enterprise Objects eine Prüfung der Eingabewerte vornehmen (einer zusätzlichen Überprüfung durch JavaScript steht natürlich nichts im Wege). Auch hierfür kann ein Event verwendet werden.

Zunächst müssen wir uns natürlich erst einmal entscheiden, bei welcher Column wir auf Eingabe von Daten bestehen wollen. Wählen wir doch einfach die Column „name“.

Ein entsprechender Event würde folgendermaßen aussehen:

```
PBE:attributeEvent( "tutcust", "name", "before:new,save:required" );
```

Auch hier gibt der erste Parameter wieder an, für welches Objekt dieser Event bestimmt ist. Da es sich aber um ein Attribut Event handelt, müssen wir zusätzlich

noch angeben, für welches Attribut unseres Objekts dieser Event gedacht ist, was mit dem zweiten Parameter geschieht.

In diesem Falle soll der Event nicht erst aufgerufen werden, nachdem etwas passiert ist, sondern davor. Schließlich wollen wir ja das Hinzufügen und das Speichern eines Datensatzes nur zulassen, wenn für den Namen eine Eingabe erfolgt. Daher verwenden wir in diesem Event nicht „after“ (wie bei unserem ersten Event), sondern „before“. Durch „new“ und „save“ wird dann noch angegeben, dass dieser Event beim Auslösen dieser beiden Buttons ausgeführt werden soll.

Hier unsere um den zweiten Event erweiterte Anwendungsdefinition:

```
static
{
    PBE:declareModule( "GuestLoginModule", "Tutorial Demo", "custform.lml", "tutdemo" );

    PBE:declareObject( "PBETable", "tutcust", "tutdemo!common.customers", "num" );
    PBE:declareAttribute( "PBEColumn", "name" );
    PBE:declareAttribute( "PBEColumn", "address" );
    PBE:declareAttribute( "PBEColumn", "country" );
    PBE:declareAttribute( "PBEColumn", "discount" );

    PBE:objectEvent( "tutcust", "after:find:fetch( first )" );
    PBE:attributeEvent( "tutcust", "name", "before:new,save:required" );
}
```

UNSERE ANWENDUNGSDEFINITION MIT EINEM WEITEREN EVENT

Anzeigen von Muss-Feldern in der HTML-Seite.

Überlegen wir einmal, was passieren würde, wenn wir jetzt auf „New“ drücken, während das Eingabefeld „Name“ leer ist?

Es würde kein neuer Datensatz erstellt werden, da wir dies mit dem Attribut Event so festgelegt haben.

Was würde sonst noch passieren? Nichts, leider – dem Anwender des Moduls würde im Browser kein Hinweis auf Probleme angezeigt werden.

Das wollen wir ändern und zwar indem wir in diesem Fall (dem vergeblichen Versuch, einen Datensatz hinzuzufügen oder zu speichern) vor dem Eingabefeld „Name“ drei rote Sternchen erscheinen lassen.

- Öffnen Sie die Datei „custform.lml“ in einem beliebigen Texteditor.
- Ändern Sie Zeile 20 von ...

```
<input name="name" type="text" size="30">
```

... in ...

```
<container type="error" name="name"><font
color="red">***</font></container><input name="name" type="text" size="30">
```

Das <container>-LiveTag

Das <container>-LiveTag beherrscht verschiedene Funktionen.

Für dieses Beispiel eines Muss-Feldes wird nur die „error“-Funktion dieses Tags verwendet.

Umgangssprachlich würde man diese Funktion etwa so formulieren:

Wenn ein Fehler (error) bei dem Feld „name“ auftritt, füge den folgenden HTML-Code in die Seite ein.

So, HTML-Datei und Anwendungsdefinition haben wir angepasst, also wollen wir unsere Änderungen auch gleich testen:

- Laden Sie „http://127.0.0.1:47120/system/restart.exec“ im Browser.
- Laden Sie „http://127.0.0.1:47120/“ im Browser.
- Klicken Sie auf den Link „Tutorial Demo“.
- Drücken Sie den Button „New“.

Customer Information: [Hier soll eine ID Nummer für jeden Kunden eingeblendet werden.]

Name
*** ①

Address

Country:

Discount:

Search |< < > >|

Clear Save New Delete

SCREENSHOT 44: FEHLENDE EINGABEWERTE IN „NAME“ WERDEN MARKIERT.

Auch hier wieder ein voller Erfolg: Das Muss-Feld „Name“ wird, wenn es leer ist, nach Drücken des „New“-Buttons markiert dargestellt (44.1).

Das Gleiche funktioniert übrigens auch, wenn Sie einen vorhandenen Datensatz suchen und anzeigen lassen, dann den Inhalt des Feldes „Name“ löschen und anschließend den Button „Save“ drücken.

Der letzte Event: Auf eine andere HTML-Seite wechseln.

Stellen Sie sich vor, wir hätten tausende von Kunden in unsere Tabelle eingetragen.

Mit der Suchfunktion kann man sich problemlos einzelne oder sogar mehrere Kunden gleichzeitig herausuchen. Das Problem ist aber, dass man nie auf einen Blick sehen kann, welche Datensätze gefunden wurden.

Wäre es nicht schön, zu diesem Zweck eine tabellarische Übersicht zu haben?

Genau zu diesem Zweck wurde von unserem Web-Designer die HTML-Seite „custfound.htm“ erstellt. Wir haben diese Datei mittlerweile schon unter dem Namen „custfound.lml“ in unserem Modulverzeichnis abgelegt.

Was möchten wir erreichen? Wir möchten, dass der Browser nach einer Suche automatisch auf die HTML-Seite „custfound.lml“ geht und dort die gefundenen Datensätze tabellarisch anzeigt.

Wie sähe der Event für so etwas aus? Und würde ein Objekt Event oder doch ein Attribut Event Verwendung finden?

Dies müsste als Objekt Event formuliert werden, da das Ereignis (Druck auf den Knopf „Search“) direkt nichts mit einer unserer Spalten zu tun hat. Probieren wir es also mal so:

```
PBE:objectEvent( "tutcust", "after:find:goto('custfound.lml' )" );
```

Schaut doch gut aus, oder? Das einzige neue ist jetzt der Event-Befehl „goto“.

Beachten Sie unbedingt, dass der Parameter für den „goto“-Event-Befehl in einfachen Anführungszeichen stehen muss.

Schauen wir doch mal, wie unsere Anwendungsdefinition mit diesem Event aussieht:

```
static
{
    PBE:declareModule( "GuestLoginModule", "Tutorial Demo", "custform.lml", "tutdemo" );

    PBE:declareObject( "PBETable", "tutcust", "tutdemo!common.customers", "num" );
    PBE:declareAttribute( "PBEColumn", "name" );
    PBE:declareAttribute( "PBEColumn", "address" );
    PBE:declareAttribute( "PBEColumn", "country" );
    PBE:declareAttribute( "PBEColumn", "discount" );

    PBE:objectEvent( "tutcust", "after:find:fetch( first )" );
    PBE:objectEvent( "tutcust", "after:find:goto('custfound.lml' )" );
    PBE:attributeEvent( "tutcust", "name", "before:new,save:required" );
}
```

UNSERE ANWENDUNGSDEFINITION MIT EINEM WEITEREN EVENT

Die Reihenfolge von Events

Es ist sehr wichtig, auf eine sinnvolle Reihenfolge von Events zu achten. Events eines einzelnen Objekts oder Attributs werden in der Reihenfolge ihrer Deklaration ausgeführt.

- Laden Sie „<http://127.0.0.1:47120/system/restart.exec>“ im Browser.
- Laden Sie „<http://127.0.0.1:47120/>“ im Browser.
- Klicken Sie auf den Link „Tutorial Demo“.
- Drücken Sie den Button „Search“.

Hoppla, eine Fehlermeldung:

```

Error in "/tutdemo/custfound.lml"
-1 "pbttag.pbt"@client line 128: Form name not specified.


| No. | Function               | File/String            | Line |
|-----|------------------------|------------------------|------|
| 1   | PBEFormTag:generate    | pbttag.pbt             | 128  |
| 2   | PBEApplication:liveTag | pbeapplication.pbt     | 296  |
| 3   |                        | /tutdemo/custfound.lml | 9    |


```

SCREENSHOT 45: DER GOTO-EVENT HAT FUNKTIONIERT, ABER „CUSTFOUND.LML“ IST FEHLERHAFT.

Mal sehen, der Fehler soll in Zeile 9 der Datei „custfound.lml“ auftreten:

```
<form name="" action="custfound.lml" method="post">
```

Und tatsächlich, da fehlt noch der Inhalt des HTML-Attributs „name“, was wir natürlich sofort korrigieren können:

```
<form name="tutcust" action="custfound.lml" method="post">
```

Das Problem sollte damit beseitigt sein, also probieren wir es noch einmal mit dem kompletten Testvorgang:

- Laden Sie „http://127.0.0.1:47120/system/restart.exec“ im Browser.
- Laden Sie „http://127.0.0.1:47120/“ im Browser.
- Klicken Sie auf den Link „Tutorial Demo“.
- Drücken Sie den Button „Search“.

Number	Name	Address	Country	Discount
[Kunden ID hier.]	[Name des Kunden hier.]	[Adresse des Kunden hier.]	[Land des Kunden hier.]	[Eingeräumter Discount hier.]
[Kunden ID hier.]	[Name des Kunden hier.]	[Adresse des Kunden hier.]	[Land des Kunden hier.]	[Eingeräumter Discount hier.]

SCREENSHOT 46: DER GOTO-EVENT AUF DIE DATEI „CUSTFOUND.LML“ HAT FUNKTIONIERT.

Gut, der Goto-Event, der uns nach einem Druck auf den Knopf „Search“ auf die Seite „custfound.lml“ bringen soll, funktioniert nun. Dass die Auflistung der gefundenen Datensätze in „custfound.lml“ noch nicht funktioniert, überrascht nicht, schließlich haben wir diesbezüglich noch keinerlei Anpassungen an der Datei vorgenommen.

Das <output>-LiveTag

Betrachten Sie doch mal folgenden Screenshot:

Customer Information: [Hier soll eine ID Nummer für jeden Kunden eingeblendet werden.] ①

Name
Name 1

Address
Adresse 1

Country:
Australia

Discount:
1

Search < < > >|

Clear Save New Delete

SCREENSHOT 47: DIE KOMMENTARTE DES WEB-DESIGNERS SOLLTEN ERSETZT WERDEN.

Den Kommentar „[Hier soll eine ID Nummer für jeden Kunden eingeblendet werden.]“ (47.1) sollten wir durch etwas sinngemäßes ersetzen, aber woher nehmen wir eine ID-Nummer für jeden Kunden? Und wie geben wir selbige dann anstelle des Kommentars aus?

Die Primary Key Column als Kunden ID-Nummer verwenden.

Zuerst mal zu der ID für jeden Kunden: Jeder Kunde bekommt einen Datensatz in unserer Tabelle. Jeder Datensatz muss – wie wir in den First Steps gelernt haben – mit einer Column versehen sein, die einen eindeutigen Wert aufweist, dem so genannten Primary Key.

Diese Column trägt in unserer Tabelle „customers“ den Namen „num“ und ist hervorragend als Kunden-ID zu verwenden, oder etwa nicht?

Ausgeben von Columns in der HTML-Seite.

Nun zum zweiten Problem: Wie geben wir die Column „num“, die wir als Kunden-Id verwenden wollen, in der HTML-Seite aus? Das Hilfsmittel hierfür ist das LiveTag <output>: <output> kennt zwei HTML-Attribute, „type“ und „name“. Beide HTML-Attribute müssen angegeben werden, damit <output> korrekt funktioniert und keine Fehlermeldung erscheint.

Was geben wir bei „name“ an? Ganz einfach: Den Namen der Column, die wir ausgeben möchten, also „num“.

Und bei „type“? Bei „type“ geben wir „field“ an, um klarzustellen, dass wir den Inhalt einer Column ausgeben möchte. Schauen wir doch einfach mal, wie es in der Praxis aussieht:

- Öffnen Sie die Datei „custform.lml“ in einem beliebigen Texteditor.
- Ändern Sie Zeile 12 von ...

```
[Hier soll<br> eine ID Nummer<br> f&uuml;r jeden Kunden<br> eingeblendet werden.]<br>
```

- ... in ...

```
<output type="field" name="num"><br>
```

- Speichern Sie die Änderungen.
- Klicken Sie auf den Link „Tutorial Demo“.
- Drücken Sie den Button „Clear“.
- Drücken Sie den Button „Search“.

Jetzt erscheint die Seite „custfound.lml“:

Number	Name	Address	Country	Discount
<u>[Kunden ID hier.]</u>	[Name des Kunden hier.]	[Adresse des Kunden hier.]	[Land des Kunden hier.]	[Eingeräumter Discount hier.]
<u>[Kunden ID hier.]</u>	[Name des Kunden hier.]	[Adresse des Kunden hier.]	[Land des Kunden hier.]	[Eingeräumter Discount hier.]

SCREENSHOT 48: WIR VERWEILEN HIER DIESMAL NUR KURZ.

- **Klicken Sie auf einen der Links mit „[Kunden ID hier.]“ als Text (48.1)**

SCREENSHOT 49: STATT DES KOMMENTARS WIRD EINE KUNDEN-ID ANGEZEIGT.

Geschafft – der Kommentar des Web-Designers ist weg und wurde durch eine Nummer ersetzt. Betätigen Sie ruhig die „Navigations“-Buttons auf unserer Seite, um sicherzustellen, ob auch bei den anderen Datensätzen eine ID angezeigt wird.

Endspurt: Eine Liste der gefundenen Datensätze in „custfound.lml“

Betrachten wir doch noch einmal den derzeitigen Zustand von „custfound.lml“:

Number	Name	Address	Country	Discount
[Kunden ID hier.]	[Name des Kunden hier.]	[Adresse des Kunden hier.]	[Land des Kunden hier.]	[Eingeräumter Discount hier.]
[Kunden ID hier.]	[Name des Kunden hier.]	[Adresse des Kunden hier.]	[Land des Kunden hier.]	[Eingeräumter Discount hier.]

SCREENSHOT 50: AUF DIESER SEITE SOLLEN DIE ERGEBNISSE DER SUCHE AUFGELISTET WERDEN.

Die letzten Kommentare ersetzen.

Wir wollen jetzt statt der Kommentare die Inhalte der entsprechenden Columns anzeigen lassen.

Wie das geht? Ganz genauso, wie wir das zuvor im Abschnitt 'Ausgeben von Columns in der HTML-Seite.' auf Seite 53 gemacht haben, mit Hilfe des LiveTags <output>.

Immer noch ratlos? Kein Problem, so geht es:

- Öffnen Sie die Datei „custfound.lml“ in einem beliebigen Texteditor.
- Ändern Sie die Zeilen 22 und 31 von ...

```
<td><a href="custform.lml"><b>[Kunden ID hier.]</b></a></td>
```

... in ...

```
<td><a href="custform.lml"><b><output type="field" name="num"></b></a></td>
```

- Ändern Sie die Zeilen 23 und 32 von ...

```
<td width="85" nowrap>[Name des Kunden hier.]</td>
```

... in ...

```
<td width="85" nowrap><output type="field" name="name"></td>
```

- Ändern Sie die Zeilen 24 und 33 von...

```
<td width="114">[Adresse des Kunden hier.]</td>
```

...in...

```
<td width="114"><output type="field" name="address"></td>
```

- Ändern Sie die Zeilen 25 und 34 von...

```
<td width="52">[Land des Kunden hier.]</td>
```

...in...

```
<td width="52"><output type="field" name="country"></td>
```

- Ändern Sie die Zeilen 26 und 35 von...

```
<td>[Einger&auml;umter Discount hier.]</td>
```

...in...

```
<td><output type="field" name="discount"></td>
```

Jetzt wollen wir mal schauen, ob sich die ganze Mühe gelohnt hat:

- Klicken Sie auf den Link „Tutorial Demo“.
- Drücken Sie den Button „Clear“.
- Drücken Sie den Button „Search“.

Number	Name	Address	Country	Discount
<u>1</u>	Name 1	Adresse 1	AUS	0
<u>1</u>	Name 1	Adresse 1	AUS	0

SCREENSHOT 51: SIEHT SCHON BESSER AUS!

Wir sehen nur zweimal die Daten des ersten Datensatzes – was ist schiefgelaufen?

Eigentlich gar nichts: Wir haben die Kommentare des Web-Designers ersetzt und lassen stattdessen den Inhalt der entsprechenden Columns ausgeben. Und genau das Ergebnis sehen wir jetzt in dem obigen Screenshot.

Also ist unser Vorgehen bis hierher scheinbar noch nicht ganz richtig gewesen. Wir müssen einen Weg finden, nicht nur die Daten des ersten Datensatzes anzeigen zu lassen, sondern auch die der anderen gefundenen Datensätze.

Zu diesem Zweck ist eine Art Schleife hilfreich, die anfangs den ersten Datensatz auswählt und dann bei jedem Schleifendurchlauf den jeweils nächsten Datensatz.

Das <container>-LiveTag als Schleife verwenden.

Erinnern Sie sich noch an das <container>-LiveTag aus dem Abschnitt 'Anzeigen von Muss-Feldern in der HTML-Seite.' auf Seite 49? Dort hatten wir es zum Anzeigen eines Hinweises auf eine fehlende Eingabe in ein Muss-Feld verwendet.

Genau dieses LiveTag können wir jetzt verwenden, um unsere Auflistung der Suchergebnisse auf solide Füße zu stellen.

Das LiveTag <container> kennt das HTML-Attribut „type“. Wenn wir in „type“ „loopall“ eingeben, erzeugen wir ein Schleifenkonstrukt, welches uns erlaubt, die Inhalte von allen gefundenen Datensätzen auszugeben.

Schauen wir doch mal, wie wir unsere HTML-Seite modifizieren müssen, damit dieses Schleifenkonstrukt funktioniert. Bisher sieht unsere Seite „custfound.lml“ so aus:

[Zeile 1 bis 18]

```
<tr bgcolor="#eeeeee">
<td><a href="custform.lml"><b><output type="field" name="num"></b></a></td>
<td width="85" nowrap><output type="field" name="name"></td>
<td width="114"><output type="field" name="address"></td>
<td width="52"><output type="field" name="country"></td>
<td><output type="field" name="discount"></td>
</tr>

<tr bgcolor="#bbbbbb">
<td><a href="custform.lml"><b><output type="field" name="num"></b></a></td>
<td width="85" nowrap><output type="field" name="name"></td>
<td width="114"><output type="field" name="address"></td>
<td width="52"><output type="field" name="country"></td>
<td><output type="field" name="discount"></td>
</tr>
```

[Zeile 38 und folgende]

Das ändern wir jetzt wie folgt:

[Zeile 1 bis 18]

```
<container type="loopall">

<container type="nextrow">
<tr bgcolor="#eeeeee">
<td><a href="custform.lml"><b><output type="field" name="num"></b></a></td>
<td width="85" nowrap><output type="field" name="name"></td>
<td width="114"><output type="field" name="address"></td>
<td width="52"><output type="field" name="country"></td>
<td><output type="field" name="discount"></td>
</tr>
</container>
...
```

```

...
<container type="nextrow">
<tr bgcolor="#bbbbbb">
<td><a href="custform.lml"><b><output type="field" name="num"></b></a></td>
<td width="85" nowrap><output type="field" name="name"></td>
<td width="114"><output type="field" name="address"></td>
<td width="52"><output type="field" name="country"></td>
<td><output type="field" name="discount"></td>
</tr>
</container>

</container>

[Zeile 45 und folgende]

```

Container-LiveTags vom Typ „nextrow“ müssen immer innerhalb eines anderen Container-LiveTags stehen, welches eine Schleifenfunktion erfüllt. In diesem Fall werden die Container-LiveTags vom Typ „nextrow“ von dem Container-LiveTag des Typs „loopall“ umschlossen.

Die mit „loopall“ Containern begonnene Schleife mit „nextrow“ Containern durchlaufen.

Warum überhaupt die „nextrow“-Container-Tags?

Irgendwie muss man, nachdem die Schleife begonnen wurde und die Columns des ersten Datensatzes ausgegeben wurden, den zweiten Datensatz auswählen, um dann dessen Columns auszugeben.

Und nachdem das gemacht wurde, muss man wiederum den nächsten Datensatz auswählen und dessen Columns ausgeben usw.

Das Auswählen des nächsten Datensatzes geschieht durch Container-LiveTags des Typs „nextrow“.

Endlosschleife ohne <container type="nextrow">

Im Umkehrschluss bedeutet dies übrigens, dass eine Loopall-Schleife ohne Nextrow-Containertags eine Endlosschleife ist. Sollten Sie mal in die Verlegenheit kommen, so etwas produziert zu haben, geben Sie einfach auf der Application Server Console „halt“ ein, gefolgt von RETURN oder ENTER. Anschliessend sollten Sie natürlich das Problem beseitigen.

Nun denn, die Datei „custfound.lml“ haben wir angepasst und nun wollen wir doch mal nachsehen, wie es jetzt aussieht:

- **Klicken Sie auf den Link „Tutorial Demo“.**
- **Drücken Sie den Button „Clear“.**
- **Drücken Sie den Button „Search“.**

Number	Name	Address	Country	Discount
1	Name 1	Adresse 1	AUS	1
1	Name 1	Adresse 1	AUS	1
1	Name 1	Adresse 1	AUS	1

SCREENSHOT 52: ES WERDEN DREI ZEILEN ANGEZEIGT, ABER IMMER DER GLEICHE INHALT

Nun ja, da stimmt wohl immer noch irgendetwas nicht. Zwar bekommen wir jetzt drei Einträge angezeigt, was der Anzahl unserer Standard-Datensätze entspricht, warum aber werden nicht die unterschiedlichen Datensätze angezeigt?

Ausgewählter Datensatz vs. aktueller Datensatz.

Das Problem ist das <output>-LiveTag – oder besser gesagt: wie wir damit umgehen.

Wollen wir Daten des innerhalb einer Schleife jeweils ausgewählten Datensatzes zur Anzeige verwenden, müssen wir innerhalb von HTML-Seiten an den Namen des Attributes „name“ einen Stern anfügen (z. B. <output type=“field“ name=“**name***“>).

Das veranlasst die PrimeBase Enterprise Objects dazu, nicht den normal ausgewählten Datensatz zu verwenden, sondern den jeweils „aktuellen“ Datensatz, also den Datensatz, der durch die Schleife momentan ausgewählt ist.

Wir wollen das gleich ausprobieren und fügen hinter allen HTML-Attributen mit dem Namen „name“ innerhalb des <output>-LiveTags einen Stern ein:

[Zeile 1 bis 16]

```
<container type="loopall">

<container type="nextrow">
<tr bgcolor="#eeeeee">
<td><a href="custform.lml"><b><output type="field" name="num*"></b></a></td>
<td width="85" nowrap><output type="field" name="name*"></td>
<td width="114"><output type="field" name="address*"></td>
<td width="52"><output type="field" name="country*"></td>
<td><output type="field" name="discount*"></td>
</tr>
</container>

<container type="nextrow">
<tr bgcolor="#bbbbbb">
<td><a href="custform.lml"><b><output type="field" name="num*"></b></a></td>
<td width="85" nowrap><output type="field" name="name*"></td>
<td width="114"><output type="field" name="address*"></td>
<td width="52"><output type="field" name="country*"></td>
<td><output type="field" name="discount*"></td>
</tr>
</container>

</container>
```

[Zeile 45 und folgende]

- **Klicken Sie auf den Link „Tutorial Demo“.**
- **Drücken Sie den Button „Clear“.**
- **Drücken Sie den Button „Search“.**

Number	Name	Address	Country	Discount
<u>1</u>	Name 1	Adresse 1	AUS	1
<u>2</u>	Name 2	Adresse 2	GER	2
<u>3</u>	Name 3	Adresse 3	US	3

SCREENSHOT 53: NUN STIMMT DIE ANZEIGE DER INFORMATIONEN.

Und tatsächlich, jetzt werden die Inhalte der einzelnen Datensätze für die Ausgabe verwendet.

Nun wartet nur noch ein kleines Problem auf uns: Der Link unter „Number“ stimmt leider noch nicht. Dieser Link sollte so funktionieren, dass man beim Anklicken immer auf die Seite „custform.lml“ kommt und dort der entsprechende Datensatz automatisch angezeigt wird. Ersteres klappt, letzteres leider noch nicht.

Die letzte Änderung: Die HTML-Links anpassen.

Schauen wir uns doch mal die beiden Links in „custfound.lml“ in den Zeilen 25 und 35 an:

```
<td><a href="custform.lml"><b><output type="field" name="num*"></b></a></td>
```

Diese beiden Zeilen, also 25 und 35, ersetzen wir jeweils durch die folgende Zeile:

```
<td><a href="custform.lml" value="^pbe_current_row**  
name="pbe_selected_row"><b><output type="field" name="num*"></b></a></td>
```

Auf diese Weise wird in einen normalen HTML-Link etwas „eingebaut“, was die PrimeBase Enterprise Objects dazu veranlasst, vor dem Anzeigen der entsprechend referenzierten Seite („custform.lml“) den gewünschten Datensatz auszuwählen.

Und tatsächlich, wenn wir den angezeigten Link für die erste Tabellenzeile vor dieser Änderung im Browser betrachten (in der Statusleiste des Browsers, während der Mauszeiger über dem Link steht) ...

```
http://127.0.0.1:47120/tutdemo/custform.lml?4dwd-id=wd9f53c2e57e786bbe
```

... und dann mit dem Link vergleichen, der nach dieser Änderung im Browser angezeigt wird ...

```
http://127.0.0.1:47120/tutdemo/custform.lml?a-tutcust-pbe_selected_row=1&dewd-  
id=wd9f53c2e57e786bbe
```

... fällt der Zusatz „a-tutcust-pbe_selected_row=1“ ins Auge.

Der Application Server ist in der Lage, diese zusätzlichen Informationen einer Adresse (URL) auszuwerten.

In der Standardkonfiguration des Application Servers reicht dieser die Informationen an die PrimeBase Enterprise Objects weiter, die für die weitere Auswertung sorgen.

In diesem Fall z. B. wird der Zeiger auf den ausgewählten Datensatz des Objektes „tutcust“ auf „1“ gesetzt, also der erste (richtige) Datensatz ausgewählt – und zwar bevor auf die in dem Link referenzierte Seite „custform.lml“ gewechselt wird.

Damit haben wir dann auch unser letztes Problem gelöst, unser Ziel erreicht und unsere kleine Web-Anwendung fertiggestellt.

Fühlen Sie sich herzlich eingeladen, weiter mit diesem Modul zu experimentieren!

Das Hinzufügen weiterer Columns

Beachten Sie bitte, dass der Application Server Änderungen an einem Datenbank Schema nicht automatisch mitgeteilt bekommt. Sollten Sie also zum Beispiel eine Column zu unserer Tabelle „customers“ hinzufügen, müssen Sie den Application Server stoppen und wieder starten, damit die Änderungen für diesen sichtbar werden.

**Der große Bruder von „Tutorial Demo“:
Das Modul „PBE Demo“.**

Der große Bruder unseres Moduls „Tutorial Demo“ heißt „PBE Demo“ und wird nur nach einer gültigen Anmeldung im Modulmenü des Application Servers angezeigt.

Das Modulverzeichnis von „PBE Demo“ heißt „demo“ und auch dort finden Sie ein „exec“-Verzeichnis mit einer Anwendungsdefinition, die im wesentlichen dem entspricht, was wir bisher in diesem Tutorial kennengelernt haben.